On Regular Temporal Logics with Past*, **

Christian Dax[1], Felix Klaedtke[1], and Martin Lange[2]

[1] ETH Zurich, Switzerland
[2] Ludwig-Maximilians-University Munich, Germany

Abstract. The IEEE standardized *Property Specification Language*, PSL for short, extends the well-known linear-time temporal logic LTL with so-called semi-extended regular expressions. PSL and the closely related *SystemVerilog Assertions*, SVA for short, are increasingly used in many phases of the hardware design cycle, from specification to verification. In this paper, we extend the common core of these specification languages with past operators. We name this extension RTL. Although all ω-regular properties are expressible in PSL, SVA, and RTL, past operators often allow one to specify properties more naturally and concisely. In fact, we show that RTL is exponentially more succinct than the cores of PSL and SVA. Furthermore, we present a translation of RTL into language-equivalent nondeterministic Büchi automata, which is based on novel constructions for 2-way alternating automata. Our translation has almost the same worst-case complexity in terms of the size of the resulting nondeterministic Büchi automata as the existing translations for PSL and SVA. Consequently, the satisfiability and the model-checking problem for RTL fall into the same complexity classes as the corresponding problems for PSL and SVA. From the translation it also follows that the blowup of translating RTL formulas into initially equivalent PSL/SVA formulas is at most triply exponential.

1 Introduction

The industry standardized temporal logics PSL [1] and SVA (the assertion language of SystemVerilog [2]) are increasingly used in the hardware industry to formally express, validate, and verify the requirements of circuit designs. The linear-time core of PSL extends the well-known linear-time temporal logic LTL with semi-extended regular expressions (SEREs), which are essentially regular expressions with an additional operator for expressing the intersection of languages. The core of SVA can be seen as a subset of PSL.[1] The prominence of PSL and SVA in industry over other specification languages like LTL [23],

* Partly supported by the Swiss National Science Foundation (SNF).

** Due to space limitations, most proofs have been omitted. These can be found in an extended version of the paper, which is available from the authors' webpages.

[1] For the ease of exposition, we identify, similar to [5, 9, 7, 24], PSL and SVA with their respective cores. In particular, the cores are "unclocked," they do not contain local variables (which are not part of the PSL standard), and their semantics is only defined over infinite words.

S. Albers et al. (Eds.): ICALP 2009, Part II, LNCS 5556, pp. 175–187, 2009.

μLTL [4], and ETL [28] is that PSL and SVA balance well the competing needs of a specification language like *expressiveness*, *usability*, and *implementability* [3]: all ω-regular languages are expressible in PSL/SVA, specifications in PSL/SVA are fairly easy to read and write, and relevant verification problems (e.g. model checking) for PSL/SVA are automatically solvable in practice.

Although temporal operators that refer to the past have been found natural and useful when expressing temporal properties [20, 16, 21, 10, 9], the PSL and SVA standards support temporal past operators only in a restrictive way. This design choice has already been made for the predecessor ForSpec [3] of PSL/SVA and has been justified by the argument that handling "arbitrary mixing of past and future operators results in nonnegligible implementation cost" [3]. One reason for this belief is that in the automata-theoretic approach to model checking [27], one uses 2-way automata to deal with past and future operators rather than 1-way automata when only future operators are present. The nowadays used automata constructions for 2-way automata are more involved than the corresponding ones for 1-way automata. For instance, with the state-of-the-art construction in [16], we can translate a 2-way alternating Büchi automaton with n states into a language-equivalent nondeterministic Büchi automaton (NBA) with $2^{\mathcal{O}(n^2)}$ states. For a given 1-way alternating Büchi automaton, we obtain with the Miyano-Hayashi construction [22] an NBA with only $2^{\mathcal{O}(n)}$ states. Nevertheless, in this paper, we give arguments in favor of extending PSL and SVA with past operators and we argue against this assumed additional implementation cost. In particular, one of our results shows that a restricted class of 2-way automata suffices and the additional cost for this class is small.

In more detail, the content of the paper is as follows. We first propose an extension of PSL with past operators, which we name *Regular Temporal Logic*, RTL for short. RTL extends PSL by the standard past operators from linear-time temporal logic and by the corresponding past operators of the PSL/SVA-specific operators for SEREs. For example, the PSL/SVA-specific operator $\alpha \diamond\!\!\rightarrow \varphi$ describes that a system trace fulfills from the current time point the pattern given by the SERE α and at the end the *post-condition* φ holds, where φ is a PSL/SVA formula. RTL additionally contains the corresponding counterpart $\alpha \mathbin{\diamond\!\!\!\!\leftarrow} \varphi$. This describes that the *pre-condition* φ holds at some time point in the past and at that time point the system trace fulfills up to the current time point the pattern α. Note that the temporal operator $\alpha \diamond\!\!\rightarrow \varphi$ is closely related to the modality $\langle\alpha\rangle\varphi$ in dynamic logic [15]. However, PSL/SVA uses SEREs over state predicates and in dynamic logic, the expressions are over program statements.

PSL, SVA, and RTL have the same expressive power: they all describe the class of ω-regular languages. However, RTL allows one to describe ω-regular languages more concisely than PSL and SVA. To show this, we establish a lower bound on the succinctness of RTL and SVA. We define a family of ω-regular languages and prove that these languages can be described in RTL exponentially more succinctly than in SVA. For the LTL-expressible properties, i.e, the ω-regular languages that are star-free, we obtain as a byproduct that RTL is double

exponentially more succinct than LTL, even when extended with the classical temporal past operators Y (yesterday) and S (since).

Furthermore, we investigate the additional computational cost for solving the satisfiability problem and the model-checking problem for RTL. As for PSL and SVA, these problems are EXPSPACE-complete for RTL. In practice, the satisfiability problem and the model-checking problem for PSL and SVA are solved by using an automata-theoretic approach [5, 9, 7], translating a given formula into an NBA. With the standard automata constructions for PSL and SVA, one obtains for a PSL/SVA formula of size n an NBA of size $\mathcal{O}(2^{2\cdot 2^{2n}})$ [5,7]. We present a novel construction for RTL that translates an RTL formula of size n into an NBA of size $\mathcal{O}(2^{3\cdot 2^{2n}})$. Note that the upper bounds of the sizes of the resulting automata for PSL/SVA and RTL only differ by a small constant in the exponent despite the richer structure of RTL. Our translation is based on alternation-elimination constructions for restricted classes of 2-way alternating automata that were recently presented in [12] and which we further improve in this paper for the alternating automata that we obtain from our translation of RTL formulas into alternating automata. This construction can also be used to translate a given RTL formula into an initially equivalent SVA formula whose size is triple exponentially larger, not quite matching the lower bound mentioned above. One of these three exponentials is due to the fact that the resulting SVA formulas do not contain SEREs anymore, but only regular expressions.

