Symbolic Model Checking of Relative Safety LTL Properties

Alberto Bombardelli[0000−0003−3385−3205],
Alessandro Cimatti[0000−0002−1315−6990],
Stefano Tonetta[0000−0001−9091−7899], and
Marco Zamboni

Fondazione Bruno Kessler, Via Sommarive, 18, 38123 Povo TN
{abombardelli,cimatti,tonettas,mazamboni}@fbk.eu

Abstract. A well-known classification in formal methods distinguishes between safety and liveness properties. A generalization of these concepts defines safety and liveness relative to another property, usually considered an assumption on the traces of the language. Safety properties have the advantage that their model checking problem can be reduced to simple reachability. However, the generalization of such reduction to the case of relative safety has not yet been investigated.
In this paper, we study the problem of model checking relative safety properties in the context of Linear-time Temporal Logic. We show that the problem can be reduced to reachability when the system model is free of livelocks related to the assumptions. More in general, we provide an algorithm that removes such livelocks from the system model driven by counterexamples to the relative safety property. We compare the approach with other reduction to safety algorithms on several benchmarks.

1 Introduction

The distinction between safety and liveness properties is very well known by the formal methods community. It was first introduced by Lamport in [26] to reason about the correctness of programs. Safety properties like mutual exclusion demand that "bad things" never happen, while liveness properties like non-starvation require that "good things" will eventually be achieved by the system. Alpern and Schneider in [2] formalized the definitions in terms of languages of infinite words. A safety property is a property whose counterexamples are characterized by a prefix reaching a bad state, while a liveness property is a property that can be fulfilled by extending any prefix to an infinite execution with good states. The classification is complete in the sense that every omega-regular language can be decomposed into a safety and a liveness property. As proven by Alpern and Schneider in [3], model checking safety properties can be reduced to checking an invariant.

In [24], Henzinger introduced the notion of relative safety and relative liveness, which extend the standard definitions considering an assumption property.

For example, a property φ is safety relative to an assumption α if any counterexample to φ satisfying α has a bad prefix. This is the case for example of a bounded response property that states that every request is followed by a response within δ time units: this property is safety relative to the assumption that requires δ time units to always eventually elapse (i.e., avoiding Zeno executions).

In this paper, we study the model checking problem of Linear-time Temporal Logic properties in the form $\alpha \rightarrow \phi$, where ϕ is safety. These properties are not usually safety, but are safety relative to α. We prove that if the system model M is live with respect to α, i.e., that prefixes of M can be extended to infinite executions satisfying α, then the model checking problem $M \models \alpha \rightarrow \phi$ can be reduced to an invariant check. More in general, we provide an algorithm that iteratively removes the states that are not live with respect to α (α-livelocks). These iterations are driven by (finite) counterexamples to ϕ.

We implemented the algorithm in the nuXmv[8] model checker on top of SMT-based symbolic model checking for finite and infinite-state systems. As additional contributions, we provide a symbolic compilation of SafetyLTL and a look-ahead heuristics that avoids deadlocks by requiring a bounded extension of finite counterexamples. We evaluated the algorithm on a number of benchmarks including assume-guarantee reasoning, bounded response properties assuming non-Zenoness, and asynchronous properties assuming a fair scheduling. The results show the benefit of reducing to invariants whenever the number of α-livelocks / iterations are limited.

Outline The rest of the paper is organized as follows. Section 2, we provide some examples that motivates the form of properties addressed in this paper. In Section 3, we compare with other approaches based on reduction to safety. Section 4 gives basic definitions of safety, relative safety, and LTL. Section 5 details the compilation of SafetyLTL into symbolic transition systems. Section 6 provides the algorithms and proves their correctness and completeness. In Section 7, we describe the experimental evaluation and results. Finally, in Section 8, we draw some conclusions and directions for future works.

2 Motivating examples

In this section, we give various examples where we are interested to prove properties of the form $\alpha \rightarrow \phi$ where ϕ is safety.

2.1 Bounded response example

The key observation of Henzinger in [24] was that some properties like bounded response in dense time are safety only restricting the language to non-Zeno paths. Although the contribution of this paper can be lifted to dense and super-dense models of time, we port the example to the discrete-time setting using an explicit real *time* variable. Therefore, let us consider the property:

$$\psi := G((p \wedge time = t) \rightarrow F(q \wedge time \leq t + \delta))$$

for some constant δ.

Let us assume that the values of *time* are diverging over infinite execution traces, for example by posing the following constraints:

$$\alpha := G(next(time) \geq time) \wedge GF(next(time) - time > \zeta)$$

for some constant ζ.

Every trace π satisfying α and violating ψ will have a a state $\pi[i]$ in which p holds and *time* has some value τ and a following state $\pi[j]$ with $j \geq i$ in which *time* has a value greater than or equal to $\tau+\delta$ and for states between $\pi[i]$ and $\pi[j]$ q is false. Note that the assumption α is necessary to rule out counterexamples to ψ where *time* is always less than $\tau + \delta$. The prefix of π up to $\pi[j]$ violates ψ for any possible suffix. Thus it is a finite trace witnessing the violation of ψ.

In fact, we can be prove that:

$$\alpha \rightarrow (\psi \leftrightarrow \phi)$$

where

$$\phi := G((p \wedge time = t) \rightarrow (time \leq t + \delta)W(q \wedge time \leq t + \delta))$$

which is safety. Thus, instead of $\alpha \rightarrow \psi$, we can prove $\alpha \rightarrow \phi$.

2.2 Safety contracts

In contract-based design and assume-guarantee reasoning, the properties are often in the form $\alpha \rightarrow \phi$ with both α and ϕ safety. Consider for example the case in which $\alpha = Ga$ and $\phi = Gb$. In some frameworks (e.g., in AGREE [22]), this is interpreted as $G((Ha) \rightarrow b)$, meaning that if we find a violation of b we check that a is true in the past states, without caring about the future continuation. In a sense, we assume that the counterexample can be continued to satisfy Ga.

2.3 LTL Gp vs. invariant

Similarly to the previous case, the LTL property Gp and an invariant property p are often considered interchangeably. However, they are not equivalent as the LTL is interpreted over infinite traces, while the invariant is violated by finite traces. Therefore, by reducing Gp to an invariant p, we are assuming that the counterexample violating p can be continued to an infinite trace.

3 Related Work

In [25], Kupferman and Vardi studied the verification of safety properties; they introduce the notion of informative path to formally denote a computation violating a safety property. That work also defines a PSPACE procedure to check whether a property is safety. Later, in [27], the same approach have been applied

in a BDD-based algorithm. Another related work for the verification of safety property is [19]. In [19], LTL formulas are translated into circuits. If the property is syntactically safety, it is possible to reduce its verification to reachability checking; otherwise, the verification procedure involves also liveness checking. In all these works, the reduction to reachability is limited to safety properties, while relative safety is not considered.

The approach of AGREE [22](see Section 2.2) proposes a semantics for contracts that can be seen as extension of the reduction for specific pattern of safety assumptions. However, such reductions assume implicitly the absence of deadlocks and livelocks. We are not aware of other approaches that use the assumptions explicitly and exploit the notation of relative safety for generic assumptions.

Other well-known reduction to safety are used to prove full LTL formulas. In [5], the approach duplicates the state variables and add monitoring constraints to look for a fair lasso. It works only for finite-state systems and the resulting reachability condition is related to the structure of the lasso-shaped counterexample. K-liveness [20] is more similar to our reduction as it counts and looks for a bound to the number of times the fairness condition of counterexamples can be visited. If the property is safety, the fairness cannot be reached or can be reached only once, depending on the actual construction. However, in the case of properties in the form $\alpha \rightarrow \phi$, the fairness conditions related to α are mixed with those of $\neg\phi$ in looking for a counterexample. So, in a sense, our approach gives more priority to look for a counterexample to ϕ, considering the fairness of α in a second step.

