Formal Methods in System Design, 18, 141–163, 2001

Efficient Detection of Vacuity in Temporal Model Checking

ILAN BEER
SHOHAM BEN-DAVID shoham@il.ibm.com
CINDY EISNER
IBM Research Laboratory in Haifa, Israel

YOAV RODEH
IBM Research Laboratory in Haifa, Israel; Weizmann Institute of Science, Rehovot, Israel

Received May 26, 1998; Revised January 21, 2000

Abstract. The ability to generate a counter-example is an important feature of model checking tools, because a counter-example provides information to the user in the case that the formula being checked is found to be non-valid. In this paper, we turn our attention to providing similar feedback to the user in the case that the formula is found to be valid, because valid formulas can hide real problems in the model. For instance, propositional logic formulas containing implications can suffer from antecedent failure, in which the formula is trivially valid because the pre-condition of the implication is not satisfiable. We call this vacuity, and extend the definition to cover other kinds of trivial validity. For non-vacuously valid formulas, we define an interesting witness as a non-trivial example of the validity of the formula. We formalize the notions of vacuity and interesting witness, and show how to detect vacuity and generate interesting witnesses in temporal model checking. Finally, we provide a practical solution for a useful subset of ACTL formulas.

Keywords: model checking, temporal logic, vacuity, formal verification, interesting witness

1. Introduction

The ability to generate a counter-example is an important feature of model checking tools, because a counter-example provides information to the user in the case that the formula being checked is found to be non-valid. In this paper, we turn our attention to providing similar feedback to the user in the case that the formula is found to be valid. At first glance, such a goal may seem strange, because proving formulas valid is the supposed goal of model checking. However, additional information regarding valid formulas is indeed important, because a valid formula may hide real problems in the model.

Several years of experience in practical formal verification of hardware at IBM [3] have shown us that during the first formal verification runs of a new hardware design, typically 20% of formulas are found to be trivially valid, and that trivial validity always points to a real problem in either the design or its specification or environment. Of the formulas which are found to be non-trivially valid, examination of a non-trivial example trace discovers a problem for approximately 10% of the formulas.

The problem of a trivially valid formula was first noted by Beatty and Bryant [2], who termed it *antecedent failure*. Antecedent failure means that a formula is trivially valid because the pre-condition (antecedent) of the formula is not satisfiable in the model. If the validity of a formula is trivial, this must be indicated to the user. If it is not, the usefulness of formal verification is compromised, since a trivially valid formula is not intentionally part of a specification (and therefore indicates a problem in the design or an error in the specification). For instance, consider the following formula:

$$AG(\mathit{request} \rightarrow AX\mathit{ack}) \tag{1}$$

In a model M in which a request is never made, i.e., $M \models AG\neg\mathit{request}$, Formula 1 is trivially valid.

Antecedent failure is an intuitively easy concept to grasp. However, the fact that it depends on the use of a particular operator is disturbing. We would like to capture the same problem in the equivalent formula:

$$AG(\neg\mathit{request} \vee AX\mathit{ack}) \tag{2}$$

Because we are concerned with temporal logic, we would also like the notion of a trivially valid formula to include a temporal aspect. For instance, consider the following formula:

$$AG(p \rightarrow AX(q \rightarrow AXr)) \tag{3}$$

If p never occurs, and thus $M \models AG\neg p$, then Formula 3 is trivial by antecedent failure. However, we would also like the notion of a trivially valid formula to cover the case that, while q may occur, and thus $M \models EFq$, q never occurs at a next state of p, and thus $M \models AG(p \rightarrow AX\neg q)$.

In addition, the notion of a trivially valid formula should capture other potential problems, such as that illustrated by the following formula:

$$AG(\mathit{request} \rightarrow A[\neg\mathit{data\_valid}\, U \mathit{write\_enable}]) \tag{4}$$

As with the previous examples, Formula 4 is trivially valid in a model in which a request is never made. However, even in a model M in which $M \models EF\,\mathit{request}$, it is possible that the validity of Formula 4 is trivial. If $M \models AG(\mathit{request} \rightarrow \mathit{write\_enable})$, then there are no states in which it is required that the sub-formula $\neg\mathit{data\_valid}$ hold; in other words, there is “nothing left for the model checker to check”. In such a model, the validity of Formula 4 is trivial. In this paper, we extend and formalize the notion of trivial validity to these and other cases. We use the term *vacuity* for the extended definition, and call a formula which suffers from vacuity a *vacuously valid* formula.

Trivial validity is usually an indication of a problem in the model (rather than the specification). A related problem is a formula which is non-vacuously valid, but which does not express the property that was intended by the user. In other words, we would like to provide a way to discover errors in the formula, even when the formula is non-vacuously valid. We confront this problem by formalizing the notion of an *interesting witness*: a trace which shows a non-trivial example of the validity of a formula. Examining a positive example provides some confidence that the formal specification accurately reflects the intent of the user, one of the weak links in the practical application of formal verification to hardware design.

As an example, consider Formula 3. An interesting witness to Formula 3 is a path on which p occurs at some state s_i, q occurs at state s_{i+1}, and r occurs at state s_{i+2}.

Note that simply negating the original formula will not provide a non-trivial example. If we negate Formula 3, we get:

$$EF(p \wedge EX(q \wedge EX\neg r)) \tag{5}$$

Obviously, since Formula 5 is the negation of Formula 3, Formula 5 is false if Formula 3 is true. However, because Formula 5 is an existential formula, there is no trace which can show it is false, and the counter-example mechanisms of [13] and of SMV [18, 9] will not generate a trace.

Negating the single operand of the AG operator in Formula 3 as follows:

$$AG\neg(p \rightarrow AX(q \rightarrow AXr)) \tag{6}$$

will also not guarantee an interesting witness. For instance, a valid counter-example to Formula 6 is a path to a state in which p does not occur. Once again, this is a trivial positive example of the truth of the original Formula 3.

Our motivation is temporal model checking. However, the notions of vacuity and interesting witness are not limited to temporal logics. Therefore, we will first define our terms in general, and only then discuss vacuity detection and generation of interesting witnesses in temporal model checking. Finally, we will show a practical solution for a useful subset of ACTL formulas under temporal model checking.

The remainder of this paper is organized as follows. In Section 2 we define some important temporal logics. In Section 3 we formalize the notion of vacuity, and show how to efficiently detect vacuity using a model checker. In Section 4 we formalize the notion of interesting witness and show how to generate interesting witnesses using a model checker. In Section 5 we provide a practical solution for a useful subset of ACTL. In Section 6 we compare our work with a previous version of our theory, and with related work. In Section 7 we conclude.

2. Preliminaries

CTL* [11] is a logic with the following syntax:

1. Every atomic proposition is a formula.
2. If f and g are formulas, then so are $\neg f$ and $f \wedge g$.
3. If f is a formula, then Ef is also a formula.
4. If f and g are formulas, then fUg and Xf are also formulas.

Additional operators can be viewed as abbreviations of the above, as follows:

- $f \vee g = \neg(\neg f \wedge \neg g)$
- $Fg = \mathit{true}\ U\ g$
- $Gf = \neg(\mathit{true}\ U \neg f)$
- $fVg = \neg(\neg f U \neg g)$
- $Af = \neg E \neg f$

The semantics of a CTL* formula is defined with respect to a Kripke structure K. A Kripke structure is a quadruple (S, S_0, R, L), where S is a finite set of states, $S_0 \subseteq S$ is a set of initial states, $R \subseteq S \times S$ is the transition relation, and L is the valuation, a function mapping each state with a set of atomic propositions true in that state. We require that there is at least one transition from every state.