We point out that our translation for RTL into NBAs significantly improves over translations that we obtain when utilizing automata constructions that do not take the given special class of alternating automata into account. For instance, when using the state-of-the-art construction [16] for translating 2-way alternating automata into NBAs, one obtains an NBA of size $\mathcal{O}(2^{4\cdot 2^{4n}+2^{2n}})$, where n is again the size of the given RTL formula. Overall, the presented translation indicates that extensions of temporal logics with past operators can be handled with only a minor overhead in the automata-theoretic model-checking approach when adequate constructions for 2-way alternating automata are used.

2 Preliminaries

Words and Trees. We denote the set of finite words over the alphabet Σ by Σ^* and the set of infinite words over Σ by Σ^ω. The length of a word $w \in \Sigma^*$ is written as $|w|$ and ε denotes the empty word. For a finite or infinite word w, w_i denotes the symbol of w at position $i \in \mathbb{N}$, where we assume that $i < |w|$ if w is finite. We write $v \preceq w$ if v is a prefix of the word w. For $i, j < |w|$, we write $w_{i..}$ for the suffix $w_i w_{i+1} \dots$ and $w_{i..j}$ for the subword $w_i w_{i+1} \dots w_j$.

A (Σ-labeled) *tree* is a function $t : T \to \Sigma$, where $T \subseteq \mathbb{N}^*$ satisfies the conditions: (i) T is prefix-closed (i.e., if $v \in T$ and $u \preceq v$ then $u \in T$) and (ii) if $vi \in T$ and $i > 0$ then $v(i-1) \in T$. The elements in T are called the *nodes* of t and the empty word ε is called the *root* of t. A node $vi \in T$ with $i \in \mathbb{N}$ is called a *child* of the node $v \in T$. An (infinite) *path* in t is a word $\pi \in \mathbb{N}^\omega$ such that $v \in T$, for every prefix v of π. We write $t(\pi)$ for the word $t(\pi_0)t(\pi_1) \dots \in \Sigma^\omega$.

Propositional Logic. We denote the set of *Boolean formulas* over the set P of propositions by $\mathcal{B}(P)$, i.e., $\mathcal{B}(P)$ consists of the formulas that are inductively built from the propositions in P and the connectives $\vee$, $\wedge$, and $\neg$. For $M \subseteq P$ and $b \in \mathcal{B}(P)$, we write $M \models b$ iff b evaluates to true when assigning true to the propositions in M and false to the propositions in $P \setminus M$. We write $\mathcal{B}^+(P)$ for the set of Boolean formulas in which the connective $\neg$ does not occur.

Regular Expressions. The syntax of *semi-extended regular expressions* (SEREs) over the proposition set P is defined by the grammar $\alpha ::= \varepsilon \mid b \mid \alpha \star \alpha \mid \alpha^*$, where $b \in \mathcal{B}(P)$ and $\star \in \{\cup, \cap, ;, :\}$. The language of an SERE over the proposition set P is inductively defined: (i) $L(\varepsilon) := \{\varepsilon\}$, (ii) $L(b) := \{w \in (2^P)^* \mid |w| = 1 \text{ and } w \models \alpha\}$, for $b \in \mathcal{B}(P)$, (iii) $L(\beta \star \gamma) := L(\beta) \star L(\gamma)$, for $\star \in \{\cup, \cap, ;, :\}$, where $L ; L' := \{uv \mid u \in L \text{ and } v \in L'\}$ is the concatenation of L and L', and $L : L' := \{ubv \mid ub \in L \text{ and } bv \in L' \text{ with } b \in 2^P\}$ the fusion, and (iv) $L(\beta^*) := \bigcup_{n \in \mathbb{N}} L^n(\beta)$, where $L^0 := \{\varepsilon\}$ and $L^{i+1} := L ; L^i$, for all $i \in \mathbb{N}$. The *size* of an SERE is its syntactic length, i.e., $\|\varepsilon\| := 1$, $\|b\| := 1$, for $b \in \mathcal{B}(P)$, $\|\beta \star \gamma\| := 1 + \|\beta\| + \|\gamma\|$, for $\star \in \{\cup, \cap, ;, :\}$, and $\|\beta^*\| := 1 + \|\beta\|$.

Automata. In the following, we define 2-way alternating automata, which scan input words letter by letter with their read-only head. Let $\mathbb{D} := \{-1, 0, 1\}$ be the set of directions in which the read-only head can move. A *2-way alternating Büchi automaton* (2ABA) $\mathcal{A}$ is a tuple $(Q, \Sigma, \delta, q_I, F)$, where Q is a finite set of states, Σ is a finite nonempty alphabet, $\delta : Q \times \Sigma \to \mathcal{B}^+(Q \times \mathbb{D})$ is the transition function, $q_I \in Q$ is the initial state, and $F \subseteq Q$ is the acceptance condition. The *size* $\|\mathcal{A}\|$ of the automaton $\mathcal{A}$ is $|Q|$.

A *configuration* of $\mathcal{A}$ is a pair $(q, i) \in Q \times \mathbb{N}$. Intuitively, q is the current state and the read-only head is at position i of the input word. A *run* of $\mathcal{A}$ on $w \in \Sigma^\omega$ is a tree $r : T \to Q \times \mathbb{N}$ such that $r(\varepsilon) = (q_I, 0)$ and

$$\big\{(q', j' - j) \in Q \times D \mid r(y) = (q', j'), \text{ where } y \text{ is a child of } x \text{ in } r\big\} \models \delta(q, w_j)\,,$$

for each node $x \in T$ with $r(x) = (q, j)$. For $\pi := (q_0, i_0)(q_1, i_1) \ldots \in (Q \times \mathbb{N})^\omega$, we define $\mathit{Inf}(\pi) := \{q \mid q \text{ occurs infinitely often in } q_0 q_1 \ldots \in Q^\omega\}$. A path $\pi \in T$ in a run r is *accepting* if $\mathit{Inf}(r(\pi)) \cap F \neq \emptyset$. The run r is *accepting* if every path in r is accepting. The *language* of $\mathcal{A}$ is the set $L(\mathcal{A}) := \{w \in \Sigma^\omega \mid \text{there is an accepting run of } \mathcal{A} \text{ on } w\}$.