4 Notation and Preliminary Definitions

4.1 Notation for Sequences, Concatenation and Prefix

Given a sequence u and $i < |u|$, we denote by $u[i]$ the $i+1$-th element of the sequence starting to count from 0 (thus, $u[0]$ is the first element), and by $u[i..]$ the suffix of u starting from $u[i]$. Given a finite sequence u and a finite or infinite sequence w, we denote by uw their concatenation. Given an infinite sequence, we denote by $\mathit{Pref}(\pi)$ the set of prefixes, i.e., $\mathit{Pref}(\pi) = \{u \in \Sigma^* \mid uw = \pi\}$.

4.2 Safety and Relative Safety

Given an alphabet Σ, a infinite/finite trace is an infinite/finite word over Σ, i.e., a sequence in Σ^ω or Σ^* respectively. A property is a subset of Σ^ω. A property is safety if nothing bad happens during an execution. If something bad happens, it occurs in a finite prefix of the execution. This is formalized as follows.

Definition 1. *Let P be a property. P is a safety property iff*

$$\textit{for all } \pi \in \Sigma^\omega \textit{ s.t. } \pi \notin P, \textit{ there exists } \pi_f \in \mathit{Pref}(\pi) \textit{ s.t.}$$
$$\textit{for all } \pi^\omega \in \Sigma^\omega : \pi_f \pi^\omega \notin P$$

Furthermore, we denote π_f as a bad prefix of P.

Definition 2. *Let P be a property. P is a liveness property iff*

$$\textit{for all } \pi \in \Sigma^\omega,\ \textit{for all } \pi_f \in \textit{Pref}(\pi),\ \textit{there exists } \pi^\omega \in \Sigma^\omega \textit{ s.t. } \pi_f\pi^\omega \in P$$

In [3], it was proved that for every omega-regular property P, there exists a safety property P_S and a liveness property P_L such that $P = P_S \cap P_L$.

Definition 3. *Let P and A be two properties. P is safety relative to A iff*

$$\begin{aligned}&\textit{for all } \pi \in A \textit{ s.t. } \pi \notin P,\ \textit{there exists } \pi_f \in \textit{Pref}(\pi) \textit{ s.t.}\\ &\textit{for all } \pi^\omega \in \Sigma^\omega :\ \textit{if } \pi_f\pi^\omega \in A \textit{ then } \pi_f\pi^\omega \notin P\end{aligned}$$

In [24], it was proved that for every property P relative safety to A, there exists a safety property P_W such that $A \cap P = A \cap P_W$.

4.3 LTL and Safety Fragments

In this paper, we consider LTL [29] extended with past operators [28] as well as predicates, functions and the *next* operator. For simplicity we refer to it as LTL. We work in the setting of Satisfiability Modulo Theory (SMT) [4] and LTL Modulo Theory (see, e.g., [13]). First-order formulas are built as usual by proposition logic connectives, a given set of variables V and a first-order signature Σ, and are interpreted according to a given Σ-theory $\mathcal{T}$. We assume to be given the definition of $M, \mu \models_{\mathcal{T}} \varphi$ where M is a Σ-structure, μ is a value assignment to the variables in V, and φ is a formula. Whenever $\mathcal{T}$ and M are clear from contexts we omit them and simply write $\mu \models \varphi$.

LTL syntax

Definition 4. *Given a signature Σ and a set of variables V, LTL formulas φ are defined by the following syntax:*

$$\begin{aligned}\varphi :=& \top \mid \bot \mid p(u_1, \ldots, u_n) \mid \neg\varphi \mid \varphi \wedge \varphi \mid X\varphi \mid \varphi U \varphi \mid Y\varphi \mid \varphi S \varphi\\ u :=& c \mid x \mid \textit{func}(u, \ldots, u) \mid \textit{next}(u)\end{aligned}$$

where c, $\textit{func}$, and p are respectively a constant, a function, and a predicate of the signature Σ and x is a variable in V.

LTL semantics LTL formulas are interpreted over traces, i.e., infinite sequences of assignments to the variables in V. We denote by $\Pi(V)$ the set of all possible traces over the variable set V. Given a trace $\pi = s_0 s_1 \cdots \in \Pi(V)$ and a Σ-structure M, the semantics of a formula φ is defined as follows:

- $\pi, M, i \models \top$
- $\pi, M, i \not\models \bot$
- $\pi, M, i \models p(u_1, \ldots, u_n)$ iff $p^M(\pi^M(i)(u_1), \ldots, \pi^M(i)(u_n))$
- $\pi, M, i \models \varphi_1 \wedge \varphi_2$ iff $\pi, M, i \models \varphi_1$ and $\pi, M, i \models \varphi_2$

- $\pi, M, i \models \neg\varphi$ iff $\pi, M, i \not\models \varphi$
- $\pi, M, i \models \varphi_1 U \varphi_2$ iff there exists $k \geq i, \pi, M, k \models \varphi_2$ and for all $l, i \leq l < k, \pi, M, l \models \varphi_1$
- $\pi, M, i \models \varphi_1 S \varphi_2$ iff there exists $k \leq i, \pi, M, k \models \varphi_2$ and for all $l, k < l \leq i, \pi, M, l \models \varphi_1$
- $\pi, M, i \models X\varphi$ iff $\pi, M, i+1 \models \varphi$
- $\pi, M, i \models Y\varphi$ iff $i > 0$ and $\pi, M, i-1 \models \varphi$

where the interpretation of terms $\pi^M(i)$ is defined as follows:

- $\pi^M(i)(c) = c^M$
- $\pi^M(i)(x) = s_i(x)$ if $x \in V$
- $\pi^M(i)(\mathit{func}(u_1, \ldots, u_n)) = \mathit{func}^M(\pi^M(i)(u_1), \ldots, \pi^M(i)(u_n))$
- $\pi^M(i)(\mathit{next}(u)) = \pi^M(i+1)(u)$

and the $p^M, \mathit{func}^M, c^M$ are the interpretation M of the symbols in Σ.

Finally, we have that $\pi, M \models \varphi$ iff $\pi, M, 0 \models \varphi$.

In the following, we assume to have a background theory such that the symbols in Σ are interpreted by an implicit structure M (e.g., theory of reals, integers, etc.). We therefore omit M to simplify the notation, writing $\pi, i \models \varphi$ and $\pi(i)(u)$ instead of respectively $\pi, M, i \models \varphi$ and $\pi^M(i)(u)$.

Moreover, we use the following standard abbreviations: $\varphi_1 \vee \varphi_2 := \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\varphi_1 R \varphi_2 := \neg(\neg\varphi_1 U \neg\varphi_2)$ (φ_1 releases φ_2), $F\varphi := \top U \varphi$ (sometime in the future φ), $G\varphi := \neg F \neg\varphi$ (always in the future φ), $\varphi_1 T \varphi_2 := \neg(\neg\varphi_1 S \neg\varphi_2)$ (φ_1 is triggered by φ), $O\varphi := \top S \varphi$ (once in the past φ), $H\varphi := \neg O \neg\varphi$ (historically in the past φ), $Z\varphi := \neg Y \neg\varphi$ (yesterday φ or at initial state), $X^n\varphi := XX^{n-1}\varphi$ with $X^0\varphi := \varphi$, $Y^n\varphi := YY^{n-1}\varphi$ with $Y^0\varphi := \varphi$, $Z^n\varphi := ZZ^{n-1}\varphi$ with $Z^0\varphi := \varphi$.

Safety fragments of LTL SafetyLTL is a fragment of Linear Temporal Logic which disallows positive occurrence of until.

Definition 5. *The syntax of SafetyLTL is defined as follows:*

$$\begin{aligned}\varphi :=& \top \mid \bot \mid p(u, \ldots, u) \mid \neg p(u, \ldots, u) \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \\ & \varphi R \varphi \mid X\varphi \mid Y\varphi \mid Z\varphi \mid \varphi S \varphi \mid \varphi T \varphi \\ u :=& c \mid x \mid \mathit{func}(u, \ldots, u) \mid \mathit{next}(u)\end{aligned}$$

Note that we use include also past operators in SafetyLTL although this is typically defined only with the future ones.

Another safe fragment of LTL is full-past LTL. Although the syntax of the fragment is far stricter than SafetyLTL, the two logics have the same expressibility.

