Monitor Circuits for LTL with Bounded and Unbounded Future*

Bernd Finkbeiner and Lars Kuhtz

Universität des Saarlandes
66123 Saarbrücken, Germany
{finkbeiner|kuhtz}@cs.uni-sb.de

Abstract. Synthesizing monitor circuits for LTL formulas is expensive, because the number of flip-flops in the circuit is exponential in the length of the formula. As a result, the IEEE standard PSL recommends to restrict monitoring to the *simple subset* and use the full logic only for static verification. We present a novel construction for the synthesis of monitor circuits from specifications in LTL. In our construction, only subformulas with unbounded-future operators contribute to the exponential blowup. We split the specification into a bounded and an unbounded part, apply specialized constructions for each part, and then compose the results into a monitor for the original specification. Since the unbounded part in practical specifications is often very small, we argue that, with the new construction, it is no longer necessary to restrict runtime verification to the simple subset.

1 Introduction

In runtime verification, we monitor the running system and check on-the-fly whether the desired properties hold. Unlike in static verification, where the verification algorithm is executed at design-time and can therefore afford to spend significant time and resources, runtime verification algorithms must run in synchrony with the monitored system and usually even share the resources of the implementation platform.

For specifications in succinct temporal logics, such as LTL this is problematic, because one can easily specify properties that are hard to monitor. For example, a simple cache property like "it is always the case that if the present input vector has previously been seen in the last 100 steps, a cache hit is reported" can be specified with an LTL formula that is linear in the size of the input vector, but the construction of a deterministic monitor automaton would yield an intractable number of states, because every possible combination of the vectors needs a separate state (cf. [1]).

In the IEEE standard PSL [2], which is based on LTL, these considerations have led to the recommendation that only a restricted sublogic, the so-called

* This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS).

simple subset, is to be used in runtime verification (cf. [3]). The simple subset restricts the use of disjunctions in the specification. While the simple subset has been shown to lead to small monitoring circuits, the restriction is often unfortunate, especially when specifications are shared between model checking and runtime verification. Rather than stating, for example, that a *disjunction* of temporal output patterns is safe, the simple subset requires that every output pattern be described as a deterministic consequence of a specific input pattern.

From an automata-theoretic standpoint, the temporal formulas in the simple subset correspond to *universal automata*, where the transitions relate states to conjunctions of successor states. There is a linear translation from temporal formulas in the simple subset to universal automata, and universal automata can be implemented with a linear number of flip-flops. For unrestricted formulas, on the other hand, a direct translation results in an *alternating automaton*, whose transitions have both conjunctions and disjunctions. It is the translation from alternating to universal automata that causes the exponential blow-up.

However, it is well-known that the membership problem for alternating automata can be solved directly, without a translation to universal automata and *in linear time*, as long as the relevant part of the input word is available *in reverse order*. Rather than evaluating in a forward manner, which corresponds to determinization, the automaton is simply evaluated backward, like a combinatorial circuit. The question arises, if, by using this alternative membership test, one can avoid the exponential blow-up in the size of the monitoring circuit. Is the restriction to the simple subset in fact unnecessary?

In this paper, we present a new monitoring approach for general temporal specifications that avoids the translation to universal automata when possible. For example, the truth value of the cache specification at some position i is determined by the observations at positions $i, i+1, \ldots, i+99$. The specification can therefore be evaluated by unrolling the alternating automaton over 100 steps, avoiding the exponential increase in the size of the circuit.

To make this idea precise, we define, for each subformula of a specification, its temporal *horizon*, which indicates a future point in time by which the value of the subformula for the present position is guaranteed to be determined. Subformulas with finite horizons define languages that are finite themselves.

The study of events characterized by finite languages goes back to Kleene's *definite events* [4] and the *locally testable events* of McNaughton and Papert [5]. In the terminology of McNaughton and Papert, a set E of words is called a *locally testable event in the strict sense* if there exists a finite language L, such that all subwords of each word in E have a prefix in L. McNaughton and Papert construct an automaton that maintains an input buffer that is large enough to capture the largest words in L. In each step, a combinatorial circuit checks if the pipeline content belongs to L.

In our setting, the languages recognizable by such an automaton correspond to LTL formulas of the form $\mathrm{G}\,\phi$, where ϕ contains only bounded future operators. In this paper, we extend this idea to allow the bounded subformulas to occur within general temporal formulas. For each subformula with finite horizon

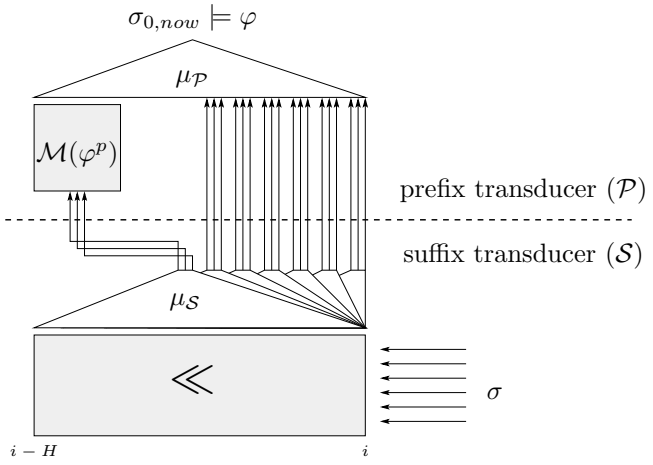

Fig. 1. Overview of the monitor construction.

t we introduce a pipeline and a combinatorial circuit that computes, online as new elements enter the pipeline, a Boolean value that corresponds to the truth of the formula from the perspective of t steps ago in the past. From the *delayed* truth values of the subformulas we extrapolate the *current* truth value of the formula. This is possible because the *truncated-path semantics* [6] (as used, for example, in PSL) provides default values for subformulas that refer to the future beyond the current cut-off point. The truncated-paths semantics distinguishes between *strong* and *weak* subformulas: for example, the strong specification "$\mathrm{X}\,\mathrm{X}\,p$" is true only if the visible trace is at least two positions long and p holds in the second position. Negation flips between the strong and weak interpretation. Given a pipeline that contains the delayed truth values of the subformulas, we can therefore construct an extrapolation circuit that applies, at each position, the truncated-trace semantics instantly to the entire path suffix stored in the pipeline.

Figure 1 gives an overview of our construction. We say a subformula is *bounded* if its horizon is finite, and *unbounded* otherwise. We call the part of the monitor that deals with the pipeline storage and the evaluation of the bounded formulas the *suffix transducer* $\mathcal{S}$: for some infinite trace σ, the suffix transducer evaluates the *suffix* of σ, from the delayed position in the trace onward, to derive the truth value of the bounded-future formulas.

