Acta Informatica (2018) 55:703–732
https://doi.org/10.1007/s00236-017-0307-4

ORIGINAL ARTICLE

Spanning the spectrum from safety to liveness

Rachel Faran[1] · **Orna Kupferman**[1]

Received: 6 June 2016 / Accepted: 13 October 2017 / Published online: 23 October 2017

Abstract Of special interest in formal verification are *safety* specifications, which assert that the system stays within some allowed region, in which nothing "bad" happens. Equivalently, a computation violates a safety specification if it has a "bad prefix"—a prefix all whose extensions violate the specification. The theoretical properties of safety specifications as well as their practical advantages with respect to general specifications have been widely studied. Safety is binary: a specification is either safety or not safety. We introduce a quantitative measure for safety. Intuitively, the *safety level* of a language L measures the fraction of words not in L that have a bad prefix. In particular, a safety language has safety level 1 and a liveness language has safety level 0. Thus, our study spans the spectrum between traditional safety and liveness. The formal definition of safety level is based on probability and measures the probability of a random word not in L to have a bad prefix. We study the problem of finding the safety level of languages given by means of deterministic and nondeterministic automata as well as LTL formulas, and the problem of deciding their membership in specific classes along the spectrum (safety, almost-safety, fraction-safety, etc.). We also study properties of the different classes and the structure of deterministic automata for them.

1 Introduction

Today's rapid development of complex and safety-critical systems requires reliable verification methods. In formal verification, 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. Of special interest are specifications asserting that the observed behavior of

The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC Grant Agreement No 278410, and from The Israel Science Foundation (Grant No 1229/10).

✉ Orna Kupferman
orna@cs.huji.ac.il

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

the system always stays within some allowed region, in which nothing "bad" happens. For example, we may want to assert that every message sent is acknowledged in the next cycle. Such specifications of systems are called *safety specifications*. Intuitively, a specification ψ is a safety specification if every violation of ψ occurs after a finite execution of the system. In our example, if in a computation of the system a message is sent without being acknowledged in the next cycle, this occurs after some finite execution of the system. Also, once this violation occurs, there is no way to "fix" the computation.

In order to define safety formally, we refer to computations of a non-terminating system as infinite words over an alphabet Σ. Consider a language L of infinite words over Σ. That is, $L \subseteq \Sigma^\omega$. A finite word $x \in \Sigma^*$ is a *bad prefix* for L iff 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 an infinite word in L. A language L is a *safety language* if every word not in L has a finite bad prefix.[1]

The interest in safety started with the quest for natural classes of specifications. The theoretical aspects of safety have been extensively studied [2,22,23,29]. With the growing success and use of formal verification, safety has turned out to be interesting also from a practical point of view [9,12,16]. Indeed, the ability to reason about finite prefixes significantly simplifies both enumerative and symbolic algorithms. In the first, safety circumvents the need to reason about complex ω-regular acceptance conditions. For example, methods for synthesis, program repair, or parametric reasoning are much simpler for safety properties [11,27]. In the second, it circumvents the need to reason about cycles, which is significant in both BDD-based and SAT-based methods [3,4].

In addition to a rich literature on safety, researchers have studied additional classes, such as co-safety and liveness [2,22]. A language L is a *co-safety* language if the complement of L, namely the language of words not in L, is safety. Equivalently, every word in L has a *good prefix*—one all whose extensions result in a word in L. A language L is a *liveness* language if it does not have bad prefixes. Thus, every word in Σ^* can be extended to a word in L. For example, if $\Sigma = \{a, b\}$, then $L = \{a^\omega, b^\omega\}$ is a safety language, its complement is both co-safety and liveness, and $L = (a + b)^* \cdot b^\omega$ is a liveness language that is neither safety nor co-safety.

From a theoretical point of view, the importance of safety and liveness languages stems from their topological characteristics. Consider the natural topology on Σ^ω, where similarity between words corresponds to the 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 languages are exactly the closed sets, co-safety languages are the open sets, and liveness languages are the dense sets (that is, they intersect every nonempty open set) [1]. This, for example, implies that every linear specification can be expressed as a conjunction of a safety and a liveness specification [1,2].

Safety is binary: a specification is either safety or not safety. In this work, we introduce a quantitative measure for safety. We define the *safety level* of a language L as the probability of a word not in L to have a bad prefix. From a theoretical point of view, our study spans the spectrum between traditional safety and liveness. From a practical point of view, the higher the safety level of L is, the bigger is the chance that algorithms designed for safety languages would work appropriately for L. For example, when we apply algorithms designed for safety languages to languages with safety level 1, we may get a one-sided error for some words,

[1] The definition of safety we consider here is given in [1,2], it coincides with the definition of limit closure defined in [8], and is different from the definition in [19], which also refers to the property being closed under stuttering.

namely words that are not in the language yet have no bad prefix, yet the measure of such words is zero.

Let us describe our framework and results in more detail. A random word over an alphabet Σ is a word in which for all indices i, the i-th letter is drawn uniformly at random. Thus, we assume a uniform probability distribution on Σ^ω, and the probability of a language $L \subseteq \Sigma^\omega$ is the probability of the event L. For example, the probability of $Pr(a \cdot (a+b)^\omega)$ is $\frac{1}{2}$. Now, for a language $L \subseteq \Sigma^\omega$, the safety level of L is the conditional probability of the event "words with a bad prefix for L" given the event "words not in L". That is, intuitively, the safety level measures the fraction of words not in L that have a bad prefix. When L is a safety language, all the words not in L have a bad prefix, thus the safety level of L is 1. When L is liveness, no word has a bad prefix, thus the safety level is 0. A language may have safety level 1 without being safety. For example, when $AP = \{a, b\}$, then the safety level of the non-safety specification $\psi = aUb$ (that is, a until b) is 1. Indeed, almost all computations (that is, all but $\{a\}^\omega$) that violate ψ have a bad prefix.[2] Also, languages may have a fractional safety level. For example, $\varphi = a \wedge FGb$ (that is, a and eventually always b) has safety level $\frac{1}{2}$, as only words that start with a letter satisfying $\neg a$ have a bad prefix for φ and almost all words that start with a letter satisfying a do not satisfy φ either.

We partition safety levels to four classes: safety, *almost-safety* [the safety level is 1 but the language is not safety), *frac-safety* (the safety level is in (0, 1)], and liveness (the safety level is 0, as we prove in Proposition 3.3). We define a dual classification for co-safety and examine possible combinations. For example, it is shown in [17] that the intersection of safety and co-safety languages is exactly the set of *bounded languages*—these that are recognizable by cycle-free automata, which correspond to *clopen* sets in topology. We study all intersections, some of which we prove to be empty. For example, there is no language that is both co-safety and frac-safety. We study the problem of calculating the safety level of a given language and prove that it can be solved in linear, exponential, and doubly-exponential time for languages given by deterministic parity automata, nondeterministic Büchi automata, and LTL formulas, respectively.

We then turn to study the classification problem, where the goal is to decide whether a given language belongs to a given class. The problem was studied for the classes of safety and liveness properties [15,29], and we study it for almost-safety and frac-safety. We show that the complexities for almost-safety coincide with these known for safety; that is, the problem is NLOGSPACE-complete for deterministic automata, and is PSPACE-complete for nondeterministic automata and LTL formulas. The complexities for frac-safety coincide with these known for livensess, where the treatment of LTL formulas is exponentially harder than that of nondeterministic automata and is EXPSPACE-complete. Our results are based on an analysis of the strongly connected components of deterministic automata for the language, and involve other results of interest on the expressive power of deterministic automata. In particular, we prove that frac-safety languages cannot be recognized by deterministic Büchi automata. This is in contrast with safety and almost-safety languages, which can always be recognized by deterministic Büchi automata, and liveness languages, some of which can be recognized by such automata.

Recall that the intersection of safety and co-safety languages is exactly the set of bounded languages, which correspond to clopen sets in topology. Intuitively, a language L is bounded if every word $w \in \Sigma^*$ has a *determined prefix*, namely a prefix that is either good (in case $w \in L$) or bad (in case $w \notin L$). We define the *bounding level* of a language L as the

[2] Note that $\Sigma = 2^{AP}$ and our probability distribution is such that for each atomic proposition a and for each position in a computation, the probability that a holds in the position is $\frac{1}{2}$.

probability of a word to have a determined prefix. We show how different safety and co-safety levels induce different bounding levels, and we study four classes of specifications that are induced by their bounding levels. For example, a language L is *pending*, which is the dual of clopen, if it is both liveness and co-liveness. In other words, every word in Σ^* can be extended both to a word in L and to a word not in L. We study the problem of classifying specifications into their bounding class. We show that the classification can be based on the classification to safety and co-safety classes, or by a direct analysis of deterministic automata for the language.

The paper is organized as follows. In Sect. 2, we define the basic notions of LTL, automata, and the probabilitic setting. In Sect. 3, we introduce the classes of safety and co-safety, and show how to find the safety level of a given language. Section 4 discusses the problem of finding the safety class of a given language, and shows several interesting conclusions regarding expressive power. In Sects. 5 and 6 we extend our results to clopen and bounded languages, and present the relation between the safety, co-safety and bounded classes. In Sect. 7, we discuss possible extensions of our approach as well as its practical applications.

2 Preliminaries

2.1 Linear temporal logic

For a finite alphabet Σ, an infinite *word* $w = \sigma_1 \cdot \sigma_2 \cdots$ 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. Formulas of LTL are constructed from a set AP of atomic propositions using the usual Boolean operators and the temporal operators X ("next time") and U ("until"). Formally, an LTL formula over AP is defined as follows:

- *true*, *false*, or p, for $p \in AP$.
- $\neg\psi_1$, $\psi_1 \wedge \psi_2$, $X\psi_1$, or $\psi_1 U \psi_2$, where ψ_1 and ψ_2 are LTL formulas.

The semantics of LTL is defined with respect to infinite computations $\pi = \sigma_1, \sigma_2, \sigma_3, \ldots$, where for every $i \geq 1$, the set $\sigma_i \in 2^{AP}$ is the set of atomic propositions that hold in the i-th position of π. Consider a computation $\pi = \sigma_1, \sigma_2, \sigma_3, \ldots \in (2^{AP})^\omega$. We use $\pi, i \models \psi$ to indicate that an LTL formula ψ holds in position i of π. The relation $\models$ is inductively defined, for every computation π and position i, as follows.

- $\pi, i \models \mathit{true}$ and $\pi, \quad i \not\models \mathit{false}$.
- $\pi, i \models p$ iff $p \in \sigma_i$, where $p \in AP$ is an atomic proposition.
- $\pi, i \models \neg\psi_1$ iff $\pi, \quad i \not\models \psi_1$.
- $\pi, i \models \psi_1 \wedge \psi_2$ iff $\pi, \quad i \models \psi_1$ and $\pi, i \models \psi_2$.
- $\pi, i \models X\psi_1$ iff $\pi, \quad i + 1 \models \psi_1$.
- $\pi, i \models \psi_1 U \psi_2$ iff there exists $k \geq 0$ such that $\pi, i + k \models \psi_2$ and $\pi, i + j \models \psi_1$ for all $0 \leq j < k$.

Writing LTL formulas, it is convenient to use the abbreviations G ("always") and F ("eventually"). Formally, the abbreviations follow the following semantics.

- $F\psi_1 = \mathit{true} U \psi_1$. That is, $\pi, i \models F\psi_1$ iff there is $k \geq 0$ such that $\pi, i + k \models \psi_1$.
- $G\psi_1 = \neg F \neg \psi_1$. That is, $\pi, i \models G\psi_1$ iff for all $k \geq 0$ we have that $\pi, i + k \models \psi_1$.

We use $\pi \models \psi$ to indicate that $\pi, 1 \models \psi$.

Each LTL formula ψ over AP defines a language $[\![\psi]\!] \subseteq (2^{AP})^\omega$ of the computations that satisfy ψ. Formally, $[\![\psi]\!] = \{\pi \in (2^{AP})^\omega : \pi \models \psi\}$.

2.2 Automata

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 set of *initial states*, $\delta : Q \times \Sigma \to 2^Q$ is a *transition function*, and α is an *acceptance condition*. The automaton $\mathcal{A}$ is *deterministic* if $|Q_0| = 1$ and $|\delta(q, \sigma)| \leq 1$ for all states $q \in Q$ and letters $\sigma \in \Sigma$.

A *run* of $\mathcal{A}$ on an infinite input word $w = \sigma_1 \cdot \sigma_2 \cdots \in \Sigma^\omega$, is an infinite sequence of states $r = q_0, q_1, \ldots$ such that $q_0 \in Q_0$, and for all $i \geq 0$, we have $q_{i+1} \in \delta(q_i, \sigma_{i+1})$. The acceptance condition α determines which runs are accepting. 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. We consider two acceptance conditions.

A set S of states satisfies a *Büchi* acceptance condition $\alpha \subseteq Q$ if $S \cap \alpha \neq \emptyset$. That is, a run r is accepting iff $\inf(r) \cap \alpha \neq \emptyset$. A richer acceptance condition is *parity*, where $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_k\}$, with $\alpha_1 \subseteq \alpha_2 \subseteq \cdots \subseteq \alpha_k = Q$, and a set S satisfies α if the minimal index i for which $S \cap \alpha_i \neq \emptyset$ is even. That is, a run r is accepting iff the minimal index i for which $\inf(r) \cap \alpha_i \neq \emptyset$ is even.

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 $L(\mathcal{A})$, is the set of words that $\mathcal{A}$ accepts. We use NBW, DBW, and DPW to abbreviate nondeterministic Büchi, deterministic Büchi, and deterministic parity word automata, respectively.

Theorem 2.1 [26,28,32]

- *Given an LTL formula ψ of lenght n, we can construct an NBW $\mathcal{A}_\psi$ such that $\mathcal{A}_\psi$ has $2^{O(n)}$ states and $L(\mathcal{A}_\psi) = [\![\psi]\!]$.*
- *Given an NBW with n states, we can construct an equivalent DPW with $2^{O(n \log n)}$ states.*

Consider a directed graph $G = \langle V, E \rangle$. A *strongly connected set* of G (SCS) is a set $C \subseteq V$ of vertices such that for every two vertices $v, v' \in C$, there is a path from v to v'. An SCS C is *maximal* if it cannot be extended to a larger SCS. Formally, for every nonempty set $C' \subseteq V \setminus C$, we have that $C \cup C'$ is not an SCS. The maximal strongly connected sets are also termed *strongly connected components* (SCC). An automaton $\mathcal{A} = \langle \Sigma, Q, Q_0, \delta, \alpha \rangle$ induces a directed graph $G_\mathcal{A} = \langle Q, E \rangle$ in which $\langle q, q' \rangle \in E$ iff there is a letter σ such that $q' \in \delta(q, \sigma)$. When we talk about the SCSs and SCCs of $\mathcal{A}$, we refer to these of $G_\mathcal{A}$.

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. We say that a path $\pi = \pi_0, \pi_1, \pi_2, \ldots$ in G reaches an ergodic SCC C, if there exists $i \geq 0$ such that for all $j \geq i$, we have that $\pi_j \in C$. Note that if $\pi_i \in C$ for some i, then π reaches the ergodic SCC C.