Definition 6. *The syntax of full-past LTL is of the form $\phi := G\beta$ where β is as follows:*

$$\begin{aligned}\beta :=& \beta \wedge \beta \mid \neg\beta \mid Y\beta \mid \beta S \beta \mid p(u, \ldots, u) \\ u :=& c \mid x \mid \mathit{func}(u, \ldots, u)\end{aligned}$$

It was proved in [9] that both fragments, SafetyLTL (even without past operators) and full-past LTL express all and only the safety properties of LTL.

In [25], the authors introduce the notion of *informative* prefix. Informally, an *informative* prefix for an LTL property is a finite path that witnesses the violation of the formula. The formal definition introduces a mapping from the finite path to the sub-formulas of the property. The definition is as follows:

Definition 7. *[25] Let ψ be an LTL formula in negative normal form, $Sub(\psi)$ be the set of sub-formulas of ψ and let π be a finite path of length n over the language of ψ. We say that π is informative for ψ iff there exists a mapping $L : \{0, \ldots, n\} \to 2^{Sub(\neg\psi)}$ such that:*

1. $\neg\psi \in L(0)$.
2. $L(n) = \emptyset$.
3. *For all* $0 \leq i < n$, *forall* $\varphi \in L(i)$:
   - *If* φ *is propositional,* $\pi, i \models \varphi$.
   - *If* $\varphi = \varphi_1 \vee \varphi_2$, $\varphi_1 \in L(i)$ *or* $\varphi_2 \in L(i)$.
   - *If* $\varphi = \varphi_1 \wedge \varphi_2$, $\varphi_1 \in L(i)$ *and* $\varphi_2 \in L(i)$.
   - *If* $\varphi = X\varphi_1$, $\varphi_1 \in L(i+1)$
   - *If* $\varphi = \varphi_1 U \varphi_2$, $\varphi_2 \in L(i)$ *or* $[\varphi_1 \in L(i)$ *and* $\varphi_1 U \varphi_2 \in L(i+1)]$.
   - *If* $\varphi = \varphi_1 R \varphi_2$, $\varphi_2 \in L(i)$ *and* $[\varphi_1 \in L(i)$ *or* $\varphi_1 R \varphi_2 \in L(i+1)]$.

4.4 Symbolic Transition System and Invariant Checking

A *Symbolic Transition System* (STS) M is a tuple $M = \langle V, I, T\rangle$ where V is a set of (state) variables, $I(V)$ is a formula representing the initial states, and $T(V, V')$ is a formula representing the transitions. A *state* of M is an assignment to the variables V. A [finite] *path* of M is an infinite sequence $s_0, s_1, \ldots$ [resp., finite sequence $s_0, s_1, \ldots, s_k$] of states such that $s_0 \models I$ and, for all $i \geq 0$ [resp., $0 \leq i < k$], $s_i, s'_{i+1} \models T$. Given two transitions systems $M_1 = \langle V_1, I_1, T_1\rangle$ and $M_2 = \langle V_2, I_2, T_2\rangle$, we denote with $M_1 \times M_2$ the synchronous product $\langle V_1 \cup V_2, I_1 \wedge I_2, T_1 \wedge T_2\rangle$.

A property of $M = \langle V, I, T\rangle$ is specified over a set $V' \subseteq V$ of variables. The alphabet is therefore given by the set of assignments to V'. A finite or infinite path defines a corresponding trace given by restricting the assignments to V'.

An invariant property is a Boolean combination of predicates. Given an invariant property ϕ, the invariant model checking problem, denoted with $M \models_{inv} \phi$, is the problem to check if, for all finite paths $s_0, s_1, \ldots, s_k$ of M, for all i, $0 \leq i \leq k$, $s_i \models \phi$.

Given an LTL formula ϕ, the LTL model checking problem, denoted with $M \models \phi$, is the problem to check if, for all (infinite) paths π of M, $\pi \models \phi$.

The automata-based approach [30] to LTL model checking is to build an automaton (or in our case an STS) $M_{\neg\phi}$ with multiple fairness conditions $f^i_{\neg\phi}$ such that $M \models \phi$ iff $M \times M_{\neg\phi} \models \bigvee_i FG\neg f^i_{\neg\phi}$. This reduces to finding a counterexample as a fair path, i.e., a path of the system that visits each fairness condition $f^i_{\neg\phi}$ infinitely many times. In case of finite-state systems, if the property fails

there is always a counterexample in a lasso-shape, i.e., formed by a prefix and a loop.

As anticipated in Section 2.3, given an invariant property ϕ, it may be the case that $M \models G\phi$ but $M \not\models_{inv} \phi$ because there is a finite path violating ϕ that cannot be extended to an infinite path. We thus introduce the notion of live system as follows.

Definition 8. *An STS $M = \langle V, I, T\rangle$ is live with respect to an LTL property α iff for every finite trace σ and finite path π of M over σ, if there exists a trace σ' such that $\sigma\sigma' \models \alpha$, then there exists an infinite path π^ω of M such that $\pi \in Pref(\pi^\omega)$ and $\pi^\omega \models \alpha$.*

Note that if $\alpha = \alpha_S \wedge \alpha_L$ where α_S is safety and α_L is liveness, then M is live w.r.t. α iff for every finite trace σ and finite path π of M over σ, if σ is not a bad prefix of α_S then there exists an infinite path π^ω of M such that $\pi \in Pref(\pi^\omega)$ and $\pi^\omega \models \alpha_L$ (and thus $\pi^\omega \models \alpha$). If such path does not exist, we call the states of π livelocks with respect to α.

4.5 Symbolic techniques reducing LTL model checking to invariant checking

SAT/SMT-based model checking techniques reduce full LTL model checking to invariant model checking. Such techniques typically apply a construction called *degeneralisation*, which is able to combine all the fairness conditions $f^i_{\neg\phi}$ into a single fairness condition f. For simplicity, in this section, we assume that the problem consists of a single fairness condition f.

Liveness to Safety The *liveness-to-safety reduction* (L2S) [6] is a technique for reducing an LTL model checking problem on a finite-state transition system to an invariant model checking problem. The idea is to encode the absence of a lasso-shaped path violating the LTL property $FG\neg f$ as an invariant property.

The encoding is achieved by transforming the original STS S to the STS S_{L2S}, introducing a set $\overline{X}$ of variables containing a copy $\overline{x}$ for each state variable x of the original system, plus additional variables *seen*, *triggered* and *loop*. Let $S \doteq \langle X, I, T\rangle$. L2S transforms the STS in $S_{\text{L2S}} \doteq \langle X_{\text{L2S}}, I_{\text{L2S}}, T_{\text{L2S}}\rangle$ so that $S \models FG\neg f$ if and only if $S_{\text{L2S}} \models \neg bad_{\text{L2S}}$, where:

$$
\begin{aligned}
X_{\text{L2S}} &\doteq X \cup \overline{X} \cup \{seen, triggered, loop\} \\
I_{\text{L2S}} &\doteq I \wedge \neg seen \wedge \neg triggered \wedge \neg loop \\
T_{\text{L2S}} &\doteq T \wedge \big[\textstyle\bigwedge_X \overline{x} \iff \overline{x}'\big] \\
&\quad \wedge \big[seen' \iff (seen \vee \textstyle\bigwedge_X (x \iff \overline{x}))\big] \\
&\quad \wedge \big[triggered' \iff (triggered \vee (f \wedge seen'))\big] \\
&\quad \wedge \big[loop' \iff (triggered' \wedge \textstyle\bigwedge_X (x' \iff \overline{x}'))\big] \\
bad_{\text{L2S}} &\doteq loop
\end{aligned}
$$

The variables $\overline{X}$ are used to non-deterministically guess a state of the system from which a reachable fair loop starts. The additional variables are used to

remember that the guessed state was seen once and that the signal f was true at least once afterwards.

K-liveness K-liveness[20] is an algorithm that reduces symbolic LTL model checking to invariant checking. In order to prove the validity of an LTL formula ψ in an STS M, the algorithm count the occurrence of the fairness condition in the composed automaton $M \times M_{\neg\psi}$. If the algorithm is able to find a bound k such that the fairness condition f is visited at most k times, then the property is valid. It should be noted that the algorithm is not able to disprove the property; therefore, it is usually executed in lockstep with BMC.

