Safety and Liveness on Finite Words

Luca Geatti
University of Udine, Italy

Stefano Pessotto
University of Udine, Italy

Stefano Tonetta
Fondazione Bruno Kessler, Italy

Abstract

The study of safety and liveness is crucial in the context of formal languages on infinite words, providing a fundamental classification of system properties. They have been studied extensively as fragments for regular languages and Linear Temporal Logic (LTL), both from the theoretical and practical point of view, especially in the context of model checking. In contrast, despite the growing interest in Linear Temporal Logic over finite traces (LTLf) as a specification formalism for finite-length executions, the notions of safety and liveness for finite words have remained largely unexplored.

In this work, we address this gap by defining the safety and liveness fragments of languages on finite words, mirroring the definition used for infinite words. We show that safety languages are exactly those that are prefix-closed, from which a bounded model property for all safety languages follows. We also provide criteria for determining whether a given language belongs to the safety or liveness fragment and analyze the computational complexity of this classification problems. Moreover, we show that certain LTL formulas classified as safety or liveness over infinite words may not preserve this classification when interpreted over finite words, and *vice versa*. We further establish that the safety-liveness decomposition theorem – asserting that every ω-regular language can be expressed as the intersection of a safety language and a liveness language – also holds in the finite-word setting. Finally, we examine the implications of these results for the model checking problem in LTLf.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics

Keywords and phrases Safety, Liveness, Temporal Logic

Digital Object Identifier 10.4230/LIPIcs.TIME.2025.10

Related Version *Extended Version*: `https://users.dimi.uniud.it/~luca.geatti/data/time-25/time25.pdf` [14]

Funding Luca Geatti acknowledges the support from the project "ENTAIL – intEgrazioNe tra runTime verification e mAchlne Learning" – funded by the European Union – NextGenerationEU, under the PNRR- M4C2I1.5, Program "iNEST – interconnected nord-est innovation ecosystem" – Creazione e rafforzamento di "Ecosistemi dell'Innovazione per la sostenibilità" – ECS_00000043, CUP G23C22001130006 – R.S. Geatti.

1 Introduction

The concepts of *safety* and *liveness* form a fundamental classification of system properties that describe how systems behave over time. The distinction was first introduced by Leslie Lamport in his 1977 paper on proving the correctness of concurrent programs [20], where he informally characterized safety properties as those stipulating that *"nothing bad happens"*, and liveness properties as those ensuring that *"something good eventually happens"*. The formalization of these concepts was provided by Alpern and Schneider in [1], by defining safety and liveness properties as ω-regular languages (*i.e.* sets of infinite words) such that:

- safety: all violations of the property are irremediable, that is, there exists a finite prefix of each violation (bad prefix) such that all its extensions violate the property;
- liveness: there are no irremediable violations, that is, for any finite prefix, there exists a continuation that satisfies the property.

They also proved the *safety-liveness decomposition theorem* for ω-regular languages: every property on infinite words can be expressed as the intersection of a safety property and a liveness property. This result has been recently extended also to quantitative properties [16].

The classification of properties into safety and liveness has become fundamental in the formal verification of reactive systems, particularly within the context of *model checking* for Linear Temporal Logic (LTL [23]) properties, that is, the problem of checking whether all executions of a system satisfy an LTL formula. This distinction enables the application of specialized verification techniques: once a property is identified as safety or liveness, efficient algorithms tailored to each class – such as the early proof systems by Manna and Pnueli [21, 19] or `IC3` for safety properties [7, 8] and `K-Liveness` for liveness properties [11] – can be employed. Furthermore, the identification of the safety fragment of LTL has led to syntactic characterizations that capture exactly the safety properties expressible in LTL [9, 24, 10], thereby allowing one to express safety properties directly, without the need to verify whether a given formula satisfies the safety condition.

In the last decade, Linear Temporal Logic on finite words (LTLf [13]) has emerged as a useful formalism for reasoning about systems with inherently finite executions (for example, in the context of planning). LTLf preserves the syntax of standard LTL but interprets formulae over finite words. Despite the growing interest in LTLf, the adaptation of safety and liveness to finite words has received little attention. In [10], a logical characterization of the safety fragment of LTLf has been proved complete. However, most research questions remain still open.

This work addresses this gap by studying safety and liveness for languages over finite words, revisiting classical results from the infinite-word setting – such as the decomposition theorem – and establishing new results specific to the finite-word case. Our main contributions are as follows.

First, we examine key properties of the safety fragment over finite words. We show that safety languages in this setting are precisely the *prefix-closed* languages; that is, if a word belongs to the language, then all of its prefixes must also belong to it. This characterization allows us to establish a bounded model property: a safety language is non-empty if and only if it contains the empty word. Furthermore, we demonstrate that, analogous to the infinite-word setting, a regular language over finite words is a safety language if and only if its *closure* – defined as the automaton obtained by marking all states as final – is equivalent to the original automaton. We then study the complexity of deciding whether a regular language is a safety language. We prove that this problem is PSPACE-complete, both when the language is given by a Nondeterministic Finite Automaton (NFA) and when it is specified by an LTLf formula – matching the known complexity for the infinite-word setting [24].

Second, we analyze the liveness fragment of languages over finite words. We prove that, similarly to the case of ω-regular languages, a regular language is liveness if and only if its closure recognizes the universal language. This characterization allows us to derive complexity results for the liveness recognition problem: it is PSPACE-complete when the language is specified by an NFA, and in EXPSPACE when given by an LTLf formula. Moreover, we highlight a subtle but important distinction that arises when transitioning between infinite-word and finite-word semantics in temporal logic. Specifically, we show that certain formulae classified as safety (resp., liveness) under the infinite-word interpretation (LTL) are not safety (resp., liveness) under the finite-word interpretation (LTLf), and *vice versa*.

Third, we prove that every regular language can be expressed as the intersection of a safety language and a liveness language, extending the classical Alpern-Schneider decomposition to the finite-word setting.

Last but not least, we investigate the model checking of LTLf properties over safety systems – namely, systems represented by NFAs recognizing safety properties. We demonstrate that, within this context, model checking requires careful consideration. In numerous instances, formulae that are semantically meaningful and nontrivial in the infinite-word setting lead to degenerate cases in the finite-word framework. In such cases, we show that the model checking problem becomes trivial – either invariably false or reducible to a simple condition, such as verifying whether the empty word ϵ is accepted by the property, or to the model checking of a substantially simpler formula.

Related Work

In [18], Kupferman and Vardi were the first to show that the problem of determining whether an LTL formula defines a liveness language is EXPSPACE-complete. The complexity of this problem – open for over three decades – highlights that recognizing liveness is substantially more difficult when starting from LTL formulae than from NFAs. This challenge is further exacerbated by the absence of syntactic characterizations for liveness properties, in contrast to the case of safety.

In [4], Basin *et al.* address the problem of deciding whether a formula of Timed Linear Temporal Logic (TLTL) expresses a safety or a liveness property. They prove that the problem is EXPSPACE-complete for safety and 2EXPSPACE-complete for liveness, thereby effectively adding one exponential of complexity compared to the untimed LTL case.

In [3], the model checking problem for LTLf formulae is examined. It is shown that, when restricting the model checking problem to traces that both start from an initial state and end in a final state, the problem is PSPACE-complete – matching the complexity of standard LTL model checking. Conversely, when considering all traces originating from an initial state – regardless of whether they reach a final state – the problem becomes EXPSPACE-complete.

The work presented in [22] refines the safety-liveness classification of LTL properties by considering their monitorability. It focuses on runtime verification, thus considering finite words (sequence of observations) as prefixes of infinite executions. For this reason, the definitions of safety and liveness are the classical ones based on prefixes of infinite words. In contrast, our work directly defines and explores safety and liveness fragments for languages on finite words, and specifically for LTLf. This distinction is crucial as it leads to many different properties compared to the infinite word setting.

Outline of the paper

Section 2 reviews the necessary background on LTL, LTLf, and automata. Sections 3 and 4 investigate the properties and computational complexity of safety and liveness languages over finite words, respectively. Section 5 establishes the safety-liveness decomposition theorem in the finite-word setting. Section 6 examines the implications of our results for the model checking of LTLf specifications. Section 7 summarizes the achieved results and outlines directions for future work. Proofs omitted from the main text are provided in Appendix A or in the extended version of this paper [14].

2 Background

In this section, we give the necessary background.

From now on, let $\Sigma = \{a, b, c, \ldots\}$ be an alphabet, *i.e.* a finite set of symbols. A *finite word* (over Σ) is any finite, possibly empty, sequence of symbols in Σ. An *infinite word* (over Σ) is any infinite sequence of symbols in Σ. We denote with Σ^* (resp., Σ^ω) the set of all finite and possibly empty (resp., infinite) words over Σ. We denote with ε the empty word. We define the *length* of σ as $|\sigma| = 0$ if $\sigma = \varepsilon$, as $|\sigma| = n$ if $\sigma = \langle \sigma_0, \ldots, \sigma_{n-1} \rangle \in \Sigma^*$, and as $|\sigma| = \omega$ if $\sigma \in \Sigma^\omega$. A *language of finite words* $\mathcal{L}$ is a subset of Σ^*, while a *language of infinite words* $\mathcal{L}$ is a subset of Σ^ω. We denote with $\overline{\mathcal{L}}$ the complement of $\mathcal{L}$.

2.1 Linear Temporal Logic

We start by giving the syntax of *Linear Temporal Logic on finite words* (LTLf [13]) and of *Linear Temporal Logic* (LTL [23]), both of which are defined by the same grammar. From now on, let $\mathcal{AP} = \{p, q, r, \ldots\}$ be a set of atomic propositions.

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

$$\phi := \top \mid p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \mathsf{X}\phi \mid \phi_1 \,\mathsf{U}\, \phi_2$$

where $p \in \mathcal{AP}$.