The automaton $\mathcal{A}$ is *1-way* if $\delta(q, a) \in \mathcal{B}^+(Q \times \{1\})$, for all $q \in Q$ and $a \in \Sigma$. That means, $\mathcal{A}$ can only move the read-only head to the right. If $\mathcal{A}$ is 1-way, we assume that δ is of the form $\delta : Q \times \Sigma \to \mathcal{B}^+(Q)$. We call a 1-way automaton a *nondeterministic Büchi automaton* (NBA) if its transition function returns a disjunction of states for all inputs. We view the transition function δ of an NBA as a function of the form $\delta : Q \times \Sigma \to 2^Q$. This means that clauses are written as sets. Note that a run $r : T \to Q \times \mathbb{N}$ of an NBA $\mathcal{A}$ on $w \in \Sigma^\omega$ can be reduced to a single path π in r that is consistent with the transition function. Using standard terminology, we also call $r(\pi) \in (Q \times \mathbb{N})^\omega$ a run of $\mathcal{A}$ on w.

$$
\begin{array}{lll}
w,i \models p & \text{iff} & p \in w_i \\
w,i \models \mathsf{cl}(\alpha) & \text{iff} & \exists k \geq i : w_{i..k} \in L(\alpha), \text{ or } \forall k \geq i : \exists v \in L(\alpha) : w_{i..k} \preceq v \\
w,i \models \varphi \wedge \psi & \text{iff} & w,i \models \varphi \text{ and } w,i \models \psi \\
w,i \models \neg\varphi & \text{iff} & w,i \not\models \varphi \\
w,i \models \mathsf{X}\varphi & \text{iff} & w,i+1 \models \varphi \\
w,i \models \varphi \,\mathsf{U}\, \psi & \text{iff} & \exists k \geq i : w,k \models \psi \text{ and } \forall j : \text{if } i \leq j < k \text{ then } w,j \models \varphi \\
w,i \models \alpha \diamond\!\!\rightarrow \varphi & \text{iff} & \exists k \geq i : w_{i..k} \in L(\alpha) \text{ and } w,k \models \varphi \\
w,i \models \mathsf{Y}\varphi & \text{iff} & i > 0 \text{ and } w,i-1 \models \varphi \\
w,i \models \varphi \,\mathsf{S}\, \psi & \text{iff} & \exists k \leq i : w,k \models \psi \text{ and } \forall j : \text{if } k < j \leq i \text{ then } w,j \models \varphi \\
w,i \models \alpha \mathbin{\diamond\!\!\!\!-\!\!\!\rightarrow} \varphi & \text{iff} & \exists k \leq i : w_{k..i} \in L(\alpha) \text{ and } w,k \models \varphi
\end{array}
$$

Fig. 1. Interpretation of an RTL formula over P at a position $i \geq 0$ of a word $w \in (2^P)^\omega$

3 Temporal Logics with Expressions and Past Operators

In this section, we extend LTL with SEREs and past operators. We call the extension *Regular Temporal Logic*, RTL for short. The cores of the two industrial-standard property-specification languages PSL [1] and SVA [2] are fragments of RTL. The syntax of RTL over the set P of propositions is given by the grammar

$$\varphi ::= p \mid \mathsf{cl}(\alpha) \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathsf{X}\varphi \mid \varphi \,\mathsf{U}\, \varphi \mid \alpha \diamond\!\!\rightarrow \varphi \mid \mathsf{Y}\varphi \mid \varphi \,\mathsf{S}\, \varphi \mid \alpha \mathbin{\diamond\!\!\!\!-\!\!\!\rightarrow} \varphi\,,$$

where $p \in P$ and α is an SERE over P. The semantics of RTL is given in Figure 1. A word $w \in (2^P)^\omega$ is a *model* of an RTL formula φ if $w,0 \models \varphi$. The *language* of an RTL formula φ is $L(\varphi) := \{w \in (2^P)^\omega \mid w,0 \models \varphi\}$. The RTL formulas φ and ψ are *initially equivalent* if $L(\varphi) = L(\psi)$. They are *logically equivalent*, written as $\varphi \equiv \psi$, if $w,i \models \varphi \Leftrightarrow w,i \models \psi$, for all $i \in \mathbb{N}$ and $w \in (2^P)^\omega$. As for SEREs, we define the *size* $\|\varphi\|$ of an RTL formula φ as its syntactic length.

We define the following fragments of RTL. We call an RTL formula a *PSL formula* if it does not contain the operators Y, S, and $\mathbin{\diamond\!\!\!\!-\!\!\!\rightarrow}$. An *LTL formula* is a PSL formula that does not contain the operators cl and $\diamond\!\!\rightarrow$. An *SVA formula* is a PSL formula that does not contain the operators cl, X, and U. The fragments PLTL and PSVA, which extend LTL and SVA, respectively, with past operators, are defined as expected. Note that RTL and PSL extended with the past operators Y, S, and $\mathbin{\diamond\!\!\!\!-\!\!\!\rightarrow}$ coincide.

We use standard syntactic sugar, like the Boolean constants and connectives ff, tt, $\vee$, $\rightarrow$, and we define $\varphi \,\mathsf{R}\, \psi := \neg(\neg\varphi \,\mathsf{U}\, \neg\psi)$, $\varphi \,\mathsf{T}\, \psi := \neg(\neg\varphi \,\mathsf{S}\, \neg\psi)$, $\mathsf{Z}\varphi := \mathsf{Ytt} \rightarrow \mathsf{Y}\varphi$. Moreover, for an RTL formula φ and an SERE α, we write $\alpha \,\Box\!\!\rightarrow \varphi$ for $\neg(\alpha \diamond\!\!\rightarrow \neg\varphi)$ and $\alpha \,\boxminus\!\!\rightarrow \varphi$ for $\neg(\alpha \mathbin{\diamond\!\!\!\!-\!\!\!\rightarrow} \neg\varphi)$. Note that the standard unary temporal operators can easily be defined in the respective fragment. For instance, for PSVA we define $\mathsf{G}\varphi := \mathsf{tt}^* \,\Box\!\!\rightarrow \varphi$, $\mathsf{F}\varphi := \mathsf{tt}^* \diamond\!\!\rightarrow \varphi$, $\mathsf{H}\varphi := \mathsf{tt}^* \,\boxminus\!\!\rightarrow \varphi$, and $\mathsf{O}\varphi := \mathsf{tt}^* \mathbin{\diamond\!\!\!\!-\!\!\!\rightarrow} \varphi$.

