On Relative and Probabilistic Finite Counterability

Orna Kupferman and Gal Vardi

School of Engineering and Computer Science, Hebrew University, Israel

Abstract

A *counterexample* to the satisfaction of a linear property ψ in a system $\mathcal{S}$ is an infinite computation of $\mathcal{S}$ that violates ψ. When ψ is a safety property, a counterexample to its satisfaction need not be infinite. Rather, it is a *bad-prefix* for ψ: a finite word all whose extensions violate ψ. The existence of finite counterexamples is very helpful in practice. Liveness properties do not have bad-prefixes and thus do not have finite counterexamples.

We extend the notion of finite counterexamples to non-safety properties. We study *counterable languages* – ones that have at least one bad-prefix. Thus, a language is counterable iff it is not liveness. Three natural problems arise: (1) Given a language, decide whether it is counterable, (2) study the length of minimal bad-prefixes for counterable languages, and (3) develop algorithms for detecting bad-prefixes for counterable languages. We solve the problems for languages given by means of LTL formulas or nondeterministic Büchi automata. In particular, our EXPSPACE-completeness proof for the problem of deciding whether a given LTL formula is counterable, and hence also for deciding liveness, settles a long-standing open problem.

In addition, we make finite counterexamples more relevant and helpful by introducing two variants of the traditional definition of bad-prefixes. The first adds a *probabilistic* component to the definition. There, a prefix is bad if almost all its extensions violate the property. The second makes it *relative* to the system. There, a prefix is bad if all its extensions in the system violate the property. We also study the combination of the probabilistic and relative variants. Our framework suggests new variants also for safety and liveness languages. We solve the above three problems for the different variants. Interestingly, the probabilistic variant not only increases the chances to return finite counterexamples, but also makes the solution of the three problems exponentially easier.

1998 ACM Subject Classification F.4.3 Formal Languages, B.8.2 Performance Analysis and Design Aids, F.1.1 Models of Computation

Keywords and phrases Model Checking, Counterexamples, Safety, Liveness, Probability

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.175

1 Introduction

In model checking, we verify that a system meets a desired property by checking that a mathematical model of the system meets a formal specification that describes the property. *Safety* and *liveness* [2] are two classes of system properties. Essentially, a safety property states that something "bad" never happens and a liveness property states that something "good" eventually happens. Formally, consider a language L of infinite words over an alphabet Σ. A finite word $x \in \Sigma^*$ is a *bad-prefix* for L if for all infinite words $y \in \Sigma^\omega$, the concatenation $x \cdot y$ is not in L. Thus, a bad-prefix for L is a finite word that cannot be extended to a word in L. A language L is *safety* if every word not in L has a bad-prefix, and is *liveness* if it has no bad-prefixes. Thus, every word in Σ^* can be extended to a word in L.

The classes of safety and liveness properties have been extensively studied. From a theoretical point of view, their importance stems from their topological characteristics. Consider the natural topology on Σ^ω, where similarity between words corresponds to the

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 175–192

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

length of the prefix they share. Formally, the distance between w and w' is 2^{-i}, where $i \geq 0$ is the position of the first letter in which w and w' differ. In this topology, safety and liveness properties are exactly the closed and dense sets, respectively [2]. This, for example, implies that every linear property can be expressed as a conjunction of a safety and a liveness property [2, 3].

From a practical point of view, reasoning about safety and liveness properties require different methodologies, and the distinction between them pinpoints different challenges of formal methods. In liveness, one has to demonstrate progress towards fulfilling eventualities. Thus, liveness is the concept behind fairness [15], and behind the need for rich ω-regular acceptance conditions [37], progress measures [30, 27], and many more. On the other hand, in safety one can reason about finite computations of the system [35]. The latter has significant advantages in symbolic methods [23], bounded model checking [7], run-time verification [17], synthesis [25], and many more.

An important advantage of model-checking tools is their ability to accompany a negative answer to the correctness query by a *counterexample* to the satisfaction of the specification in the system. Thus, together with a negative answer, the model checker returns some erroneous execution of the system. These counterexamples are not only essential in detecting subtle errors in complex systems [9], but also in improving the modeling of systems. For example, in CEGAR, counterexamples are used in order to guide the refinement of abstract models [8]. In the general case, the erroneous execution of the system is infinite. It is known, however, that for linear temporal logic (LTL) properties, there is always a *lasso-shaped* counterexample – one of the form uv^ω, for finite computations u and v. Clearly, the simpler the counterexample is, the more helpful it is for the user, and indeed there have been efforts for designing algorithms that return short counterexamples [33, 22]. The analysis of counterexamples makes safety properties even more appealing: rather than a lasso-shaped counterexample, it is possible to return to the user a bad-prefix. This enables the user to find errors as soon as they appear. In addition, the analysis of bad-prefixes is often more helpful, as they point the user not just to one erroneous execution, but rather to a finite execution all whose continuations are erroneous.

We extend the notion of finite counterexamples to non-safety specifications. We also make finite counterexamples more relevant and helpful by introducing two variants of the traditional definition of bad-prefixes. The first adds a *probabilistic* component to the definition. The second makes it *relative* to the system. We also consider the combination of the probabilistic and relative variants. Before we describe our contribution in detail, let us demonstrate the idea with the following example. Consider a system $\mathcal{S}$ and a specification ψ stating that every request is eventually followed by a response. There might be some input sequence that leads $\mathcal{S}$ to an error state in which it stops sending responses. While ψ is not safety, the system $\mathcal{S}$ has a computation with a prefix that is bad with respect to $\mathcal{S}$: all its extensions in $\mathcal{S}$ do not satisfy ψ. Returning this prefix to the user, with its identification as bad with respect to $\mathcal{S}$, is more helpful than returning a lasso-shaped counterexample. Consider now a specification φ stating that the system eventually stops allocating memory. There might be some input sequence that leads $\mathcal{S}$ to a state in which every request is followed by a memory allocation. A computation that reaches this state almost surely violates the specification. Indeed, it is possible that requests eventually stop arriving and the specification would be satisfied, but the probability of this behavior of the input is 0. Thus, the system $\mathcal{S}$ has a computation with a prefix that is bad with respect to $\mathcal{S}$ in a probabilistic sense: almost all of its extensions in $\mathcal{S}$ do not satisfy φ. Again, we want to return this prefix to the user, identified as bad with high probability.

Recall that a language L is liveness if every finite word can be extended to an infinite word in L. Equivalently, L has no bad-prefixes. We say that L is *counterable* if it has a bad-prefix. That is, L is counterable iff it is not liveness. Note that a language, for example $a^* \cdot b \cdot (a + b + c)^\omega$, may be counterable and not safety. When a system does not satisfy a counterable specification ψ, it may contain a bad-prefix for ψ, which we would like to return to the user. Three natural problems arise: (1) Given a language, decide whether it is counterable, (2) study the length of minimal bad-prefixes for counterable languages, and (3) develop algorithms for detecting bad-prefixes for counterable languages.

In fact, the last two problems are open also for safety languages. Deciding whether a given language is safety is known to be PSPACE-complete for languages given by LTL formulas or nondeterministic Büchi word automata (NBWs, for short). For the problem of deciding whether a language is counterable, an EXPSPACE upper-bound for languages given by LTL formulas is not difficult [35], yet the tight complexity is open. This is surprising, as recall that a language is counterable iff it is not liveness, and one could expect the complexity of deciding liveness to be settled by now. As it turns out, the problem was studied in [29], where it is stated to be PSPACE-complete. The proof in [29], however, is not convincing, and indeed efforts to solve the problem have continued, and the problem was declared open in [4] (see also [26]). Our first contribution is an EXPSPACE lower bound, implying that the long-standing open problem of deciding liveness (and hence, also counterability) of a given LTL formula is EXPSPACE-complete. In a recent communication with Diekert, Muscholl, and Walukiewicz, we have learned that they recently came-up with an independent EXPSPACE lower bound, in the context of monitoring or infinite computations [14]. For languages given by means of an NBW, the problem is PSPACE-complete [35, 29]. Thus, interestingly, while in deciding safety the exponential succinctness of LTL with respect to NBWs does not make the problem more complex, in deciding liveness it makes the problem exponentially more complex. This phenomenon is reflected also in the solutions to the problems about the length and the detection of bad-prefixes. We also show that when a language given by an LTL formula is safety, the solutions for the three problems become exponentially easier.

Let us return to our primary interest, of finding finite counterexamples.

