Impartial Anticipation in Runtime-Verification*

Wei Dong[1], Martin Leucker[2], and Christian Schallhart[3]

[1] School of Computer, National University of Defense Technology, P.R. China
[2] Institut für Informatik, Technische Universität München, Germany
[3] Formal Methods in Systems Engineering, FB Informatik, TU Darmstadt, Germany

Abstract. In this paper, a uniform approach for synthesizing monitors checking correctness properties specified in linear-time logics at runtime is provided. Therefore, a generic three-valued semantics is introduced reflecting the idea that *prefixes* of infinite computations are checked. Then a conceptual framework to synthesize monitors from a logical specification to check an execution incrementally is established, with special focus on resorting to the *automata-theoretic approach*. The merits of the presented framework are shown by providing monitor synthesis approaches for a variety of different logics such as **LTL**, the linear-time μ-calculus, **PLTL**$^{\text{mod}}$, **S1S**, and **RLTL**.

1 Introduction

Runtime verification (RV) is an emerging lightweight verification technique in which executions of systems under scrutiny are checked for satisfaction or violation of given correctness properties. While it complements verification techniques such as model checking and testing, it also paves the way for not-only detecting incorrect behavior of a software system but also for reacting and potentially *healing* the system when a correctness violation is encountered.

Typically, a *monitor* is employed in RV, checking whether the execution meets a certain correctness property. Such a monitor may on one hand be used to check the *current* execution of a system. In this setting, which is termed *online monitoring*, the monitor should be designed to consider executions in an *incremental fashion* and in an *efficient manner*. On the other hand, a monitor may work on a (finite set of) *recorded* execution(s), in which case we speak of *offline monitoring*. In this paper we focus on online monitoring.

In online monitoring, often an—at least ideally—non-terminating system is checked. In the very end, this asks for checking correctness of an infinite execution trace. Clearly, this cannot be done at runtime. In fact, we aim at deriving a verdict whether an infinite execution satisfies a correctness property by considering its *finite prefixes*. In [1], we formulated two maxims a monitor should ideally follow to capture implications of this idea:

* This work has been supported in part by National Natural Science Foundation of China (Grant No.60673118).

– *Impartiality* requires that a finite trace is not evaluated to *true* or *false*, if there still exists an (infinite) continuation leading to another verdict.
– *Anticipation* requires that once every (infinite) continuation of a finite trace leads to the same verdict, then the finite trace evaluates to this verdict.

Intuitively, the first maxim postulates that a monitor only decides for *false*—meaning that a misbehavior has been observed—or *true*—meaning that the current behavior fulfills the correctness property, regardless of how it continues—only if this is indeed the case. Clearly, this maxim requires to have at least three different truth values: *true*, *false*, and *inconclusive*, but of course more than three truth values might give a more precise assessment of correctness. The second maxim requires a monitor to indeed report *true* or *false*, if the correctness property is indeed violated or satisfied.

Typically, monitors are generated automatically from some high-level specification. Runtime verification, which has its roots in model checking, often employs some variant of linear temporal logic, such as Pnueli's **LTL** [2]. However, typically these logics and corresponding verification algorithms are considered on infinite executions. To follow the ideas of impartiality and anticipation, we defined in [3] a three-valued semantics for **LTL** obtaining the logic $\mathbf{LTL}_3$. Moreover, in [3,4], we presented a monitor synthesis algorithm for $\mathbf{LTL}_3$. Using similar ideas, we also introduced a three-valued semantics for a real-time version of **LTL** and provided a corresponding monitor synthesis algorithm.

However, there is large variety of linear-time logics, for which monitor synthesis algorithms are of interest. In this paper, a uniform approach for synthesizing monitors checking correctness properties specified in linear-time logics at runtime is provided, which is based on our approach in [3]. To this end, we define *linear-time logics* as logics interpreted over infinite words, for example, **LTL**, $\mathbf{PLTL}^{\text{mod}}$ [5], linear μ-calculus, timed **LTL** etc. Uniformly, we give an impartial and anticipatory semantics for linear-time logics suitable for runtime verification. We identify key *decision* and *abstraction* functions from which a monitor for a formula of the respective logic is directly obtained.