2.3 Probability

Given a set S of elements, a *probability distribution* on S is a function $Pr : S \to [0, 1]$ such that $\Sigma_{s \in S} Pr(s) = 1$. An *event* is a set $A \subseteq S$. The probability of A is then $\Sigma_{s \in A} Pr(s)$. Given two events A and B with $Pr(B) > 0$, the *conditional probability of A given B*, denoted $Pr(A \mid B)$, is defined as the ratio between the probability of the joint of events A and B, and the probability of B. That is, $Pr(A \mid B) = \dfrac{Pr(A \cap B)}{Pr(B)}$.

Consider an alphabet Σ. A random word over Σ is a word in which for all indices i, the i-th letter is drawn 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 that q holds in the position is $\frac{1}{2}$. The probabilistic model above induces a probability distribution on Σ^ω, where the probability of a word is the product of the probabilities of its letters. The probability of a language $L \subseteq \Sigma^\omega$, denoted $Pr(L)$, is then the probability of the event L. It is known that regular languages have measurable probabilities [6]. For an LTL formula ψ, we use $Pr(\psi)$ to denote $Pr([\![\psi]\!])$. For example, the probabilities of Xp, Gp, and Fp are $\frac{1}{2}$, 0, and 1, respectively. See also Remark 2.3 below.

The following lemma states two fundamental properties of runs of automata on random words (see, for example, [14]).

Lemma 2.2 *Consider an automaton $\mathcal{A}$.*

1. *A run of $\mathcal{A}$ on a random word reaches some ergodic SCC with probability* 1.
2. *An infinite run that reaches an ergodic SCC C of $\mathcal{A}$ visits all the states in C infinitely often with probability* 1.

We assume that all SCCs of automata are reachable. In addition, we distinguish between several types of ergodic SCCs. Let $\mathcal{A}$ be an automaton. An ergodic SCC C of $\mathcal{A}$ is *accepting* if a random path that reaches C is accepting with probability 1. Similarly, C is *rejecting* if a random path that reaches C is rejecting with probability 1. Recall that for an acceptance condition α, a run r is accepting iff $\inf(r)$ satisfies α. It follows from Lemma 2.2 that an ergodic SCC C is accepting iff it satisfies α. We say that an ergodic SCC C of $\mathcal{A}$ is *pure accepting* if every path π that reaches C is accepting. Similarly, an ergodic SCC C of $\mathcal{A}$ is *pure rejecting* if every path π that reaches C is rejecting. Note that pure accepting and pure rejecting ergodic SCCs are equivalent to accepting and rejecting sinks, respectively.

Remark 2.3 The described characterization of ergodic SCCs suggests an algorithm for calculating the probability of a language given by a DPW. Given a DPW $\mathcal{A}$, we can find its ergodic SCCs and classify them to accepting and rejecting SCCs in linear time [31]. Recall that an accepting ergodic SCC accepts words with probability 1, and a rejecting ergodic SCC rejects words with probability 1. Therefore, if we sum the probabilities of reaching every accepting ergodic SCCs in $\mathcal{A}$, we get $Pr(L(\mathcal{A}))$. It is easy to see that in automata for Xp, Gp, and Fp, the probability to reach an accepting ergodic SCC are $\frac{1}{2}$, 0, and 1, respectively. Since LTL formulas can be translated to DPWs with a doubly-exponential blow-up, and the construction can be done on-the-fly, the above algorithm implies an EXPSPACE algorithm for calculating the probability of LTL formulas. As shown, however, in [5], it is possible to circumvent the translation into DPW and find the probability in PSPACE.

3 Classes of safety

In this section we define safety, safety levels, and four classes of safety levels. We also define the dual notion, of co-safety levels, and study the calculation of the safety level.

3.1 Safety, levels and classes

Consider a language $L \subseteq \Sigma^\omega$ of infinite words over an alphabet Σ. A finite word $x \in \Sigma^*$ is a *bad prefix* for L if for all infinite words $y \in \Sigma^\omega$, we have that $x \cdot y \notin L$. In other words, a bad prefix for L is a finite word that cannot be extended into an infinite word in L. Let $safe(L) =$

$\{w : w \text{ has a bad prefix for } L\}$ and $comp(L) = \Sigma^\omega \backslash L$. Note that $safe(L) \subseteq comp(L)$. We define the *safety level* of a language L, denoted $slevel(L)$, as the probability of a word not in L to have a bad prefix. Formally, $slevel(L) = Pr(safe(L) \mid comp(L))$. When $slevel(L) = p$, we say that L is a *p-safety language*. By the definition of conditional probability, we have that

$$slevel(L) = Pr(safe(L) \mid comp(L)) = \frac{Pr(safe(L) \cap comp(L))}{Pr(comp(L))} = \frac{Pr(safe(L))}{Pr(comp(L))}.$$

Note that if $Pr(L) = 1$, then $Pr(comp(L)) = 0$ so $slevel(L)$ is undefined. In this case, we define the safety level of L to be 0, as justified in Proposition 3.1 below.[3]

Proposition 3.1 *For every language L, if $Pr(L) = 1$, then $safe(L) = \emptyset$.*

Proof Consider a language L with $Pr(L) = 1$. Assume, by way of contradiction, that there is a bad prefix x of L. For every $y \in \Sigma^\omega$, we have that $x \cdot y \notin L$. It follows that $Pr(comp(L)) > 0$, contradicting the assumption that $Pr(L) = 1$. □

Remark 3.2 Note that the other direction of Proposition 3.1 does not hold. That is, there exists a language L such that L has no bad prefixes and still $Pr(L) \neq 1$. As an example, consider the language $L = [\![FGa]\!]$ with $AP = \{a\}$. Every finite prefix can be extended to a word in L by the suffix $\{a\}^\omega$, thus L has no bad prefixes. Nevertheless, it is easy to see that $Pr(L) = 0$.

We define four classes of languages, describing their safety level:

- *Safety* [2,29] A language $L \subseteq \Sigma^\omega$ is a safety language if $safe(L) = comp(L)$. That is, every word not in L has a bad prefix. For example, $L = [\![Ga]\!]$ is a safety language, as every word not in L has a prefix in which a does not hold, and this prefix cannot be extended to a word in L.
- *Almost-safety* A language $L \subseteq \Sigma^\omega$ is an almost-safety language if it is p-safety for $p = 1$ but is not safety. As an example, consider the language $L = [\![aUb]\!]$. The language L is not a safety language, as the word $a^\omega \notin L$ has no bad prefix. Indeed, we can concatenate b^ω to every prefix of a^ω and get a word in L. In addition, every word not in L except for a^ω has a bad prefix for L,and the set of these words have measure 1 Accordingly, $\frac{Pr(safe(L))}{Pr(comp(L))} = 1$.
- *Frac-safety* A language $L \subseteq \Sigma^\omega$ is a frac-safety language if it is p-safety for $0 < p < 1$. As an example, consider the language $L = [\![a \wedge FGa]\!]$. We show that L is $\frac{1}{2}$-safety. The words not in L are these that satisfy $\neg a \vee GF\neg a$. Since $Pr(GF\neg a) = 1$, we have that $Pr(comp(L)) = Pr(\neg a \vee GF\neg a) = 1$. Note that every prefix can be extended to a word that satisfies FGa, simply by concatenating the suffix a^ω. That is, words that do not satisfy FGa have no bad prefixes. On the other hand, every word that does not satisfy a has the bad prefix $\neg a$. It follows that a word has a bad prefix for L iff it does not satisfy a. Accordingly, $Pr(safe(L)) = Pr(\neg a) = \frac{1}{2}$. Hence, $\frac{Pr(safe(L))}{Pr(comp(L))} = \frac{\frac{1}{2}}{1} = \frac{1}{2}$, and L is $\frac{1}{2}$-safety.

[3] An anomaly of this definition is the language $L = \Sigma^\omega$. While $Pr(L) = 1$, making its safety level 0, we also have that L is a safety language. Thus, L is the only language that is both safety and has safety level 0.

- *Liveness* A language $L \subseteq \Sigma^\omega$ is a liveness language if $safe(L) = \emptyset$. For example, the language $L = [\![Fa]\!]$ is liveness, as no word in $comp(L)$ has a bad prefix. Indeed, the only word in $comp(L)$ is $(\neg a)^\omega$, and it has no bad prefixes. Note that the definition is equivalent to the one in [1], according to which L is liveness if every word in Σ^* can be extended to a word in L.

We extend the classification to LTL formulas. We say that a formula φ is *p-safety* if $[\![\varphi]\!]$ is p-safety, and that φ is in one of the classes above if $[\![\varphi]\!]$ is in the class.

Recall that an almost-safety language is a language that is 1-safety but not safety. Dually, we can relate to 0-safety languages that are not liveness. While, however, 1-safety is distinct from safety, Proposition 3.3 below states that 0-safety and liveness coincide.

Proposition 3.3 *A language is liveness iff it is* 0*-safety.*

Proof Consider a language $L \subseteq \Sigma^\omega$. By the definition of liveness, if L is liveness, then $safe(L) = \emptyset$, so it is 0-safety. For the other direction, we prove that if L is not liveness then it is not 0-safety. Assume that L is not liveness, thus $safe(L) \neq \emptyset$. Let $w \in safe(L)$ and let u be a bad prefix for w. Every word that starts with u is both in $comp(L)$ and in $safe(L)$. Therefore, the measure of words in $safe(L)$ is at least the measure of words with prefix u, which is strictly greater than 0. Hence, if L is not liveness then it is not 0-safety. □

3.2 Co-safety, levels and classes

We turn to study the dual notion, of *co-safety languages*. A finite word $x \in \Sigma^*$ is a *good prefix* for a language $L \subseteq \Sigma^\omega$, if for all infinite words $y \in \Sigma^\omega$, we have that $x \cdot y \in L$. In other words, a *good prefix* is a finite word all whose extentions are in L. A language L is a *co-safety language* if every word in L has a good prefix. Equivalently, L is co-safety iff $comp(L)$ is safety. We define $co\text{-}safe(L) = \{w : w \text{ has a good prefix for } L\}$. The classes defined above for safety can be dualized to classes on co-safety. Thus, the *co-safety level* of a language L, denoted $co\text{-}slevel(L)$, is $Pr(co\text{-}safe(L) \mid L)$. Also, a language L is *co-liveness* if $co\text{-}safe(L) = \emptyset$. Propositions 3.1 and 3.3 can be dualized too. Formally, we have the following.

Lemma 3.4 *For every language L, we have* $slevel(L) = co\text{-}slevel(comp(L))$.

Proof Consider a language L. By the definition of good and bad prefixes, every bad prefix for L is a good prefix for $comp(L)$, thus $Pr(safe(L)) = Pr(co\text{-}safe(comp(L)))$. Hence, $slevel(L) = Pr(safe(L) \mid comp(L)) = \frac{Pr(safe(L))}{Pr(comp(L))} = \frac{Pr(co\text{-}safe(comp(L)))}{Pr(comp(L))} = Pr(co\text{-}safe(comp(L)) \mid comp(L)) = co\text{-}slevel(comp(L))$, and we are done. □

Table 1 describes LTL formulas of different safety and co-safety levels. The number within brackets next to a formula φ is $Pr(\varphi)$. For example, the LTL formula $aUb \vee Gc$ is almost-safety, almost-co-safety, and has probability $\frac{2}{3}$.

Note that, by Lemma 3.4, a language L is a member of the (i, j)-th cell in the table iff $comp(L)$ is a member of the (j, i)-th cell. In addition, it follows from the definition of a safety level that if $Pr(\varphi) = 1$, then φ is 0-safety, so, by Proposition 3.3, the formula φ is in the right column. Dually, if $Pr(\varphi) = 0$, then φ is 0-co-safety, so, by the dual of Proposition 3.3, it is in the bottom row. Propositions 4.6 and 4.9 below state some additional properties of the table, and they are based on some observations on automata.

Table 1 Examples to formulas of different safety and co-safety levels, with $\varphi_1 = a \wedge GFb$, $\varphi_2 = c \wedge FGd$, and $\varphi_3 = \neg a \wedge FGb$

	Safety	Almost-safety	Frac-safety	Liveness
Co-safety	a $(\frac{1}{2})$	aUb $(\frac{2}{3})$	–	Fa (1)
		$a \wedge Fb$ $(\frac{1}{2})$	(see Lemma 4.9)	
Almost-co-safety	$\neg(aUb)$ $(\frac{1}{3})$	$aUb \vee Gc$ $(\frac{2}{3})$	$\neg\varphi_1 \wedge c$ $(\frac{1}{4})$	$\neg\varphi_1$ $(\frac{1}{2})$
	$a \vee Gb$ $(\frac{1}{2})$		$a \vee \varphi_2$ $(\frac{1}{2})$	$Ga \vee Fb$ (1)
Frac-co-safety	–	$\varphi_1 \vee c$ $(\frac{3}{4})$	$\varphi_1 \vee \varphi_2 \vee e$ $(\frac{3}{4})$	$\neg\varphi_2$ (1)
	(see Lemma 4.9)	$a \wedge \neg\varphi_2$ $(\frac{1}{2})$		$\neg\varphi_1 \wedge \neg\varphi_2$ $(\frac{1}{2})$
Co-liveness	Ga (0)	φ_1 $(\frac{1}{2})$	φ_2 (0)	FGa (0)
		$Ga \wedge Fb$ (0)	$\varphi_1 \vee \varphi_2$ $(\frac{1}{2})$	GFa (1)
				$\varphi_1 \vee \varphi_3$ $(\frac{1}{2})$

3.3 Finding the safety level

We now study the problem of calculating the the safety level of a language given by a DPW, an NBW, or an LTL formula. Note that the safety level of a language does not depend on the formalism in which it is given. The latter is going to affect only the complexity of the problem.

Theorem 3.5 *Calculating the safety level of a language L can be done in linear, exponential, and doubly-exponential time for L given by a DPW, an NBW, and an LTL formula, respectively.*

Proof We describe a linear-time algorithm for DPWs. The complexity for the other classes then follows from Theorem 2.1. Consider a DPW $\mathcal{A}$. We calculate $slevel(L(\mathcal{A}))$ by calculating $Pr(safe(L(\mathcal{A})))$, $Pr(comp(L(\mathcal{A})))$, and the ratio between them. Let $\mathcal{C} = \{C_1, \ldots, C_m\}$ be the ergodic SCCs of $\mathcal{A}$. We can find $\mathcal{C}$ in linear time, and can also mark the ergodic SCCs that are rejecting and these that are pure rejecting [31]. Recall that $safe(L(\mathcal{A}))$ includes all the words with bad prefix for $L(\mathcal{A})$, and note a path to an ergodic SCC C in $\mathcal{A}$ is labeled with a bad prefix for $L(\mathcal{A})$ iff C is pure rejecting. In addition, by Lemma 2.2, a path reaches some ergodic SCC with probability 1. Therefore, it is easy to see that $Pr(safe(L(\mathcal{A})))$ is the probability that a random run reaches a pure rejecting SCC, and $Pr(comp(L(\mathcal{A})))$ is the probability that a random run reaches a rejecting SCC. Both can be calculated by assigning to each SCC the probability that a random run reaches it. Finally, by Proposition 3.1, if $Pr(comp(L(\mathcal{A}))) = 0$, then the algorithm returns 0. □

Remark 3.6 While we assume a uniform probability distribution on Σ^ω, it is possible to extend our results to a setting in which the probability of a computation π is defined by a Markov chain, such that for each atomic proposition q and for each position in π, the probability that q holds in the position is in $(0, 1)$. This assumption may affect the exact safety level of a language, yet it is not hard to see that the class of safety level agrees with the one obtained for uniform distribution.