Consider a system modelled by a Kripke structure K over a set AP of atomic propositions. Let $\Sigma = 2^{AP}$, and consider an ω-regular language $L \subseteq \Sigma^\omega$. We say that a finite computation $x \in \Sigma^*$ of K is a *K-bad-prefix*, if x cannot be extended to an infinite computation of K that is in L. Formally, for all $y \in \Sigma^\omega$, if $x \cdot y$ is a computation of K, then it is not in L. Once we define K-bad-prefixes, the definitions of safety and counterability are naturally extended to the relative setting: A language L is K-counterable if it has a K-bad-prefix and is K-safety if every computation of K that is not in L has a K-bad-prefix. Using a product of K with an NBW for L, we are able to show that the solutions we suggest for the three problems in the non-relative setting apply also to the relative one, with an additional NLOGSPACE or linear-time dependency in the size of K. We also study K-safety, and the case L is K-safety. We note that relative bad prefixes have already been considered in the literature, with different motivation and results. In [18], where the notion is explicitly defined, as well as in [5, 34], where it is implicit, the goal is to lift the practical advantages of safety to liveness properties, typically by taking the finiteness of the system into an account. In [29], the idea is to relay on fairness conditions known about the system in order to simplify the reasoning about liveness properties, especially in a setting where an abstract model of the system is used.

We continue to the probabilistic view.[1] A random word over Σ is a word in which all letters are drawn from Σ uniformly at random.[2] In particular, when $\Sigma = 2^{AP}$, then the probability of each atomic proposition to hold in each position is $\frac{1}{2}$. Consider a language $L \subseteq \Sigma^\omega$. We say that a finite word $x \in \Sigma^*$ is a *prob-bad-prefix* for L if the probability of an infinite word with prefix x to be in L is 0. Formally, $Pr(\{y \in \Sigma^\omega : x \cdot y \in L\}) = 0$. Then, L is *prob-counterable* if it has a prob-bad-prefix. Now, given a Kripke structure K, we combine the relative and probabilistic views in the expected way: a finite computation $x \in (2^{AP})^*$ of K is a *K-prob-bad-prefix* for L if a computation of K obtained by continuing x with some random walk on K, is almost surely not in L. Thus, a computation of K that starts with x and continues according to some random walk on K is in L with probability 0. We show that this definition is independent of the probabilities of the transitions in the random walk on K. Again, L is *K-prob-counterable* if it has a K-prob-bad-prefix.

We note that a different approach to probabilistic counterexamples is taken in [1]. There, the focus is on reachability properties, namely properties of the form "the probability of reaching a set T of states starting from state s is at most p". Accordingly, a counterexample is a set of paths from s to T, such that the probability of the event of taking some path in the set is greater than p. We, on the other hand, cover all ω-regular languages, and a counterexample is a single finite path – one whose extension result in a counterexample in high probability.

We study the theoretical properties of the probabilistic setting and show that an ω-regular language L is prob-counterable iff the probability of a random word to be in L is less than 1. We also show that ω-regular languages have a "safety-like" behavior in the sense that the probability of a word not to be in L and not to have a prob-bad-prefix is 0. Similar properties hold in the relative setting and suggest that attempts to return to the user prob-bad-prefixes and K-prob-bad-prefixes are likely to succeed.

From a practical point of view, we show that the probabilistic setting not only increases our chances to return finite counterexamples, but also makes the solution of our three basic problems easier: deciding prob-counterability and K-prob-counterability for LTL formulas is exponentially easier than deciding counterability and K-counterability! Moreover, the length of bad-prefixes is exponentially smaller, and finding them is exponentially easier. Our results involve a careful analysis of the product of K with an automaton for L. Now, the product is defined as a Markov chain, and we also need the automaton to be deterministic. Our construction also suggest a simpler proof to the known probabilistic NBW model-checking result of [11]. While the blow-up determinization involves is legitimate in the case L is given by an NBW, it leads to a doubly-exponential blow-up in the case L is given by an LTL formula ψ. We show that in this case, we can avoid the construction of a product Markov chain and, adopting an idea from [11], generate instead a sequence of Markov chains, each obtained from its predecessor by refining the states according to the probability of the innermost temporal subformula of ψ.

It is easy to see that there is a trade-off between the length of a counterexample and its "precision", in the sense that the longer a finite prefix of an erroneous computation is, the larger is the probability in which it is a K-prob-bad-prefix. We allow the user to play with this trade-off and study both the problem in which the user provides, in addition to K and

[1] In [19], the authors study safety and liveness in probabilistic systems. The setting, definitions, and goals are different from ours here, and focus on the safety and liveness fragments of the probabilistic branching-time logic PCTL.

[2] Our definitions and results apply for all distributions in which all letters are drawn with a positive probability.

ψ, also a probability $0 < \gamma < 1$, and gets back a shortest finite computation x of K such that the probability of a computation of K that starts with x to satisfy ψ is less than γ, and the problem in which the user provides a length $m \geq 1$ and gets back a finite computation x of K of length at most m such that the probability of a computation of K that starts with x to satisfy ψ is minimal.

Due to lack of space, detailed proofs can be found in the full version, in the authors' home pages.

2 Preliminaries

2.1 Automata and LTL

A *nondeterministic automaton on infinite words* is a tuple $\mathcal{A} = \langle \Sigma, Q, Q_0, \delta, \alpha \rangle$, where Q is a set of states, $Q_0 \subseteq Q$ is a set of initial states, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, and α is an acceptance condition whose type depends on the class of $\mathcal{A}$. A run of $\mathcal{A}$ on a word $w = \sigma_0 \cdot \sigma_1 \cdots \in \Sigma^\omega$ is a sequence of states $r = q_0, q_1, \ldots$ such that $q_0 \in Q_0$ and $q_{i+1} \in \delta(q_i, \sigma_i)$ for all $i \geq 0$. The run is accepting if it satisfies the condition α. We consider here *Büchi* and *parity* automata. In a Büchi automaton, $\alpha \subseteq Q$ and the run r satisfies α if it visits some state in α infinitely often. Formally, let $inf(r) = \{q : q = q_i \text{ for infinitely many } i\text{'s}\}$ be the set of states that r visits infinitely often. Then, r satisfies α iff $inf(r) \cap \alpha \neq \emptyset$ [6]. In a parity automaton, $\alpha : Q \to \{0, \ldots, k\}$ maps each state to a color in $\{0, \ldots, k\}$. A run r satisfies α if the minimal color that is visited infinitely often is even. Formally, the minimal color c such that $inf(r) \cap \alpha^{-1}(c) \neq \emptyset$ is even. A word $w \in \Sigma^\omega$ is *accepted* by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ on w. The *language* of $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of words that $\mathcal{A}$ accepts. When $|Q_0| = 1$ and $|\delta(q, \sigma)| = 1$ for all $q \in Q$ and $\sigma \in \Sigma$, then $\mathcal{A}$ is *deterministic*. When a state $q \in Q$ is such that no word is accepted from q (equivalently, the language of $\mathcal{A}$ with initial state q is empty), we say that q is *empty*. We use the acronyms NBW, DBW, NPW, and DPW to denote nondeterministic and deterministic Büchi and parity word automata, respectively. We also refer to the standard nondeterministic and deterministic automaton on finite words, abbreviated NFW and DFW, respectively. We define the size of $\mathcal{A}$, denoted $|\mathcal{A}|$, as the size of δ.

An automaton $\mathcal{A}$ induces a graph $G_\mathcal{A} = \langle Q, E \rangle$ where $(q, q') \in E$ iff there is $\sigma \in \Sigma$ such that $q' \in \delta(q, \sigma)$. When we refer to the *strongly connected sets* (SCSs) of $\mathcal{A}$, we refer to the SCSs of this graph. Formally, a set $C \subseteq Q$ of states is an SCS of $\mathcal{A}$ if for all $q, q' \in C$, there is a path from q to q' in $G_\mathcal{A}$. An SCS C is *maximal* if for all sets C' such that $C' \not\subseteq C$, the set $C \cup C'$ is no longer an SCS. A maximal SCS is termed a *strongly connected component* (SCC). An SCC C is *accepting* if a run that visits exactly all the states in C infinitely often satisfies α. For example, when α is a parity condition, then C is accepting if the minimal color c such that $C \cap \alpha^{-1}(c) \neq \emptyset$ is even. An SCC C is *ergodic* iff for all $(q, q') \in E$, if $q \in C$ then $q' \in C$. That is, an SCC is ergodic if no edge leaves it.