The formal construction of the problem is as follows:

$$M_e = \langle V_\psi \cup \{c\}, I_\psi \wedge c = 0, T_\psi \wedge (f \rightarrow c' = c + 1) \wedge (\neg f \rightarrow c' = c)\rangle$$
$$M \times M_e \models c \leq k$$

where M_ψ is the automata construction of ψ, c is the counter variable, M_e is the automata of ψ extended with the fairness counter and k is a positive integer constant.

5 Symbolic Compilation of Safety LTL

5.1 Symbolic Compilation of Full LTL

Following [21], the encoding of an LTL formula ϕ over variables V into a transition system $M_{\neg\phi} = \langle V_{\neg\phi}, I_{\neg\phi}, T_{\neg\phi}\rangle$ with a set $F_{\neg\phi}$ of fairness conditions is defined as follows:

- $V_{\neg\phi} = V \cup \{v_{X\beta} \mid X\beta \in Sub(\phi)\} \cup \{v_{X(\beta_1 U \beta_2)} \mid \beta_1 U \beta_2 \in Sub(\phi)\} \cup \{v_{Y\beta} \mid Y\beta \in Sub(\phi)\} \cup \{v_{Y\beta_1 S\beta_2} \mid \beta_1 S \beta_2 \in Sub(\phi)\}$
- $I_{\neg\phi} = Enc(\neg\phi) \wedge \bigwedge_{v_{Y\beta} \in V_{\neg\phi}} \neg v_{Y\beta}$
- $T_{\neg\phi} = \bigwedge_{v_{X\beta} \in V_{\neg\phi}} v_{X\beta} \leftrightarrow Enc(\beta)' \wedge \bigwedge_{v_{Y\beta} \in V_{\neg\phi}} Enc(\beta) \leftrightarrow v'_{Y\beta}$
- $F_{\neg\phi} = \{Enc(\beta_1 U \beta_2 \rightarrow \beta_2) \mid \beta_1 U \beta_2 \in Sub(\phi)\}$

where Sub is a function that maps a formula ϕ to the set of its subformulas, and Enc is defined recursively as:

- $Enc(\top) = \top$
- $Enc(v) = v$
- $Enc(\phi_1 \wedge \phi_2) = Enc(\phi_1) \wedge Enc(\phi_2)$
- $Enc(\neg\phi_1) = \neg Enc(\phi_1)$
- $Enc(X\phi_1) = v_{X\phi_1}$
- $Enc(\phi_1 U \phi_2) = Enc(\phi_2) \vee (Enc(\phi_1) \wedge v_{X(\phi_1 U \phi_2)})$
- $Enc(Y\phi_1) = v_{Y\phi_1}$
- $Enc(\phi_1 S \phi_2) = Enc(\phi_2) \vee (Enc(\phi_1) \wedge v_{Y(\phi_1 S \phi_2)})$

This construction can be easily extended to reason about First-Order LTL defined in Section 4.3. The details can be found in [1].

5.2 Symbolic Compilation of SafetyLTL

We propose a novel symbolic compilation of SafetyLTL formulas into STSs, that enables the reduction of their model checking problem to reachability. The resulting algorithm follows the ideas already presented in [25] and [27]. However, the construction of the STS is aligned with the one of full LTL proposed in [21] and recalled in the previous section.

We denote such construction as *SafetyLTL2STS*. The construction of the STS is based on the construction of Section 5.1, the only differences are:

- $\neg\phi$ is rewritten in negative normal form (*nnf*) by pushing $\neg$ to the leafs. Since ϕ is a SafetyLTL formula, then the negative normal form of its negation does not contain the R operator.
- Since the construction is applied to the formula in *nnf*, the encoding *Enc* is extended to the derived operators in the straightforward way (e.g., $Enc(\phi_1 \vee \phi_2)$ is defined as $Enc(\phi_1) \vee Enc(\phi_2)$).
- Instead of constructing $T_{\neg\phi}$ using double implication for $v_{X\beta}$ and $v_{Y\beta}$ we use single implication i.e.

$$T_{\neg\phi} := \bigwedge_{v_{X\beta} \in V_{\neg\phi}} (v_{X\beta} \rightarrow Enc(\beta)') \wedge \bigwedge_{v_{Y\beta} \in V_{\neg\phi}} (v'_{Y\beta} \rightarrow Enc(\beta))$$

- Instead of using the set $F_{\neg\phi}$ of fairness conditions, we generate an invariant INV_ϕ defined as

$$INV_\phi := \neg(\bigwedge_{v_{X\beta} \in V_{\neg\phi}} \neg v_{X\beta})$$

Finally, we define $\mathit{SafetyLTL2STS}(\phi) := \langle M^{saf}_{\neg\phi}, INV_\phi \rangle$.

The idea behind this construction is to find a finite trace that witnesses $\neg\phi$. The variables of the form $v_{X\beta}$ are proof obligations for the next states. Thus, as before, the initial states of the automaton must contain the encoding of $\neg\phi$, initializing the proof obligations of the $v_{X\beta}$ variables. The transition condition propagates the needed proof obligations to the next state. Since $\neg\phi$ is co-safety, eventually a state is reached in which all $v_{X\beta}$ variables are false. This is equivalent to the algorithm in [25] for finding *informative prefixes* (see Definition 7). In fact, we can prove that any trace violating the invariant condition is informative for ϕ.

To clarify further, we provide an example considering the safety LTL formula $\psi_{ex} := G(a \rightarrow Xb)$. The construction of $M_{\neg\psi_{ex}}$ is as follows:

$$\begin{aligned} I_{\neg\psi_{ex}} &:= (a \wedge v_{X\neg b}) \vee v_{XF(a \wedge X\neg b)} \\ T_{\neg\psi_{ex}} &:= (v_{X\neg b} \rightarrow \neg b') \wedge (v_{XF(a\wedge X\neg b)} \rightarrow ((a \wedge X\neg b) \vee v'_{XF(a\wedge X\neg b)})) \\ INV_{\psi_{ex}} &:= v_{Xb} \vee v_{XF(a\wedge X\neg b)} \end{aligned}$$

To falsify ψ_{ex}, our counterexample π must satisfy $I_{\neg\psi_{ex}}$ at position 0; thus, π satisfies one of the two next variables and $INV_{\psi_{ex}}$ is true at the beginning. If b is true at the second step and at the initial step a and v_{Xb} were true, then

we found a finite counterexample to the property $\pi = \{a\}\{\neg b\}$. Otherwise, we might continue for an arbitrary amount of steps in which $v_{XF(a \wedge X \neg b)}$ is true until we find a point i such that $\pi, i \models a \wedge v_{X \neg b}$ and $\pi, i+1 \models b \wedge \neg v_{X \neg b}$. If such a state is not reachable, then ψ_{ex} is valid.

Theorem 1. *Let M be an STS, ϕ be a SafetyLTL formula, $\langle M^{saf}_{\neg\phi}, INV_\phi \rangle = SafetyLTL2STS(\phi)$. Then, $M \models \phi$ iff $M \times M^{saf}_{\neg\phi} \models_{inv} INV_\phi$*

The proof can be find in [1].

5.3 Symbolic construction of live systems

Note that the above constructions use in general prophecy variables that guess the future satisfaction of subformulas and may lead in general to systems that are not live. However, if α is in full-past LTL, then M_α in both constructions is live with respect to α. A live system can be built also for a SafetyLTL formula but with a more expensive BDD-based computation of the fair states (this is for example done for runtime verification of LTL properties, see, e.g., [17]).

6 Algorithm

In this section, we define a novel algorithm to verify relative safety properties. Section 6.1 defines a simple algorithm based on the SafetyLTL compilation to STS that verifies relative safety properties of the form $(\alpha_S \wedge \alpha_L) \rightarrow \varphi$. In section 6.2, we extend that algorithm by iteratively blocking livelock counterexamples. Finally, in section 6.3, we provide an optimization to reduce the amount of iteration required to verify relative safety properties.