The temporal operators X and U are called respectively *next* and *until*. We define the following classic shortcut operators: (i) $\bot := \neg\top$; (ii) $\phi_1 \wedge \phi_2 := \neg(\neg\phi_1 \vee \neg\phi_2)$; (iii) $\widetilde{\mathsf{X}}\phi := \neg\mathsf{X}\neg\phi_1$; (iv) $\mathsf{F}\phi := \top \,\mathsf{U}\, \phi$; (v) $\mathsf{G}\phi := \neg\mathsf{F}\neg\phi$; (vi) $\phi_1 \,\mathsf{R}\, \phi_2 := \neg((\neg\phi_1) \,\mathsf{U}\, (\neg\phi_2))$. The temporal operators $\widetilde{\mathsf{X}}$, F, G, and R are called respectively *weak next*, *eventually*, *globally*, and *release*. The *size* of ϕ is the size of its parse tree.

A notable fragment of LTLf and of LTL is SafetyLTL. Formulae in this fragment restrict the use of temporal operators to $\widetilde{\mathsf{X}}$, G and R, and allow negation only in front of atomic propositions. The syntax of SafetyLTL is presented below.

▶ **Definition 2.** *A formula ϕ of* SafetyLTL *over $\mathcal{AP}$ is inductively defined as follows:*

$$\phi := \top \mid \bot \mid p \mid \neg p \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \widetilde{\mathsf{X}}\phi \mid \mathsf{G}\phi \mid \phi_1 \,\mathsf{R}\, \phi_2$$

where $p \in \mathcal{AP}$.

We now define the semantics of LTLf and LTL. Formulae of LTLf (resp., of LTL) are interpreted over finite (resp., infinite) words. More precisely, LTLf is interpreted over finite (possibly empty) words in $(2^{\mathcal{AP}})^*$, while LTL is interpreted over infinite words in $(2^{\mathcal{AP}})^\omega$. The satisfaction of a formula ϕ of LTLf (resp., of LTL) by a finite word (resp., by an infinite word) $\sigma = \langle \sigma_0, \sigma_1, \ldots \rangle$ at position i, denoted by $\sigma, i \models \phi$, is inductively defined as follows:

- $\sigma, i \models \top$ is always true;
- $\sigma, i \models p$ iff $0 \leq i < |\sigma|$ and $p \in \sigma_i$;
- $\sigma, i \models \neg\phi$ iff $\sigma, i \not\models \phi$;
- $\sigma, i \models \phi_1 \vee \phi_2$ iff $\sigma, i \models \phi_1$ or $\sigma, i \models \phi_2$;
- $\sigma, i \models \mathsf{X}\phi$ iff $i + 1 < |\sigma|$ and $\sigma, i + 1 \models \phi$;
- $\sigma, i \models \phi_1 \,\mathsf{U}\, \phi_2$ iff $\exists j \,.\, i \leq j < |\sigma| (\sigma, j \models \phi_2 \wedge \forall k \,.\, i \leq k < j (\sigma, k \models \phi_1))$.

We write $\sigma \models \phi$ to denote $\sigma, 0 \models \phi$. The language of an LTLf formula ϕ over the set of atomic propositions $\mathcal{AP}$, denoted as $\mathcal{L}(\phi)$, is defined as the set $\{\sigma \in (2^{\mathcal{AP}})^* \mid \sigma \models \phi\}$. Similarly, the language of an LTL formula ϕ, denoted with $\mathcal{L}^\infty(\phi)$, is defined as the set $\{\sigma \in (2^{\mathcal{AP}})^\omega \mid \sigma \models \phi\}$. We say that ϕ is valid on finite words (resp., on infinite words) if and only if $\mathcal{L}(\phi) = (2^{\mathcal{AP}})^*$ (resp., $\mathcal{L}^\infty(\phi) = (2^{\mathcal{AP}})^\omega$).

We highlight an important aspect of the difference between LTLf and LTL, which will be relevant in the next section. Consider the formula $\widetilde{\mathsf{X}}\bot$. Under finite words semantics, we can use $\widetilde{\mathsf{X}}\bot$ to hook the final position of a word: for all finite words σ, it holds that $\sigma, i \models \widetilde{\mathsf{X}}\bot$ if and only if $i = |\sigma| - 1$. This allows us to express formulae like $\mathsf{G}(q)$ as $q \,\mathsf{U}\, (\widetilde{\mathsf{X}}\bot \wedge q)$, without the need of the *globally* operator. However, under infinite words semantics, the formula $\widetilde{\mathsf{X}}\bot$ is always false, and thus formulae like $q \,\mathsf{U}\, (\widetilde{\mathsf{X}}\bot \wedge q)$ are always false as well.

2.2 The safety and the liveness fragments of infinite words

Let Σ be an alphabet. Given an infinite word $\sigma \in \Sigma^\omega$, we define $\texttt{pref}(\sigma) \coloneqq \{\sigma' \in \Sigma^* \mid \exists \sigma'' \in \Sigma^\omega \text{ such that } \sigma = \sigma' \cdot \sigma''\}$.

The definition of safety language of infinite words is given as follows.

▶ **Definition 3.** *A language $\mathcal{L} \subseteq \Sigma^\omega$ is* safety *if and only if, for all $\sigma \notin \mathcal{L}$, there exists $\sigma' \in \texttt{pref}(\sigma)$ such that $\sigma' \cdot \sigma'' \notin \mathcal{L}$, for all $\sigma'' \in \Sigma^\omega$. Such prefix σ' is called a* bad prefix *for $\mathcal{L}$.*

Given a formula ϕ of LTL over the set of atomic propositions $\mathcal{AP}$, we say that ϕ is a safety formula iff $\mathcal{L}^\infty(\phi)$ is a safety language over the alphabet $2^{\mathcal{AP}}$.

In [9], it is proved that SafetyLTL, when interpreted on infinite words, captures exactly the set of all safety languages that are definable in LTL.

▶ **Proposition 4** ([9])**.** *Let $\mathcal{L} \subseteq \Sigma^\omega$ be a language definable in* LTL*. It holds that $\mathcal{L}$ is safety if and only if $\mathcal{L} = \mathcal{L}^\infty(\phi)$, for some formula $\phi \in$* SafetyLTL*.*

Liveness languages of infinite words are defined as follows.

▶ **Definition 5.** *A language $\mathcal{L} \subseteq \Sigma^\omega$ is* liveness *if and only if, for all $\sigma \in \Sigma^*$, there exists $\sigma' \in \Sigma^\omega$ such that $\sigma \cdot \sigma' \in \mathcal{L}$.*

Given a formula ϕ of LTL over the set of atomic propositions $\mathcal{AP}$, we say that ϕ is a liveness formula iff $\mathcal{L}^\infty(\phi)$ is a liveness language over the alphabet $2^{\mathcal{AP}}$.

2.3 Finite Automata

We define the classic notion of Nondeterministic Finite Automaton.

▶ **Definition 6.** *A* Nondeterministic Finite Automaton *(*NFA*) is a tuple $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ such that: (i) Q is a finite set of states; (ii) Σ is a finite alphabet; (iii) $I \subseteq Q$ is the set of initial states; (iv) $\Delta \subseteq Q \times \Sigma \times Q$ is the transition relation; and (v) $F \subseteq Q$ is the set of final states.*

Given an NFA $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$, a state $q \in Q$ and a finite word $\sigma = \langle \sigma_0, \dots, \sigma_{n-1} \rangle \in \Sigma^*$, we define $\hat{\Delta}(q, \sigma)$ as the set

$$\{q' \in Q \mid \exists \langle q_0, q_1, \dots, q_n \rangle \,.\, q_0 = q,\ q_n = q',\ (q_i, \sigma_i, q_{i+1}) \in \Delta,\ \forall 0 \leq i < n\}.$$

We say that $\mathcal{A}$ *reaches state* q' *reading* σ iff $q' \in \bigcup_{q_0 \in I} \hat{\Delta}(q_0, \sigma)$, for some $q_0 \in I$. A word $\sigma \in \Sigma^*$ is *accepted* by $\mathcal{A}$ if $\bigcup_{q \in I} \hat{\Delta}(q, \sigma) \cap F \neq \varnothing$. The language of $\mathcal{A}$, denoted with $\mathcal{L}(\mathcal{A})$, is the set of words accepted by $\mathcal{A}$. We say that two NFAs $\mathcal{A}$ and $\mathcal{A}'$ are equivalent if $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$. If a language $\mathcal{L}$ is such that $\mathcal{L} = \mathcal{L}(\mathcal{A})$, for some NFA $\mathcal{A}$, then $\mathcal{L}$ is called a *regular language*.

Let $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ be an NFA. $\mathcal{A}$ is called a *partial Deterministic Finite Automaton* (DFA) if $|I| \leq 1$ and, for all $q \in Q$ and for all $a \in \Sigma$, there exists at most one $q' \in Q'$ such that $(q, a, q') \in \Delta$.

The notion of reduced automata [2] is defined as follows.

▶ **Definition 7** (Reduced NFA). *Let $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ be an* NFA. *We say that $\mathcal{A}$ is* reduced *if, for all $q \in Q$, there exists $\sigma \in \Sigma^*$ such that $\hat{\Delta}(q, \sigma) \cap F \neq \varnothing$. We denote with $\mathcal{R}(\mathcal{A})$ the* NFA *obtained from $\mathcal{A}$ by removing all states q that do* not *satisfy the condition $\hat{\Delta}(q, \sigma) \cap F \neq \varnothing$ for any $\sigma \in \Sigma^*$.*

Clearly, it always holds that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{R}(\mathcal{A}))$. Notice that, if $\mathcal{L}(\mathcal{A}) = \varnothing$, then also all the initial states (along with all the states reachable from them) are removed from $\mathcal{R}(\mathcal{A})$, leading to a degenerate case of an automaton without initial state ($I = \varnothing$). Moreover, it is worth pointing out that every NFA can be transformed into an equivalent reduced DFA, first by applying determinization and then by removing all states that do not lead to any final state. Finally, we define the *closure* of an NFA [2] as follows.