A path π of a Kripke structure K is an infinite sequence of states $\pi = (\pi_0, \pi_1, \pi_2, \ldots)$ such that $R(\pi_i, \pi_{i+1})$ is true for every i. Given a path π, we will denote by π^{+i} the path starting from the i-th state in π. More formally:

$$\pi^{+i} = (\pi_i, \pi_{i+1}, \pi_{i+2}, \ldots)$$

The semantics of CTL* is then as follows:

- $(K, \pi) \models p \Leftrightarrow p \in L(\pi_0)$, where p is an atomic proposition.
- $(K, \pi) \models \neg f \Leftrightarrow (K, \pi) \not\models f$.
- $(K, \pi) \models f_1 \wedge f_2 \Leftrightarrow (K, \pi) \models f_1$ and $(K, \pi) \models f_2$.
- $(K, \pi) \models Ef \Leftrightarrow$ for some path π' in K, starting from π_0, $(K, \pi') \models f$.
- $(K, \pi) \models Xf \Leftrightarrow (M, \pi^{+1}) \models f$.
- $(K, \pi) \models f_1 U f_2 \Leftrightarrow \exists n \geq 0$ such that $(K, \pi^{+n}) \models f_2$, and for all i such that $0 \leq i < n$, we have $(K, \pi^{+i}) \models f_1$.

We say that $K \models f$ iff for every path π in K, such that $\pi_0 \in S_0$, we have $(K, \pi) \models f$.

A CTL* formula is in *normal form* when the operator $\neg$ modifies only atomic propositions.

ACTL* is a subset of CTL* in which the only path quantifier is A (when the formula is in normal form).

CTL [7] is a subset of CTL* in which each temporal operator (F, G, U, V, and X) must be immediately preceded by a path quantifier (A or E).

ACTL [12] is a subset of CTL in which the only path quantifier is A (when the formula is in normal form).

LTL [21] is a subset of CTL* is which there are no path quantifiers.

3. Vacuity

The intuitive notion of vacuity derives from that of propositional antecedent failure. Propositional antecedent failure means that a formula is trivially valid because some pre-condition is not satisfiable, where a pre-condition is the left-hand-side of an implication. Another way to think of it is to say that the right-hand-side of the implication does not affect the validity of the formula. This gives an intuitive extension of vacuity to any operator: vacuity occurs when one of the operands does not affect the validity of the formula. We use the notation $\varphi[\psi \leftarrow \psi']$ to denote the formula obtained from φ by replacing sub-formula ψ with ψ'.

Definition 1 (Affect). A sub-formula ψ of formula φ affects φ in model M if there is a formula ψ', such that the truth values of φ and $\varphi[\psi \leftarrow \psi']$ are different in M.

Definition 2 (Vacuity). Formula φ is vacuous in model M if there is a sub-formula ψ of φ such that ψ does not affect φ in M.

These definitions capture the intuitive notion of vacuity in a manner which is independent of a particular logic. However, they are not very useful when it comes to vacuity detection, because there are an infinite number of cases to check. In the remainder of this section we will show sufficient conditions on logics such that only a finite and small number of cases are required. We will first show that it is enough to check only a subset of the sub-formulas. Then, we will define logics with polarity for which it is enough to check the replacement of a sub-formula by either *true* or *false*.

3.1. Vacuity with respect to a minimal set of sub-formulas

In this section, we will show that vacuity can be checked by examining only a subset of the sub-formulas. These will be the sub-formulas which are minimal with respect to the sub-formula pre-order (denoted $\leq$). We assume that each sub-formula is unique. That is, we consider two separate occurrences of the same sub-formula to be different sub-formulas.

Lemma 3. *If $\chi \leq \psi \leq \varphi$, and ψ does not affect φ in model M, then χ does not affect φ in M.*

Proof: Assume ψ does not affect φ in M, but χ does affect φ. Then there is a formula χ' such that the truth value of $\varphi[\chi \leftarrow \chi']$ in M is different than that of φ in M. Since $\chi \leq \psi$ we get that

$$\varphi[\psi \leftarrow \underbrace{\psi[\chi \leftarrow \chi']}_{\psi'}] = \varphi[\chi \leftarrow \chi']$$

So $\varphi[\psi \leftarrow \psi'] = \varphi[\chi \leftarrow \chi']$. But the truth value of $\varphi[\chi \leftarrow \chi']$ in M is different than the truth value of φ in M. Thus the truth value of $\varphi[\psi \leftarrow \psi']$ in M is different than the truth value of φ in M, which means that ψ affects the value of φ in M, contradicting our assumption. □

For the sequel, we will need the following definitions:

Definition 4 (Vacuity with respect to a sub-formula). Let χ be a sub-formula of formula φ (denoted $\chi \leq \varphi$). Formula φ is χ-vacuous in model M if χ does not affect φ in M.

Definition 5 (Vacuity with respect to a set of sub-formulas). Let S be a set of sub-formulas of formula φ ($S \subseteq \{\chi \mid \chi \leq \varphi\}$). Formula φ is S-vacuous in model M if there exists $\chi \in S$ such that φ is χ-vacuous in M.

Definition 6 (Minimal sub-formulas). Let S be a set of sub-formulas. We define the minimal sub-formulas of S as:

$$\min(S) = \{\chi \in S \mid \textit{There is no } \chi' \in S \textit{ s.t. } \chi' \leq \chi\}$$

Theorem 7. *φ is S-vacuous iff φ is $\min(S)$-vacuous.*

Proof.

- ($\Rightarrow$) If φ is S-vacuous in M, there is a $\chi \in S$ that does not affect φ. Since S is finite and $\leq$ is a pre-order, there is a $\chi' \in \min(S)$ such that $\chi' \leq \chi$. Using Lemma 3, since χ does not affect φ in M, χ' does not affect φ in M either. This means that φ is $\min(S)$-vacuous in M.
- ($\Leftarrow$) If φ is $\min(S)$-vacuous in M, there is a $\chi \in \min(S) \subseteq S$ that does not affect φ in M, and therefore φ is S-vacuous in M. □

It follows immediately that to check vacuity of φ it is enough to check for vacuity with respect to only the minimal sub-formulas of φ. We will now show that for *logics with polarity*, it is enough to check the replacement of a sub-formula by either *true* or *false*.

3.2. Logics with polarity

In this section we will define *logics with polarity*. First, we will need a notation with which to denote all models M in which a formula φ is valid. We use the following notation:

$$[\![\varphi]\!] = \{M \mid M \models \varphi\}$$

We use the notation $[\![\varphi]\!]^c$ to denote the complement of $[\![\varphi]\!]$. We will now define what we mean by the polarity of an operand, then define operators with polarity, and finally define logics with polarity.

Definition 8 (Polarity of an operand). If σ is an n-ary operator in a logic, we say that the i-th operand of σ has positive(negative) polarity if for every fixing of $\varphi_1, \ldots, \varphi_{i-1}$, $\varphi_{i+1}, \ldots, \varphi_n$, and two formulas ψ_1, ψ_2, such that $[\![\psi_1]\!] \subseteq [\![\psi_2]\!]$ ($[\![\psi_2]\!] \subseteq [\![\psi_1]\!]$) we have that

$$[\![\sigma(\varphi_1, \ldots, \varphi_{i-1}, \psi_1, \varphi_{i+1}, \ldots, \varphi_n)]\!] \subseteq [\![\sigma(\varphi_1, \ldots, \varphi_{i-1}, \psi_2, \varphi_{i+1}, \ldots, \varphi_n)]\!]$$

We say that an operator has polarity if every one of its operands has some polarity (positive or negative).

