The Safety Simple Subset ⋆

Shoham Ben-David[1] Dana Fisman[2,3⋆⋆] Sitvanit Ruah[3]

[1] University of Waterloo
[2] Weizmann Institute of Science
[3] IBM Haifa Research Lab

Abstract. Regular-LTL (RLTL), extends LTL with regular expressions, and it is the core of the IEEE standard temporal logic PSL. Safety formulas of RLTL, as well as of other temporal logics, are easier to verify than other formulas. This is because verification of safety formulas can be reduced to invariance checking using an auxiliary automaton recognizing violating prefixes.
In this paper we define a special subset of safety RLTL formulas, called RLTL$^{\mathsf{LV}}$, for which the automaton built is *linear* in the size of the formula. We then give two procedures for constructing such an automaton, the first provides a translation into a regular expression of linear size, while the second constructs the automaton directly from the given formula. We have derived the definition of RLTL$^{\mathsf{LV}}$ by combining several results in the literature, and we devote a major part of the paper to reviewing these results and exploring the involved relationships.

1 Introduction

The specification language PSL [10] is an IEEE standard temporal logic which is widely used in industry, both for simulation and for model checking. While PSL is a rich and expressive language, some of its formulas are hard to implement and expensive to verify. For this reason, an effort has been made in the PSL language reference manual (LRM), to define the *simple subset*, which should consist of formulas that are easy to implement and to verify. The LRM, however, provides no proof for the simplicity of the subset.

In this paper we deal with the simple subset of the LRM. We note that this subset, as defined in the LRM, is geared towards simulation tools. In order to accommodate model checking as well, we focus on the subset of the LRM simple subset, that consists of safety formulas only. A formula φ is a safety formula if every violation of φ occurs after a finite execution of the system. Verification of safety formulas is easier for model checking than verification of general formulas.

The core of PSL is the language Regular-LTL (RLTL), which extends linear temporal logic LTL [16] with regular expressions. The verification of a general RLTL formula typically involves the construction of an automaton on *infinite* words with size exponential in the size of the formula [**?**]. The restriction to safety formulas allows the use of automata on *finite* words. The use of a finite-word automata is important. It allows

⋆ This work was partially supported by the European Community project FP6-IST-507219 (PROSYD)

⋆⋆ The work of this author was carried out at the John von-Neumann Minerva Center for the Verification of Reactive Systems.

verification of safety formulas to be reduced to invariance checking, by stating the invariance "The violation automaton is not in an accepting state". Invariance verification is, for most of the verification methods, easier to perform than verification of a general RLTL formula, making safety formulas easier to verify than others.

Kupferman and Vardi in [13], classify safety formulas according to efficiency of verification. They define pathologically safe formulas, and show that non-pathologically safe formulas are easier to verify than pathologically safe ones. In particular, they show that violation of non-pathologically safe formulas can be detected by an automaton on finite words with size exponential in the size of the formula.

In this document we define a subset of safety RLTL formulas, for which violation can be detected by an automaton on finite words with size *linear* in the size of the formula. This subset corresponds to the definition of the safety simple subset in PSL's LRM. We term this subset $\text{RLTL}^{\mathsf{LV}}$, where LV stands for "linear violation". The definition of $\text{RLTL}^{\mathsf{LV}}$ restricts the syntax of the formulas to be specified. However, experience shows that the vast majority of the safety formulas written in practice, are expressible in $\text{RLTL}^{\mathsf{LV}}$. Thus this subset is both easier to verify and very useful in practice.

We provide two procedures for the construction of a linear-sized automaton (NFA) for an $\text{RLTL}^{\mathsf{LV}}$ formula. The first goes through building a linear-sized regular expression, and the other directly constructs the automaton from the given formula. These procedure can serve PSL tool implementors.

We have derived the definition of $\text{RLTL}^{\mathsf{LV}}$ by combining several results in the literature. We devote a major part of the paper to reviewing these results and exploring the involved relationships.

Maidl [14] has studied the common subset of the temporal logics LTL and ACTL. She defined a syntactic subset of LTL, $\text{LTL}^{\mathsf{DET}}$, such that every formula in the common fragment of LTL and ACTL has an equivalent in $\text{LTL}^{\mathsf{DET}}$. By this she strengthened the result of Clarke and Draghicescu [4] who gave a characterization of the CTL formulas that can be expressed in LTL and the result of Kupferman and Vardi [**?**] who solved the opposite problem of deciding whether an LTL formula can be specified in the alternation free μ-calculus. She further showed that for formulas in $\text{LTL}^{\mathsf{DET}}$ there exists a 1-weak Büchi automaton, linear in the size of the formula, recognizing the negation of the formula. The subset $\text{RLTL}^{\mathsf{LV}}$ can be seen as the safety subset of $\text{LTL}^{\mathsf{DET}}$, augmented with regular expressions. The augmentation with regular expressions is important as it increases the expressive power. While Maidl was engaged with the expressiveness of LTL and ACTL, it is interesting to observe that the safety subset of $\text{LTL}^{\mathsf{DET}}$ is associated with efficient verification.

In [1] Beer et al. have defined the logic RCTL, an extension of CTL with regular expressions via a suffix implication operator. They were interested in "on-the-fly model checking", which in our terminology, is called invariance verification. We therefore term their subset $\text{RCTL}^{\mathsf{OTF}}$, where OTF stands for "on-the-fly". Beer et al. gave a procedure to translate an $\text{RCTL}^{\mathsf{OTF}}$ formula into a regular expression. Our results extend theirs, in the sense that the subset $\text{RCTL}^{\mathsf{LV}}$ which corresponds to $\text{RLTL}^{\mathsf{LV}}$ subsumes $\text{RCTL}^{\mathsf{OTF}}$. We elaborate more on this in Section 4.

The remainder of the paper is organized as follows. In Section 2 we provide some preliminaries. In Section 3 we define $\text{RLTL}^{\mathsf{LV}}$ and provide a succinct construction yield-

ing a regular expression of linear size recognizing violation. In Section 4 we discuss the relation of this subset with the results of Kupferman and Vardi [13], Maidl [14] and Beer et al. [1]. In Section 5 we provide a direct construction for an NFA of linear size, and in Section 6 we conclude.

2 Preliminaries

2.1 Notations

Given a set of atomic propositions $\mathbb{P}$, we use $\Sigma_{\mathbb{P}}$ to denote the alphabet $2^{\mathbb{P}} \cup \{\top, \bot\}$. That is, the set of interpretations of atomic propositions extended with two special symbols $\top$ and $\bot$.[1] We use $\mathbb{B}_{\mathbb{P}}$ to denote the set of boolean expressions over $\mathbb{P}$, which we identify with $2^{2^{\mathbb{P}}}$. That is, every boolean expression is associated with the set of interpretations satisfying it. The boolean expressions *true* and *false* denote the elements $2^{\mathbb{P}}$ and $\emptyset$ of $\mathbb{B}_{\mathbb{P}}$, respectively. When the set of atomic propositions is assumed to be known, we often omit the subscript $\mathbb{P}$ from $\Sigma_{\mathbb{P}}$ and $\mathbb{B}_{\mathbb{P}}$.