Satisfiability and *model checking* are common problems addressed for logics. A pattern emerging for solutions of these problems is the so-called *automata-theoretic approach*: In satisfiability checking, it means to construct for a given formula ϕ the automaton $\mathcal{A}_\phi$ accepting (abstractions of) words satisfying ϕ, so that the language of $\mathcal{A}_\phi$ is non-empty iff ϕ is satisfiable. Model checking, though, is often reduced to constructing the automaton $\mathcal{A}_{\neg\phi}$ accepting the *counter examples* of ϕ and checking the intersection of a model and $\mathcal{A}_{\neg\phi}$ for emptiness. Also for the exemplifying linear-time logics the automata-theoretic approach for checking satisfiability has been studied and it seems beneficial to reuse such automata constructions when looking for monitors—provided this is possible. We define precisely the automata-theoretic approach and we elaborate certain criteria (*forgettable past* and *faithfulness of abstraction*) under which a monitor is directly obtained from automata accepting the models of a formula at hand. We show that automata constructions existing in the literature for several linear-time logics satisfy the introduced criteria such that we derive easily impartial and anticipating monitors, e.g., for the linear-time μ-calculus [6,7], monadic second-order logic (over words) [8], **RLTL** [9], and $\mathbf{PLTL}^{\mathrm{mod}}$ [5] which is interpreted over sequences of integer valuations.

2 Anticipatory Monitors

Definition 1 (Linear-time Logic). *A* linear-time logic L *defines a set* F_L *of* L-formulae *and a two-valued* semantics $\models_L$. *Every* L*-formula* $\phi \in F_L$ *has an associated and possibly infinite* alphabet Σ_ϕ. *For every formula* $\phi \in F_L$ *and every word* $\sigma \in \Sigma_\phi^\omega$*, we require the semantics to be well-defined, i.e., either* $\sigma \models_L \phi$ *or* $\sigma \not\models_L \phi$ *must hold.*

Furthermore, we require a linear-time logic L to satisfy the following properties:

(L1) $\forall \phi \in F_L \;:\; \neg\phi \in F_L$. Note that this property does *not* require that negation is applicable to every *subformula* of ϕ.
(L2) $\forall \sigma \in \Sigma_\phi^\omega \;:\; (\sigma \models_L \phi \;\Leftrightarrow\; \sigma \not\models_L \neg\phi)$. Note that $\neg\phi \in F_L$ must hold because of property **(L1)**.

Definition 2 (Anticipation Semantics). *If* L *is a logic following Definition 1, then we define the* anticipation semantics $[\pi \models \phi]_L$ *of an* L*-formula* $\phi \in F_L$ *and a finite word* $\pi \in \Sigma_\phi^*$ *with*

$$[\pi \models \phi]_L = \begin{cases} \top & \text{if } \forall \sigma \in \Sigma_\phi^\omega \;:\; \pi\sigma \models_L \phi \\ \bot & \text{if } \forall \sigma \in \Sigma_\phi^\omega \;:\; \pi\sigma \not\models_L \phi \\ ? & \text{otherwise} \end{cases}$$

Note that the definition of anticipation semantics fulfills both, the *impartiality* and *anticipation* requirements stated in the introduction: It is impartial since for every prefix $\pi \in \Sigma^*$ with two continuations $\sigma, \sigma' \in \Sigma^\omega$ such that $\pi\sigma \models_L \phi$ and $\pi\sigma' \not\models_L \phi$ hold, the semantics $[\pi \models \phi]_L$ evaluates to the inconclusive verdict ?. On the other hand, once only satisfying or unsatisfying continuations exist, the semantics $[\pi \models \phi]_L$ evaluates to the corresponding verdict $\top$ or $\bot$.

Since we want to use the anticipation semantics in runtime verification, we have to develop a monitor procedure $\mathsf{monitor}_\phi(a)$ which reads a trace incrementally: It takes a symbol a in each invocation and returns thereupon the valuation of the currently processed prefix. To this end, we evaluate the core question arising in the anticipation semantics $\forall \sigma \in \Sigma_\phi^\omega \;:\; \pi\sigma \models_L \phi$ using the equivalence

$$\forall \sigma \in \Sigma_\phi^\omega \;:\; \pi\sigma \models_L \phi \;\Leftrightarrow\; \nexists \sigma \in \Sigma_\phi^\omega \;:\; \pi\sigma \not\models_L \phi \;\Leftrightarrow\; \nexists \sigma \in \Sigma_\phi^\omega \;:\; \pi\sigma \models_L \neg\phi$$

which holds for every logic L which satisfies property **(L2)**. By handling the complemented case analogously, we obtain the following rule to evaluate the anticipation semantics:

$$[\pi \models \phi]_L = \begin{cases} \top & \text{if } \mathsf{decide}_{\neg\phi}(\pi) = \bot \\ \bot & \text{if } \mathsf{decide}_{\phi}(\pi) = \bot \\ ? & \text{otherwise} \end{cases}$$

where $\mathsf{decide}_\phi(\pi)$ is defined to return $\top$ for $\phi \in F_L$ and $\pi \in \Sigma_\phi$ if $\exists \sigma \in \Sigma_\phi^\omega : \pi\sigma \models_L \phi$ holds, and $\bot$ otherwise. Note that $\mathsf{decide}_\phi(\pi)$ is well-defined since ϕ and $\neg\phi$ are both in F_L. Observe that a computable anticipatory semantics requires the satisfiability problem of the underlying logic to be decidable.

