Automata-less Monitoring via Trace-Checking

Andrea Brunello, Luca Geatti, Angelo Montanari, Nicola Saccomanno

Department of Mathematics, Computer Science and Physics
University of Udine
Via delle Scienze 206, Udine, Italy
{andrea.brunello, luca.geatti, angelo.montanari, nicola.saccomanno}@uniud.it

Abstract

In runtime verification, *monitoring* consists of analyzing the current execution of a system and determining, on the basis of the observed finite trace, whether all its possible continuations satisfy or violate a given specification. This is typically done by synthesizing a monitor–often a Deterministic Finite State Automaton (DFA)–from logical specifications expressed in Linear Temporal Logic (LTL) or in its finite-word variant (LTLf). Unfortunately, the size of the resulting DFA may incur a doubly exponential blow-up in the size of the formula.

In this paper, we identify some conditions under which monitoring can be done *without* constructing such a DFA. We build on the notion of *intentionally safe and cosafe* formulas to show that monitoring of these formulas can be carried out through *trace-checking*, that is, by directly evaluating them on the current system trace, with a polynomial complexity in the size of both the trace and the formula.

In addition, we investigate the complexity of recognizing intentionally safe and cosafe formulas for the safety and cosafety fragments of LTL and LTLf. As for LTLf, we show that all formulas in these fragments are intentionally safe and cosafe, thus removing the need for the check. As for LTL, we prove that the problem is in PSPACE, significantly improving over the EXPSPACE complexity of full LTL.

Extended version — https://arxiv.org/abs/2511.11072

1 Introduction

Runtime verification (Leucker and Schallhart 2009) refers to a family of techniques to check whether the execution of a system under scrutiny complies with the desired properties. Unlike model checking, which exhaustively explores all possible system behaviors, runtime verification analyzes only the current execution trace. Nevertheless, runtime verification techniques offer substantial advantages: (i) they do not require a complete model of the system, (ii) they can be directly applied to the system implementation, and (iii) they are generally more efficient.

One of the most commonly used approaches in runtime verification is *monitoring* (Leucker and Schallhart 2009; Bauer, Leucker, and Schallhart 2011), which involves the automatic construction of a *monitor* from a logical property. This monitor takes as input the current execution trace and outputs: (i) $\top$ if all extensions of the trace satisfy the property; (ii) $\bot$ if all extensions violate the property; or (iii) ? if the verdict is still undetermined. The monitor's verdict is, by definition, irrevocable. It follows that not all logical properties are monitorable. For example, it is not possible to construct a monitor meeting the specified criteria for properties such as "infinitely many times p."

Typically, properties suitable for monitoring fall into two classes: (i) *safety properties*, which assert that "something bad never happens", and (ii) *cosafety properties*, which assert that "something good eventually happens". Every violation of a safety property contains a *bad prefix*, *i.e.*, a finite word such that all its extensions violate the property. Conversely, every execution that satisfies a cosafety property contains a *good prefix*, *i.e.*, a finite word such that all its extensions satisfy the property (Kupferman and Vardi 2001).

The logical framework we consider to specify monitorable properties is LTL (Pnueli 1977) and its finite-word counterpart LTLf (De Giacomo and Vardi 2013). Both extend propositional logic with temporal modalities. As an example, the property $\mathsf{G}(\neg deadlock)$ is a typical safety property, which states that the system never enters a deadlock. An example of a cosafety property is $(rules)\ \mathsf{U}\ (goal)$, stating that the system eventually reaches a goal and respects certain rules up to that point.

Two syntactic fragments of LTL and LTLf are particularly significant in the context of safety and cosafety properties. The first one is $\mathsf{F}(\mathsf{pLTL})$, which consists of formulas of the form $\mathsf{F}(\psi)$, where F is the "eventually" modality and ψ is a pure past formula, *i.e.*, one with past temporal modalities only. Dually, the fragment $\mathsf{G}(\mathsf{pLTL})$ consists of formulas of the form $\mathsf{G}(\psi)$, where G is the "always" modality and ψ is a pure past formula. It has been shown that a safety (resp., cosafety) property is expressible in LTL if and only if it is expressible in $\mathsf{G}(\mathsf{pLTL})$ (resp., in $\mathsf{F}(\mathsf{pLTL})$) (Chang, Manna, and Pnueli 1992).

A significant drawback of LTL and LTLf monitoring is that it typically requires constructing a Deterministic Finite Automaton (DFA) from the input formula (Leucker and Schallhart 2009; Bauer, Leucker, and Schallhart 2011). Even for safety and cosafety properties, this automaton can be doubly exponential in the size of the formula (Kupferman

and Vardi 2001). This severely limits the practical applicability of monitoring as the size of the formula increases.

In this paper, we investigate under which conditions monitoring for LTL and LTLf can be reduced to *trace-checking*– that is, deciding the truth of a formula directly on the execution trace, without the need to build a DFA. To this end, we exploit the notions of *intentionally safe* and *intentionally cosafe* formulas, introduced by Kupferman and Vardi in the early 2000s in the context of model checking (Kupferman and Vardi 2001). Both notions are based on the concept of *informative models*. A model is informative for a formula ϕ if it contains sufficient information to determine whether ϕ is true or false. As an example, the word $\langle\{p\}\rangle$ is an informative model for the LTL formula $\mathsf{F}(p)$, since p holds at the first position. In contrast, the same word is *not* informative for $\mathsf{F}(p \wedge (\mathsf{X}q \vee \mathsf{X}\neg q))$, because evaluating the formula requires a position satisfying p followed by a position where either q or $\neg q$ holds, but $\langle\{p\}\rangle$ has no successor. Notably, since $\mathsf{X}q \vee \mathsf{X}\neg q$ is a tautology[1], the formulas $\mathsf{F}(p)$ and $\mathsf{F}(p \wedge (\mathsf{X}q \vee \mathsf{X}\neg q))$ are logically equivalent, but only the former has $\langle\{p\}\rangle$ as an informative model.

According to Kupferman and Vardi (Kupferman and Vardi 2001), an LTL formula is intentionally safe (resp., intentionally cosafe) if it specifies a safety (resp., cosafety) property and all of its bad (resp., good) prefixes are informative for the formula. As an example, $\mathsf{F}(p)$ is intentionally cosafe, because every good prefix includes a position where p holds. Conversely, $\mathsf{F}(p \wedge (\mathsf{X}q \vee \mathsf{X}\neg q))$ is *not* intentionally cosafe, because $\langle\{p\}\rangle$ is a good prefix (all extensions satisfy the formula), but it is not informative.

This paper makes three main contributions:

First, we prove that for all intentionally cosafe (resp., safe) LTL or LTLf formulas, monitoring can be performed in an *automata-less* fashion. Specifically, the monitoring task can be reduced to checking whether the formula is satisfied (resp., violated) by the current trace (*trace-checking* problem), which has two key advantages: (i) it does not require automaton construction, and (ii) it can be solved in time $\mathcal{O}(|\sigma| \cdot |\phi|)$, where σ is the trace and ϕ is the formula.