We denote a boolean expression by b, c or d, a letter from Σ by ℓ (possibly with subscripts) and a word from Σ by u, v, or w. The *concatenation* of u and v is denoted by uv. If u is infinite, then $uv = u$. The empty word is denoted by ϵ, so that $w\epsilon = \epsilon w = w$. If $w = uv$, we define $w/u = v$ and we say that u is a *prefix* of w, denoted $u \preceq w$, that v is a *suffix* of w, and that w is an *extension* of u, denoted $w \succeq u$.

We denote the length of a word w by $|w|$. The empty word $w = \epsilon$ has length 0, a finite non-empty word $w = (s_0 s_1 s_2 \cdots s_n)$ has length $n + 1$, and an infinite word has length ∞. We use i, j, and k to denote non-negative integers. For $i < |w|$, we use w^i to denote the $(i + 1)^{th}$ letter of w (since counting of letters starts at zero). We denote by $\overline{w}$ the word obtained by replacing every $\top$ in w with a $\bot$ and vice versa. We refer to $\overline{w}$ as the *dual* of w.

We denote a set of finite/infinite words by U, V or W and refer to them as *properties*. The *concatenation* of U and V, denoted UV, is the set $\{uv \mid u \in U,\ v \in V\}$. Define $V^0 = \{\epsilon\}$ and $V^k = VV^{k-1}$ for $k \geq 1$. The *Kleene closure* of V, denoted V^*, is the set $V^* = \bigcup_{k<\omega} V^k$. The infinite concatenation of V to itself is denoted V^ω. The union $V^* \cup V^\omega$ is denoted V^∞. For a letter ℓ we use ℓ^* and ℓ^ω as abbreviations of $\{\ell\}^*$ and $\{\ell\}^\omega$, respectively.

2.2 Regular Expressions and Finite Automata

Traditional regular expressions are defined using the operators of concatenation ($\cdot$), union ($\cup$) and Kleene closure (*). We define regular expressions using also the operator *fusion* ($\circ$), also known as *overlapping concatenation*. The fusion operator does not add expressive power to regular expressions (see e.g. [1]).

Definition 1 (Regular Expressions (REs)) *Let b be a boolean expression. The set of* REs *is recursively defined as follows:*

$$r ::= b \mid r \cdot r \mid r \cup r \mid r^* \mid r \circ r$$

[1] The role of $\top$ and $\bot$ in the alphabet is explained in Section 2.3.

The semantics of regular expressions is defined inductively, using the semantics of boolean expressions in the base case. For a boolean expression $b \in \mathbb{B}$ and a letter $\ell \in \Sigma$, we define the boolean satisfaction relation $\models$ as follows. For $\ell \in 2^{\mathbb{P}}$, we define $\ell \models b \iff \ell \in b$. We define $\top \models b$ and $\bot \not\models b$. Note that in particular $\top \models \mathit{false}$ and $\bot \not\models \mathit{true}$.

Definition 2 (Tight Satisfaction) *Let v denote a finite (possibly empty) word over Σ; b denote a boolean expression; and r, r_1, and r_2 denote* RES*. The notation $v \models\!\models r$ means that v tightly satisfies r. The relation $\models\!\models$ is defined as follows:*

- $v \models\!\models b \iff |v| = 1$ *and* $v^0 \models b$
- $v \models\!\models r_1 \cdot r_2 \iff \exists v_1, v_2$ *s.t.* $v = v_1 v_2$ *and* $v_1 \models\!\models r_1$ *and* $v_2 \models\!\models r_2$
- $v \models\!\models r_1 \cup r_2 \iff v \models\!\models r_1$ *or* $v \models\!\models r_2$
- $v \models\!\models r^* \iff$ *either* $v = \epsilon$ *or* $\exists v_1, v_2$ *s.t.* $v_1 \neq \epsilon$, $v = v_1 v_2$, $v_1 \models\!\models r$ *and* $v_2 \models\!\models r^*$
- $v \models\!\models r_1 \circ r_2 \iff \exists v_1, v_2$ *and* ℓ *s.t.* $v = v_1 \ell v_2$ *and* $v_1 \ell \models\!\models r_1$ *and* $\ell v_2 \models\!\models r_2$

We use $\mathcal{L}(r)$ to denote the set of finite words tightly satisfying r. That is $\mathcal{L}(r) = \{w \in \Sigma^* \mid w \models\!\models r\}$.

The literature usually provides different definitions for automata over finite vs. infinite words. When infinite words are considered by *finite acceptance* [18] (i.e. an automaton accepts an infinite word if it accepts some prefix of the word by the standard notion of acceptance for finite words) the same automaton can serve for both finite and infinite words. Automata are usually defined over a syntactic alphabet. We define them over a semantic alphabet, using a given set of atomic propositions as follows.

Definition 3 (NFA**,** CO-UFA**)** *An* automaton *$\mathcal{A}$ is a tuple $\mathcal{A} = \langle P, Q, Q_0, \delta, F\rangle$ consisting of the following components:*

- *$P = \{p_1, \ldots, p_n\}$: A finite set of* atomic propositions
- *Q: A finite set of automata* locations
- *$Q_0 \subseteq Q$: A set of* initial *locations*
- *$\delta \subseteq Q \times \mathbb{B}_P \times Q$: A transition relation*
- *$F \subseteq Q$: A set of* final *locations*

Let $\mathcal{A}$ be an automaton for which the above components have been defined. The input to $\mathcal{A}$ is a finite/infinite word over Σ_P. We define a run *of $\mathcal{A}$ over a word $w = w^0 w^1 \ldots$ to be a finite or infinite non-empty sequence $\sigma : q_0 q_1 \ldots$ of locations in Q satisfying the requirements of* initiality *i.e. that $q_0 \in Q_0$; and of* consecution *i.e. that $|\sigma| \leq |w| + 1$ and for each $0 \leq j < |w|$ there exists $b \in \mathbb{B}_P$ such that $(q_j, b, q_{j+1}) \in \delta$ and $w^j \models b$. A run satisfying the requirement of* maximality *i.e. that it is either infinite, or terminates at a location q_k which has no successors w.r.t. δ is termed a* maximal run*. Let $\sigma : q_0 q_1 q_2 \ldots$ be a run of $\mathcal{A}$ over a word w. The run σ* accepts *w iff w is finite and $q_{|w|} \in F$ or w is infinite and there exists $0 \leq i < |w|$ such that $q_i \in F$. The run σ* co-accepts *w iff $q_i \notin F$ for all i such that $0 \leq i < |w|$. If acceptance is determined by the existence of at least one accepting run then we refer to $\mathcal{A}$ as an* NFA *(non-deterministic or existential finite automaton). If acceptance is determined by the fact that all runs are co-accepting then we refer to $\mathcal{A}$ as a* CO-UFA *(co-universal finite automaton).*

The expressive power of regular expressions is the same as that of automata over finite words (both NFAs and co-UFAs) [9]. Moreover, as stated formally below, given a regular expression r (that may be composed using fusion as well) it is possible to generate an equivalent NFA which is *linear* in the size of the RE.