Correspondingly, the part of the monitor that deals with unbounded formulas is called the *prefix transducer* $\mathcal{P}$: the prefix transducer evaluates the *prefix* of σ, up to the currently observed position i, to derive the truth value of the complete specification. The prefix transducer is based on a universal automaton $\mathcal{U}(\varphi)$, which checks whether a given prefix of the trace satisfies φ.

The extrapolation function, denoted by $\mu_{\mathcal{P}}$ in Figure 1, evaluates the part of the trace that is currently stored in the pipeline, i.e., the difference between the delayed position considered by $\mathcal{U}(\varphi)$ and the present position i.

The resulting circuit has the following properties. If the specification is (1) *simple*, (2) *bounded*, or (3) a *combination* thereof (a formula that is simple except for subformulas that are not simple but bounded), the circuit is polynomial in the specification. If the specification is (4) neither simple nor bounded, then the circuit is exponential in the size of the specification *after* removing all bounded subformulas.

While the possibility of an exponential blow-up is thus not excluded, it is our experience that even case (4) rarely leads to a blow-up in practice. Specifications that are neither simple nor bounded mostly occur when the correct behavior is specified in terms of a correlation of different events such as "$\mathrm{G}((A or B)\,\mathrm{U}(C or D))$," where the events A, B, C and D are specified by bounded formulas expressing certain finite input or output patterns that constitute events. Once the bounded subformulas have been removed, the specification becomes very small and the resulting monitoring circuit typically fits easily on an FPGA board.

Related Work. Monitoring LTL is a key problem in runtime verification (cf. [7,8,9,10,11]). The two most prominent tools for the synthesis of monitor circuits from the simple subset of PSL are FoCs [12], developed at IBM Haifa, and MBAC by Boulé and Zilic [13]. For unrestricted temporal logic, an automata-theoretic construction (based on determinization) is due to Armoni et al. [14]. Our prefix transducer is inspired by this construction.

More generally, the problem of translating LTL and logics based on LTL to automata occurs in both runtime verification and model checking. Constructions aimed at model checking (cf. [15,16,17,18]) are, however, not immediately applicable to runtime verification. First, such constructions typically only produce nondeterministic automata, rather than deterministic monitors. Hence, a further exponential determinization step is required to obtain a monitor. Second, these constructions typically produce automata over infinite words rather than automata or transducers over finite words.

Our approach is based on the truncated-path semantics [6] used in PSL. The truncated-path semantics differs from the bad-prefix semantics used in several monitoring approaches (cf. [8,19,20]), where a finite-word automaton is constructed that recognizes the "bad prefixes" of the language of an infinite-word automaton, i.e., the set of prefixes that cannot be extended to accepted infinite words [1]. In the truncated-path semantics, strong specifications may be violated on a prefix even if a satisfying extension exists.

Locally testable events were introduced by [21] and [5] and broadly studied in the literature (refer e.g. to [22]). In [23] Kupferman, Lustig, and Vardi point out the particular relevance of locally testable events in a strict sense (as introduced in [5]), which they call locally checkable properties. They emphasize the low memory footprint of monitors for locally checkable properties, since their size depend only on the number of variables and the length of the pipeline.

The key contribution of this paper is to exploit the local testablility of bounded subformulas that occur within general temporal properties by the introduction of a pipeline into the monitoring circuit. Because bounded subformulas are evaluated directly, based on the pipeline content, rather than folded into the determinization of the prefix transducer, the resulting circuit can be exponentially smaller than the circuits constructed by previous approaches.

2 Temporal Specifications

Our approach is based on LTL with an bounded and an unbounded version of the temporal operators.[1]

Definition 1 (Syntax). *Given a set of atomic propositions AP, let φ_1 and φ_2 be temporal formulas, and let $i, j \in \mathbb{N} \cup \{\infty\}$. Then the following are temporal formulas over AP:*

$$\text{all } p \in AP \qquad \neg\varphi_1 \qquad \varphi_1 \wedge \varphi_2 \qquad \varphi_1 \,\mathrm{U}_{(i,j)}\, \varphi_2.$$

The main operator of the logic is the Until operator $\varphi_1 \,\mathrm{U}_{(l,u)}\, \varphi_2$, which we use in its parameterized form, where $l, u \in \mathbb{N} \cup \{\infty\}$ indicate a lower and upper bound, respectively, of the interval within which φ_2 must hold. As usual, the Until operator subsumes the Next, Eventually, and Always operators:

$$\mathrm{X}\,\varphi \;\equiv\; \mathit{true}\,\mathrm{U}_{(1,1)}\,\varphi \qquad \mathrm{F}\,\varphi \;\equiv\; \mathit{true}\,\mathrm{U}_{(0,\infty)}\,\varphi \qquad \mathrm{G}\,\varphi \;\equiv\; \neg\,\mathrm{F}\,\neg\varphi$$

We call a formula *simple* if the operand of every negation and the right-hand operand of every Until is a Boolean expression over AP. The *size* $|\varphi|$ of a formula φ is the number of subformulas plus, for parameterized subformulas, the sum of all constants.

We use a *truncated semantics* [6], defined over finite words from the alphabet 2^{AP}. We denote the length of a finite or infinite word σ by $|\sigma|$, where the empty word has length $|\epsilon| = 0$, a finite word $\sigma = \sigma(0), \sigma(1), \sigma(2), \ldots \sigma(n-1)$ has length $|\sigma| = n$ and an infinite word $\sigma = \sigma(0), \sigma(1), \sigma(2), \ldots$ has length $|\sigma| = \infty$. For a finite or infinite word σ and $i < j \leq |\sigma|$, $\sigma_{(i,j)} = \sigma(i), \sigma(i+1), \ldots, \sigma(j)$ denotes the subword of length $j - i + 1$ starting at index i. $\sigma_{(i,\ldots)} = \sigma(i), \sigma(i+1), \ldots$ denotes the suffix of σ starting at index i.

The truncated semantics is defined with respect to a context indicating either *weak* or *strong* strength. We use $\sigma \models^{\mathrm{s}} \varphi$ to denote that σ satisfies formula φ strongly, and $\sigma \models^{\mathrm{w}} \varphi$ to denote that σ satisfies φ weakly. We say σ *satisfies* φ, denoted by $\sigma \models \varphi$, iff σ satisfies φ strongly. Negation switches between the weak and strong contexts:

[1] Our implementation is based on the Property Specification Language PSL, defined in the IEEE standard 1850 [2]. PSL is a rich logic defined on top of the hardware description languages VHDL and Verilog, which combines temporal operators with extended regular expressions. It is straightforward to extend the approach presented in this paper with standard constructions for SEREs etc. (cf. [16]).