The logic *LTL* is a linear temporal logic [32]. Formulas of LTL are constructed from a set AP of atomic proposition using the usual Boolean operators and the temporal operators G ("always"), F ("eventually"), X ("next time"), and U ("until"). The semantics of LTL is defined with respect to infinite computations over AP. We use $w \models \psi$ to indicate that the computation $w \in (2^{AP})^\omega$ satisfies the LTL formula ψ. The language of an LTL formula ψ, denoted $L(\psi)$, is the set of infinite computations that satisfy ψ. For the full syntax and semantics of LTL see [32]. We define the size of an LTL formula ψ, denoted $|\psi|$, to be the number of its Boolean and temporal operators. Given an LTL formula ψ, one can construct an NBW $\mathcal{A}_\psi$ that accepts exactly all the computations that satisfy ψ. The size of $\mathcal{A}_\psi$ is, in the worst case, exponential in $|\psi|$ [37].

We model systems by *Kripke structures*. A Kripke structure is a tuple $K = \langle AP, W, W_0, R, l\rangle$, where W is the set of states, $R \subseteq W \times W$ is a total transition relation (that is, for every $w \in W$, there is at least one state w' such that $R(w, w')$), $W_0 \subseteq W$ is a set of initial states, and $l : W \to 2^{AP}$ maps each state to the set of atomic propositions that hold in it. A *path* in K is a (finite or infinite) sequence $w_0, w_1, \ldots$ of states in W such that $w_0 \in W_0$ and for all $i \geq 0$ we have $R(w_i, w_{i+1})$. A *computation* of K is a (finite or infinite) sequence $l(w_0), l(w_1), \ldots$ of assignments in 2^{AP} for a path $w_0, w_1, \ldots$ in K. We assume that different states of K are labeled differently. That is, for all states $w, w' \in W$ such that $w \neq w'$, we have $l(w) \neq l(w')$. The assumption makes our setting cleaner, as it amount to working with deterministic systems, so all the nondeterminism and probabilistic choices are linked to the specification and the distribution of the inputs, which is our focus. The simplest way to adjust nondeterministic systems to our setting is to add atomic propositions that resolve nondeterminism. The language of K, denoted $L(K)$, is the set of its infinite computations. We say that K satisfies an LTL formula ψ, denoted $K \models \psi$, if all the computations of K satisfy ψ, thus $L(K) \subseteq L(\psi)$. We define the size of a Kripke structure K, denoted $|K|$, as $|W| + |R|$.

For a set AP of atomic propositions, we define the Kripke structure $K_{AP} = \langle AP, 2^{AP}, 2^{AP}, 2^{AP} \times 2^{AP}, l\rangle$, where $l(w) = w$ for all $w \in 2^{AP}$. Thus, K_{AP} is a 2^{AP}-clique satisfying $L(K_{AP}) = (2^{AP})^\omega$.

2.2 Safety, Liveness, and Counterable Languages

Consider an alphabet Σ, a language $L \subseteq \Sigma^\omega$, and a finite word $u \in \Sigma^*$. We say that u is a *prefix for* L if it can be extended to an infinite word in L, thus there is $v \in \Sigma^\omega$ such that $uv \in L$. Then, u is a *bad-prefix for* L if it cannot be extended to an infinite word in L, thus for every $v \in \Sigma^\omega$, we have that $uv \notin L$. Note that if u is a bad-prefix, so are all its finite extensions. We denote by *pref*(L) the set of all prefixes for L.

The following classes of languages have been extensively studied (c.f., [2, 3]). A language $L \subseteq \Sigma^\omega$ is a *safety* language if every word not in L has a bad-prefix. For example, $\{a^\omega\}$ over $\Sigma = \{a, b, c\}$ is safety, as every word not in L has a bad-prefix – one that contains the letter b or c. A language L is a *liveness* language if every finite word can be extended to a word in L. Thus, *pref*$(L) = \Sigma^*$. For example, the language $(a + b + c)^* \cdot a^\omega$ is a liveness language: by concatenating a^ω to every word in Σ^*, we end up with a word in the language. When L is not liveness, namely *pref*$(L) \neq \Sigma^*$, we say that L is *counterable*. Note that while a liveness language has no bad-prefix, a counterable language has at least one bad-prefix. For example, $L = a^* \cdot b \cdot (a + b + c)^\omega$ is a counterable language. Indeed, c is a bad-prefix for L. It is not hard to see that if L is safety and $L \neq \Sigma^\omega$, then L is counterable. The other direction does not hold. For example, L above is not safety, as the word a^ω has no bad-prefix.

We extend the definitions and classes above to specifications given by LTL formulas or by NBWs. For example, an LTL formula ψ is counterable iff $L(\psi)$ is counterable.

3 Probabilistic and Relative Counterability

In this section we introduce and make some observations on two variants to counterability. The first variant adds a probabilistic component to the definitions. The second makes them relative to a Kripke structure. We also consider the combination of the probabilistic and relative variants.

3.1 Probabilistic Counterability

For a finite or countable set X, a *probability distribution* on X is a function $Pr : X \to [0,1]$ assigning a probability to each element in X. Accordingly, $\sum_{x \in X} Pr(x) = 1$. A *finite Markov chain* is a tuple $\mathcal{M} = \langle V, p_{in}, p \rangle$, where V is a finite set of states, $p_{in} : V \to [0,1]$ is a probabilistic distribution on V that describes the probability of a path to start in the state, and $p : V \times V \to [0,1]$ is a function describing a distribution over the transitions. Formally, for $v \in V$, let $p_v : V \to [0,1]$ be such that $p_v(u) = p(v,u)$ for all $u \in V$. For all $v \in V$, the function p_v is a probabilistic distribution on V. The Markov chain $\mathcal{M}$ induces the directed graph $G = \langle V, E \rangle$ in which for all $u, v \in V$, we have that $(u,v) \in E$ iff $p(u,v) > 0$. Thus, G includes transitions that have a positive probability in $\mathcal{M}$. When we talk about the SCCs of $\mathcal{M}$, we refer to these of G.

A *random walk* on $\mathcal{M}$ is an infinite path $v_0, v_1, \ldots$ in G such that v_0 is drawn at random according to p_{in} and the i-th state v_i is drawn at random according to $p_{v_{i-1}}$. More formally, there is a probability space $\langle V^\omega, \mathcal{F}, Pr_{\mathcal{M}} \rangle$ defined on the set V^ω of infinite sequences of states. The family of measurable sets $\mathcal{F}$ is the σ-algebra (also called *Borel field*) generated by $C = \{C(x) : x \in V^*\}$ where $C(x)$ is the set of infinite sequences with prefix x. The measure $Pr_{\mathcal{M}}$ is defined on C (and can be extended uniquely to the rest of $\mathcal{F}$) as follows: $Pr_{\mathcal{M}}[C(x_0, \ldots, x_n)] = p_{in}(x_0) \cdot p(x_0, x_1) \cdot \ldots \cdot p(x_{n-1}, x_n)$. For more background on the construction of this probability space, see, for example, [20]. A random walk on $\mathcal{M}$ from a state $v \in V$ is a random walk on the Markov chain $\mathcal{M}^v = \langle V, p^v_{in}, p \rangle$, where $p^v_{in}(v) = 1$.

▶ **Lemma 1** ([20]). *Consider a Markov chain $M = \langle V, p_{in}, p \rangle$ and a state $v \in V$.*

1. *An infinite random walk on $\mathcal{M}^v$ reaches some ergodic SCC with probability* 1.
2. *Once a random walk on $\mathcal{M}^v$ reaches an ergodic SCC, it visits all its states infinitely often with probability* 1.

A *labeled finite Markov chain* is a tuple $\mathcal{S} = \langle \Sigma, V, p_{in}, p, \tau \rangle$, where Σ is an alphabet, $\mathcal{M} = \langle V, p_{in}, p \rangle$ is a finite Markov chain, and $\tau : V \to \Sigma$ maps each state in V to a letter in Σ. We extend τ to paths in the expected way, thus for $\pi = v_0, v_1, \ldots \in V^\omega$, we define $\tau(\pi) = \tau(v_0), \tau(v_1), \ldots$. A random walk on $\mathcal{S}$ is a random walk on $\mathcal{M}$. The chain $\mathcal{S}$ induces a probability space on Σ^ω, induced from $\mathcal{M}$. That is, for $L \subseteq \Sigma^\omega$, we have $Pr_{\mathcal{S}}[L] = Pr_{\mathcal{M}}[\{\pi \in V^\omega : \tau(\pi) \in L\}]$. It is known that ω-regular languages are measurable in $\mathcal{S}$ (c.f., [36]).