Remark 1. A linear-time logic has a computable anticipatory semantics, only if the satisfiability problem for the logic is decidable.

In order to give an *incrementally* working monitor procedure, we have to avoid reevaluating the entire prefix π in $\mathsf{decide}_\phi(\pi)$ whenever a symbol is read. Instead, we want to use an *automaton construction* to compute $\mathsf{decide}_\phi(\pi a)$ for $\pi \in \Sigma_\phi^*$ and $a \in \Sigma_\phi$ after having already processed π. Hence,we introduce a procedure $\mathsf{step}_\phi(S, a)$ which takes a set $S \subseteq \mathcal{S}_\phi$ of states and a symbol $a \in \Sigma_\phi$ and returns a new set $S' \subseteq \mathcal{S}_\phi$. By executing step_ϕ stepwise on a finite prefix $\pi = a_1 \dots a_n$ we obtain the *automaton abstraction* $\alpha_\phi(\pi)$ of π with

$$\alpha_\phi(\pi) = \mathsf{step}_\phi(\dots(\mathsf{step}_\phi(\mathsf{step}_\phi(I_\phi, a_1), a_2), \dots), a_n)$$

where I_ϕ is an initial set of states for ϕ. Then we apply a suitably defined procedure check_ϕ on the resulting set of states $\alpha_\phi(\pi)$ to obtain $\mathsf{check}_\phi(\alpha_\phi(\pi)) = \mathsf{decide}_\phi(\pi)$. We summarize these terms in the following definition:

Definition 3 (Automaton Construction with Emptiness Check). *A logic L has an* automaton construction with emptiness check *if we have for every formula $\phi \in F_L$, (a) a finite set $\mathcal{S}_\phi$ of states, (b) a set $I_\phi \subseteq \mathcal{S}_\phi$ of initial states, (c) a transition function* $\mathsf{step}_\phi(S, a)$ *which maps a set of states $S \subseteq \mathcal{S}$ and a symbol $a \in \Sigma_\phi$ to a new set $S' \subseteq \mathcal{S}$ of states, and (d) a function* $\mathsf{check}_\phi(S)$ *with* $\mathsf{check}_\phi(\alpha_\phi(\pi)) = \mathsf{decide}_\phi(\pi)$ *for all $\pi \in \Sigma_\phi^*$ where we define* automaton abstraction α_ϕ *with $\alpha_\phi(\epsilon) = I_\phi$ and recursively with* $\alpha_\phi(\pi a) = \mathsf{step}_\phi(\alpha_\phi(\pi), a)$ *for all $\pi \in \Sigma_\phi^*$ and $a \in \Sigma_\phi$.*

If a logic L has an automaton construction with emptiness check, we use the following rule to evaluate the anticipation semantics

$$[\pi \models \phi]_L = \begin{cases} \top & \text{if } \mathsf{check}_{\neg\phi}(\alpha_{\neg\phi}(\pi)) = \bot \\ \bot & \text{if } \mathsf{check}_\phi(\alpha_\phi(\pi)) = \bot \\ ? & \text{otherwise} \end{cases}$$

which leads to the procedure $\mathsf{monitor}_\phi(a)$ as shown in Figure 1.

```
procedure monitor_φ(a)
init
  $S_\phi$ := $I_\phi$; $S_{\neg\phi}$ := $I_{\neg\phi}$
begin
  $S_\phi$ := $\mathsf{step}_\phi(S_\phi, a)$;
  $S_{\neg\phi}$ := $\mathsf{step}_{\neg\phi}(S_{\neg\phi}, a)$;
  if $\mathsf{check}_\phi(S_\phi) = \bot$
    then return $\bot$;
  if $\mathsf{check}_{\neg\phi}(S_{\neg\phi}) = \bot$
    then return $\top$;
  return ?;
end
```

Fig. 1. Procedure $\mathsf{monitor}_\phi(a)$

As an example for a logic which directly yields an automaton abstraction with emptiness check, consider **LTL** [2]. The set of **LTL** formulae is inductively defined by the grammar $\phi ::= true \mid p \mid \neg\phi \mid \phi \vee \phi \mid \phi\ U\ \phi \mid X\phi$. Recall that a *(nondeterministic) Büchi automaton (NBA)* is a tuple $\mathcal{A} = (\Sigma, Q, Q_0, \delta, F)$, where Σ is a finite alphabet, Q is a finite, non-empty set of states, $Q_0 \subseteq Q$ is a set of initial states, $\delta : Q \times \Sigma \to 2^Q$ is the transition function, and $F \subseteq Q$ is a set of accepting states. For an NBA $\mathcal{A}$, we denote by $\mathcal{A}(q)$ the NBA that coincides with $\mathcal{A}$ except for the set of initial state Q_0, which is redefined in $\mathcal{A}(q)$ as $Q_0 = \{q\}$.