Definition 2 (Semantics). *A finite word σ over AP satisfies a temporal formula φ, denoted by $\sigma \models \varphi$, iff $\sigma \models^s \varphi$, where $\models^s$ and $\models^w$ are defined as follows:*

$$
\begin{aligned}
\sigma \models^s p \;&\textit{ iff }\; |\sigma| > 0 \textit{ and } p \in \sigma(0),\\
\sigma \models^s \neg\varphi \;&\textit{ iff }\; \textit{not } \sigma \models^w \varphi,\\
\sigma \models^s \varphi_1 \wedge \varphi_2 \;&\textit{ iff }\; \sigma \models^s \varphi_1 \textit{ and } \sigma \models^s \varphi_2,\\
\sigma \models^s \varphi_1 \,\mathrm{U}_{(l,u)}\, \varphi_2 \;&\textit{ iff }\; \textit{there is an } i \textit{ such that } l \le i \le u < |\sigma|\\
&\qquad \textit{and } \sigma_{(i,\dots)} \models^s \varphi_2 \textit{ and } \sigma(j,\dots) \models^s \varphi_1 \textit{ for all } l \le j < i,\\
\sigma \models^w p \;&\textit{ iff }\; |\sigma| = 0 \textit{ or } p \in \sigma(0),\\
\sigma \models^w \neg\varphi \;&\textit{ iff }\; \textit{not } \sigma \models^s \varphi,\\
\sigma \models^w \varphi_1 \wedge \varphi_2 \;&\textit{ iff }\; \sigma \models^w \varphi_1 \textit{ and } \sigma \models^w \varphi_2,\\
\sigma \models^w \varphi_1 \,\mathrm{U}_{(l,u)}\, \varphi_2 \;&\textit{ iff }\; \textit{for } u' = \min\{u, |\sigma|\},\\
&\qquad \textit{there is an } i \textit{ such that } l \le i \le u' \textit{ and } \sigma_{(i,\dots)} \models^w \varphi_2\\
&\qquad\qquad \textit{and } \sigma_{(j,\dots)} \models^w \varphi_1 \textit{ for all } l \le j < i,\\
&\qquad \textit{or } \sigma_{(k,\dots)} \models^w \varphi_1 \textit{ for all for all } l \le k \le u',
\end{aligned}
$$

where $p \in AP$ and φ_1 and φ_2 are temporal formulas.

3 Monitoring Temporal Specifications

Monitoring a specification φ means to decide for each prefix of a (possibly infinite) word over 2^{AP} whether the prefix satisfies φ.

Definition 3 (The Monitoring Problem). *Given a temporal formula φ over a set of atomic propositions AP, and a word σ over 2^{AP}, the monitoring problem consists of constructing a word σ' over $2^{\{\varphi\}}$ such that $\varphi \in \sigma'(i)$ iff $\sigma(0,i) \models \varphi$.*

A characteristic of the monitoring problem is that, since the length of the trace σ may grow beyond any bound, the space complexity of any reasonable solution must be constant in $|\sigma|$. This entails that the problem should be solved online, i.e., by reading new observations as they become available.

We now give an overview of our monitoring approach. As shown in Figure 1, our construction is split into two parts: the *suffix transducer* $\mathcal{S}$, which evaluates the bounded subformulas on the suffix of the trace, and the *prefix transducer* $\mathcal{P}$, which evaluates the complete specification on the prefix that has been seen so far. To formally describe the interface between the two transducers, we need a few auxiliary definitions.

Let φ be a temporal formula. The set of *strong subformulas* $Sub^s(\varphi)$ contains all subformulas that occur in the scope of an even number of negations (including 0). The set of *weak subformulas* $Sub^w(\varphi)$ contains all subformulas that occur in the scope of an odd number of negations. The set of *subformulas* is the union $Sub(\varphi) = Sub^s(\varphi) \cup Sub^w(\varphi)$.

For each temporal formula φ, we define the *horizon* of φ as the number of steps into the future the truth value of the formula may depend on, i.e.,

$$h(p) = 0, \qquad h(\varphi_1 \wedge \varphi_2) = \max\{h(\varphi_1), h(\varphi_2)\},$$
$$h(\neg\varphi) = h(\varphi), \qquad h(\varphi_1 \,\mathrm{U}_{(l,u)}\, \varphi_2) = \max\{u - 1 + h(\varphi_1), u + h(\varphi_2)\},$$

where $p \in AP$ and φ_1 and φ_2 are temporal formulas. A temporal formula φ is called *unbounded* if $h(\varphi) = \infty$. Otherwise, φ is called *bounded*. A formula $\psi \in \mathit{Sub}^s(\varphi)$ or $\psi \in \mathit{Sub}^w$ is a *maximal bounded* (*strong* or *weak*, respectively) subformula of φ if it is bounded and has a (strong or weak, respectively) occurrence that is not within another bounded subformula. We call the sets $\Gamma^s \subseteq \mathit{Sub}^s(\varphi)$ and $\Gamma^w \subseteq \mathit{Sub}^w(\varphi)$ of maximal bounded (strong and weak) subformulas the *separation formulas* of φ. Let $\Gamma = \Gamma^s \cup \Gamma^w$. The maximal horizon of the formulas in Γ is called the *separation horizon* H.

The separation formulas form the interface between the prefix and suffix transducers. Reading an input word σ over 2^{AP}, the *suffix transducer* computes, for each separation formula $\gamma \in \Gamma^c$ (where $c \in \{s, w\}$), each position i, and each offset $j \leq H$, the value of the additional propositions $\langle \gamma, j, c\rangle$, such that $\langle \gamma, j, c\rangle$ is true iff the truncated suffix $\sigma_{(i-H+j,i)}$ satisfies γ (strongly or weakly, depending on c). Reading an input word over $2^{AP'}$, where

$$AP' = \{\langle \gamma, j, s\rangle \mid \gamma \in \Gamma^s, 0 \leq j \leq H\} \cup \{\langle \gamma, j, w\rangle \mid \gamma \in \Gamma^w, 0 \leq j \leq H\},$$

the *prefix transducer* then treats the separation formulas as atomic propositions.

