A Framework for Ranking Vacuity Results

Shoham Ben-David and Orna Kupferman

School of Engineering and Computer Science
The Hebrew University, Jerusalem, Israel

Abstract. *Vacuity* detection is a method for finding errors in the model-checking process when the specification is found to hold in the model. Most vacuity algorithms are based on checking the effect of applying mutations on the specification. It has been recognized that vacuity results differ in their significance. While in many cases vacuity results are valued as highly informative, there are also cases in which the results are viewed as meaningless by users. As of today, there is no study about ranking vacuity results according to their level of importance, and there is no formal framework or algorithms for defining and finding such ranks. The lack of framework often causes designers to ignore vacuity information altogether, potentially causing real problems to be overlooked.

We suggest and study such a framework, based on the *probability* of the mutated specification to hold in a random computation. For example, two natural mutations of the specification $G(req \rightarrow Fready)$ are $G(\neg req)$ and $GFready$. It is agreed that vacuity information about satisfying the first mutation is more alarming than information about satisfying the second. Our methodology formally explains this, as the probability of $G(\neg req)$ to hold in a random computation is 0, whereas the probability of $GFready$ is 1. From a theoretical point of view, our contribution includes a study of the problem of finding the probability of LTL formulas to be satisfied in a random computation and the existence and use of 0/1-laws for fragments of LTL. From a practical point of view, we propose an efficient algorithm for estimating the probability of LTL formulas, and argue that ranking vacuity results according to our probability-based criteria corresponds to our intuition about their level of importance.

1 Introduction

In temporal logic model checking, we verify the correctness of a system with respect to a desired behavior by checking whether a mathematical model of the system satisfies a temporal-logic formula that specifies this behavior [10]. When the formula fails to hold in the model, the model checker returns a counterexample — some erroneous execution of the system [11]. When the formula holds in the system on the other hand, most model-checking tools provide no additional information. While this might seem like a reasonable policy, there has been growing awareness to the need of suspecting the result in the case model checking succeeds, since errors may hide in the modeling of the system or the behavior [19].

As an example, consider the specification $\varphi = G(req \rightarrow Fready)$ ("every request is eventually followed by a ready signal"). One should distinguish between satisfaction of φ in a model in which requests are never sent, and satisfaction in which φ's

precondition is sometimes satisfied. Evidently, the first, vacuous, type of satisfaction suggests a suspicious behavior of the model.

In [2], Beer et al. suggested a first formal treatment of vacuity. The definition of vacuity according to [2] is based on mutations applied to the specification, checking whether the model actually satisfies a specification that is stronger than the original one. In the last decade, the challenge of detecting vacuous satisfaction has attracted significant attention (see [1, 4, 5, 7, 9, 14, 16, 17, 20] for a partial list).

Different definitions for vacuity exist in the literature and are used in practice. The most commonly used ones are based on the "mutation approach" of [2]. We focus here on its generalization, as defined in [21]. Consider a model M satisfying a specification φ. A subformula ψ of φ *does not affect* (the satisfaction of) φ in M if M satisfies also the (stronger) formula $\varphi[\psi \leftarrow \bot]$, obtained from φ by changing ψ in the most challenging way. Thus, if ψ appears positively in φ, the symbol $\bot$ stands for *false*, and if ψ is negative, then $\bot$ is *true*[1]. We say that M satisfies φ vacuously if φ has a subformula that does not affect φ in M. Consider for example the formula $\varphi = G(req \rightarrow Fready)$ described above. In order to check whether the subformula *ready* affects the satisfaction of φ, we model check $\varphi[ready \leftarrow false]$, which is equivalent to $G\neg req$. That is, a model with no requests satisfies φ vacuously. In order to check whether the subformula *req* affects the satisfaction of φ, we model check $\varphi[req \leftarrow true]$. This is equivalent to $GFready$, thus a model with infinitely many *ready* signals satisfies φ vacuously.

In [5], Chechik et al. observe that in many cases, vacuities detected according to the above definition are not considered a problem by the verifier of the system. A similar observation was reported later in [3]. For example consider a specification similar to φ discussed above, describing the desired behavior of a traffic light [5]: $\psi = G(car \rightarrow Fgreen)$, stating that whenever a car approaches the traffic light, it will eventually be granted green light. In many traffic light systems however, green light is given in a "round robin" manner, granting green light periodically, whether a car is waiting or not. The mutation $\psi[car \leftarrow \bot] = GFgreen$ is thus satisfied in such a system, causing ψ to be wrongly declared as vacuous. One may argue that ψ is indeed too weak for the traffic light above; we note however, that it is common to have a set of generic formulas which are applied to a variety of systems. A set of formulas for traffic light systems should include ψ and not its mutation $GFgreen$, to accommodate different types of systems.

"False alarms" as above should be avoided as much as possible. When a significant portion of vacuity results are false alarms, it might cause the user to disregard vacuity information altogether, potentially causing real problems to be ignored. In fact, the tendency of designers to decry vacuity results has led some vacuity tools to report only particular cases of vacuity, hoping not to lose the users' patience [6]. In this paper we suggest and study a formal framework for ranking vacuity results according to their level of importance.

Earlier work on identifying false alarms in vacuity includes two approaches: In [5], the authors suggest to identify as vacuous only specifications whose satisfaction depend solely on the environment of the system. While such cases should definitely

[1] The above definition assumes that ψ appears once in φ, or at least that all its occurrences are of the same polarity; a definition that is independent of this assumption replaces ψ by a universally quantified proposition [1]. Our methodology here applies also to this definition.

be considered vacuous, we believe that there exist real vacuities that do not fulfill the condition of [5]. In [3], the authors focus on detecting vacuities that are due to *temporal antecedent failure*, claiming that those are never debatable. In fact, already in the original definition of vacuity in [2], not all subformulas of the specification are checked for affecting the satisfaction, as the authors claim that some possibilities do not result in interesting vacuities. No attempt has been made in [2, 3] to formally justify this claim or to argue that the classification is exhaustive.

Our framework for ranking vacuity results is based on the *probability* of the mutated specification to hold on a random computation. To see the idea behind the framework, let us consider the mutations $G(\neg req)$ and $GF ready$ of the specification φ discussed above. Consider a random computation π. The probabilistic model we assume is that for each atomic proposition p and for each state in π, the probability of p to hold in the state is $\frac{1}{2}$. In this model, the probability of $G(\neg req)$ to hold in a random computation is 0, whereas the probability of $GF ready$ to hold is 1. We argue that the lower is the probability of the mutation to hold in a random computation, the higher the vacuity rank should be. In particular, vacuities in which the probability of the mutation is 0, as is the case with $G(\neg req)$, should get the highest rank and vacuities in which the probability is 1, as is the case with $GF(ready)$, should get the lowest rank. Note that we do not claim that low-rank vacuities are not important. In particular, satisfying $GF(ready)$ suggests that the system satisfies some fairness condition that the designer is not aware of. In an ideal world, we would have liked to examine all vacuity results. Experience shows, however, that designers simply cannot address the overwhelmingly large information generated by vacuity checkers, so ranks are essential. If we can only check one alarm, we would better check the $G(\neg req)$ one, rather than $GF(ready)$.

In Section 4, we give further examples and evidences that our probability-based criteria indeed corresponds to our intuition about the level of importance. In particular, our ranking agrees with, unifies and generalizes the approaches in all earlier work. We also suggest a refinement of the criteria that takes into account the probability of the original specifications, and a refinement of the analysis for the case of invariants.