Definition 9 (Logic with polarity). A logic with polarity is a logic in which every operator has polarity.

For example, the standard Boolean logic with operators $\vee$, $\wedge$, $\neg$ is a logic with polarity, since for every two formulas φ_1, φ_2, we have

$$\begin{aligned} [\![\varphi_1 \wedge \varphi_2]\!] &= [\![\varphi_1]\!] \cap [\![\varphi_2]\!] \\ [\![\varphi_1 \vee \varphi_2]\!] &= [\![\varphi_1]\!] \cup [\![\varphi_2]\!] \\ [\![\neg\varphi_1]\!] &= [\![\varphi_1]\!]^c \end{aligned}$$

This immediately implies that the operands of $\vee$ and $\wedge$ have positive polarity, and the single operand of $\neg$ has negative polarity.

An example of a logic which is not a logic with polarity is the standard Boolean logic with the addition of the exclusive-or operator:

$$[\![a \oplus b]\!] = ([\![a]\!] \cap [\![b]\!]^c) \cup ([\![a]\!]^c \cap [\![b]\!])$$

If we set $a = \mathit{true}$, we get:

$$[\![\mathit{true} \oplus b]\!] = [\![b]\!]^c$$

But if we set $a = \mathit{false}$, we get

$$[\![\mathit{false} \oplus b]\!] = [\![b]\!]$$

In the first case the polarity of the second operand is negative, and in the second positive. This means that $\oplus$ does not have polarity.

Proposition 10. *CTL* is a logic with polarity.*

Proof: First, note that the set of models that a CTL* formula satisfies is a subset of $\{(K, \pi) | \pi$ is a path in the structure $K\}$. As we have already shown, the standard Boolean operators $\vee$, $\wedge$ and $\neg$ all have polarity.

We now show that the single operand of the path quantifier E has a positive polarity. Given φ_1 and φ_2, where $[\![\varphi_1]\!] \subseteq [\![\varphi_2]\!]$, if $(K, \pi) \models E(\varphi_1)$ then there is a path π' in K that starts at the same state that π does, and $(K, \pi') \models \varphi_1$. This implies that $(K, \pi') \models \varphi_2$, and therefore $(K, \pi) \models E(\varphi_2)$.

We proceed to prove that both operands of the U operator have positive polarity.

1. Let us fix the second operand of the U operator to be some ψ. Given φ_1 and φ_2, where $[\![\varphi_1]\!] \subseteq [\![\varphi_2]\!]$, if $(K, \pi) \models \varphi_1 U \psi$, then there is an integer n, s.t.
   - for all $0 \leq i < n$ we have $(K, \pi^{+i}) \models \varphi_1$, and therefore, $(K, \pi^{+i}) \models \varphi_2$.
   - $(K, \pi^{+n}) \models \psi$

   Which proves that $(K, \pi) \models \varphi_2 U \psi$

2. Let us fix the first operand of U to be some ψ. Given φ_1 and φ_2, where $[\![\varphi_1]\!] \subseteq [\![\varphi_2]\!]$, if $(K, \pi) \models \psi U \varphi_1$, then there is an integer n, s.t.
   - for all $0 \leq i < n$ we have $(K, \pi^{+i}) \models \psi$
   - $(K, \pi^{+n}) \models \varphi_1$, and therefore $(K, \pi^{+n}) \models \varphi_2$.

   which proves that $(K, \pi) \models \psi U \varphi_2$

Finally, we show that the single operand of the operator X has positive polarity. Given φ_1 and φ_2, where $[\![\varphi_1]\!] \subseteq [\![\varphi_2]\!]$, if $(K, \pi) \models X(\varphi_1)$, then $(K, \pi^{+1}) \models \varphi_1$. By the assumption $(K, \pi^{+1}) \models \varphi_2$, meaning that $(K, \pi) \models X(\varphi_2)$, which concludes the proof. □

3.3. Vacuity detection in logics with polarity

In this section we will show that in logics with polarity it is enough to check the replacement of a sub-formula by either *true* or *false*.

First we define the polarity of a sub-formula, then we will present the main result of this section.

Definition 11 (Polarity of sub-formula). Given a formula φ, we define the polarity of sub-formulas of φ recursively:

- φ has positive polarity
- If $\chi = \sigma(\chi_1, \ldots, \chi_n)$, and χ is of positive(negative) polarity, then χ_i has positive polarity if the i-th operand of σ has a positive(negative) polarity, and χ_i has negative polarity otherwise.

Lemma 12. *In a logic with polarity, if $\chi \leq \varphi$, and χ is with a positive(negative) polarity, then if $[\![\chi]\!] \subseteq [\![\chi']\!]$ ($[\![\chi']\!] \subseteq [\![\chi]\!]$)*

$$[\![\varphi]\!] \subseteq [\![\varphi[\chi \leftarrow \chi']]\!]$$

Proof: The proof proceeds by induction on the size of the formula φ.

- *(base case:)* φ is an atom, so $\chi = \varphi$, χ has positive polarity, and $\varphi[\chi \leftarrow \chi'] = \chi'$. Therefore, if $[\![\chi]\!] \subseteq [\![\chi']\!]$, we get $[\![\varphi]\!] \subseteq [\![\varphi[\chi \leftarrow \chi']]\!]$.
- *(induction step:)* $\varphi = \sigma(\psi_1, \ldots, \psi_n)$. If $\chi = \varphi$, then we have the same as in the base case. Otherwise, we know that there is one i such that $\chi \leq \psi_i$. There are two cases:
  1. The i-th operand of σ is of positive polarity. In this case, the polarity of χ in ψ_i is as it is in φ, therefore, according to the induction hypothesis, $[\![\psi_i]\!] \subseteq [\![\psi_i[\chi \leftarrow \chi']]\!]$ and since the i-th operand of σ is of positive polarity, then by Definition 8 we have $[\![\varphi]\!] \subseteq [\![\varphi[\chi \leftarrow \chi']]\!]$.
  2. The i-th operand of σ is of negative polarity. In this case, the polarity of χ in ψ_i is the opposite of its polarity in φ. Therefore, by the induction hypothesis $[\![\psi_i]\!] \supseteq [\![\psi_i[\chi \leftarrow \chi']]\!]$ and since the i-th operand of σ is of negative polarity, we have $[\![\varphi]\!] \subseteq [\![\varphi[\chi \leftarrow \chi']]\!]$. □

In [4] we defined a subset of ACTL, and a set of *important* sub-formulas, and proved that in order to detect vacuity with respect to this set it is enough to show that $M \models \varphi[\psi \leftarrow \mathit{false}]$ where ψ is the minimal sub-formula of all the important sub-formulas (See section 5). In [14], Kupferman and Vardi expand on this result by showing that for CTL*, a formula φ is vacuous iff there is some minimal sub-formula ψ of φ such that M satisfies $\varphi[\psi \leftarrow \mathit{true}]$ iff M satisfies $\varphi[\psi \leftarrow \mathit{false}]$. We will now prove a very similar result that holds for all logics with polarity. The proof is practically the same as the one in [14]; we give it here for the sake of completeness.

We define the semantics of *true* and *false* as follows: $[\![\mathit{true}]\!] = \{M | M \text{ is a model}\}$ and $[\![\mathit{false}]\!] = \emptyset$.

Theorem 13. *Let ψ be a sub-formula of formula φ in a logic with polarity. Then, for every model M the following are equivalent:*

1. *ψ does not affect φ in M.*
2. $$M \models \varphi \Leftrightarrow M \models \varphi[\psi \leftarrow X]$$

   Where $X = \mathit{false}$ if $M \models \varphi$ and ψ is of positive polarity, or $M \not\models \varphi$ and ψ is of negative polarity. Otherwise $X = \mathit{true}$.