Example 1. Consider the temporal formula $\neg(\mathit{true}\ \mathrm{U}_{(0,\infty)}\ ((\neg(a\ \mathrm{U}_{(0,1)}\ b)) \wedge (\mathit{true}\ \mathrm{U}_{(0,\infty)}\ b)))$. The maximal bounded subformulas are $\neg(a\ \mathrm{U}_{(0,1)}\ b)$ and b, where $h(\neg(a\ \mathrm{U}_{(0,1)}\ b)) = 1$ and $h(b) = 0$. Hence, $H = 1$. Subformula $\neg(a\ \mathrm{U}_{(0,1)}\ b)$ is weak, b occurs both as a strong and a weak subformula, but only as a maximal weak subformula. Reading an input word over $AP = \{a, b\}$, the suffix transducer produces an output word over $AP' = \{\langle \neg(a\ \mathrm{U}_{(0,1)}\ b), 0, w\rangle, \langle \neg(a\ \mathrm{U}_{(0,1)}\ b), 1, w\rangle, \langle b, 0, w\rangle, \langle b, 1, w\rangle\}$. □

The overall monitoring problem is solved by the functional composition of the suffix and prefix transducers. The resulting transducer is implemented in hardware through a linear translation to a circuit built from flip-flops and Boolean gates. In the following sections we describe the construction of the prefix and suffix transducers and the translation to the circuit in more detail.

4 Automata and Transducers

4.1 Alternating and Universal Automata

While our constructions are based on automata transformations, our target is a circuit that monitors the given specification. For this reason we define automata in a symbolic setting that facilitates the eventual translation to a circuit: rather

than referring to an explicit alphabet, our automata are defined over the set AP of atomic propositions. We use $\overline{AP}$ to denote the set $\{a, \neg a \mid a \in AP\}$ of literals.

An *alternating automaton on finite words* over a set AP of atomic propositions is a tuple $\mathcal{A} = (Q, I, F, \delta)$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is a subset of final states, and $\delta : Q \to \mathbb{B}^+(Q \cup \overline{AP})$ is the transition condition, where $\mathbb{B}^+(X)$ denotes the set of positive Boolean expressions over X, i.e., the formulas built from elements of X using $\vee, \wedge$, *true* and *false*. An alternating automaton $\mathcal{A}$ is called *universal*, if $\delta(q)$ can be written as a conjunction where each conjunct is an element of $\mathbb{B}^+(\overline{AP} \cup \{q'\})$ for some $q' \in Q$.

The direction of evaluation in an automaton is *backward*. A run of $\mathcal{A}$ on a finite input word σ is a Q-labeled tree, such that (1) all nodes at level $|\sigma|$ (i.e., all nodes where the path from the root has length $|\sigma| + 1$) are childless and are labeled with states in F; (2) the root is labeled with q_0; and the following condition holds for every node n on some level $i = 0, \ldots, |\sigma| - 1$: let n be labeled with state q. Then the set S, consisting of the states on the children of n and the elements of $\sigma(i)$ *satisfies* $\delta(q)$, i.e., replacing every state or atomic proposition in $\delta(q)$ with *true* if it is an element of S and with *false* if it is not, results in a Boolean expression equivalent to *true*. The set of words that are accepted by $\mathcal{A}$ is called the *language of* $\mathcal{A}$, denoted by $\mathcal{L}(\mathcal{A})$.

Corresponding to an evaluation in a strong or weak context, we translate a temporal formula φ into one of two alternating automata $\mathcal{A}^s(\varphi)$ or $\mathcal{A}^w(\varphi)$: automaton $\mathcal{A}^s(\varphi)$ accepts a finite word σ iff σ satisfies φ strongly; analogously, $\mathcal{A}^w(\varphi)$ accepts σ iff σ satisfies φ weakly. As detailed in the following theorem, the translation is a simple linear-time induction:

Theorem 1. *For each temporal formula φ over AP there are two alternating automata $\mathcal{A}^s(\varphi)$ and $\mathcal{A}^w(\varphi)$ over AP such that, for every finite word σ,*

$$\sigma \models^s \varphi \text{ iff } \sigma \in \mathcal{L}(\mathcal{A}^s(\varphi)) \quad \text{and} \quad \sigma \models^w \varphi \text{ iff } \sigma \in \mathcal{L}(\mathcal{A}^w(\varphi)).$$

The sizes of $\mathcal{A}^s(\varphi)$ and $\mathcal{A}^w(\varphi)$ are linear in the size of φ. If φ is simple, then $\mathcal{A}^s(\varphi)$ and $\mathcal{A}^w(\varphi)$ are universal.

Since the context of a temporal formula is, by default, strong, we define the alternating automaton associated with a formula φ as $\mathcal{A}(\varphi) = \mathcal{A}^s(\varphi)$.

Example 2. Consider the temporal formula $\varphi = \mathrm{F}\, a \vee \mathrm{G}\, b$, which is equivalent to *true* $\mathrm{U}_{(0,\infty)}$ $a \vee \neg($*true* $\mathrm{U}_{(0,\infty)}$ $\neg b)$. The alternating automaton $\mathcal{A}(\varphi) = (\{s_0, s_1, s_2\}, s_0, \delta, F = \{s_2\})$, with $\delta : s_0 \mapsto (s_1 \vee a) \vee (s_2 \wedge b), s_1 \mapsto s_1 \vee a$, and $s_2 \mapsto s_2 \wedge b$, has three states s_0, s_1, s_2, where s_0 corresponds to φ, s_1 corresponds to $\mathrm{F}\, a$, and s_2 corresponds to $\mathrm{G}\, b$. □

Every alternating automaton can be translated into an equivalent universal automaton by a simple subset construction.

Theorem 2. *For each alternating automaton $\mathcal{A}$ there exists a universal automaton $\mathcal{U}$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{U})$. The size of $\mathcal{U}$ is exponential in the size of $\mathcal{A}$.*

4.2 Transducers

Automata evaluate the words in a backward manner: the transition expression $\delta(q)$ is a Boolean expression over the input and the *successor* states. We now change the direction of the evaluation. In order to evaluate a word in *forward* direction, a state machine is equipped with a *next-state* function τ which defines for each state q' a Boolean expression over the input and the *predecessor* states.

A *state machine* over a set AP of atomic propositions is a tuple $\mathcal{M} = (Q, Q_0, \tau)$, where Q is a set of states, $Q_0 \subseteq Q$ is a subset of initial states, and $\tau : Q \to \mathbb{B}^+(Q \cup \overline{AP})$ is the *next-state function*.

The motivation for this definition is that we wish to simulate universal automata in hardware, by representing each state as a flip-flop. The states of the state machine can thus be seen as the states of a universal automaton, and sets of states as the states of an implicit determinization.