▶ **Definition 8** (Closure automaton). *Let $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ be an* NFA. *The closure of $\mathcal{A}$, denoted with $\mathcal{C}(\mathcal{A})$, is the automaton obtained from $\mathcal{A}$ by setting all states as final,* i.e. *$(Q, \Sigma, I, \Delta, Q)$.*

The interesting property of closures in NFAs is that they reject words solely by attempting undefined transitions. This form of rejection is *irremediable*, as no subsequent extension of the input word can lead the automaton to accept it.

3 The safety fragment on finite words

In this section, we define the *safety* fragment of languages of finite words. We then study some properties of this fragment as well as some complexity-related issues. Finally we compare safety languages definable in LTLf with those definable in LTL.

3.1 Safety languages of finite words

From now on, let Σ be a finite alphabet. Given a finite word $\sigma \in \Sigma^*$, we define $\texttt{pref}(\sigma) \coloneqq \{\sigma' \in \Sigma^* \mid \exists \sigma'' \in \Sigma^* \,.\, \sigma = \sigma' \cdot \sigma''\}$. We define safety languages of finite words as follows.

▶ **Definition 9** (Safety languages of finite words). *A language $\mathcal{L} \subseteq \Sigma^*$ is* safety *if and only if, for all $\sigma \notin \mathcal{L}$, there exists $\sigma' \in \texttt{pref}(\sigma)$ such that $\sigma' \cdot \sigma'' \notin \mathcal{L}$, for all $\sigma'' \in \Sigma^*$. Such prefix σ' is called a* bad prefix *for $\mathcal{L}$.*

Given a formula ϕ of LTLf over the set of atomic propositions $\mathcal{AP}$, we say that ϕ is a safety formula if $\mathcal{L}(\phi)$ is a safety language over the alphabet $2^{\mathcal{AP}}$.

In contrast with safety languages of infinite words, in this setting we require that all the *finite* continuations of a bad prefix do not belong the language. From the definition, it immediately follows that every safety language $\mathcal{L}$ is such that $\overline{\mathcal{L}} = K \cdot \Sigma^*$, where $K \subseteq \Sigma^*$ is the set of bad prefixes.

Examples. Let $\Sigma = \{a, b\}$. The language b^* is a safety language, because every violation (*i.e.* every word not belonging to the language) contains a prefix ending with the symbol a such that all possible continuations of the prefix are violations. In this case, the set of bad prefixes is $b^* \cdot a \cdot \Sigma^*$. On the contrary, the language $b \cdot b^*$ is *not* of safety, since $\varepsilon \notin \mathcal{L}$ but it is not true that $\varepsilon \cdot \sigma' \notin \mathcal{L}$ for all $\sigma' \in \Sigma^*$. The LTLf formula $\mathsf{G}(p \rightarrow \widetilde{\mathsf{X}} q)$ recognizes a safety language, because any violating trace contains two adjacent positions where p is true in the first one and q is false in the next one. Any continuation of the trace starting from this point is a violation of the formula.

3.2 Properties of the safety fragment on finite words

First, we show that the safety condition is not limited to regular languages. In fact, there exist safety languages over finite words that are not regular.

▶ **Proposition 10.** *Let $\Sigma = \{a, b\}$ and let $\mathcal{L} = \{a^n \cdot b^n \mid n > 0\} \cdot \Sigma^*$. It holds that $\overline{\mathcal{L}}$ is safety and not regular.*

Proof. We provide the complete proof in [14]. ◀

Second, we show that safety languages over finite words are precisely those languages that are prefix-closed, *i.e.* if a word belongs to the language, then all of its prefixes are also included. The notion of prefix-closure is formally defined as follows.

▶ **Definition 11** (Prefix-closure). *Let $\mathcal{L} \subseteq \Sigma^*$. We say that $\mathcal{L}$ is* prefix-closed *if, for all $\sigma \in \mathcal{L}$, it holds that $\texttt{pref}(\sigma) \subseteq \mathcal{L}$.*

The following proposition proves that all safety languages of finite words are prefix-closed and *vice versa*. Therefore, prefix-closure provides an alternative and equivalent characterization of safety languages of finite words.

▶ **Proposition 12.** *Let $\mathcal{L} \subseteq \Sigma^*$. It holds that $\mathcal{L}$ is safety if and only if $\mathcal{L}$ is prefix-closed.*

Proof. We begin proving the left-to-right direction. Let $\mathcal{L} \subseteq \Sigma^*$ be a safety language. Suppose by contradiction that $\mathcal{L}$ is *not* prefix-closed, that is, there exists $\sigma \in \mathcal{L}$ and a prefix $\sigma' \in \texttt{pref}(\sigma)$ such that $\sigma' \notin \mathcal{L}$. Since $\mathcal{L}$ is safety, by Definition 9, it holds that there exists a prefix $\sigma'' \in \texttt{pref}(\sigma')$ of σ' such that $\sigma'' \cdot \sigma''' \notin \mathcal{L}$, for all $\sigma''' \in \Sigma^*$. In the particular case in which σ''' is the suffix of σ obtained from σ by removing its prefix σ'', we have that $\sigma'' \cdot \sigma''' = \sigma$ and $\sigma \notin \mathcal{L}$, which is a contradiction since we supposed that $\sigma \in \mathcal{L}$. Therefore, it must be that $\mathcal{L}$ is prefix-closed.

We now prove the right-to-left direction. Let $\mathcal{L} \subseteq \Sigma^*$ be a prefix-closed language. Suppose by contradiction that $\mathcal{L}$ is *not* safety, that is, there exists $\sigma \notin \mathcal{L}$ such that, for all $\sigma' \in \texttt{pref}(\sigma)$, there exists a $\sigma'' \in \Sigma^*$ such that $\sigma' \cdot \sigma'' \in \mathcal{L}$. In the particular case in which $\sigma' = \sigma$, we have that $\sigma \cdot \sigma'' \in \mathcal{L}$, for some $\sigma'' \in \Sigma^*$. But, since by hypothesis $\mathcal{L}$ is prefix-closed, all prefixes of $\sigma \cdot \sigma''$, and in particular σ, must belong to $\mathcal{L}$. However, this is a contradiction since we supposed that $\sigma \notin \mathcal{L}$. Therefore, $\mathcal{L}$ must be a safety language. ◀

As a corollary of Proposition 12, a bounded model property for safety languages over finite words follows directly, showing that any nonempty language of this kind necessarily includes the empty word (its proof is provided in Appendix A).

▶ **Corollary 13** (Small and Bounded Model Property for safety languages). *Let $\mathcal{L} \subseteq \Sigma^*$ be a safety language. It holds that $\mathcal{L} \neq \varnothing$ if and only if $\varepsilon \in \mathcal{L}$.*

Third, we show an effective way to establish whether the language recognized by a given NFA is safety. The procedure consists in checking whether the reduced version of the given NFA and its closure are equivalent.

▶ **Proposition 14.** *Let $\mathcal{A}$ be an* NFA. *$\mathcal{L}(\mathcal{A})$ is safety if and only if $\mathcal{L}(\mathcal{R}(\mathcal{A})) = \mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A})))$.*

Proof. We consider first the right-to-left direction. Suppose that $\mathcal{L}(\mathcal{R}(\mathcal{A})) = \mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A})))$ and consider the automaton $\mathcal{C}(\mathcal{R}(\mathcal{A}))$. Since the closure of any automaton, by definition, is such that all of its states are final, it is straightforward to see that its language is prefix-closed. Therefore, we have that $\mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A})))$ is prefix-closed, and so is $\mathcal{L}(\mathcal{R}(\mathcal{A}))$. Then, by Proposition 12, $\mathcal{L}(\mathcal{R}(\mathcal{A}))$ is a safety language.

To prove the left-to-right direction, notice that, since $\mathcal{C}(\mathcal{R}(\mathcal{A}))$ is obtained from $\mathcal{A}$ by setting all its states as final, it holds that $\mathcal{L}(\mathcal{R}(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A})))$. Therefore, only the inclusion $\mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A}))) \subseteq \mathcal{L}(\mathcal{R}(\mathcal{A}))$ has to be proved.

To prove that $\mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A}))) \subseteq \mathcal{L}(\mathcal{R}(\mathcal{A}))$, we divide in cases depending on whether the set I of initial states of $\mathcal{C}(\mathcal{R}(\mathcal{A}))$ is empty.

If $I = \varnothing$, then $\mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A}))) = \varnothing$ and clearly $\mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A}))) \subseteq \mathcal{L}(\mathcal{R}(\mathcal{A}))$.

If $I \neq \varnothing$, let q_0 be one of the initial states of $\mathcal{C}(\mathcal{R}(\mathcal{A}))$, let $\sigma \in \mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A})))$, and let q be one of the final states reached by $\mathcal{C}(\mathcal{R}(\mathcal{A}))$ after reading σ. Since $\mathcal{R}(\mathcal{A})$ shares with its closure the same set of states and transition relation, q is a state of $\mathcal{R}(\mathcal{A})$ and is reached by $\mathcal{R}(\mathcal{A})$ after reading σ. Moreover, since $\mathcal{R}(\mathcal{A})$ is reduced, by definition of reduced automaton, there exists a $\sigma' \in \Sigma^*$ such that $\hat{\Delta}(q, \sigma') \cap F \neq \varnothing$, and thus $\sigma \cdot \sigma' \in \mathcal{L}(\mathcal{R}(\mathcal{A}))$. Since by hypothesis $\mathcal{L}(\mathcal{R}(\mathcal{A}))$ is safety, it is also prefix-closed (Proposition 12), and thus $\sigma \in \mathcal{L}(\mathcal{R}(\mathcal{A}))$. ◀

3.3 The complexity of recognizing safety languages of finite words

Now, we investigate the complexity of determining whether a language of finite words is a safety language, depending on the form in which the language is represented – either by automata or temporal logic. In both cases, we prove that the problem is PSPACE-complete. Interestingly, this is the same complexity as for the case of infinite words [25].