Proof:

– ($\Rightarrow$) ψ does not affect φ in M. This means that for every ψ', and specifically for $\psi' = \mathit{true}$ and $\psi' = \mathit{false}$, $M \models \varphi \Leftrightarrow M \models \varphi[\psi \leftarrow \psi']$ which concludes this part of the proof.
– ($\Leftarrow$) Note that for every ψ' we have:

$$[\![\mathit{false}]\!] \subseteq [\![\psi']\!] \subseteq [\![\mathit{true}]\!]$$

Two cases:

1. If ψ is of positive polarity, then using Lemma 12, we get
   $$[\![\varphi[\psi \leftarrow \mathit{false}]]\!] \subseteq [\![\varphi[\psi \leftarrow \psi']]\!] \subseteq [\![\varphi[\psi \leftarrow \mathit{true}]]\!]$$
   - If $M \models \varphi$ then by the assumption $M \models \varphi[\psi \leftarrow \mathit{false}]$, but by the containment above, this implies that for every ψ', $M \models \varphi[\psi \leftarrow \psi']$, meaning that ψ does not affect φ in M.
   - If $M \not\models \varphi$ then by the assumption $M \not\models \varphi[\psi \leftarrow \mathit{true}]$. By the same argument as above we get that for every ψ', $M \not\models \varphi[\psi \leftarrow \psi']$, meaning that ψ does not affect φ in M.
2. If ψ is of negative polarity, then using Lemma 12, we get
   $$[\![\varphi[\psi \leftarrow \mathit{true}]]\!] \subseteq [\![\varphi[\psi \leftarrow \psi']]\!] \subseteq [\![\varphi[\psi \leftarrow \mathit{false}]]\!]$$
   - If $M \models \varphi$ then by the assumption $M \models \varphi[\psi \leftarrow \mathit{true}]$, but by the containment above, this implies that for every ψ', $M \models \varphi[\psi \leftarrow \psi']$, meaning that ψ does not affect φ in M.

- If $M \not\models \varphi$ then by the assumption $M \not\models \varphi[\psi \leftarrow \mathit{false}]$, but by the containment above, this implies that for every ψ', $M \not\models \varphi[\psi \leftarrow \psi']$, meaning that ψ does not affect φ in M. □

In Section 3.1 we showed that it is enough to check vacuity for φ with respect to a subset of the sub-formulas φ. In this section we showed that for logics with polarity, it is enough to check the replacement of a sub-formula by either *true* or *false*. We now combine these two results in the following corollary:

Corollary 14. *In a logic with polarity, for a formula φ, and a set S of sub-formulas of φ, for every model M, the following are equivalent*:

– *φ is S-vacuous in M*
– *There is $\psi \in min(S)$ such that*:

$$M \models \varphi \Longleftrightarrow M \models \varphi[\psi \leftarrow X]$$

Where $X = \mathit{false}$ if $M \models \varphi (M \not\models \varphi)$ and ψ is of positive(negative) polarity. Otherwise, $X = \mathit{true}$.

This corollary gives us the ability to check vacuity of a formula in a logic with polarity by checking a relatively small number of other formulas, each of them of size not greater than that of φ. To be more precise, for S-vacuity, we need to check $|min(S)| + 1$ formulas:

1. Check φ.
2. For each sub-formula $\psi \in min(S)$, check the new formula $\varphi[\psi \leftarrow X]$. The value of X is either *true* or *false*, according to whether φ is valid or not, and the polarity of ψ.

Formula φ is S-vacuous iff at least one of these formulas has the same truth value as that of φ.

Since CTL* is a logic with polarity, we have shown the result of [14]: We can use a CTL* model checker to check vacuity in complexity $O(|\varphi| \cdot C_M(|\varphi|))$, where $C_M(n)$ is the complexity of checking a formula of size n in model M.

4. Interesting witnesses

The definition of vacuity refines the traditional distinction between valid and non-valid formulas with respect to a model M. We now classify formulas as either non-valid, vacuously valid, or non-vacuously valid. We would like to make the same refinement in the method we use to distinguish between the classes. Traditionally, we show that a formula is valid by means of a proof, and that a formula is non-valid by means of a counter-example. We will now define an *interesting witness*, which is the means we will use to show that a formula is non-vacuously valid. In this section we assume that the formula in question φ is valid in model M.

To make our definitions clear we use model checking problem of propositional logic as an example: The logic is the standard propositional logic on n boolean variables. A Model M is some non-empty subset of the set of assignments to the n variables. We say that $M \models \varphi$ if φ is true under all assignments in M. For example, if M is the set of all assignments, then $M \models \varphi$ iff φ is a tautology.

4.1. Pre-order and counter-examples

Before defining an interesting witness, we first formalize the notion of a counter-example. We will require two things from a counter-example to a formula.

1. That its existence proves the non-validity of the formula.
2. That it is small.

The second requirement is natural, since the smaller the counter-example is, the more useful it is to the user. Our approach is to define a pre-order on the set of models, such that non-validity on a smaller model always proves non-validity of a larger one. Then, we will require that a counter-example be a model which is minimal with respect to this pre-order.

Definition 15 (The natural pre-order of a logic). Given a logic L, we define the natural pre-order of the logic $\prec_{\mathrm{L}}$ on the set of models. $M' \prec_{\mathrm{L}} M$ iff for all $\varphi \in L$ we have that $M \models \varphi \Rightarrow M' \models \varphi$

Claim 16. The natural pre-order of propositional logic is containment.

Proof:

– ($\Rightarrow$) If $M' \subseteq M$, then any propositional formula that is valid for all assignments in M, will also be valid for all assignments in M'.
– ($\Leftarrow$) if $M' \not\subseteq M$, then there is at least one assignment $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ to the n variables that is in M' and not in M. We define the following propositional formula:

$$\varphi(x_1, x_2, \ldots, x_n) = \neg \bigwedge_{i=1}^{n} (x_i \wedge \alpha_i) \vee (\neg x_i \wedge \neg\alpha_i)$$

This formula is true on any assignment that is not equal to α, and false on α itself. Therefore we have that $M' \not\models \varphi$, and $M \models \varphi$. □

We can now define a counter-example:

Definition 17 (Counter-example). In logic L, a model C is a counter-example to φ in model M, if it satisfies the following conditions:

1. $C \prec_{\text{L}} M$.
2. $C \not\models \varphi$.
3. C is minimal w.r.t. $\prec_{\text{L}}$ among the models that satisfy properties 1 and 2.

It follows immediately from the definition that:

Claim 18. $M \not\models \varphi$ iff there exists a counter-example to φ in M.

We now return to our example of propositional logic, and show that counter-examples are as we expected:

Claim 19. In propositional logic, If C is a counter-example to formula φ in model M, then C is a model with one assignment.

Proof: If C is a counter-example, then $C \not\models \varphi$, therefore there is an assignment $\alpha \in C$ such that $\varphi(\alpha)$ is false. $A = \{\alpha\}$ is a model, $A \subseteq C$ and $A \not\models \varphi$. Since C is minimal w.r.t. $\subseteq$, we get that $C = A$, since there is no model A' such that $A' \subseteq A$ (we required that models are non-empty). □

4.2. Pre-orders and counter-examples in temporal logic

We have previously shown that for the case of propositional logic, Definition 17 captures our intuitive notion of what a counter-example is. Since the motivation of this paper is temporal logic, we would like to examine more closely the properties of a counter-example in some important temporal logics.