For an input word σ, the state machine defines a run $R_0, R_1, \ldots$, where each R_i is a set of states. The run starts with the set of initial states $R_0 = Q_0$, and for all $i > 0$, the set R_i includes all states whose next-state function (with *true* substituted for all states in R_{i-1} and *false* substituted for all states not in R_{i-1} is satisfied: i.e.,

$$q' \in R_i \quad \text{iff} \quad s_i \models \tau(q')\,[q \mapsto \mathit{true} \text{ for } q \in R_{i-1} \text{ and } q \mapsto \mathit{false} \text{ for } q \notin R_{i-1}]\,.$$

For a given universal automaton $\mathcal{U} = (Q, q_0, F, \delta)$, we define the state machine $\mathcal{M} = (Q, Q_0, \tau)$ that simulates $\mathcal{U}$: the next-state function τ is chosen to precisely provide those successor states that are needed to satisfy the transition function δ:

- $Q_0 = \{q_0\}$;
- $\tau(q') = \bigvee_{\delta(q)=\ldots\wedge q'\wedge\ldots} q$.

Finally, we define *transducers*, which are state machines that are additionally equipped with an *output function*: Let $AP = AP_I \cup AP_O$ be a set of atomic propositions that is partitioned into a set AP_I of input propositions and a set AP_O of output propositions. A *transducer* $\mathcal{T} = (Q, Q_0, \tau, \{\vartheta_p\}_{p \in AP_O})$ over AP is a state machine over AP_I with an *output function* $\vartheta_p : Q \to \mathbb{B}^+(\overline{AP}_I \cup Q)$ for each $p \in AP_O$.

For an input word σ over 2^{AP_I}, the run $R_0, R_1, \ldots$ of the transducer is the run of the state machine. The transducer additionally defines an output word σ' over 2^{AP_O}, where, for all $i \geq 0$, and all $p \in AP_O$, $p \in \sigma'(i)$ iff $\sigma(i) \models \bigwedge_{q \in R_i} \vartheta_p(q)$.

5 The Suffix Transducer

We start by translating the specification into automata, using Theorems 1 and 2. Let φ be a temporal formula and let $\mathcal{A}(\varphi)$, $\mathcal{U}(\varphi)$, and $\mathcal{M}(\varphi)$ be the alternating automaton, the universal automaton, and the state machine, respectively, that are defined by φ.

When the transducer reads position i, it produces the truth values for all positions from $i - H$ to the cut-off position i. For this purpose, the suffix transducer contains a *pipeline*, which stores, for each atomic proposition p, H copies

$p^0, p^1, \ldots, p^{H-1}$, where p^j indicates the truth value of p at position $i - H + j$. Since p^H is the the value of p at the currently available position i, there is no need to store p^H in the pipeline.

Let $\pi \subseteq \bigcup_{p \in AP} \{p^0, p^1, \ldots p^{h-1}\}$ denote the pipeline content, and let $\mathcal{A}^s(\gamma) = (Q^s, q_0^s, F^s, \delta^s)$ and $\mathcal{A}^w(\gamma) = (Q^w, q_0^w, F^w, \delta^w)$ be the alternating automata for formula γ in strong and weak context, respectively. We define, for each state $q \in Q$ and each offset $j \in \{0, \ldots, H\}$, Boolean expressions $\lambda^s(\pi, q, j), \lambda^w(\pi, q, j)$ that indicate if the strong and weak automaton, respectively, starting in state q, accept the word represented by the pipeline content starting from position j. For $c \in \{s, w\}$:

$$\lambda^c(\pi, q, H) = \delta^c(q)\left[q' \mapsto \begin{cases} \mathit{true}, & \text{if } q \in F^c, \\ \mathit{false}, & \text{otherwise;} \end{cases}, q' \in Q^c\right]$$

$$\lambda^c(\pi, q, j) = \delta^c(q)\begin{bmatrix} q' \mapsto \lambda^c(\pi, q', j+1), q' \in Q^c, \\ p \mapsto \pi(p^j), p \in AP \end{bmatrix} \text{ for } j < H.$$

The truth value of the atomic proposition $\langle \gamma, j, c \rangle$ in AP' is then defined by the Boolean expression $\mu^c(\pi, \gamma, j)$, where $\mu^c(\pi, \gamma, j) = \lambda^c(\pi, q_0^c, j)$.

Example 3. The weak subformula $\psi = \neg(a \ \mathrm{U}_{(0,1)} \ b)$ from Example 1 can be translated into the alternating automaton $\mathcal{A}^w(\psi) = (\{s_0, s_1\}, s_0, \delta : s_0 \mapsto (\neg a \vee s_1) \wedge \neg b; s_1 \mapsto \neg b, F = \{s_0, s_1\})$. Since $H = 1$, the pipeline stores the values of a and b for one step (as a^0 and b^0). We obtain $\mu^w(\pi, \psi, 0) = (\neg\pi(a^0) \vee \neg b) \wedge \neg\pi(b^0)$. □

We construct the suffix transducer $\mathcal{T}(\varphi)$:

Theorem 3. *For each temporal formula φ with separation formulas Γ^s, Γ^w, there exists a transducer $\mathcal{T}(\varphi)$ with input propositions AP and output propositions AP', such that the following holds for each $\langle \gamma, j, c \rangle \in AP'$, $j \in \{0, \ldots, H\}$, $i \geq H - j$, and each input word σ and output word $O_0, O_1, \ldots$:*

$$\langle \gamma, j, c \rangle \in O_i \quad \textit{iff} \quad \sigma(i - H + j, i) \models^c \gamma.$$

The set of states is formed by the possible pipeline contents. The transition function shifts the contents of the pipeline by one position and adds the new observation. The output interprets each atomic proposition $\langle \gamma, j, c \rangle$ in AP' as $\mu^c(\pi, \gamma, j)$.

6 The Prefix Transducer

The prefix transducer computes the truth value of the specification φ based on the extended trace provided by the suffix transducer. For this purpose, the separation formulas in the specification are replaced by atomic propositions. To ensure that the substitution respects the context, we introduce, in addition to the

standard substitution operator $\varphi[\psi \mapsto \psi']$, which replaces every occurrence of ψ in φ with ψ', a strong and a weak version: In the *strong substitution* $\varphi[\psi \mapsto \psi']^s$, all occurrences of ψ that are in the scope of an even number of negations are replaced by ψ', in the *weak substitution* $\varphi[\psi \mapsto \psi']^s$, all occurrences of ψ that are in the scope of an odd number of negations are replaced by ψ'. We generalize the substitution operators to sets of replacement pairs in the obvious way.