Second, we show that in the case of the $\mathsf{G}(\mathsf{pLTL})$ and $\mathsf{F}(\mathsf{pLTL})$ fragments of LTLf (*i.e.*, interpreted over finite words), every formula is intentionally safe and intentionally cosafe, respectively. This result enables monitoring of formulas in these fragments via trace-checking without explicitly verifying the intentionally (co)safety condition.

Third, we present an algorithm to decide whether a formula in $\mathsf{G}(\mathsf{pLTL})$ (resp., $\mathsf{F}(\mathsf{pLTL})$), interpreted over infinite words, is intentionally safe (resp., intentionally cosafe). This serves as a prerequisite for enabling the monitoring of such formulas without the construction of an automaton–that is, through trace-checking, as described in Section 3. Our main technical result is that this problem is in PSPACE, as the algorithm operates in nondeterministic polynomial space. This improves upon the known complexity of deciding intentionally (co)safety for full LTL, which is EXPSPACE (Kupferman and Vardi 2001).

The paper is structured as follows. Section 2 provides background knowledge. Section 3 shows the reduction of monitoring to trace-checking for intentionally (co)safe formulas, and it proves that all formulas in $\mathsf{G}(\mathsf{pLTL})$ and $\mathsf{F}(\mathsf{pLTL})$, interpreted over finite words, are intentionally safe and intentionally cosafe, respectively. Section 4 presents a PSPACE algorithm to check whether $\mathsf{G}(\mathsf{pLTL})$ and $\mathsf{F}(\mathsf{pLTL})$ formulas over infinite words are intentionally (co)safe. Finally, Section 5 summarizes the achieved results and outlines directions for future work. The proofs of all the theorems are reported in (Brunello et al. 2025a, Section E).

2 Background

In this section, we provide the necessary background.

Let $\Sigma = \{a, b, c, \ldots\}$ be an alphabet, *i.e.*, a finite set of symbols. A *finite word* over Σ is defined as a finite (possibly empty) sequence of symbols in Σ. The empty word is denoted by ε. An *infinite word* over Σ is an infinite sequence of symbols in Σ. The set of all finite words over Σ (including the empty one) is denoted by Σ^*, while the set of all infinite words over Σ is denoted by Σ^ω. Additionally, we define $\Sigma^+ := \Sigma^* \setminus \{\varepsilon\}$. A *language of finite words* is a subset $\mathcal{L} \subseteq \Sigma^*$. Similarly, a *language of infinite words* is a subset $\mathcal{L} \subseteq \Sigma^\omega$. The complement of a language $\mathcal{L}$ is denoted by $\overline{\mathcal{L}}$. Henceforth, the term *property* will be used as a synonym for language. The *length* of a word σ is defined as follows: $|\sigma| = 0$ if $\sigma = \varepsilon$; $|\sigma| = n$ if $\sigma = \langle \sigma_0, \ldots, \sigma_{n-1} \rangle \in \Sigma^*$; and $|\sigma| = \omega$ if $\sigma \in \Sigma^\omega$. Given $\sigma \in \Sigma^+ \cup \Sigma^\omega$, and a position $i \in \{0, \ldots, |\sigma| - 1\}$, we define $\sigma_{[0,i]}$ as the prefix of σ up to position i. We denote with $\sigma \cdot \sigma'$ the concatenation of the finite word σ with the finite or infinite word σ'.

Linear Temporal Logic

We begin by introducing the syntax of *Linear Temporal Logic* (LTL) (Pnueli 1977) and *Linear Temporal Logic over finite words* (LTLf) (De Giacomo and Vardi 2013). The two logics share the grammar. From now on, let $\mathcal{AP} = \{p, q, r, \ldots\}$ be a set of atomic propositions.

Definition 1. *A formula ϕ of* LTL *or* LTLf *over $\mathcal{AP}$ is inductively defined as follows:*

$$\begin{aligned}\phi :=& p \mid \neg p \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \\ & \mathsf{X}\phi \mid \widetilde{\mathsf{X}}\phi \mid \phi_1 \,\mathsf{U}\, \phi_2 \mid \phi_1 \,\mathsf{R}\, \phi_2 \mid \\ & \mathsf{Y}\phi \mid \widetilde{\mathsf{Y}}\phi \mid \phi_1 \,\mathsf{S}\, \phi_2 \mid \phi_1 \,\mathsf{T}\, \phi_2\end{aligned}$$

Temporal modalities X, $\widetilde{\mathsf{X}}$, U, and R are called *future* modalities. Similarly, temporal modalities Y, $\widetilde{\mathsf{Y}}$, S, and T are called *past* modalities. The *size* of ϕ, denoted by $|\phi|$, is defined as the number of its symbols.

Let us now define the semantics of LTL and LTLf. LTL formulas are interpreted over infinite words over the alphabet $2^{\mathcal{AP}}$, *i.e.*, over words $\sigma \in (2^{\mathcal{AP}})^\omega$, while LTLf formulas are interpreted over finite and nonempty words over the alphabet $2^{\mathcal{AP}}$, *i.e.*, over words $\sigma \in (2^{\mathcal{AP}})^+$.

The satisfaction of a formula ϕ of LTL (resp., LTLf) by an infinite word (resp., a finite, nonempty word) $\sigma =$

[1] In (Brunello et al. 2025a, Section C), we explain why the problem of determining whether a formula is intentionally (co)safe is not equivalent to removing tautological subformulas from the original formula, as the example above might misleadingly suggest.

$\langle \sigma_0, \sigma_1, \ldots \rangle$ at position i, denoted by $\sigma, i \models \phi$, is inductively defined as follows:

- $\sigma, i \models p$ (resp., $\neg p$) iff $p \in \sigma_i$ (resp., $p \notin \sigma_i$);
- $\sigma, i \models \phi_1 \vee \phi_2$ iff $\sigma, i \models \phi_1$ or $\sigma, i \models \phi_2$;
- $\sigma, i \models \phi_1 \wedge \phi_2$ iff $\sigma, i \models \phi_1$ and $\sigma, i \models \phi_2$;
- $\sigma, i \models \mathsf{X}\phi$ iff $i + 1 < |\sigma|$ and $\sigma, i + 1 \models \phi$;
- $\sigma, i \models \widetilde{\mathsf{X}}\phi$ iff $i + 1 = |\sigma|$ or $\sigma, i + 1 \models \phi$;
- $\sigma, i \models \phi_1 \,\mathsf{U}\, \phi_2$ iff $\exists i \leq j < |\sigma| \,.\, (\sigma, j \models \phi_2 \wedge \forall i \leq k < j \,.\, \sigma, k \models \phi_1)$;
- $\sigma, i \models \phi_1 \,\mathsf{R}\, \phi_2$ iff $(\forall j \geq i \,.\, \sigma, j \models \phi_2) \vee (\exists k \geq i \,.\, \sigma, k \models \phi_1 \wedge \forall i \leq j \leq k \,.\, \sigma, j \models \phi_2)$;
- $\sigma, i \models \mathsf{Y}\phi$ iff $i > 0$ and $\sigma, i - 1 \models \phi$;
- $\sigma, i \models \widetilde{\mathsf{Y}}\phi$ iff $i = 0$ or $\sigma, i - 1 \models \phi$;
- $\sigma, i \models \phi_1 \,\mathsf{S}\, \phi_2$ iff $\exists j \leq i \,.\, (\sigma, j \models \phi_2 \wedge \forall j < k \leq i \,.\, \sigma, k \models \phi_1)$;
- $\sigma, i \models \phi_1 \,\mathsf{T}\, \phi_2$ iff $(\forall 0 \leq j \leq i \,.\, \sigma, j \models \phi_2) \vee (\exists k \leq i \,.\, \sigma, k \models \phi_1 \wedge \forall i \geq j \geq k \,.\, \sigma, j \models \phi_2)$.