▶ **Proposition 15.** *Establishing whether the language accepted by an* NFA *is safety is* PSPACE*-complete.*

Proof (sketch). The membership in PSPACE follows from Proposition 14, and from the fact that the equivalence problem of two NFAs is a PSPACE problem. For the PSPACE-hardness, we use a reduction from the *universality problem* for NFAs, which is PSPACE-complete. The complete proof is provided in Appendix A. ◀

▶ **Proposition 16.** *Establishing whether the language of an* LTLf *formula is safety is* PSPACE*-complete.*

Proof (sketch). The PSPACE upper bound follows from the singly exponential construction of equivalent NFAs starting from LTLf formulas [13] and from the fact that the equivalence problem of two NFAs is a PSPACE problem. For proving the PSPACE-hardness, we use a reduction from the LTLf validity problem, which is PSPACE-complete. The complete proof is provided in Appendix A. ◀

3.4 Comparison of the safety fragments of LTL and LTLf

In this part, we compare the safety fragments of LTLf and LTL, and observe that certain formulae in LTLf, when interpreted over infinite words, are no longer safety, and conversely, some formulae in LTL cease to be safety when interpreted over finite words. This highlights that, with respect to the safety fragment, transitioning between LTLf and LTL must be done with care.

▶ **Proposition 17.** *It holds that:*
- *there exists an* LTL *formula* ϕ *such that* $\mathcal{L}^\infty(\phi)$ *is safety but* $\mathcal{L}(\phi)$ *is* not *safety; and*
- *there exists an* LTL *formula* ϕ *such that* $\mathcal{L}(\phi)$ *is safety but* $\mathcal{L}^\infty(\phi)$ *is* not *safety.*

Proof. To prove the first point, we take $\phi \coloneqq \mathsf{G}(p \to \mathsf{X}q)$. When interpreted over finite words, the language of ϕ is *not* safety. In fact, consider the word $\sigma \coloneqq \langle\{p\}\rangle$ of length 1. It holds that $\sigma \notin \mathcal{L}(\phi)$ because there is an occurrence of p that is not followed by any q. But none of its prefixes $\sigma' \in \mathtt{pref}(\sigma)$ is such that $\sigma' \cdot \sigma'' \notin \mathcal{L}(\phi)$, for all $\sigma'' \in (2^{\mathcal{AP}})^*$. In fact, if $\sigma' = \varepsilon$ then $\sigma' \cdot \{p\} \cdot \{q\} \in \mathcal{L}(\phi)$, while if $\sigma' = \langle\{p\}\rangle$ then $\sigma' \cdot \{q\} \in \mathcal{L}(\phi)$. It follows that $\mathcal{L}(\phi)$ is not a safety language. However, it is worth pointing out that, when interpreted over infinite words, the language of ϕ is safety. In fact, ϕ belongs to the syntactic safety fragment of LTL, *i.e.* SafetyLTL, and, by Proposition 4, $\mathcal{L}^\infty(\phi)$ is a safety language of infinite words.

To prove the second point, consider the following formula: $\phi \coloneqq \mathsf{G}(q\,\mathsf{U}\,(p \wedge q) \vee q\,\mathsf{U}\,(\widetilde{\mathsf{X}}\bot \wedge q))$. Under the interpretation over finite words, ϕ is equivalent to $\mathsf{G}(q\,\mathsf{U}\,(p \wedge q) \vee \mathsf{G}q)$, which in turn is equivalent to $\mathsf{G}(p\,\mathsf{R}\,q)$. To prove that $\mathcal{L}(\mathsf{G}(p\,\mathsf{R}\,q))$ is a safety language, observe that, for all $\sigma \in (2^{\mathcal{AP}})^*$, $\sigma \not\models \mathsf{G}(p\,\mathsf{R}\,q)$ if and only if $\sigma \models \mathsf{F}(\neg p\,\mathsf{U}\,\neg q)$. Now, for each of these words σ, there exists a prefix $\sigma' \in \mathtt{pref}(\sigma)$ which does not contain a q in its last position and thus it is irremediable, that is, $\sigma' \cdot \sigma'' \models \mathsf{F}(\neg p\,\mathsf{U}\,\neg q)$, for all $\sigma'' \in (2^{\mathcal{AP}})^*$. Therefore, $\mathcal{L}(\phi)$ is a safety language. However, under the interpretation over infinite words, ϕ is equivalent to $\mathsf{G}(q\,\mathsf{U}\,(p \wedge q))$, whose language is *not* safety: the infinite word $\{q\}^\omega$ does not satisfy $\mathsf{G}(q\,\mathsf{U}\,(p \wedge q))$ because p is never true, but each of its prefixes $\langle\{q\}, \ldots, \{q\}\rangle$ can be extended to an infinite word satisfying the formula, for example, by concatenating $\{p, q\}^\omega$. ◀

4 The liveness fragment on finite words

In this section, we define the *liveness* fragment of languages of finite words. We then show an effective way to recognize whether a language is liveness and we study the complexity of the problem. Finally, we compare liveness languages definable in LTLf with those definable in LTL.

4.1 Liveness languages of finite words

We define the liveness condition for languages of finite words as follows.

▶ **Definition 18.** *A language* $\mathcal{L} \subseteq \Sigma^*$ *is* liveness *if and only if, for all* $\sigma \in \Sigma^*$*, there exists* $\sigma' \in \Sigma^*$ *such that* $\sigma \cdot \sigma' \in \mathcal{L}$*.*

Given a formula ϕ of LTLf over the set of atomic propositions $\mathcal{AP}$, we say that ϕ is a liveness formula if $\mathcal{L}(\phi)$ is a liveness language over the alphabet $2^{\mathcal{AP}}$.

Examples. The LTLf formula $\mathsf{F}(p)$ over $\mathcal{AP} = \{p\}$ recognizes a liveness language, because any finite word $\sigma \in (2^{\mathcal{AP}})^*$ can be extended to a trace in the language of $\mathsf{F}(p)$ by concatenating $\langle\{p\}\rangle$. The same holds for the formula $\mathsf{GF}(p)$ and for all formulae of type $\mathsf{F}(\psi)$, where ψ

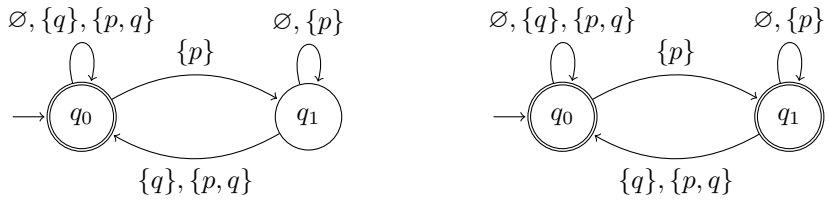

Figure 1 On the left, an automaton recognizing the LTLf formula $\mathsf{G}(p \to \mathsf{F}q)$. On the right, its closure, which accepts every finite word. By Proposition 19, the language of $\mathsf{G}(p \to \mathsf{F}q)$ fulfills the liveness condition.

is an LTLf formula. Conversely, the formula $\mathsf{G}(\neg p)$ does not recognize a liveness language, because the word $\langle\{p\}\rangle$ cannot be extended to a trace in the language of $\mathsf{G}(\neg p)$. Consider now the LTLf formula $\mathsf{F}(p \wedge \widetilde{\mathsf{X}}\bot)$ over $\mathcal{AP} = \{p\}$, stating that p holds in the final position of a word. Under finite word interpretation, we have $\mathcal{L}(\phi) = (2^{\mathcal{AP}})^* \cdot \{p\}$. Clearly, $\mathcal{L}(\phi)$ is a liveness language. Under infinite word interpretation, instead, we have that $\mathcal{L}^\infty(\phi) = \varnothing$, because there exists no word containing a position whose successor satisfies the formula $\bot$. It follows that $\mathcal{L}^\infty(\phi)$ is *not* a liveness language of infinite words.

4.2 Recognizing liveness languages of finite words

In this section, we present an effective method for determining whether a language of finite words satisfies the liveness property. We subsequently analyze the computational complexity of this decision problem in two scenarios: when the language is represented by an NFA and when it is specified using an LTLf formula.

▶ **Proposition 19.** *Let $\mathcal{A}$ be an* NFA *over the alphabet Σ. $\mathcal{L}(\mathcal{A})$ is liveness if and only if $\mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A}))) = \Sigma^*$.*

Proof. We start proving the left-to-right direction. Suppose that $\mathcal{L}(\mathcal{A})$ is liveness. Clearly, it always holds that $\mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A}))) \subseteq \Sigma^*$, thus we need only to prove that $\Sigma^* \subseteq \mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A})))$. Let $\sigma \in \Sigma^*$ be a finite word. Since $\mathcal{L}(\mathcal{A})$ is a liveness language and since $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{R}(\mathcal{A}))$, by Definition 18, there there exists $\sigma' \in \Sigma^*$ such that $\sigma \cdot \sigma' \in \mathcal{L}(\mathcal{R}(\mathcal{A}))$, *i.e.* σ is the prefix of a word accepted by $\mathcal{L}(\mathcal{R}(\mathcal{A}))$. Therefore, there exists a state q of $\mathcal{R}(\mathcal{A})$ reached by $\mathcal{R}(\mathcal{A})$ after reading σ. By definition of closure automaton, state q is final in $\mathcal{C}(\mathcal{R}(\mathcal{A}))$ and is reached by $\mathcal{C}(\mathcal{R}(\mathcal{A}))$ after reading σ. It follows that $\sigma \in \mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A})))$.