4 Finding the safety class

In this section we study the problem of classifying languages to the four classes of safety level, hoping to end up with algorithms that are simpler than the one described for finding the exact safety level. Deciding membership in the classes of safety and liveness have already been studied, and we focus on almost-safety and frac-safety. We first recall the known results for safety and liveness:

Theorem 4.1 [15,29] *Consider a language $L \subseteq \Sigma^\omega$.*

- *Deciding whether L is safety is NLOGSPACE-complete for L given by a DPW and PSPACE-complete for L given by an NBW or an LTL formula.*
- *Deciding whether L is liveness is NLOGSPACE-complete for L given by a DPW, PSPACE-complete for L given by an NBW, and EXPSPACE-complete for L given by an LTL formula.*

We did not find in the literature an NLOGSPACE-complete result for deciding liveness of a DPW. The proof, however, follows standard considerations, and we give it here for completeness.

Proposition 4.2 *Deciding whether L is liveness is NLOGSPACE-complete for L given by a DPW.*

Proof The upper bound follows from Theorem 4.5(1). We prove the lower bound by showing a reduction from the non-reachability problem, proven to be NLOGSPACE-hard in [13]. Given a graph $G = \langle V, E\rangle$ and two vertices u and v in V, we construct a DPW $\mathcal{A} = \langle \Sigma, Q, Q_0, \delta, \alpha\rangle$ such that $L(\mathcal{A})$ is liveness iff v is not reachable from u.

The DPW $\mathcal{A}$ is similar to G, with a new state and additional transitions from each state in $V\backslash\{v\}$ to the new state and from v to itself. See Fig. 1 for illustration. Intuitively, the new state is an accepting sink and v is a rejecting sink, so v is not reachable from u iff $safe(L(\mathcal{A})) = \emptyset$. Recall that by the definition of liveness, a language L is liveness iff $safe(L) = \emptyset$.

Formally, $\mathcal{A} = \langle E \cup \{e_{new}\}, V \cup \{q_{acc}\}, \{u\}, \delta, \{\emptyset, \{q_{acc}\}, V \cup \{q_{acc}\}\}\rangle$, where δ is such that for every edge $e = \langle w, w'\rangle \in E$ we have a transition $\langle w, e, w'\rangle$ in δ. That is, all the edges of the graph are transitions in $\mathcal{A}$, all labeled differently. In addition, δ has the transitions $\langle v, e, v\rangle$ and $\langle q_{acc}, e, q_{acc}\rangle$ for all $e \in E \cup \{e_{new}\}$, and the transitions $\langle w, e_{new}, q_{acc}\rangle$ for all $w \in V\backslash\{v\}$. Note that $\mathcal{A}$ is a DPW, as required, and that the reduction is computable using logarithmic space.

We now prove that $L(\mathcal{A})$ is liveness iff v is not reachable from u. Since v is a rejecting sink in $\mathcal{A}$, if v is reachable from u then the path from u to v is labeled by a bad prefix for $L(\mathcal{A})$. Recall that a language has a bad prefix iff it is not liveness, thus $L(\mathcal{A})$ is not liveness. For the other direction, assume that v is not reachable from u. Recall that u is the initial state, and every state but v has a transition to the accepting sink q. Thus, every prefix can be extended to a word in $L(\mathcal{A})$ by the suffix $\{e_{new}\}^\omega$, so $safe(L(\mathcal{A})) = \emptyset$, thus $L(\mathcal{A})$ is liveness. □

Theorem 4.1 hints that at least for one of the classes almost-safety and frac-safety, the classification for LTL formulas is going to be EXPSPACE-complete. We turn to study the problems in detail. We do this by analyzing the structure of deterministic automata for the language. As detailed in Sect. 4.2, the analysis leads to interesting results on the expressive power of deterministic automata beyond the classification to safety level. We first need some definitions and observations on automata.

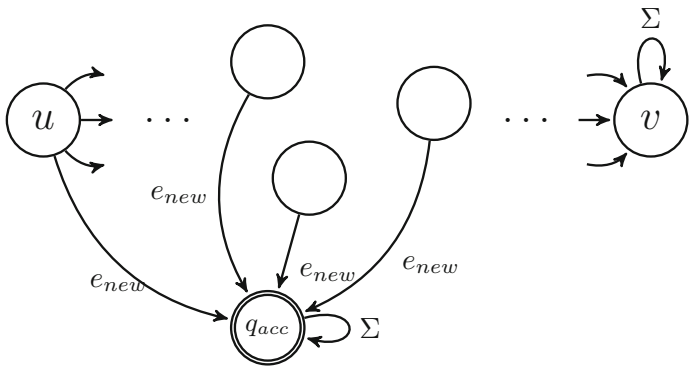

Fig. 1 The reduction from non-reachability to liveness decidability for DPWs

4.1 Observations on automata

Consider a deterministic automaton $\mathcal{A}$. Recall the partition of $\mathcal{A}$ to SCC as defined in Sect. 2. An ergodic SCC C of $\mathcal{A}$ is *mixed* if there exist both accepting and rejecting paths that reach C. Note that a mixed ergodic SCC of a DPW may be either accepting or rejecting, whereas a mixed ergodic SCC of a DBW is accepting. In terms of safety and co-safety, a run that reaches a pure accepting (rejecting) ergodic SCC is an accepting (rejecting) run and it is labeled by a word that has a good (bad, respectively) prefix. On the other hand, a run that reaches a mixed ergodic SCC is labeled by a word that has neither a good prefix nor a bad prefix. For an automaton $\mathcal{A}$ and an ergodic SCC C of $\mathcal{A}$, we define $reach(\mathcal{A}, C) = \{w : \text{the run of } \mathcal{A} \text{ on } w \text{ reaches } C\}$. For a language L, we define $mixed\text{-}in(L) = L \backslash co\text{-}safe(L)$. That is, $mixed\text{-}in(L)$ contains all words w such that $w \in L$ and w has no good prefix. Dually, we define $mixed\text{-}out(L) = comp(L) \backslash safe(L)$. That is, $mixed\text{-}out(L)$ contains all words w such that $w \notin L$ and w has no bad prefix.

Our goal is to characterize classes of safety level by the structure of deterministic automata that recognize them. Lemmas 4.3 and 4.4 below state relevant observations on the ergodic SCC of automata of languages in the different classes.

Lemma 4.3 *Consider a deterministic automaton $\mathcal{A}$, and let $L = L(\mathcal{A})$.*

1. *$safe(L) \neq \emptyset$ iff $\mathcal{A}$ has a pure rejecting ergodic SCC. Dually, $co\text{-}safe(L) \neq \emptyset$ iff $\mathcal{A}$ has a pure accepting ergodic SCC.*
2. *Consider a mixed ergodic SCC C of $\mathcal{A}$. Let $R_{in} = reach(\mathcal{A}, C) \cap mixed\text{-}in(L)$ and $R_{out} = reach(\mathcal{A}, C) \cap mixed\text{-}out(L)$. Then, R_{in} and R_{out} form a partition of $reach(\mathcal{A}, C)$ with $R_{in} \neq \emptyset$ and $R_{out} \neq \emptyset$. If C is rejecting, then $Pr(R_{in}) = 0$ and $Pr(R_{out}) > 0$. Dually, if C is accepting, then $Pr(R_{in}) > 0$ and $Pr(R_{out}) = 0$.*

Proof Each of the clauses in the proposition consist of a claim and its dual. For both clauses, we prove only the claim. The proof for the dual of the claim is dual.

1. First, if $\mathcal{A}$ has a pure rejecting ergodic SCC C, then clearly, there is a bad prefix for L, thus $safe(L) \neq \emptyset$. Indeed, every word that reaches C is rejected, thus every path from an initial state to a state of C is a bad prefix. For the other direction, assume that $safe(L) \neq \emptyset$ and let $x \in \Sigma^*$ be a bad prefix for L. Recall that a path that reaches a mixed ergodic SCC is labeled by a word that has no bad prefix. It follows that for every $y \in \Sigma^\omega$, the run of

$\mathcal{A}$ on $x \cdot y$ reaches a pure rejecting ergodic SCC. Thus, if $safe(L) \neq \emptyset$ then $\mathcal{A}$ has a pure rejecting ergodic SCC.

2. Let C be a mixed ergodic SCC. By the definition of mixed ergodic SCC, there are both accepting and rejecting paths that reach C. In addition, a path that reaches a mixed ergodic SCC is labeled by a word that is neither in $co\text{-}safe(L)$ nor in $safe(L)$. Therefore, $reach(\mathcal{A}, C)$ can be partitioned into $R_{in} = reach(\mathcal{A}, C) \cap mixed\text{-}in(L)$ and $R_{out} = reach(\mathcal{A}, C) \cap mixed\text{-}out(L)$ such that $R_{in} \neq \emptyset$ and $R_{out} \neq \emptyset$. Assume that C is rejecting. Then, by the definition of rejecting ergodic SCC, a random path that reaches C is rejecting with probability 1 and accepting with probability 0. That is, $Pr(R_{in}) = 0$, $Pr(R_{out}) > 0$ and we are done. □

Lemma 4.4 *Consider a deterministic automaton $\mathcal{A}$. If $Pr(mixed\text{-}out(L(\mathcal{A}))) = 0$ and $mixed\text{-}out(L(\mathcal{A})) \neq \emptyset$, then at least one of the following conditions holds:*

1. *$\mathcal{A}$ has a mixed accepting ergodic SCC.*
2. *There is a rejecting run of $\mathcal{A}$ that does not reach an ergodic SCC.*

Proof Consider a deterministic automaton $\mathcal{A}$ and its language $L = L(\mathcal{A})$. Assume that $Pr(mixed\text{-}out(L)) = 0$, and consider a word $w \in mixed\text{-}out(L)$. By definition, w does not have a bad prefix. If for some ergodic SCC C it holds that $w \in reach(\mathcal{A}, C)$, then, by Lemma 4.3(2), the SCC C is mixed accepting, so the first condition holds. Also, if there is no ergodic SCC C such that $w \in reach(\mathcal{A}, C)$, then the second condition holds. Indeed, since $w \in mixed\text{-}out(L)$ and since $w \notin reach(\mathcal{A}, C)$ for every ergodic SCC C, we have that the run of $\mathcal{A}$ on w is a rejecting run of $\mathcal{A}$ that does not reach an ergodic SCC. That is, if $Pr(mixed\text{-}out(L)) = 0$ and $mixed\text{-}out(L) \neq \emptyset$, then at least one of the mentioned conditions holds. □

We can now reduce questions about the class of a given language to questions about the structure of a deterministic automaton for it. We assume that automata have at most one rejecting sink. Indeed, multiple rejecting sinks may be merged. We refer to rejecting sinks as *empty states*. Similarly, we assume that automata have at most one accepting sink, and refer to accepting sinks as *universal states*.

Theorem 4.5 *Consider a deterministic automaton $\mathcal{A}$ with at most one empty state, and let $L = L(\mathcal{A})$.*

1. *The language L is liveness iff $\mathcal{A}$ does not have a pure rejecting ergodic SCC.*
2. *The language L is frac-safety iff $\mathcal{A}$ has a pure rejecting ergodic SCC and a mixed rejecting ergodic SCC.*
3. *The language L is almost-safety iff $\mathcal{A}$ has a pure rejecting ergodic SCC, does not have a mixed rejecting ergodic SCC, and has either a mixed accepting ergodic SCC or a rejecting path that does not reach a pure rejecting ergodic SCC.*
4. *The language L is safety iff $\mathcal{A}$ has a pure rejecting ergodic SCC and all its rejecting paths reach a pure rejecting ergodic SCC.*

Proof Consider a deterministic automaton $\mathcal{A}$ with at most one empty state, and let $L = L(\mathcal{A})$.

1. By Lemma 4.3(1), the automaton $\mathcal{A}$ has a pure rejecting ergodic SCC iff $safe(L) \neq \emptyset$. Recall that L is liveness iff it has no bad prefixes. It follows that L is liveness iff $\mathcal{A}$ does not have pure rejecting ergodic SCCs.
2. For the first direction, assume that L is frac-safety. By the definition of frac-safety languages, we have that $safe(L) \neq \emptyset$. According to Lemma 4.3(1), it follows that $\mathcal{A}$

has a pure rejecting ergodic SCC. In addition, for a frac-safety language L it holds that $Pr(\textit{mixed-out}(L)) > 0$. Since an ergodic SCC may be either pure or mixed and either accepting or rejecting, it follows from Lemma 4.3(2) that a positive measure of words in $\textit{mixed-out}(L)$ can be induced only by a mixed rejecting ergodic SCC. Therefore, $\mathcal{A}$ has a mixed rejecting ergodic SCC, so we are done. For the other direction, assume that $\mathcal{A}$ has both pure rejecting and mixed rejecting ergodic SCCs. By Lemma 4.3, it holds that $\textit{safe}(L) \neq \emptyset$ and $Pr(\textit{mixed-out}(L)) > 0$. Thus, we have that $0 < Pr(\textit{safe}(L) \mid \textit{comp}(L)) < 1$. In addition, since $\textit{safe}(L) \neq \emptyset$, we have that $Pr(L) < 1$. Recall that for a language L with $Pr(L) \neq 1$, the level of safety is defined as $Pr(\textit{safe}(L) \mid \textit{comp}(L))$. Therefore, we have that the level of safety of L is greater than 0 and less than 1, so L is frac-safety, and we are done.

3. For the first direction, assume that L is almost-safety. Since L is almost-safety, we have that $Pr(\textit{mixed-out}(L)) = 0$ and $\textit{mixed-out}(L) \neq \emptyset$. By Lemma 4.4, the automaton $\mathcal{A}$ has either a mixed accepting ergodic SCC or a rejecting path that does not reach a pure rejecting ergodic SCC. By Lemma 4.3(2), $\mathcal{A}$ does not have mixed rejecting ergodic SCC. In addition, an almost-safety language has a bad prefix. Therefore, by Lemma 4.3(1), the automaton $\mathcal{A}$ has a pure rejecting ergodic SCC, and we are done. For the other direction, assume that $\mathcal{A}$ has a pure rejecting ergodic SCC, does not have a mixed rejecting ergodic SCC, and has either a mixed accepting ergodic SCC or a rejecting path that does not reach a pure rejecting ergodic SCC. From the first two conditions and Lemma 4.3 it follows that $Pr(\textit{safe}(L) \mid \textit{comp}(L)) = 1$. From the last condition and the assumption that $\mathcal{A}$ includes at most one empty state, it follows that $\textit{mixed-out}(L) \neq \emptyset$, and therefore L is not safety. Then, by the definition of almost-safety languages, the language L is almost-safety.
4. For the first direction, assume that L is safety. Since every word in $\textit{comp}(L)$ has a bad prefix, the automaton $\mathcal{A}$ does not have mixed ergodic SCCs. Since $\textit{safe}(L) \neq \emptyset$, by Lemma 4.3(1) $\mathcal{A}$ has a pure rejecting ergodic SCC. In other words, if L is safety then $\mathcal{A}$ has a pure rejecting ergodic SCC and all its rejecting paths reach a pure rejecting ergodic SCC. For the other direction, assume that $\mathcal{A}$ has a pure rejecting ergodic SCC and all its rejecting paths reach a pure rejecting ergodic SCC. It follows that every word in $\textit{comp}(L)$ has a bad prefix, thus L is safety. □

4.2 Expressive power

Before we study the decision procedure that follow from Theorem 4.5, let us point to some interesting conclusions regarding expressive power that follow from the theorem.