Let us elaborate on our framework and the theoretical and practical challenges it involves. For a specification φ, let $Pr(\varphi)$ denote the probability for φ to be satisfied in a random computation, as formalized above. Glebskii et al. in [15], and later Fagin in [13], proved a 0/1-law for first order logic. That is, for φ in first order logic, the probability of φ to be satisfied in a random model is either 0 or 1. It is not hard to see that a 0/1-law does not hold for LTL. For example, for an atomic proposition p, we have that $Pr(p) = \frac{1}{2}$. We study the problem of finding $Pr(\varphi)$, for φ in LTL, and show that it is PSPACE-complete. The upper bound follows easily from the probabilistic model-checking algorithm of [12]. The algorithm in [12] is complicated. For the case we only want to find out whether $Pr(\varphi) = 0$ or whether $Pr(\varphi) = 1$, we suggest a much simpler algorithm, which analyzes the connected components of a nondeterministic Büchi automaton for φ.[2] We also show that large fragments of LTL do respect the 0/1-law. In particular, all invariants (formulas for the form $G\psi$)

[2] Working with a deterministic automaton for φ, it is possible to calculate $Pr(\varphi)$ easily. The novelty of our algorithm is that for the polar cases of 0 and 1, it is possible to work with the nondeterministic one.

respect it. Note that for these fragments, our simpler algorithm solves the problem of finding $Pr(\varphi)$.

The high complexity of finding $Pr(\varphi)$ leads us to suggest a heuristic for estimating it in a simple syntax-based bottom-up algorithm. We argue that not only our heuristic often calculates $Pr(\varphi)$ precisely, but that its errors are actually welcome. Specifically, we distinguish between two cases in which our heuristic errs. The first is when it returns a probability that is *lower* than $Pr(\varphi)$. In this case, the error is a result of a peculiar structure of the specification, like subformulas that are valid. We are thus happy to declare the specification as one that requires a check, which is indeed the fate of mutations with low probability. The second is when our heuristic returns a probability that is higher than $Pr(\varphi)$. Here, the heuristic may cause real vacuities to get a low rank. We show, however, that these cases are rare. As discussed in Section 6, we believe that a framework for ranking vacuity results, and in particular the one suggested here, offers a useful and simple-to-implement upgrade of existing vacuity-checking tools.

Due to the lack of space, proofs of theorems appear in the appendix.

2 Preliminaries

2.1 LTL and Vacuity

For a finite alphabet Σ, an infinite *word* $w = \sigma_0 \cdot \sigma_1 \cdots$ is an infinite sequence of letters from Σ. We use Σ^ω to denote the set of all infinite words over the alphabet Σ. A *language* $L \subseteq \Sigma^\omega$ is a set of words. When $\Sigma = 2^{AP}$, for a set AP of atomic propositions, each infinite word corresponds to a *computation* over AP.

The logic *LTL* is a linear temporal logic [23]. LTL formulas describe languages over the alphabet 2^{AP}, and are constructed from a set AP of atomic propositions using the Boolean operators $\neg$ and $\wedge$, and the temporal operators X ("next time") and U ("until"). The semantics of LTL is defined with respect to computations over AP. Typically, computations are given by means of a *Kripke structure* – a state-transition graph in which each state is labeled by a set of atomic propositions true in this state. The Kripke structure models a system, and we also refer to it as "the model". For an LTL formula φ, we use $\mathcal{L}(\varphi)$ to denote the language of φ, namely the set of computations that satisfy φ.

Given a model M and an LTL formula φ, the model-checking problem is to decide whether all the computations of M satisfy φ, denoted $M \models \varphi$. Describing formulas, we use the standard abbreviations $\vee$ and $\rightarrow$ for Boolean operators and F ("eventually") and G ("always") for temporal operators.

Let φ be an LTL formula and ψ an occurrence of a subformula of φ. We say that ψ has a positive polarity in φ if ψ is in the scope of an even number of negations in φ. Dually, ψ has a negative polarity in φ if ψ is in the scope of an odd number of negations in φ. Note that the definition assumes a syntax of LTL with only $\neg, \wedge, X$ and U. In particular, since $\alpha \rightarrow \beta$ is equivalent to $\neg\alpha \vee \beta$, the polarity of α there is negative.

Consider a model M and an LTL formula φ. *Vacuity* is a sanity check applied when $M \models \varphi$, aiming to find errors in the modeling of the system or the behavior by M and φ. Most vacuity algorithms proceed by checking whether M also satisfies formulas obtained by strengthening φ.

For a formula φ and an occurrence ψ of a subformula of φ, let $\varphi[\psi \leftarrow \bot]$ denote the formula obtained by replacing ψ by $\bot$ in φ, where $\bot = \mathit{false}$ if ψ has a positive polarity in φ, and $\bot = \mathit{true}$ otherwise. Consider a model M such that $M \models \varphi$. We say that ψ *does not affect the satisfaction* of φ in M if $M \models \varphi[\psi \leftarrow \bot]$. We use the definition of vacuity given in [21]: the formula φ is *vacuous* in M if there exists a subformula ψ of φ that does not affect the satisfaction of φ in M. Note that the definition of [21] refers to a single occurrence of ψ in φ, and it can be easily extended to multiple occurrences all with the same polarity. A definition that is independent of this assumption replaces ψ by a universally quantified proposition [1]. As we demonstrate in Example 5, our framework here can be extended to the definition in [1].

2.2 The Probabilistic Setting

Given a set of elements S, a *probability distribution* on S is a function $\mu : S \to [0,1]$ such that $\Sigma_{s \in S}\ \mu(s) = 1$. The *support* set of μ, denoted $\mathit{supp}(\mu)$, is the set of all elements s for which $\mu(s) > 0$.

A *finite Markov chain* is a tuple $M = \langle V, p \rangle$, where V is a finite set of states and $p : V \times V \to [0,1]$ is a function. For $v \in V$, let $p_v : V \to [0,1]$ be such that $p_v(u) = p(v,u)$ for all $u \in V$. We require that for all $v \in V$, the function p_v is a probability distribution on V. A *probabilistic labeled structure* is $\mathcal{S} = \langle \Sigma, V, p, p_{init}, \tau \rangle$, where $\langle V, p \rangle$ is a Markov chain, $p_{init} : V \to [0,1]$ is a probability distribution on V that describes the probability of a computation to start in the state, and $\tau : V \to \Sigma$ maps each state to a letter in Σ.

A *path* in a Markov chain M is an infinite sequence $\pi = v_0, v_1, v_2, \ldots$ of vertices. The probability of π from v_0 is $\Pi_{i \geq 0} p(v_i, v_{i+1})$. Once we add a probability distribution also on the starting state of paths, the probabilities of events are uniquely defined, where an *event* is a measurable set of paths.