We now prove the right-to-left direction. Suppose that $\mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A}))) = \Sigma^*$. We prove that the condition of liveness is satisfied by $\mathcal{L}(\mathcal{A})$. Let $\sigma \in \Sigma^*$. Since $\mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A}))) = \Sigma^*$, it follows that there exists a state q reached by $\mathcal{C}(\mathcal{R}(\mathcal{A}))$ after reading σ. Since, $\mathcal{C}(\mathcal{R}(\mathcal{A}))$ and $\mathcal{R}(\mathcal{A})$ share the same set of states and the same transition relation, q is also a state of $\mathcal{R}(\mathcal{A})$ and is reached by $\mathcal{R}(\mathcal{A})$ after reading σ. Since $\mathcal{R}(\mathcal{A})$ is reduced, a final state of $\mathcal{R}(\mathcal{A})$ is reachable from state q, impling that there exists a word $\sigma' \in \Sigma^*$ such that $\sigma \cdot \sigma' \in \mathcal{L}(\mathcal{R}(\mathcal{A}))$, proving that the liveness condition holds for $\mathcal{L}(\mathcal{R}(\mathcal{A}))$. Since the NFAs $\mathcal{R}(\mathcal{A})$ and $\mathcal{A}$ are equivalent, this means that also $\mathcal{L}(\mathcal{A})$ is a liveness language. ◀

As an example of application of Proposition 19, we consider the LTLf formula $\phi \coloneqq \mathsf{G}(p \to \mathsf{F}q)$ over the set of atomic propositions $\mathcal{AP} \coloneqq \{p, q\}$. The automaton $\mathcal{A}_\phi$, which is equal to its reduced version $\mathcal{R}(\mathcal{A}_\phi)$, recognizing $\mathcal{L}(\phi)$ is depicted in Figure 1 (left). The right side of Figure 1 displays its closure $\mathcal{C}(\mathcal{R}(\mathcal{A}_\phi))$. Since $\mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A}_\phi))) = (2^{\mathcal{AP}})^*$, by Proposition 19, the language of $\mathsf{G}(p \to \mathsf{F}q)$ is a liveness language.

We now investigate the computational complexity of deciding whether a language satisfies the liveness condition, in the cases where the language is given by an NFA and by an LTLf formula. In the former case, the problem is PSPACE-complete, as stated by the following proposition.

▶ **Proposition 20.** *Establishing whether the language accepted by an* NFA *is liveness is* PSPACE*-complete.*

Proof (sketch). Both the membership in PSPACE (*cf.* Proposition 19) and the PSPACE-hardness follows from the universality problem of NFAs, which is PSPACE-complete. The complete proof is provided in [14]. ◀

Interestingly, the best algorithm we have devised so far for deciding whether an LTLf formula is liveness requires exponential space. This is due to the fact that, given an LTLf formula ϕ, the algorithm first constructs the NFA corresponding to ϕ, which requires exponential space in $|\phi|$, and then checks the universality of its closure (*cf.* Proposition 19), a step that requires polynomial space in the size of the automaton. Overall, the algorithm operates in exponential space with respect to $|\phi|$. Whether this algorithm is optimal (*i.e.* EXPSPACE-hard), or whether a more efficient solution exists, remains an open problem – even in the case of infinite words.

▶ **Proposition 21.** *Establishing whether the language of an* LTLf *formula is liveness is in* EXPSPACE.

Proof. Let ϕ be an LTLf formula of size n over $\mathcal{AP}$. The algorithm to check whether ϕ is liveness proceeds as follows. First, it builds the NFA $\mathcal{A}_\phi$ equivalent to ϕ, which is of size $2^{\mathcal{O}(n)}$ [13]. Then, it constructs $\mathcal{C}(\mathcal{R}(\mathcal{A}_\phi))$ and checks whether $\mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A}_\phi))) = (2^{\mathcal{AP}})^*$, in space polynomial in the size of $\mathcal{A}_\phi$, that is, $2^{\mathcal{O}(n)}$. If this is the case, then ϕ is liveness, otherwise ϕ is not liveness. Overall, the algorithm requires space exponential in n, and thus the problem is in EXPSPACE. ◀

4.3 Comparison of the liveness fragments of LTL and LTLf

In the proposition below, we show that, as in the case of the safety fragment, there exist liveness formulae of LTLf that, when interpreted over infinite words, no longer accept liveness languages. Conversely, there also exist liveness formulae of LTL that, when interpreted over finite words, do not define a liveness language.

▶ **Proposition 22.** *It holds that:*

- *there exists an* LTL *formula ϕ such that $\mathcal{L}(\phi)$ is liveness but $\mathcal{L}^\infty(\phi)$ is not liveness; and*
- *there exists an* LTL *formula ϕ such that $\mathcal{L}^\infty(\phi)$ is liveness but $\mathcal{L}(\phi)$ is not liveness.*

Proof. To prove the first point, it suffices to take the formula $\mathsf{F}(p \wedge \widetilde{\mathsf{X}}\bot)$. As we shown above, when interpreted over finite words, the formula recognize a liveness language, while over infinite words it recognizes a language which is not liveness.

To prove the second point, consider the following formula: $\phi \coloneqq \mathsf{G}((\widetilde{\mathsf{X}}\bot \wedge q) \rightarrow \mathsf{F}(\widetilde{\mathsf{X}}\bot \wedge \neg q))$. Over finite words, it holds that $\mathcal{L}(\phi) = \varnothing$, because the formula forces the final time point of any word to satisfy both q and $\neg q$. It follows that $\mathcal{L}(\phi)$ is *not* a liveness language. However, over the infinite word interpretation, we have that $\mathcal{L}^\infty(\phi) = (2^{\mathcal{AP}})^\omega$, because the antecedent of the implication is always false. Therefore, $\mathcal{L}^\infty(\phi)$ is liveness. ◀

5 Decomposition of regular languages

In this section, we present a decomposition result for regular languages that leverages the characterization of the safety and liveness fragments. Specifically, we show that for any regular language $\mathcal{L}$, one can construct a safety language and a liveness language whose intersection is precisely $\mathcal{L}$.

From this point onward, we restrict our attention to DFAs. This assumption simplifies certain proofs, most notably that of Proposition 25. This restriction comes with no loss of generality, as every regular language $\mathcal{L}$ admits an equivalent DFA.

As a first step, we give the following definitions.

▶ **Definition 23** (The `Safe` and the `Live` versions of a DFA). *Let $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ be a* DFA. *We define* $\texttt{Safe}(\mathcal{A})$ *as* $\mathcal{C}(\mathcal{R}(\mathcal{A}))$. *We define* $\texttt{Live}(\mathcal{A})$ *as the automaton* $\mathcal{R}(\mathcal{A})$ *augmented with a new final state $q_f \notin Q$, and the following transitions: (i) (q_f, a, q_f), for all $a \in \Sigma$; (ii) (q, a, q_f), for all $q \in Q$ and for all $a \in \Sigma$ such that $\Delta(q, a) = \varnothing$.*

The proposition below states that, for any DFA $\mathcal{A}$, the automata $\texttt{Safe}(\mathcal{A})$ and $\texttt{Live}(\mathcal{A})$ recognize a safety and a liveness language, respectively. We provide the proof in Appendix A.

▶ **Proposition 24.** *Let $\mathcal{A}$ be a* DFA. *It holds that $\mathcal{L}(\texttt{Safe}(\mathcal{A}))$ (resp., $\mathcal{L}(\texttt{Live}(\mathcal{A}))$) is a safety language (resp., a liveness language).*

The final step before presenting the decomposition algorithm is to show that, for any DFA $\mathcal{A}$, the automaton $\texttt{Live}(\mathcal{A})$ recognizes exactly the set of words that are either accepted by $\mathcal{A}$ or not accepted by $\texttt{Safe}(\mathcal{A})$. This property is essential to ensure that the intersection of $\texttt{Safe}(\mathcal{A})$ and $\texttt{Live}(\mathcal{A})$ recognizes precisely the language $\mathcal{L}(\mathcal{A})$.

▶ **Proposition 25.** *Let $\mathcal{A}$ be a* DFA. $\texttt{Live}(\mathcal{A})$ *recognizes the language $\mathcal{L}(\mathcal{A}) \cup (\Sigma^* \setminus \mathcal{L}(\texttt{Safe}(\mathcal{A})))$.*

Proof. We provide the proof in [14]. ◀

The following theorem states and proves the decomposition theorem for regular languages, establishing that every regular language over an alphabet Σ is expressible as an intersection of a safety language over Σ and a liveness language over Σ.

▶ **Theorem 26.** *Let $\mathcal{L} \subseteq \Sigma^*$ be a regular language. There exist two regular languages $\mathcal{L}_{safe} \subseteq \Sigma^*$ and $\mathcal{L}_{live} \subseteq \Sigma^*$ such that: (i) $\mathcal{L}_{safe}$ is a safety language; (ii) $\mathcal{L}_{live}$ is a liveness language; (iii) $\mathcal{L} = \mathcal{L}_{safe} \cap \mathcal{L}_{live}$.*

Proof. Since $\mathcal{L}$ is a regular language, there exists a DFA $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}$. Let $\mathcal{L}_{safe} = \mathcal{L}(\texttt{Safe}(\mathcal{A}))$ and $\mathcal{L}_{live} = \mathcal{L}(\texttt{Live}(\mathcal{A}))$.