We write $\sigma \models \phi$ to denote $\sigma, 0 \models \phi$. In such a case, we say that σ is a *model* of ϕ. The language of an LTL formula ϕ over $\mathcal{AP}$, denoted by $\mathcal{L}(\phi)$, is the set $\{\sigma \in (2^{\mathcal{AP}})^\omega \mid \sigma \models \phi\}$. Similarly, the language of an LTLf formula ϕ over $\mathcal{AP}$, denoted by $\mathcal{L}^{<\omega}(\phi)$, is defined as the set $\{\sigma \in (2^{\mathcal{AP}})^+ \mid \sigma \models \phi\}$. Two LTL (resp., LTLf) formulas ϕ, ϕ' are *equivalent* if and only if $\mathcal{L}(\phi) = \mathcal{L}(\phi')$ (resp., $\mathcal{L}^{<\omega}(\phi) = \mathcal{L}^{<\omega}(\phi')$).

Fragments of LTL and LTLf. A number of derived modalities can be defined by using the basic ones of LTL and LTLf. Among them, we would like to remind the following ones: (i) $\top := p \vee \neg p$; (ii) $\mathsf{F}\phi := \top \,\mathsf{U}\, \phi$; (iii) $\mathsf{G}\phi := \bot \,\mathsf{R}\, \phi$; (iv) $\mathsf{O}\phi := \top \,\mathsf{S}\, \phi$; (v) $\mathsf{H}\phi := \bot \,\mathsf{T}\, \phi$. Temporal modalities F, G, O, and H are commonly referred to as *eventually*, *globally*, *once*, and *historically*, respectively. Here is an intuitive account of their semantics:

- $\mathsf{F}(p)$ (resp., $\mathsf{O}(p)$) reads *"there exists a time point in the future (resp., past) where p holds"*;
- $\mathsf{G}(p)$ (resp., $\mathsf{H}(p)$) forces each position in the future (resp., past) to contain p.

In the following, we consider three specific fragments of LTL and LTLf, which are characterized by the use of pure past temporal formulas. We define the *pure past fragment* of LTL (and LTLf), denoted by pLTL, as the set of LTL (and LTLf) formulas that do not contain any future temporal modality.

Pure past formulas ϕ of pLTL are interpreted at the *last* time position of a finite, nonempty word, and we write $\sigma \models \phi$ if and only if $\sigma, n \models \phi$ where $n = |\sigma| - 1$. The language of a pLTL formula ϕ is defined as $\mathcal{L}^{<\omega}(\phi) = \{\sigma \in (2^{\mathcal{AP}})^+ \mid \sigma \models \phi\}$. As an example, the pLTL formula $\mathsf{Y}\top$ is satisfied by all the finite, nonempty words of length at least two.

On the basis of the pure past fragment of LTL, as well as of LTLf, the logics F(pLTL) and G(pLTL) (Chang, Manna, and Pnueli 1992; Artale et al. 2023) are defined as follows.

Definition 2. F(pLTL) *is the set of formulas of the form* $\mathsf{F}(\psi)$ *where* ψ *is a* pLTL *formula.* G(pLTL) *is the set of formulas of the form* $\mathsf{G}(\psi)$ *where* ψ *is a* pLTL *formula.*

As an example, the formula $\mathsf{F}(p \wedge \mathsf{Y}\mathsf{H}q)$ asks for the existence of a time point in the future where p holds, and forces q to be true in the prefix of the word up to that point.

Safety and Cosafety Properties

Safety properties state the absence of undesirable behaviors, typically phrased as *"something bad never happens"*, e.g., the absence of deadlocks. Dually, cosafety properties guarantee that *"something good will eventually happen"*, such as the reachability of a target state in a planning problem.

The definition of safety languages of infinite or finite, nonempty words is as follows.

Definition 3 (Safety properties)**.** *A language* $\mathcal{L} \subseteq \Sigma^\omega$ *(resp.,* $\mathcal{L} \subseteq \Sigma^+$*) is* safety *if, for all* $\sigma \notin \mathcal{L}$*, there exists a position* $i \geq 0$ *(resp.,* $i \in \{0, \ldots, |\sigma| - 1\}$*) such that* $\sigma_{[0,i]} \cdot \sigma' \notin \mathcal{L}$*, for all* $\sigma' \in \Sigma^\omega$ *(resp.,* $\sigma' \in \Sigma^*$*). The prefix* $\sigma_{[0,i]}$ *is called a* bad prefix *for* $\mathcal{L}$*.*

Cosafety languages are defined dually: every word in the language must have a *good prefix*, *i.e.*, a prefix such that all its possible continuations belong to the language.

Definition 4 (Cosafety properties)**.** *A language* $\mathcal{L} \subseteq \Sigma^\omega$ *(resp.,* $\mathcal{L} \subseteq \Sigma^+$*) is* cosafety *if, for all* $\sigma \in \mathcal{L}$*, there exists a position* $i \geq 0$ *(resp.,* $i \in \{0, \ldots, |\sigma| - 1\}$*) such that* $\sigma_{[0,i]} \cdot \sigma' \in \mathcal{L}$*, for all* $\sigma' \in \Sigma^\omega$ *(resp.,* $\sigma' \in \Sigma^*$*). The prefix* $\sigma_{[0,i]}$ *is called a* good prefix *for* $\mathcal{L}$*.*

Given a formula ϕ of LTL over the set of atomic propositions $\mathcal{AP}$, we say that ϕ is a *safe* (resp., *cosafe*) formula if and only if $\mathcal{L}(\phi)$ is a safety (resp., cosafety) language over the alphabet $2^{\mathcal{AP}}$. The same applies to the case of LTLf, considering $\mathcal{L}^{<\omega}(\phi)$ in place of $\mathcal{L}(\phi)$.

It is immediate to observe that a language $\mathcal{L}$ is safety if and only if its complement $\overline{\mathcal{L}}$ is cosafety. Owing to this duality, cosafety properties are often preferred in practical applications, as they are typically easier to handle.