Consider a language L. By the dual of Remark 3.2, co-liveness of a language L is not a sufficient condition for $Pr(L) = 0$. It is easy to see that also safety is not a sufficient condition for $Pr(L) = 0$. For example, the language $[\![a]\!]$ is safety and $Pr(a) = \frac{1}{2}$. Proposition 4.6 below shows that the combination of co-liveness and safety is sufficient.

Proposition 4.6 *If a language L is safety and co-liveness, then $Pr(L) = 0$.*

Proof Consider a deterministic automaton $\mathcal{A}$ for a safety and co-liveness language L. According to Theorem 4.5(4), since L is safety, the automaton $\mathcal{A}$ does not have mixed ergodic SCCs and has a pure rejecting ergodic SCC. According to the dual (for co-liveness) of Theorem 4.5(1), since L is co-liveness, the automaton $\mathcal{A}$ does not have pure accepting ergodic SCCs. Hence, $\mathcal{A}$ has only rejecting ergodic SCCs, so $Pr(L) = 0$. □

It is known that safety languages can be recognized by looping automata (that is, Büchi automata in which all states are accepting). Propositions 4.7 and 4.8 below relate the other

classes with recognizability by deterministic Büchi automata, which are known to be less expressive than deterministic parity automata [20].

Proposition 4.7 *The language of a DBW is safety, almost-safety, or liveness.*

Proof Consider a DBW $\mathcal{A}$. By Lemma 2.2, if a path in $\mathcal{A}$ reaches an ergodic SCC C, it visits all states in C infinitely often with probability 1. Therefore, a mixed ergodic SCC of a DBW is accepting. On the other hand, by Theorem 4.5(2), a DBW of a frac-safety language has a mixed rejecting ergodic SCC. □

Note that the other direction of Proposition 4.7 does not hold for liveness languages. That is, a liveness language may not have a DBW. For example, the language $[\![FGa]\!]$ is liveness, but it does not have a DBW [20]. Proposition 4.8 states that the other direction of Proposition 4.7 does hold for safety and almost-safety languages.

Proposition 4.8 *If a language is safety or almost-safety, then it can be recognized by a DBW.*

Proof Consider a DPW $\mathcal{A}$ for a safety or almost-safety language. Let $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_k\}$. By Theorems 4.5(3) and 4.5(4), the DPW $\mathcal{A}$ does not have a mixed rejecting ergodic SCC. Recall that an ergodic SCC of a DBW is either pure accepting, pure rejecting, or mixed accepting. Let $\alpha' = \bigcup \alpha_i : i$ is even. Since $\mathcal{A}$ does not have a mixed rejecting ergodic SCC, the DBW obtained from $\mathcal{A}$ by defining the acceptance condition to be α' is equivalent to $\mathcal{A}$. Hence, L can be recognized by a DBW and we are done. □

Recall that in Table 1, we left some intersections empty. We can now prove that the corresponding combinations are indeed impossible.

Proposition 4.9 *A language cannot be both safety and frac-co-safety or both frac-safety and co-safety.*

Proof We prove that a language L cannot be both safety and frac-co-safety. The proof of the second claim is dual. Let L be a frac-co-safety language, and let $\mathcal{A}$ be a DPW for L. As follows from the dual of Theorem 4.5(2), the DPW $\mathcal{A}$ has a mixed accepting ergodic SCC. Since a mixed accepting ergodic SCC induces at least one word in *mixed-out*(L), it follows that L is not safety, and we are done. □

4.3 Decision procedures

We now use Theorem 4.5 in order to find safety classes. Essentially, the characterization there reduces the problem to a search for "witness SCCs" or "witness paths". We elaborate on the algorithms for the specific cases.

Theorem 4.10 *Deciding whether a language $L \subseteq \Sigma^\omega$ is almost-safety is NLOGSPACE-complete for L given by a DPW and PSPACE-complete for L given by an NBW or an LTL formula.*

Proof We start with the upper bounds and show that the problem is in NLOGSPACE for DPWs. The upper bound for NBWs then follow from Theorem 2.1. Consider a DPW $\mathcal{A}$. By Theorem 4.5(3), we have that $L(\mathcal{A})$ is almost-safety iff $\mathcal{A}$ has a pure rejecting ergodic SCC, does not have a mixed rejecting ergodic SCC, and has either a mixed accepting ergodic SCC or a rejecting path that does not reach a pure rejecting ergodic SCC. Since NLOGSPACE is closed under complementation, and since we can verify the classification of a given

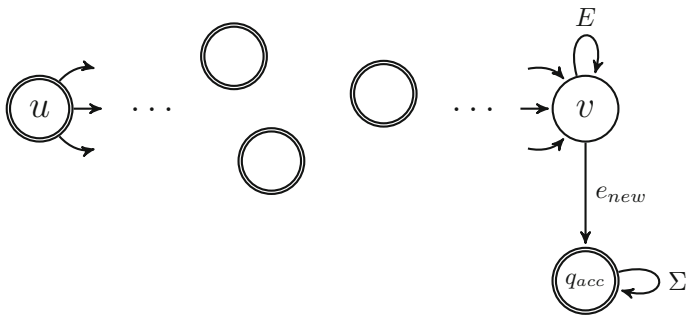

Fig. 2 The reduction from reachability to almost-safety decidability for DBWs

SCC in NLOGSPACE, checking whether $\mathcal{A}$ satisfies the condition above can be done in NLOGSPACE.

We turn to an upper bound for LTL. Note that a naive application of Theorems 4.5(3) and 2.1 only gives an EXPSPACE upper bound. Consider an LTL formula φ. First, we construct an NBW $\mathcal{A}_\varphi$ for φ and remove from it empty states (that is, states from which no word is accepted). We then construct an NBW $\mathcal{A}_\varphi^{loop}$ by making all of the states of $\mathcal{A}_\varphi$ accepting. That is, the NBW $\mathcal{A}_\varphi^{loop}$ has the same structure as $\mathcal{A}_\varphi$, and all of its states are in α. Note that $\mathcal{A}_\varphi^{loop}$ accepts the words all whose prefixes can be extended to a word in $[\![\varphi]\!]$. It is easy to see that for a safety language L, these words are exactly the words in L. As argued in [29], the language $[\![\varphi]\!]$ is safety iff $L(\mathcal{A}_\varphi^{loop}) = L(\mathcal{A}_\varphi)$. For almost-safety languages, this equality holds with probability 1. That is, $[\![\varphi]\!]$ is almost-safety iff it is not safety and $Pr(L(\mathcal{A}_\varphi^{loop}) \cap L(\mathcal{A}_{\neg\varphi})) = 0$. Indeed, every word without a bad prefix is accepted both by $\mathcal{A}_\varphi^{loop}$ and $\mathcal{A}_{\neg\varphi}$. Therefore, $[\![\varphi]\!]$ is almost-safety iff the set $L(\mathcal{A}_\varphi^{loop}) \cap L(\mathcal{A}_{\neg\varphi})$ is of measure 0. Accordingly, our algorithm constructs the NBWs $\mathcal{A}_\varphi^{loop}$ and $\mathcal{A}_{\neg\varphi}$, and then check whether $Pr(L(\mathcal{A}_\varphi^{loop}) \cap L(\mathcal{A}_{\neg\varphi})) = 0$. The latter can be done in NLOGSPACE($|\mathcal{A}_\varphi|$), which is PSPACE($|\varphi|$), as required. Indeed, we only have to check that the product automaton does not have an accepting SCC.

We turn to the lower bounds, and start with DPWs. In fact, we show that the problem is NLOGSPACE-hard already for DBWs. We describe a reduction from the reachability problem, proven to be NLOGSPACE-hard in [13]. Given a graph G and two vertices u and v in V, we construct a DBW $\mathcal{A}$ such that $L(\mathcal{A})$ is almost-safety iff v is reachable from u. Intuitively, the DBW $\mathcal{A}$ adds to G self loop in v and transitions to an accepting and a rejecting sink in such a way so that if v is not reachable from u, the language of $\mathcal{A}$ is safety. If, however, v is reachable from u, an infinite path that loops forever in v makes its language almost-safe.

The DBW $\mathcal{A}$ is similar to G, with a new state and some additional transitions (see an illustration in Fig. 2. Formally, $\mathcal{A} = \langle E \cup \{e_{new}\}, V \cup \{q_{acc}\}, \{u\}, \delta, \{(V \setminus \{v\}) \cup \{q_{acc}\}\rangle$. Intuitively, all the states in V except for v are accepting, and so is q_{acc}. The transition function δ is such that for every edge $e = \langle w, w'\rangle \in E$ we have a transition $\langle w, e, w'\rangle$ in δ. That is, all the edges of the graph are transitions in $\mathcal{A}$, all labeled differently. In addition, δ has the transition $\langle v, e_{new}, q_{acc}\rangle$, the self-loop $\langle v, e, v\rangle$ for all $e \in E$, and the self loop $\langle q_{acc}, e, q_{acc}\rangle$ for all $e \in E \cup \{e_{new}\}$. Note that $\mathcal{A}$ is a DBW, as required, and that the reduction is computable using logarithmic space.

We prove that v is reachable from u iff $L(\mathcal{A})$ is almost-safety. If v is not reachable from u then the language $L(\mathcal{A})$ contains exactly all words whose run do not get stuck. That is, they

do not contain e_{new} and successive edges in the word are successive in the graph. This is a safety language. Thus, if v is not reachable from u then $L(\mathcal{A})$ is not almost-safety. For the other direction, assume that v is reachable from u. Therefore, the run that reaches v from u and stays in v is a rejecting run that does not reach any ergodic SCC. A run that gets stuck is equivalent to a run that moves to a rejecting sink. Thus, $\mathcal{A}$ having runs that get stuck implies that $\mathcal{A}$ includes a pure rejecting ergodic SCC. In addition, $\mathcal{A}$ does not have mixed rejecting ergodic SCCs. It follows from Theorem 4.5(3) that $L(\mathcal{A})$ is almost-safety.

For NBWs, we again show a reduction from the non-universality problem for safety NBWs, namely the problem of deciding, given an NBW $\mathcal{A}$ for a safety language, whether $L(\mathcal{A}) \neq \Sigma^\omega$. The latter is proven to be PSPACE-hard in [24]. Given an NBW $\mathcal{A}$ for a safety language, we define an NBW $\mathcal{B}$ such that $\mathcal{A}$ is not universal iff $L(\mathcal{B})$ is almost-safety. Let $\mathcal{A} = \langle \Sigma, Q, Q_0, \delta, \alpha \rangle$, and let $\mathcal{A}' = \langle \{a, b\}, Q', Q'_0, \delta', \alpha' \rangle$ be an NBW for the language $[\![aUb]\!]$. We define $\mathcal{B} = \langle \Sigma \times \{a, b\}, Q \times Q', Q_0 \times Q'_0, \delta'', \alpha'' \rangle$, where

- $\delta''(\langle q, q' \rangle, \langle \sigma, \sigma' \rangle) = \langle \delta(q, \sigma), \delta'(q', \sigma') \rangle$, and
- $\alpha'' = \{\langle Q, Q' \rangle : Q \in \alpha\} \cup \{\langle Q, Q' \rangle : Q' \in \alpha'\}$.

For a word $w \in (\Sigma \times \{a, b\})^\omega$, let $w_1 \in \Sigma^\omega$ be the word obtained from w by projecting its letters on Σ, and similarly for w_2 and $\{a, b\}$. It is easy to see that $L(\mathcal{B}) = \{w : w_1 \in L(\mathcal{A}) \text{ or } w_2 \in [\![aUb]\!]\}$. We prove that $\mathcal{A}$ is not universal iff $L(\mathcal{B})$ is almost-safety. First, if $\mathcal{A}$ is universal, then so is $\mathcal{B}$, thus $L(\mathcal{B})$ is not almost-safety. Assume now that $\mathcal{A}$ is not universal, we show that $L(\mathcal{B})$ is almost-safety. By the definition of $\mathcal{B}$, $comp(L(\mathcal{B})) = \{w : w_1 \in comp(L(\mathcal{A})) \text{ and } w_2 \in comp(L(\mathcal{A}'))\}$. In addition, $safe(L(\mathcal{B})) = \{w : w_1 \in safe(L(\mathcal{A})) \text{ and } w_2 \in safe(L(\mathcal{A}'))\}$. Recall that $L(\mathcal{A})$ is safety, thus every word in $comp(L(\mathcal{A}))$ has a bad prefix for $L(\mathcal{A})$. It follows that a word $w \in comp(L(\mathcal{B}))$ has a bad prefix for $L(\mathcal{B})$ iff w_2 has a bad prefix for $L(\mathcal{A}')$. That is, $Pr(safe(L(\mathcal{B}))) = Pr(safe(L(\mathcal{A}')))$. Since $L(\mathcal{A}')$ is almost-safety, we have that $\frac{Pr(safe(L(\mathcal{A}')))}{Pr(comp(L(\mathcal{A}')))} = 1$. It follows that

$$\frac{Pr(safe(L(\mathcal{B})))}{Pr(comp(L(\mathcal{B})))} = \frac{Pr(safe(L(\mathcal{A}')))}{Pr(comp(L(\mathcal{B})))} \geq \frac{Pr(safe(L(\mathcal{A}')))}{Pr(comp(L(\mathcal{A}')))} = 1.$$

That is, the safety level of $L(\mathcal{B})$ is 1. We show that $L(\mathcal{B})$ is almost-safety. Since $\mathcal{A}$ is not universal, there is a word x_1 in $comp(L(\mathcal{A}))$. In addition, since $L(\mathcal{A}')$ is almost-safety, there is a word x_2 in $comp(L(\mathcal{A}'))$ that does not have a bad prefix for $L(\mathcal{A}')$. The word $w \in (\Sigma \times \{a, b\})^\omega$ with $w_1 = x_1$ and $w_2 = x_2$ is in $comp(L(\mathcal{B}))\backslash safe(L(\mathcal{B}))$. That is, $L(\mathcal{B})$ is almost-safety. Concluding, we have that $\mathcal{A}$ is not universal iff $\mathcal{B}$ is almost-safety.