6.1 Simple algorithm without loops

```
1  Function verifyRelativeSafetyNoLoop(M, α, φ):
2      M' ← M × M_{α_S};
3      ⟨M^{saf}_{¬φ}, INV_φ⟩ ← SafetyLTL2STS(φ);
4      if M' × M^{saf}_{¬φ} ⊨_inv INV_φ then
5          return VALID
6      end
       /* π_f is a finite cex, try to extend it to α                    */
7      if M' ⊨ α_L → G(¬l(π_f)) then
8          return UNKNOWN
9      end
10     return INVALID
```

Algorithm 1: Algorithm to verify $\alpha \rightarrow \varphi$ without loops

Algorithm 1 verifies that an STS M satisfies $\alpha \rightarrow \varphi$, where φ is a SafetyLTL property and $\alpha = \alpha_S \wedge \alpha_L$ is an LTL property decomposed a safety part α_S and liveness one α_L.

The algorithm computes the STS satisfying α_S (i.e. M_{α_S}) using the construction of section 5.1, it composes it with M into a new STS M'. It uses the construction of section 5.2 to build an automaton $M^{saf}_{\neg\phi}$ and an invariant INV_φ such that $M' \models \varphi$ iff $M' \times M^{saf}_{\neg\phi} \models_{inv} INV_\varphi$. Then, if $M' \times M^{saf}_{\neg\phi} \models_{inv} INV_\varphi$, then $M \models \alpha \rightarrow \varphi$; otherwise, the algorithm provides finite counterexample π_f. However, as discussed in section 2.3, a counterexample for an invariant might not be a real counterexample for φ due to livelocks in M', if M is not live w.r.t. α. Therefore, the algorithm checks if π_f is extensible to infinity through LTL model checking. If $M' \not\models \alpha_L \rightarrow G\neg l(\pi_f)$ (in which $l(\pi_f)$ denotes $\pi_f(|\pi_f| - 1)$, the trace has an infinite continuation that violates φ; on the other hand, if $\alpha_L \rightarrow G\neg l(\pi_f)$ is valid, we cannot determine whether or not $M \models \alpha \rightarrow \varphi$ because π_f contains a livelock with respect to α.

Theorem 2. *Algorithm 1 is sound.*

Proof. We consider the two VALID, INVALID cases:

If $M' \times M^{saf}_{\neg\phi} \models_{inv} INV_\varphi$, there is no finite trace of M' violating φ; Since φ is a safety property there is no infinite trace of M' that violates φ. Moreover, the finite traces of M' are the conjunctions of finite traces of M with the finite traces of α_S (due to the construction of M_{α_S}). Therefore, $M \models \alpha_S \rightarrow \varphi$, which entails that $M \models \alpha \rightarrow \varphi$.

If $M' \not\models \alpha_L \rightarrow G\neg l(\pi_f)$, then the finite counterexample π_f is a prefix of M' and can be extended by a suffix π^ω (which is part of the violation of $\alpha_L \rightarrow G\neg l(\pi_f)$) making $\pi = \pi_f\pi^\omega$ a trace of M' that satisfies α_L and, consequently, a trace of M and a trace of α. Therefore, since π_f is a bad prefix for φ, π is a trace of $\alpha \wedge \neg\varphi$ which is a legitimate counterexample of $\alpha \rightarrow \varphi$.

Theorem 3. *If LTL and invariant model checking terminate, α_S is a full-past LTL formula and M is live w.r.t. to α then Algorithm 1 is complete.*

Proof. It is sufficient to prove that Algorithm 1 does never return UNKNOWN. In fact, it can only return UNKNOWN when there is a counterexample π_f to $M' \times M^{saf}_{\neg\phi} \models_{inv} INV_\varphi$ and $M' \models \alpha_L \rightarrow G\neg l(\pi_f)$. Let σ be the trace of π_f. σ is a bad predix for φ and any extension violates φ. Moreover, α_L is liveness, thus σ can be extended to a trace σ^ω satisfying α_L. Since $M' = M \times M_{\alpha_S}$, $\sigma^\omega \models \alpha_S$ and thus $\sigma^\omega \models \alpha$. By assumption M is live with respect to α. Thus, there exists an infinite path π_M of M over σ^ω. Since α_S is full-past, the construction M_α is also live with respect to α and there exists a path π_α of M_α over σ^ω. Since the variables in common between M and M_α have the same value in π_M and in π_α because assigned by σ^ω, then their composition is a path of M' over the trace σ^ω. Thus, $M' \not\models \alpha_L \rightarrow G\neg l(\pi_f)$.

We observe that if M is a finite-state STS, then both invariant and LTL model checking terminates.

6.2 CEGAR loop algorithm

```
1  Function verifyRelativeSafety(M, α, φ):
2  |  M' ← M × M_α;
3  |  ⟨M^saf_¬φ, INV_φ⟩ ← SafetyLTL2STS(φ);
4  |  if M' × M^saf_¬φ ⊨_inv INV_φ then
5  |  |  return VALID
6  |  end
   |  /* π_f is a finite cex, try to extend it to α                    */
7  |  if M' ⊨ ⋀_{f∈F_α}(GFf) → G(¬l(π_f)) then
8  |  |  M' ← ⟨V' ∪ V_gen, I', T' ∧ ¬last(π_f), F'⟩;
9  |  |  goto 4;
10 |  end
11 |  return INVALID
```

Algorithm 2: Algorithm to verify $\alpha \rightarrow \varphi$

Algorithm 2 generalises algorithm 1 to deal with α with a generic structure by introducing a loop to block livelock states w.r.t. α. The algorithm keeps looping until it either proves or disproves the property.

Theorem 4. *Algorithm 2 is sound.*

Proof. If $M' \times M^{saf}_{\neg\phi} \models_{inv} INV_\varphi$, then there is no finite trace of M' violating φ; therefore, since φ is a safety property there is no infinite trace violating φ that satisfies α without fairness and consequently α with fairness. If $M' \models \bigwedge_{f\in F_\alpha}(GFf) \rightarrow G\neg l(\pi_f)$, then $l(\pi_f)$ is a state in M' that cannot be extended visiting infinitely often F_α; thus, blocking it preserves satisfiability

Finally, as for algorithm 1, if $M' \not\models \bigwedge_{f\in F_\alpha}(GFf) \rightarrow G\neg l(\pi_f)$, then the finite counterexample π_f is a prefix of M' and can be extended by a suffix π^ω (which is part of the violation of $GFf_\alpha \rightarrow G\neg l(\pi_f)$)making $\pi = \pi_f\pi^\omega$ a trace of M' and, consequently, a trace of M and a trace of α. Therefore, since π_f is a bad prefix for φ, π is a trace of $\alpha \wedge \neg\varphi$ which is a legitimate counterexample of $\alpha \rightarrow \varphi$.

Theorem 5. *If M is a finite state STS, algorithm 2 is complete.*

Proof. We omit the proof for Line 4 and line 7 since they are identical to the proof of Algorithm 1. We only need to prove that at some point we stop blocking bad states. Since M is a finite state STS and M_α does not introduce infinite states, then the livelock states are finite and will eventually be all blocked forcing the algorithm to exit either with VALID or INVALID result.

6.3 Extending algorithm with lookahead

The main weakness of Algorithm 2 is the overhead given by the loop iterations for any found livelock. To face this issue, we introduce an optimization that

computes a lookahead of length n on each invariant counterexample. The idea is that the invariant check must compute n successor states after the bad prefix of φ. The new construction is built on top of $SafetyLTL2STS$ as follows:

Definition 9. *We define $Xdepth(\phi)$ recursively as follows:*
(i) $Xdepth(p) = 0$ *(ii)* $Xdepth(\neg\phi) = Xdepth(\phi)$
(iii) $Xdepth(X\phi) = Xdepth(\phi) + 1$ *(iv)* $Xdepth(next(\phi)) = Xdepth(\phi) + 1$
(v) $Xdepth(\phi_1 \vee \phi_2) = max(Xdepth(\phi_1), Xdepth(\phi_2))$
(vi) $Xdepth(\phi_1 U \phi_2) = max(Xdepth(\phi_1), Xdepth(\phi_2))$.