In (Chang, Manna, and Pnueli 1992), it is proved that G(pLTL) (resp., F(pLTL)), when interpreted on infinite words, captures exactly the set of all safety languages (resp., all cosafety languages) that are definable in LTL. The same holds under finite-word interpretation (Cimatti et al. 2023).

Proposition 1. *Let* $\mathcal{L} \subseteq \Sigma^\omega$ *be a language definable in* LTL*. It holds that* $\mathcal{L}$ *is safety (resp., cosafety) if and only if* $\mathcal{L} = \mathcal{L}(\phi)$*, for some formula* $\phi \in$ G(pLTL) *(resp., for some formula* $\phi \in$ F(pLTL)*). The same holds under finite-word interpretation.*

Automata on Finite Words

We recall the standard definition of a Deterministic Finite Automaton (DFA (Hopcroft, Motwani, and Ullman 2001)).

Definition 5. *A* Deterministic Finite Automaton *(*DFA*) is a tuple* $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$*, where (i)* Q *is a finite set of states; (ii)* Σ *is a finite alphabet; (iii)* $q_0 \in Q$ *is the initial state; (iv)* $\delta : Q \times \Sigma \to Q$ *is the transition function; and (v)* $F \subseteq Q$ *is the set of final states.*

Given a DFA $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$, we say that a word $\sigma \in \Sigma^*$ is *accepted* by $\mathcal{A}$ if *the* state reached by $\mathcal{A}$ reading σ is final. The language of $\mathcal{A}$, denoted by $\mathcal{L}(\mathcal{A})$, is the set of

words accepted by $\mathcal{A}$. We say that two DFAs $\mathcal{A}$ and $\mathcal{A}'$ are equivalent if $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$. The size of $\mathcal{A}$, written $|\mathcal{A}|$, is defined as the cardinality of Q.

All formulas in LTLf and pLTL can be effectively translated into a DFA that recognizes the same language. However, there is a fundamental difference between the two settings: while for LTLf the size of the minimal equivalent DFA is subject to a doubly-exponential lower bound in the size of the formula (De Giacomo and Vardi 2013), for pLTL there exist algorithms that allow one to build an equivalent DFA whose size is only exponential in the size of the formula (De Giacomo et al. 2021).

Proposition 2. *For every formula ϕ of* LTLf*, there exists a* DFA *$\mathcal{A}_\phi$ such that $\mathcal{L}^{<\omega}(\phi) = \mathcal{L}(\mathcal{A}_\phi)$ and $|\mathcal{A}_\phi| \in 2^{2^{poly(|\phi|)}}$. For every formula ϕ of* pLTL*, there exists a* DFA *$\mathcal{A}_\phi$ such that $\mathcal{L}^{<\omega}(\phi) = \mathcal{L}(\mathcal{A}_\phi)$ and $|\mathcal{A}_\phi| \in 2^{poly(|\phi|)}$.*

Monitoring

Monitoring is a lightweight runtime verification technique (Leucker and Schallhart 2009), which involves generating a monitor to analyze an execution of the system under consideration either online (*i.e.*, during runtime) or offline (*e.g.*, by processing the system's logs). The monitor outputs either an inconclusive result (denoted by *?*) or a definitive verdict: a violation (resp., satisfaction) if all possible continuations of the observed execution are bad (resp., good).

The *monitor for a language of infinite words* $\mathcal{L} \subseteq \Sigma^\omega$ (resp., *for a language of finite words* $\mathcal{L} \subseteq \Sigma^+$) is defined as a function $\mathrm{mon}_\mathcal{L} : \Sigma^* \to \{\top, \bot, ?\}$ such that, for all $\sigma \in \Sigma^*$,

$$\mathrm{mon}_\mathcal{L}(\sigma) := \begin{cases} \top \text{ iff } \forall \sigma' \in \Sigma^\omega \text{ (resp., } \in \Sigma^*) : \sigma \cdot \sigma' \in \mathcal{L} \\ \bot \text{ iff } \forall \sigma' \in \Sigma^\omega \text{ (resp., } \in \Sigma^*) : \sigma \cdot \sigma' \notin \mathcal{L} \\ ? \text{ otherwise} \end{cases}$$

When the monitor returns $\top$, it indicates that all possible continuations satisfy the property $\mathcal{L}$; this corresponds to the case of cosafety languages. Conversely, if the monitor returns $\bot$, it means that an irremediable violation of $\mathcal{L}$ has occurred, as in the case of safety properties. Given an LTL or LTLf formula ϕ, we will denote by mon_ϕ the monitor $\mathrm{mon}_{\mathcal{L}(\phi)}$. The *monitoring problem* for LTL and LTLf is the problem of building a monitor mon_ϕ for a given LTL or LTLf formula ϕ.

Given a temporal formula ϕ, the classical approach to monitoring involves building one DFA for the good prefixes of ϕ and one for its bad prefixes. The monitor basically consists of these two automata: as it reads the input word, *i.e.*, the system execution, it produces the output $\top$, if the automaton for the good prefixes has reached one of its final states, or $\bot$, if the automaton for the bad prefixes has reached one of its final states. In all other cases, the output is *?*.

3 Automata-less Monitoring

In this section, we identify the conditions under which monitoring of LTL and LTLf formulas can be reduced to the *trace-checking problem*–that is, determining whether σ is a model of ϕ, where σ represents the system trace and ϕ is a formula. This reduction eliminates the need to construct a deterministic automaton for ϕ, a process that incurs doubly-exponential complexity in the worst case (Proposition 2).

To establish these conditions, we utilize the classification of safety properties proposed in (Kupferman and Vardi 2001), based on the notion of *intentionally safe formulas*, and extend this notion to *intentionally cosafe formulas*.

Intentionally (Co)safe Formulas

Intuitively, a finite word $\sigma \in (2^{\mathcal{AP}})^+$ is an informative model for a formula ϕ over $\mathcal{AP}$ if it contains all the information necessary to determine the satisfaction of ϕ by σ. For instance, consider the LTL formula $\phi := p \wedge \mathsf{X}p$ and the finite word $\sigma := \langle\{p\}\rangle$. Although σ satisfies p at the first (and only) position, it is not sufficient to conclude the satisfaction of ϕ over an infinite extension of σ. Indeed, σ can be extended by a trace beginning with a position in which p is false, yielding an infinite word that violates ϕ. The same considerations hold also for formulas of type $\mathsf{G}(\psi)$ and for the case of LTLf formulas. Below, we provide the definition of an informative model.