- LTL: in [22] Pnueli has proved that the natural pre-order for LTL according to Definition 15 is:
  $M_1 \prec_{\text{LTL}} M_2$ iff $L(M_1) \subseteq L(M_2)$, where $L(M) = \{\pi | \pi$ is a computation path in $M\}$. In LTL, if $M \not\models \varphi$, then there is a computation path π in M, such that $\{\pi\} \not\models \varphi$. Using the same arguments as in the proof of claim 19, we can show that a counter-example to φ will always be a model with one computation path in it.
- CTL and CTL*: Milner in [19] has proved that for CTL and CTL*, the natural pre-order is:
  $M_1 \prec_{\text{CTL}} M_2$ iff M_1 bi-simulates M_2. This means that CTL and CTL* have only trivial counter-examples that are the model itself. Indeed, the formula $EF(p)$ cannot be shown false by any model that has less behavior than the original, since we might have removed states where p was true. Note that even if we did have some method of specifying larger models as counter-examples, CTL would still be problematic. The formula $EF(p) \vee AG(q)$, cannot be proved false using neither a larger model, nor a smaller one.
- ACTL and ACTL*: Using the same proof as in [19], it can be shown that for ACTL and ACTL*, the natural pre-order is:
  $M_1 \prec_{\text{ACTL}} M_2$ iff M_2 simulates M_1. For these logics, it is difficult to characterize counter-examples. A model M always simulates a computation path π in it ($\pi \prec_{\text{ACTL}} M$), meaning

that computation paths *may* serve as counter-examples. For instance, a counter-example to the formula $AG(p)$ is a path on which $\neg p$ holds at some state. However, there are formulas and models for which a path cannot serve as a counter-example. The formula $AX(p) \vee AX(\neg p)$ cannot have a path as a counter-example, since on any deterministic path it will be evaluated to be true. A counter-example for this formula must be more complex. Buccafurri, Eiter, Gottlob and Leone have addressed this problem in detail in [6].

4.3. Interesting witnesses

In the case of a non-valid formula, a standard model checker provides a counter-example to the user. If the formula is valid, and using our vacuity checking procedure we can prove it non-vacuous, we would like to provide an interesting witness to the user, which is an analog of a counter-example — it proves non-vacuity, while a counter-example proves non-validity.

Definition 20 (Interesting witness with respect to a sub-formula). In logic L, a model W is a ψ-interesting witness to φ in M, if it satisfies the following conditions:

1. $W \prec_{\text{L}} M$.
2. $W \models \varphi$, and φ is not ψ-vacuous in W.
3. W is minimal w.r.t $\prec_{\text{L}}$ among the models that satisfy properties 1 and 2.

We now get an analogous claim to claim 18:

Claim 21. if $M \models \varphi$, then there exists a ψ-interesting witness W to φ in M iff φ is not ψ-vacuous in M.

Proof:

– ($\Rightarrow$) $W \prec_{\text{L}} M$, and $M \models \varphi$. Therefore, $W \models \varphi$. Since φ is not ψ-vacuous in W, there is a ψ' such that $W \not\models \varphi[\psi \leftarrow \psi']$. Again, since $W \prec_{\text{L}} M$, $M \not\models \varphi[\psi \leftarrow \psi']$. This means that φ is not ψ-vacuous in M.
– ($\Leftarrow$) The set of models that are smaller than M, and φ is not ψ-vacuous in them is non-empty, since M is such a model. Therefore any one of the minimal elements in this set is a ψ-interesting witness to φ in M. □

So, under the assumption that φ is valid in M, an interesting witness proves the non-vacuity of one sub-formula. Now we would like to have such proofs of more general non-vacuity, for sets of sub-formulas (and in particular , for the set of all sub-formulas). However, a single interesting witness will not always suffice.

Consider the formula $\varphi = (p \vee q)$, in a model M such that $M \models \varphi$. If φ is non-vacuous there is no single example which can show non-vacuity. In order to show p-non-vacuity, q must be set to 0, and in order to show q-non-vacuity, p must be set to 0. But since $M \models \varphi$, we cannot show an example in which both p and q are 0 simultaneously.

The naive solution would be to generate one interesting witness for every sub-formula. However, an interesting witness to one sub-formula, may also be an interesting witness to a different sub-formula. This is shown in the following proposition.

Proposition 22. *Assume $M \models \varphi$. If W is a ψ-interesting witness to φ in M, and $\psi \leq \chi \leq \varphi$, then W is also a χ-interesting witness to φ in M.*

Proof: Since W is a ψ-interesting witness to φ, ψ affects φ in W, and according to Lemma 3 χ affects φ in W, meaning that W is a χ-interesting witness to φ in M.

We shall now use Proposition 22 to get a more general result:

Corollary 23. *If $M \models \varphi$, and φ is not S-vacuous in M, then a set that has a ψ-interesting witness for every ψ in $min(S)$ also has a ψ-interesting witness for every ψ in S.*

4.4. Interesting witness generation in logics with polarity

In Section 3.3 we have shown that if our logic is a logic with polarity, then checking vacuity is much easier than the general case. The same result holds for interesting witness generation.

Lemma 24. *In a logic with polarity L, if $M \models \varphi$, $C \prec_L M$, and $\psi \leq \varphi$ is of positive(negative) polarity, then the following are equivalent:*

- $C \not\models \varphi[\psi \leftarrow X]$, where $X = false$ $(X = true)$.
- φ is not ψ-vacuous in C.

Proof:

- $(\Rightarrow)$ Since $M \models \varphi$, and $C \prec_L M$, we get that $C \models \varphi$. Now, since $C \not\models \varphi[\psi \leftarrow X]$, φ is not ψ-vacuous in C.
- $(\Leftarrow)$ Since $M \models \varphi$, and $C \prec_L M$, we get that $C \models \varphi$. Since φ is not ψ-vacuous in C, then using Theorem 13, we get that $C \not\models \varphi[\psi \leftarrow X]$. □

Theorem 25. *In a logic with polarity L, if $M \models \varphi$, and $\psi \leq \varphi$ is of positive(negative) polarity in φ, the following are equivalent:*

- *C is a counter-example to $\varphi[\psi \leftarrow X]$, Where $X = false$ $(X = true)$.*
- *C is a ψ-interesting witness to φ in M.*

The proof follows directly from Lemma 24, which proves that the two are equivalent, but omits the requirement of minimality. Adding this requirement to both of them obviously leaves them equivalent.

This theorem gives us the ability to easily generate interesting witnesses if we can generate counter-examples to the formulas in the logic: a ψ-interesting witness to φ in M is really a counter-example to one specific formula obtained by replacing ψ by *true* or *false*, depending on the polarity of ψ in φ. Note that if this formula is valid in M, then φ is ψ-vacuous in M.

If we now assume that we have a logic with polarity, and that we have a model checker for this logic that generates counter-examples to non-valid formulas, then we can enhance our model checker to have the following properties:

Enhanced model-checker. Given a formula φ, a model M, and a set S of sub-formulas of φ:

1. If $M \not\models \varphi$, generate a counter-example.
2. If $M \models \varphi$, and φ is S-vacuous in M, then output all sub-formulas in $\min(S)$ that do not affect φ in M.
3. If $M \models \varphi$, and φ is not S-vacuous in M, then generate $|\min(S)|$ interesting witnesses of M, such that for each $\psi \in S$, at least one of them is a ψ-interesting witness for φ in M.