Consider an alphabet Σ. A random word over Σ is a word in which for all indices i, the i-th letter is drawn from Σ uniformly at random. We denote by $Pr[L]$ the probability of a language $L \subseteq \Sigma^\omega$ in this uniform distribution. For a finite word $u \in \Sigma^*$, we denote by $Pr^u[L]$ the probability that a word obtained by concatenating an infinite random word to u is in L. Formally, $Pr^u[L] = Pr[\{v \in \Sigma^\omega : uv \in L\}]$. For example, let $\Sigma = \{a, b, c\}$ and $L = a^* \cdot b \cdot (a + b + c)^\omega$. Then, $Pr[L] = \sum_{i=1}^{\infty} \frac{1}{3^i} = \frac{1}{2}$, $Pr^a[L] = Pr[L] = \frac{1}{2}$, $Pr^{ab}[L] = 1$, and $Pr^c[L] = 0$.

Consider a language $L \subseteq \Sigma^\omega$. We say that a finite word $u \in \Sigma^*$ is a *prob-bad-prefix* for L if $Pr^u[L] = 0$. That is, u is a prob-bad-prefix if an infinite word obtained by continuing u randomly is almost surely not in L. We say that L is *prob-counterable* if it has a prob-bad-prefix. Consider for example the language $L = a \cdot (a+b)^\omega + b^\omega$ over $\Sigma = \{a, b\}$. All the words $u \in b^+$ are not bad-prefixes for L, but are prob-bad-prefixes, as $Pr^u[L] = Pr[b^\omega] = 0$. As another example, consider the LTL formula $\psi = (req \wedge GF\,grant) \vee (\neg req \wedge FG\neg grant)$. The formula ψ is a liveness formula and does not have a bad-prefix. Thus, ψ is not counterable. All finite computations in which a request is not sent in the beginning of the computation are, however, prob-bad-prefixes for ψ, as the probability of satisfying $FG\neg grant$ is 0. Hence, ψ is prob-counterable.

Prob-counterability talks about prefixes after which the probability of being in L is 0. We can relate such prefixes to words that lead to rejecting ergodic SCCs in DPWs that recognize L, giving rise to the following alternative definitions:

▶ **Theorem 2.** *Consider an ω-regular language L. The following are equivalent:*
1. *The language L is prob-counterable.*
2. *$Pr[L] < 1$. That is, the probability of an infinite random word to be in L is strictly smaller than 1.*
3. *Every DPW that recognizes L has a rejecting ergodic SCC.*

Analyzing the SCCs of DPWs for L also implies that ω-regular languages have a "safety-like" behavior in the following probabilistic sense:

▶ **Theorem 3.** *For every ω-regular language L, we have $Pr[\{v \in \Sigma^\omega : v \notin L$ and v does not have a prob-bad-prefix$\}] = 0$.*

Note that if L is prob-counterable, thus $Pr[v \notin L] > 0$, then $Pr[v$ has a prob-bad-prefix $\mid v \notin L] = 1$.

3.2 Relative Counterability

Recall that the standard definitions of bad-prefixes consider extensions in Σ^ω. When $\Sigma = 2^{AP}$ and the language L is examined with respect to a Kripke structure K over AP, it is interesting to restrict attention to extensions that are feasible in K. Consider a finite word $u \in \Sigma^*$. We say that u is a *bad-prefix for L with respect to K* (K*-bad-prefix*, for short) if u is a finite computation of K that cannot be extended to a computation of K that is in L. Thus, $u \in pref(L(K)) \setminus pref(L(K) \cap L)$. We say that L is *safety with respect to K* (K*-safety*, for short) if every computation of K that is not in L has a K-bad-prefix. We say that L is *counterable with respect to K* (K*-counterable*, for short) if the language L has a K-bad-prefix. Thus, $pref(L(K)) \not\subseteq pref(L(K) \cap L)$.

▶ **Theorem 4.** *Consider an ω-regular language $L \subseteq (2^{AP})^\omega$.*
1. *L is safety iff L is K-safety for every Kripke structure K over AP.*
2. *For every Kripke structure K over AP, we have that L is K-safety iff $L(K) \cap L$ is safety.*

Recall that if $L \subseteq \Sigma^\omega$ is safety and $L \neq \Sigma^\omega$, then L is counterable. Also, if L is K-safety and $L(K) \not\subseteq L$ then L is K-counterable. Note that it is possible that $L(K) \cap L$ is counterable but L is not K-counterable. For example, we can choose K and L such that $L(K) \subseteq L \neq (2^{AP})^\omega$. Then, L is not K-counterable, but a word u that is not a computation of K is a bad-prefix for $L(K)$, making it also a bad-prefix for $L(K) \cap L$. Hence, $L(K) \cap L$ is counterable.

3.3 Probabilistic Relative Counterability

We combine the probabilistic and relative definitions. Consider a Kripke structure $K = \langle AP, W, W_0, R, l\rangle$. A *$K$-walk-distribution* is a tuple $P = \langle p_{in}, p\rangle$ such that $M_{K,P} = \langle 2^{AP}, W, p_{in}, p, l\rangle$ is a labeled Markov chain that induces a graph that agrees with K. Thus, $p_{in}(w) > 0$ iff $w \in W_0$, and $p(w, w') > 0$ iff $R(w, w')$. A random walk on K with respect to P is a random walk on the Markov chain $\langle W, p_{in}, p\rangle$. We define the probability of an ω-regular language $L \subseteq (2^{AP})^\omega$ with respect to K and P as $Pr_{K,P}[L] = Pr_{M_{K,P}}[L]$. Namely, $Pr_{K,P}[L]$ is the probability that a computation obtained by a random walk on K with respect to P is in L. Let u be a finite computation of K and let $w_0, \ldots, w_k \in W^*$ be such that

$u = l(w_0), \ldots, l(w_k)$. We say that an infinite computation u' is a continuation of u with a random walk on K with respect to P if $u' = l(w_0), \ldots, l(w_{k-1}), l(w'_0), l(w'_1), \ldots$, where $w'_0, w'_1, \ldots$ is obtained by a random walk on K from w_k with respect to P (recall that we assume that different states in K are labeled differently). We define $Pr^u_{K,P}[L]$ as the probability that a computation obtained by continuing u with a random walk on K with respect to P is in L. Formally, we define $Pr^u_{K,P}$ using a conditional probability: $Pr^u_{K,P}[L] = Pr_{K,P}[L|$ the word starts with $u]$. We extend the definition to every labeled Markov chain $\mathcal{M}$; that is $Pr^u_{\mathcal{M}}[L] = Pr_{\mathcal{M}}[L|$ the word starts with $u]$. Thus, $Pr^u_{K,P}[L] = Pr^u_{M_{K,P}}[L]$.

Consider a Kripke structure K over AP and an ω-regular language $L \subseteq (2^{AP})^\omega$. We say that $u \in (2^{AP})^*$ is a *prob-bad-prefix for L with respect to K* (*K-prob-bad-prefix*, for short) if u is a finite computation of K such that $Pr^u_{K,P}[L] = 0$, for some K-walk-distribution P. Thus, a computation obtained by continuing u with some random walk on K is almost surely not in L. As we show in Lemma 5 below, the existential quantification on the K-walk-distribution P can be replaced by a universal one, or by the specific K-walk-distribution that traverses K uniformly at random. We say that L is *prob-counterable with respect to K* (*K-prob-counterable*, for short) if it has a K-prob-bad-prefix.

Note that if L is counterable, then L is prob-counterable. Also, if L is K-counterable then L is K-prob-counterable. As we have seen above, a language may be prob-counterable but not counterable. Taking $K = K_{AP}$, this implies that a language may be K-prob-counterable but not K-counterable. As an example with an explicit dependency in K, consider the counterable LTL formula $\psi = G(req \rightarrow X\, ack) \wedge FG(idle)$. Let K over $AP = \{req, ack, idle\}$ be such that the atomic propositions in AP are mutually exclusive and $L(K)$ contains exactly the computations that start in *req* and in which every *req* is immediately followed by *ack*, or computations in which *idle* is always valid. Note that while ψ is not K-counterable, it is K-prob-counterable, as every finite computation of K that starts with *req* is a K-prob-bad-prefix for ψ.

Consider an NBW $\mathcal{A}$. By [36, 11], deciding whether $Pr_{K,P}[L(\mathcal{A})] = 0$ or whether $Pr_{K,P}[L(\mathcal{A})] = 1$ is independent of the K-walk-distribution P. Consequently, we have the following.