Let $\mathcal{A}^\phi = (\Sigma, Q^\phi, Q_0^\phi, \delta^\phi, F^\phi)$ denote the NBA which accepts all models of the **LTL**-formula ϕ, and let $\mathcal{A}^{\neg\phi} = (\Sigma, Q^{\neg\phi}, Q_0^{\neg\phi}, \delta^{\neg\phi}, F^{\neg\phi})$ denote the NBA which accepts all words falsifying ϕ. The corresponding construction is standard [10]. Now we define step and check for **LTL** as follows:

- $\mathsf{step}_\phi : 2^{Q^\phi} \times \Sigma \to 2^{Q^\phi}$ does the Büchi automaton steps of $\mathcal{A}^\phi$. That is, $\mathsf{step}_\phi(S, a) = \bigcup_{q' \in S} \delta^\phi(q', a)$ for $S \subseteq Q^\phi$ and $a \in \Sigma$. Analogously, $\mathsf{step}_{\neg\phi} : 2^{Q^{\neg\phi}} \times \Sigma \to 2^{Q^{\neg\phi}}$ is defined based on $\mathcal{A}^{\neg\phi}$.
- $\mathsf{check}_\phi : 2^{Q^\phi} \to \{\top, \bot\}$ does the emptiness check for the states. That is, $\mathsf{check}_\phi(S) = \top$ iff $\bigcup_{q' \in S} \mathcal{L}(\mathcal{A}^\phi(q')) \neq \emptyset$ for $S \subseteq Q^\phi$, otherwise $\mathsf{check}_\phi(S) = \bot$. Analogously, $\mathsf{check}_{\neg\phi} : 2^{Q^{\neg\phi}} \to \{\top, \bot\}$ is defined in terms of $\mathcal{A}^{\neg\phi}$.

Note that we essentially get the monitor procedure established in [3].

3 Monitors Via the *Automata-Theoretic Approach*

So far, we have understood that there is a canonical anticipatory semantics well-suited in runtime verification for a wide range of linear-time logics, which is based on the function decide_ϕ. Thus, for any linear-time logic, a (non-incremental) RV semantics can be computed, provided that for each formula ϕ of the logic a computable decide_ϕ can be constructed. Moreover, a monitor construction for checking an input sequence incrementally was developed for linear-time logics, provided an *automaton abstraction* is given.

In consequence, runtime verification support for a linear-time logic is reduced to providing the corresponding functions/abstractions. We have shown that this task is simple in the setting of Pnueli's **LTL**. However, for some relevant richer logics, like real-time logics [11] or logics interpreted over the integers [5], this task is considerably more involved [4]. Nevertheless, as the approach given for **LTL** suggests, there is a typical *pattern* for deriving those functions, which we describe in this section. In simple words, we show that whenever for the underlying logic the so-called (a) *automata-theoretic approach* to satisfiability checking is followed—as it is the case for many logics (see also Section 4)—and certain (b) *accuracy criteria* are fulfilled, a general pattern is applicable to derive monitors in a uniform manner.

We first give a generic definition of nondeterministic ω-automata which covers various well-known classes of ω-automata such as Büchi, Muller, Rabin, Street, Parity etc. Note that we decided against generalizing the concept to cover also event-clock or timed automata, mainly to keep the technical details simple.

Definition 4 (Nondeterministic ω-Automata). *A* (non-deterministic) ω-automaton $\mathcal{A}$ *is a tuple* $\mathcal{A} = (\Sigma, Q, Q_0, \delta, Acc)$*, where* Σ *is a (possibly infinite)* alphabet*,* Q *is a finite non-empty set of* states*,* $Q_0 \subseteq Q$ *is a set of* initial states*,*

$\delta : Q \times \Sigma \to 2^Q$ *is the* transition function*, and Acc is an* accepting component *(which varies for different automata types.)*

For example, we derive the notion of non-deterministic Büchi automata with a subset of the states $Acc \subseteq Q$ as accepting component.

A *run* of an ω-automaton $\mathcal{A} = (\Sigma, Q, Q_0, \delta, Acc)$ on a word $w = a_0 a_1 \cdots \in \Sigma^\omega$ is a sequence $q_0, q_1, \ldots$ with $q_0 \in Q_0$ and $q_{i+1} \in \delta(q_i, a_i)$ for $i \geq 0$. The run is called *accepting* if it meets the *acceptance condition*. For example in case of Büchi automata, the acceptance condition requires at least one of the states $q \in Acc$ to be visited infinitely often. The *accepted language* (or *language*, for short) $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$ is the set of words w for which an accepting run exists.