It is left to prove PSPACE-hardness the LTL setting. We do this by a reduction from the non-validity problem for safety LTL formulas, proven to be PSPACE-hard in [30]. Given a safety LTL formula φ over AP, let a and b be atomic propositions not in AP. We prove that φ is not valid iff $\varphi \vee (aUb)$ is almost-safety. Assume first that φ is valid. Then, $\varphi \vee aUb$ is valid and therefore is not almost-safety. Assume now that φ is not valid, we show that $\varphi \vee aUb$ is almost-safety. Since φ is safety, we have that a word is in $comp(L(\varphi \vee aUb))\backslash safe(L(\varphi \vee aUb))$ iff its projection on $\{a, b\}$ is in $comp(L(aUb))\backslash safe(L(aUb))$. Note that $comp(L(aUb))\backslash safe(L(aUb)) = \{a^\omega\}$. It follows that $Pr(comp(L(\varphi \vee aUb))\backslash safe(L(\varphi \vee aUb))) = 0$ and $comp(L(\varphi \vee aUb))\backslash safe(L(\varphi \vee aUb)) \neq \emptyset$. That is, $\varphi \vee aUb$ is almost-safety, and we are done. □

Theorem 4.11 *Deciding whether a language $L \subseteq \Sigma^\omega$ is frac-safety is NLOGSPACE-complete for L given by a DPW, PSPACE-complete for L given by an NBW, and EXPSPACE-complete for L given by an LTL formula.*

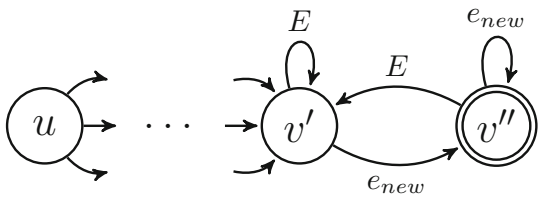

Fig. 3 The reduction from reachability to frac-safety decidability for DPWs

Proof We start with the upper bounds and show that the problem is in NLOGSPACE for DPWs. The upper bounds for NBWs and LTL formulas then follow from Theorem 2.1. Consider a DPW $\mathcal{A}$. By Theorem 4.5(2), we have that $L(\mathcal{A})$ is frac-safety iff $\mathcal{A}$ has a pure rejecting ergodic SCC and a mixed rejecting ergodic SCC. Since we can verify the classification of a given SCC in NLOGSPACE, checking whether $\mathcal{A}$ satisfies the condition above can be done in NLOGSPACE.

We proceed to the lower bounds. For DPWs, we describe a reduction from the reachability problem. Given a graph $G = \langle V, E\rangle$ and two vertices u and v in V, we construct a DPW $\mathcal{A}$ such that $L(\mathcal{A})$ is frac-safety iff v is reachable from u. Intuitively, the DPW is constructed in the following way. A new rejecting sink is added, with a transition from u. In addition, v is replaced by a mixed rejecting component. We get a DPW with a reachable pure rejecting ergodic SCC. In addition, it has a mixed rejecting ergodic SCC which is reachable iff v is reachable from u. Recall that by Theorem 4.5(2), a language is frac-safety iff its automaton has a pure rejecting ergodic SCC and a mixed rejecting ergodic SCC. See an illustration in Fig. 3.

Formally, $\mathcal{A} = \langle E \cup \{e_{new}\}, (V \backslash \{v\}) \cup \{q, v', v''\}, u, \delta, \{\{v'\}, \{v', v''\}, Q\}\rangle$, where δ is such that for every edge $e = \langle w, w'\rangle \in E$ with $w' \neq v$ we have a transition $\langle w, e, w'\rangle$ in δ. For every edge $e = \langle w, v\rangle \in E$ we have in δ a transition $\langle w, e, v'\rangle$. That is, all the edges of the graph are transitions in $\mathcal{A}$, all labeled differently, and v is replaced by v'. In addition, δ has the transitions $\langle v', e_{new}, v''\rangle$, $\langle v'', e_{new}, v''\rangle$, $\langle v', e, v'\rangle$ for all $e \in E$, and $\langle v'', e, v'\rangle$ for all $e \in E$. In other words, the acceptance condition requires a finite number of visits in v', and an infinite number of visits in v''. Note that $\mathcal{A}$ is a DPW, as required, and that the reduction is computable using logarithmic space.

We prove that v is reachable from u iff $L(\mathcal{A})$ is frac-safety. For the first direction, assume that v is reachable from u. We have that $\mathcal{A}$ has both a reachable mixed rejecting ergodic SCC, namely $\{v', v''\}$, and an implicit pure rejecting ergodic SCC to which runs "move" when they get stuck. By Theorem 4.5(2), it follows that $L(\mathcal{A})$ is frac-safety. For the other direction, assume that v is not reachable from u, thus $\mathcal{A}$ does not have a reachable mixed rejecting ergodic SCC. It follows from Theorem 4.5(2) that $L(\mathcal{A})$ is not frac-safety.

We turn to prove that the problem is PSPACE-hard for NBWs, again by a reduction from the non-universality problem for safety NBWs. Let $\mathcal{A}$ be an NBW for a safety language. The reduction is similar to the one for NBWs in the proof of Theorem 4.10, except that now we take $\mathcal{A}'$ to be an NBW for the language $[\![a \wedge FGb]\!]$, which is a $\frac{1}{2}$-safety language. Consequently, $L(\mathcal{B}) = \{w : w_1 \in L(\mathcal{A} \text{ or } w_2 \in [\![a \wedge FGb]\!]\}$.

We prove that $\mathcal{A}$ is not universal iff $L(\mathcal{B})$ is frac-safety. First, if $\mathcal{A}$ is universal, then so is $\mathcal{B}$, so $L(\mathcal{B})$ is not frac-safety. Assume now that $\mathcal{A}$ is not universal, we show that $L(\mathcal{B})$ is frac-safety. In order to find the safety level of $L(\mathcal{B})$, we wish to compute $Pr(safe(L(\mathcal{B})))$ and $Pr(comp(L(\mathcal{B})))$. By the definition of $\mathcal{B}$, $comp(L(\mathcal{B})) = \{w : w_1 \in comp(L(\mathcal{A}))$ and $w_2 \in comp(L(\mathcal{A}'))\}$. Since w_1 and w_2 are independent, we have that $Pr(comp(L(\mathcal{B}))) = Pr(comp(L(\mathcal{A}))) \cdot Pr(comp(L(\mathcal{A}')))$. In addition, $safe(L(\mathcal{B})) = \{w :$

$w_1 \in safe(L(\mathcal{A}))$ and $w_2 \in safe(L(\mathcal{A}'))\}$. That is, a prefix is a bad prefix for $L(\mathcal{B})$ iff it is a bad prefix both for $L(\mathcal{A})$ and for $L(\mathcal{A}')$. With the same reasoning as for $comp(L(\mathcal{B}))$, we have that $Pr(safe(L(\mathcal{B}))) = Pr(safe(L(\mathcal{A}))) \cdot Pr(safe(L(\mathcal{A}')))$. Recall that $L(\mathcal{A})$ is safety, thus $\frac{Pr(safe(L(\mathcal{A})))}{Pr(comp(L(\mathcal{A})))} = 1$. In addition, $\mathcal{A}$ is not universal, thus $Pr(comp(L(\mathcal{A}))) > 0$. Therefore, the safety level of $L(\mathcal{B})$ is

$$\frac{Pr(safe(L(\mathcal{B})))}{Pr(comp(L(\mathcal{B})))} = \frac{Pr(safe(L(\mathcal{A}))) \cdot Pr(safe(L(\mathcal{A}')))}{Pr(comp(L(\mathcal{A}))) \cdot Pr(comp(L(\mathcal{A}')))} = 1 \cdot \frac{Pr(safe(L(\mathcal{A}')))}{Pr(comp(L(\mathcal{A}')))} = \frac{1}{2}.$$

That is, the safety level of $L(\mathcal{B})$ is equal to the safety level of $L(\mathcal{A}')$. Since $L(\mathcal{A}')$ is frac-safety, we have that $L(\mathcal{B})$ is frac-safety too, and we are done.

It is left to prove that the problem is EXPSPACE-hard for LTL formulas. We show a reduction from the problem of deciding non-liveness of LTL formulas, which is EXPSPACE-hard [15]. Given an LTL formula ψ over AP, we construct an LTL formula φ such that ψ is not liveness iff φ is frac-safety.

The construction of φ is as follows. Let a, b, and c be propositions not in AP. We define $\varphi = ((a \wedge b) \rightarrow \psi) \wedge ((\neg a \vee \neg b) \rightarrow FGc)$. We prove that ψ is not liveness iff φ is frac-safety. First, if ψ is liveness then φ is liveness, since nor ψ neither FGc have a bad prefix. Therefore, φ is not frac-safety. That is, if ψ is liveness then φ is not frac-safety. For the other direction, assume that ψ is not liveness. Then, $0 < Pr(safe([\![\psi]\!])) \leq 1$. Note that if a computation for φ starts with $\{a, b\}$, then the computation has a bad prefix with probability $Pr(safe([\![\psi]\!]))$. Otherwise, the computation has a bad prefix with probability $Pr(safe([\![FGp]\!]))$, which is 0. That is, $Pr(safe([\![\varphi]\!])) = \frac{Pr(safe([\![\psi]\!]))}{4}$, so $0 < Pr(safe([\![\varphi]\!])) \leq \frac{1}{4}$. In order to find the safety level of φ, it is left to find $Pr(comp([\![\varphi]\!]))$, which is equal to $Pr(\neg\varphi)$. If a computation for φ starts with $\{a, b\}$, then the computation satisfies φ with probability $Pr(\psi)$. Otherwise, the computation satisfies φ with probability $Pr(FGc)$, which is 0. That is, $Pr(\varphi) = \frac{Pr(\psi)}{4}$, so $0 \leq Pr(\varphi) \leq \frac{1}{4}$. Therefore, $\frac{3}{4} \leq Pr(\neg\varphi) \leq 1$. Since $0 < Pr(safe([\![\varphi]\!])) \leq \frac{1}{4}$, we have that $0 < \frac{Pr(safe([\![\varphi]\!]))}{Pr(\neg\varphi)} \leq \frac{1}{3}$. Note that the safety level of $L(\varphi)$ is equal to $\frac{Pr(safe([\![\varphi]\!])}{Pr(\neg\varphi)}$, thus φ is frac-safety, and we are done. □

We note that the problem of deciding frac-safety for DBWs is in $O(1)$, as, by Proposition 4.7, the language of all DBWs is not frac-safety.

To conclude, the complexities of deciding almost-safety and safety coincide, and so do the complexities of deciding frac-safety and liveness. In the case of LTL formulas, the difference in the complexity of deciding safety and liveness is carried over to a difference in deciding almost-safety and frac-safety, and the latter is exponentially more expensive. Intuitively, it follows from the structural similarity between safety and almost-safety – both search a word with no bad prefix, which can be done in the nondeterministic automaton, and between frac-safety and liveness – both search for a word that cannot be extended to a word in the language, which should be done in the deterministic automaton for the language.

5 Classes of clopen

Consider a language $L \subseteq \Sigma^\omega$ of infinite words over an alphabet Σ. A finite word in Σ^* is a *determined prefix* for L if it is either a bad or a good prefix for L. Note that if x is a determinized prefix, then so are all its extensions. For $k \geq 0$, we say that L is *k-bounded* if every word in Σ^k is a determined prefix for L. In other words, a language L is k-bounded if

for every word $w \in \Sigma^\omega$, the mempership of w in L can be determined by observing only the prefix of length k of w. We say that a language is *bounded* if it is k-bounded for some $k \geq 0$.

For $k \geq 0$, let $determined_k(L) = \{w : w \text{ has a determined prefix for } L \text{ of length } k\}$. We define the *k-bounding level* of a language L, denoted $blevel_k(L)$, as the probability of a word in Σ^ω to have a determined prefix for L of length k. That is, $blevel_k(L) = Pr(determined_k(L))$. Note that, equivalently, the k-bounding level of a language L is the probability that a random word in Σ^k is a determined prefix for L. We define the *bounding level* of a language L, denoted $blevel(L)$, as $\lim_{k\to\infty} blevel_k(L)$. Since every extension of a determined prefix in Σ^k is a determined prefix, we have that $blevel_k(L) \leq blevel_{k+1}(L)$. Thus, $blevel_k(L)$ is monotonically increasing with k. In addition, $blevel_k(L)$ is bounded by 1. Therefore, for every language L, we have that $\lim_{k\to\infty} blevel_k(L)$ exists, and thus $blevel(L)$ is well defined.

Let us show some examples. Let $AP = \{a\}$. The bounding level of $L = [\![a]\!]$ is 1, as $blevel_k(L) = 1$ for every $k \geq 1$. The bounding level of $L = [\![Fa]\!]$ is 1. Indeed, for every $k \geq 0$, the only prefix of length k that is not determined for L is $(\neg a)^k$. Since its probability is $\frac{1}{2^k}$, we have that $blevel_k(L) = Pr(determined_k(L)) = 1 - \frac{1}{2^k}$, which tends to 1 as k tends to ∞. The bounding level of $L = [\![FGa]\!]$ is 0, as $blevel_k(L) = 0$ for every $k \geq 0$. Indeed, the language $[\![FGa]\!]$ does not have determined prefixes.

Consider a language $L \subseteq \Sigma^\omega$. We say that L is *clopen* if it is both safety and co-safety [18]. In other words, a language L is clopen if every word in Σ^ω has a determined prefix for L. Let us first review some of the relevant terminology from set-theoretic topology. Consider a set X and a distance function $d : X \times X \to \mathbb{R}$ between the elements of X. For an element $x \in X$ and $\gamma \geq 0$, let $K(x, \gamma)$ be the set of elements x' such that $d(x, x') \leq \gamma$. Consider a set $S \subseteq X$. An element $x \in S$ is called an *interior element* of S if there is $\gamma > 0$ such that $K(x, \gamma) \subseteq S$. The set S is *open* if all the elements in S are interior. A set S is closed if $X \backslash S$ is open. So, a set S is open if every element in S has a nonempty "neighborhood" contained in S, and a set S is closed if every element not in S has a nonempty neighborhood whose intersection with S is empty. A set that is both open and close is called a *clopen* set.

A *Cantor space* consists of $X = D^\omega$, for some finite D, and d defined by $d(w, w') = \frac{1}{2^n}$, where n is the first position where w and w' differ. Thus, elements of X can be viewed as infinite words over D and two words are close to each other if they have a long common prefix. If $w = w'$, then $d(w, w') = 0$. It is known that clopen sets in Cantor space are *bounded* [17], where a set S is bounded if it is of the form $W \cdot D^\omega$ for some finite set $W \subseteq D^*$. Hence, clopen sets in our Cantor space correspond exactly to the bounded properties we are looking for: each clopen language $L \subseteq \Sigma^\omega$ has a bound $k \geq 0$ such that membership in L can be determined by the prefixes of length k of words in Σ^ω.

What are these clopen sets in Σ^ω? It turns out that topology has an answer to this question as well [10,21]: it is not hard to see that a language $L \subseteq \Sigma^\omega$ is co-safety iff L is an open set in our Cantor space. To see that, consider a word w in a co-safety language L, and let x be a good prefix of w. All the words w' with $d(w, w') \leq \frac{1}{2^{|x|}}$ have x as their prefix, so they all belong to L. For the second direction, consider a word w in an open set L, and let $\gamma > 0$ be such that $K(w, \gamma) \subseteq L$. The prefix of w of length $\lfloor \log \frac{1}{\gamma} \rfloor$ is a good prefix for L. It follows that the clopen sets in Σ^ω, namely the bounded properties we are after, are exactly these properties that are both safety and co-safety.