Definition 6 (Informative model in LTL and LTLf)**.** *Let $\sigma := \langle\sigma_0, \ldots, \sigma_{n-1}\rangle \in (2^{\mathcal{AP}})^+$ be a finite word over $2^{\mathcal{AP}}$ and let ϕ be a formula of* LTL *and* LTLf *over $\mathcal{AP}$. For all $0 \leq i < n$, we define $\sigma, i \models_B \phi$ as follows:*

1. *$\sigma, i \models_B \widetilde{\mathsf{X}}\psi$ iff $\sigma, i \models_B \mathsf{X}\psi$;*
2. *$\sigma, i \models_B \psi_1 \,\mathsf{R}\, \psi_2$ iff $\sigma, i \models_B \psi_2 \,\mathsf{U}\, (\psi_1 \wedge \psi_2)$;*
3. *all the other cases are defined as in the definition of $\models$, up to replacing $\models$ with $\models_B$.*

We write $\sigma \models_B \phi$ iff $\sigma, 0 \models_B \phi$. We say that σ is an informative model for *ϕ iff $\sigma \models_B \phi$.*

In (Brunello et al. 2025a, Section D), we discuss about the differences between Definition 6 and the original definition of informative model given in (Kupferman and Vardi 2001), and we show that they are equivalent, up to considering ϕ in place of $\neg\phi$ and the use of $\models_B$ (which simplifies the proofs of the results) in place of the labeling function L.

Examples. Consider the word $\sigma := \langle\{p\}\rangle$. It holds that σ is an informative model for the formula $\mathsf{F}(p)$ because in the first position p is true. However, σ is *not* an informative model for the formula $\mathsf{F}(p \wedge (\mathsf{X}q \vee \mathsf{X}\neg q))$ because, at the first (and only) position of σ, proposition p holds but both $\mathsf{X}q$ and $\mathsf{X}\neg q$ are false due to the absence of a successor position, *i.e.*, $\sigma, 0 \not\models \mathsf{X}q \vee \mathsf{X}\neg q$ and thus $\sigma \not\models_B \mathsf{F}(p \wedge (\mathsf{X}q \vee \mathsf{X}\neg q))$. Intuitively, this means that σ does not provide sufficient information to determine the satisfaction of the formula over any of its possible extensions.

As another example, consider the trace $\sigma := \langle\{p\}, \{p\}, \{p\}\rangle$ and the formula $\phi := \mathsf{FG}(p)$, which is shorthand for $\top \,\mathsf{U}\, (\bot \,\mathsf{R}\, p)$. It holds that, for every position i in σ, we have $\sigma, i \not\models_B \bot \,\mathsf{R}\, p$, since, according to the definition of $\models_B$, this would be equivalent to requiring that $\sigma, i \models p \,\mathsf{U}\, (p \wedge \bot)$. It follows that σ is *not* an informative model for ϕ. Intuitively, the reason is that the formula $\mathsf{FG}(p)$ always requires an infinite trace to be satisfied, and no finite trace (such as σ) can contain sufficient information to determine whether it can be satisfied.

We define the *trace-checking problem* for LTL and LTLf as follows.

Definition 7 (Trace-checking problem)**.** *Let $\sigma \in (2^{\mathcal{AP}})^+$ be a word and let ϕ be an* LTL *(or* LTLf*) formula over the set of atomic propositions $\mathcal{AP}$. The* trace-checking problem *is the problem of establishing whether $\sigma \models_B \phi$.*

The time complexity of the trace-checking problem is polynomial in the size of both the word and the formula. Specifically, given a word $\sigma \in \Sigma^*$ and a formula ϕ, the problem of deciding whether $\sigma \models_B \phi$ holds can be solved in $\mathcal{O}(|\sigma| \cdot |\phi|)$ time (Kupferman and Vardi 2001; Sistla and Clarke 1985).

Starting from the notion of informative models, Kupferman and Vardi (2001) define the concept of *intentionally safe formulas*, which we extend here below to *intentionally cosafe formulas*.

Definition 8 (Intentionally safe formulas (Kupferman and Vardi 2001))**.** *A safe formula ϕ of* LTL *(resp.,* LTLf*) is* intentionally safe *iff all bad prefixes of $\mathcal{L}(\phi)$ (resp., $\mathcal{L}^{<\omega}(\phi)$) are also informative models for $\neg\phi$.*

Definition 9 (Intentionally cosafe formulas)**.** *A cosafe formula ϕ of* LTL *(resp.,* LTLf*) is* intentionally cosafe *iff all good prefixes of $\mathcal{L}(\phi)$ (resp., $\mathcal{L}^{<\omega}(\phi)$) are also informative models for ϕ.*

Examples. The LTL formula $\phi := \mathsf{F}(p \wedge (\mathsf{X}q \vee \mathsf{X}\neg q))$ is *not* intentionally cosafe, because the finite word $\langle\{p\}\rangle$ is a good prefix (in fact, for every infinite continuation, the resulting word satisfies the formula), but it is not an informative model. The same holds for the LTLf formula $\mathsf{F}(p \wedge (\mathsf{F}q \vee \mathsf{G}\neg q))$. Moreover, by a simple duality, it is easy to check that the LTL formula $\mathsf{G}(p \vee (\mathsf{X}q \wedge \mathsf{X}\neg q))$ is *not* intentionally safe.

On the contrary, formula $\phi' := \mathsf{F}(p)$ is intentionally cosafe. Interestingly, ϕ and ϕ' are equivalent (they recognize the same language). As we will point out, this means that we *cannot* reduce monitoring of $\mathsf{F}(p \wedge (\mathsf{X}q \vee \mathsf{X}\neg q))$ to trace-checking, because, with the input trace $\sigma := \langle\{p\}\rangle$, the monitoring should output $\top$ whereas the trace-checking would return the inconclusive verdict (*?*).

In (Kupferman and Vardi 2001), the notion of informative model is employed to introduce a three-level classification of safe formulas, one of which consists of the *intentionally safe* formulas. Intuitively, this term reflects the idea that the user has accurately expressed the intended property, without introducing errors or redundant subformulas. For example, the formula $\mathsf{G}(p \vee (\mathsf{X}q \wedge \mathsf{X}\neg q))$ is not intentionally safe, as it contains the subformula $\mathsf{X}q \wedge \mathsf{X}\neg q$, which evidently reflects a user error.

In this paper, we adopt the class of intentionally (co)safe formulas from this classification as a sufficient condition for enabling *automata-less monitoring*–that is, monitoring that does not require the construction of a DFA and reduces directly to the trace-checking problem (Definition 7).

Automata-Less Monitoring via Trace-Checking

The following theorem shows that, for intentionally (co)safe formulas, the monitoring problem can be reduced to trace-checking, which admits a polynomial-time solution. Intuitively, this holds because, for such formulas, every good prefix encodes all the information necessary to determine the truth of the formula over the system trace (and *vice versa*).