By Proposition 24, $\mathcal{L}_{safe}$ and $\mathcal{L}_{live}$ are a safety language and a liveness language, respectively. Moreover, by Proposition 25, it holds that:

$$\begin{aligned}\mathcal{L}_{safe} \cap \mathcal{L}_{live} &= \mathcal{L}(\texttt{Safe}(\mathcal{A})) \cap \mathcal{L}(\texttt{Live}(\mathcal{A})) = \mathcal{L}(\texttt{Safe}(\mathcal{A})) \cap (\mathcal{L}(\mathcal{A}) \cup (\Sigma^* \setminus \mathcal{L}(\texttt{Safe}(\mathcal{A}))))\\ &= \mathcal{L}(\texttt{Safe}(\mathcal{A})) \cap \mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A})\end{aligned}$$

Therefore, $\mathcal{L} = \mathcal{L}_{safe} \cap \mathcal{L}_{live}$. ◀

6 Implications on LTLf Model Checking

In this section, we examine certain implications of the safety and liveness fragments on the *model checking* problem [12] over finite words. Specifically, we address the problem of verifying whether an NFA M satisfies an LTLf formula ϕ, *i.e.* whether $\mathcal{L}(M) \subseteq \mathcal{L}(\phi)$, denoted as $M \models \phi$.

The identification of safety and liveness fragments within both LTLf and the class of regular languages over finite words opens avenues for designing specialized model checking procedures and conducting a more granular complexity analysis – paralleling existing results in the setting of infinite words [19, 11].

However, we show that when $\mathcal{L}(M)$ is assumed to be a nonempty safety language – a common scenario in practice, particularly in invariant checking benchmarks [5] – the model checking problem for LTLf formulae often becomes uninformative. In many such cases, formulae that are semantically meaningful and nontrivial in the infinite-word setting give rise to degenerate instances in the finite-word setting, where the model checking task becomes trivial: either always false, or reducible to a simple condition such as whether the empty word ε belongs to $\mathcal{L}(M)$ or the model checking of a much simpler formula.

6.1 The cosafety case

We begin by analyzing the case where $\mathcal{L}(\phi)$ is a *cosafety* language – that is, the complement of a safety language. Consider, for instance, the formula $\phi \coloneqq \mathsf{F}p$. Since $\mathcal{L}(M)$ is assumed to be a safety language and thus is prefix-closed (*cf.* Definition 11 and Proposition 12), it necessarily contains the empty word ε. However, since ϕ is a cosafety formula, $\overline{\mathcal{L}(\phi)}$ is a safety language, and it holds that $\varepsilon \in \overline{\mathcal{L}(\phi)}$, that is, $\varepsilon \notin \mathcal{L}(\phi)$. Consequently, $\mathcal{L}(M) \not\subseteq \mathcal{L}(\phi)$, *i.e.* $M \not\models \phi$. More generally, there does not exist an NFA M such that $\mathcal{L}(M)$ is a safety, nonempty language and $M \models \phi$, with $\phi \coloneqq \mathsf{F}p$. It is worth noting that this conclusion remains valid even if one were to disregard ε from the language, in which case the model checking task reduces to a trivial condition: verifying whether the initial state satisfies p.

This reasoning extends to arbitrary cosafety properties. Let ϕ be a cosafety LTLf formula and let M be an NFA such that $\mathcal{L}(M)$ is safety and nonempty. One has that $M \models \phi$ if and only if $\mathcal{L}(M) \cap \overline{\mathcal{L}(\phi)} = \varnothing$. Since $\mathcal{L}(M)$ is safety and thus prefix-closed, by Corollary 13 we have that $\varepsilon \in \mathcal{L}(M)$, and thus:

1. if $\varepsilon \not\models \phi$ (that is, $\varepsilon \in \overline{\mathcal{L}(\phi)}$), then $\mathcal{L}(M) \cap \overline{\mathcal{L}(\phi)} \neq \varnothing$, and thus $M \not\models \phi$;
2. if $\varepsilon \models \phi$ (that is, $\varepsilon \in \mathcal{L}(\phi)$), then $\varepsilon \notin \overline{\mathcal{L}(\phi)}$; but since $\overline{\mathcal{L}(\phi)}$ is a safety language, by Corollary 13 it holds that $\overline{\mathcal{L}(\phi)} = \varnothing$, implying that $\mathcal{L}(M) \cap \overline{\mathcal{L}(\phi)} = \varnothing$ and $M \models \phi$.

Therefore, checking whether $M \models \phi$ is equivalent to checking whether $\varepsilon \not\models \phi$, for all cosafety formulas ϕ and for all safety, nonempty NFAs M.

6.2 The general case

We now establish that for any LTLf formula ϕ and any NFA M such that $\mathcal{L}(M)$ is a safety language, the model checking problem $M \models \phi$ reduces to verifying whether $\mathcal{L}(M)$ is contained within a substantially simpler language than $\mathcal{L}(\phi)$. Specifically, it suffices to check inclusion in the language recognized by the automaton corresponding to ϕ after removing all non-final states. As an example, suppose that $\phi \coloneqq \mathsf{GF}p$. For any finite word σ, we have that $\sigma \models \mathsf{GF}p$ if and only if σ satisfies p in the last position. Moreover, since $\mathcal{L}(M)$ is prefix-closed, it means that, for every $\sigma \in \mathcal{L}(M)$, proposition p holds in *every* position of σ. Therefore, $M \models \mathsf{GF}p$ is equivalent to $M \models \mathsf{G}p$.

We define the *subclosure* of an NFA as the automaton resulting from the removal of all non-final states along with their associated incoming and outgoing transitions.

▶ **Definition 27** (Subclosure automaton). *Let $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ be an* NFA*. The subclosure of $\mathcal{A}$, denoted with $\mathcal{S}(\mathcal{A})$, is the automaton obtained from $\mathcal{A}$ by removing non-final states,* i.e. *$(Q \cap F, \Sigma, I \cap F, \Delta \cap (F \times \Sigma \times F), F)$.*

The following proposition establishes that, for any NFA M such that $\mathcal{L}(M)$ is safety and any LTLf formula ϕ, the model checking problem $M \models \phi$ is equivalent to checking the language inclusion $\mathcal{L}(M) \subseteq \mathcal{L}(\mathcal{S}(\mathcal{A}_\phi))$, where $\mathcal{A}_\phi$ is the DFA corresponding to ϕ.

▶ **Proposition 28.** *Let ϕ be an* LTLf *formula over $\mathcal{AP}$ and let $\mathcal{A}_\phi$ be its equivalent* DFA*. Let M be an* NFA *such that $\mathcal{L}(M) \subseteq (2^{\mathcal{AP}})^*$ is a safety language. It holds that $M \models \phi$ if and only if $\mathcal{L}(M) \subseteq \mathcal{L}(\mathcal{S}(\mathcal{A}_\phi))$.*

Proof. The right-to-left direction is trivial since $\mathcal{L}(\mathcal{S}(\mathcal{A}_\phi)) \subseteq \mathcal{L}(\mathcal{A}_\phi)$ and thus, if $\mathcal{L}(M) \subseteq \mathcal{L}(\mathcal{S}(\mathcal{A}_\phi))$, then $\mathcal{L}(M) \subseteq \mathcal{L}(\mathcal{A}_\phi)$, that is $M \models \phi$.

To prove the left-to-right direction, assume that $M \models \phi$, that is $\mathcal{L}(M) \subseteq \mathcal{L}(\mathcal{A}_\phi)$, and let $\sigma \in \mathcal{L}(M)$. We prove that $\sigma \in \mathcal{L}(\mathcal{S}(\mathcal{A}_\phi))$ by induction on the length of σ.

Let $|\sigma| = 0$. Thus $\sigma = \varepsilon$. Since σ belongs also to $\mathcal{L}(\mathcal{A}_\phi)$, there exists an initial state of $\mathcal{A}_\phi$ that is final. By Definition 27, such state is also initial and final in $\mathcal{S}(\mathcal{A}_\phi)$. Thus, $\varepsilon \in \mathcal{L}(\mathcal{S}(\mathcal{A}_\phi))$.

Let $|\sigma| = n > 0$. Since $\sigma \in \mathcal{L}(\mathcal{A}_\phi)$, there exists a path $\langle q_0, q_1, \ldots, q_n \rangle$ accepting σ. Since $\mathcal{L}(M)$ is safety and thus prefix-closed, also the prefix σ' of size $n-1$ is in $\mathcal{L}(\mathcal{A}_\phi)$. Moreover, since $\mathcal{A}_\phi$ is a DFA, $\langle q_0, q_1, \ldots, q_{n-1} \rangle$ must be accepting, having that σ' belongs to $\mathcal{L}(\mathcal{A}_\phi)$. By inductive hypothesis, $\sigma' \in \mathcal{L}(\mathcal{S}(\mathcal{A}_\phi))$. Since both q_n and q_{n-1} are final states in $\mathcal{A}_\phi$, by the definition of $\mathcal{S}$ there is a transition from q_{n-1} to q_n, having that $\langle q_0, q_1, \ldots, q_n \rangle$ must be a sequence of $\mathcal{S}(\mathcal{A}_\phi)$ accepting σ. Therefore, $\sigma \in \mathcal{L}(\mathcal{S}(\mathcal{A}_\phi))$. ◀

It is important to note that Proposition 28 remains valid regardless of whether the empty word ε is included in the semantics. In particular, if the model checking problem $M \models \varphi$ is to be interpreted with the exclusion of ε, one can equivalently verify $M \models (\phi \vee \mathsf{G}\bot)$, since $\mathcal{L}(\mathsf{G}\bot) = \{\varepsilon\}$.