The number of formulas checked if the formula is valid is $|\min(S)| + 1$, since for each ψ in $\min(S)$ we generate a formula to be model checked. If it is valid, then φ is S-vacuously valid. Otherwise, the model checker returns a counter-example, which is an ψ-interesting witness for φ. Since all formulas we generate are smaller in size than φ, we get that the complexity of the enhanced model checker is $O(|\min(S)| \cdot C_M(|\varphi|))$, where $C_M(n)$ is the complexity of model checking a formula of size n.

In the case where φ is S-vacuous, the enhanced model checker only outputs all the minimal sub-formulas that do not affect φ. However, the user may be interested in knowing exactly which of the sub-formulas in S are vacuous. To achieve this goal, we may need to check as many as $|S|$ formulas.

5. Practical vacuity detection and interesting witness generation

The motivation of this work was to provide an indication of vacuity and interesting witnesses to users of model checking. However, the complexity results of Sections 3.3 and 4.4 do not allow this in reasonable time. While the complexity of determining vacuity and generating interesting witnesses is only $|\varphi|$ times the complexity of model checking a formula of size $|\varphi|$, in practical terms this is too high, because a typical formula φ may take hours of CPU time to verify. We would like a method of determining vacuity and generating an interesting witness for a formula φ that is no more expensive than model checking φ.

In order to give an efficient solution, we will limit ourselves to a subset of ACTL, called w-ACTL, and to a subset of the sub-formulas, called *important sub-formulas* with respect to which we will check vacuity. We will then show that the complexity of checking vacuity of important sub-formulas in w-ACTL is exactly the complexity of the model checking φ in M. Finally, we show some examples.

5.1. Witness-ACTL (w-ACTL)

In this section we define witness-ACTL (w-ACTL), a subset of ACTL, which is in turn a subset of CTL. Informally, w-ACTL formulas are ACTL formulas in which for all binary operators ($\wedge$, $\vee$, AU, AV), at least one of the operands is a propositional formula. We divide the ACTL operators into propositional operators ($\neg$, $\wedge$, $\vee$) and temporal operators (AX, AG, AF, AU, AV), and call a formula which has only propositional operators, a *simple* formula. w-ACTL is the set of state formulas described by the following:

Definition 26 (w-ACTL).

1. Every simple formula is a state formula.
2. If f is a simple formula, χ is a state formula, and $\circ$ is some binary operator of ACTL ($\vee$, $\wedge$, AV, AU), then $f \circ \chi$ and $\chi \circ f$ are state formulas.
3. If χ is a state formula, and $\circ$ is some unary temporal operator of ACTL (AG, AF, AX), then $\circ(\chi)$ is a state formula.

The definition of w-ACTL may seem artificial at first glance. However, in our experience this is not the case. Most of the formulas written by users are w-ACTL formulas, which capture nicely the linear nature of most specifications.

5.2. *Important sub-formulas*

In order to be able to efficiently check vacuity and generate interesting witnesses for w-ACTL formulas, we have to restrict ourselves to a subset of the sub-formulas for which vacuity will be detected. Rather than being a drawback, we show that distinguishing between important and non-important sub-formulas is an advantage, as it is a reflection of how engineers use CTL to specify their designs.

We first define the set of *important sub-formulas* of a formula, with respect to which vacuity will be checked. Basically, the important sub-formulas are all the temporal (non-simple) ones.

Definition 27 (Important sub-formulas). Let φ be a w-ACTL formula, we define $Imp(\varphi)$ recursively:

1. If φ is simple, then $Imp(\varphi) = \{\varphi\}$.
2. If $\varphi = \psi \circ f$ or $\varphi = f \circ \psi$, where ψ is non-simple, and f is simple, then $Imp(\varphi) = \{\varphi\} \cup Imp(\psi)$.
3. If $\varphi = A[f_1 \text{ U } f_2]$ or $\varphi = A[f_1 \text{ V } f_2]$, where f_1 and f_2 are simple, then $Imp(\varphi) = \{\varphi\} \cup Imp(f_1)$.
4. If $\varphi = \circ(\psi)$, then $Imp(\varphi) = \{\varphi\} \cup Imp(\psi)$.

The choice made in item 3 above may seem arbitrary. The reason that only f_1 is important is that f_2 is the only operand that can cause vacuity. For A[f_1 U f_2], f_2 can cause vacuity of f_1 if it is always true immediately. However, f_1 cannot cause vacuity of f_2 because even if f_1 is always true forever, the AU operator still requires something of f_2: that eventually it occurs. For the AV operator, f_2 can cause vacuity of f_1 if it is always true forever, because then nothing is required of f_1. However, f_1 cannot cause vacuity of f_2 if it is always true immediately, because in that case, the AV operator still requires something of f_2: that it occurs at the same time.

We justify our choice of the *temporal* sub-formula of a binary operator as an important sub-formula as follows. The choice is simply a reflection of how engineers tend to use CTL to code a specification, as well as how they tend to design their hardware. For instance, consider the following specification:

$$AG(\mathit{request} \rightarrow AX(\mathit{req\_accepted} \rightarrow AXAX(\mathit{read\_busy} \vee \mathit{write\_busy}))) \tag{7}$$

which expresses the requirement that if a request is accepted (which happens or not one cycle after it appears), then two cycles later either the read_busy signal is asserted, or the write_busy signal is asserted. Logically, this is equivalent to the formula:

$$AG(\neg \mathit{request} \vee AX(\neg \mathit{req\_accepted} \vee AXAX(\mathit{read\_busy} \vee \mathit{write\_busy}))) \tag{8}$$

Vacuity of Formula 7, which detects that $M \models AG(\neg \mathit{request})$ would probably detect a problem in the model, because otherwise the signal called request is meaningless. However, a vacuity which detects that $M \models AG(AX(\neg \mathit{req\_accepted} \vee AXAX(\mathit{read\_busy} \vee \mathit{write\_busy})))$ is quite often useless to the engineer, as it is highly likely that she has designed her logic intentionally for this to be so, and prevents read_busy or write_busy from being asserted spuriously by not asserting req_accepted if there was not a request the previous cycle. Thus, for the binary operators, we have chosen the non-simple operand to be the important sub-formula.

5.3. Vacuity and interesting witnesses for w-ACTL formulas

Recall that φ is $Imp(\varphi)$-vacuous, if it is vacuous with respect to a sub-formula $\chi \in \min(Imp(\varphi))$ (Theorem 7). We now show that $\min(Imp(\varphi))$ has only one sub-formula in it, meaning that $Imp(\varphi)$-vacuity checking will be easy.

Claim 28. For every φ in w-ACTL the size of $min(Imp(\varphi))$ is one.