Fact 4 (see e.g. [11]) *Let r be an* RE. *There exists an* NFA N_r *with size linear in $|r|$ such that for every word v, N_r accepts v iff $v \models r$.*

2.3 The Logic RLTL

The logic Regular-LTL (RLTL) extends LTL [16] in two ways. First it interprets the formulas over finite (possibly truncated and possibly empty) as well as infinite words. Second it adds regular expressions to the logic.

In order to incorporate finite words, the syntax of LTL is extended to include two *next-time* operators, weak (X) and strong ($\mathsf{X!}$) [15, pp. 272-273]. The semantics distinguish between the weak and strong versions only on finite words and only on the last letter of a finite word: $\mathsf{X}\,\varphi$ holds on the last letter of any word for any φ, and $\mathsf{X!}\,\varphi$ does not. Similarly, since the logic is interpreted over the empty word as well, there are two versions of a boolean expression: weak and strong [6]. The strong boolean expression is satisfied over a word if the first letter satisfies the boolean expression, and the weak boolean expression is satisfied also if there is no first letter, i.e. if the word is empty.

Regular expressions are added to the logic via the operator $\mapsto$ or its dual $\diamondsuit\!\!\rightarrow$. We refer to the $\mapsto$ operator as the *suffix implication operator* $\mapsto$, since $r \mapsto \varphi$ (read r "suffix-implies" φ) requires that *if* there exists a non-empty prefix of the path tightly satisfying r *then* the suffix starting at the last letter of the prefix should satisfy φ. We refer to the $\diamondsuit\!\!\rightarrow$ operator as the *suffix conjunction operator*, since $r \diamondsuit\!\!\rightarrow \varphi$ (read r "suffix-and" φ) demands the existence of a non-empty prefix of the path tightly satisfying r *and* that the suffix starting at the last letter of the prefix satisfy φ. These operators are essentially the "diamond" and "box" modalities of propositional dynamic logic (PDL) [8], resp.

The syntax of RLTL is formally defined as follows.

Definition 5 (RLTL**)** *Let b a boolean expression and r a regular expression (*RE*). The set of* RLTL *formulas is recursively defined as follows:*

$$\varphi ::= b! \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathsf{X!}\varphi \mid \varphi \mathsf{U} \varphi \mid r \mapsto \varphi$$

Additional operators are defined as syntactic sugaring of the above operators:

- $\varphi \vee \psi \stackrel{\text{def}}{=} \neg(\neg\varphi \wedge \neg\psi)$
- $\varphi \rightarrow \psi \stackrel{\text{def}}{=} \neg\varphi \vee \psi$
- $b \stackrel{\text{def}}{=} \neg(\neg b!)$
- $\varphi\ \mathsf{W}\ \psi \stackrel{\text{def}}{=} \neg(\neg\psi\ \mathsf{U}\ (\neg\varphi \wedge \neg\psi))$
- $\mathsf{F}\,\varphi \stackrel{\text{def}}{=} \mathit{true}\ \mathsf{U}\ \varphi$
- $r \diamondsuit\!\!\rightarrow \varphi \stackrel{\text{def}}{=} \neg(r \mapsto \neg\varphi)$
- $\mathsf{X}\,\varphi \stackrel{\text{def}}{=} \neg(\mathsf{X!}\,\neg\varphi)$
- $\mathsf{G}\,\varphi \stackrel{\text{def}}{=} \varphi\ \mathsf{W}\ \mathit{false}$
- $r! \stackrel{\text{def}}{=} r \diamondsuit\!\!\rightarrow \mathit{true}$

The formula $r!$ holds on a word w iff there exists a prefix u of w which satisfies r tightly.

The semantics of RLTL formulas is defined inductively, using the semantics of boolean and regular expressions in the base case, as follows.

Definition 6 (Formula Satisfaction) *Let v denote a word over Σ; b a boolean expression; r an* RE*; and φ and ψ* RLTL *formulas. The notation $v \models \varphi$ means that v satisfies φ. The relation $\models$ is defined as follows:*

1. $v \models b! \iff |v| > 0$ *and* $v^0 \Vdash b$
2. $v \models \neg\varphi \iff \overline{v} \not\models \varphi$
3. $v \models \varphi \wedge \psi \iff v \models \varphi$ *and* $v \models \psi$
4. $v \models \mathsf{X!}\,\varphi \iff |v| > 1$ *and* $v^{1\cdot\cdot} \models \varphi$
5. $v \models \varphi \mathsf{U} \psi \iff \exists\, 0 \leq k < |v|$ *s.t.* $v^{k\cdot\cdot} \models \psi$ *and* $\forall\, 0 \leq j < k,\ v^{j\cdot\cdot} \models \varphi$
6. $v \models r \mapsto \psi \iff \forall\, 0 \leq j < |v|$ *s.t.* $v^{0..j} \Vdash r$, $v^{j\cdot\cdot} \models \psi$

We use $[\![\varphi]\!]$ to denote the set of finite/infinite words satisfying φ. That is $[\![\varphi]\!] = \{w \in \Sigma^\infty \mid w \models \varphi\}$.

Note that, as suggested in [6], the semantics is defined with respect to finite (possibly empty) as well as infinite words over the alphabet $\Sigma = 2^{\mathbb{P}} \cup \{\top, \bot\}$. That is, the alphabet consists of the set of interpretations of atomic propositions extended with two special symbols $\top$ and $\bot$. Below we mention a few facts about the role of $\top$ and $\bot$ in the semantics, for a deep understanding please refer to [6].

By definition we have that $\top$ satisfies every boolean expression including *false* while $\bot$ satisfies no boolean expression and in particular, not even the boolean expression *true*. By the inductive definition of the semantics, we get that $\top^\omega$ satisfies every formula while $\bot^\omega$ satisfies none. Using the notions of [7] if $v\top^\omega \models \varphi$ we say that v *satisfies* φ *weakly* and denote it $v \models^{-} \varphi$. If $v\bot^\omega \models \varphi$ we say that v *satisfies* φ *strongly* and denote it $v \models^{+} \varphi$. If $v \models \varphi$ we often say that v *satisfies* φ *neutrally*. The *strength relation theorem* [7] gives us that if v satisfies φ strongly then it also satisfies it neutrally, and if v satisfies φ neutrally then it also satisfies it weakly. The *prefix/extension theorem* [7] gives us that if v satisfies φ strongly then any extension w of v also satisfies φ strongly. And, if v satisfies φ weakly then any prefix u of v also satisfies φ weakly.