Yet dependent on the actual acceptance condition, emptiness of the accepted language of an automaton can usually be checked easily and is one of the standard problems extensively studied for various automata types. In the following, we silently assume that every automaton type comes with an emptiness procedure.

Definition 5 (Satisfiability Check by Automata Abstraction). *Given a linear-time logic L with its formulae F_L, the* satisfiability check by automata abstraction *proceeds as follows.*

1. *Define an* alphabet abstraction *which yields for each formula $\phi \in F_L$ with its possibly infinite alphabet Σ_ϕ an* abstract alphabet *$\bar{\Sigma}_\phi$, which is finite.*
2. *Define a* word abstraction *which yields an abstraction function $\beta_\phi : \Sigma_\phi^\omega \to \bar{\Sigma}_\phi^\omega$ for each $\phi \in F_L$.*
3. *Define an automaton construction (a computable function), which yields for all $\phi \in F_L$ an ω-automaton $\mathcal{A}_\phi$ reading words over $\bar{\Sigma}_\phi$, such that for all $\bar{\sigma} \in \bar{\Sigma}_\phi^\omega$ it holds $\bar{\sigma} \in \mathcal{L}(\mathcal{A}_\phi)$ iff $\exists \sigma \in \Sigma^\omega : \bar{\sigma} = \beta_\phi(\sigma)$ and $\sigma \models \phi$.*

The satisfiability check by automata abstraction then proceeds as follows: For a given formula $\phi \in F_L$ of the logic L construct the automaton $\mathcal{A}_\phi$ and check the language of $\mathcal{A}_\phi$ for emptiness. Clearly, ϕ is satisfiable iff $\mathcal{L}(\mathcal{A}_\phi) \neq \emptyset$.

To distinguish Σ_ϕ and $\bar{\Sigma}_\phi$ and corresponding words literally, we call $\bar{\Sigma}_\phi$ an *abstract* alphabet and elements of $\bar{\Sigma}_\phi^*$ or $\bar{\Sigma}_\phi^\omega$ *abstract* words or *symbolic abstractions*. To simplify notation, we often drop the subscript ϕ when ϕ is given by the context. For example, we write $\mathcal{A} = (\Sigma, Q, Q_0, \delta, Acc)$ for the automaton accepting symbolic abstractions $\bar{\sigma} \in \bar{\Sigma}^\omega$ of words $\sigma \in \Sigma^\omega$ (i.e. $\bar{\sigma} = \beta(\sigma)$) satisfying a fixed formula ϕ.

For a wide range of linear-time logics, the satisfiability check by automata abstraction is followed and corresponding automata constructions are provided in the literature. For example, for **LTL** as described in the previous section, the abstraction function β is simply the identity function and the automaton construction was first described in [10]. In the setting of the real-time logic **TLTL** [11], an *event-clock automaton* is constructed accepting precisely the models of the formula at hand. In the setting of Demri's $\mathbf{PLTL}^{\mathrm{mod}}$ [5], words over an infinite alphabet (representing integer valuations) are abstracted to words of a finite alphabet and a suitable construction of a Büchi automaton accepting these symbolic evaluations is provided.

The goal is now to reuse such automata constructions for generating monitors. Reconsidering the example for **LTL** given the previous section, one is drawn to the following approach: Given an ω-automaton accepting symbolic abstractions of words satisfying the formula to check, reuse its transition function δ for defining the function step. Moreover, check may be defined as checking emptiness of the accepted language of the automaton when starting the automaton in the states reachable via step/δ. Recall that we assume the satisfiability check by automata abstraction to come with an emptiness check for the automaton at hand.

However, δ reads words $\bar{\pi}$ over the symbolic alphabet while, in runtime verification, we want to derive the semantics for words π over Σ. Hence, we would like to use the symbolic abstraction function β to abstract π to $\bar{\pi}$. However, β is defined for ω-words rather than for finite words. To deal with symbolic abstractions of finite prefixes of infinite words, we introduce $\mathsf{extrapolate}(\pi)$ as

$$\mathsf{extrapolate}(\pi) = \left\{\beta(\pi\sigma)^{0\dots i} \mid i+1 = |\pi|, \sigma \in \Sigma^\omega\right\} \tag{1}$$

as the set of possible abstractions of π where $\beta(\pi\sigma)^{0\dots i}$ denotes the first $i+1$ symbols of $\beta(\pi\sigma)$. We require that there is an algorithm that yields for each ϕ a computable function $\mathsf{extrapolate} : \Sigma^* \to 2^{\bar{\Sigma}^*}$.