Remark 1. In the PSL standard [1], we also have atomic formulas of the form $\mathsf{ended}(\alpha)$ and $\mathsf{prev}(\alpha)$, where α is an SERE. For instance, the word w satisfies $\mathsf{ended}(\alpha)$ at position i iff there is a subword u of w that ends at i and $u \in L(\alpha)$. The operators ended and prev can be seen as restricted variants of the past operator $\mathbin{\diamond\!\!\!\!-\!\!\!\rightarrow}$. For instance, in RTL, if $\varepsilon \notin L(\alpha)$, $\mathsf{ended}(\alpha)$ is syntactic

sugar for $\alpha \mathbin{\Diamond\!\!\!\!-\!\!\!\!\rightarrow} \mathsf{tt}$, and tt otherwise. Observe that ended and prev can only be applied to SEREs, and in contrast to $\mathbin{\Diamond\!\!\!\!-\!\!\!\!\rightarrow}$, it is not possible to define the classical past operators Y, H, and O with them. We also remark that the literature, e.g. [5, 9, 17, 24, 7] usually considers the essential core of the PSL standard to which the operators ended and prev do not belong. We follow this convention, i.e., the formulas in our fragment PSL of RTL do not contain $\mathsf{ended}(\alpha)$ and $\mathsf{prev}(\alpha)$. Finally, we remark that the automata constructions [5, 7] for PSL and SVA cannot cope with the operators ended and prev, which are handled by our construction in Section 4 for RTL.

Example 2. A standard example for showing that the past operators of PLTL can lead to more intuitive specifications is $\mathsf{G}(\mathit{grant} \rightarrow \mathsf{O}\mathit{request})$, i.e., every grant is preceded by a request [20]. An initially equivalent LTL formula is $\mathit{request}\ \mathsf{R}\ (\neg \mathit{grant} \vee \mathit{request})$. Let us now illustrate the beneficial use of SEREs and past operators. Suppose that a request is not a single event but a sequence of events, e.g., a request consists of a *start* event followed eventually by an *end* event and no *cancel* event happens between the *start* and the *end* event. Such sequences are naturally described by the SERE $(\mathit{start} \,;\, \mathsf{tt}^* \,;\, \mathit{end}) \cap (\neg \mathit{cancel})^*$. Using this SERE and the new past operator $\mathbin{\Diamond\!\!\!\!-\!\!\!\!\rightarrow}$, we can easily express in RTL the property that every grant is preceded by a request:

$$\mathsf{G}\big(\mathit{grant} \rightarrow \big(((\mathit{start} \,;\, \mathsf{tt}^* \,;\, \mathit{end}) \cap (\neg \mathit{cancel})^*) \,;\, \mathsf{tt}^* \mathbin{\Diamond\!\!\!\!-\!\!\!\!\rightarrow} \mathsf{tt}\big)\big)\,. \tag{1}$$

Note that according to the semantics of the operator $\mathbin{\Diamond\!\!\!\!-\!\!\!\!\rightarrow}$, the *end* event has to happen before or at the same time as the *grant* event. Alternatively, we can express the property in PLTL as

$$\mathsf{G}\big(\mathit{grant} \rightarrow \mathsf{O}\big(\mathit{end} \wedge \neg \mathit{cancel} \wedge \mathsf{Y}(\neg \mathit{cancel}\ \mathsf{S}\ (\mathit{start} \wedge \neg \mathit{cancel}))\big)\big)\,. \tag{2}$$

Although debatable, we consider that the RTL formula (1) is easier to understand than the PLTL formula (2). In SVA, we can express the property as $\mathit{norequest} \mathbin{\square\!\!\!\!\rightarrow} \neg \mathit{grant}$, where the SERE *norequest* describes the complement of the language $L\big(\mathsf{tt}^* \,;\, ((\mathit{start} \,;\, \mathsf{tt}^* \,;\, \mathit{end}) \cap (\neg \mathit{cancel})^*) \,;\, \mathsf{tt}^*\big)$, that is, $\mathit{norequest} := (a \cup b \,;\, d^* \,;\, c)^* \,;\, (c^* \cup b \,;\, d)$, where a, b, c, and d are the Boolean formulas $\neg \mathit{start} \vee \mathit{cancel}$, $\mathit{start} \wedge \neg \mathit{cancel}$, cancel, and $\neg \mathit{cancel} \wedge \neg \mathit{end}$, respectively. Note that in general, complementation of SEREs is difficult and can result in an exponential blowup with respect to the size of the given SERE.

Example 3. Let us give another example to illustrate the usefulness of past operators, in particular, the operator $\mathbin{\Diamond\!\!\!\!-\!\!\!\!\rightarrow}$. For $N \geq 1$ and $i \in \{0, \ldots, N-1\}$, consider the RTL formula $\Phi_{N,i} := \mathsf{G}\big(\mathit{send}_i \rightarrow \big(\mathit{switch}_i \cap (\mathit{init} \,;\, (\neg \mathit{init})^*) \mathbin{\Diamond\!\!\!\!-\!\!\!\!\rightarrow} \mathsf{tt}\big)\big)$, where switch_i counts the number of *switch* events modulo N, i.e.,

$$\begin{aligned} \mathit{switch}_i := \Big(\underbrace{(\neg \mathit{switch})^* \,;\, \mathit{switch} \,;\, \ldots \,;\, (\neg \mathit{switch})^* \,;\, \mathit{switch}}_{N \text{ times}} \Big)^* \,; \\ \underbrace{(\neg \mathit{switch})^* \,;\, \mathit{switch} \,;\, \ldots \,;\, (\neg \mathit{switch})^* \,;\, \mathit{switch}}_{i \text{ times}} \,;\, (\neg \mathit{switch})^*\,. \end{aligned} \tag{3}$$

Intuitively, $\Phi_{N,i}$ expresses the property that the process i is only allowed to send a data item if it possesses the token. The process i possesses the token iff $i \equiv 0 \bmod N$ *switch* events occurred previously since the last *init* event. Note that this property is not expressible in LTL since it is not star-free.