Some important consequences of Proposition 28 are highlighted by the following representative examples, which hold for any NFA M such that $\mathcal{L}(M)$ is safety and nonempty:

- if $\phi = \mathsf{F}p$, then $\mathcal{L}(\mathcal{S}(\mathcal{A}_\phi)) = \varnothing$ and therefore it *never* holds that $M \models \phi$;
- if $\phi = \mathsf{F}p \vee \mathsf{G}\bot$, then $\mathcal{L}(\mathcal{S}(\mathcal{A}_\phi)) = \{\varepsilon\} \cup \{p\} \cdot \Sigma^*$ and, since $\varepsilon \in L(M)$, it holds that $M \models \phi$ iff $M \models p$;
- if $\phi = \mathsf{GF}p$, then $\mathcal{L}(\mathcal{S}(\mathcal{A}_\phi)) = \mathcal{L}(\mathsf{G}p)$ and $M \models \phi$ iff $M \models \mathsf{G}p$;
- if $\phi = \mathsf{FG}p$, then $\mathcal{L}(\mathcal{S}(\mathcal{A}_\phi)) = \varnothing$ and thus it *never* holds that $M \models \phi$;
- if $\phi = \mathsf{FG}p \vee \mathsf{G}\bot$, then $\mathcal{L}(\mathcal{S}(\mathcal{A}_\phi)) = \mathcal{L}(\mathsf{G}p)$, and thus $M \models \phi$ iff $M \models \mathsf{G}p$;

7 Conclusions

In this work, we investigated the notions of safety and liveness languages over finite words. We established several fundamental properties of these classes, including the prefix-closed nature of safety languages. Furthermore, we presented effective procedures to determine whether the language recognized by an NFA or specified by an LTLf formula is safety or liveness. We also discussed key distinctions between the finite-word and infinite-word settings. In addition, we proved that, analogously to the infinite-word case, every regular language can

be represented as the intersection of a safety language and a liveness language. We further examined implications for model checking of LTLf formulae, particularly when the program under verification recognizes a safety regular language.

As a direction for future work, we conjecture that the problem of deciding whether the language defined by an LTLf formula is a liveness language is EXPSPACE-complete, a result that remains open in this paper. Furthermore, a natural extension of our results involves the study of *relative safety* and *relative liveness*, as introduced by T. Henzinger in [15] and studied further in [6], that refine the classical definitions by characterizing safety and liveness properties with respect to a given environmental assumption. Investigating these refined notions in the finite-word setting represents an interesting direction for future research.

References

1 Bowen Alpern and Fred B Schneider. Defining liveness. *Information processing letters*, 21(4):181–185, 1985. doi:10.1016/0020-0190(85)90056-0.

2 Bowen Alpern and Fred B. Schneider. Recognizing safety and liveness. *Distributed Comput.*, 2(3):117–126, 1987. doi:10.1007/BF01782772.

3 Suguman Bansal, Yong Li, Lucas M. Tabajara, Moshe Y. Vardi, and Andrew M. Wells. Model checking strategies from synthesis over finite traces. In Étienne André and Jun Sun, editors, *Automated Technology for Verification and Analysis - 21st International Symposium, ATVA 2023, Singapore, October 24-27, 2023, Proceedings, Part I*, volume 14215 of *Lecture Notes in Computer Science*, pages 227–247. Springer, 2023. doi:10.1007/978-3-031-45329-8_11.

4 David A. Basin, Carlos Cotrini Jiménez, Felix Klaedtke, and Eugen Zalinescu. Deciding safety and liveness in TPTL. *Inf. Process. Lett.*, 114(12):680–688, 2014. doi:10.1016/J.IPL.2014.06.005.

5 Armin Biere, Nils Froleyks, and Mathias Preiner. Hardware model checking competition 2024. In Nina Narodytska and Philipp Rümmer, editors, *Formal Methods in Computer-Aided Design, FMCAD 2024, Prague, Czech Republic, October 15-18, 2024*, page 1. IEEE, 2024. doi:10.34727/2024/ISBN.978-3-85448-065-5_6.

6 Alberto Bombardelli, Alessandro Cimatti, Stefano Tonetta, and Marco Zamboni. Symbolic Model Checking of Relative Safety LTL Properties. In *iFM*, volume 14300 of *Lecture Notes in Computer Science*, pages 302–320. Springer, 2023. doi:10.1007/978-3-031-47705-8_16.

7 Aaron R Bradley. SAT-based model checking without unrolling. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 70–87. Springer, 2011. doi:10.1007/978-3-642-18275-4_7.

8 Aaron R Bradley. Understanding IC3. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 1–14. Springer, 2012. doi:10.1007/978-3-642-31612-8_1.

9 Edward Y. Chang, Zohar Manna, and Amir Pnueli. Characterization of temporal property classes. In Werner Kuich, editor, *Proceedings of the 19th International Colloquium on Automata, Languages and Programming*, volume 623 of *Lecture Notes in Computer Science*, pages 474–486. Springer, 1992. doi:10.1007/3-540-55719-9_97.

10 Alessandro Cimatti, Luca Geatti, Nicola Gigante, Angelo Montanari, and Stefano Tonetta. A first-order logic characterisation of safety and co-safety languages. In Patricia Bouyer and Lutz Schröder, editors, *Foundations of Software Science and Computation Structures - 25th International Conference, FOSSACS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings*, volume 13242 of *Lecture Notes in Computer Science*, pages 244–263. Springer, 2022. doi:10.1007/978-3-030-99253-8_13.

11 Koen Claessen and Niklas Sörensson. A liveness checking algorithm that counts. In *2012 Formal Methods in Computer-Aided Design (FMCAD)*, pages 52–59. IEEE, 2012. URL: https://ieeexplore.ieee.org/document/6462555/.

12 Edmund M Clarke, Thomas A Henzinger, Helmut Veith, and Roderick Bloem. *Handbook of model checking*, volume 10. Springer, 2018. doi:10.1007/978-3-319-10575-8.

13 Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI'13*, pages 854–860, 2013. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6997.

14 Luca Geatti, Stefano Pessotto, and Stefano Tonetta. Safety and liveness on finite words (extended version). https://users.dimi.uniud.it/~luca.geatti/data/time-25/time25.pdf, 2025. Extended version of the paper.

15 Thomas A. Henzinger. Sooner is safer than later. *Inf. Process. Lett.*, 43(3):135–141, 1992. doi:10.1016/0020-0190(92)90005-G.

16 Thomas A. Henzinger, Nicolas Mazzocchi, and N. Ege Saraç. Quantitative Safety and Liveness. In *FoSSaCS*, volume 13992 of *Lecture Notes in Computer Science*, pages 349–370. Springer, 2023. doi:10.1007/978-3-031-30829-1_17.

17 John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. Introduction to automata theory, languages, and computation. *Acm Sigact News*, 32(1):60–65, 2001. doi:10.1145/568438.568455.

18 Orna Kupferman and Gal Vardi. On relative and probabilistic finite counterability. *Formal Methods Syst. Des.*, 52(2):117–146, 2018. doi:10.1007/S10703-017-0277-8.

19 Orna Kupferman and Moshe Y Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001. doi:10.1023/A:1011254632723.

20 Leslie Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, SE-3(2):125–143, 1977. doi:10.1109/TSE.1977.229904.

21 Zohar Manna and Amir Pnueli. *Temporal verification of reactive systems - safety*. Springer, 1995.

22 Doron Peled and Klaus Havelund. Refining the Safety-Liveness Classification of Temporal Properties According to Monitorability. In *Models, Mindsets, Meta*, volume 11200 of *Lecture Notes in Computer Science*, pages 218–234. Springer, 2018. doi:10.1007/978-3-030-22348-9_14.

23 Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57. IEEE, 1977. doi:10.1109/SFCS.1977.32.

24 A Prasad Sistla. On characterization of safety and liveness properties in temporal logic. In *Proceedings of the fourth annual ACM symposium on Principles of distributed computing*, pages 39–48, 1985. doi:10.1145/323596.323600.

25 A Prasad Sistla. Safety, liveness and fairness in temporal logic. *Formal Aspects of Computing*, 6(5):495–511, 1994. doi:10.1007/BF01211865.

A Omitted proofs

▶ **Corollary 13** (Small and Bounded Model Property for safety languages). *Let $\mathcal{L} \subseteq \Sigma^*$ be a safety language. It holds that $\mathcal{L} \neq \varnothing$ if and only if $\varepsilon \in \mathcal{L}$.*

Proof. The right-to-left direction is trivial. To prove the left-to-right direction, suppose that $\mathcal{L}$ is not empty and let $\sigma \in \mathcal{L}$. Since by hypothesis $\mathcal{L}$ is safety, by Proposition 12 we have that $\mathcal{L}$ is prefix-closed. Since $\sigma \in \mathcal{L}$ and since ε is a prefix of σ, by Definition 11 we have that $\varepsilon \in \mathcal{L}$. ◀

▶ **Proposition 15.** *Establishing whether the language accepted by an* NFA *is safety is* PSPACE*-complete.*

Proof. For the membership in PSPACE, we show the following procedure to check whether the language of a given reduced NFA is equal to the language of its closure (Proposition 14). Let $\mathcal{A}$ be an NFA of size n. By means of a sequence of reachability checks, in nondeterministic

logarithmic space, we can turn $\mathcal{A}$ into an equivalent reduced NFA $\mathcal{R}(\mathcal{A})$ of size $\mathcal{O}(n)$. The procedure checks whether $\mathcal{L}(\mathcal{R}(\mathcal{A})) = \mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A})))$. Notice that, since $\mathcal{L}(\mathcal{R}(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A})))$, only the opposite inclusion needs to be checked. Checking whether $\mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A}))) \subseteq \mathcal{L}(\mathcal{R}(\mathcal{A}))$ is equivalent to check whether:

$$\mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A}))) \cap \overline{\mathcal{L}(\mathcal{R}(\mathcal{A}))} = \varnothing$$

Building an automaton for $\overline{\mathcal{L}(\mathcal{R}(\mathcal{A}))}$ requires $2^{\mathcal{O}(n)}$ space [17], computing the intersection between two automata requires polynomial space with respect to the size of the two automata, and checking the emptiness can be performed *on-the-fly* in nondeterminstic logarithmic space. The total complexity is thus nondeterministic polynomial space, and thus the problem is in PSPACE.