▶ **Lemma 5.** [36, 11] *Let u be a finite computation of a Kripke structure K over AP, and let $L \subseteq (2^{AP})^\omega$ be an ω-regular language. For all pairs P and P' of K-walk-distributions, we have that $Pr^u_{K,P}[L] = 0$ iff $Pr^u_{K,P'}[L] = 0$ and $Pr^u_{K,P}[L] = 1$ iff $Pr^u_{K,P'}[L] = 1$.*

We can now point to equivalent definitions of K-prob-counterability.

▶ **Theorem 6.** *Consider an ω-regular language $L \subseteq (2^{AP})^\omega$ and a Kripke structure K over AP. The following are equivalent:*

1. *The language L is K-prob-counterable.*
2. *There is a finite computation u of K and a K-walk-distribution P such that $Pr^u_{K,P}[L] < 1$.*
3. *There is a finite computation u of K such that for all K-walk-distribution P, we have $Pr^u_{K,P}[L] < 1$.*
4. *There is a K-walk-distribution P s.t. $Pr_{K,P}[L] < 1$.*
5. *For all K-walk-distributions P, we have $Pr_{K,P}[L] < 1$.*

We can also generalize Theorem 3, and show that ω-regular languages have "safety-like" behaviors also with respect to Kripke structures, in the following probabilistic sense:

▶ **Theorem 7.** *Consider an ω-regular language $L \subseteq (2^{AP})^\omega$, a Kripke structure K over AP, and a K-walk-distribution P. Then, $Pr_{K,P}[\{u \in (2^{AP})^\omega : u \notin L$ and u does not have a K-prob-bad-prefix for $L\}] = 0$.*

If L is also K-prob-counterable then we also have $Pr_{K,P}[u \text{ has a } K\text{-prob-bad-prefix for } L \mid u \notin L] = 1$ for every K-walk-distribution P.

Conceptually, Theorem 6 implies that if an error has a positive probability to occur in a random execution of the system, then the specification is prob-counterable with respect to the system. Theorem 7 then suggests that in this case, a computation of the system that does not satisfy the specification, almost surely has a prob-bad-prefix with respect to the system. Thus, almost all the computations that violate the specification start with a prob-bad-prefix with respect to the system. Hence, attempts to find and return to the user such bad-prefixes are very likely to succeed.

4 Deciding Liveness

Recall that a language L is counterable iff L is not liveness. As discussed in Section 1, the complexity of the problem of deciding whether a given LTL formula is liveness is open [4]. In this section we solve this problem and prove that it is EXPSPACE-complete. The result would be handy also for our study of the probabilistic and relative variants.

▶ **Theorem 8.** *The problem of deciding whether a given LTL formula is liveness is EXPSPACE-complete.*

Proof. The upper-bound is known [35], and follows from the fact that every LTL formula ψ can be translated to an NBW $\mathcal{A}_\psi$ with an exponential blow-up [37]. By removing empty states from $\mathcal{A}_\psi$ and making all other states accepting, we get an NFW for $\mathit{pref}(L(\psi))$, which is universal iff ψ is liveness. The latter can be checked on-the-fly and in PSPACE, implying the EXPSPACE upper bound.

For the lower bound, we show a reduction from an exponent version of the *tiling problem*, defined as follows. We are given a finite set T, two relations $V \subseteq T \times T$ and $H \subseteq T \times T$ of tiles, an initial tile t_0, a final tile t_f, and a bound $n > 0$. We have to decide whether there is some $m > 0$ and a tiling of a $2^n \times m$-grid such that (1) The tile t_0 is in the bottom left corner and the tile t_f is in the top left corner, (2) A horizontal condition: every pair of horizontal neighbors is in H, and (3) A vertical condition: every pair of vertical neighbors is in V. Formally, we have to decide whether there is a function $f : \{0, \ldots, 2^n - 1\} \times \{0, \ldots, m-1\} \to T$ such that (1) $f(0,0) = t_0$ and $f(0, m-1) = t_f$, (2) For every $0 \leq i \leq 2^n - 2$ and $0 \leq j \leq m-1$, we have that $(f(i,j), f(i+1,j)) \in H$, and (3) For every $0 \leq i \leq 2^n - 1$ and $0 \leq j \leq m-2$, we have that $(f(i,j), f(i,j+1)) \in V$. When n is given in unary, the problem is known to be EXPSPACE-complete.

We reduce this problem to the problem of deciding whether an LTL formula is not liveness. Given a tiling problem $\tau = \langle T, H, V, t_0, t_f, n \rangle$, we construct a formula φ such that τ admits tiling iff φ has a *good-prefix* – one all whose extensions satisfy φ. Formally, $x \in \Sigma^*$ is a good prefix for φ iff for all $y \in \Sigma^\omega$, we have that $x \cdot y$ satisfies φ. Therefore, for $\psi = \neg\varphi$, we have τ admits tiling iff ψ is not liveness. The idea is to encode a tiling as a word over T, consisting of a sequence of rows (each row is of length 2^n). Such a word represents a proper tiling if it starts with t_0, has a last row that starts with t_f, every pair of adjacent tiles in a row are in H, and every pair of tiles that are 2^n tiles apart are in V. The difficulty is in relating tiles that are far apart. To do that we represent every tile by a block of length n, which encodes the tile's position in the row. Even with such an encoding, we have to specify a property of the form "for every i, if we meet a block with position counter i, then the next time we meet a block with position counter i, the tiles in the two blocks are in V". Such a property can be expressed in an LTL formula of polynomial length, but there are exponentially many

i's to check. The way to use liveness in order to mimic the universal quantification on all i's is essentially the following: the good prefix for φ encodes the tiling. The set of atomic propositions in φ includes a proposition \$ that is not restricted beyond this prefix. The property that checks V then has to hold in blocks whose counter equals the counter of the block that starts at the last \$ in the computation. Thus, universal quantification in i is replaced by the explicit universal quantification on suffixes in the definition of good prefixes. A detailed description of the reduction is included in the full version of the paper. ◀

When a language is given by an NBW, the complexity of deciding its liveness is much simpler:

▶ **Theorem 9.** *The problem of deciding whether a given NBW is liveness is PSPACE-complete.*

5 On Counterability

In this section we study the problem of deciding whether a given language is counterable as well as the length of short bad-prefixes and their detection. In order to complete the picture, we also compare the results to those of safety languages.

We start with the complexity of deciding safety and counterability. The results for safety are from [35]. These for counterability follow from Theorems 8 and 9 and the fact that L is counterable iff it is not liveness.

▶ **Theorem 10.** *Consider a language L.*

1. [35] *The problem of deciding whether L is safety is PSPACE-complete for L given by an LTL formula or by an NBW.*
2. *The problem of deciding whether L is counterable is EXPSPACE-complete for L given by an LTL formula and is PSPACE-complete for L given by an NBW.*

We find Theorem 10 surprising: both safety and counterability ask about the existence of bad-prefixes. In safety, a bad-prefix should exist to all bad words. In counterability, not all bad words have a bad-prefix, but at least some should have. Theorem 10 implies that there is something in LTL, yet not in NBWs, that makes the second type of existence condition much more complex.

We now turn to study the length of shortest bad-prefixes. Both (non-valid) safety and counterable languages have bad-prefixes. As we show, however, the complexity of counterability continues, and a tight bound on shortest bad-prefixes for counterable languages is exponentially bigger than that of safety languages. The gap follows from our ability to construct a *fine automaton* $\mathcal{A}_\psi^{fine}$ for all safety LTL formulas ψ [21]. The NFW $\mathcal{A}_\psi^{fine}$ is exponential in $|\psi|$, it accepts only bad-prefixes for ψ, and each computation that does not satisfy ψ has at least one bad-prefix accepted by $\mathcal{A}_\psi^{fine}$. A shortest witness to the nonemptiness of $\mathcal{A}_\psi^{fine}$ can serve as a bad-prefix. On the other hand, nothing is guaranteed about the behavior of $\mathcal{A}_\psi^{fine}$ when constructed for a non-safety formula ψ, thus it is of no help in the case of counterable languages that are not safety. In particular, the LTL formula used in the proof of Theorem 8 is neither safety nor its complement is safety, thus its doubly-exponential shortest bad-prefix does not contradict upper bounds known for these classes of languages.