Let φ be a temporal formula. The prefix transducer is based on a simplified *prefix formula* φ^p, where we replace every separation formula with a proposition from $\Gamma^s \times \{s\} \ \cup \ \Gamma^w \times \{w\}$, i.e., with a proposition indicating the separation formula together with the strong or weak context.

$$\varphi^p = \varphi[\gamma \mapsto \langle \gamma, s\rangle \mid \gamma \in \Gamma^s][\gamma \mapsto \langle \gamma, w\rangle \mid \gamma \in \Gamma^w].$$

Example 4. Consider again the specification from Example 1:

$$\varphi \ = \neg(\mathit{true}\ \mathrm{U}_{(0,\infty)}\ ((\neg(a\ \mathrm{U}_{(0,1)}\ b)) \wedge (\mathit{true}\ \mathrm{U}_{(0,\infty)}\ b))).$$

$$\text{Hence, } \ \varphi^p = \neg(\mathit{true}\ \mathrm{U}_{(0,\infty)}\ (\langle\neg(a\ \mathrm{U}_{(0,1)}\ b), w\rangle \wedge (\mathit{true}\ \mathrm{U}_{(0,\infty)}\ \langle b, w\rangle))).$$

□

The idea for the construction of the prefix transducer is to check for the existence of a run of the universal automaton $\mathcal{U}(\varphi^p)$ on the prefix up to position i. Intuitively, the prefix is split into two parts. The first part, up to position $(i - H)$, is handled by the state machine $\mathcal{M}(\varphi^p)$, which we run with a delay of H steps. In the transition function of the state machine, we therefore replace every proposition $\langle \gamma, c\rangle$ with the proposition $\langle \gamma, 0, c\rangle$ delivered by the suffix automaton.

The second part, from position $(i - H)$ to position i, is handled by the output function of the transducer. For this purpose, we unroll the transition function of $\mathcal{U}(\varphi^p)$ for H steps, and accordingly replace, in the jth unrolling, the proposition $\langle \gamma, c\rangle$ with the proposition $\langle \gamma, j, c\rangle$ provided by the suffix automaton. Let $\mathcal{U}(\varphi^p) = (Q, q_0, \delta, F)$. We define inductively:

$$\begin{aligned}
\nu(q, H) &= \delta(q)\,[\langle \gamma, c\rangle \mapsto \langle \gamma, H, c\rangle \mid \gamma \in \Gamma, c \in \{s, w\}] \\
&\qquad \left[q' \mapsto \begin{cases} \mathit{true}, & \text{if } q \in F, \\ \mathit{false}, & \text{otherwise;} \end{cases} \;\middle|\; q' \in Q \right]; \\
\nu(q, j) &= \delta(q)\,[\langle \gamma, c\rangle \mapsto \langle \gamma, j, c\rangle \mid \gamma \in \Gamma, c \in \{s, w\}] \\
&\qquad [q' \mapsto \nu(q', j+1) \mid q' \in Q] \qquad\qquad \text{for } j < H.
\end{aligned}$$

Suppose the state machine has computed the state set S when reaching its delayed position $(i-H)$. Then this partial run can be completed into an accepting run on the full prefix iff $\nu(q, 0)$ is *true* for all states $q \in S$.

The prefix transducer $\mathcal{P}$ with input propositions AP' and output propositions AP'' is obtained from the state machine $\mathcal{M}(\varphi^p)$ by encoding the delay of H steps. For this purpose, the transducer starts by counting H steps. In the i$^{\text{th}}$ step the output is $\{\varphi\}$ if $\nu(q, H-i)$ is true for all intial states of $\mathcal{M}(\varphi^p)$. Then it proceeds with the initial states of $\mathcal{M}(\varphi^p)$. The output is $\{\varphi\}$ if the $\nu(q, 0)$ is true for all active states.

Theorem 4. *For each temporal formula φ with separation formulas Γ^s, Γ^w, there exists a transducer $\mathcal{P}(\varphi)$ with input propositions AP' and output propositions $AP_O = \{\varphi\}$, such that for all words σ over 2^{AP}, σ' over $2^{AP'}$, and σ'' over $2^{AP''}$, if $\mathcal{T}(\varphi)$ produces output σ' reading input σ, and $\mathcal{P}(\varphi)$ produces output σ'' reading input σ', then for all $i \leq |\sigma|$, $\varphi \in \sigma''(i)$ iff $\sigma(0,i) \models \varphi$.*

7 The Monitor Circuit

As shown in Figure 1, the monitor circuit is built from four main components: the pipeline circuit for the the suffix transducer $\mathcal{S}$, the output function of the suffix transducer, the state machine of the prefix transducer $\mathcal{P}$, and the output function of the prefix transducer. The circuits for the pipeline and the prefix state machine maintain their internal state via D flip-flops, interconnected via Boolean circuits that implement the next-state function. The circuits for the output functions are pure Boolean circuits without internal state.

The pipeline circuit. The states of the suffix transducer $\mathcal{S}$ are defined by the pipeline that buffers the truth values of the atomic propositions. For each atomic proposition $p \in AP$ and each offset $j, 0 \leq j < H$, the pipeline contains a D flip-flop $f_{p,j}$. The input to $f_{p,H-1}$ is the current input signal for p. The output of $f_{p,j}$ is connected to the input of $f_{p,j-1}$, thus shifting the values of p in each clock-cycle by one position.

The state machine of the prefix transducer. Each state q of the state machine of the prefix transducer $\mathcal{P}$ is implemented by a D flip-flop. The next-state function is translated into Boolean circuits that are connected to the outputs of the flip-flops representing the states and the output gates of the circuit for the output functions of the suffix transducer $\mathcal{S}$.

The output functions of the transducers. The output functions of the transducers are implemented in hardware as pure Boolean circuits. The input gates of the circuit for the output function of the suffix transducer $\mathcal{S}$ are connected to the output of the flip-flops of the state machine of the suffix transducer $\mathcal{S}$ and with the signals of the atomic propositions in AP.

The input gates of the output function of the prefix transducer $\mathcal{P}$ are connected to the outputs of the flip-flops for the state machine of $\mathcal{P}$ and the output gates of the output function of $\mathcal{S}$. Its single output gate represents the output of the monitor for φ on the prefix of the current input.

This implementation of the monitor circuit is well-suited for reprogrammable hardware such as FPGAs. The actual translation of the Boolean functions into the specific hardware can be realized by standard tools for the computer-aided design of digital circuits.