We integrate this construction into Algorithm 2. At the first iteration parameter n is initialised heuristically to $Xdepth(\alpha) + 1$, then it is incremented by one at each step. Moreover, when we check that a finite counterexample is extensible, instead of considering the last state of π_f, we pick the last state of the violation of φ. We observe that all the states after the bad state in π_f are livelocks because otherwise our violation would be extensible. Therefore, we can safely block those states as well.

7 Experimental Evaluation

We evaluated the performances of our algorithm by comparing it to the other two well known algorithms used in the reduction of LTL to invariant model checking: k-liveness and liveness to safety (adapted for infinite state systems as proposed in [23]). All the algorithms are implemented inside the nuXmv symbolic model checker. Since the k-liveness algorithm is not able to disprove properties, it has been executed in lockstep with BMC. In our implementation, the algorithms are constructed on top of MathSAT5[15] SMT-solver, which supports combinations of theories such as $\mathcal{LIA}$, $\mathcal{LRA}$ and $\mathcal{EUF}$. Moreover, k-liveness and the algorithms presented in this paper are built on top of an infinite state version of the IC3 algorithm [14].

The experiments[1]were run in parallel on a cluster with nodes with Intel Xeon CPU 6226R running at 2.9GHz with 32CPU, 12GB. The timeout for each run was one hour and the memory cap was set to 1GB.

We carried out the experimental evaluation comparing the execution time of each algorithm on each instance. In the remainder of this section, we denote the relative safety algorithm with the lookahead construction as rels-la, the relative safety algorithm without lookahead construction as rels-no-la, the liveness to safety algorithm as l2s and kliveness algorithm as klive.

7.1 Benchmarks

The benchmarks have been taken from different sources: (i) Assume-guarantee contracts models of OCRA[10] representing a simplified Wheel Brake System,

[1] The results of the experimental evaluation can be found at `https://es-static.fbk.eu/people/bombardelli/papers/ifm23/ifm23.tar.gz`

Redundant Sensors and other models. (ii) A subset of finite state models from NuSMV examples. (iii) Automatically discretized timed nuXmv benchmarks [12] such as the emergency diesel and a modified version of the Fischer algorithm. (iv) Handcrafted parametrized benchmarks to study the scalability of rels-la compared to kliveness and liveness to safety.

Overall, we collected roughly 850 formulas to be verified by each algorithm. The evaluation considered both valid and invalid properties. Regarding the structure of the formulas, we considered both pure safety properties (i.e. properties in which $\alpha := \top$) and relative safety properties (with the form $\alpha \rightarrow \varphi$). The structure of α was not limited as well, we considered cases in which α was pure safety, pure liveness and also a generic LTL property. Regarding the type of models, they are both from discrete and timed benchmarks, with variables ranging from *enumeratives*, *Booleans*, *integers* and *reals*.

In the following, we describe the handcrafted parameterised benchmarks.

Asynchronous discrete bounded response: This benchmark considers the asynchronous composition of l bounded response components. Each component receives the same input $trig$ and if it keeps receiving the input for n steps, then in m steps it set the variable res to *true*. When a component is not running it stutters i.e. it ignores inputs and its variable remain unchanged.

The property ψ in form $\alpha \rightarrow \varphi$ is as follows: α is a liveness property that guarantees that each component will be scheduled infinitely often i.e. $\alpha := GFrun_i$. φ asks that if a trigger remains true for g_n steps, then in g_m steps a variable result becomes true i.e $\varphi := G(G_{[0,g_n]}trig \rightarrow F_{[0,g_m]}res)$. The parameters g_n and g_m depends on the parameters l, n and m. Each parameter n, m, l is instantiated with different values; moreover, the model gives different assignments to g_n and g_m to produce both valid and invalid properties.

Monitor sensor This benchmark is composed of a sensor and a monitor. The sensor reads a *real* input signal and, if there are no faults, it returns a signal with a perturbation bounded by a constant; otherwise, it replicates its last value forever. The monitor reads the sensor output every q time unit and if it reads the same value twice it raises an alarm.

The property to be verified is as follows. Assuming non-Zenoness of time, if there is a fault an alarm is raised in at most p time units. Parameters p, q are instantiated with different values and as both *integers* and *reals*. Depending on the assignments of p and q the property can be either valid or invalid. Moreover, we considered a variation of the model that introduces livelocks.

7.2 Experimental results and analysis

Figure 1 shows scatter plots comparing the rels-la algorithm with k-liveness, liveness to safety and rels-no-la in terms of execution time. The y-coordinate of each point represents the execution time of rels-la in that particular instance; conversely, the x-coordinate represents the execution time of the other algorithm. When a point is below the diagonal, it means that rels-la is faster on the given

instance. When the point is above the diagonal the other algorithm is faster than rels-la.

Comparison with l2s and k-liveness

Comparison with Liveness to safety: Figure 1b and Figure 1e highlight significant differences in the performance between l2s and rels-la. Although there are few instances that can be solved by l2s in a few seconds and are not solved by rels-la, the vast majority of the instances timed out with liveness to safety. This result is due to the weaknesses that liveness to safety has with infinite-state systems because it needs a finite-state abstraction.

Comparison with K-liveness: As Figure 1d shows, klive appears to be more performant than rels-la with invalid instances even though rels-la still outperforms klive in several invalid instances and overall is able to solve more invalid instances. We think that in many cases BMC, which runs in lockstep with klive, can be significantly faster in finding counterexample because rels-la has an overhead of time to check the extendibility of π_f and can still need multiple interactions to converge. On the other hand, when the liveness part of α is significant, rels-la outperforms klive.

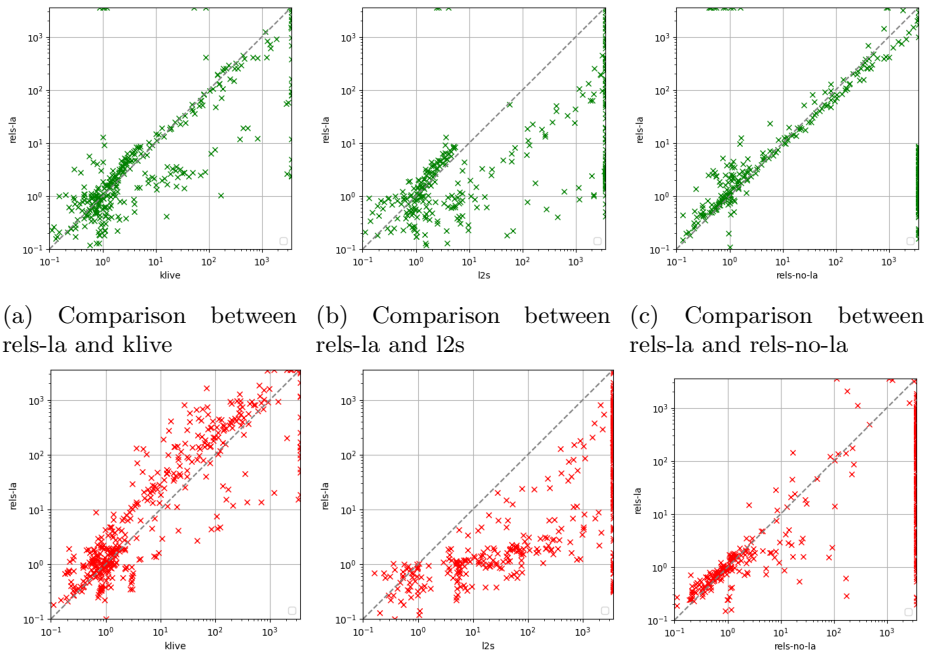
Regarding the valid instances, Figure 1a shows that rels-la outperforms in most cases klive. Few instances times out with rels-la while they are solved in a short time with the klive algorithm and rels-no-la. Apparently, in these instances, the lookahead counter prevents ic3 to converge. Moreover, we observed that with small variation over parameters of the monitor sensor models neither rels-la nor rels-no-la are able to converge. This result occurs because the algorithm keeps finding counterexamples in an unfair region and it is not able to generalise them. Moreover, the lookahead construction is not effective in this case because the trace extendibility depends on the assignment of two parameters. When livelocks are removed from the model the algorithm converges easily.