3 Probability of LTL properties

Consider an alphabet Σ. A random word over Σ is a word in which for all indices i, the i-th letter is drown uniformly at random. In particular, when $\Sigma = 2^{AP}$, then a random computation π is such that for each atomic proposition q and for each position in π, the probability of q to hold in the position is $\frac{1}{2}$. An alternative definition of our probabilistic model is by means of the probabilistic labeled structure $\mathcal{U}_\Sigma$, which generates computations in a uniform distribution. Formally, $\mathcal{U}_\Sigma = \langle \Sigma, \Sigma, p, p_{init}, \tau \rangle$, where for every $\sigma, \sigma' \in \Sigma$, we have $p(\sigma, \sigma') = p_{init}(\sigma) = \frac{1}{|\Sigma|}$, and $\tau(\sigma) = \sigma$. Thus, $\mathcal{U}_\Sigma$ is a clique with a uniform distribution. We define the probability of a language $\mathcal{L} \subseteq \Sigma^\omega$, denoted $Pr(\mathcal{L})$, as the probability of the event $\{\pi : \pi \text{ is a path in } \mathcal{U}_\Sigma \text{ that is labeled by a word in } \mathcal{L}\}$. Similarly, for an LTL formula φ, we define $Pr(\varphi)$ as the probability of the event $\{\pi : \pi \text{ is a path in } \mathcal{U}_{2^{AP}} \text{ that satisfies } \varphi\}$. For example, the probabilities of Xp, Gp, and Fp are $\frac{1}{2}, 0$, and 1, respectively.

Using $\mathcal{U}_\Sigma$ we can reduce the problem of finding $Pr(\varphi)$ to φ's model checking. Results on probabilistic LTL model checking [12] then imply the upper bound in the theorem below. For the lower bound, we do a generic reduction, and prove hardness already for the easier problem, of deciding whether the probability is 0 or 1.

Theorem 1. *The problem of finding the probability of LTL formulas is PSPACE-complete.*

First order logic respects the 0/1-law: the probability of a formula to be satisfied in a random model is either 0 or 1 [15, 13]. It is not hard to see that a 0/1-law does not hold for LTL. For example, for an atomic proposition p, we have that $Pr(p) = \frac{1}{2}$. We now show that invariants do satisfy the 0/1-law. On the other hand, trigger formulas, which can be viewed as "conditional invariants" do not respect it, even when the condition holds infinitely often. A trigger formula is of the form $r \mapsto \varphi$, where r is a regular expression and φ is an LTL formula. A computation π satisfies the formula $r \mapsto \varphi$ if for every index i, if the prefix $\pi[1, i]$ is in the language of r, then the suffix π^i satisfies φ. We use $\mathcal{L}(r)$ to denote the language of r. For the full definition of trigger formulas and their vacuity see [4].

Theorem 2. *Formulas of the form $G\varphi$ satisfy the 0/1-law. On the other hand, formulas of the form $r \mapsto \varphi$ need not satisfy the 0/1-law even when $\mathcal{L}(r)$ is infinite.*

3.1 A Practical Approach

By Theorem 1, calculating the exact probability of an LTL formula, and even only deciding whether it is 0 or 1, is PSPACE-complete. Moreover, the algorithm of [12] iteratively eliminates the temporal operators in φ, while preserving the probability of its satisfaction, and is complicated to implement. In this section we describe a simple linear-time method for estimating $Pr(\varphi)$. The estimation is based on the syntax of the formula and thus ignores possible dependencies between subformulas. As it turns out, however, the estimation often serves our purpose better than the precise algorithm, as its errors originate from a peculiar structure of the specification, which is something we want to draw the attention of the designer to. We elaborate more on this in the sequel.

Definition 3 (Estimation) *The estimated probability of an LTL formula φ, denoted $Est(\varphi)$, is defined by induction on the structure of φ as follows.*

- $Est(false) = 0$
- $Est(true) = 1$
- $Est(p) = \frac{1}{2}$
- $Est(\neg\varphi) = 1 - Est(\varphi)$
- $Est(\varphi \wedge \psi) = Est(\varphi) \cdot Est(\psi)$
- $Est(X\varphi) = Est(\varphi)$
- $Est(\varphi U \psi) = \frac{Est(\psi)}{1-(1-Est(\psi))\cdot Est(\varphi)}$

The estimation for $Est(\varphi U\psi)$ needs some explanation. We use the fixed-point characterization $\varphi U\psi = \psi \vee [\varphi \wedge X(\varphi U\psi)]$. For calculating probability, we have to make the two disjuncts disjoint, resulting in $\varphi U\psi = \psi \vee [(\neg\psi) \wedge \varphi \wedge X(\varphi U\psi)]$. Accordingly, $Est(\varphi U\psi) = Est(\psi) + [(1 - Est(\psi)) \cdot Est(\varphi) \cdot Est(\varphi U\psi)]$.

Isolating $Est(\varphi U\psi)$ results in $Est(\varphi U\psi) = \frac{Est(\psi)}{1-(1-Est(\psi))\cdot Est(\varphi)}$ as stated in Definition 3. It is interesting to note the following special cases: if $Est(\psi) = 0$, then $Est(\varphi U\psi) = 0$ as well, and if $Est(\psi) = 1$ then $Est(\varphi U\psi) = 1$. The probability of $\varphi U\psi$ is 1 also in the case where $Est(\varphi) = 1$ and $Est(\psi) \neq 0$. This is the

case in the abbreviation $F\psi = (true\ U\psi)$, where $Est(F\psi)$ is 0 if $Est(\psi) = 0$ and is 1 otherwise. Accordingly, as $G\psi = \neg F\neg\psi$, we have that $Est(G\psi)$ is 1 if $Est(\psi) = 1$ and is 0 otherwise. Finally, by applying De-Morgan rules, we have that $Est(\varphi \vee \psi) = Est(\varphi) + Est(\psi) - Est(\varphi) \cdot Est(\psi)$.

To see why the estimation is not precise, consider for example the specification $p \wedge \neg p$. By definition, $Est(p \wedge \neg p) = Est(p) \cdot Est(\neg p) = \frac{1}{2} \cdot (1 - \frac{1}{2}) = \frac{1}{4}$. On the other hand, $p \wedge \neg p$ is unsatisfiable and thus $Pr(p \wedge \neg p) = 0$. Note that indeed the estimation errs only in steps associated with a binary operator where the two subformulas are not independent. In particular, no errors are introduced in steps associated with unary operators. For example, if $Est(\psi) = Pr(\psi)$, then also $Est(F\psi) = Pr(F\psi)$. Indeed, if $Pr(\psi) = 0$, then $Pr(F\psi)$ is 0 too, and if $Pr(\psi) > 0$, then ψ eventually holds in a random computation, thus $Pr(F\psi)$ is 1.

Note that if we want to find $Est(\varphi[\psi \leftarrow \bot])$ for several different subformulas ψ of φ, we can reuse information already calculated for common parts of the formula, as mutations differ from each other only by some subformulas.

Sometimes, the user does not care for the exact probability and only wants to know whether it is 0 or 1. In particular, in the context of vacuity, a user may want the algorithm to report only those mutations that are satisfied with probability 0, or those that are satisfied with probability strictly smaller than 1. In this case, we can simplify the estimation algorithm further and abstract all values $0 < \gamma < 1$ as shown below. This enables the reasoning to avoid the arithmetics that the estimation algorithm involves.