Proof: The proof proceeds by induction:

1. If φ is simple, then $|Imp(\varphi)| = 1$, and we are done.
2. If $\varphi = \circ(\psi)$, then $Imp(\varphi) = \{\varphi\} \cup Imp(\psi)$. Every sub-formula in $Imp(\psi)$ is a sub-formula of ψ and therefore of φ. This means that φ is not minimal in $Imp(\varphi)$, so $\min(Imp(\varphi)) = \min(Imp(\psi))$. Using the induction hypothesis $|\min(Imp(\psi))| = 1$.
3. If $\varphi = \psi_1 \circ \psi_2$, then $Imp(\varphi) = \{\varphi\} \cup Imp(\psi_1)$, or $Imp(\varphi) = \{\varphi\} \cup Imp(\psi_2)$. Using the same argument as above, φ is not in $\min(Imp(\varphi))$, meaning that $\min(Imp(\varphi)) = \min(Imp(\psi_1))$, or in the second case $\min(Imp(\varphi)) = \min(Imp(\psi_2))$. Again, using the induction hypothesis, we conclude that $|\min(Imp(\varphi))| = 1$. □

Since we are dealing with $ACTL$ formulas (negation can be applied to atomic propositions only), and because of the way we choose the important sub-formula (an important sub-formula is never an operand of "$\neg$"), we get that $\min(Imp(\varphi))$ always has a positive polarity in φ. We now define the formula *witness*(φ) as follows:

$$\mathit{witness}(\varphi) = \varphi[\min(Imp(\varphi)) \leftarrow \mathit{false}]$$

According to Corollary 14 and Theorem 25, it is enough to check *witness*(φ) in order to detect $Imp(\varphi)$-vacuity and generate an $Imp(\varphi)$-interesting witness. Given a model checker that can generate counter-examples for ACTL formulas, we can design an enhanced model checker for w-ACTL (see Section 4.4) with the following properties: Given a w-ACTL formula φ and model M,

1. If $M \not\models \varphi$ generate a counter example.
2. If $M \models \varphi$ and $M \models \mathit{witness}(\varphi)$, output that the formula passed vacuously.
3. If $M \models \varphi$ and $M \not\models \mathit{witness}(\varphi)$ output one interesting witness W, such that $W \models \varphi$, and for every important sub-formula ψ ($\psi \in Imp(\varphi)$), W is a ψ-interesting witness to φ in M.

5.4. *Detailed vacuity*

If $Imp(\varphi)$-vacuity is detected by our enhanced model checker, there is no indication of which of the pre-conditions caused the vacuity. As we said before, we can solve this by checking $|Imp(\varphi)| + 1$ formulas instead of just 2. However, in our specific case, we can actually check only $\log_2(|Imp(\varphi)|) + O(1)$ formulas. One can easily prove (using the same proof as in Claim 28) that the sub-formulas in $Imp(\varphi)$ are linearly ordered. Also, it follows directly from Lemma 3 that if $\chi \leq \psi$ then if φ is ψ-vacuous, then φ is also χ-vacuous. Combining these observations, we get that there is one minimal sub-formula $\psi \in Imp(\varphi)$, such that for all $\chi \in Imp(\varphi)$, φ is χ-vacuous iff $\chi \leq \psi$. This means that we can use binary search on $Imp(\varphi)$ to find this ψ. To Implement this, we need only check $\log_2(|Imp(\varphi)|) + O(1)$ formulas.

5.5. *Semantic refinements*

The careful reader will have noted that our definition of important sub-formulas will not detect vacuity in some basic cases, among them propositional antecedent failure. For instance, consider the following formula:

$$AG(\mathit{read\_request} \rightarrow \mathit{read\_enable}) \tag{9}$$

The vacuity detection (and witness generation) formula we generate for Formula 9 as defined above is:

$$AG(\mathit{false}) \tag{10}$$

Formula 10 is valid only in a model with no fair paths, and thus detects vacuity only in that case. Intuitively, this is not satisfying. We would like to be able to detect propositional antecedent failure.

Another problem is shown by the following Sugar[1] formula:

$$AG(\mathit{request} \rightarrow \mathit{next\_event}(\mathit{grant})(\mathit{acknowledge})) \tag{11}$$

Formula 11 expresses the requirement that the first grant after a request must be accompanied by an acknowledge. The ACTL normal form of Formula 11 is:

$$AG(\neg \mathit{request} \vee A[\mathit{grant} V(\neg \mathit{grant} \vee \mathit{acknowledge})]) \tag{12}$$

Thus, the vacuity detection formula for Formula 11 as defined above is:

$$AG(\neg \mathit{request} \vee A[\mathit{false} V \neg \mathit{grant} \vee \mathit{acknowledge}]) \tag{13}$$

Simplification of the above formula gives:

$$AG(\neg \mathit{request} \vee AG(\neg \mathit{grant} \vee \mathit{acknowledge})) \tag{14}$$

Formula 14 will not detect vacuity in the case that a request is never followed by a grant. Once again, this is not intuitively satisfying. The *next_event* operator expresses a kind of temporal implication, thus the failure of a grant to occur is a kind of temporal antecedent failure, and we would like to detect it.

We therefore expand our definition of important sub-formulas as follows:

1. If $\varphi = f_1 \vee f_2$ or $\varphi = f_2 \vee f_1$, where both f_1 and f_2 are simple, and the $\vee$ operator is derived from the use of the $\rightarrow$ operator by the user ($\varphi = \neg f_1 \rightarrow f_2$), then $Imp(\varphi) = \{\varphi\} \cup Imp(f_1)$.
2. If $\varphi = \mathit{next\_event}(f_1)(f_2)$, where both f_1 and f_2 are simple, then $Imp(\varphi) = \{\varphi\} \cup Imp(f_1)$.

5.6. Implementation details

In theory, a computation path is infinite and therefore, every example is infinite. In practice, however, the algorithm of [9] will sometimes give finite counter-examples, when a finite counter-example is enough to show that the formula is false. In every case but one, the finite counter-example given by [9] is "interesting enough" for our purposes. The exception is the AU operator. As a positive example to $A[\chi U \psi]$, we would like to see a trace on which ψ occurs, but [9] may give us a counter-example to $A[\mathit{false} U \psi]$ which ends before ψ has occurred. Therefore, we use $A[(AF\,\mathit{false}) U \psi]$ to get an infinite counter-example, just as [9] uses EG *true* to get an infinite example.

5.7. Examples

We now show the generation of an interesting witness formula. We use a typical Sugar formula as an example:

$$AG(request \rightarrow \ next\_event(data)[4](last\_data)) \tag{15}$$

Formula 15 states that last_data should be asserted with the fourth data after a request. Since last_data is considered to be non-simple (because it is the second operand of a next_event operator) the interesting witness formula is:

$$AG(request \rightarrow \ next\_event(data)[4](false)) \tag{16}$$

We convert Formula 16 into ACTL normal form:

$$\begin{aligned}&AG(\neg request \vee A[data V(\neg data \vee AX\, A[data V(\neg data \vee \\ &AX\, A[data V(\neg data \vee AX\, A[data V(\neg data \vee false)])])])])\end{aligned} \tag{17}$$

It is easy to see that Formula 17 is valid iff either a request never occurs, or no request is ever followed by four datas. Also, it is clear that if Formula 17 is found to be non-valid, the counter-example will be an interesting witness of Formula 15, on which a request followed by four datas will occur.

Now examine the following formula, which expresses the fact that we require q to occur an infinite number of times:

$$AG\ AF\ q \tag{18}$$

The interesting witness formula for Formula 18 is:

$$AG\ AF\ false \tag{19}$$

If Formula 18 is valid, it cannot be vacuously valid unless there are no fair paths, and indeed Formula 19 is non-valid in all non-empty models. The counter-example to Formula 19 will be a computation path, on which q will appear infinitely many times (because Formula 18 is valid).

6. Comparison with previous and related work

In this section, we compare our work with a previous version of our theory, and with related work.

6.1. Comparison with previous work

In a previous version of this paper [4], we required that an interesting witness to formula φ to be a single path, on which all important sub-formulas affect the validity of φ. This requirement was a result of the practical motivation of our original work. In this paper, an interesting witness is defined per sub-formula, so that interesting validity is demonstrated by multiple paths. The new definition is more natural, because, as we showed in Section 4.4, it allows us to guarantee interesting witnesses whenever we can guarantee counter-examples. It thus solves the problem raised by [14] of the following formula:

$$G(request \rightarrow F grant) \tag{20}$$

Consider a model M with two paths, one path that never satisfies *request* and a second path that always satisfies *grant*. If we require that an interesting witness be a single path, then there is no interesting witness to Formula 20 in such a model, despite the fact that there exists a counter-example to Formula 20 in any model in which Formula 20 is not valid.