For proving the PSPACE-hardness, we consider the *universality problem* for NFAs, *i.e.* the problem of checking whether the language of a given NFA over the alphabet Σ is Σ^*, which is known to be PSPACE-complete.

Let $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ be an NFA. We define the automaton $\mathcal{B}$ as the NFA $(Q', \Sigma', I', \Delta', F')$ such that:

- $Q' \coloneqq Q \cup \{q_{sink}, q_{loop}\}$;
- $\Sigma' \coloneqq \Sigma \cup \{f\}$, where $f \notin \Sigma$;
- $I' \coloneqq I$;
- $\Delta' \coloneqq \Delta \cup \{(q, f, q_{loop}) \mid q \in Q'\} \cup$
  $\{(q, a, q_{sink}) \mid q \in Q,\ a \in \Sigma,\ \Delta(q, a) = \varnothing\} \cup$
  $\{(q, a, q) \mid a \in \Sigma,\ q \in \{q_{sink}, q_{loop}\}\}$;
- $F' \coloneqq F \cup \{q_{loop}\}$.

Automaton $\mathcal{B}$ has two crucial properties: (i) it has no undefined transitions; (ii) $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{A}) \cup (\Sigma')^* \cdot f \cdot (\Sigma')^*$; note that this implies that $\mathcal{L}(\mathcal{B}) \cap (\Sigma)^* = \mathcal{L}(\mathcal{A})$.

We prove that $\mathcal{L}(\mathcal{A}) = \Sigma^*$ if and only if $\mathcal{L}(\mathcal{B})$ is safety.

Suppose that $\mathcal{L}(\mathcal{A}) = \Sigma^*$ and let σ be any word in $(\Sigma')^*$. We divide in cases, depending on whether σ contains at least one f.

Case 1. If σ does not contain any f, then $\sigma \in \Sigma^*$ and thus $\sigma \in \mathcal{L}(\mathcal{A})$. Since by construction $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{A}) \cup (\Sigma')^* \cdot f \cdot (\Sigma')^*$, we have that $\sigma \in \mathcal{L}(\mathcal{B})$.

Case 2. If σ contains at least one f, then $\sigma \in (\Sigma')^* \cdot f \cdot (\Sigma')^*$, and thus, also in this case, $\sigma \in \mathcal{L}(\mathcal{B})$.

Therefore $\mathcal{L}(\mathcal{B}) = (\Sigma')^*$ and, clearly, $\mathcal{L}(\mathcal{B})$ is a safety language over the alphabet Σ'.

Suppose that $\mathcal{L}(\mathcal{B})$ is a safety language. We first prove that $\mathcal{L}(\mathcal{B}) = (\Sigma')^*$. Suppose by contradiction that this is not case and let $\sigma \notin \mathcal{L}(\mathcal{B})$. Since $\mathcal{L}(\mathcal{B})$ is a safety language, there exists a $\sigma' \in \texttt{pref}(\sigma)$ such that $\sigma' \cdot \sigma'' \notin \mathcal{L}(\mathcal{B})$, for all $\sigma'' \in (\Sigma')^*$. Now, let q be any state reached by $\mathcal{B}$ after reading σ' (notice that, since $\mathcal{B}$ does not contain undefined transitions, state q always exists). By construction of $\mathcal{B}$, reading the symbol f, $\mathcal{B}$ transitions from state q to state q_{loop}. Since q_{loop} is a final state, this means that $\sigma' \cdot f \in \mathcal{L}(\mathcal{B})$. But this is a contradiction with the fact that $\sigma' \cdot \sigma'' \notin \mathcal{L}(\mathcal{B})$, for all $\sigma'' \in (\Sigma')^*$. Therefore, $\mathcal{L}(\mathcal{B}) = (\Sigma')^*$.

As observed above, by construction, $\mathcal{L}(\mathcal{B}) \cap (\Sigma)^* = \mathcal{L}(\mathcal{A})$. Thus, the fact that $\mathcal{L}(\mathcal{B}) = (\Sigma')^*$ implies that $\mathcal{L}(\mathcal{A}) = \Sigma^*$. ◀

▶ **Proposition 16.** *Establishing whether the language of an* LTLf *formula is safety is* PSPACE*-complete.*

Proof. We follow the same approach as Sistla [25] for proving that the problem of establishing whether the language recognized by an LTL formula is safety is PSPACE-complete.

As for the membership in PSPACE, we follow this procedure. Given an LTLf formula ϕ of size n over the atomic propositions $\mathcal{AP}$, the procedure builds two NFAs over the alphabet $2^{\mathcal{AP}}$, both of size $2^{\mathcal{O}(n)}$ [13]: the automaton $\mathcal{A}_\phi$ equivalent to the formula ϕ, and the automaton $\mathcal{A}_{\neg\phi}$ equivalent to the formula $\neg\phi$. To check whether $\mathcal{L}(\phi)$ is a safety language, it suffices to check whether $\mathcal{L}(\mathcal{R}(\mathcal{A}_\phi)) = \mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A}_\phi)))$ (Proposition 14), which is equivalent to check whether $\mathcal{L}(\mathcal{C}(\mathcal{R}(\mathcal{A}_\phi))) \cap \mathcal{L}(\mathcal{R}(\mathcal{A}_{\neg\phi})) = \varnothing$. This check can be done in nondeterministic logarithmic space, *on-the-fly* while building the two automata. Therefore, the problem is in PSPACE.

To prove the PSPACE-hardness, we use the validity problem for LTLf formulae, *i.e.* checking whether the language of a given LTLf formula over the set of atomic propositions $\mathcal{AP}$ is $(2^{\mathcal{AP}})^*$. Let ϕ be an LTLf formula over $\mathcal{AP}$. We prove that ϕ is valid (*i.e.* $\mathcal{L}(\phi) = (2^{\mathcal{AP}})^*$) if and only if $\mathcal{L}(\phi \vee \mathsf{F}p)$ is safety, where $p \notin \mathcal{AP}$.

Proving that if $\mathcal{L}(\phi) = (2^{\mathcal{AP}})^*$ then $\mathcal{L}(\phi \vee \mathsf{F}p)$ is safety is straightforward. Since $\mathcal{L}(\phi) = (2^{\mathcal{AP}})^*$ and p does not appear in ϕ, it also holds that $\mathcal{L}(\phi \vee \mathsf{F}p) = (2^{\mathcal{AP}\cup\{p\}})^*$, and thus $\mathcal{L}(\phi \vee \mathsf{F}p)$ is safety.

We now prove the opposite direction. Suppose that $\mathcal{L}(\phi \vee \mathsf{F}p)$ is safety. Suppose by contradiction that $\phi \vee \mathsf{F}p$ is not valid and let $\sigma \in (2^{\mathcal{AP}\cup\{p\}})^*$ be a word such that $\sigma \not\models \phi \vee \mathsf{F}p$. Since by hypothesis the language of $\phi \vee \mathsf{F}p$ is safety, there exists a $\sigma' \in \mathtt{pref}(\sigma)$ such that $\sigma' \cdot \sigma'' \not\models \phi \vee \mathsf{F}p$, for all $\sigma'' \in (2^{\mathcal{AP}\cup\{p\}})^*$. However, this is a contradition because the $\sigma' \cdot \{p\} \models \phi \vee \mathsf{F}p$. Therefore, $\phi \vee \mathsf{F}p$ is valid, *i.e.* $\mathcal{L}(\phi \vee \mathsf{F}p) = (2^{\mathcal{AP}\cup\{p\}})^*$. Since $p \notin \mathcal{AP}$, this means that $\mathcal{L}(\phi) = (2^{\mathcal{AP}})^*$. ◀

▶ **Proposition 24.** *Let $\mathcal{A}$ be a* DFA*. It holds that $\mathcal{L}(\mathtt{Safe}(\mathcal{A}))$ (resp., $\mathcal{L}(\mathtt{Live}(\mathcal{A}))$) is a safety language (resp., a liveness language).*

Proof. We first prove the case for $\mathtt{Safe}(\mathcal{A})$. Since, by Definition 23, $\mathtt{Safe}(\mathcal{A})$ is defined as $\mathcal{C}(\mathcal{R}(\mathcal{A}))$, it holds that $\mathcal{C}(\mathcal{R}(\mathtt{Safe}(\mathcal{A})))$ is exactly $\mathtt{Safe}(\mathcal{A})$. In particular $\mathcal{L}(\mathtt{Safe}(\mathcal{A})) = \mathcal{L}(\mathcal{C}(\mathcal{R}(\mathtt{Safe}(\mathcal{A}))))$. By Proposition 14, we have that $\mathcal{L}(\mathtt{Safe}(\mathcal{A}))$ is a safety language.

We now prove the case for $\mathtt{Live}(\mathcal{A})$. The definition of $\mathtt{Live}(\mathcal{A})$ states that all undefined transitions of $\mathcal{R}(\mathcal{A})$ are replaced with a transition into a new, final state (to which the automaton is forced to remain reading any symbol). This means that the transition relation of $\mathtt{Live}(\mathcal{A})$ is *complete*: $\Delta(q, a) \neq \varnothing$, for all states q of $\mathtt{Live}(\mathcal{A})$ and for all symbols $a \in \Sigma$. Since all the states in $\mathcal{C}(\mathcal{R}(\mathtt{Live}(\mathcal{A})))$ are set to final, it follows that $\mathcal{L}(\mathcal{C}(\mathcal{R}(\mathtt{Live}(\mathcal{A})))) = \Sigma^*$. By Proposition 19, it means that $\mathcal{L}(\mathtt{Live}(\mathcal{A}))$ is a liveness language. ◀