Definition 4 (Three-Valued Estimation) *The 3-valued estimated probability of an LTL formula φ, denoted $ETpr(\varphi)$, is defined by induction on the structure of φ as follows.*

- $ETpr(false) = 0$
- $ETpr(true) = 1$
- $ETpr(p) = \gamma$
- $ETpr(\neg\varphi) = \begin{cases} 1 & \text{if } ETpr(\varphi) = 0 \\ 0 & \text{if } ETpr(\varphi) = 1 \\ \gamma & \text{otherwise} \end{cases}$
- $ETpr(\varphi \wedge \psi) = \begin{cases} 1 & \text{if } ETpr(\varphi) = 1 \text{ and } ETpr(\psi) = 1 \\ 0 & \text{if } ETpr(\varphi) = 0 \text{ or } ETpr(\psi) = 0 \\ \gamma & \text{otherwise} \end{cases}$
- $ETpr(X\varphi) = ETpr(\varphi)$
- $ETpr(\varphi U\psi) = \begin{cases} 0 & \text{if } ETpr(\psi) = 0 \\ 1 & \text{if } (ETpr(\psi) = 1) \text{ or } (ETpr(\varphi) = 1 \text{ and } ETpr(\psi) \neq 0) \\ \gamma & \text{otherwise} \end{cases}$

Theorem 5 below states that the three-valued estimation abstracts the estimation correctly.

Theorem 5. *Consider an LTL formula φ. If $Est(\varphi) = 0$, then $ETpr(\varphi) = 0$; if $Est(\varphi) = 1$ then $ETpr(\varphi) = 1$; and if $0 < Est(\varphi) < 1$, then $ETpr(\varphi) = \gamma$.*

3.2 Analyzing the Estimated Results

The table in Fig. 1 analyzes the different possible relations between the estimated and the precise probabilities. We use γ to stand for a value $0 < \gamma < 1$. For example, Line 3 in the table refers to cases in which the estimated calculation returns the value 0, whereas the precise probability is some $0 < \gamma < 1$. The table also includes examples corresponding to the different cases and some observation about the cases. We elaborate on these observations below.

	$Est(\varphi)$	$Pr(\varphi)$	Example	Observation
1	0	0	Gq	The estimation is precise
2	0	1	$G(p \vee \neg p)$	The estimation is more helpful
3	0	γ	—	This case is impossible
4	1	0	$F(p \wedge \neg p) \vee Gq$	This case is rare
5	1	1	Fq	The estimation is precise
6	1	γ	—	This case is impossible
7	γ	0	$qU(p \wedge \neg p)$	This case is rare
8	γ	1	$(p \vee \neg p)Uq$	The estimation is more helpful
9	γ	γ	pUq	The three-valued abstraction is precise

Fig. 1. Comparing the estimated probability with the precise probability.

We first observe that the estimation has only a one-sided error: the estimation algorithm may decide that the probability is γ whereas the precise probability is 0 or 1, but never the other way round. Indeed, γ is introduced to $Est(\varphi)$ only if some of the subformulas have probability that is not 0 or 1. It follows that the cases described in Lines 3 and 6 are impossible. Other favorable cases are those described in Lines 1 and 5, where the precise and estimated probabilities coincide.

When the estimation is not precise, it is due to some relationship between different parts of the formula. As described above, the definition of Est ignores such a relationship. Consider for example the invariant $G(p \vee \neg p)$. While $Pr(p \vee \neg p) = 1$, we have that $Est(p \vee \neg p) = \frac{3}{4}$. As a result, while $Pr(G(p \vee \neg p)) = 1$, our estimation algorithm returns $Est(G(p \vee \neg p)) = 0$. When the estimation is lower than the precise probability, as is the case in Lines 2 and 8, we are going to report the vacuity result with a higher rank than it seemingly "deserves". Note that this would rightfully draw the attention of the user to the peculiar structure of the formula. We conclude that when the estimation is higher than the precise one, the result we get is more helpful than calculating the precise probability.

The cases where the estimation is higher than the real probability, as is the case in Lines 4 and 7, are the least favorable ones, since here too, there is some problem in the formula, but the vacuity result might get a lower ranking than it should. For example, $Pr(F(p \wedge \neg p) \vee Gq) = 0$ but $Est(F(p \wedge \neg p) \vee Gq) = 1$. Note however, that the subformula causing the discrepancy, $F(p \wedge \neg p)$ in our example, is not the cause of vacuity – the formula holds in the model *in spite of* this odd subformula. Typically, another mutation of the formula, which refers to the odd subformula, would be checked. This latter check would reveal the real problem and would be estimated correctly by our algorithm. In our example, the other check replaces $(p \wedge \neg p)$ with $\bot$, leaving the rest of the formula untouched.

Finally, note that while Line 9 is favorable in the three-valued estimation, if we care for the exact value of the probability, this case can be refined further to the cases $Est(\varphi) > Pr(\varphi)$, $Est(\varphi) = Pr(\varphi)$, and $Est(\varphi) < Pr(\varphi)$. Again, only the first one is bothering, as it may lead to a low ranking of a mutation that exhibit a real problem. Here too, the problem is less alarming than one in which $Pr(\varphi) = 0$, and, since $Pr(\varphi)$ is low, is rare.

4 Ranking Vacuities

In Section 3 we studied the probability of LTL formulas to hold in a random computation, and suggested algorithms for calculating the probability and estimating it. In this section we suggest several applications of a probability-based criteria for ranking of vacuity results. We focus on three approaches. In all of them, the idea is that the lower the probability of the mutation is, the more alarmed the user should be when it is satisfied.

1. **Simple analysis**. For each mutation φ' that is satisfied in the system, potentially pointing to real vacuity, we find $Pr(\varphi')$ (or an estimation of it). We report to the user only mutations whose probability is below some threshold, given by the user, or we order the mutations according to the probability of their satisfaction, with mutations of low probability being first.
2. **Drop analysis.** For each specification φ and mutation φ' for it that is satisfied in the system, we find $Pr(\varphi) - Pr(\varphi')$. Note that the mutation is always stronger than the original specification, thus $Pr(\varphi) \geq Pr(\varphi')$ and the drop $Pr(\varphi) - Pr(\varphi')$ is greater or equal to 0. We proceed as in the simple analysis, with respect to this drop. Note that since φ' is obtained from φ by replacing some subformula by $\bot$, a lot of the work done in calculating $Pr(\varphi)$ can be reused when calculating $Pr(\varphi')$.
3. **Invariants.** Recall that the probability of an invariant to hold in a random computation is equal to 0 or 1. The mutations of invariants are invariants too. Thus, for invariants, which are the vast majority of specifications written in practice, our framework is two-valued and it does not use the full range of values between 0 and 1. We suggest a refinement to our ranking procedure that leads to more accurate results. In the refined procedure, we perform a simple or drop analysis with respect to the invariant itself; that is, the specification without the outermost G. Note that the invariant itself need not have probability 0 or 1.

Below we demonstrate the three approaches and argue that the ranking they lead to correspond to our intuition of "interesting vacuities".

Example 1. Consider the two specifications $\varphi_1 = G(a \rightarrow Fb)$ and $\varphi_2 = G(a \rightarrow Xc)$. For each specification, we examine two mutations, as detailed below.

$$\varphi_1' = \varphi_1[a \leftarrow \mathit{true}] = GFb, \quad \varphi_2' = \varphi_2[a \leftarrow \mathit{true}] = GXc,$$
$$\varphi_1'' = \varphi_1[b \leftarrow \mathit{false}] = G\neg a, \quad \varphi_2'' = \varphi_2[c \leftarrow \mathit{false}] = G\neg a.$$

Note that $\varphi_1'' = \varphi_2''$. We start with the simple analysis. Calculating the probability of each mutation, we get the following:

$$Pr(\varphi_1') = 1,\ Pr(\varphi_1'') = 0,\ Pr(\varphi_2') = 0,\ \text{and}\ \ Pr(\varphi_2'') = 0.$$

Hence, if we choose to return only mutations whose probability is below some threshold, we return only $G\neg a$ and GXc.