Safety and Liveness Intuitively, a formula is said to be safety iff it characterizes that "something bad" will never happen. A formula is said to be liveness iff it characterizes that "something good" will eventually happen. More formally, a formula φ defines a *safety* property iff any word violating φ contains a finite prefix all of whose extensions violate φ. A formula φ defines a *liveness* property iff any arbitrary finite word has an extension satisfying φ. We use the definitions of safety and liveness as suggested in [6]. These definitions modify those of [15] to reason about finite words as well.

Definition 7 (Safety,Liveness) *Let $W \subseteq \Sigma^\infty$.*

- *W is a* safety *property if for all $w \in \Sigma^\infty - W$ there exists a finite prefix $u \preceq w$ such that for all $v \succeq u$, $v \in \Sigma^\infty - W$.*
- *W is a* liveness *property if for all finite u, there exists $v \succeq u$ such that $v \in W$.*

A formula φ is said to be a *safety (liveness)* formula iff $[\![\varphi]\!]$ is a safety (liveness) property. Some formulas are neither safety nor liveness. For instance, $\mathsf{G}\, p$ is a safety formula, $\mathsf{F}\, q$ is a liveness formula, and $p\ \mathsf{U}\ q$, equivalent to $(p\ \mathsf{W}\ q) \wedge \mathsf{F}\, q$, is neither.

3 The Subset of Linear Violation

Recall that a formula is said to be safety iff it characterizes that "something bad" will never happen. Thus a safety formula is violated iff something bad has happened. Since the "bad thing" happens after a finite execution, for any safety formula it is possible to characterize the set of violating prefixes by an automaton on finite words. In this document we focus on formulas for which this violation can be characterized by an automaton on finite words and by a regular expression, both of *linear* size.

Intuitively, a word v violates a formula φ iff v carries enough evidence to conclude that φ does not hold on v and any of its extensions. Using the terminology of [7] we get that a word v violates φ iff $v \models^{+} \neg\varphi$. Thus, we define:

Definition 8 *Let φ be a safety formula. We say that a set of words L* recognizes violation *of φ iff $L = \{v \mid v \models^{+} \neg\varphi\}$.*

The following definition captures the set of RLTL formulas whose violation can be recognized by a automaton or a regular expression, both of linear size. We denote this set by $\text{RLTL}^{\mathsf{LV}}$ (where LV stands for "linear violation").

Definition 9 ($\text{RLTL}^{\mathsf{LV}}$) *If b is a boolean expression, r is an* RE *and φ, φ_1 and φ_2 are $\text{RLTL}^{\mathsf{LV}}$ then the following are in $\text{RLTL}^{\mathsf{LV}}$:*

1. b
2. $\varphi_1 \wedge \varphi_2$
3. $\mathsf{X}\,\varphi$
4. $(b \wedge \varphi_1) \vee (\neg b \wedge \varphi_2)$
5. $(b \wedge \varphi_1)\ \mathsf{W}\ (\neg b \wedge \varphi_2)$
6. $r \mapsto \varphi$

Construction via Violating RE

In the following we define inductively the RE recognizing the violation of $\text{RLTL}^{\mathsf{LV}}$. By Fact 4 this RE can be translated into an automaton on finite words linear in the size of the RE. Since the RE itself is linear in the size of the input formula, this gives a procedure for generating an automaton of linear size for every formula in $\text{RLTL}^{\mathsf{LV}}$. In Section 5 we provide a direct procedure for generating an automaton of linear size. The benefits of the regular expression are in its succinctness and in the ability to use existing code for translating regular expressions into finite automata.

Definition 10 (Violating RE**)** *Let b be a boolean expression, r an* RE*, and $\varphi, \varphi_1, \varphi_2$ $\text{RLTL}^{\mathsf{LV}}$ formulas. The violating* RE *of an $\text{RLTL}^{\mathsf{LV}}$ formula φ, denoted $\mathcal{V}(\varphi)$ is defined as follows:*

1. $\mathcal{V}(b) = \neg b$
2. $\mathcal{V}(\varphi_1 \wedge \varphi_2) = \mathcal{V}(\varphi_1) \cup \mathcal{V}(\varphi_2)$
3. $\mathcal{V}(\mathsf{X}\varphi) = \mathit{true} \cdot \mathcal{V}(\varphi)$
4. $\mathcal{V}((b \wedge \varphi_1) \vee (\neg b \wedge \varphi_2)) = (b \circ \mathcal{V}(\varphi_1)) \cup (\neg b \circ \mathcal{V}(\varphi_2))$
5. $\mathcal{V}((b \wedge \varphi_1)\ \mathsf{W}\ (\neg b \wedge \varphi_2)) = b^* \cdot ((b \circ \mathcal{V}(\varphi_1)) \cup (\neg b \circ \mathcal{V}(\varphi_2)))$
6. $\mathcal{V}(r \mapsto \varphi) = r \circ \mathcal{V}(\varphi)$

That is, a boolean expression is violated iff its negation holds. The conjunction of two formulas is violated iff either one of them is violated. The formula $\mathsf{X}\varphi$ is violated iff φ is violated at the next letter. The formula $(b \wedge \varphi_1) \vee (\neg b \wedge \varphi_2)$ is violated iff either b holds and φ_1 is violated or b does not hold and φ_2 is violated. The formula $(b \wedge \varphi_1)\ \mathsf{W}\ (\neg b \wedge \varphi_2)$ is violated iff after a finite number of letters satisfying b either b holds and φ_1 is violated or b does not hold and φ_2 is violated. The formula $r \mapsto \varphi$ is violated iff φ is violated after a prefix tightly satisfying r.

The following theorem, which is proven in the appendix, states that $[\![\mathcal{V}(\varphi)!]\!]$ recognizes violation of the formula φ. That is, that a word v has a prefix u tightly satisfying $\mathcal{V}(\varphi)$ if and only if $v \models^{+} \neg\varphi$.

Theorem 11 *Let φ be an* RLTL$^{\mathsf{LV}}$ *formula over $\mathbb{P}$ and let v be a word over $\Sigma_{\mathbb{P}}$. Then, $|\mathcal{V}(\varphi)| = O(|\varphi|)$ and $[\![\mathcal{V}(\varphi)!]\!]$ recognizes violation of φ.*

Thus, as stated by the following corollary, for any formula φ of RLTL$^{\mathsf{LV}}$ one can construct a monitor recognizing the violation of φ, with size linear in φ.

Corollary 12 *Let φ be an* RLTL$^{\mathsf{LV}}$ *formula. Then there exists an* NFA *of linear size recognizing the violation of φ.*

Proof Sketch. Direct corollary of Theorem 11 and Fact 4. □

In Section 5 we give a direct construction for such an automaton.