▶ **Theorem 11.** *The length of shortest bad-prefixes for a language given by an LTL formula ψ is tightly exponential in $|\psi|$ in case ψ is safety, and is tightly doubly-exponential in $|\psi|$ in case ψ is counterable.*

When the specification formalism is automata, the difference between safety and counterable languages disappears:

▶ **Theorem 12.** *The length of shortest bad-prefixes for a safety or a counterable language given by an NBW $\mathcal{A}$ is tightly exponential in $|\mathcal{A}|$.*

6 On Relative Counterability

In this section we add a Kripke structure K to the setting and study K-counterablity and shortest K-bad-prefixes. Our results use variants of the product of automata for K and L, and we first define this product below. Consider a Kripke structure $K = \langle AP, W, W_0, R, l\rangle$ and an NBW $\mathcal{A} = \langle 2^{AP}, Q, Q_0, \delta, \alpha\rangle$. Essentially, the states of the product $\mathcal{A}_{K\times\mathcal{A}}$ are pairs in $W \times Q$. Recall that the states in K are differently labeled. Thus, we can define the product so that whenever it reads a letter in 2^{AP}, its next W-component is determined. Formally, we define the NBW $\mathcal{A}_{K\times\mathcal{A}} = \langle 2^{AP}, (W \cup \{s_0\}) \times Q, \{s_0\} \times Q_0, \rho, W \times \alpha\rangle$, where for all $\sigma \in 2^{AP}$, we have $\langle w', q'\rangle \in \rho(\langle s_0, q\rangle, \sigma)$ iff $w' \in W_0$ and $l(w') = \sigma$ and $q' \in \delta(q, \sigma)$, and for $w \in W$ we have $\langle w', q'\rangle \in \rho(\langle w, q\rangle, \sigma)$ iff $R(w, w')$ and $l(w') = \sigma$ and $q' \in \delta(q, \sigma)$. Thus, when the product $\mathcal{A}_{K\times\mathcal{A}}$ proceeds from state $\langle w, q\rangle$ with σ, its new W-component is the single successor of w that is labeled σ, paired with the σ-successors of q. It is easy to see that $L(\mathcal{A}_{K\times\mathcal{A}}) = L(K) \cap L(\mathcal{A})$. When $\mathcal{A}$ corresponds to an LTL formula ψ (that is, $L(\mathcal{A}) = L(\psi)$), we denote the product by $\mathcal{A}_{K\times\psi}$.

We start with the problem of deciding relative safety. By Theorem 4, a language L is K-safety iff $L(K) \cap L$ is safety. Thus, the check can be reduced to checking the safety of $\mathcal{A}_{K\times\mathcal{A}}$ (respectively $\mathcal{A}_{K\times\psi}$). This check, however, if done naively, is PSPACE in $|\mathcal{A}_{K\times\mathcal{A}}|$ (respectively $|\mathcal{A}_{K\times\psi}|$), which is PSPACE in $|K|$. The technical challenge is to find a more efficient way to do the check, and the one we describe in the proof is based on decomposing $\mathcal{A}_{K\times\mathcal{A}}$ so that the complementation that its safety check involves is circumvented. As for the lower bound, note that using the Kripke structure K_{AP}, one can reduce traditional safety to relative safety.[3] Our reduction, however, shows that the complexity of deciding K-safety coincides with that of model checking in both its parameters.

▶ **Theorem 13.** *Consider a Kripke structure K over AP and a language $L \subseteq (2^{AP})^\omega$. The problem of deciding whether L is K-safety is PSPACE-complete for L given by an NBW or by an LTL formula. In both cases it can be done in time linear and space polylogarithmic in $|K|$.*

We continue to relative counterability. We first show that the complexity of deciding counterability is carried over to the relative setting. For the upper bound, note that a language L is K-counterable iff $\mathit{pref}(L(K)) \cap \mathit{comp}(\mathit{pref}(L(K) \cap L)) \neq \emptyset$. Again, this check, if done naively, is PSPACE in $|K|$, and the challenge in the proof is to use the deterministic behavior of $\mathcal{A}_{K\times\mathcal{A}}$ with respect to the W-component of its states in order to avoid a blow-up in K in its complementation.

▶ **Theorem 14.** *Consider a Kripke structure K over AP and a language $L \subseteq (2^{AP})^\omega$. The problem of deciding whether L is K-counterable and finding a shortest K-bad-prefix is PSPACE-complete for L given by an NBW and is EXPSPACE-complete for L given by an LTL formula. In both cases it can be done in time linear and space polylogarithmic in $|K|$.*

[3] Indeed K_{AP} is exponential in $|AP|$, but safety is known to be PSPACE-hard also when the number of atomic propositions is fixed.

We now study the length of K-bad-prefixes. The upper bounds follow from the proof of Theorem 14, and for the lower ones, we rely on K_{AP} and the constructions described in the proofs of Theorems 11 and 12 in order to prove the dependency on $|\psi|$ and $|\mathcal{A}|$, and describe a family of Kripke structures requiring linear dependency in $|K|$:

▶ **Theorem 15.** *The length of a shortest K-bad-prefix for a K-counterable language L is tightly doubly-exponential in $|\psi|$, in case L is given by means of an LTL formula ψ, and is tightly exponential in $|\mathcal{A}|$, in case L is given by an NBW $\mathcal{A}$. In both cases, it is also tightly linear in $|K|$.*

Interestingly, when the LTL formula is K-safety, deciding its K-counterability and finding a K-bad-prefix can be done more efficiently, and its K-bad-prefixes are shorter. For the decidability problem, note that if an LTL formula ψ is K-safety, then ψ is K-counterable iff $K \not\models \psi$. Also, by Theorem 4, the fact ψ is K-safety implies that the NBW $\mathcal{A}_{K\times\psi}$ is safety, which is useful in the construction of fine automata. Formally, we have the following.

▶ **Theorem 16.** *Let K be a Kripke structure and let ψ be a K-safety LTL formula. Deciding whether ψ is K-counterable and finding a K-bad-prefix is PSPACE-complete in $|\psi|$. The length of shortest K-bad-prefixes is tightly exponential in $|\psi|$.*

Note that the space complexity of the algorithm described in the proof of Theorem 16 in also polylogarithmic in $|K|$. Finally, when the specification formalism is automata, the difference between K-safety and K-counterable languages disappears, and a short K-bad-prefix for a K-safety or K-counterable language given by an NBW $\mathcal{A}$ is tightly exponential in $|\mathcal{A}|$ and linear in $|K|$.

7 On Probabilistic Relative Counterability

In this section we study K-prob-counterability. By Theorem 6, an ω-regular language L is K-prob-counterable iff $Pr_{K,P}[L] < 1$ for some K-walk-distribution P. This, together with [11], imply the upper bound for the corresponding decision problem:

▶ **Theorem 17.** *Consider a language $L \subseteq (2^{AP})^\omega$ and a Kripke structure K over AP. Deciding whether L is K-prob-counterable can be done in time $O(|K| \cdot 2^{O(|L|)})$ or in space polynomial in $|L|$ and polylogarithmic in $|K|$, for L given by an LTL formula ψ, in which case $|L| = |\psi|$, or by an NBW $\mathcal{A}$, in which case $|L| = |\mathcal{A}|$. In both cases, the problem is PSPACE-complete.*

Thus, deciding whether an LTL formula is K-prob-counterable is exponentially easier than in the non-probabilistic case.

Recall that the study of K-counterability involved reasoning about the product of K and a nondeterministic automaton for the language. In the probabilistic setting, we need the automaton for the language to be deterministic. Let $\mathcal{D} = \langle 2^{AP}, S, s_0, \delta_{\mathcal{D}}, \alpha\rangle$ be a DPW for a language L and let $K = \langle AP, W, W_0, R, l\rangle$ be a Kripke structure. We define $\mathcal{D}_{K\times\mathcal{D}} = \langle 2^{AP}, W \times S, W_0 \times \{s_0\}, \rho, \alpha'\rangle$ as the product DPW of K and $\mathcal{D}$. Formally, we have $\alpha'(\langle w, s\rangle) = \alpha(s)$, and $\langle w', s'\rangle \in \rho(\langle w, s\rangle, \sigma)$ iff $[l(w) = \sigma$, $R(w, w')$ and $s' = \delta_{\mathcal{D}}(s, \sigma)]$. Note that $L(\mathcal{D}_{K\times\mathcal{D}}) = L(K) \cap L(\mathcal{D})$. Also, note that all of the successors of a state in $\mathcal{D}_{K\times\mathcal{D}}$ share the same second component.