The simple analysis suggests that the mutation GFb is not of much interest, which meets our intuition, but offers no order between GXc and $G\neg a$. We proceed to drop analysis. We first calculate $Pr(\varphi_1) = 1$ and $Pr(\varphi_2) = 0$. Accordingly,[3]

$$\begin{array}{ll} Pr(\varphi_1) - Pr(\varphi_1') = 1 - 1 = 0, & Pr(\varphi_2) - Pr(\varphi_2') = 0 - 0 = 0, \\ Pr(\varphi_1) - Pr(\varphi_1'') = 1 - 0 = 1, & Pr(\varphi_2) - Pr(\varphi_2'') = 0 - 0 = 0. \end{array}$$

It follows that the biggest drop is in the mutation $G\neg a$ of φ_1, so we order it above GXc (and, following the simple analysis, above GFb). Note that while $\varphi_1'' = \varphi_2''$, only in φ_1'' we exhibit a drop.

We proceed to examine the behavior of the invariants. Note that $\varphi_1 = G\theta_1$ and $\varphi_2 = G\theta_2$, for $\theta_1 = (a \rightarrow Fb)$ and $\theta_2 = (a \rightarrow Xc)$. Applying the corresponding mutations of the invariants, we have the following.

$$\begin{array}{ll} \theta_1' = \theta_1[a \leftarrow \mathit{true}] = Fb, & \theta_2' = \theta_2[a \leftarrow \mathit{true}] = Xc, \\ \theta_1'' = \theta_1[b \leftarrow \mathit{false}] = \neg a, & \theta_2'' = \theta_2[c \leftarrow \mathit{false}] = \neg a. \end{array}$$

Calculating probabilities, we obtain the following.

$$\begin{array}{ll} Pr(\theta_1') = 1, & Pr(\theta_2') = \frac{1}{2}, \\ Pr(\theta_1'') = \frac{1}{2}, & Pr(\theta_2'') = \frac{1}{2}. \end{array}$$

This again shows, in a more refined way, that the mutation GFb is the least interesting one. Combining the invariants with the drop analysis, we take into account the fact that $Pr(\theta_1) = 1$ and $Pr(\theta_2) = \frac{3}{4}$. Accordingly,

$$\begin{array}{ll} Pr(\theta_1) - Pr(\theta_1') = 1 - 1 = 0, & Pr(\theta_2) - Pr(\theta_2') = \frac{3}{4} - \frac{1}{2} = \frac{1}{4}, \\ Pr(\theta_1) - Pr(\theta_1'') = 1 - \frac{1}{2} = \frac{1}{2}, & Pr(\theta_2) - Pr(\theta_2'') = \frac{3}{4} - \frac{1}{2} = \frac{1}{4}. \end{array}$$

This strengthens the order above, with no drop for φ_1', a drop of $\frac{1}{2}$ for φ_1'', and drops of $\frac{1}{4}$ for both φ_2' and φ_2''.

We note that throughout the above example, working with the estimated probabilities would lead to exactly the same results.

Example 2. Consider the specification $\varphi_3 = G(a \rightarrow X(c_1 \vee c_2 \vee \cdots \vee c_m))$. We consider two types of mutations.

- $\varphi_3' = \varphi_3[a \leftarrow \mathit{true}] = GX(c_1 \vee \cdots \vee c_m)$, and
- $\varphi_3'' = \varphi_3[c_1 \leftarrow \mathit{false}] = G(a \rightarrow X(c_2 \vee \cdots \vee c_m))$, and similarly for the other c_i's, with $1 < i \leq m$.

Since the probability of φ_3 is 0, the probability of the mutations is 0 too, suggesting that the simple and drop analysis are not informative. We examine the invariants themselves. Let $\theta_3 = a \rightarrow X(c_1 \vee \cdots \vee c_m)$, and consider the mutations

- $\theta_3' = \theta_3[a \leftarrow \mathit{true}] = X(c_1 \vee \cdots \vee c_m)$, and
- $\theta_3'' = \theta_3[c_1 \leftarrow \mathit{false}] = a \rightarrow X(c_2 \vee \cdots \vee c_m)$.

[3] Note that if the probability of a specification is 0, the probability of its mutations is 0 too, so there is no need to calculate it; the drop is going to be 0 as well.

We have that $\Pr(\theta_3) = \frac{2^{m+1}-1}{2^{m+1}}$, whereas $\Pr(\theta'_3) = \Pr(\theta''_3) = \frac{2^m-1}{2^m}$. Thus, the mutation strictly decreases the probability of the invariant to hold in a position. Moreover, the bigger m is, the smaller is the drop. In particular, when $m > 1$, the mutations for φ_2 from Example 1 should be ranked above those for φ_3.

Example 2 demonstrates that the probability-based criteria are useful also for the analysis of responsibility and blame studied in the context of coverage in [8].

Example 3. Consider the specification $\varphi_4 = GFb_1 \vee GFb_2 \vee ... \vee GFb_m$. For each mutation of the form $\varphi'_4 = \varphi_4[b_i \leftarrow \mathit{false}]$ with $1 \leq i \leq m$, we have that $Pr(\varphi'_4) = Pr(\varphi_4) = 1$. Also, while φ_4 is not an invariant, it is equivalent to $G\theta_4$, for $\theta_4 = Fb_1 \vee Fb_2 \vee ... \vee Fb_m$. Here too, for each mutation of the form $\theta'_4 = \theta_4[b_i \leftarrow \mathit{false}]$, we have that $Pr(\theta'_4) = Pr(\theta_4) = 1$. Thus, all approaches lead to the conclusion that mutations of this form are not of big interest.

Example 4. Consider the specification $\varphi_5 = G(a \rightarrow (bRc))$. The operator R (release) is dual to the Until operator: a computation satisfies bRc if c holds until $b \wedge c$ holds, where possibly $b \wedge c$ never holds, in which case c holds forever. Let $\theta_5 = a \rightarrow (bRc)$.

We first note that $Pr(\varphi_5) = 0$. To see this, observe that $Pr(\theta_5) = \frac{5}{6}$. In fact, in this case also $Est(\theta_5) = \frac{5}{6}$. Indeed, $Est(a \rightarrow (bRc)) = Est(\neg(a \wedge \neg(bRc))) = 1 - Est(a) \cdot Est((\neg b)U(\neg c)) = 1 - \frac{1}{2} \cdot \frac{\frac{2}{3}}{1-(1-\frac{1}{2})\frac{1}{2}} = 1 - \frac{1}{2} \cdot \frac{2}{3} = \frac{5}{6}$. It follows that probabilities of all mutations φ'_5 of φ_5 are also 0, and we proceed to analyze the invariants themselves. We consider the following mutations.

- $\theta'_5[a \leftarrow \mathit{true}] = bRc$, so $Pr(\theta'_5) = \frac{2}{3}$,
- $\theta''_5[b \leftarrow \mathit{false}] = a \rightarrow Gc$, so $Pr(\theta''_5) = \frac{1}{2}$,
- $\theta'''_5[c \leftarrow \mathit{false}] = \neg a$, so $Pr(\theta'''_5) = \frac{1}{2}$.

This suggests that the last two vacuities are more interesting than the first one, which meets our experience.

Example 5. In this example we demonstrate the extension of our framework to the definition of vacuity studied in [1], which handles also subformulas ψ with multiple occurrences, possibly of different polarity. As suggested in [1], such mutations replace ψ by a universally quantified proposition. We also demonstrate how errors in the estimated algorithm may actually be helpful.