6.2. Comparison with related work

Other works, including [2] and [20], have noted the problem of trivial validity, and shown how to avoid them using hand-written checks. Our original paper [4] was, we believe, the first attempt to formalize the notion of trivial validity, as well as the first attempt to detect it automatically under symbolic model checking.

Philosophers have also dealt with the problem of trivial validity. Relevance logic (also known as relevant logic) is a non-standard logic designed to prevent the paradoxes of material and strict implication. These occur when an antecedent is irrelevant to the consequent, as in the formula $p \rightarrow (q \rightarrow p)$ [1, 17]. While relevance logic deals with the problem by defining a new logic, our approach is different. We formalize the notion of vacuity and provide a method to detect it while leaving the logic itself unchanged.

In this paper, we use the term interesting witness to mean a computation path showing one non-trivial example of the validity of a valid formula. We are the first to use the term interesting witness, and the first to generate positive examples for valid non-existential formulas. In [13], Hojati, Brayton and Kurshan describe counter-example generation for model checking using CTL and language containment using L-automata [15]. In [9], Clarke, Grumberg, McMillan and Zhao describe the counter-example and witness generation algorithm of SMV [18]. Neither [13] nor [9] produce interesting witnesses for valid non-existential formulas.

In [14], Kupferman and Vardi presented an extension of [4] from w-ACTL to CTL*. Their results for vacuity are the same as those presented here, but they require that an interesting witness to a CTL* formula be a single path.

7. Conclusions and future work

We have formalized the notion of vacuity and interesting witness for logics with polarity. We have shown a practical method for detecting vacuity and generating interesting witnesses for w-ACTL formulas. As discussed above, the ability to detect vacuity and provide an interesting witness are extremely important in the practical application of model checking to industrial hardware designs.

Although the definition of vacuity we have presented is simple and elegant, there is still territory left uncovered. Pnueli [24] has suggested the following example: in a model M such that $M \models AGp$, the formula $AGAFp$ is valid, but our intuition tells us that the user is somehow "missing the point". A possible approach for solving this problem, is to refine our definition of vacuity. Instead of checking whether a sub-formula can be replaced by *any* other sub-formula, we will check whether it can be replaced by *some* "simpler" formula. The term "simpler" is a vague notion, but there are some immediate examples: p is simpler than $AF(p)$, $AG(p)$ is simpler than $AF(p)$, and perhaps even $AG(p) \vee AF(q)$ is simpler than $A[pUq]$.

A possible improvement could be done to the efficiency of vacuity checking. Instead of using the model checker as a black box, devise efficient model checking algorithms specifically for vacuity checking. A trivial enhancement would be to cache intermediate results in the model checker, since all the vacuity checking formulas are very similar.

Note

1. Sugar is a syntactic sugaring of CTL [7] formulas, and is the specification language used by the RuleBase formal verification tool. In [3] we outlined its basic features.

References

1. A.R. Anderson and N.D. Belnap, Jr., *Entailment: The Logic of Relevance and Necessity*, Princeton University Press, Princeton, Vol. 1, 1975, Vol. 2 (with J. Michael Dunn), 1992.
2. D. Beatty and R. Bryant, "Formally verifying a microprocessor using a simulation methodology," in *Design Automation Conference '94*, pp. 596–602.
3. I. Beer, S. Ben-David, C. Eisner, and A. Landver, "RuleBase: An industry-oriented formal verification tool," in *Proc. 33rd Design Automation Conference 1996*, pp. 655–660.
4. I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh, "Efficient detection of vacuity in ACTL formulas," in *CAV '97*, LNCS 1254, pp. 279–290.
5. M.C. Browne, E.M. Clarke, and O. Grumberg, "Characterizing finite Kripke structures in propositional temporal logic," *Theoretical Computer Science*, Vol. 59, No. 1–2, 1988, pp. 115–131.
6. F. Buccafurri, T. Eiter, G. Gottlob, and N. Leone, "On ACTL Formulas Having Deterministic Counterexamples," University of Technology Vienna Technical Report INFSYS RR-1843-99-01.
7. E.M. Clarke and E.A. Emerson, "Design and synthesis of synchronization skeletons using Branching Time Temporal Logic," in *Proc. Workshop on Logics of Programs*, Lecture Notes in Computer Science, Vol. 131 (Springer, Berlin, 1981) pp. 52–71.
8. E.M. Clarke and E.A. Emerson, "Characterizing properties of parallel programs as fixed-point," in *Seventh International Colloquium on Automata, Languages, and Programming*, Vol. 85 of LNCS, 1981.
9. E. Clarke, O. Grumberg, K. McMillan, and X. Zhao, "Efficient generation of counterexamples and witnesses in symbolic model checking," in *Design Automation Conference 1995*, pp. 427–432.
10. E.M. Clarke, O. Grumberg, and D. Peled, *Model Checking*, MIT Press, 1999.
11. E.A. Emerson and J.Y. Halpern, "'Sometimes' and 'Not Never' revisited: On branching versus linear time temporal logic," *Journal of the Association for Computing Machinery*, Vol. 33, No. 1, pp. 151–178, 1986. .
12. O. Grumberg and D. Long, "Model checking and modular verification," in J.C.M. Baeten and J.F. Groote, (Eds), *Procceedings of CONCUR '91: 2nd International Conference on Concurrency Theory*, Vol. 527 of LNCS, 1991.
13. R. Hojati, R.K. Brayton, and R.P. Kurshan, "BDD-based debugging of designs using language containment and fair CTL," in *CAV '93*, pp. 41–58.

14. O. Kupferman and M.Y. Vardi, “Vacuity Detection in Temporal Model Checking,” in *CHARME 99*, LNCS 1703, Springer-Verlag 1999.
15. R. Kurshan, Analysis of Discrete Event Coordination, LNCS 1990.
16. D. Long, “Model Checking, Abstraction and Compositional Verification,” Ph.D. Thesis, CMU, 1993.
17. Edwin D. Mares, “Relevance Logic,” *The Stanford Encyclopedia of Philosophy*, (Fall 1999 Edition), Edward N. Zalta (Ed.), URL=http://plato.stanford.edu/archives/win1999/entries/logic-relevance/.
18. K.L. McMillan, Symbolic Model Checking, Kluwer Academic Publishers, 1993.
19. R. Milner. “An algebraic definition of simulation between programs,” in *Proc. 2nd International Joint Conference on Artificial Intelligence*, British Computer Society, September 1971.
20. B. Plessier and C. Pixley, “Formal verification of a commercial serial bus interface,” in *International Phoenix Conference on Computers and Communications*, 1995, pp. 378–382.
21. A. Pnueli, “A temporal logic of concurrent programs,” *Theoretical Computer Science*, Vol. 13, pp. 45–60, 1981.
22. A. Pnueli, “Linear and Branching Structures in the semantics and logics of reactive systems,” in *Proc. 12th Int. Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science, Springer-Verlag, 1985.
23. A. Pnueli, N. Shankar, and E. Singerman, “Fair synchronous transition systems and their liveness proofs,” in A.P. Ravn and H. Rischel, (Eds), *FTRTFT 98: 5th International School and Symposium on Formal Techniques in Real-time and Fault-tolerant Systems*, Lecture Notes in Computer Science, Springer-Verlag, 1998.
24. A. Pnueli, Question from the Audience, CAV ’97.
25. G. Shurek and O. Grumberg, “The computer-aided modular framework—motivation, solutions and evaluation criteria,” in *Workshop on Computer Aided Verification*, 1990.