In the probabilistic setting, we also need to define the product as a Markov chain. Let $\mathcal{D}$ and K be as above and let $P = \langle p_{in}, p\rangle$ be a K-walk-distribution. We define a labeled Markov chain $M' = \langle 2^{AP}, W \times S, p'_{in}, p', l'\rangle$, where $p' : (W \times S) \times (W \times S) \to [0, 1]$ is such that

$p'(\langle w,s\rangle,\langle w',s'\rangle) = p(\langle w,w'\rangle)$ if $\delta_{\mathcal{D}}(s,l(w)) = s'$, and otherwise $p'(\langle w,s\rangle,\langle w',s'\rangle) = 0$. Also, $p'_{in} : W \times S \to [0,1]$ is such that $p'_{in}(\langle w,s\rangle) = p_{in}(w)$ if $s = s_0$, and otherwise $p'_{in}(\langle w,s\rangle) = 0$. Finally, $l' : W \times S \to 2^{AP}$ is such that $l'(\langle w,s\rangle) = l(w)$. Thus, p'_{in} and p' attribute the states of $M = M_{K,P}$ by the deterministic behavior of $\mathcal{D}$. In particular, note that for every $\langle w,s\rangle \in W \times S$, we have $\sum_{\langle w',s'\rangle \in W\times S} p'(\langle w,s\rangle,\langle w',s'\rangle) = \sum_{w'\in W} p\langle w,w'\rangle = 1$.

Let $g : W \times S \to W$ be a function that projects pairs in $W \times S$ on their W-component, namely $g(\langle w,s\rangle) = w$. Consider random walks $X = X_1, X_2 \ldots$ and $X' = X'_1, X'_2 \ldots$ on M and M', respectively. Let $Y = Y_1, Y_2, \ldots$ be the projection of X' on W, thus $Y_i = g(X'_i)$ for $i = 1, 2, \ldots$. Note that the processes X and Y both take values in W^ω and that they have the same distribution. Therefore, we have $Pr_M[L] = Pr_{M'}[L]$. Also, for a finite computation $u = l(w_0), \ldots, l(w_n)$ of K, we have $Pr^u_M[L] = Pr^u_{M'}[L]$. Note that $w_0, \ldots, w_n$ induces a single finite path $\langle w_0,s_0\rangle, \ldots, \langle w_n,s_n\rangle$ in $\mathcal{D}_{K\times\mathcal{D}}$, and that every infinite path in K induces a single infinite path in $\mathcal{D}_{K\times\mathcal{D}}$. For a finite computation u, let $reach(u)$ be the state reached in $\mathcal{D}_{K\times\mathcal{D}}$ after traversing u. Thus, $reach(u) = \langle w_n,s_n\rangle$. By the above, $Pr^u_{M'}[L]$ is the probability that a random walk in M' from $reach(u)$ is an accepting run in $\mathcal{D}_{K\times\mathcal{D}}$. For a state x of M', we denote by γ_x the probability that a random walk from x in M' is an accepting run in $\mathcal{D}_{K\times\mathcal{D}}$. Note that a finite computation u is a K-prob-bad-prefix iff $\gamma_{reach(u)} = 0$.

▶ **Lemma 18.** *Deciding whether γ_x is 0, 1 or in $(0,1)$, for all states x in M', can be done in time linear in $|\mathcal{D}|$ and in $|K|$ or in space polylogarithmic in $|\mathcal{D}|$ and in $|K|$. Furthermore, the probability γ_x can be calculated in time polynomial in $|\mathcal{D}|$ and in $|K|$.*

We turn to study the complexity of probabilistic relative counterability of ω-regular languages. Handling a language given by an NBW can proceed by an exponential translation to a DPW [31]. For languages given by LTL formulas, going to DPWs involves a doubly-exponential blow-up. We show that in order to find a K-prob-bad-prefix for an LTL formula, we can carefully proceed according to the syntax of the formula and do exponentially better than an algorithm that translates the formulas to automata. We note that the PSPACE-hardness in Theorem 17 is by a reduction from the universality problem. Thus, we cannot hope to obtain a PSPACE algorithm by translating LTL formulas to NBWs, unless the structure of the latter is analyzed to a level in which it essentially follows the structure of the LTL formula (see, for example, [12] for such an approach applied in probabilistic LTL model checking).

7.1 On probabilistic relative counterability of NBWs

We start with an algorithm for finding a shortest K-prob-bad-prefix for a language given by an NBW $\mathcal{A}$. For that, we need to find a shortest word whose path in M' reaches a state x for which $\gamma_x = 0$. By Lemma 18, we thus have the following:

▶ **Theorem 19.** *Consider an NBW $\mathcal{A}$ over the alphabet 2^{AP} and a Kripke structure K over AP. Finding a shortest K-prob-bad-prefix for $L(\mathcal{A})$ can be done in space polynomial in $|\mathcal{A}|$ and polylogarithmic in $|K|$ or in time exponential in $|\mathcal{A}|$ and linear in $|K|$. Furthermore, the length of the shortest K-prob-bad-prefix for $L(\mathcal{A})$ is tightly exponential in $|\mathcal{A}|$ and tightly linear in $|K|$.*

Consider a Kripke structure K over AP and a language $L \subseteq (2^{AP})^\omega$ that is K-prob-counterable. In practice, the user of the model-checking tool often has some estimation of the likelihood of every transition in the system. That is, we assume that the user knows what the typical K-walk-distribution P in a typical behavior of the system is. Clearly, there is a trade-off between the length of a counterexample and its "precision", in the sense that the

longer a finite prefix of an erroneous computation is, the larger is the probability in which it is a K-prob-bad-prefix. We want to allow the user to play with this trade-off and thus define the following two problems:

- The *shortest bounded-prob-K-bad-prefix* problem is to return, given K, L, and $0 < \gamma < 1$, a shortest finite computation u of K such that $Pr^u_{K,P}[L] < \gamma$.
- The *bounded-length prob-K-bad-prefix* problem is to return, given K, L, and $m \geq 1$, a finite computation u of K such that $|u| \leq m$ and $Pr^u_{K,P}[L]$ is minimal.

Using Lemma 18, we can carefully reduce both problems to classical problems in graph algorithms, applied to $\mathcal{D}_{K \times \mathcal{D}}$:

▶ **Theorem 20.** *The shortest bounded-prob-K-bad-prefix and the bounded-length prob-K-bad-prefix problems, for a language given by an NBW $\mathcal{A}$, can be solved in time exponential in $|\mathcal{A}|$ and polynomial in $|K|$.*

We note that using our construction of M', together with Lemma 18, we can reduce the calculation of $Pr_{K,P}[L(\mathcal{A})]$ or the problem of its classification to 1, 0, or $(0, 1)$ to a sequence of calculations in M', simplifying the known result of [11] (Theorem 4.1.7 there).

7.2 On Probabilistic Relative Counterability of LTL formulas

We describe an algorithm for finding a K-prob-bad-prefix for an LTL formula θ. Like the model-checking algorithm in [11], our algorithm proceeds by iteratively replacing the innermost temporal subformula of θ by a fresh atomic proposition and adjusting K so that the probability of a computation obtained by a random walk to satisfy the specification is maintained. In more detail, we construct a sequence $K^0, K^1, \ldots$ of Kripke structures and a sequence $\theta^0, \theta^1, \ldots$ of LTL formulas such that $K^0 = K$ and $\theta^0 = \theta$, and K^{i+1} is obtained from K^i by applying a transformation that depends on the innermost temporal operator in θ^i, which is replaced by a fresh atomic proposition in θ^{i+1}. We show that a K^i-prob-bad-prefix for θ^i can be constructed by extending a K^{i+1}-prob-bad-prefix for θ^{i+1}, resulting in a recursive construction of a K-prob-bad-prefix for θ.

▶ **Theorem 21.** *Finding a K-prob-bad-prefix for a K-prob-counterable LTL formula θ can be done in time $O(|K| \cdot 2^{|\theta|})$ or in space polynomial in $|\theta|$ and polylogarithmic in $|K|$. Furthermore, the K-prob-bad-prefix that is found is of length $O(|K| \cdot 2^{|\theta|})$.*

We now study the length of shortest K-prob-bad-prefixes.

▶ **Theorem 22.** *The length of a shortest K-prob-bad-prefix for a K-prob-counterable language given by an LTL formula ψ is tightly exponential in $|\psi|$ and tightly linear in $|K|$.*

Note that the K-prob-bad-prefix that our algorithm finds is not necessarily the shortest, however it matches the lower bound from Theorem 22.

Thus, we showed that the probabilistic approach for relative bad prefixes for LTL formulas is exponentially better than the non-probabilistic approach both in its complexity and in the length of the prefixes.

8 On Probabilistic Counterability

In this section we study prob-counterability. The solutions to the three basic problems are specified in Theorems 23, 24, and 25 below. The upper bound for the first follows from the fact that, by Theorem 2, an ω-regular language L is prob-counterable iff $Pr[L] < 1$, which,

by [11], can be checked in PSPACE. The latter two follow from the results in Section 7, taking the Kripke structure to be K_{AP}.