The negation of the PSL formula

$$\big((\neg \mathit{init})^* \diamond\!\!\rightarrow \mathit{send}_i\big) \vee \mathsf{F}\big(\mathit{init} \wedge \big((\mathsf{tt}\,;(\neg \mathit{init})^*) \cap \big(\bigcup_{j\neq i} \mathit{switch}_j\big) \diamond\!\!\rightarrow \mathit{send}_i\big)\big) \quad (4)$$

is initially equivalent to $\Phi_{N,i}$. Note that the size of the formula (4) is quadratic in N, whereas the size of the formula (3) is only linear in N. In Section 5, we prove that PSVA is exponentially more succinct than PSL.

In general, for writing specifications, RTL possesses the advantage of PLTL over LTL and the advantage of PSL/SVA over LTL, namely, additional operators for referring to the past and SEREs for describing sequences of events.

4 From RTL to Nondeterministic Automata

In this section, we present a translation from RTL formulas into language-equivalent NBAs. Similar to the well-known translation for LTL formulas into NBAs, our translation comprises two steps: for a given RTL formula, we first construct an alternating automaton, which we then translate into an NBA. Throughout this section, we fix a finite set P of propositions.

4.1 From RTL to Loop-Free and Locally 1-Way 2ABAs

In this subsection, we assume that φ is an RTL formula over P and φ is in *negation normal form*, i.e., the negation symbol $\neg$ only occurs directly in front of the atomic subformulas of φ. Note that every RTL formula ψ can be rewritten into a logically equivalent RTL formula in negation normal form over an extended language, where we use the additional Boolean connective $\vee$ and the additional operators R, T, Z, $\Box\!\!\rightarrow$, and $\boxminus\!\!\rightarrow$ as primitives. The size of the resulting formula is at most $2\|\psi\|$. For rewriting a formula into negation normal form, we use the logical equivalences $\neg\neg\gamma \equiv \gamma$, $\neg\mathsf{X}\gamma \equiv \mathsf{X}\neg\gamma$, $\neg\mathsf{Y}\gamma \equiv \mathsf{Z}\neg\gamma$, and $\neg\mathsf{Z}\gamma \equiv \mathsf{Y}\neg\gamma$.

Due to space limitations, we do not provide the construction of the 2ABA $\mathcal{A}_\varphi$ for the RTL formula φ here. Instead, we only briefly highlight the similarities and the differences to the standard constructions for LTL, PLTL, SVA, and PSL [26, 14, 5, 7]. The construction in [7] additionally handles SEREs with local variables. Our construction can easily be extended by this feature. However, for the ease of exposition, we focus here on how to handle the temporal past and future operators of RTL efficiently. As the standard construction for PSL [5], the state space of the 2ABA $\mathcal{A}_\varphi$ consists of the subformulas of the given RTL formula and the states of the automata for the SEREs. We introduce a special symbol # to mark the beginning of the input word. With this symbol, $\mathcal{A}_\varphi$ checks in a run whether the read-only head is at the first position of the input word. We need some auxiliary states for such a check. The new operators $\boxminus\!\!\rightarrow$ and $\diamondminus\!\!\rightarrow$ are then easily handled since $\mathcal{A}_\varphi$ is alternating and 2-way. From the construction, we obtain the following lemmas.

Lemma 4. *The 2ABA $\mathcal{A}_\varphi$ accepts the language $\{\#w \mid w \in L(\varphi)\}$.*

Lemma 5. *The 2ABA $\mathcal{A}_\varphi$ has size at most $4 + 2^{\|\varphi\|}$.*

The 2ABA $\mathcal{A}_\varphi$ has some additional properties, which we exploit in Section 4.2 for constructing the NBA. Namely, $\mathcal{A}_\varphi$ is loop-free [14, 12] and locally 1-way.

Intuitively speaking, loop-freeness means that an automaton cannot visit a configuration twice on the same computation branch. Formally, it is defined as follows for a 2ABA $\mathcal{B} = (S, \Sigma, \eta, s_I, E)$. Let $\Pi(\mathcal{B})$ be the set of words of the form $(s_0, j_0)(s_1, j_1) \ldots \in (S \times \mathbb{N})^\omega$ such that $(s_0, j_0) = (s_I, 0)$ and for all $i \in \mathbb{N}$, there is some $a \in \Sigma$ and a set $M \subseteq S \times \mathbb{Z}$ with $(s_{i+1}, j_{i+1} - j_i) \in M$ and M is a minimal model of $\eta(s_i, a)$, i.e, $M \vDash \eta(s_i, a)$ and $M \setminus \{c\} \nvDash \eta(s_i, a)$, for all $c \in M$. The automaton $\mathcal{B}$ is *loop-free* if for all words $\pi \in \Pi(\mathcal{B})$, there are no integers $i, j \in \mathbb{N}$ with $i \neq j$ such that $\pi_i = \pi_j$. Recall that π_i and π_j are configurations, which consist of the current state and the current position of the read-only head.

Lemma 6. *The 2ABA $\mathcal{A}_\varphi$ is loop-free.*

A 2ABA $\mathcal{B} = (S, \Sigma, \eta, s_I, E)$ is *locally 1-way* if $\eta(s, b) \in \mathcal{B}^+(S \times \{0, 1\}) \cup \mathcal{B}^+(S \times \{-1, 0\})$, for every $s \in S$ and $b \in \Sigma$. We remark that any 2ABA can be transformed into a language-equivalent 2ABA that is locally 1-way by doubling the state space. However, such a transformation is not needed for $\mathcal{A}_\varphi$, since A_φ is already constructed in such a way that it is locally 1-way.

Lemma 7. *The 2ABA $\mathcal{A}_\varphi$ is locally 1-way.*

4.2 From Loop-Free and Locally 1-Way 2ABAs to NBAs

In the following, we show how the alternating automaton from the previous subsection for an RTL formula in negation normal form can be translated into an NBA. The presented construction is based on an improvement of an alternation-elimination construction from [12]. Here, we additionally exploit the fact that the given 2ABA is locally 1-way. Overall, for an RTL formula ψ, the resulting language-equivalent NBA has size $\mathcal{O}(2^{3 \cdot 2^{2\|\psi\|}})$. With the construction in [12], we would obtain an NBA of size $\mathcal{O}(2^{4 \cdot 2^{2\|\psi\|}})$. Another advantage of the new construction is that it avoids the explicit representation of an extended alphabet, which is used in one of the intermediate construction steps in [12] and which is of exponential size. The presented construction also allows for a symbolic implementation [11], which can be used in tools like NuSMV [8] for satisfiability and finite-state model checking. See [6] for such implementations and an evaluation of constructions for the special case of 1-way alternating Büchi automata.