We define the *clopen level* of a language L, denoted $clevel(L)$, as the probability of a word to have a determined prefix for L. Formally, $clevel(L) = Pr(safe(L) \cup co\text{-}safe(L)) = Pr(safe(L)) + Pr(co\text{-}safe(L))$. The latter equality holds because the intersection of $safe(L)$ and $co\text{-}safe(L)$ is empty. When $clevel(L) = p$, we say that L is a *p-clopen language*. For example, the language $L = [\![a]\!]$ has $Pr(safe(L)) = \frac{1}{2}$ and $Pr(co\text{-}safe(L)) = \frac{1}{2}$, thus

$clevel([\![a]\!]) = 1$. The language $L = [\![a \wedge FGa]\!]$ has $Pr(safe(L)) = \frac{1}{2}$ and $Pr(co\text{-}safe(L)) = 0$, thus $clevel([\![a \wedge FGa]\!]) = \frac{1}{2}$.

As shown in [18], the clopen and bounded classes concide. That is, a language is bounded iff it is clopen. We give here the proof, for completeness.

Lemma 5.1 [18] *A language $L \subseteq \Sigma^\omega$ is bounded iff it is clopen.*

Proof Consider a language L. We first prove that if L is bounded then it is clopen. If L is bounded, then it is k-bounded for some $k \geq 0$. That is, for every word $w \in \Sigma^\omega$, either $w \in L$, in which case w has a good prefix of length at most k, or $w \notin L$, in which case w has a bad prefix of length at most k. Thus, L is both safety and co-safety, so it is clopen. For the other direction, assume that L is clopen. We first prove that every word in Σ^ω has only finitely many prefixes that are undetermined with respect to L. Consider a word $w \in \Sigma^\omega$. Since L is both safety and co-safety, w has a bad or good prefix. Let x be the minimal determined prefix of w. Clearly, w has $|x|$ undetermined prefixes (exactly all the strict prefixes of x), and we are done. Now assume, by way of contradiction, that L is not bounded. Thus, there are infinitely many $x \in \Sigma^*$ such that x is undetermined with respect to L. Since Σ is finite and the set of undetermined words is prefix closed, it follows by König's Lemma, that there is an infinite word w in Σ^ω all of whose prefixes are undetermined with respect to L, and we reached a contradiction. □

Theorem 5.2 shows that the equivalence of bounded and clopen classes is maintained in their spectrum. This observation validates our definitions of bounding and clopen levels.

Theorem 5.2 *For a language $L \subseteq \Sigma^\omega$, $blevel(L) = clevel(L)$.*

Proof Consider a language L. Note that $blevel = Pr(\bigcup_{k=0}^{\infty} determined_k(L))$ and $clevel(L) = Pr(safe(L) \cup co\text{-}safe(L))$. We prove that $\bigcup_{k=0}^{\infty} determined_k(L) = safe(L) \cup co\text{-}safe(L)$. First, we show that $\bigcup_{k=0}^{\infty} determined_k(L) \subseteq safe(L) \cup co\text{-}safe(L)$. Assume that $w \in \bigcup_{k=0}^{\infty} determined_k(L)$. That is, w has a determined prefix of length k for some $k \geq 0$. In particular, w has a determined prefix. Therefore, $w \in safe(L) \cup co\text{-}safe(L)$ and we are done. We left to show that $safe(L) \cup co\text{-}safe(L) \subseteq \bigcup_{k=0}^{\infty} determined_k(L)$. Assume, by way of contradiction, that for some $w \in \Sigma^\omega$ it holds that $w \in safe(L) \cup co\text{-}safe(L)$ and $w \notin \bigcup_{k=0}^{\infty} determined_k(L)$. That is, w has a bad or good prefix, but there is no $k \geq 0$ such that $w \in determined_k(L)$, so we reached a contradiction. □

We turn to study the relation between the bounding level, the safety level, and the co-safety level of a given language. The following lemma follows from Proposition 5.2 and from the definition of $clevel(L)$.

Lemma 5.3 *For a language $L \subseteq \Sigma^\omega$,*

$$blevel(L) = clevel(L) = Pr(L) \cdot co\text{-}slevel(L) + Pr(comp(L)) \cdot slevel(L).$$

We can now study classes of bounding level. Similarly to the safety setting, we define four classes of languages, describing their bounding level:

- *Bounded* A language $L \subseteq \Sigma^\omega$ is a bounded language if the mempership of every word in Σ^ω in L can be determined by observing only the prefix of length k of the word, for some $k \geq 0$. That is, every word has either a bad or a good prefix of a bounded length. For example, $L = [\![a]\!]$ is a 1-bounded language. Indeed, every word not in L has a prefix of length 1 in which a does not hold, and this prefix cannot be extended to a word in L, and every word in L has a prefix of length 1 in which a holds, and every extention of this prefix is in L. As mentioned above, we have that $bounded = safety \cap co\text{-}safety$.

Table 2 Bounding classes of languages with different safety and co-safety levels

	Safety	Almost-safety	Frac-safety	Liveness
Co-safety	Bounded	Almost-bounded	–	Almost-bounded
Almost-co-safety	Almost-bounded	Almost-bounded	Frac-bounded	Frac-bounded or almost-bounded
Frac-co-safety	–	Frac-bounded	Frac-bounded	Frac-bounded
Co-liveness	Almost-bounded	Frac-bounded or almost-bounded	Frac-bounded	Pending

- *Almost-bounded* A language $L \subseteq \Sigma^\omega$ is an almost-bounded language if $blevel(L) = 1$ and L is not bounded. As an example, consider the language $L = [\![aUb]\!]$. The language L is not a bounded language, since a^ω does not have a determined prefix. Every word except for a^ω has a determined prefix for L. Accordingly, $Pr(determined(L)) = 1$.
- *Frac-bounded* A language $L \subseteq \Sigma^\omega$ is a frac-bounded language if its bounding level is p for some $0 < p < 1$. As an example, consider the language $L = [\![a \wedge FGa]\!]$. This language is $\frac{1}{2}$-safety and co-liveness, and its probability is 0. By Lemma 5.3, $blevel(L) = Pr(L) \cdot co\text{-}slevel(L) + Pr(comp(L)) \cdot slevel(L) = 0 \cdot 0 + 1 \cdot \frac{1}{2}$. Hence, $blevel(L) = \frac{1}{2}$.
- *Pending* A language $L \subseteq \Sigma^\omega$ is a pending language if $safe(L) \cup co\text{-}safe(L) = \emptyset$. That is, a language is a pending language if it is both liveness and co-liveness. In other words, if L is a pending language then every word in Σ^* can be extended both to a word in L and to a word in $comp(L)$. For example, the language $L = [\![GFa]\!]$ is pending, as it is both liveness and co-liveness. Indeed, we can concatenate a^ω to every word in Σ^* and get a word in L, and we can concatenate $(\neg a)^\omega$ to every word in Σ^* and get a word not in L.

Proposition 5.4 *A language $L \subseteq \Sigma^\omega$ is pending iff $blevel(L) = 0$.*

Proof Consider a language $L \subseteq \Sigma^\omega$. Clearly, if L is a pending language, then $slevel(L) = 0$ and $co\text{-}slevel(L) = 0$. Therefore, by Lemma 5.3, we have $blevel(L) = 0$. For the other direction, note that the only way to obtain $blevel(L) = 0$ is by having both $slevel(L) = 0$ and $co\text{-}slevel(L) = 0$. It follows that if $blevel(L) = 0$, then, by Proposition 3.3, L is both liveness and co-liveness language, so it is a pending language. □

We are interested in determining the bounding class of a language, given its safety and co-safety classes. In Table 2 we show the classes of bounding that correspond to different combinations of safety and co-safety classes.

Note that since the definition of bounded languages does not distinguish between bad and good prefixes, then for every language L, we have $blevel(L) = blevel(comp(L))$. Hence, by Lemma 3.4, the table is symmetric, in the sense that the (i, j)-th entry agrees with the (j, i)-th entry.

Below we explain the different entries in the table. Consider a language L. First, if L is both a safety and a co-safety language, then L is a bounded language by the definition. Similarly, if L is both a liveness and a co-liveness language, then L is a pending language by the definition. For the other cases, recall that $blevel(L) = Pr(L) \cdot co\text{-}slevel(L) + Pr(comp(L)) \cdot slevel(L)$. If L is an almost-safety and either a co-safety or an almost-co-safety language, then $blevel(L) = Pr(L) \cdot 1 + Pr(comp(L)) \cdot 1 = 1$, and L is an almost-bounded language. Dually, if L is an almost-co-safety and either a safety or an almost-safety language, then $blevel(L) = 1$, so L is an almost-bounded language.

If L is a frac-safety language, then $blevel(L) = Pr(L) \cdot p + Pr(comp(L)) \cdot slevel(L)$ for some p. Note that $slevel(L) < 1$ and $p \leq 1$, thus $blevel(L) < 1$. In addition, we have $slevel(L) > 0$, and since the language is not a liveness language, we also have $Pr(comp(L)) > 0$. Therefore, we have $blevel(L) > 0$, thus the language is a frac-bounded language. Dually, if L is a frac-co-safety language, then $0 < blevel(L) < 1$ and L is a frac-bounded language. If L is both a safety and a co-liveness language, then $blevel(L) = Pr(L) \cdot 0 + Pr(comp(L)) \cdot 1$. By Proposition 4.6, we have that $Pr(L) = 0$. Therefore, $Pr(comp(L)) = 1$, thus $blevel(L) = 1$. In addition, the words in L have no good prefixes, thus L is not a bounded language. It follows that L is an almost-bounded language. Dually, if L is both a liveness and a co-safety language, then L is an almost-bounded language. If L is both an almost-safety and a co-liveness language, then $blevel(L) = Pr(L) \cdot 0 + Pr(comp(L)) \cdot 1$. By Proposition 3.1, we have that $Pr(comp(L)) > 0$. It follows that $0 < blevel(L) \leq 1$. Note that since L is an almost-safety language, it is not bounded. Therefore, L is either a frac-bounded or an almost-bounded language. Dually, if L is both a liveness and an almost-co-safety language, then L is either a frac-bounded or an almost-bounded language.

Lemma 5.5 *Consider a language $L \subseteq \Sigma^\omega$. If L is almost-safety and co-liveness, then it is almost-bounded iff $Pr(L) = 0$. Dually, If L is almost-co-safety and liveness, then it is almost-bounded iff $Pr(L) = 1$.*

Proof We prove the first claim, and the proof for the second claim is dual. Consider a language $L \subseteq \Sigma^\omega$, and assume that L is almost-safety and co-liveness. As explained below Table 2, we have that $blevel(L) = Pr(L) \cdot 0 + Pr(comp(L)) \cdot 1$, so $blevel(L) \leq 1$. An equality is reached iff $Pr(comp(L)) = 1$. Therefore, we have that $blevel(L) = 1$ iff $Pr(L) = 0$. Note that L is not bounded anyway, since it is almost-safety. □

6 Deciding bounding classes

In this section we study the problem of deciding membership in the four classes of bounding level. The problem of classification to classes of bounding level is strongly related to the problems of classification to classes of safety and co-safety level. Accordingly, in Sect. 6.1, we first study the problem of deciding co-safety classes. Then, in Sect. 6.2, we combine these results with the results on safety from Sect. 4, and study the problem of deciding bounding classes.

6.1 Deciding co-safety classes

We show that the complexity of deciding membership in the four classes of co-safety level coincides with the complexity of deciding membership in the four classes of safety level. This is straightforward for the formalisms of DPW and LTL, where complementation involves no blow-up, and requires some care in the case of NBWs.

Theorem 6.1 *Consider a language $L \subseteq \Sigma^\omega$.*

1. *Deciding whether L is co-safety and whether L is almost-co-safety is NLOGSPACE-complete for L given by a DPW and PSPACE-complete for L given by an NBW or an LTL formula.*

2. *Deciding whether L is frac-co-safety and whether L is co-liveness is NLOGSPACE-complete for L given by a DPW, PSPACE-complete for L given by an NBW, and EXPSPACE-complete for L given by an LTL formula.*

Proof First, note that DPW and LTL are formalisms that are closed under complementation in $O(1)$. Thus, the tight bounds for LTL and for DPW follow from Theorems 4.1, 4.10, and 4.11.

Since we have upper bounds of NLOGSPACE for DPWs, the upper bounds of PSPACE for NBWs follow from Theorem 2.1.

We turn to the lower bounds for NBWs, describing reductions from a PSPACE Turing machine. The details of the generic reduction are given in [25]. For the following reductions, we add to the alphabet of the NBW two letters: \$ and \$′. We start with co-safety. Given a PSPACE Turing machine T and an input x to it, we can generate an NBW $\mathcal{A}$ that accepts a word w iff w either starts not with a legal encoding of a computation of T over the input x, or starts with an encoding of a legal rejecting computation of T over x, or is an encoding of a legal accepting computation of T over x followed by infinitely many occurrences of the letter \$. We show that T accepts x iff $L(\mathcal{A})$ is not co-safety. First, if T accepts x, then $\mathcal{A}$ accepts words that encode a legal accepting computation of T over the input x followed by infinitely many occurrences of the letter \$. These words do not have a good prefix, and therefore $L(\mathcal{A})$ is not co-safety. If T rejects x, then $\mathcal{A}$ accepts words that encode a legal rejecting computation of T over x or do not encode a legal computation. Words from both categories have a good prefixes, so $L(\mathcal{A})$ is co-safety.

We continue to almost-co-safety and frac-co-safety, describing a reduction from a PSPACE Turing machine. The reduction is similar to the one for co-safety. We generate, given a PSPACE Turing machine T and an input x to it, an NBW $\mathcal{A}$ that accepts a word w iff w either starts not with a legal encoding of a computation of T over the input x, or is an encoding of a legal rejecting computation of T over x, followed by a string over $\{\$, \$'\}$ that satisfies the (described below) LTL formula ψ, or starts with an encoding of a legal accepting computation of T over x. For almost-co-safety, the formula ψ is the almost-co-safety formula $\neg(\$ U \$')$. For frac-co-safety, the formula ψ is the frac-co-safety formula $\$ \vee GF\$'$. Note that words that do not start with an encoding of a legal computation and words that do not start with an encoding of a legal accepting computation have a good prefix. For almost-co-safety, a word that starts with an encoding of a legal rejecting computation has a good prefix with probability 1, but does not have a good prefix if it is followed by $\$^{\omega}$. Therefore, $L(\mathcal{A})$ is almost-co-safety iff T rejects x. For frac-co-safety, a word that starts with an encoding of a legal rejecting computation has a good prefix with probability $\frac{1}{2}$, so $L(\mathcal{A})$ is frac-co-safety iff T rejects x.