Consider the specification $\varphi_6 = \mathit{idle}U(\neg \mathit{idle} \vee \mathit{err})$. The specification states that eventually we get to a position that is not idle or in which the error signal is active, and until then, all positions are idle. We consider two mutations.

- $\varphi'_6 = \varphi_6[\mathit{err} \leftarrow \mathit{false}] = \mathit{idle}U(\neg \mathit{idle})$,
- $\varphi''_6 = \varphi_6[\mathit{idle} \leftarrow \bot] = \forall x.(xU((\neg x) \vee \mathit{err}))$.

The probability of φ_6 as well as the probability of both mutations is 1. Indeed, φ'_6 is equivalent to $F(\neg \mathit{idle})$ and φ''_6 is equivalent to $F(\mathit{err})$. This suggests that vacuity detection in this case is not too interesting. This meets our intuition, as the specification is quite weak. Nevertheless, if we examine the estimated analysis of the mutation, we reveal something interesting: while $Est(\varphi_6) = \frac{\frac{3}{4}}{1-(1-\frac{3}{4})\frac{1}{2}} = \frac{6}{7}$, we have that $Est(\varphi'_2) = \frac{\frac{1}{2}}{1-(1-\frac{1}{2})\frac{1}{2}} = \frac{2}{3}$. Thus, the estimated analysis, which ignores

the relation between *idle* and $\neg$*idle* shows a drop in the probability. Reporting this to the user is helpful, as φ is *inherently vacuous* [14]: it is equivalent to its mutation $F(\neg idle \vee err)$, obtained by replacing the first occurrence of *idle* by *true*.

5 A Simpler PSPACE Algorithm

Recall that the PSPACE algorithm for calculating $Pr(\varphi)$ involves the probabilistic model-checking algorithm of [12], which is complicated. In this section we describe a much simpler algorithm for the case we only want to find whether $Pr(\varphi) = 0$ or $Pr(\varphi) = 1$. By the lower-bound proof of Theorem 1, this problem is PSPACE-hard too. As we show, however, it can be solved by a simple analysis of the connected components of a nondeterministic automaton for φ. We first need some definitions and notations.

5.1 Graphs and Automata

A set of vertices $C \subseteq V$ is *strongly connected* in G if there is a path from each vertex in C to every other vertex in C. The *strongly connected components* (SCC) of a graph G are its maximal strongly connected subsets of vertices. Note that every graph has a single partition into SCCs. An SCC C of a graph G is *Ergodic* iff for all $\langle u, v\rangle \in E$, if $u \in C$ then $v \in C$. That is, an SCC is Ergodic if no edge leaves it. Note that in a finite graph there must exist an Ergodic SCC. We say that a path π *reaches an Ergodic SCC* C, if there exists i such that for all $j \geq i$ we have that $\pi_j \in C$.

Consider a Markov chain $M = \langle V, p\rangle$. For an event $S \subseteq V^\omega$ and a vertex v, we denote by $Pr(S, v)$ the probability that a path that starts at v belongs to S. The probability of reaching each Ergodic SCC in M can be calculated. Given M and an initial state v, it is possible to calculate, for each vertex u, the probability $Pr(S_u, v)$, where S_u is the set of paths that reach u; that is, $Pr(S_u, v)$ is the probability in which a random path starting at v reaches u. The calculation of $Pr(S_u, v)$ is done for all vertices u together, by solving a system of linear equations describing the probability of reaching a vertex in terms of reaching its predecessors. Using $Pr(S_u, v)$, we can calculate, for each Ergodic SCC C, the probability $Pr(S_C, v)$, where $S_C = \bigcup_{u \in C} S_u$; that is, $Pr(S_C, v)$ is the probability in which a random path that starts at v reaches the Ergodic SCC C [18].

Lemma 6. [18] *Let M be a finite Markov chain, and E the set of its Ergodic SCCs.*

1. *For every vertex v, we have that $\sum_{C \in \mathsf{E}} Pr(S_C, v) = 1$. That is, an infinite random path reaches some Ergodic SCC in probability* 1.
2. *Let C be an Ergodic SCC and v and u be vertices in C. Then, $Pr(S_u, v) = 1$. That is, once a random path reaches an Ergodic SCC C, it visits all the vertices in C with probability* 1.
3. *For every vertex v and an Ergodic SCC C reachable from v, we have that $Pr(S_C, v) > 0$. That is, the probability of a random infinite path to reach a reachable Ergodic SCC is greater than 0.*

A *nondeterministic finite automaton* is a tuple $\mathcal{A} = \langle \Sigma, Q, Q_0, \delta, \alpha\rangle$, where Σ is a finite non-empty alphabet, Q is a finite non-empty set of *states*, $Q_0 \subseteq Q$ is a

nonempty set of *initial states*, $\delta : Q \times \Sigma \to (2^Q \setminus \emptyset)$ is a *transition function*, and α is an *acceptance condition*.

The automaton $\mathcal{A}$ is *deterministic* if $|Q_0| = 1$ and $|\delta(q, \sigma)| = 1$ for all states $q \in Q$ and symbols $\sigma \in \Sigma$. Note that $\mathcal{A}$ is *complete*, in the sense that Q_0 is nonempty and for each $q \in Q$ and $\sigma \in \Sigma$, the set $\delta(q, \sigma)$ contains at least one successor state.

A run of $\mathcal{A}$ on an infinite input word $w = \sigma_0 \cdot \sigma_1 \cdot \sigma_2 \cdots \in \Sigma^\omega$, is an infinite sequence of states. The acceptance condition α determines which runs are accepting. A set S of states satisfies a *Büchi* acceptance condition $\alpha \subseteq Q$ if $S \cap \alpha \neq \emptyset$. A richer acceptance condition is *parity*. For a run r, let $inf(r) = \{q : q_i = q \text{ for infinitely many } i\text{'s }\}$. That is, $inf(r)$ is the set of states that r visits infinitely often. Then, r is accepting iff $inf(r)$ satisfies α.

A word w is accepted by an automaton $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ on w. The language of $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set of words that $\mathcal{A}$ accepts. We use NBW and DBW to abbreviate nondeterministic and deterministic Büchi automata, respectively. We use DPW to abbreviate deterministic parity automata.

Theorem 7. *[22, 24] For every LTL formula φ of length n there exists an NBW $\mathcal{A}_\varphi$ and a DPW $\mathcal{D}_\varphi$ such that $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\mathcal{D}_\varphi) = \mathcal{L}(\varphi)$, and the sizes of $\mathcal{A}_\varphi$ and $\mathcal{D}_\varphi$ are at most $2^{O(n)}$ and $2^{2^{O(n)}}$, respectively.*

A deterministic automaton $\mathcal{A}$ induces a Markov chain $M_\mathcal{A} = \langle V, p \rangle$, where $V = Q$ and $p(q, q') = \frac{|\{\sigma : \delta(q,\sigma) = q'\}|}{|\Sigma|}$.

5.2 Calculating $Pr(\varphi)$

Before we describe the simpler algorithm, let us first describe an algorithm that is based on the deterministic automaton for φ. The algorithm calculates $Pr(\varphi)$ and its complexity is higher than the one presented in Section 3. We still present it, as it explains nicely the setting and the challenges in working with the nondeterministic automaton.