Algorithm 1: Monitoring algorithm for intentionally (co)safe formulas based on trace-checking.

```
1 fun mon_cosafe(σ,φ):
2  if(σ ⊨_B φ):
3   return ⊤;
4  return ?;
```

```
1 fun mon_safe(σ,φ):
2  if(σ ⊨_B ¬φ):
3   return ⊥;
4  return ?;
```

Theorem 1. *For all intentionally cosafe (resp., intentionally safe) formulas ϕ of* LTL *and* LTLf *over $\mathcal{AP}$, and for all finite words $\sigma \in (2^{\mathcal{AP}})^+$, it holds that:*

$$\mathrm{mon}_\phi(\sigma) = \top \text{ (resp., } \bot) \quad \Leftrightarrow \quad \sigma \models_B \phi \text{ (resp., } \neg\phi)$$

Based on Theorem 1, we present in Algorithm 1 the trace-checking-based monitoring algorithm for intentionally (co)safe formulas. For intentionally cosafe formulas ϕ, the algorithm returns $\top$ if the input trace σ is such that $\sigma \models_B \phi$ (*i.e.*, a good prefix has been found); otherwise, it returns *?*. For intentionally safe formulas ϕ, the algorithm returns $\bot$ if $\sigma \models_B \neg\phi$ (*i.e.*, a bad prefix has been found); otherwise, it returns *?*.

A key observation is that the satisfaction relation $\sigma \models_B \phi$ (as well as $\sigma \models_B \neg\phi$) can be decided *without* constructing any automaton. In particular, the trace-checking procedure can be performed in time $\mathcal{O}(|\sigma| \cdot |\phi|)$. This stands in contrast to the automata-theoretic approach, where the automaton corresponding to ϕ may be of doubly-exponential size in the worst case relative to the size of ϕ.

Intentionality in $\mathsf{F}(\mathsf{pLTL})$ and $\mathsf{G}(\mathsf{pLTL})$ on Finite Words

An important question concerns the computational complexity of determining whether a given formula of LTL or LTLf is intentionally (co)safe. In (Kupferman and Vardi 2001), it is established that, in the case of LTL, this decision problem is in EXPSPACE. In the following, we demonstrate that the situation is substantially more favorable for the fragments $\mathsf{F}(\mathsf{pLTL})$ and $\mathsf{G}(\mathsf{pLTL})$. Specifically: (i) under the finite–word semantics, *every* formula in these fragments is trivially intentionally (co)safe (Theorem 2), thereby rendering the verification procedure unnecessary; (ii) under the infinite–word semantics, the problem can be decided within PSPACE (Section 4).

Theorem 2. *All formulas of $\mathsf{F}(\mathsf{pLTL})$ (resp., of $\mathsf{G}(\mathsf{pLTL})$) interpreted over finite words are intentionally cosafe (resp., intentionally safe).*

Intuitively, Theorem 2 follows from the observation that, for these fragments, the $\models_B$ semantics (Definition 6) coincides with the standard LTLf semantics.

Theorem 2 establishes that, for formulas belonging to the fragments $\mathsf{F}(\mathsf{pLTL})$ and $\mathsf{G}(\mathsf{pLTL})$ of LTLf, the "intentionally (co)safety" condition is always satisfied. Consequently, by Theorem 1, the monitoring problem for such formulas can be soundly reduced to the trace-checking problem, entirely avoiding the need for automaton construction.

4 Intentionality in $\mathsf{F}(\mathsf{pLTL})$ and $\mathsf{G}(\mathsf{pLTL})$ on Infinite Words

In this section, we present a PSPACE algorithm for deciding intentionally (co)safety for formulas in the fragments $\mathsf{F}(\mathsf{pLTL})$ and $\mathsf{G}(\mathsf{pLTL})$ under infinite-word semantics. As previously mentioned, the complexity is significantly lower than in the general case, where the best known algorithm for determining intentionally (co)safety for LTL formulas operates in EXPSPACE.

When the fragments $\mathsf{F}(\mathsf{pLTL})$ and $\mathsf{G}(\mathsf{pLTL})$ are interpreted over infinite words, the statement of Theorem 2 no longer holds. In particular, there exist formulas in $\mathsf{F}(\mathsf{pLTL})$ (resp., $\mathsf{G}(\mathsf{pLTL})$) that are not intentionally cosafe (resp., not intentionally safe). As a counterexample, consider the formula $\mathsf{F}(\mathsf{Y}p)$ over $\mathcal{AP} := \{p\}$, which expresses that *"there exists a future position whose predecessor satisfies p"*. The finite word $\langle\{p\}\rangle$ constitutes a good prefix for $\mathsf{F}(\mathsf{Y}p)$, since for every infinite continuation $\sigma' \in (2^{\mathcal{AP}})^\omega$, the concatenated word $\langle\{p\}\rangle \cdot \sigma'$ satisfies $\mathsf{F}(\mathsf{Y}p)$; indeed, the subformula $\mathsf{Y}p$ is fulfilled at the second position ($i = 1$). However, $\langle\{p\}\rangle$ is *not informative* for $\mathsf{F}(\mathsf{Y}p)$, as its unary length forces the F operator to select $i = 0$. At this position, $\mathsf{Y}p$ evaluates to false, since the predecessor position does not exist in the word $\langle\{p\}\rangle$ and thus $\langle\{p\}\rangle \not\models_B \mathsf{F}(\mathsf{Y}p)$.

The remainder of this section is devoted to the proof of the following theorem.

Theorem 3. *Checking whether a formula of $\mathsf{F}(\mathsf{pLTL})$ (resp., $\mathsf{G}(\mathsf{pLTL})$), interpreted over infinite words, is intentionally cosafe (resp., intentionally safe) is in* PSPACE.

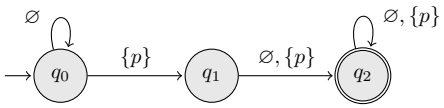
In the following, we present the algorithm for the fragment $\mathsf{F}(\mathsf{pLTL})$. The case of $\mathsf{G}(\mathsf{pLTL})$ can be handled analogously by exploiting the standard duality argument between safety and cosafety properties.[2]

The algorithm that we propose for deciding intentional cosafety of a formula of the form $\mathsf{F}(\psi)$ in $\mathsf{F}(\mathsf{pLTL})$ proceeds in two steps:

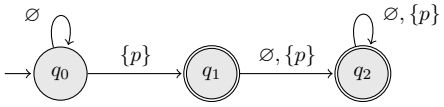
- *Step 1*. Construct a DFA $\mathcal{A}_{inf}$ that recognizes the set of finite models that are informative for $\mathsf{F}(\psi)$.
- *Step 2*. For all non-final states q of $\mathcal{A}_{inf}$, *mark q as final* iff there is a good prefix of $\mathsf{F}(\psi)$ inducing $\mathcal{A}_{inf}$ to q. Intuitively, if this is the case, then there is at least one good prefix which is *not* informative, so $\mathsf{F}(\psi)$ is not intentionally cosafe. Otherwise, $\mathsf{F}(\psi)$ is intentionally cosafe.

Conceptually, as we will discuss, this procedure implements the verification that the set of good prefixes of $\mathsf{F}(\psi)$ is included within the set of informative models of $\mathsf{F}(\psi)$ (*cf.*, Definition 9). To achieve PSPACE complexity, we argue that the second step must be carried out *on-the-fly* during the construction of the automaton in the first step.

[2]It is straightforward to show that a formula of the form $\mathsf{G}(\psi)$ is intentionally safe if and only if its negation is intentionally cosafe. Consequently, to handle formulas of the form $\mathsf{G}(\psi)$, it suffices to verify whether $\mathsf{F}(\neg\psi)$ is intentionally cosafe.