We complete the proof by showing a similar reduction, from a PSPACE Turing machine, for co-liveness. Given a PSPACE Turing machine T and an input x to it, we generate an NBW $\mathcal{A}$ that accepts a word w iff w either starts not with a legal encoding of a computation of T over the input x and also includes infinitely many occurrences of the letter \$, or is an encoding of a legal rejecting computation of T over x, followed by infinitely many occurrences of the letter \$, or starts with an encoding of a legal accepting computation of T over x. It is easy to see that if T rejects x, then no word in $L(\mathcal{A})$ has a good prefix, and if T accepts x, then $\mathcal{A}$ accepts words with a good prefix, namely the words that start with an encoding of a legal accepting computation of T over x. Hence, T rejects x iff $L(\mathcal{A})$ is co-liveness. □

6.2 From safety and co-safety classes to bounding classes

We study the complexity of deciding membership in the four classes of bounding level. Interestingly, although an almost-bounded language may be liveness, the results show that the complexity of deciding almost-boundedness coincides with the complexity of deciding boundedness, as well as with the complexities of deciding safety and almost-safety. Also, the complexities of deciding membership in the classes of frac-bounded and pending languages coincide with the complexities of deciding frac-safety and liveness.

The procedure of deciding bounding class can be based on deciding safety and co-safety classes and relying on Table 2 and Lemma 5.5. We also describe direct algorithms for which, as in the case of safety, we first need some observations about deterministic automata. The following theorem is analogous to Theorem 4.5.

Theorem 6.2 *Consider a deterministic automaton $\mathcal{A}$ with at most one empty state and at most one universal state, and let $L = L(\mathcal{A})$.*

1. *The language L is bounded iff $\mathcal{A}$ does not have a mixed ergodic SCC and all its paths reach an ergodic SCC.*
2. *The language L is almost-bounded iff $\mathcal{A}$ does not have a mixed ergodic SCC and has a path that does not reach an ergodic SCC.*
3. *The language L is frac-bounded iff $\mathcal{A}$ has a pure ergodic SCC and a mixed ergodic SCC.*
4. *The language L is pending iff $\mathcal{A}$ does not have a pure ergodic SCC.*

Proof Consider a deterministic automaton $\mathcal{A}$ with at most one empty state and at most one universal state, and let $L = L(\mathcal{A})$.

1. Recall that a language is bounded iff it is both safety and co-safety. Combining Theorem 4.5(4) and its dual, we have that L is bounding iff $\mathcal{A}$ does not have mixed ergodic SCCs and all its paths reach an ergodic SCC.
2. First, assume that $\mathcal{A}$ does not have a mixed ergodic SCC and has a path that does not reach an ergodic SCC. Note that $\mathcal{A}$ does not have empty or universal states that are not ergodic. Since $\mathcal{A}$ has a path that does not reach an ergodic SCC, there is a word without a determined prefix for L. In addition, since $\mathcal{A}$ does not have a mixed ergodic SCC, we have that $Pr(determined(L)) = 1$. Therefore, L is almost-bounded. For the other direction, assume that L is almost-bounded. Hence, $Pr(determined(L)) = 1$, so $\mathcal{A}$ does not have a mixed ergodic SCC. Yet, there is a word without a determined prefix for L. Therefore, $\mathcal{A}$ has a path that does not reach an ergodic SCC, and we are done.
3. Note that L is frac-bounded iff $0 < Pr(determined(L)) < 1$. It is easy to see that $Pr(determined(L)) < 1$ iff $\mathcal{A}$ has a mixed ergodic SCC, and that $Pr(determined(L)) > 0$ iff $\mathcal{A}$ has a pure ergodic SCC. Therefore, we have that L is frac-bounded iff $\mathcal{A}$ has a pure ergodic SCC and a mixed ergodic SCC.
4. Since L is pending iff it has no determined prefixes and since a path to a pure ergodic SCC in $\mathcal{A}$ induces a determined prefix for L, we have that L is pending iff $\mathcal{A}$ does not have a pure ergodic SCC. □

Note that, similarly to the case of safety classes, the characterization in Theorem 6.2 reduces the problem of finding the bounding class to a search for "witness SCCs" or "witness paths". We can now study the complexities for the different classes and formalisms.

Theorem 6.3 *Deciding whether a language $L \subseteq \Sigma^\omega$ is bounded and deciding whether it is almost-bounded is NLOGSPACE-complete for L given by a DPW and PSPACE-complete for L given by an NBW or an LTL formula.*

Proof We start with the upper bounds. Both problems can be solved by checking safety and co-safety classes, using Table 2, or directly, based on Theorem 6.2. We first describe the direct algorithms, showing that the problems are in NLOGSPACE for DPWs. The upper bounds for NBWs then follow from Theorem 2.1. Consider a DPW $\mathcal{A}$. By Theorem 6.2(2), we have that $L(\mathcal{A})$ is bounded iff $\mathcal{A}$ does not have a mixed ergodic SCC and all its paths reach an ergodic SCC. In addition, $L(\mathcal{A})$ is almost-bounded iff $\mathcal{A}$ does not have a mixed ergodic SCC and has a path that does not reach an ergodic SCC. Since NLOGSPACE is closed under complementation, and since we can verify the classification of a given SCC in NLOGSPACE, checking whether $\mathcal{A}$ satisfies each of the conditions above can be done in NLOGSPACE.

We show now the algorithms that are based on finding the safety and co-safety classes. While a naive application of Theorems 6.2(1), 6.2(2) and 2.1 only gives EXPSPACE upper bounds for LTL, these algorithms give upper bounds of PSPACE.

For bounded, consider a language L. By Theorems 4.1 and 6.1(1), we can check whether L is safety and whether it is co-safety in NLOGSPACE if L is given by a DPW and in PSPACE if L is given by NBW or by an LTL formula. If L is both safety and co-safety, then it is a bounded language. Otherwise, L is not a bounded language.

Regarding almost-bounded, consider the following procedure. Given a language L, we check whether it is safety or almost-safety and whether it is co-safety or almost-co-safety. Note that each of the above can be done in NLOGSPACE if L is given by a DPW and in PSPACE if L is given by NBW or by an LTL formula. If the check results give a combination that yields, according to Table 2, an almost-bounded language, then we conclude that L is almost-bounded. For example, if L is safety and almost-co-safety, then it is almost-bounded. If L is safety and it is nor co-safety neither almost-co-safety, then, since it cannot be frac-co-safety, it is almost-bounded too. If the results give a language that is not almost-bounded according to Table 2, then we conclude that L is not almost-bounded. For example, if L is safety and co-safety, then it is not almost-bounded. The only case in which we cannot make a decision yet is when L is almost-safety and is nor co-safety neither almost-co-safety, or, dually, when L is almost-co-safety and is nor safety neither almost-safety. We describe the procedure for the first case. The procedure for the second case is dual. As explained below Table 2 and by Lemma 5.5, an almost-safety language that is also a frac-co-safety or a co-liveness language, can be almost-bounded only if it is co-liveness and its probability is 0. In addition, according to the dual of Proposition 3.1, a language with probability 0 cannot be a frac-co-safety language. Therefore, it is sufficient to check whether $Pr(L) = 0$, which can be done in NLOGSPACE for NBWs and for DPWs by checking that the automaton does not have an accepting SCC. For LTL formulas, it can be done in PSPACE by translating the formula to an NBW and checking whether its probability is 0.

We turn to prove the lower bounds, and start with the lower bounds for bounded. We show first a lower bounds of NLOGSPACE for DPWs, describing a reduction from the non-reachability problem. Given a graph $G = \langle V, E\rangle$ and two vertices u and v in V, we construct a DPW $\mathcal{A}$ such that $L(\mathcal{A})$ is bounded iff v is not reachable from u. The DPW is constructed exactly in the same way as in the proof of Theorem 4.11. We prove that $L(\mathcal{A})$ is bounded iff v is not reachable from u. If v is not reachable from u, then $L(\mathcal{A})$ is empty and therefore it is bounded. If v is reachable from u, then $\mathcal{A}$ has a mixed ergodic SCC. Then, by Theorem 6.2(1), we have that $L(\mathcal{A})$ is not bounded.

We continue to the lower bound for NBWs, showing a reduction from a PSPACE Turing machine. Given a PSPACE Turing machine T and an input x to it, we generate an NBW $\mathcal{A}$ such that T rejects x iff $L(\mathcal{A})$ is bounded. The reduction is exactly the same as the one that shown in Theorem 6.1 for co-safety. We prove that T rejects x iff $L(\mathcal{A})$ is bounded. If T

reject x then $L(\mathcal{A})$ is universal, and hence, it is bounded. If T accepts x, then $L(\mathcal{A})$ is not co-safety. Therefore, it is not bounded.

We complete the proof for bounded by proving a lower bound of PSPACE for LTL formulas. We do this by a reduction from the validity problem. We use the same reduction that is shown in Theorem 4.10, and prove that φ is valid iff $\varphi \vee (aUb)$ is bounded, where φ is an LTL formula over AP, and a and b are atomic propositions not in AP. First, if φ is valid then so is $\varphi \vee (aUb)$, and therefore it is bounded. For the other direction, assume that φ is not valid. Note that a bad prefix for $\varphi \vee (aUb)$ is a bad prefix both for φ and for aUb. Since φ is not valid, there is a computation that does not satisfy it. Consider a computation whose projection on $\{a, b\}$ is a^ω and whose projection on AP is a computation that does not satisfy φ. Clearly, this computation does not satisfy $\varphi \vee (aUb)$ and has no bad prefix, so $\varphi \vee (aUb)$ is not bounded.

We turn to prove the lower bounds for almost-bounded, and start with DPWs. We describe a reduction from the reachability problem. Given a graph $G = \langle V, E\rangle$ and two vertices u and v in V, we construct a DPW $\mathcal{A} = \langle \Sigma, Q, Q_0, \delta, \alpha\rangle$ such that v is reachable from u iff $L(\mathcal{A})$ is almost-bounded. The DPW $\mathcal{A}$ is very similar to the one that is described in the proof of Theorem 4.10. The only difference is that now the originally implicit rejecting sink becomes an accepting one. In other words, every computation of $\mathcal{A}$ is accepting, except the one that reaches v and stays there forever. Note that such a computation does not reach an ergodic SCC. We prove that v is reachable from u iff $L(\mathcal{A})$ is almost-bounded. Consider the automaton $\mathcal{A}'$ that is equal to $\mathcal{A}$, except that the universal states are merged to one accepting sink. Note that $L(\mathcal{A}') = L(\mathcal{A})$, so it is sufficient to prove that v is reachable from u iff $L(\mathcal{A}')$ is almost-bounded. First, if v is reachable from u then $\mathcal{A}'$ does not have a mixed ergodic SCC, and the path from u to v that stays in v is a path that does not reach an ergodic SCC. Therefore, by Theorem 6.2(2), we have that $L(\mathcal{A}')$ is almost-bounded. If v is not reachable from u, then $\mathcal{A}'$ is universal and therefore is not almost-bounded, and we are done.

For NBWs, we use the same reduction from a PSPACE Turing machine as in the proof of Theorem 6.1 for almost-co-safety. We show that $L(\mathcal{A})$ is almost-bounded iff T rejects x. The first direction is easy: if T accepts x then $L(\mathcal{A})$ is universal and is therefore not almost-bounded. For the other direction, assume that T rejects x. Then, as shown in the proof of Theorem 6.1, $L(\mathcal{A})$ is almost-co-safety. In addition, note that $\mathcal{A}$ rejects only these computations that are an encoding of a legal rejecting computation of T over x followed by a string over $\{\$, \$'\}$ that does not satisfy $\neg(\$U\$')$. These computations have a bad prefix, so $L(\mathcal{A})$ is safety. By Table 2, this implies that $L(\mathcal{A})$ is almost-bounded.

It is left to prove a lower bound of PSPACE for LTL formulas. We show a reduction from the non-validity problem for safety LTL formulas. Given a safety LTL formula φ over AP, let a be an atomic proposition not in AP. We prove that φ is not valid iff $\varphi \vee Fa$ is almost-bounded. Note that a bad prefix for $\varphi \vee Fa$ is a bad prefix both for φ and for Fa, and that a good prefix for $\varphi \vee Fa$ is a good prefix either for φ or for Fa. For the first direction, assume that φ is not valid. Therefore, there is a computation that does not satisfy φ. A computation that does not satisfy φ and does not satisfy Fa is a computation that does not satisfy $\varphi \vee Fa$ and also has no bad prefix. Therefore, $\varphi \vee Fa$ is not bounded. We show that $blevel([\![\varphi \vee Fa]\!]) = 1$. First, note that $Pr(\varphi \vee Fa) = 1$. We are interested in finding $co\text{-}slevel([\![\varphi \vee Fa]\!])$. Since Fa is co-safety, a computation that satisfies $\varphi \vee Fa$ does not have a good prefix iff it does not satisfy Fa, satisfies φ, and does not have a good prefix for φ. The probability of that is zero, because even the probability of not satisfying Fa is zero. Hence, $co\text{-}slevel([\![\varphi \vee Fa]\!]) = 1$. Then, by Lemma 5.3, we have that $blevel([\![\varphi \vee Fa]\!]) = 1$. Therefore, if φ is not valid, then $\varphi \vee Fa$ is almost-bounded. The other direction is easy: if φ is valid then so is $\varphi \vee Fa$, implying that $\varphi \vee Fa$ is not almost-bounded. □

Theorem 6.4 *Deciding whether a language $L \subseteq \Sigma^\omega$ is frac-bounded and deciding whether it is pending is NLOGSPACE-complete for L given by a DPW, PSPACE-complete for L given by an NBW, and EXPSPACE-complete for L given by an LTL formula.*

Proof We start with the upper bounds. As in the proof of Theorem 6.3, the problems can be solved directly by observing a DPW for a given language, or by deciding safety and co-safety classes and relying on Table 2. We describe both ways, starting with the direct algorithms. We show that the problems are in NLOGSPACE for DPWs. The upper bounds for NBWs and LTL formulas then follow from Theorem 2.1. Consider a DPW $\mathcal{A}$. By Theorem 6.2(3, 4), we have that $L(\mathcal{A})$ is frac-bounded iff $\mathcal{A}$ has a pure ergodic SCC and a mixed ergodic SCC, and it is pending iff $\mathcal{A}$ does not have a pure ergodic SCC. Since NLOGSPACE is closed under complementation, and since we can verify the classification of a given SCC in NLOGSPACE, checking whether $\mathcal{A}$ satisfies each of the conditions above can be done in NLOGSPACE.

We turn to the algorithms that are based on finding the safety and co-safety classes, giving the same upper bounds as the direct algorithms.

For pending, the procedure is easy. Given a language L, we check whether L is liveness and whether it is co-liveness. By Theorems 4.1 and 6.1(2), it can be done in NLOGSPACE if L is given by a DPW, in PSPACE if L is given by NBW, and in EXPSPACE if L is given by an LTL formula. If L is both liveness and co-liveness, then it is a pending language. Otherwise, L is not a pending language.