The size of the circuits. The size of the pipeline circuit is linear in $H \cdot |\varphi|$. The output circuit of $\mathcal{S}$ consists of sub-circuits for each separation formula and

each position within the delayed fragment of the input trace. Each of these subcircuits is linear in the size of H and linear in the size of φ. Hence, the overall size for the output circuit is quadratic in $H \cdot |\varphi|$.

The size of the circuit for the state machine of $\mathcal{P}$ is linear in $|\mathcal{U}(\varphi)|$ and hence linear in $|\varphi|$ if φ is simple and exponential in $|\varphi|$ otherwise. The size of the Boolean circuit that computes the output function of $\mathcal{P}$ is of the same order as the state machine of $\mathcal{P}$ multiplied by H.

Theorem 5. *The number of gates of the monitoring circuit for a temporal specification φ is quadratic in $H \cdot |\varphi|$ if φ is simple except for bounded subformulas; otherwise, the number of gates is exponential in $|\varphi|$.*

8 Experimental Results

Our implementation takes as input an LTL formula and produces synthesizable VHDL code for a cirucit that monitors the input formula. The code is then passed to a synthesis tool for a specific hardware platform. In this section we report on experimental results obtained with our implementation in conjunction with the Xilinx Virtex-5 FPGA synthesis tool.

Our benchmarks, shown in Figure 2, include Etessami and Holzmann's list of commonly used LTL specifications [24] (formulas 1–12, adapted to our setting by the introduction of parametric bounds), as well as a variation of the cache specification from the introduction (formulas c_n). The formulas r_n specify fair bounded response, a recurring pattern in many specifications. Table 1 shows the results for the formuals from Figure 2. The number of signals and the number of flip-flops are with respect to the VHDL description of the monitor circuit. The MHz values are computed by the Xilinx Virtex-5 tool.

The first two sections of the table compare, for formulas with bound 2, the performance of our construction ($b = 2$) with a direct approach ($b = 2$ direct), based on building a universal automaton without pipeline. The presence of already very moderate bounds or a small number of nested Next-modalities can yield a direct universalization of the alternating automaton of the specification infeasable. As long as the bounds (or Next-modalities) are properly nested within the unbounded operators, our construction circumvents the combinatorial blow-up of the forward universalization and produces rather small monitors. Even for bounds up to several dozens or even hundreds of steps, our approach produces monitor circuits that fit on standard FPGAs. Bounds of this size are clearly far beyond what a forward universalization can handle with an reasonable amount of computational resources.

Our results also provide evidence that introducing bounds into a given specification can be helpful in order to simplify the monitoring. The third and fourth section of the table report on the performance for higher bounds ($b = 40$) and unbounded ($b = \infty$) formulas. For small specifications, bounds complicate the monitoring problem. For larger specifications, however, the combinatorial overhead introduced by the bounds, which is just polynomial, outplays the exponential blow-up caused by the richer combinatorial structure of the specification.

1. $p\,\mathrm{U}_{(0,3b)}\,(q \wedge\ \mathrm{G}_{(0,3b)}\,r)$
2. $p\,\mathrm{U}(q \wedge \mathrm{X}(r\,\mathrm{U}_{(0,2b)}\,s))$
3. $p\,\mathrm{U}(q \wedge \mathrm{X}(r \wedge\ \mathrm{F}_{(0,2b)}\,(s \wedge \mathrm{X}\ \mathrm{F}_{(0,2b)}\,(t \wedge \mathrm{X}\ \mathrm{F}_{(0,2b)}\,(u \wedge \mathrm{X}\ \mathrm{F}_{(0,2b)}\,v)))))$
4. $\mathrm{F}_{(0,3b)}\,(p \wedge \mathrm{X}\ \mathrm{G}_{(0,2b)}\,q)$
5. $\mathrm{F}(p \wedge \mathrm{X}(q \wedge \mathrm{X}(\mathrm{F}_{(0,b)}\,r)))$
6. $\mathrm{F}(q \wedge \mathrm{X}(p\,\mathrm{U}_{(0,2b)}\,r))$
7. $(\mathrm{F}\ \mathrm{G}_{(0,3b)}\,q) \vee (\mathrm{F}\ \mathrm{G}_{(0,3b)}\,p)$
8. $\mathrm{G}(\neg p \vee (q\,\mathrm{U}_{(0,b)}\,r))$
9. $\mathrm{F}(p \wedge \mathrm{X}\ \mathrm{F}_{(0,b)}\,(q \wedge \mathrm{X}\ \mathrm{F}_{(0,b)}\,(r \wedge \mathrm{X}\ \mathrm{F}_{(0,b)}\,s)))$
10. $(\mathrm{G}\ \mathrm{F}_{(0,2b)}\,p) \wedge (\mathrm{G}\ \mathrm{F}_{(0,2b)}\,q) \wedge (\mathrm{G}\ \mathrm{F}_{(0,2b)}\,r) \wedge (\mathrm{G}\ \mathrm{F}_{(0,2b)}\,s) \wedge (\mathrm{G}\ \mathrm{F}_{(0,2b)}\,t)$
11. $(p\,\mathrm{U}_{(0,2b)}\,q\,\mathrm{U}_{(0,2b)}\,r) \vee (q\,\mathrm{U}_{(0,2b)}\,r\,\mathrm{U}_{(0,2b)}\,p) \vee (r\,\mathrm{U}_{(0,2b)}\,p\,\mathrm{U}_{(0,2b)}\,q)$
12. $\mathrm{G}(\neg p \vee (q\,\mathrm{U}_{(0,b)}\,((\,\mathrm{G}_{(0,3b)}\,r) \vee (\,\mathrm{G}_{(0,3b)}\,s))))$

$$c_n = \mathrm{F}(\neg x \wedge \bigwedge_{0 \le i \le n} (v_i \leftrightarrow\ \mathrm{F}_{(0,b)}\,(v_i \wedge x))) \wedge \mathrm{G}(x \rightarrow \mathrm{X}\,\mathrm{G}\,\neg x) \wedge \mathrm{F}\,x$$

$$r_n = \mathrm{G}((\bigwedge_{0 \le i < n} (\mathrm{G}\ \mathrm{F}^{\forall}_{(0,3b)}\,f_i)) \rightarrow (a \rightarrow (\,\mathrm{F}^{\forall}_{(0,b)}\,b)))$$

Fig. 2. Benchmark formulas.