By means of $\mathsf{extrapolate}$, we transfer a (finite) word π to a set of symbolic words $\mathsf{extrapolate}(\pi)$, which guide the automaton from its initial states Q_0 to a set of states $\bigcup_{q' \in Q_0} \delta(q', \mathsf{extrapolate}(\pi))$, for which we check the emptiness with $\mathsf{check}(S) = \top$, iff $\bigcup_{q' \in S} \mathcal{L}(\mathcal{A}(q')) \neq \emptyset$, and $\mathsf{check}(S) = \bot$ in all other cases, where S is a subset of state set Q of the automaton for ϕ.

Now we are tempted to assume that function decide is obtained as $\mathsf{decide}(\pi) = \mathsf{check}\left(\bigcup_{q' \in Q_0, \bar{\pi} \in \mathsf{extrapolate}(\pi)} \delta(q', \bar{\pi})\right)$. However, in general this might not be correct. In the real-time setting, for example, a prefix of a timed trace typically imposes a *post-condition* for the remainder of the string. Depending on the automaton construction employed, such post-conditions of prefixes are overlooked when solely checking emptiness for states. This may then result in incorrect results. See [4] for a detailed discussion of the real-time case.

If, however, the automaton abstraction satisfies a certain accuracy condition, our intuition meets the facts:

Definition 6 (Accuracy of Abstract Automata). *A satisfiability check by automata abstraction for a given linear-time logic L is said to satisfy the* accuracy of abstract automata *property, if, for all* $\pi \in \Sigma^*$,

- *if* π *has a satisfying continuation* σ*, then there must exist an accepting abstract continuation* $\bar{\sigma}$ *for some* $\bar{\pi} \in \mathsf{extrapolate}(\pi)$*, i.e.:* $(\exists \sigma : \pi\sigma \models_L \phi) \Rightarrow (\exists \bar{\pi} \exists \bar{\sigma} : \bar{\pi}\bar{\sigma} \in \mathcal{L}(\mathcal{A}_\phi))$ *with* $\bar{\pi} \in \mathsf{extrapolate}(\pi)$,
- *and if an abstract prefix* $\bar{\pi}$ *has an accepting abstract continuation* $\bar{\sigma}$ *then there must exist a satisfying concretization* σ *for some* π *with* $\bar{\pi} \in \mathsf{extrapolate}(\pi)$*, i.e.:* $(\exists \bar{\sigma} : \bar{\pi}\bar{\sigma} \in \mathcal{L}(\mathcal{A}_\phi)) \Rightarrow (\exists \pi \exists \sigma : \pi\sigma \models_L \phi)$ *with* $\bar{\pi} \in \mathsf{extrapolate}(\pi)$.

Note that the accuracy of abstract automata property implies that the automaton only accepts valid symbolic abstractions, i.e., $\mathcal{A}$ only accepts words $\bar{\sigma}$ which are indeed images of some σ under β.

The discussion and notions introduced so far give now rise to the following theorem:

Theorem 1 (Correctness of decide). *Given a satisfiability check by automata abstraction for a linear-time logic L satisfying the* accuracy of automata *property, we have* $\mathsf{decide}(\pi) = \mathsf{check}\left(\bigcup_{q' \in Q_0, \bar{\pi} \in \mathsf{extrapolate}(\pi)} \delta(q', \bar{\pi})\right)$. *Moreover, if there is an algorithm yielding for each formula of L a computable function* extrapolate *satisfying Equation 1,* decide *is computable.*

However, as mentioned in the previous section, decide is mainly useful for obtaining non-incremental monitors, as π has to be stored for deriving abstractions of its extensions. Nevertheless, if the abstraction β satisfies further properties, we come up with an incremental monitor construction:

Definition 7 (Forgettable Past and Faithful Abstraction). *Given β of a satisfiability check by automata abstraction. We say that*

- *β satisfies the* forgettable past property*, iff $\beta(\pi a \sigma)^{i+1 \ldots i+1} = \beta(a\sigma)^{0 \ldots 0}$ for all $\pi \in \Sigma^*$, $|\pi| = i + 1$, $a \in \Sigma$, and $\sigma \in \Sigma^\omega$.*
- *β is called* faithful*, iff for all $\pi \in \Sigma^*$, $|\pi| = i + 1$, $a \in \Sigma$, $\sigma, \sigma' \in \Sigma^\omega$ for which there is some $\sigma'' \in \Sigma^\omega$ with $\beta(\pi\sigma)^{0 \ldots i}\beta(a\sigma')^{0 \ldots 0} = \beta(\sigma'')^{0 \ldots i+1}$ there also exists a $\sigma''' \in \Sigma^\omega$ with $\beta(\pi\sigma)^{0 \ldots i}\beta(a\sigma')^{0 \ldots 0} = \beta(\pi a \sigma''')^{0 \ldots i+1}$*