Theorem 8. *For a loop-free and locally 1-way 2ABA $\mathcal{A}$, there is a language-equivalent NBA $\mathcal{B}$ of size $\mathcal{O}(|\Sigma| \cdot 2^{2\|\mathcal{A}\|})$, where Σ is the alphabet of $\mathcal{A}$.*

The intuition for the construction of Theorem 8 is as follows. For an input word w, the NBA $\mathcal{B}$ guesses a run r of $\mathcal{A} = (Q, \Sigma, \delta, q_I, F)$ on w and checks whether this run is accepting. For this, as in [25,12], $\mathcal{B}$ represents r as a sequence of state sets $R_0 R_1 \ldots \in (2^Q)^\omega$, where each R_i contains the state q iff there is a path in

r that visits (q, i). In the case where $\mathcal{A}$ is 1-way, each R_i consists of the states that occur in the ith level of the run r. Note that in the general case where $\mathcal{A}$ is 2-way, R_i might contain states that occur in different levels of r. For instance, R_i contains the states q and q' from different levels if r contains a path of the form $(q_I, 0) \dots (q, i) \dots (q', i) \dots$. Since $\mathcal{A}$ is locally 1-way, we can locally check whether such a sequence $R_0 R_1 \dots$ represents a run of $\mathcal{A}$ on w. For doing so, $\mathcal{B}$ stores the set R_{i+1} and the letter w_{i+2} after reading the ith letter of w. For a state $q \in R_i$ with $\delta(q, w_i) \in \mathcal{B}^+(Q \times \{0, 1\})$, the set $(R_i \times \{0\}) \cup (R_{i+1} \times \{1\})$ must be a model of $\delta(q, w_i)$. $\mathcal{B}$ checks this when reading the letter w_i. For $\delta(q, w_i) \in \mathcal{B}^+(Q \times \{-1, 0\})$ and $i > 0$, $(R_{i-1} \times \{-1\}) \cup (R_i \times \{0\})$ must be a model of $\delta(q, w_i)$. $\mathcal{B}$ already checks this when it reads the $(i-1)$th input letter by using the guessed letter w_i. Additionally, $\mathcal{B}$ must check that every path in r visits configurations with an accepting state infinitely often. Since $\mathcal{A}$ is loop-free the run r is accepting iff there are indexes $i_0 < i_1 < \dots$ such that each path in r that visits a configuration (q, i_j) visits a configuration with an accepting state before visiting (q', i_{j+1}), for every $j \in \mathbb{N}$. Similar to the alternation-elimination construction by Miyano and Hayashi [22] for 1-way alternating Büchi automata, $\mathcal{B}$ checks this property with an additional component in the state space and its set of accepting states.

We obtain the following result by putting the two constructions together.

Theorem 9. *For any RTL formula ψ, there is a language-equivalent NBA $\mathcal{C}$ of size $\mathcal{O}(2^{3 \cdot 2^{2\|\psi\|}})$.*

Proof. First, we transform ψ into a logically equivalent formula ψ' that is in negation normal of size $2\|\psi\|$. Let $\mathcal{A}_{\psi'}$ be the 2ABA that we obtain from ψ' by the construction in Section 4.1. By the Lemmas 5, 6, and 7, $\mathcal{A}_{\psi'}$ is loop-free, locally 1-way, and $\|\mathcal{A}_{\psi'}\| \leq 4 + 2^{2\|\psi\|}$. By Lemma 4, $\mathcal{A}_{\psi'}$ accepts the language $\{\#w \mid w \in L(\psi)\}$. By Theorem 8, we translate $\mathcal{A}_{\psi'}$ into a language-equivalent NBA $\mathcal{B}$ with $\mathcal{O}(2^{3 \cdot 2^{2\|\psi\|}})$ states. From $\mathcal{B}$, it is easy to obtain an NBA $\mathcal{C}$ with $L(\mathcal{C}) = L(\psi)$ and $\|\mathcal{C}\| \in \mathcal{O}(2^{3 \cdot 2^{2\|\psi\|}})$. □

We remark that the upper bound of the NBA in Theorem 9 can be improved by taking the number of distinct subformulas into account instead of the syntactic length of the given RTL formula. We omit such a refined analysis here.

4.3 Consequences of the Translation

We conclude this section by proving some facts that follow from Theorem 9.

Since SVA can already express all ω-regular languages, we have that RTL describes exactly the ω-regular languages. Moreover, SVA, PSL, and RTL share the same computational complexity. In particular, the satisfiability and the model-checking problem for RTL are EXPSPACE-complete in general and PSPACE-complete for RTL formulas with a bounded number of intersection operators. Another similarity between the logics is that they all have the small model property of doubly exponential size. In particular, there is a constant $c > 0$ such that a satisfiable RTL formula φ has a model of the form uv^ω with $|uv| \leq c \cdot 2^{3 \cdot 2^{2\|\varphi\|}}$.

Since PSL/SVA and RTL describe the same class of properties, the question arises of their relative succinctness. The next theorem states an upper bound on the translation from RTL to SVA. Roughly speaking, for the proof, we translate an RTL formula into an NBA and then into an ω-regular expression, which we finally translate into an SVA formula.

Theorem 10. *For any RTL formula φ, there is an initially equivalent SVA formula of size $2^{\mathcal{O}(2^{2^{2\|\varphi\|+2}})}$ and in which the intersection operator does not occur.*

It is fair to ask whether the upper bound in Theorem 10 is optimal, i.e., whether there is a family of RTL formulas such that every initially equivalent family of PSL formulas must be triply exponentially larger. The result on the small model property shows that such a lower bound cannot be proved by comparing the model sizes (see, e.g., the Gap Lemma in [18]). We were only able to establish an exponential lower bound. This result is presented in the next section.