Relative safety algorithm performs significantly better than k-liveness with benchmarks from *Asynchronous discrete bounded response* and with the valid instances of *mutual exclusion* benchmarks. We think that the reasons why this happens are the following. In these benchmarks, φ is slightly complicated; this characteristic penalizes k-liveness because the construction of the model with the counter needs a degeneralization of $M_{\neg\varphi}$ to produce single fairness. This situation hinders ic3 in the process of inductive invariant construction. On the opposite side, the construction of *SafetyLTL2STS* is quite effective and does not suffer the overhead of the liveness conditions of α; therefore, invariant checking is significantly faster with our algorithm with these instances.

In many instances, the two algorithms appear to have comparable results. We observed that that is often the case when φ and α are both simple safety properties, the model does not contain many livelocks and the fairness is either absent or simple. Our theory is that for such instances k-liveness construction is similar to ours and the counter does not provide an overhead with valid formulas.

However, when the formulas are invalid, we observe that such instances are easier to verify with BMC, which runs in lockstep with klive.

Impact of lookahead construction: The lookahead construction forces each finite counterexample to be extended with n steps. As Figure 1c and Figure 1f shows, this optimization provide a significant performance boost. In particular with the instances that contain several deadlocks or livelocks in M'. The intuition behind this result is that with a lightweight overhead in the invariant verification, the algorithm is able to discard a potentially infinite amount of livelocks that could potentially block the algorithm forever. Moreover, the livelocks caused by prophecy variables for X are all discarded by a sufficiently large lookahead.



(a) Comparison between rels-la and klive (b) Comparison between rels-la and l2s (c) Comparison between rels-la and rels-no-la

(d) Comparison between rels-la and klive (e) Comparison between rels-la and l2s (f) Comparison between rels-la and rels-no-la

Fig. 1: Scatter plots comparing rels-la with the other algorithms. Plots with green crosses represent valid properties while red crosses represent invalid properties.

8 Conclusions and Future Work

In this work, we proposed an approach that extends the standard reduction to safety properties to invariants by exploiting the concept of relative safety. We

thus focused on LTL properties of the form $\alpha \rightarrow \phi$ where ϕ is safety, providing various examples to motivate the choice. We proved that the reduction is complete when the system model is live with respect to the assumption α. We then provided a general algorithm that performs iteratively more than one invariant checking blocking at each time livelocks. We compared the approach to other techniques for generic LTL properties based on reduction to safety showing that is much more efficient in case of valid properties and complementary to BMC in finding counterexamples.

The directions for future work are many. In order to improve the efficiency of the approach we may consider various options: use BMC in parallel to the reduction to find counterexamples; explore incrementality in the different iteration of invariant model checking; investigate more efficient ways to generalize the livelocks to be blocked. In terms of directions for further extensions, we will apply the approach to contract-based compositional reasoning [18] and asynchronous composition of properties [7], we will consider other safety fragments such as the one defined in [11], and we will study the relation with assumption-based runtime verification [16].

References

1. S. T. Alberto Bombardelli, Alessandro Cimatti and M. Zamboni. Symbolic Model Checking of Relative Safety LTL Properties - extended with proofs. `https://es-static.fbk.eu/people/bombardelli/papers/ifm23/ifm23_ext.pdf`.
2. B. Alpern and F. B. Schneider. Defining liveness. *Information processing letters*, 21(4):181–185, 1985.
3. B. Alpern and F. B. Schneider. Recognizing Safety and Liveness. *Distributed Comput.*, 2(3):117–126, 1987.
4. C. W. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Satisfiability Modulo Theories. In *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 1267–1329. IOS Press, 2021.
5. A. Biere, C. Artho, and V. Schuppan. Liveness checking as safety checking. *Electronic Notes in Theoretical Computer Science*, 66(2):160–177, 2002.
6. A. Biere, C. Artho, and V. Schuppan. Liveness checking as safety checking. In *International Workshop on Formal Methods for Industrial Critical Systems*, 2002.
7. A. Bombardelli and S. Tonetta. Asynchronous Composition of Local Interface LTL Properties. In *NFM*, volume 13260 of *Lecture Notes in Computer Science*, pages 508–526. Springer, 2022.
8. R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta. The nuxmv symbolic model checker. In *Proceedings of the 16th International Conference on Computer Aided Verification - Volume 8559*, page 334–342, Berlin, Heidelberg, 2014. Springer-Verlag.
9. E. Y. Chang, Z. Manna, and A. Pnueli. Characterization of Temporal Property Classes. In *ICALP*, volume 623 of *Lecture Notes in Computer Science*, pages 474–486. Springer, 1992.
10. A. Cimatti, M. Dorigatti, and S. Tonetta. Ocra: A tool for checking the refinement of temporal contracts. pages 702–705, 11 2013.
11. A. Cimatti, L. Geatti, N. Gigante, A. Montanari, and S. Tonetta. Reactive Synthesis from Extended Bounded Response LTL Specifications. In *FMCAD*, pages 83–92. IEEE, 2020.

12. A. Cimatti, A. Griggio, E. Magnago, M. Roveri, and S. Tonetta. Extending nuxmv with timed transition systems and timed temporal properties. In *International Conference on Computer Aided Verification*, 2019.

13. A. Cimatti, A. Griggio, E. Magnago, M. Roveri, and S. Tonetta. SMT-based satisfiability of first-order LTL with event freezing functions and metric operators. *Inf. Comput.*, 272:104502, 2020.

14. A. Cimatti, A. Griggio, S. Mover, and S. Tonetta. IC3 Modulo Theories via Implicit Predicate Abstraction. In *TACAS*, volume 8413 of *Lecture Notes in Computer Science*, pages 46–61. Springer, 2014.

15. A. Cimatti, A. Griggio, B. J. Schaafsma, and R. Sebastiani. The MathSAT 5 SMT Solver. 2012.

16. A. Cimatti, C. Tian, and S. Tonetta. Assumption-Based Runtime Verification with Partial Observability and Resets. In *RV*, volume 11757 of *Lecture Notes in Computer Science*, pages 165–184. Springer, 2019.

17. A. Cimatti, C. Tian, and S. Tonetta. Assumption-based Runtime Verification. *Formal Methods Syst. Des.*, 60(2):277–324, 2022.

18. A. Cimatti and S. Tonetta. Contracts-refinement proof system for component-based embedded systems. *Sci. Comput. Program.*, 97:333–348, 2015.

19. K. Claessen, N. Eén, and B. Sterin. A circuit approach to ltl model checking. *2013 Formal Methods in Computer-Aided Design*, pages 53–60, 2013.

20. K. Claessen and N. Sörensson. A liveness checking algorithm that counts. *2012 Formal Methods in Computer-Aided Design (FMCAD)*, pages 52–59, 2012.

21. E. M. Clarke, O. Grumberg, and K. Hamaguchi. Another look at ltl model checking. *Formal Methods in System Design*, 10:47–71, 1994.

22. D. D. Cofer, A. Gacek, S. P. Miller, M. W. Whalen, B. LaValley, and L. Sha. Compositional Verification of Architectural Models. In *NASA Formal Methods*, volume 7226 of *Lecture Notes in Computer Science*, pages 126–140. Springer, 2012.

23. J. Daniel, A. Cimatti, A. Griggio, S. Tonetta, and S. Mover. Infinite-state liveness-to-safety via implicit abstraction and well-founded relations. volume 9779, pages 271–291, 07 2016.

24. T. A. Henzinger. Sooner is safer than later. *Inf. Process. Lett.*, 43:135–141, 1992.

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

26. L. Lamport. Proving the Correctness of Multiprocess Programs. *IEEE Trans. Software Eng.*, 3(2):125–143, 1977.

27. T. Latvala. Efficient Model Checking of Safety Properties. In *SPIN*, volume 2648 of *Lecture Notes in Computer Science*, pages 74–88. Springer, 2003.