4 Discussion

As mentioned in the introduction, we have derived the definition of RLTL$^{\mathsf{LV}}$ by combining several results in the literature. In this section we discuss the relation of RLTL$^{\mathsf{LV}}$ to other logics in the literature associated with efficient verification, thus clarifying the nature of this subset.

4.1 Relation to classification of safety formulas

In [13] Kupferman and Vardi classify safety formulas according to whether all, some, or none of their bad prefixes are informative, and name them *intentionally* safe, *accidentally* safe and *pathologically* safe, respectively. A finite word v is a *bad prefix* for φ iff for any extension w of v, $w \not\models \varphi$. A finite word v is an *informative bad prefix* for φ iff $v \models^{+} \neg\varphi$.[2] It follows that the violating regular expression $\mathcal{V}(\varphi)$ characterizes exactly the set of informative bad prefixes of φ.

Theorem 13 *Let φ be an* RLTL$^{\mathsf{LV}}$ *formula. Then $[\![\mathcal{V}(\varphi)!]\!]$ defines the set of informative bad prefixes for φ.*

[2] The concept of an informative prefix was defined syntactically in [13]. We have used the semantic equivalent definition provided by [7].

Proof Sketch. Follows immediately from Theorem 11 and Definition 8. □

The subset RLTL^{LV} is syntactically weak by definition (since all formulas are in positive normal form and use only weak operators). Thus, by [13] all RLTL^{LV} formulas are non-pathologically safe (i.e. they are either intentionally safe or accidentally safe). Therefore, every computation that violates an RLTL^{LV} formula has at least one informative bad prefix.

Theorem 14 *All formulas of* RLTL^{LV} *are non-pathologically safe.*

Proof Sketch. The formulas of RLTL^{LV} are syntactically weak by definition. They are non-pathologically safe by [13, Theorem 5.3]. □

[13] characterize an automaton as *fine* for φ iff for every word violating φ it recognizes at least one informative bad prefix for φ. They show that violation of non-pathologically safe formulas can be detected by a fine alternating automaton on infinite words of linear size or by a fine non-deterministic automaton on finite words of *exponential* size. Corollary 12 gives us that for RLTL^{LV} violation can be detected by a fine *linear*-sized finite automaton on finite words.

4.2 Expressibility and relation to the common fragment of LTL and ACTL

In [14] Maidl studied the subset of ACTL formulas which have an equivalent in LTL. She defined the fragments $\text{ACTL}^{\text{DET}}$and $\text{LTL}^{\text{DET}}$of ACTL and LTL, respectively, and proved that any formula of LTL that has an equivalent in ACTL is expressible in LTL^{DET}. And vice versa, any formula of ACTL that has an equivalent in LTL is expressible in ACTL^{DET}.

Below we repeat the definition of LTL^{DET}, and give its restriction to safety formulas which we denote LTL^{LV}.[3]

Definition 15 (LTL^{DET},LTL^{LV}) *Let b be a boolean expression.*

- *The set of* LTL^{DET} *formulas is recursively defined as follows:*

  $\varphi ::= b \mid \varphi \wedge \varphi \mid (b \wedge \varphi) \vee (\neg b \wedge \varphi) \mid \mathsf{X}\,\varphi \mid (b \wedge \varphi)\,\mathsf{W}\,(\neg b \wedge \varphi) \mid (b \wedge \varphi)\,\mathsf{U}\,(\neg b \wedge \varphi)$

- *The set of* LTL^{LV} *formulas is recursively defined as follows:*

  $\varphi ::= b \mid \varphi \wedge \varphi \mid (b \wedge \varphi) \vee (\neg b \wedge \varphi) \mid \mathsf{X}\,\varphi \mid (b \wedge \varphi)\,\mathsf{W}\,(\neg b \wedge \varphi)$

Theorem 16 LTL^{LV} *is a strict subset of* RLTL^{LV}.

Proof Sketch. Follows from Definitions 9 and 15. □

Maidl additionally showed that the negation of a formula φ in LTL^{DET} is recognizable by a 1-weak Büchi automaton with size linear in φ. Thus intuitively, the safety

[3] The set ACTL^{DET} and its restriction to safety formulas, which we denote ACTL^{LV}, are obtained by replacing the operators X with AX, W with AW, and U with AU in the definitions of LTL^{DET} and LTL^{LV}, respectively.

formulas of $\text{LTL}^{\mathsf{DET}}$ should have an NFA of linear size. Theorem 16 together with Corollary 12 state that indeed the restriction of $\text{LTL}^{\mathsf{DET}}$ to safety lies in the subset of formulas whose violation can be detected by an NFA of linear size.

Note that LTL^{LV} is not only syntactically weaker than $\text{RLTL}^{\mathsf{LV}}$ it is also semantically weaker than $\text{RLTL}^{\mathsf{LV}}$ (i.e. it has less expressive power). This can be seen by taking the same example as Wolper's in showing that LTL is as expressive as star free omega regular languages rather than entire omega regular languages [19]. That is, by showing that the property "b holds at every even position (yet b may hold on some odd positions as well)" is not expressible in LTL^{LV} (since it is not expressible in LTL) while it is expressible in $\text{RLTL}^{\mathsf{LV}}$ by the formula: $\mathit{true} \cdot (\mathit{true} \cdot \mathit{true})^* \mapsto b$.

4.3 Relation to on-the-fly verification of RCTL

In [1] Beer et al. defined the logic RCTL which extends CTL with regular expressions via a suffix implication operator. In addition they defined a subset of this logic which can be verified by on-the-fly model checking (we term this subset $\text{RCTL}^{\mathsf{OTF}}$, where OTF stands for "on-the-fly"). A formula can be verified by on-the-fly model checking iff it can be verified by model checking an invariant property over a parallel composition of the design model with a finite automaton. Their definitions can be rephrased as follows.

Definition 17 (RCTL**,**$\text{RCTL}^{\mathsf{OTF}}$**)** *Let b be a boolean expression, and r a regular expression.*

– *The set of* RCTL *formulas is recursively defined as follows:*

$$\varphi ::= b \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathsf{EX}\,\varphi \mid \mathsf{EG}\,\varphi \mid \varphi\,\mathsf{EU}\,\varphi \mid r \mapsto \varphi$$

Additional operators are defined as syntactic sugaring of the above operators:

- $\varphi \vee \psi \stackrel{\text{def}}{=} \neg(\neg\varphi \wedge \neg\psi)$
- $\mathsf{AX}\,\varphi \stackrel{\text{def}}{=} \neg(\mathsf{EX}\,\neg\varphi)$
- $\varphi\,\mathsf{EW}\,\psi \stackrel{\text{def}}{=} \mathsf{EG}\,\varphi \vee (\varphi\,\mathsf{EU}\,\psi)$
- $\varphi \rightarrow \psi \stackrel{\text{def}}{=} \neg\varphi \vee \psi$
- $\mathsf{AF}\,\varphi \stackrel{\text{def}}{=} \neg(\mathsf{EG}\,\neg\varphi)$
- $\varphi\,\mathsf{AW}\,\psi \stackrel{\text{def}}{=} \neg(\neg\psi\,\mathsf{EU}\,\neg(\varphi \vee \psi))$
- $\mathsf{EF}\,\varphi \stackrel{\text{def}}{=} \mathit{true}\,\mathsf{EU}\,\varphi$
- $\mathsf{AG}\,\varphi \stackrel{\text{def}}{=} \neg(\mathsf{EF}\,\neg\varphi)$
- $\varphi\,\mathsf{AU}\,\psi \stackrel{\text{def}}{=} \mathsf{AF}\,\psi \wedge (\varphi\,\mathsf{AW}\,\psi)$

– *The set of* $\text{RCTL}^{\mathsf{OTF}}$ *formulas is recursively defined as follows:*

$$\varphi ::= b \mid \varphi \wedge \varphi \mid b \rightarrow \varphi \mid \mathsf{AX}\varphi \mid \mathsf{AG}\varphi \mid r \mapsto \varphi$$

It can easily be seen that the subset obtained from $\text{RCTL}^{\mathsf{OTF}}$ by omitting the path quantifiers (replacing the operators AX with X and AG with G) is a strict subset of $\text{RLTL}^{\mathsf{LV}}$. And as stated by the following theorem, the subset of RCTL formulas that can be verified on-the-fly can be extended to include all formulas in the subset obtained from $\text{RLTL}^{\mathsf{LV}}$ by adding the universal path quantifier.

Theorem 18 *Let* $\text{RCTL}^{\mathsf{LV}}$ *be the subset of* RCTL *formulas obtained by adding the universal path quantifier to the operators X and W in the definition of* $\text{RLTL}^{\mathsf{LV}}$*. Then*

1. $\text{RCTL}^{\mathsf{OTF}}$ *is a strict subset of* $\text{RCTL}^{\mathsf{LV}}$*, and*

2. *for every formula φ in* RCTL$^{\mathsf{LV}}$ *there exists an* NFA *of linear size recognizing the informative bad prefixes of φ.*

Proof Sketch. The first item is correct by definition. The second item follows by applying the construction of the violating regular expression (Definition 10) to the formula obtained by removing the path quantifiers, using Maidl's result (that an ACTL formula has an equivalent in LTL iff it has an equivalent in ACTL$^{\mathsf{DET}}$ [14, Theorem 2]) together with Clarke and Draghicescu's result [4] (that a CTL formula φ has an equivalent in LTL iff it is equivalent to the formula φ' obtained by removing the path quantifiers from φ) and Corollary 12. □

4.4 Relation to PSL

The language reference manual (LRM) of PSL [10] defines *the simple subset*, which intuitively conforms to the notion of monotonic advancement of time, left to right through the property, as in a timing diagram. For example, the property $\mathsf{G}(a \rightarrow \mathsf{X}b)$ is in the simple subset whereas the property $\mathsf{G}((a \wedge \mathsf{X}b) \rightarrow c)$ is not. This characteristic should in turn ensure that properties within the subset can be verified easily by simulation tools, though no construction or proof for this is given. The exact definition is given in terms of restriction on operators and their operands as can be depicted in [10, subsection 4.4.4, p. 29].

As explained in the introduction since we are interested in model checking as well, we focus on the safety formulas of the simple subset. There are two differences between the definition of the safety simple subset in the LRM and the definition of RLTL$^{\mathsf{LV}}$.

1. The definition in the LRM considers weak regular expressions while RLTL does not. A construction for the safety simple subset of the LRM, including weak regular expressions, is available in [17]. The reason we exclude weak regular expressions is that their verification involves determinism which results in an automaton exponential rather than linear in the size of the formula.
2. The restrictions in the LRM are a bit more restrictive than those in RLTL$^{\mathsf{LV}}$. To be exact, for the operators $\vee$ and W the simple subset allows only one operand to be non-boolean, whereas RLTL$^{\mathsf{LV}}$ allows both to be non-boolean, conditioned they can be conjuncted with some Boolean and its negation. It is quite clear that the motivation for over restricting these operators in the definition of the simple subset in the LRM is to simplify the restricting rule.

5 Direct Construction

In the following we give a direct construction for an NFA recognizing the violation of a formula. This automaton recognize a bad informative prefix for any computation violating the given formula: if a final location is reached on some run, then the observed prefix is a bad informative prefix. The advantage of this construction over the previous one is in it being a direct one, thus enabling various optimizations.

Theorem 19 *Let φ be a* RLTL$^{\mathsf{LV}}$ *formula over $\mathbb{P}$ and let w be a word over $\Sigma_{\mathbb{P}}$. Then there exists an* NFA *$\mathcal{N}_\varphi$ linear in the size of φ recognizing violation of φ.*

In the following subsection we provide the construction, we give the proof of its correctness in the appendix.

As $\mathcal{N}_\varphi$ recognizes bad prefixes we can reduce the verification of φ to verification of the invariant property $\mathsf{AG}\ (\neg$"The automaton $\mathcal{N}_\varphi$ is in a final state"$)$ on a parallel composition of the design model with $\mathcal{N}_\varphi$. In order to state this formally one needs to define a symbolic transition system which can model both a given design and an NFA, and define the result of a parallel composition of two such systems. An appropriate mathematical model for this is a discrete transition system DTS (see e.g. [3]), inspired from the fair transition system of [12]. The following corollary states the result formally using the notion of a DTS.

Corollary 20 *Let φ be an* RLTL$^{\mathsf{LV}}$ *formula, then there exists an* NFA *$\mathcal{N}_\varphi = (P, S, S_0, \delta, F)$ linear in the size of φ such that for any model M*

$$\mathcal{D}(M) \models \varphi \quad \textit{iff} \quad \mathcal{D}(M) \,|||\, \mathcal{D}(\mathcal{N}_\varphi) \models \mathsf{AG}(\neg \mathsf{at}(F))$$

where $\mathcal{D}(M)$ and $\mathcal{D}(\mathcal{N}_\varphi)$ are the DTS*s for M and $\mathcal{N}_\varphi$ respectively, and $\mathsf{at}(F)$ is a boolean expression stating that $\mathcal{N}_\varphi$ is in a final location.*

Note that if we define a co-UFA $\mathcal{C}_\varphi$ with the same components of N_φ (i.e. $\mathcal{C}_\varphi = \langle P, S, S_0, \delta, F\rangle$) then $\mathcal{C}_\varphi$ accepts a word w iff $w \models \varphi$.