Theorem 8. *Consider a deterministic automaton $\mathcal{A}$. Let q_0 be the initial state of $\mathcal{A}$ and E_{acc} the set of the Ergodic accepting SCCs in $M_\mathcal{A}$. Then, $Pr(\mathcal{L}(\mathcal{A})) = \sum_{C \in \mathsf{E}_{acc}} Pr(S_C, q_0)$.*

Theorem 8 suggests an algorithm for calculating $Pr(\varphi)$: given an LTL formula φ, build the DPW $\mathcal{D}_\varphi$ and calculate the probability of reaching each of its accepting Ergodic SCC. The probability $Pr(\varphi)$ is then the sum of the calculated probabilities. Note, however, that the size of $\mathcal{D}_\varphi$ may be doubly exponential in the length of φ, thus the algorithm is less efficient than the one that uses the model-checking procedure of [12]. In the next subsection we discuss cases where an analysis can be performed on the NBW for φ.

5.3 Deciding 0/1 Probability

Many LTL formulas have the *0/1 property* [13, 15]: their probability to hold on a random path is either 0 or 1. In particular, as discussed in Section 3, invariants have this property. We now show that if we only want to find whether $Pr(\varphi) = 0$ or $Pr(\varphi) = 1$, we can reason about the connected components of the NBW for φ. We first need to adjust the notion of Ergodic SCC to the nondeterministic setting.

Consider an NBW $\mathcal{A}$. For simplicity, we assume that all the states in $\mathcal{A}$ are reachable. Let C be a connected component in $\mathcal{A}$. For a state $q \in C$, we say that q is an *escape state* from C if there is a letter $\sigma \in \Sigma$ such that all the σ-transitions from q leave C. Formally, $\delta(q, \sigma) \cap C = \emptyset$. We say that a connected component C is *terminal* if C contains no escape states. That is, for all states $q \in C$ and letters $\sigma \in \Sigma$, there is a σ-transition from q that stays in C. Formally. $\delta(q, \sigma) \cap C \neq \emptyset$.

Theorem 9. *An NBW $\mathcal{A}$ has a terminal accepting component iff $Pr(\mathcal{L}(\mathcal{A})) > 0$.*

In order to use Theorem 9 for checking whether $Pr(\mathcal{L}(\mathcal{A})) > 0$, we still have to describe how we detect a terminal connected component in NLOGSPACE, which would imply a PSPACE algorithm in terms of the LTL formula that induces $\mathcal{A}$. Since we can check whether a state is an escape state by examining the transitions from it, the procedure is similar to the NLOGSPACE algorithm for checking the non-emptiness of an NBW, except that when we check for the reachability of a state in α to itself, we go only through states that are not escape states.

Finally, since LTL is closed under complementation, we can use Theorem 9 in order to decide both whether $Pr(\varphi) = 0$, simply by checking that it is not the case $Pr(\mathcal{L}(\mathcal{A}_\psi)) > 0$, and whether $Pr(\varphi) = 1$, by checking that it is not the case $Pr(\mathcal{L}(\mathcal{A}_{\neg\psi})) > 0$.

6 Future Work

We find the probability of LTL formulas to be an interesting question on its own, beyond the application to vacuity ranking. The definition of $Pr(\varphi)$ in the paper, assumes that for each atomic proposition p and for each state in the computation, the probability of p to hold in the state is $\frac{1}{2}$. This corresponds to computations in an infinite-state system and is the standard approach taken in studies of 0/1-laws. Alternatively, one can also study the probability of formulas to hold in computations of finite-state systems. Formally, for integers $k \geq 0$ and $l \geq 1$, let $Pr_{k,l}(\varphi)$ denote the probability that φ holds in a random lasso-shape computation with a prefix of length k and a loop of length l. Here too, the probability of each atomic proposition to hold in a state is $\frac{1}{2}$, yet we have only $k + l$ states to fix an assignment to. So, for example, while $Pr(Gp) = 0$, we have that $Pr_{0,1}(Gp) = \frac{1}{2}$ and $Pr_{0,2}(Gp) = \frac{1}{4}$.

There are several interesting issues in the finite-state approach. First, it may seem obvious that the bigger k and l are, the closer $Pr_{k,l}(\varphi)$ gets to $Pr(\varphi)$. This is, however, not so simple. For example, issues like cycles in φ can cause $Pr_{k,l}(\varphi)$ to be non-monotonic. For example, when φ requires p to hold in exactly all even positions, then $Pr_{0,1}(\varphi) = 0, Pr_{0,2}(\varphi) = \frac{1}{4}, Pr_{0,3}(\varphi) = 0, Pr_{0,1}(\varphi) = \frac{1}{16}$, and so on. It may also seem that, after cleaning the cycle-based issue (for example by restricting attention to formulas without Xs), one can characterize safety and liveness properties by means of the asymptotic behavior of $Pr_{k,l}(\varphi)$. For example, clearly $Pr_{k,l}(Gp)$ goes to 0 as k and l increase, whereas $Pr_{k,l}(Fp)$ goes to 1. Here too, however, the picture is not clean. For example, FGp is a liveness formula, but $Pr_{k,l}(FGp)$ decreases as k and l increase. Finding a characterization of properties that is based on the analysis of $Pr_{k,l}$ is a very interesting question, and we are currently studying it. It should also be noted that, unlike $Pr(\varphi)$, it is possible to estimate $Pr_{k,l}(\varphi)$ using model checking of φ in (sufficiently many) randomly generated lassos.

Acknowledgment. We thank Moshe Vardi for helpful discussions, and reviewers of an earlier version of this paper for helpful comments and suggestions.

References

1. R. Armoni, L. Fix, A. Flaisher, O. Grumberg, N. Piterman, A. Tiemeyer, and M.Y. Vardi. Enhanced vacuity detection for linear temporal logic. In *Proc. 15th CAV*, pages 368-380. Springer, 2003.
2. I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. Efficient detection of vacuity in ACTL formulas. *Formal Methods in System Design*, 18(2):141–162, 2001.
3. S. Ben-David, D. Fisman, and S. Ruah. Temporal antecedent failure: Refining vacuity. In *Proc. 18th CONCUR*, pages 492–506, 2007.
4. D. Bustan, A. Flaisher, O. Grumberg, O. Kupferman, and M.Y. Vardi. Regular vacuity. In *Proc. 13th CHARME*, *LNCS* 3725, pages 191–206, 2005.
5. M. Chechik, M. Gheorghiu, and A. Gurfinkel. Finding environment guarantees. In *Proc. 10th FASE*, *LNCS* 4422, pages 352–367, 2007.
6. H. Chockler. Experience with vacuity checking in IBM Haifa. Personal communication.
7. H. Chockler, A. Gurfinkel, and O. Strichman. Beyond vacuity: Towards the strongest passing formula. In *FMCAD*, pages 1–8, 2008.
8. H. Chockler and J.Y. Halpern. Responsibility and blame: a structural-model approach. In *Proc. 19th IJCAI*, pages 147–153, 2003.
9. H. Chockler and O. Strichman. Easier and more informative vacuity checks. In *Proc. 5th MEMOCODE*, pages 189–198, 2007.
10. E.M. Clarke, O. Grumberg, and D. Long. Verification tools for finite-state concurrent systems. In *Decade of Concurrency – Reflections and Perspectives* , *LNCS* 803 , pages 124–175, 1993.
11. E.M. Clarke, O. Grumberg, K.L. McMillan, and X. Zhao. Efficient generation of counterexamples and witnesses in symbolic model checking. In *Proc. 32st DAC*, pages 427–432. IEEE Computer Society, 1995.
12. C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42:857–907, 1995.
13. R. Fagin. Probabilities in finite models. *Journal of Symb. Logic*, 41(1):50–58, 1976.
14. D. Fisman, O. Kupferman, S. Seinvald, and M.Y. Vardi. A framework for inherent vacuity. In *HVC08*, *LNCS* 5394, pages 7–22, 2008.
15. Y.V. Glebskii, D.I. Kogan, M.I. Liogonkii and V.A. Talanov. Range and degree of realizability of formulas in the restricted predicate calc. *Kibernetika*, 2:17–28, 1969.
16. A. Gurfinkel and M. Chechik. Extending extended vacuity. In *Proc. 5th FMCAD*, *LNCS* 3312, pages 306–321, 2004.
17. A. Gurfinkel and M. Chechik. How vacuous is vacuous. In *Proc. 10th TACAS*, *LNCS* 2988, pages 451–466, 2004.
18. J.G. Kemeny and J.L. Snell. *Finite Markov Chains.* Van Nostrad, 1960.
19. O. Kupferman. Sanity checks in formal verification. In *Proc. 17th CONCUR*, *LNCS* 4137, pages 37–51, 2006.
20. O. Kupferman, W. Li, and S. A. Seshia. A theory of mutations with applications to vacuity, coverage, and fault tolerance. In *FMCAD*, pages 1–9, 2008.
21. O. Kupferman and M.Y. Vardi. Vacuity detection in temporal model checking. *Software Tools for Technology Transfer*, 4(2):224–233, 2003.
22. N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. *Logical Methods in Computer Science*, 3(3):5, 2007.
23. A. Pnueli. The temporal logic of programs. In *Proc. 18th IEEE Symp. on Foundations of Computer Science*, pages 46–57, 1977.
24. M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.