5 Succinctness Gaps

In this section, we prove an exponential succinctness gap between RTL and PSL/SVA, i.e., there is a family $(\Phi_n)_{n>0}$ of RTL formulas such that for every family $(\Psi_n)_{n>0}$ of PSL or SVA formulas, if Ψ_n is initially equivalent to Φ_n for all $n > 0$, then $\|\Psi_n\|$ is exponential in $\|\Phi_n\|$. In fact, our result is stronger since the formulas Φ_n that we define are just PSVA formulas. The proof of this succinctness result can easily be adapted to show that PSVA and, hence, RTL, is double exponentially more succinct than PLTL.

Our proof for the succinctness gap between PSVA and SVA has a similar flavor as the proof in [21], which shows that PLTL is exponentially more succinct than LTL. However, our proof is more involved since we must take SEREs into account. In fact, the formulas in the family of PLTL formulas that is used in [21] are initially equivalent to SVA formulas of linear size. From this observation, we conclude that SVA is exponentially more succinct than LTL.

Lemma 11. *For every $n > 0$, there is an SVA formula Θ_n such that for any LTL formula Ξ_n, if $L(\Xi_n) = L(\Theta_n)$ then $\|\Xi_n\| \in \Omega(2^{\|\Theta_n\|})$.*

Let us now turn to the succinctness gap between PSVA and SVA. For this, we first introduce so-called n-counting words, which can be defined in SVA by formulas of size $\mathcal{O}(n)$. In the following, let $n > 0$, P_n be the set $\{c_0, \ldots, c_{n-1}, p, q\}$ of propositions, and Σ_n the alphabet 2^{P_n}. The *n-value* of the letter $b \in \Sigma_n$ is $val_n(b) := \sum_{0 \le i < n} 2^{c'_i}$ with $c'_i := 1$ if $c_i \in b$ and $c'_i := 0$, otherwise. In other words, the n-value of b is obtained by reading $c_0, \ldots, c_{n-1}$ as bits of a positive integer in binary representation. A word $w \in \Sigma_n^\omega$ is *n-counting* if $val_n(w_0) = 0$ and $val_n(w_{i+1}) \equiv val_n(w_i) + 1 \mod 2^n$, for all $i \in \mathbb{N}$.

Lemma 12. *For every $n > 0$, there is an SVA formula $count_n$ of size $\mathcal{O}(n)$ such that $L(count_n) \subseteq \Sigma_n^\omega$ is the language of n-counting words.*

An *n-segment* of a word $w \in \Sigma_n^\omega$ is a subword $v = w_i \ldots w_{i+2^n-1}$ such that $i \equiv 0 \mod 2^n$, for some $i \in \mathbb{N}$. The n-segment v is *initial* if $i = 0$. For a proposition

$r \in P$, the words $u, v \in \Sigma_n^*$ are *r-equal* if $|u| = |v|$ and $r \in u_i \Leftrightarrow r \in v_i$, for all $i \in \mathbb{N}$ with $i < |v|$. Let L_n and L'_n be the following languages:

- L_n consists of the n-counting words $w \in \Sigma_n^\omega$ such that if an n-segment of w is p-equal to the initial n-segment w then they are also q-equal.
- L'_n consists of the n-counting words $w \in \Sigma_n^\omega$ such that if the n-segments u and v of w are p-equal then they are also q-equal.

Lemma 13. *For every $n > 0$, there is a PSVA formula Φ_n of size $\mathcal{O}(n)$ such that $L(\Phi_n) = L_n$.*

Lemma 14. *For every $n > 0$, if $\mathcal{B}$ is an NBA with $L(\mathcal{B}) = L'_n$ then $\|\mathcal{B}\| \geq 2^{2^{2^n}}$.*

With the above lemmas we obtain our succinctness result for PSVA and SVA.

Theorem 15. *For every $n > 0$, there is a PSVA formula Φ_n such that $L(\Phi_n) = L_n$ and for every SVA formula Ψ_n, if $L(\Psi_n) = L_n$ then $\|\Psi_n\| \in \Omega(2^{\|\Phi_n\|})$.*

Proof. For a given $n > 0$, take the PSVA formula Φ_n from Lemma 13. Suppose that Ψ_n is an SVA formula that is initially equivalent to Φ_n. Let $\Psi'_n := \mathit{count}_n \wedge \mathsf{G}(\neg c_0 \wedge \cdots \wedge \neg c_{n-1} \rightarrow \Psi_n)$. Note that Ψ'_n expresses that a model is n-counting and each two p-equal n-segments in a model are also q-equal, i.e., $L(\Psi'_n) = L'_n$. By Theorem 9, there is an NBA $\mathcal{B}$ of size $2^{2^{\mathcal{O}(\|\Psi'_n\|)}}$ and $L(\mathcal{B}) = L(\Psi'_n)$. By Lemma 14, we have that $\|\mathcal{B}\| \geq 2^{2^{2^n}}$. It follows that $\|\Psi'_n\| \in \Omega(2^{\|\Phi_n\|})$. Since Ψ'_n is linear in the size of Ψ_n, we conclude that $\|\Psi_n\| \in \Omega(2^{\|\Phi_n\|})$. □

Note that L_n is a star-free language, i.e., there is an LTL formula φ_n such that $L(\varphi_n) = L_n$. We can easily adapt the proof of Theorem 15 to obtain a double exponential succinctness gap between PSVA and PLTL.

Corollary 16. *For every $n > 0$, there is a PSVA formula Φ_n such that $L(\Phi_n) = L_n$ and for any PLTL formula Ξ_n, if $L(\Xi_n) = L_n$ then $\|\Xi_n\| \in \Omega(2^{2^{\|\Phi_n\|}})$.*

Remark 17. We conclude this section by stating some open problems related to the presented succinctness gaps. First, it remains open whether the exponential succinctness gap still holds between RTL and extensions of PSL/SVA with restricted variants of the past operators like the ones discussed in Remark 1. We did not succeeded in proving such a gap, neither did we succeed in expressing the languages L_n concisely in such an extension. Second, it remains open whether the succinctness gaps carry over to a fixed and finite proposition set. Note that the proposition sets P_n over which the PSVA formulas Φ_n are defined grow linearly in n. As shown in [13], we can encode any number of propositions by a single proposition. However, the sizes of the adapted formulas for Φ_n are no longer linear in n. In particular, the sizes of the adapted SEREs in Lemma 13 are quadratic in n. It is not obvious how to adapt these SEREs so that their sizes remain linear in n. Therefore, for a fixed and finite proposition set, we only obtain a superpolynomial succinctness gap between PSVA and SVA. Note that for similar reasons, the adapted proof of the succinctness gap between PLTL and LTL in [21, 19] for a fixed and finite proposition set also only shows that PLTL is superpolynomially more succinct than LTL.