Constructing an NFA for a given RLTL$^{\mathsf{LV}}$ formula

Let r be an RE such that $\epsilon \notin \mathcal{L}(r)$, b a boolean expression, $\varphi, \varphi_1, \varphi_2$ RLTL$^{\mathsf{LV}}$ formulas. For the induction construction, let $\mathcal{N}_\varphi = \langle P, Q, Q_0, \delta, F\rangle$, $\mathcal{N}_{\varphi_1} = \langle P^1, Q^1, Q_0^1, \delta^1, F^1\rangle$ and $\mathcal{N}_{\varphi_2} = \langle P^2, Q^2, Q_0^2, \delta^2, F^2\rangle$.

1. Case b.

   $$\mathcal{N}_b = \langle P, \{q_0, q_1, q_2\}, \{q_1\}, \delta, \{q_0\}\rangle$$

   where P is the set of state variables in b and

   $$\delta = \{(q_1, b, q_2), (q_1, \neg b, q_0), (q_0, \mathit{true}, q_0), (q_2, \mathit{true}, q_2)\}.$$

2. Case $\varphi_1 \wedge \varphi_2$.

   $$\mathcal{N}_{\varphi_1 \wedge \varphi_2} = \langle P^1 \cup P^2, Q^1 \cup Q^2, Q_0^1 \cup Q_0^2, \delta^1 \cup \delta^2, F^1 \cup F^2\rangle$$

   A run of $\mathcal{N}_{\varphi_1 \wedge \varphi_2}$ has a non-deterministic choice between a run of $\mathcal{N}_{\varphi_1}$ and a run of $\mathcal{N}_{\varphi_2}$. In any choice it should not reach a state in either F_1 or F_2. The resulting NFA is described in Figure 1.

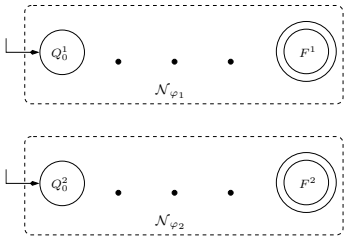

Fig 1. An NFA for $\varphi_1 \wedge \varphi_2$

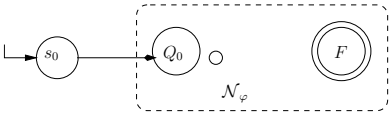

Fig 2. An NFA for $\mathsf{X}\varphi$

3. Case $\mathsf{X}\varphi$.

$$\mathcal{N}_{\mathsf{X}\varphi} = \langle P, Q \cup \{s_0\}, \{s_0\}, \delta^{'}, F\rangle$$

where s_0 is a new state and $\delta' = \delta \cup \bigcup_{q\in Q_0}(s_0, \mathit{true}, q)$. The resulting NFA is described in Figure 2.

4. Case $(b \wedge \varphi_1) \vee (\neg b \wedge \varphi_2)$.

$$\mathcal{N}_{(b\wedge\varphi_1)\vee(\neg b\wedge\varphi_2)} = \langle P^1 \cup P^2, \{s_0\} \cup Q^1 \cup Q^2, \{s_0\}, \delta', F^1 \cup F^2\rangle$$

where

$$\begin{aligned}\delta' = \delta^1 \cup \delta^2 \cup \\ \bigcup_{q_1\in Q_0^1}\bigcup_{(q_1,c,q_2)\in\delta^1}(s_0, b\wedge c, q_2) \\ \bigcup_{q_1\in Q_0^2}\bigcup_{(q_1,c,q_2)\in\delta^2}(s_0, \neg b\wedge c, q_2)\end{aligned}$$

A run of $\mathcal{N}_{(b\wedge\varphi_1)\vee(\neg b\wedge\varphi_2)}$ starts in a new state s_0, if b holds it continues on $\mathcal{N}_{\varphi_1}$ otherwise it continues on $\mathcal{N}_{\varphi_2}$. The resulting NFA is described in Figure 3.

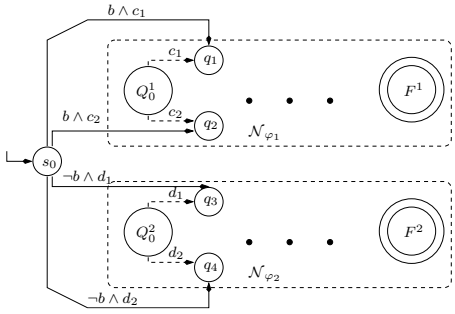

Fig 3. An NFA for $(b \wedge \varphi_1) \vee (\neg b \wedge \varphi_2)$

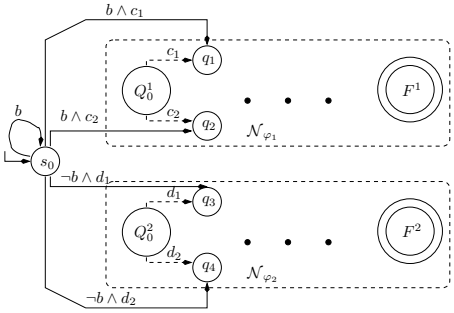

Fig 4. An NFA for $(b \wedge \varphi_1)\ \mathsf{W}\ (\neg b \wedge \varphi_2)$

5. Case $(b \wedge \varphi_1)\ \mathsf{W}\ (\neg b \wedge\ \varphi_2)$.

$$\mathcal{N}_{(b\wedge\varphi_1)\mathsf{W}(\neg b\wedge\ \varphi_2)} = \langle P^1 \cup P^2, \{s_0\} \cup Q^1 \cup Q^2, \{s_0\}, \delta', F^1 \cup F^2\rangle$$

where

$$\begin{aligned}\delta' = (s_0, b, s_0) \cup \delta^1 \cup \delta^2 \cup \\ \bigcup_{q_1\in Q_0^1}\bigcup_{(q_1,c,q_2)\in\delta^1}(s_0, b\wedge c, q_2) \\ \bigcup_{q_1\in Q_0^2}\bigcup_{(q_1,c,q_2)\in\delta^2}(s_0, \neg b\wedge c, q_2)\end{aligned}$$

The resulting NFA is described in Figure 4.

6. Case $r \mapsto \varphi_2$
   Let $N = \langle \mathbb{B}_P, Q, Q_0, \delta, F \rangle$ be a non-deterministic automata on finite words accepting $\mathcal{L}(r)$ (as in [2] for example). Convert it to an NFA $\mathcal{N}_1$ recognizing violation of $[\![r \mapsto \mathit{false}]\!]$ by sending every "missing edge" to a new trapping state and making the set of final states of N the set of bad states of $\mathcal{N}_1$. The resulting NFA, $\mathcal{N}_1 = \langle P^1, Q^1, Q_0^1, \delta^1, F^1 \rangle$ is defined as follows.

$$\mathcal{N}_1 = \langle P, Q \cup \{q_{sink}\}, Q_0, \delta^1, F \rangle$$

where P is the set of atomic propositions in r and

$$\delta^1 = \bigcup_{(q_1,c,q_2)\in\delta} \{(q_1, c, q_2).(q_1, \neg c, q_{sink})\}.$$

The resulting NFA (recognizing violation of $[\![r \mapsto \mathit{false}]\!]$) is described in Figure 5.