A Proofs

Theorem 1. *The problem of finding the probability of LTL formulas is PSPACE-complete.*

Proof. Consider an LTL formula φ. The problem of finding $Pr(\varphi)$ can be reduced to LTL probabilistic model checking. In this problem, we are given a probabilistic labeled structure $\mathcal{S}$ and an LTL formula φ, and we find the probability of paths that satisfy φ in $\mathcal{S}$. By definition, model checking φ in $\mathcal{U}_{2^{AP}}$ amounts to finding $Pr(\varphi)$. In [12], Courcoubetis and Yannakakis describe an algorithm for solving LTL probabilistic model checking. The algorithm requires space that is polynomial in the formula and polylogarithmic in the structure, implying a PSPACE complexity for our problem.

For the lower bound, we prove that already the problem of deciding whether the probability of a formula is 0 or 1 is PSPACE-hard. Recall the generic reduction used in the PSPACE-hardness proof for the validity problem of LTL. Given a polynomial space Turing machine and an input x for it, the reduction constructs an LTL formula $\varphi_{T,x}$ that is valid iff T rejects x. The formula $\varphi_{T,x}$ is satisfied in a computation iff the computation does not encode a legal computation of T on x or it encodes a legal yet rejecting computation of T on x. The reduction can be defined so that the encoding of computations of T on x concerns only the prefix of the computation in which a final configuration is reached. Thus, after T reaches a final configuration, all suffixes are considered as legal encodings. We claim that $Pr(\varphi_{T,x}) \in \{0,1\}$ iff T rejects x. First, if T rejects x, then $\varphi_{T,x}$ is valid, so $Pr(\varphi_{T,x}) = 1$. Also, if T accepts x, then the event $\{w : w \text{ encodes a legal accepting computation of } T \text{ on } x\}$ has a positive probability – it contains all the words that have the accepting computation of T on x as their prefix. □

Theorem 2. *Formulas of the form $G\varphi$ satisfy the 0/1-law. On the other hand, formulas of the form $r \mapsto \varphi$ need not satisfy the 0/1-law even when $\mathcal{L}(r)$ is infinite.*

Proof. We start with invariants. Consider a formula $G\varphi$, and let π be a random computation. If $Pr(\varphi) = 1$, then for all suffixes π^i of π, the probability of φ to hold in π^i is 1, thus so is the probability of $G\varphi$ to hold in π. Hence, $Pr(\varphi) = 1$ implies that $Pr(G\varphi) = 1$. Now, if $Pr(\varphi) < 1$, then with probability 1 we have a suffix π^i such that the probability of φ to hold in π^i is strictly less than 1, making the probability of $G\varphi$ to hold in π 0. Hence, $Pr(\varphi) < 1$ implies that $Pr(G\varphi) = 0$.

For triggers, consider the specification $\varphi = req[*]; ack \mapsto Xready$, stating that whenever we have a computation where req is active all the way (from the initial state), followed by ack, then $ready$ must follow one cycle after ack. We show that $0 < Pr(\varphi) < 1$. Note that if both req and ack do not hold in the initial state, then φ holds (since the prefix is false). The probability of this is $\frac{1}{4}$, thus $0 < Pr(\varphi)$. Note further that if req holds on the first state, ack on the second, and $ready$ does not hold on the third state, then φ fails to hold on the computation path. The probability of this is $\frac{1}{8}$; thus $Pr(\varphi) < 1$. □

Theorem 8. *Consider a deterministic automaton $\mathcal{A}$. Let q_0 be the initial state of $\mathcal{A}$ and E_{acc} the set of its Ergodic accepting SCCs. Then, $Pr(\mathcal{L}(\mathcal{A})) = \sum_{C \in \mathsf{E}_{acc}} Pr(S_C, q_0)$.*

Proof. Let w be a random word and r the run of $\mathcal{A}$ on it. By Lemma 6, once a run r reaches an Ergodic SCC C, it visits all its states infinitely often with probability 1. Accordingly, with probability 1 we have that $inf(r) = C$, thus r is accepting iff C is accepting. Since the probability of r reaching an *accepting* Ergodic SCC is the sum of probabilities $p(S_C, q_0)$ for accepting Ergodic SCC C, we are done. □

Theorem 9. *An NBW $\mathcal{A}$ has a terminal accepting component iff $Pr(\mathcal{L}(\mathcal{A})) > 0$.*

Proof. Assume first that $\mathcal{A}$ has a terminal accepting component C. By removing transitions that leave C and still keeping the transition function complete (since C has no escape states, this is possible), we can obtain a DBW $\mathcal{A}'$ embodied in $\mathcal{A}$ in which C is an accepting Ergodic SCC. By Theorem 8, we have that $Pr(\mathcal{L}(\mathcal{A}')) > 0$. Since $\mathcal{L}(\mathcal{A}') \subseteq \mathcal{L}(\mathcal{A}$, it follows that $Pr(\mathcal{L}(\mathcal{A})) > 0$, and we are done.

For the other direction, we prove that if $\mathcal{A}$ has no terminal accepting component, then $Pr(\mathcal{L}(\mathcal{A})) = 0$. We first claim that no matter how we resolve the nondeterministic choices in $\mathcal{A}$, each run reaches a terminal component with probability 1. Indeed, as in the proof of Lemma 6, as long as a run is not in a terminal component, it would reach an escape state and read the letter with which the corresponding component is left with probability 1. Assume that $\mathcal{A}$ has no terminal accepting component. Then, by the above, no matter how we resolve the nondeterministic choices in $\mathcal{A}$, each run reaches a rejecting terminal component with probability 1. Hence $Pr(\mathcal{L}(\mathcal{A})) = 0$, and we are done. □