6 Conclusion

In this paper, we have proposed the temporal logic RTL, which extends PSL and SVA with past operators. We have analyzed its complexity and our results show that RTL and PSL/SVA are similarly related as PLTL and LTL with respect to expressiveness, succinctness, and the computational complexities of the satisfiability and the model-checking problem. It remains to be seen whether the advantages of RTL over PSL and SVA pay off in practice. The presented translation for RTL into NBAs shows that the additional cost for handling past operators is small and should not be a burden in implementing RTL in system verification. Our preliminary experience with a prototype implementation for the model checker NuSMV are promising.[2]

References

1. IEEE standard for Property Specification Language (PSL). IEEE Std 1850TM (October 2005)
2. IEEE standard for SystemVerilog—unified hardware design, specification, and verification language. IEEE Std 1800TM (November 2005)
3. Armoni, R., Fix, L., Flaisher, A., Gerth, R., Ginsburg, B., Kanza, T., Landver, A., Mador-Haim, S., Singerman, E., Tiemeyer, A., Vardi, M.Y., Zbar, Y.: The ForSpec temporal logic: A new temporal property-specification language. In: Katoen, J.-P., Stevens, P. (eds.) TACAS 2002. LNCS, vol. 2280, pp. 296–311. Springer, Heidelberg (2002)
4. Banieqbal, B., Barringer, H.: Temporal logic with fixed points. In: Banieqbal, B., Pnueli, A., Barringer, H. (eds.) Temporal Logic in Specification. LNCS, vol. 398, pp. 62–74. Springer, Heidelberg (1989)
5. Ben-David, S., Bloem, R., Fisman, D., Griesmayer, A., Pill, I., Ruah, S.: Automata construction algorithms optimized for PSL. Technical report, The Prosyd Project (2005), `http://www.prosyd.org`
6. Bloem, R., Cimatti, A., Pill, I., Roveri, M.: Symbolic implementation of alternating automata. Int. J. Found. Comput. Sci. 18(4), 727–743 (2007)
7. Bustan, D., Havlicek, J.: Some complexity results for SytemVerilog assertions. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 205–218. Springer, Heidelberg (2006)
8. Cimatti, A., Clarke, E.M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: An opensource tool for symbolic model checking. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002, vol. 2404, pp. 359–364. Springer, Heidelberg (2002)
9. Cimatti, A., Roveri, M., Semprini, S., Tonetta, S.: From PSL to NBA: A modular symbolic encoding. In: FMCAD 2006, pp. 125–133. IEEE Computer Society Press, Los Alamitos (2006)
10. Cimatti, A., Roveri, M., Sheridan, D.: Bounded verification of Past LTL. In: Hu, A.J., Martin, A.K. (eds.) FMCAD 2004. LNCS, vol. 3312, pp. 245–259. Springer, Heidelberg (2004)
11. Clarke, E.M., Grumberg, O., Hamaguchi, K.: Another look at LTL model checking. Form. Method. Syst. Des. 10(1), 47–71 (1997)

[2] See `www.inf.ethz.ch/~daxc/rtl2ba` for the most recent version of our tool.

12. Dax, C., Klaedtke, F.: Alternation elimination by complementation. In: Cervesato, I., Veith, H., Voronkov, A. (eds.) LPAR 2008. LNCS, vol. 5330, pp. 214–229. Springer, Heidelberg (2008)
13. Demri, S., Schnoebelen, P.: The complexity of propositional linear temporal logics in simple cases. Inf. Comput. 174(1), 84–103 (2002)
14. Gastin, P., Oddoux, D.: LTL with past and two-way very-weak alternating automata. In: Rovan, B., Vojtáš, P. (eds.) MFCS 2003. LNCS, vol. 2747, pp. 439–448. Springer, Heidelberg (2003)
15. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic. MIT Press, Cambridge (2000)
16. Kupferman, O., Piterman, N., Vardi, M.Y.: Extended temporal logic revisited. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, pp. 519–535. Springer, Heidelberg (2001)
17. Lange, M.: Linear time logics around PSL: Complexity, expressiveness, and a little bit of succinctness. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 90–104. Springer, Heidelberg (2007)
18. Lange, M.: A purely model-theoretic proof of the exponential succinctness gap between CTL^+ and CTL. Inform. Process. Lett. 108(5), 308–312 (2008)
19. Laroussinie, F., Markey, N., Schnoebelen, P.: Temporal logic with forgettable past. In: LICS 2002, pp. 383–392. IEEE Computer Society Press, Los Alamitos (2002)
20. Lichtenstein, O., Pnueli, A., Zuck, L.D.: The glory of the past. In: Parikh, R. (ed.) Logic of Programs 1985. LNCS, vol. 193, pp. 196–218. Springer, Heidelberg (1985)
21. Markey, N.: Temporal logic with past is exponentially more succinct. Bulletin of the EATCS 79, 122–128 (2003)
22. Miyano, S., Hayashi, T.: Alternating finite automata on ω-words. Theoret. Comput. Sci. 32(3), 321–330 (1984)
23. Pnueli, A.: The temporal logic of programs. In: FOCS 1977, pp. 46–57. IEEE Computer Society Press, Los Alamitos (1977)
24. Pnueli, A., Zaks, A.: PSL model checking and run-time verification via testers. In: Misra, J., Nipkow, T., Sekerinski, E. (eds.) FM 2006. LNCS, vol. 4085, pp. 573–586. Springer, Heidelberg (2006)
25. Vardi, M.Y.: A note on the reduction of two-way automata to one-way automata. Inform. Process. Lett. 30(5), 261–264 (1989)
26. Vardi, M.Y.: An automata-theoretic approach to linear temporal logic. In: Moller, F., Birtwistle, G. (eds.) Logics for Concurrency. LNCS, vol. 1043, pp. 238–266. Springer, Heidelberg (1996)
27. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification. In: LICS 1986, pp. 332–344. IEEE Computer Society Press, Los Alamitos (1986)
28. Wolper, P.: Temporal logic can be more expressive. Information and Control 56(1/2), 72–99 (1983)