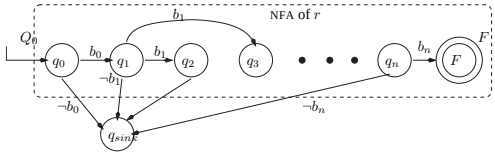

Fig 5. An NFA for $r \mapsto \mathit{false}$

Let $\mathcal{N}_2 = \mathcal{N}_{\varphi_2} = \langle P^2, Q^2, Q_0^2, \delta^2, F^2 \rangle$ be the NFA, as constructed by induction for φ_2. Then $\mathcal{N}_{r \mapsto \varphi_2}$ is constructed by concatenation of $\mathcal{N}_1$ and $\mathcal{N}_2$ as follows:

$$\mathcal{N}_{r \mapsto \varphi_2} = \langle P^1 \cup P^2, Q^1 \cup Q^2 \cup \{q_{bad}\}, Q_0^1, \delta', F^2 \cup \{q_{bad}\} \rangle$$

where q_{bad} is a new state, and

$$\delta' = \begin{array}{l} \delta^1 \cup \delta^2 \cup \\ \{(q_1, c_1 \wedge c_2, q_4) \mid \exists q_2 \in F^1, q_3 \in Q_0^2 \text{ s.t. } (q_1, c_1, q_2) \in \delta^1,\ (q_3, c_2, q_4) \in \delta^2\} \cup \\ \{(q_1, c_1 \wedge \bigwedge_{\{c' \mid \exists q_3 \in Q_0^2, q_4 \in Q^2 \text{ s.t. } (q_3, c', q_4) \in \delta^2\}} \neg c', q_{bad}) \mid \exists q_2 \in F^1 \text{ s.t. } (q_1, c_1, q_2) \in \delta^1\} \end{array}$$

The resulting NFA is described in Figure 6.

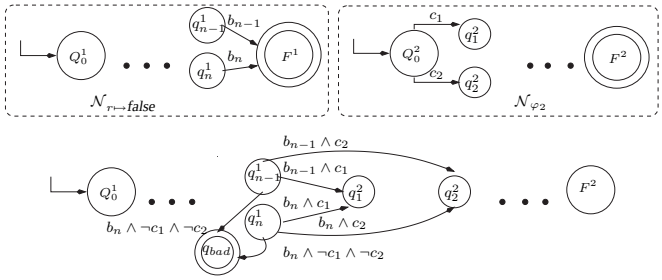

Fig 6. An NFA for $r \mapsto \varphi_2$

6 Conclusions

We have defined a subset of safety RLTL formulas. This subset, while consisting of most formulas run in hardware verification, enjoys efficient verification algorithms. Verification of general RLTL formulas requires an exponential Büchi automaton. Verification of RLTL^{LV} formulas can be reduced to invariance checking using an auxiliary automaton on finite words (NFA). Moreover, the size of the generated NFA is linear in the size of the formula.

We have presented two procedures for the construction of a linear-sized automaton for an RLTL^{LV} formula. The first goes through building an equivalent linear-sized regular expression (RE), and the other directly constructs the automaton from the given formula. Both methods provide algorithms that can be easily implemented by tool developers. We note that for traditional regular expressions, the existence of an NFA of linear size does not imply the existence of a regular expression of linear size, as REs are exponentially less succinct than NFAs [5]. Since our translation to regular expression involves the *fusion* operator, it would be interesting to find whether the result of [5] holds for this type of regular expressions as well.

The PSL language reference manual (LRM) [10], defines a subset of the language, called *the simple subset*, which intuitively should consists of the formulas which are easy to verify by state of the art verification methods. However, there is no justification for choosing the defined subset, and in particular no proof that it meets the intuition it should. The restrictions made on RLTL in the definition of the simple subset in the LRM reflect the restrictions in RLTL^{LV}. Having proved that RLTL^{LV} formulas are easy to verify by both model checking and simulation, we view this paper as providing the missing proof.

Acknowledgments

We would like to thank Cindy Eisner and Orna Lichtenstein for their helpful comments on an early draft of the paper.

References

1. I. Beer, S. Ben-David, and A. Landver. On-the-fly model checking of RCTL formulas. In *Proc. 10^{th} International Conference on Computer Aided Verification (CAV'98)*, LNCS 1427, pages 184–194. Springer-Verlag, 1998.
2. S. Ben-David, D. Fisman, and S. Ruah. Automata construction for regular expressions in model checking, June 2004. IBM research report H-0229.
3. S. Ben-David, D. Fisman, and S. Ruah. Embedding finite automata within regular expressions. In 1st International Symposium on Leveraging Applications of Formal Methods. Springer-Verlag, November 2004.
4. E.M. Clarke and I.A. Draghicescu. Expressibility results for linear-time and branching-time logics. In *Proc. Workshop on Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency*, LNCS 354, pages 428–437. Springer-Verlag, 1988.
5. Andrzej Ehrenfeucht and Paul Zeiger. Complexity measures for regular expressions. In *STOC '74: Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 75–79, New York, NY, USA, 1974. ACM Press.

6. C. Eisner, D. Fisman, and J. Havlicek. A topological characterization of weakness. In *PODC '05: Proceedings of the twenty-fourth annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 1–8, New York, NY, USA, 2005. ACM Press.
7. C. Eisner, D. Fisman, J. Havlicek, Y. Lustig, A. McIsaac, and D. Van Campenhout. Reasoning with temporal logic on truncated paths. In *The 15th International Conference on Computer Aided Verification (CAV)*, LNCS 2725, pages 27–40. Springer-Verlag, July 2003.
8. M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. In *J. Comput. Syst. Sci.*, pages 18(2), 194–211, 1979.
9. J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.
10. IEEE. IEEE standard for property specification language (PSL), October 2005. To appear.
11. Christian Josef Kargl. A Sugar translator. Master's thesis, Institut für Softwaretechnologie, Technische Univesität Graz, Graz, Austria, December 2003.
12. Y. Kesten, A. Pnueli, and L. Raviv. Algorithmic verification of linear temporal logic specifications. volume 1443, pages 1–16, 1998.
13. Orna Kupferman and Moshe Y. Vardi. Model checking of safety properties. In *Proc.* 11^{th} *International Conference on Computer Aided Verification (CAV)*, LNCS 1633, pages 172–183. Springer-Verlag, 1999.
14. Monika Maidl. The common fragment of CTL and LTL. In *IEEE Symposium on Foundations of Computer Science*, pages 643–652, 2000.
15. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, New York, 1992.
16. A. Pnueli. A temporal logic of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.
17. S. Ruah, D. Fisman, and S. Ben-David. Automata construction for on-the-fly model checking PSL safety simple subset, June 2005. Research Report H-0234.
18. Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 15 1994.
19. P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1/2):72–99, 1983.