For frac-bounded, consider the following procedure. Given a language L, we find the class of safety and the class of co-safety in which L is. If the check results give a combination that yields, according to Table 2, a frac-bounded language, then we conclude that L is frac-bounded. For example, if L is frac-safety, then it is frac-bounded. If the results give a language that is not frac-bounded according to Table 2, then we conclude that L is not frac-bounded. For example, if L is liveness and co-safety, then it is not almost-bounded. The only case in which we cannot make a decision yet is when L is almost-safety and co-liveness, or, dually, when L is almost-co-safety and liveness. We describe the procedure for the first case. The procedure for the second case is dual. By Lemma 5.5, an almost-safety and co-liveness language is frac-bounded iff its probability is greater than 0. Therefore, it is sufficient to check whether $Pr(L) = 0$. If $Pr(L) = 0$ then we conclude that L is not frac-bounded. Otherwise, we have that L is frac-bounded. Note that all of the decisions that the procedure includes can be done in NLOGSPACE if L is given by a DPW, in PSPACE if L is given by NBW, and in EXPSPACE if L is given by an LTL formula.

We continue to the lower bounds, and start with frac-bounded.

For DPWs, we base on the reduction from reachability that is shown in the proof of Theorem 4.10 for frac-safety. We show that for a graph $G = \langle V, E\rangle$ and two vertices u and v in V, the language of the DPW $\mathcal{A}$ that is shown there is frac-bounded iff v is reachable from u. First, if v is reachable from u then $L(\mathcal{A})$ is frac-safety, as proved in Theorem 4.10, and, by Table 2, is therefore frac-bounded. For the other direction, assume that v is not reachable from u. Then, $L(\mathcal{A})$ is empty, implying it is not frac-safety.

For NBWs, we use the reduction from a PSPACE Turing machine that is shown in the proof of Theorem 6.1(2) for frac-co-safety. We prove that T rejects x iff $L(\mathcal{A})$ is frac-bounded. The first direction is easy: as proved in Theorem 6.1(2), if T rejects x, then $L(\mathcal{A})$ is frac-co-safety. Therefore, as shown in Table 2, it is frac-bounded. In the other direction, if T accepts x, then $L(\mathcal{A})$ is universal, and is therefore not frac-co-safety.

It is left to show the lower bound for LTL formulas. We show a reduction from the problem of deciding non-liveness. Given an LTL formula ψ over AP, we construct an LTL formula φ such that ψ is not liveness iff φ is frac-bounded. The construction of φ is as follows. Let

a and b be atomic propositions not in AP. We define $\varphi = G\psi \vee (a \wedge GFb)$. We prove that ψ is liveness iff φ is frac-bounded. Note that a bad prefix for φ is a bad prefix for both $G\psi$ and $a \wedge GFb$, and that a good prefix for φ is a good prefix for $G\psi$ or $a \wedge GFb$. For the first direction, assume that ψ is liveness. Then, $G\psi$ has no determined prefix, so it is pending. Since $a \wedge GFb$ have no good prefixes, it follows that φ is pending, so it is not frac-bounded. For the other direction, assume that ψ is liveness, thus it has a bad prefix. Therefore, $G\psi$ also has a bad prefix. Since $a \wedge GFb$ has a bad prefix too, we have that φ has a bad prefix, so $slevel([\![\varphi]\!]) > 0$. In addition, since nor $G\psi$ neither $a \wedge GFb$ have a good prefix, we have that $co\text{-}slevel([\![\varphi]\!]) = 0$. Then, it follows from Lemma 5.3 that $blevel([\![\varphi]\!]) = Pr(comp([\![\varphi]\!])) \cdot slevel([\![\varphi]\!])$. Since φ has a bad prefix, we have that $Pr(\varphi) < 1$, so $Pr(\neg\varphi) > 0$. In addition, since $Pr(a \wedge GFb) = \frac{1}{2}$, we have that $Pr(\varphi) \geq \frac{1}{2}$, so $0 < Pr(\neg\varphi) \leq \frac{1}{2}$. Therefore, we have that $0 < blevel([\![\varphi]\!]) \leq \frac{1}{2}$, so φ is frac-bounded.

We turn to the lower bounds for pending. We start with a lower bound of NLOGSPACE for DPWs, describing a reduction from the non-reachability problem. Given a graph $G = \langle V, E\rangle$ and two vertices u and v in V, we construct a DPW $\mathcal{A} = \langle \Sigma, Q, Q_0, \delta, \alpha\rangle$ such that $L(\mathcal{A})$ is pending iff v is not reachable from u. The DPW $\mathcal{A}$ is similar to G, with a new state q and additional transitions from each state in $V\backslash\{v\}$ to q and from q to u, and with additional self loop in v and in q. Intuitively, all of the states except for q are accepting, so $(V \cup \{q\})\backslash\{v\}$ is a mixed ergodic SCC, and $\{v\}$ is a pure ergodic SCC. Then, v is not reachable from u iff $\mathcal{A}$ has no pure ergodic SCCs.

Formally, $\mathcal{A} = \langle E \cup \{e_{new}\}, V \cup \{q\}, \{u\}, \delta, \{\emptyset, V, V \cup \{q\}\}\rangle$, where δ is such that for every edge $e = \langle w, w'\rangle \in E$, we have a transition $\langle w, e, w'\rangle$ in δ. That is, all the edges of the graph are transitions in $\mathcal{A}$, all labeled differently. In addition, for all $w \in V\backslash\{v\}$ and for all $e \in E \cup \{e_{new}\}$ such that there is no transition from w that is labeled by e, we add a transition $\langle w, e, q\rangle$. Finally, we add to δ the transitions $\langle q, e_{new}, q\rangle$ and $\langle q, e, u\rangle$ for all $e \in E$, and the transition $\langle v, e, v\rangle$ for all $e \in E \cup \{e_{new}\}$. Note that $\mathcal{A}$ is a DPW, as required, and that the reduction is computable using logarithmic space.

We now prove that $L(\mathcal{A})$ is pending iff v is not reachable from u. Note that $(V \cup \{q\})\backslash\{v\}$ is a mixed ergodic SCC, and that $\{v\}$ is a pure ergodic SCC. Therefore, $\mathcal{A}$ has a mixed ergodic SCC, and it has a pure ergodic SCC iff v is reachable from u. By Theorem 6.2(4), we have that $L(\mathcal{A})$ is pending iff v is not reachable from u.

We turn to the lower bound for NBWs. We prove a lower bound of PSPACE, describing a reduction from a PSPACE Turing machine. As in the proof of Theorem 6.1, we add the letter \$ to the alphabet. Given a PSPACE Turing machine T and an input x to it, we generate an NBW $\mathcal{A}$ that accepts a word w iff w is either not a legal encoding of a computation of T over the input x followed by infinitely many occurrences of the letter \$, or an encoding of a legal rejecting computation of T over x followed by infinitely many \$'s, or starts with an encoding of a legal accepting computation of T over x. We prove that T rejects x iff $L(\mathcal{A})$ is a pending language. It is easy to see that if T rejects x then $L(\mathcal{A})$ accepts only words that are followed by infinitely many \$'s. That is, if T rejects x then $L(\mathcal{A})$ has no determined prefixes, so it is pending. If T accepts x, then $L(\mathcal{A})$ includes words that start with an encoding of a legal accepting computation of T over x. Hence, $\mathcal{A}$ accepts words with a good prefix, so $L(\mathcal{A})$ is not pending, and we are done.

We left to prove a lower bound of EXPSPACE for LTL formulas. The reduction that we show for frac-bounded can work for pending too. Recall that given an LTL formula ψ, we construct an LTL formula φ such that ψ is not liveness iff φ is frac-safety, and that if ψ is liveness then φ is pending. Therefore, the same reduction is a reduction from the problem of deciding liveness to the problem of deciding pending, and we are done. □

7 Discussion and directions for future research

We defined and studied safety levels of ω-regular languages. One can define the *relative* safety level of a specification in a system. Then, rather than taking the probability distribution with respect to Σ^ω, we take it with respect to the set of computations generated by a system §. For example, if § does not generate the computation a^ω, then the specification aUb is safety with respect to §. Relative safety and liveness have been studied in [15]. It is interesting to extend the study and add relativity to the full spectrum between safety and liveness. In particular, relativity may both increase and decrease the safety level.

A different approach to span the spectrum between safety and liveness was taken in [7]. Recall that the probability of a specification is measured with respect to random computations in Σ^ω. Alternatively, one can also study the probability of formulas to hold in computations of random *finite-state* systems. Formally, for an integer $l \geq 1$, let $Pr_l(\varphi)$ denote the probability that φ holds in a random cycle of length l. Here too, the probability that each atomic proposition holds in a state is $\frac{1}{2}$, yet we have only l states to fix an assignment to. So, for example, while $Pr(Gp) = 0$, we have that $Pr_1(Gp) = \frac{1}{2}$, $Pr_2(Gp) = \frac{1}{4}$, and in general $Pr_j(Gp) = \frac{1}{2^j}$. Indeed, an l-cycle satisfies Gp iff all its states satisfy p. It is suggested in [7] to characterize safety properties by means of the asymptotic behavior of $Pr_l(\varphi)$. The idea is to define different levels of safety according to the rate the probability decreases or increases. For example, clearly $Pr_l(Gp)$ tends to 0 as l increases, whereas $Pr_l(Fp)$ tends to 1. As it turns out, however, the characterization is not clean. For example, FGp is a liveness formula, but $Pr_l(FGp)$ decreases as l increases. It is interesting to see whether a combination of the safety level studied here and the finite-state system approach of [7] can lead to a refined spectrum.

In practice, the safety level of a language L indicates how well algorithms that are designated for safety specifications can work for L. We propose two approximated model-checking algorithms for languages with a high safety level. In one, model checking proceeds using an automaton for the bad prefixes (c.f. [16]). Here, we may get a one-sided error in which model checking succeeds even though the system has a bad computation (evidently, one with no bad prefix). In the second, model checking ignores the acceptance condition of an automaton for the specification and views it as a looping automaton (that is, all infinite runs that do not reach an empty state are accepting). Here, we may get a one-sided error in which model checking fails even though the system has no computation that violates the specification (evidently, it has a computation all whose prefixes can be extended to computations that violates the specification). When the specification is safety, no errors occur. Also, the higher its safety level is, the less probable the two types of errors are. Combining this with the relative approach described above can tighten our expectation of error further.

References

1. Alpern, B., Schneider, F.B.: Defining liveness. Inf. Process. Lett. **21**, 181–185 (1985)
2. Alpern, B., Schneider, F.B.: Recognizing safety and liveness. Distrib. Comput. **2**, 117–126 (1987)
3. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic model checking without BDDs. In: Proceedings of the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Lecture Notes in Computer Science, vol. 1579. Springer (1999)
4. Bloem, R., Gabow, H.N., and Somenzi, F.: An algorithm for strongly connected component analysis in $n \log n$ symbolic steps. In: Proceedings of the 3rd International Conference on Formal Methods in Computer-Aided Design. Lecture Notes in Computer Science, vol. 1954, pp. 37–54. Springer (2000)

5. Courcoubetis, C., Yannakakis, M.: Markov decision processes and regular events. In: Proceedings of the 17th International Colloquium on Automata, Languages, and Programming. Lecture Notes in Computer Science, vol. 443, pp. 336–349. Springer (1990)
6. Courcoubetis, C., Yannakakis, M.: The complexity of probabilistic verification. J. ACM **42**, 857–907 (1995)
7. Ben David, S., Kupferman, O.: A framework for ranking vacuity results. In: 11th International Symposium on Automated Technology for Verification and Analysis. Lecture Notes in Computer Science, vol. 8172, pp. 148–162. Springer (2013)
8. Emerson, E.A.: Alternative semantics for temporal logics. Theor. Comput. Sci. **26**, 121–130 (1983)
9. Filiot, E., Jin, N., Raskin, J.-F.: An antichain algorithm for LTL realizability. In: Proceedings of the 21st International Conference on Computer Aided Verification, vol. 5643, pp. 263–277 (2009)
10. Gumm, H.P.: Another glance at the Alpern–Schneider characterization of safety and liveness in concurrent executions. Inf. Process. Lett. **47**, 291–294 (1993)
11. Harel, D., Katz, G., Marron, A., Weiss, G.: Non-intrusive repair of reactive programs. In: ICECCS, pp. 3–12 (2012)
12. Havelund, K., Rosu, G.: Synthesizing monitors for safety properties. In: Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Lecture Notes in Computer Science, vol. 2280, pp. 342–356. Springer (2002)
13. Jones, N.D.: Space-bounded reducibility among combinatorial problems. J. Comput. Syst. Sci. **11**, 68–75 (1975)
14. Kemeny, J.G., Snell, J.L., Knapp, A.W.: Denumerable Markov Chains. Springer, Berlin (1976)
15. Kupferman, O., Vardi, G.: On relative and probabilistic finite counterability. In: Proceedings of the 24th Annual Conference of the European Association for Computer Science Logic. Leibniz International Proceedings in Informatics (LIPIcs), vol. 41, pp. 175–192 (2015)
16. Kupferman, O., Vardi, M.Y.: Model checking of safety properties. Form. Methods Syst. Des. **19**(3), 291–314 (2001)
17. Kupferman, O., Vardi, M.Y.: On bounded specifications. In Proceedings of the 8th International Conference on Logic for Programming Artificial Intelligence and Reasoning. Lecture Notes in Computer Science, vol. 2250, pp. 24–38. Springer (2001)
18. Kupferman, O., Vardi, M.Y.: Synthesizing distributed systems. In: Proceedings of the 16th IEEE Symposium on Logic in Computer Science, pp. 389–398 (2001)
19. Lamport, L.: Logical foundation. In: Distributed Systems—Methods and Tools for Specification. Lecture Notes in Computer Science, vol. 190. Springer (1985)
20. Landweber, L.H.: Decision problems for ω-automata. Math. Syst. Theory **3**, 376–384 (1969)
21. Manna, Z., Pnueli, A.: The anchored version of the temporal framework. In: Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency. Lecture Notes in Computer Science, vol. 345, pp. 201–284. Springer (1989)
22. Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer, Berlin (1992)
23. Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems: Safety. Springer, Berlin (1995)
24. Meyer, A.R., Stockmeyer, L.J.: The equivalence problem for regular expressions with squaring requires exponential time. In: Proceedings of the 13th IEEE Symposium on Switching and Automata Theory, pp. 125–129 (1972)
25. Meyer, A.R., Stockmeyer, L.J.: Word problems requiring exponential time: preliminary report. In: Proceedings of the 5th ACM Symposium on Theory of Computing, pp. 1–9 (1973)
26. Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. In: Proceedings of the 21st IEEE Symposium on Logic in Computer Science, pp. 255–264. IEEE press (2006)
27. Pnueli, A., Shahar, E.: Liveness and acceleration in parameterized verification. In: Proceedings of the 12th International Conference on Computer Aided Verification. Lecture Notes in Computer Science, pp. 328–343. Springer (2000)
28. Safra, S.: On the complexity of ω-automata. In: Proceedings of the 29th IEEE Symposium on Foundations of Computer Science, pp. 319–327 (1988)
29. Sistla, A.P.: Safety, liveness and fairness in temporal logic. Form. Asp. Comput. **6**, 495–511 (1994)
30. Sistla, A.P., Clarke, E.M.: The complexity of propositional linear temporal logic. J. ACM **32**, 733–749 (1985)
31. Tarjan, R.E.: Depth first search and linear graph algorithms. SIAM J. Comput. **1**(2), 146–160 (1972)
32. Vardi, M.Y., Wolper, P.: Reasoning about infinite computations. Inf. Comput. **115**(1), 1–37 (1994)