The intuition behind forgettable past is that a prefix of some infinite string has no effect on the abstraction of the suffix (while the suffix might influence the abstraction of the prefix). Moreover, for a faithful abstraction, we have the following: whenever the prefix of length $|\pi|$ of the abstraction of $\pi\sigma$, followed by the first letter of the abstraction of $a\sigma'$ can be written as the abstraction of some infinite word, then we obtain the same result for πa continued by a suitable suffix σ'''. Roughly speaking, this is a kind of a homomorphic property for prefixes of representatives. We then get, setting $\mathcal{L}_\beta = \{\beta(\sigma) \mid \sigma \in \Sigma^\omega\}$:

Lemma 1 (Incremental Extrapolation). *For $\pi \in \Sigma^*$, $|\pi| = i+1$, $a \in \Sigma$, we have* $\mathsf{extrapolate}(\pi)\mathsf{extrapolate}(a) \cap \mathcal{L}_\beta^{0 \ldots i+1} = \mathsf{extrapolate}(\pi a)$ *where β satisfies the forgettable past and faithful abstraction properties.*

Lemma 2 (Incremental Emptiness for Extrapolation). *Let $\mathcal{A}$ be a Büchi automaton obtained via a satisfiability check by automata abstraction satisfying the accuracy of automaton abstraction property with a faithful abstraction function having the forgettable past property. Then, for all $\pi \in \Sigma^*$ and $a \in \Sigma$, it holds* $\mathcal{L}(\mathcal{A}(\mathsf{extrapolate}(\pi a))) = \mathcal{L}(\mathcal{A}(\mathsf{extrapolate}(\pi)\mathsf{extrapolate}(a)))$.

We are now ready to define the essential procedure for an incremental monitor construction: Let step be defined by $\mathsf{step}(S, a) = \bigcup_{q' \in S, \bar{a} \in \mathsf{extrapolate}(a)} \delta(q', \bar{a})$.

Theorem 2 (Correctness of step and check). *Consider a satisfiability check by automata abstraction that has the accuracy automaton abstraction property and comes with a faithful abstraction function that moreover satisfies the forgettable past property. Then* $\mathsf{check}(\alpha(\pi)) = \mathsf{decide}(\pi)$ *for all $\pi \in \Sigma^*$. Moreover, if there is an algorithm yielding for each formula of L a computable function* extrapolate *satisfying Equation 1, α and* check *are computable.*

In other words, under the mentioned criteria, we have identified an automaton construction according to Definition 3, resulting in a monitor as depicted in Figure 1. In the next section, we illustrate the general scheme developed in this section for various linear-time logics.

4 Applications: Monitors for Various Linear-Time Logics

The anticipation semantics is suitable for various linear-time logics and a monitor synthesis by the automaton construction with emptiness check is directly obtained by establishing the functions step and check. Reusing the results of satisfiability check by automata abstraction, these functions are obtained easily. In the following, we present several linear-time logics as example applications including **LTL**, **PLTL**$^{\text{mod}}$, linear-time μ-calculus, **RLTL** and **S1S**.

Linear-time Temporal Logic (LTL) [2]: Because the alphabet of **LTL** is finite, the abstraction functions for **LTL** formulae are trivial, i.e. $\beta_\phi(\sigma) = \sigma$ and $\mathsf{extrapolate}(\pi) = \{\pi\}$. The resulting functions step_ϕ and check_ϕ are exactly the ones that we get in Section 2. Furthermore, a monitor synthesis algorithm is also obtained for the **LTL** enriched by past operators or *forgettable past operators* [12] with a corresponding satisfiability check by automata abstraction.

PLTL$^{\text{mod}}$ *[5]:* **PLTL**$^{\text{mod}}$ is a decidable fragment of Presburger **LTL**, which extends **LTL** with first-order integer arithmetic constraints. The alphabet of a **PLTL**$^{\text{mod}}$ formula is infinite because it includes all valuations of variables over $\mathbb{Z}$. [5] proposed an approach of mapping all valuations to the finite symbolic valuations which are equivalence classes. Let ϕ be a **PLTL**$^{\text{mod}}$ formula with alphabet Σ. The symbolic alphabet $\bar{\Sigma}_\phi$, the symbolic abstraction function $\beta_\phi : \Sigma_\phi^\omega \to \bar{\Sigma}_\phi^\omega$ and the automaton construction can be obtained, which are three essential elements in satisfiability check by automata abstraction. A careful investigation shows that the abstraction function is faithful and satisfies the forgettable past property and that the automaton abstraction is accurate. Thus, functions step_ϕ and check_ϕ and an anticipatory monitor can easily be constructed along the lines of Theorem 2. Details can be found in an extended version of the paper.