(a) The DFA $\mathcal{A}^{inf}_{\mathsf{F}(\mathsf{Y}p)}$.



(b) The DFA $\mathcal{A}^{gp}_{\mathsf{F}(\mathsf{Y}p)}$.

Figure 1: The DFAs (a) $\mathcal{A}_{inf}$ and (b) $\mathcal{A}_{gp}$ for the formula $\mathsf{F}(\mathsf{Y}p)$.

Step 1: Computing the Informative Models of $\mathsf{F}(\mathsf{pLTL})$

The first step consists of constructing a DFA, denoted $\mathcal{A}_{inf}$, that recognizes all informative models of ϕ. This construction reduces to building the DFA for the $\mathsf{F}(\mathsf{pLTL})$ formula interpreted over finite words, since, as established by the following proposition, for $\mathsf{F}(\mathsf{pLTL})$ formulas, the notion of informative models (*cf.*, Definition 6) coincides with the standard semantics of $\mathsf{F}(\mathsf{pLTL})$ over finite words.

Proposition 3. *For all formulas $\mathsf{F}(\psi)$ in $\mathsf{F}(\mathsf{pLTL})$, and for all finite words $\sigma \in (2^{\mathcal{AP}})^+$, it holds that σ is an informative model for $\mathsf{F}(\psi)$ iff $\sigma \models \mathsf{F}(\psi)$ (under finite-word semantics).*

Crucially, while for general formulas of LTL over finite words the automaton construction causes a doubly exponential blow-up (Proposition 2), for the case of $\mathsf{F}(\mathsf{pLTL})$ this can be avoided. Intuitively, this is because, under finite-word semantics, $\mathsf{F}(\psi)$ is equivalent to $\mathsf{O}(\psi)$, which is a pure past formula; thus, Proposition 2 can be leveraged to obtain an efficient automaton construction.

Proposition 4. *There exists a DFA recognizing the language $\mathcal{L}^{<\omega}(\mathsf{F}(\psi))$ of size $2^{poly(n)}$, where $n = |\mathsf{F}(\psi)|$.*

We denote by $\mathcal{A}_{inf}$ the DFA constructed in Proposition 4, which, by Proposition 3, recognizes the set of informative models of $\mathsf{F}(\psi)$ and its size is at most exponential in $|\mathsf{F}(\psi)|$.

Figure 1a depicts the DFA $\mathcal{A}_{inf}$ for the informative models of the formula $\mathsf{F}(\mathsf{Y}p)$. As previously noted, this automaton does not recognize all good prefixes. In particular, it does not accept the good prefix $\langle\{p\}\rangle$.

Step 2: Marking the States in $\mathcal{A}_{inf}$

The second step of the algorithm consists in what follows: starting from the automaton $\mathcal{A}_{inf}$, we examine each of its states q and mark q as *final* if and only if the following condition holds.

Condition 1. *For every infinite word $\sigma \in (2^{AP})^\omega$, there exists an index $i \geq 0$ such that, starting from q, the automaton $\mathcal{A}_{inf}$ reaches a final state after reading the first i symbols of σ.*

We denote by $\mathcal{A}_{gp}$ the DFA obtained in this way, that is, after all states have been examined. As an example, consider again the formula $\mathsf{F}(\mathsf{Y}p)$. Fig. 1b shows the DFA $\mathcal{A}_{gp}$ for the

Algorithm 2: A PSPACE algorithm for deciding whether $\mathsf{F}(\psi)$ is intentionally cosafe.

```
while(building the DFA A_inf)
  for(q state of A_inf)
    q' ← nondeterministically guess a
         state of A_inf
    if(q,q' are not final and q≠q' and
       q is reachable from the initial
       state)
      if(q' is reachable from q and from
         itself and all the states in
         between are non-final)
           leave q as non-final
      else
           mark q as final
```

formula $\mathsf{F}(\mathsf{Y}p)$, obtained from the DFA in Fig. 1(a) by marking as final those states that fulfill Condition 1–in this case, only the state q_1. It is easy to see that, in this example, $\mathcal{A}_{gp}$ accepts all and only the good prefixes of $\mathsf{F}(\mathsf{Y}p)$, in particular the word $\langle\{p\}\rangle$, which is not accepted by $\mathcal{A}_{inf}$.

The following lemma proves that $\mathcal{A}_{gp}$ accepts exactly the set of good prefixes of $\mathsf{F}(\psi)$.

Lemma 1. *The automaton $\mathcal{A}_{gp}$ recognizes the set of good prefixes of the formula $\mathsf{F}(\psi)$.*

Crucially, since $\mathcal{A}_{gp}$ has the same number of states and the same number of transitions as $\mathcal{A}_{inf}$, its size is the same as the size of $\mathcal{A}_{inf}$, that is, at most singly-exponential in the size of $\mathsf{F}(\psi)$ (Proposition 4).

Therefore, to check whether $\mathsf{F}(\psi)$ is intentionally cosafe, it suffices to check whether the language of $\mathcal{A}_{gp}$ is included in the language of $\mathcal{A}_{inf}$. The following proposition is a direct consequence of Lemma 1 and Definition 9.

Proposition 5. *The formula $\mathsf{F}(\psi)$ is intentionally cosafe iff $\mathcal{L}(\mathcal{A}_{gp}) \subseteq \mathcal{L}(\mathcal{A}_{inf})$.*

PSPACE Complexity of the Algorithm

In this section, we demonstrate that verifying $\mathcal{L}(\mathcal{A}_{gp}) \subseteq \mathcal{L}(\mathcal{A}_{inf})$ as stated in Proposition 5 can be achieved within the PSPACE complexity class, by circumventing the explicit construction of $\mathcal{A}_{gp}$ in the first instance. Specifically, we will show that it suffices to check whether a state of $\mathcal{A}_{inf}$ satisfies Condition 1 *on-the-fly* with the construction of $\mathcal{A}_{inf}$. Intuitively, this follows from the subsequent two observations:

(i) Since both automata are *deterministic* and given that $\mathcal{A}_{gp}$ is derived from $\mathcal{A}_{inf}$ by eventually designating a subset of its states as final, marking a non-final state (which is also reachable from the initial state) as final implies that $\mathcal{L}(\mathcal{A}_{gp}) \not\subseteq \mathcal{L}(\mathcal{A}_{inf})$.

(ii) Verifying Condition 1 for a state q is equivalent to not finding a lasso-shaped path starting from q and visiting only non-final states. Therefore, Condition 1 reduces to a reachability problem, which can be performed on-the-fly in *nondeterministic logarithmic space* (NL).

The pseudocode implementing the steps outlined above is presented in Algorithm 2. Intuitively, it checks whether Condition 1 holds for a given state q of $\mathcal{A}_{inf}$ by verifying the existence of a lasso-shaped path starting from q that visits only non-final states. If such a path exists, then Condition 1 is violated and q is left as non-final. Otherwise, q is marked as a new final state.