28. O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In *Workshop on Logic of Programs*, pages 196–218. Springer, 1985.

29. A. Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE Computer Society, 1977.

30. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Symposium on Logic in Computer Science*, pages 322–331. IEEE Computer Society, 1986.

A Details

A.1 Reducing First-Order LTL to Propositional LTL

Definition 10. *Let ϕ an LTL formula. We denote $NextPred(\phi)$ recursively as follows:*

$$
\begin{aligned}
NextPred(X\phi_1) :=& NextPred(\phi_1) \\
NextPred(\phi_1 U \phi_2) :=& NextPred(\phi_1) \cup NextPred(\phi_2) \\
NextPred(Y\phi_1) :=& NextPred(\phi_1) \\
NextPred(\phi_1 S \phi_2) :=& NextPred(\phi_1) \cup NextPred(\phi_2) \\
NextPred(c) :=& \emptyset \quad NextPred(x) := \emptyset \\
NextPred(\neg\phi_1) :=& \begin{cases} NextPred(\phi_1) & \text{If } \phi_1 \notin NextPred(\phi_1) \\ \{\neg\phi_1\} & \text{Otherwise} \end{cases} \\
NextPred(\phi_1 \vee \phi_2) :=& \begin{cases} NextPred(\phi_1 \vee \phi_2) & \text{If } \phi_1 \notin NextPred(\phi_1) \text{ and } \phi_2 \notin NextPred(\phi_2) \\ \{\phi_1 \vee \phi_2\} & \text{Otherwise} \end{cases} \\
NextPred(p(u_1, \ldots, u_n)) :=& \begin{cases} \emptyset & \text{If } \forall_{0<i\leq n} NextPred(u_i) = \emptyset \\ \{p(u_1, \ldots, u_n)\} & \text{Otherwise} \end{cases} \\
NextPred(func(u_1, \ldots, u_n)) :=& \begin{cases} \emptyset & \text{If } \forall_{0<i\leq n} NextPred(u_i) = \emptyset \\ \{func(u_1, \ldots, u_n)\} & \text{Otherwise} \end{cases} \\
NextPred(next(u)) :=& \{next(u)\}
\end{aligned}
$$

Definition 11. *Let M be an STS, ϕ be an LTL formula and $Nexts = NextPred(\phi)$, we define M_x, ϕ_x as follows:*

$$
\begin{aligned}
M_x :=& \langle V \cup \{v_{pred_x} | pred_x \in Nexts\}, I \wedge \bigwedge_{pred_x \in Nexts} (\neg v_{pred_x}), \\
& T \wedge \bigwedge_{pred_x \in Nexts} (v'_{pred_x} \leftrightarrow pred_x) \rangle \\
\phi_x :=& \phi\lceil Nexts / \{X v_{pred_x} | pred_x \in Nexts\} \rfloor
\end{aligned}
$$

B SafetyLTL2STS Correctness

In this section we prove the correctness of the symbolic compilation of SafetyLTL defined in Section 5.2. In the following, $M^{saf}_{\neg\phi}$ denotes the result of such construction.

Theorem 6. *Let us consider $\sigma \in \Sigma^\omega$ and a SafetyLTL property ϕ. If there exists a path π of $M^{saf}_{\neg\phi}$ that violates the invariant $Inv(\phi)$, then $\pi \not\models \phi$.*

Proof. Given a subformula ψ of ϕ, we prove that for all $i \geq 0$, if $\pi[i] \models enc(\psi)$, then $\pi[i..] \models \psi$. From this the theorem follows immediately, since the initial state $\pi[0] \models enc(\neg\phi)$.

We prove the claim by induction on ψ:

- $\psi = p$: $\psi \in \pi[i]$ by construction
- $\psi = \psi_1 \vee \psi_2$: $\psi_1 \in \pi[i] \vee \psi_2 \in \pi[i]$ by construction
- $\psi = \mathcal{X}\psi'$: $\mathcal{X}\psi \in el(\pi[i])$ by construction, then the transition requires that $\pi(i+1) \models enc(\psi')$; thus, $\pi, i+1 \models \psi'$ by induction
- $\psi = \psi_1 \mathcal{U} \psi_2$: we prove this case by induction on i; for the base case, we consider the index i in which $\pi[i]$ violates the invarian $Inv(\phi)$; in this case, all variables in $el(\phi)$ are false in $\pi[i]$ and the claim follows trivially; suppose the claim holds for $i+1$, we prove it for i: $\mathcal{X}(\psi_1 \mathcal{U} \psi_2) \in el(\pi[i])$ by construction, then the transition requires either that $\pi(i+1) \models enc(\psi_2)$ and thus $\pi, i+1 \models \psi_2$ by induction, or that $\pi(i+1) \models enc(\psi_1 \wedge \mathcal{X}(\psi_1 \mathcal{U} \psi_2))$ and thus $\pi, i+1 \models \psi$ by induction.
- $\psi = \mathcal{Y}\psi'$: $\mathcal{Y}\psi \in el(\pi[i])$ by construction, then if $i > 0$ the transition requires that $\pi(i-1) \models \psi'$; thus, $\pi, i-1 \models \psi'$
- $\psi = \psi_1 \mathcal{S} \psi_2$: we prove this case by induction on i; for the base case, we consider the index 0; in this case, all variables $v_{Y\beta}$ are false in $\pi[0]$ and the claim follows trivially; suppose the claim holds for $i-1$, we prove it for i: $\mathcal{Y}(\psi_1 \mathcal{S} \psi_2) \in el(\pi[i])$ by construction, then if $i > 0$ the transition require either that $\psi_2 \in el(\pi(i-1))$ and thus $\pi[i-1] \models \psi_2$ by induction, or that $\psi_1 \in \pi(i-1) \wedge \mathcal{Y}(\psi_1 \mathcal{S} \psi_2) \in \pi(i-1)$ and thus $\pi, i-1 \models \psi$ by induction.
- the other cases are similar to the previous ones.

Theorem 7. *Let us consider $\sigma \in \Sigma^\omega$ and a safety property ϕ. If $\sigma \not\models \phi$ then there exists a trace of $M^{saf}_{\neg\phi}$ over σ that violates the invariant $Inv(\phi)$.*

Proof. Let us define the function *State* as follows:

- $State(a, \sigma) = \emptyset$
- $State(\psi_1 \wedge \psi_2, \sigma) = State(\psi_1, \sigma) \cup State(\psi_2, \sigma)$
- $State(\psi_1 \vee \psi_2, \sigma) = \begin{cases} State(\psi_1, \sigma) & \text{if } \sigma \models \psi_1 \\ State(\psi_2, \sigma) & \text{else} \end{cases}$
- $State(X\psi, \sigma) = \{v_{X\psi}\}$
- $State(\psi_1 U \psi_2, \sigma) = \begin{cases} State(\psi_2, \sigma) & \text{if } \sigma \models \psi_2 \\ State(\psi_1, \sigma) \cup \{v_{X(\psi_1 U \psi_2}\} & \text{else} \end{cases}$
- $State(Y\psi, \sigma) = \{v_{Y\psi}\}$
- $State(\psi_1 S \psi_2, \sigma) = \begin{cases} State(\psi_2, \sigma) & \text{if } \sigma \models \psi_2 \\ State(\psi_1, \sigma) \cup \{v_{Y(\psi_1 S \psi_2}\} & \text{else} \end{cases}$

Given a trace σ and a safety property ϕ such that $\sigma \not\models \phi$. Let us define the sequence of states s_i as follows: $s_0 = State(enc(\neg\phi))$; for all $i > 0$, $s_i = State(\bigwedge_{v_{X\beta} \in s_{i-1}} \beta)$. The sequence $\pi = s_0, s_1, \ldots$ is a path of $M^{saf}_{\neg\phi}$ over the trace σ. Let d be the size of the bad prefix of σ violating ϕ. Then $\pi[d]$ violates the invariant $Inv(\phi)$.

Corollary 1. $M \models \phi$ *iff* $M \times M^{saf}_{\neg\phi} \models Inv(\phi)$