Linear-time μ-calculus (νTL) [6,13]: νTL extends standard modal logic with maximal and minimal fixpoint quantifiers, and can express regular expressions. In [7], a first automata-theoretic approach to νTL is presented. Given a νTL formula ϕ, the Büchi automata of $\mathcal{A}$ that accepts precisely the pre-models of ϕ and $\bar{\mathcal{A}}$ that seeks an infinite regeneration sequence for least fixpoint formula in *closure* of ϕ can be generated. The Büchi automaton $\mathcal{A}_\phi$ that accepts precisely the models of ϕ is the intersection of $\mathcal{A}$ and $\bar{\mathcal{A}}$. Thus, the abstraction functions for νTL formulae are also trivial ones, and the functions step_ϕ and check_ϕ will be established just like for **LTL**. A more direct automaton construction was presented in [14], which makes use of *parity automata*. While emptiness of the accepted language for a set of states is computed differently due to the parity acceptance condition, it is easily verified that the requirements for applying Theorem 2 are fulfilled.

RLTL *and* **S1S***:* Simlar as νTL, also regular linear temporal logic **RLTL** is a formalism that can express every ω-regular language [9]. For a **RLTL** formula ϕ, an alternating Büchi automaton accepting precisely the ω-words satisfying ϕ can be generated. This automaton can be translated into an equivalent Büchi automaton. A monadic second order (MSO) logic interpreted over words, also called **S1S**, consists of formulae having a single individual free variable, for which a Büchi automaton can be generated [8]. Again, it is easily verified that the requirements for applying Theorem 2 for **RLTL** and **S1S** are fulfilled.

5 Conclusion

In this paper, a uniform approach for synthesizing monitors checking correctness properties specified in a linear-time logic is provided. After making the notion of linear-time logics precise, a generic three-valued semantics has been introduced reflecting the idea that *prefixes* of infinite computations are checked for correctness. Then we established a conceptual framework to synthesize monitors from a logical specification to check an execution incrementally. Moreover, the main elements of the *automata-theoretic approach* for checking satisfiability of correctness properties are identified as starting point in reusing them as components for a general monitor generation procedure. We applied the presented framework and sketched monitor synthesis algorithms for a variety of logics such as **LTL**, the linear-time μ-calculus, $\mathbf{PLTL}^{\mathrm{mod}}$, **S1S**, and **RLTL**.

Besides the plain practical benefits of the developed framework, the results shed light on the similarities and differences of satisfiability checking, model checking, and runtime verification.

References

1. Bauer, A., Leucker, M., Schallhart, C.: The good, the bad, and the ugly—but how ugly is ugly? In: Sokolsky, O., Taşıran, S. (eds.) RV 2007. LNCS, vol. 4839, pp. 126–138. Springer, Heidelberg (2007)
2. Pnueli, A.: The temporal logic of programs. In: Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS), pp. 46–57 (1977)
3. Bauer, A., Leucker, M., Schallhart, C.: Monitoring of real-time properties. In: Arun-Kumar, S., Garg, N. (eds.) FSTTCS 2006. LNCS, vol. 4337. Springer, Heidelberg (2006)
4. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. Technical Report TUM-I0724, TU München (2007)
5. Demri, S.: LTL over integer periodicity constraints. Theoretical Computer Science 360(1-3), 96–123 (2006)
6. Emerson, E.A., Clarke, E.M.: Characterizing correctness properties of parallel programs using fixpoints. In: de Bakker, J.W., van Leeuwen, J. (eds.) ICALP 1980. LNCS, vol. 85, pp. 169–181. Springer, Heidelberg (1980)
7. Vardi, M.Y.: A temporal fixpoint calculus. In: POPL, pp. 250–259 (1988)
8. Büchi, J.: Weak second order logic and finite automata. Z. Math. Logik, Grundlag. Math. 5, 62–66 (1960)

9. Leucker, M., Sánchez, C.: Regular linear temporal logic. In: Jones, C.B., Liu, Z., Woodcock, J. (eds.) ICTAC 2007. LNCS, vol. 4711, pp. 291–305. Springer, Heidelberg (2007)
10. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification. In: Logic in Computer Science (LICS), pp. 332–345 (1986)
11. Raskin, J.F., Schobbens, P.Y.: State clock logic: A decidable real-time logic. In: Maler, O. (ed.) HART 1997. LNCS, vol. 1201, pp. 33–47. Springer, Heidelberg (1997)
12. Laroussinie, F., Markey, N., Schnoebelen, P.: Temporal logic with forgettable past. In: LICS (2002)
13. Barringer, H., Kuiper, R., Pnueli, A.: A really abstract concurrent model and its temporal logic. In: POPL, pp. 173–183 (1986)
14. Lange, M.: Weak automata for the linear time μ-calculus. In: Cousot, R. (ed.) VMCAI 2005. LNCS, vol. 3385, pp. 267–281. Springer, Heidelberg (2005)