The algorithm in Condition 1 runs in polynomial space (PSPACE). While constructing $\mathcal{A}_{inf}$–which, according to Proposition 4, has a size at most exponential in $|\mathsf{F}(\psi)|$–we can simultaneously execute a reachability check (in NL) to determine if at least one state of $\mathcal{A}_{inf}$ satisfies Condition 1. If such a state exists, then $\mathcal{L}(\mathcal{A}_{gp}) \not\subseteq \mathcal{L}(\mathcal{A}_{inf})$, and by Proposition 5, the formula $\mathsf{F}(\psi)$ is not intentionally cosafe. Conversely, after examining all states of $\mathcal{A}_{inf}$ without finding any state satisfying Condition 1, we can conclude that $\mathsf{F}(\psi)$ is intentionally cosafe. This approach yields a PSPACE complexity for the algorithm.

Lemma 2. *The algorithm in Algorithm 2 decides whether $\mathsf{F}(\psi)$ is intentionally cosafe in nondeterministic polynomial space.*

Since by Savitch's Theorem (Savitch 1970), NPSPACE = PSPACE, we have that the problem of deciding whether a formula of $\mathsf{F}(\mathsf{pLTL})$ (resp., of $\mathsf{G}(\mathsf{pLTL})$) is intentionally cosafe (resp., intentionally safe) is in PSPACE, proving Theorem 3. This is in sharp contrast with the EXPSPACE upper bound proven in (Kupferman and Vardi 2001) for deciding intentionally (co)safe formulas in full LTL.

5 Conclusions and Future Work

In this paper, we exploited the notions of intentionally (co)safe formulas, introduced in (Kupferman and Vardi 2001), to reduce the monitoring problem to trace-checking, which is solvable in polynomial time. We showed that all formulas in the fragments $\mathsf{F}(\mathsf{pLTL})$ and $\mathsf{G}(\mathsf{pLTL})$ over finite words are intentionally cosafe and intentionally safe, respectively, and we gave a PSPACE algorithm to check intentionality in these fragments over infinite words, improving the best-known doubly exponential upper bound for full LTL.

This work yields several implications for the field of runtime verification. As a matter of fact, by Proposition 1, any (co)safety property expressible in LTL or LTLf can be equivalently rewritten in the form $\mathsf{G}(\mathsf{pLTL})$ or $\mathsf{F}(\mathsf{pLTL})$, respectively. In the case of LTL, intentional (co)safety can be verified using Algorithm 2: if the formula satisfies this condition, automata-less monitoring is applicable. For finite words, our result establishes that all formulas within the $\mathsf{F}(\mathsf{pLTL})$ and $\mathsf{G}(\mathsf{pLTL})$ fragments are inherently intentionally (co)safe. Consequently, monitoring can be performed directly through trace-checking, in an automata-less fashion. In fact, in (Brunello et al. 2025b), we demonstrated that replacing the monitoring algorithm with a trace-checking approach yields a substantial performance improvement in a framework combining monitoring and machine learning for early failure detection. In this framework, past system executions are analyzed using a variation of $\mathsf{F}(\mathsf{pLTL})$/$\mathsf{G}(\mathsf{pLTL})$ formulas interpreted over finite words (specifically, their real-time extension, *i.e.*, fragments of STL).

As for future work, identifying syntactic fragments of LTL where all formulas are intentionally (co)safe remains a significant open problem.

Ethical Statement

This manuscript reflects only the authors' views and opinions, neither the European Union nor the European Commission can be considered responsible for them.

Acknowledgments

The authors acknowledge the support from the 2024 Italian INdAM-GNCS project "Certificazione, monitoraggio, ed interpretabilità in sistemi di intelligenza artificiale", ref. no. CUP E53C23001670001. Luca Geatti, Angelo Montanari and Nicola Saccomanno acknowledge the support from the Interconnected Nord-Est Innovation Ecosystem (iNEST), which received funding from the European Union Next-GenerationEU (PIANO NAZIONALE DI RIPRESA E RESILIENZA (PNRR) – MISSIONE 4 COMPONENTE 2, INVESTIMENTO 1.5 – D.D. 1058 23/06/2022, ECS00000043). In addition, Angelo Montanari acknowledges the support from the MUR PNRR project FAIR - Future AI Research (PE00000013) also funded by the European Union Next-GenerationEU.

References

Artale, A.; Geatti, L.; Gigante, N.; Mazzullo, A.; and Montanari, A. 2023. LTL over Finite Words Can Be Exponentially More Succinct Than Pure-Past LTL, and vice versa. In Artikis, A.; Bruse, F.; and Hunsberger, L., eds., *30th International Symposium on Temporal Representation and Reasoning, TIME 2023, September 25-26, 2023, NCSR Demokritos, Athens, Greece*, volume 278 of *LIPIcs*, 2:1–2:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Bauer, A.; Leucker, M.; and Schallhart, C. 2011. Runtime verification for LTL and TLTL. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 20(4): 1–64.

Brunello, A.; Geatti, L.; Montanari, A.; and Saccomanno, N. 2025a. Automata-less Monitoring via Trace-Checking (Extended Version). arXiv:2511.11072.

Brunello, A.; Geatti, L.; Montanari, A.; and Saccomanno, N. 2025b. Interpretable Early Failure Detection via Machine Learning and Trace Checking-based Monitoring. *CoRR*, abs/2508.17786.

Chang, E. Y.; Manna, Z.; and Pnueli, A. 1992. Characterization of Temporal Property Classes. In Kuich, W., ed., *Proceedings of the 19th International Colloquium on Automata, Languages and Programming*, volume 623 of *Lecture Notes in Computer Science*, 474–486. Springer.

Cimatti, A.; Geatti, L.; Gigante, N.; Montanari, A.; and Tonetta, S. 2023. A first-order logic characterization of safety and co-safety languages. *Log. Methods Comput. Sci.*, 19(3).

De Giacomo, G.; Di Stasio, A.; Fuggitti, F.; and Rubin, S. 2021. Pure-past linear temporal and dynamic logic on finite traces. In *IJCAI'21*, 4959–4965.

De Giacomo, G.; and Vardi, M. Y. 2013. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *IJCAI'13*, 854–860.

Hopcroft, J. E.; Motwani, R.; and Ullman, J. D. 2001. Introduction to automata theory, languages, and computation. *Acm Sigact News*, 32(1): 60–65.

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

Leucker, M.; and Schallhart, C. 2009. A brief account of runtime verification. *The Journal of Logic and Algebraic Programming*, 78(5): 293–303.

Pnueli, A. 1977. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, 46–57. IEEE.

Savitch, W. J. 1970. Relationships Between Nondeterministic and Deterministic Tape Complexities. *J. Comput. Syst. Sci.*, 4(2): 177–192.

Sistla, A. P.; and Clarke, E. M. 1985. The complexity of propositional linear temporal logics. *Journal of the ACM (JACM)*, 32(3): 733–749.