	$b=2$			$b=2$ direct			$b=40$			$b=\infty$		
	#signals	#ffs	MHz	#signals	#ffs	MHz	#signals	#ffs	MHz	#signals	#ffs	MHz
1.	165	7	358	5274	192	309	43876	159	169	81	6	468
2.	86	9	442	1367	66	338	997	199	145	141	10	468
3.	545	12	204	–	–	–	89339	316	75	5792	97	383
4.	109	6	463	1705	128	336	19614	120	215	52	6	468
5.	41	7	467	168	18	468	532	159	253	97	10	468
6.	53	7	440	568	34	432	661	159	143	65	6	468
7.	70	6	467	–	–	–	7299	120	203	70	6	468
8.	39	8	467	52	6	468	344	160	191	47	5	468
9.	114	8	357	10746	514	297	10893	198	123	418	24	468
10.	180	14	467	173	28	449	7224	242	184	95	12	468
11.	279	7	467	–	–	–	78547	159	467	1240	66	367
12.	134	9	373	3984	231	301	22888	199	132	193	11	468
c_3	202	13	431	8702	88	258	3101	203	188	35273	263	256
c_5	259	15	314	27232	176	336	4966	281	179	–	–	–
c_7	316	17	295	63080	296	289	5821	359	173	–	–	–
c_{10}	405	21	274	–	–	–	8042	515	169	–	–	–
r_3	180	10	382	–	–	–	5555	238	241	308	11	468
r_5	280	12	402	–	–	–	7607	316	228	9368	62	361
r_{10}	475	17	367	–	–	–	12646	511	214	–	–	–

Table 1. Experimental results.

9 Conclusions

We have presented a novel translation of temporal specifications to monitor circuits that saves exponential space in the size of bounded-future subformulas. The monitors are useful in hardware solutions for high-performance simulation and prototyping, in particular using reprogrammable hardware such as FPGAs. The new construction should allow synthesis tools for PSL to treat at least bounded specifications outside the simple subset. The free use of disjunctions in temporal specifications will make the specifications shorter and more general and will allow the user to use the same specification logic for static verification and runtime verification, without an error-prone rewrite into a restricted sublogic.

References

1. Kupferman, O., Vardi, M.: Model checking of safety properties. In: Computer Aided Verification, Proc. 11th Int. Conference. Volume 1633 of Lecture Notes in Computer Science., Springer-Verlag (1999) 172–183
2. IEEE Std 1850-2007: Standard for Property Specification Language (PSL). IEEE, New York. (2007)
3. Cohen, B., Venkataramanan, S., Kumari, A.: Using PSL/Sugar for Formal and Dynamic Verification. VhdlCohen Publishing, Los Angeles (2004)
4. Kleene, S.: Representation of events in nerve nets and finite automata. In: Automata Studies. Princeton University Press (1956)
5. McNaughton, R., Papert, S.: Counter-Free Automata. Volume 65 of Research Monograph. MIT Press (1971)
6. Eisner, C., Fisman, D., Havlicek, J., Lustig, Y., McIsaac, A., Campenhout, D.V.: Reasoning with temporal logic on truncated paths. In: CAV'03. Volume 2725 of LNCS., Springer (2003) 27–39
7. Finkbeiner, B., Sipma, H.B.: Checking finite traces using alternating automata. In Havelund, K., Roşu, G., eds.: Runtime Verification 2001. Volume 55 of Electronic Notes in Theoretical Computer Science., Elsevier (July 2001) 44–60
8. Geilen, M.: On the construction of monitors for temporal logic properties. Electr. Notes Theor. Comput. Sci. **55**(2) (2001)
9. Giannakopoulou, D., Havelund, K.: Automata-based verification of temporal properties on running programs. In: ASE, IEEE Computer Society (2001) 412 – 416
10. Havelund, K., Rosu, G.: Monitoring programs using rewriting. In: ASE, IEEE Computer Society (2001) 135 – 143
11. Bauer, A., Leucker, M., Schallhart, C.: The good, the bad, and the ugly, but how ugly is ugly? In Sokolsky, O., Tasiran, S., eds.: RV. Volume 4839 of Lecture Notes in Computer Science., Springer (2007) 126 – 138
12. Dahan, A., Geist, D., Gluhovsky, L., Pidan, D., Shapir, G., Wolfsthal, Y., Benalycherif, L., Kamdem, R., Lahbib, Y.: Combining system level modeling with assertion based verification. In: ISQED05, IEEE Comp. Soc. (2005) 310–315
13. Boule, M., Zilic, Z.: Automata-based assertion-checker synthesis of PSL properties. ACM Transactions on Design Automation of Electronic Systems (TODAES) **13**(1) (2008)
14. Armoni, R., Korchemny, D., Tiemeyer, A., Vardi, M.Y., Zbar, Y.: Deterministic dynamic monitors for linear-time assertions. In: In Proc. Workshop on Formal Approaches to Testing and Runtime Verification 2006. Volume 4262 of Lecture Notes in Computer Science., Springer (2006)

15. Ruah, S., Fisman, D., Ben-David, S.: Automata construction for on-the-fly model checking PSL safety simple subset. Technical report, IBM Research (2005)
16. Ben-David, S., Bloem, R., Fisman, D., Griesmayer, A., Pill, I., Ruah, S.: Automata construction algorithm optimized for PSL. Technical report, PROSYD (July 2005)
17. Pnueli, A., Zaks, A.: PSL model checking and run-time verification via testers. In: FM. Volume 4085 of LNCS., Springer (2006) 573–586
18. Cimatti, A., Roveri, M., Semprini, S., Tonetta, S.: From PSL to NBA: a modular symbolic encoding. In: FMCAD '06: Proceedings of the Formal Methods in Computer Aided Design, Washington, DC, USA, IEEE Computer Society (2006) 125–133
19. Ben-David, S., Fisman, D., Ruah, S.: The safety simple subset. In Ur, S., Bin, E., Wolfsthal, Y., eds.: Haifa Verification Conference. Volume 3875 of LNCS., Springer (2005) 14–29
20. d'Amorim, M., Rosu, G.: Efficient monitoring of omega-languages. In Etessami, K., Rajamani, S., eds.: CAV. Volume 3576 of Lecture Notes in Computer Science., Springer (2005) 364 – 378
21. Brzozowski, J., Simon, I.: Characterizations of locally testable events. Discrete Math. **4** (1973) 243–271
22. Wike, T.: Locally threshold testable languages of infinite words. In: STACS'93. LNCS, Springer (1993) 607–616
23. Kupferman, O., Lustig, Y., Vardi, M.: On locally checkable properties. In: Proc. 13th International Conference on Logic for Programming Artificial Intelligence and Reasoning. Lecture Notes in Computer Science, Springer-Verlag (2006)
24. Etessami, K., Holzmann, G.: Optimizing Büchi automata. In: CONCUR'00. LNCS, Springer (2000) 153–167