▶ **Theorem 23.** *The problem of deciding whether a language L is prob-counterable is PSPACE-complete for L given by an LTL formula or by an NBW.*

▶ **Theorem 24.** *Let $\mathcal{A}$ be an NBW. Finding a shortest prob-bad-prefix for $L(\mathcal{A})$ can be done in time exponential in $|\mathcal{A}|$, or in space polynomial in $|\mathcal{A}|$. Furthermore, the length of the shortest prob-bad-prefix is tightly exponential in $|\mathcal{A}|$.*

▶ **Theorem 25.** *Finding a prob-bad-prefix for a prob-counterable LTL formula ψ can be done in time $2^{O(|\psi|)}$ or in space polynomial in $|\psi|$. Furthermore, the prob-bad-prefix that is found is of length $2^{O(|\psi|)}$. The shortest prob-bad-prefix for ψ is tightly exponential in $|\psi|$.*

Thus, the exponential advantage of the probabilistic approach in the case the language is given by an LTL formula is carried over to the non-relative setting. When the specification is given by means of an NBW, the complexities of the probabilistic and non-probabilistic approaches coincide. The probabilistic approach, however, may return more bad prefixes.

9 Discussion

We extended the applicability of finite counterexamples by introducing relative and probabilistic bad-prefixes. This lifts the advantage of safety properties, which always have bad-prefixes, to ω-regular languages that are not safety. We believe that K-bad-prefixes and K-prob-bad-prefixes may be very helpful in practice, as they describe a finite execution that leads the system to an error state. From a computational point of view, finding a K-bad-prefix for an LTL formula ψ is unfortunately EXPSPACE-complete in $|\psi|$. Experience shows that even highly complex algorithms often run surprisingly well in practice. Also here, the complexity originates from the blow-up in the translation of LTL to automata, which rarely happens in practice. In cases the complexity is too high, we suggest the following two alternatives, which do not go beyond the PSPACE complexity of LTL model checking (and, like model checking, are NLOGSPACE in K): (1) Recall that when ψ is K-safety and $K \not\models \psi$, then finding a K-bad-prefix can be done in PSPACE. Thus, we suggest to check ψ for K-safety with the algorithm from Theorem 13, and then apply the algorithm from Theorem 16. (2) Recall that finding a K-prob-bad-prefix is only PSPACE-complete. Thus, we suggest to apply the algorithm from Theorem 21. Note that the probabilistic approach is not only exponentially less complex, but may be essential when ψ is K-prob-counterable and not K-counterable.

When a user gets a lasso-shaped counterexample, he can verify that indeed it does not satisfy the specification. For finite bad-prefixes, the user knows that they lead the system to an error state, and it is desirable to accompany the prefix with information explaining why these states are erroneous. We suggest the following three types of explanations. (1) A K-bad-prefix leads the product $K \times \mathcal{A}_\psi$ to states $\langle w, S\rangle$ that are empty. Recall that the states of $\mathcal{A}_\psi$ consist of subsets of subformulas of ψ, and that $\langle w, S\rangle$ being empty means that w does not satisfy the conjunction of the formulas in S [37]. Returning S to the user explains what makes w an error state. (2) Researchers have studied *certified model checking* [24], where a positive answer of model checking (that is, $K \models \psi$) is accompanied by a certificate – a compact explanation as to why $K \times \mathcal{A}_{\neg\psi}$ is empty. In our setting, certificates can provide a compact explanation as to why $K \times \mathcal{A}_\psi$ with initial state $\langle w, S\rangle$ is empty. (3) When a K-prob-bad-prefix u that is not a K-bad-prefix is returned, it may be helpful to accompany u with an infinite lasso-shaped computation τ of the system that starts with u and does satisfy

the specification. Thus, the user would get an *exception*: he would know that almost all computations that start in u except for τ (and possibly more computations, whose probability is 0) violate ψ. The exceptional correct behavior would help the user understand why almost all other behaviors are incorrect.

Acknowledgment. We thank Moshe Y. Vardi for helpful discussions.

References

1 E. Ábrahám, B. Becker, C. Dehnert, N. Jansen, J.-P. Katoen, and R. Wimmer. Counterexample generation for discrete-time Markov models. In *FMESM*, pages 65–121. Springer, 2014.

2 B. Alpern and F. B. Schneider. Defining liveness. *IPL*, 21:181–185, 1985.

3 B. Alpern and F. B. Schneider. Recognizing safety and liveness. *Dist. computing*, 2:117–126, 1987.

4 D. A. Basin, C. C. Jiménez, F. Klaedtke, and E. Zalinescu. Deciding safety and liveness in TPTL. *IPL*, 114(12):680–688, 2014.

5 A. Biere, C. Artho, and V. Schuppan. Liveness checking as safety checking. In *Proc. 7th FMICS*, LNTCS 66:2, 2002.

6 J. R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. Int. Congress on Logic, Method, and Philosophy of Science. 1960*, pages 1–12. Stanford University Press, 1962.

7 E. M. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.

8 E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Proc. 12th CAV*, LNCS 1855, pages 154–169. Springer, 2000.

9 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, 1995.

10 T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 1990.

11 C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *J. ACM*, 42:857–907, 1995.

12 J. M. Couvreur, N. Saheb, and G. Sutre. An optimal automata approach to LTL model checking of probabilistic systems. In *Proc. 10th LPAR*, LNCS 2850, pages 361–375. Springer, 2003.

13 S. Ben David and O. Kupferman. A framework for ranking vacuity results. In *Proc. 11th ATVA*, LNCS 8172, pages 148–162. Springer, 2013.

14 V. Diekert, A. Muscholl, and I. Walukiewicz. A Note on Monitors and Büchi automata In *Proc. 12th ICTAC*, to appear, 2015.

15 D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *Proc. 7th POPL*, pages 163–173, 1980.

16 C. Grinstead and J. Laurie Snell. 11:markov chains. In *Introduction to Probability*. American Mathematical Society, 1997.

17 K. Havelund and G. Rosu. Efficient monitoring of safety properties. *STTT*, 6(2):18–173, 2004.

18 T. A. Henzinger. Sooner is safer than later. *IPL*, 43(3):135–141, 1992.

19 J.-P. Katoen, L. Song, and L. Zhang. Probably safe or live. In *Proc. 29th LICS*, pages 55:1–55:10, 2014.

20 J. G. Kemeny, J. L. Snell, and A. W. Knapp. *Denumerable Markov Chains*. Springer, 1976.

21 O. Kupferman and R. Lampert. On the construction of fine automata for safety properties. In *Proc. 4th ATVA*, LNCS 4218, pages 110–124. Springer, 2006.

22 O. Kupferman and S. Sheinvald-Faragy. Finding shortest witnesses to the nonemptiness of automata on infinite words. In *Proc. 17th CONCUR*, LNCS 4137, pages 492–508. Springer, 2006.

23 O. Kupferman and M. Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001.

24 O. Kupferman and M. Y. Vardi. From complementation to certification. In *Proc. 10th TACAS*, LNCS 2988, pages 591–606. Springer, 2004.

25 O. Kupferman and M. Y. Vardi. Synthesis of trigger properties. In *Proc. 16th LPAR*, LNCS 6355, pages 312–331. Springer, 2010.

26 M. Lippmann. Temporalised description logics for monitoring partially observable events. 2014.

27 Z. Manna and A. Pnueli. Adequate proof principles for invariance and liveness properties of concurrent programs. *Science of Computer Programming*, 4:257–289, 1984.

28 A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential time. In *Proc. 13th SWAT*, pages 125–129, 1972.

29 U. Nitsche and P. Wolper. Relative liveness and behavior abstraction. In *Proc. 24th POPL*, pages 45–52. ACM, 1997.

30 S. Owicki and L. Lamport. Proving liveness properties of concurrent programs. *ACM TOPLAS*, 4(3):455–495, 1982.

31 N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. *LMCS*, 3(3):5, 2007.

32 A. Pnueli. The temporal semantics of concurrent programs. *TCS*, 13:45–60, 1981.

33 V. Schuppan and A. Biere. Shortest counterexamples for symbolic model checking of LTL with past. In *Proc. 11th TACAS*, LNCS 3440, pages 493–509. Springer, 2005.

34 V. Schuppan and A. Biere. Liveness checking as safety checking for infinite state spaces. *ENTCS*, 149(1):79–96, 2006.

35 A. P. Sistla. Safety, liveness and fairness in temporal logic. *FAC*, 6:495–511, 1994.

36 M. Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proc. 26th FOCS*, pages 327–338, 1985.

37 M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *I& C*, 115(1):1–37, 1994.