Anticipatory Recurrent Monitoring with Uncertainty and Assumptions*

Hannes Kallwies[1], Martin Leucker[1], César Sánchez[2], and Torben Scheffel[1]

[1] University of Lübeck, Lübeck, Germany
{kallwies,leucker,scheffel}@isp.uni-luebeck.de
[2] IMDEA Software Institute, Madrid, Spain
cesar.sanchez@imdea.org

Abstract. Runtime Verification is a lightweight verification approach that aims at checking that a run of a system under observation adheres to a formal specification. A classical approach is to synthesize a monitor from an LTL property. Usually, such a monitor receives the trace of the system under observation incrementally and checks the property with respect to the first position of any trace that extends the received prefix. This comes with the disadvantage that once the monitor detects a violation or satisfaction of the verdict it cannot recover and the erroneous position in the trace is not explicitly disclosed. An alternative monitoring problem, proposed for example for Past LTL evaluation, is to evaluate the LTL property repeatedly at each position in the received trace, which enables recovering and gives more information when the property is breached. In this paper we study this concept of *recurrent monitoring* in detail, particularly we investigate how the notion of anticipation (yielding future verdicts when they are inevitable) can be extended to recurrent monitoring. Furthermore, we show how two fundamental approaches in Runtime Verification can be applied to recurrent monitoring, namely *Uncertainty*—which deals with the handling of inaccurate or unavailable information in the input trace—and *Assumptions*, i.e. the inclusion of additional knowledge about system invariants in the monitoring process.

1 Introduction

Runtime verification (RV) is a lightweight dynamic verification technique that focuses on analyzing an actual execution of a system to check correctness properties, which has been studied both in theory and applications [18,1]. A common specification language for RV is Linear-time Temporal Logic (LTL) [22] which was originally introduced for infinite runs. However, in RV one necessarily deals with the finite executions, and, as such, adaptions to the original semantics have

* This work was funded in part by the Madrid Regional Government under project "S2018/TCS-4339 (BLOQUES-CM)" and by a research grant from Nomadic Labs and the Tezos Foundation.

been considered. A variety of those have been proposed in the literature including infinite extensions of the finite prefix seen so far [2], limiting the logic to use only the next-operator [16], or finite version of LTL [20], strong and weak versions of LTL [9] or the so-called mission time LTL [23]. However these monitoring approaches all attempt to answer the *initial word problem* (whether the trace at the initial position satisfies the property) following different maxims. While a two valued semantics on finite words is adequate for completed, terminated executions, ongoing executions may require semantics with multiple verdicts in order to support both the current view and potential future changes when the execution is continuing. Then, for example, *impartiality* requires a logic to not change the verdict once the verdict *true* or *false* is declared, while different assessments of the current observation may be changed once more information is received. *Anticipation* takes potential look-aheads into account to sharpen the current verdict. A comprehensive comparison of such approaches is given in [4].

The seminal work by Havelund and Rosu [12] considers a different approach of monitoring. Starting from a past fragment of LTL, their monitors produce a fresh verdict about whether the property holds at the current position of the existing trace, thus recurrently answering the word problem with potentially different outcomes. We call this variant the *recurrent word problem*. While initial approaches try to return the answer for the first position of the run, Havelund's and Rosu's computes the verdict for the *current* position of the trace. As the current position is changing with every new observation, their approach implicitly restarts monitoring with every new observation. As such, while the semantics is two valued and does not change the verdict of the position in question—and can thus be considered impartial—the verdicts may change from true to false, for example, as the point of interest varies during monitoring. In this paper, we unify these two approaches (recurrent and initial monitoring), separating the monitoring time at which the questions are answered from the time at which the verdict is referring to.

In general, the recurrent word problem for future temporal logic cannot be solved with an amount of memory that is independent of the length of the trace. Consequently, most approaches with future operators are restricted to the initial word problem. Approaches to monitoring based on stream runtime verification (SRV), see for example Lola [8] produce one output stream value at each position. This output value can encode the outcome of an initial word problem or of a recurrent word problem. The common use of SRV is to encode recurrent word monitoring problems for past (or at least bounded future) specifications because the monitor is guaranteed to run with constant memory, independently of the trace length. Modern SRV systems (both synchronous and asynchronous) including RTLola [5], Lola2.0 [10], CoPilot [21], TeSSLa [7] and Striver [11] follow this approach.

In this paper we first generalize recurrent monitoring beyond Past LTL. For example, extending Past LTL with bounded future suggests that different instants in the trace (not necessarily the current instant) could be the point of interest for a given verdict. Anticipation has been solved for LTL_3 (see [3]) so it

is natural to ask whether recurrent monitoring can also be enriched with look-aheads to improve the current verdict by producing it ahead of time. We show that recurrent monitoring can indeed be extended to produce an anticipation of the number of instants before the closest violation or satisfaction, which maybe very useful to take preventive actions.

Also in the context of initial monitoring for LTL, monitors are capable to improve the verdict using partial knowledge of the underlying system [17] or, more generally, assumptions [14,6]. A second contribution of this paper is to improve recurrent monitoring with assumptions. Finally, we also consider recurrent monitoring in the presence of uncertainties, meaning, that some of the input values are (partially) unknown. We show a solution to recurrent monitoring that provides anticipation, tolerates uncertainties and is capable of exploiting assumptions.

2 Preliminaries

We use $\mathbb{Z}$ for the set of integers, $\mathbb{N}, \mathbb{N}^+$ for the set of natural numbers with and without 0 and $\mathbb{N}^\infty = \mathbb{N} \cup \{\infty\}$.

In this paper we deal with LTL extended with past time operators. The syntax of LTL with past (Full LTL) is

$$\varphi ::= tt \mid p \mid (\varphi) \mid \neg\varphi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \varphi \,\mathcal{U}\, \varphi \mid \ominus\varphi \mid \varphi \,\mathcal{S}\, \varphi$$

where p ranges over a set of atomic propositions AP. An LTL formula is a propositional logic formula over AP extended by the operators $\bigcirc\varphi$ (next) which checks that φ holds in the next state and $\varphi_1 \,\mathcal{U}\, \varphi_2$ (until) that requires φ_2 to hold at some state in the future and that in all states up to that state φ_1 holds. Additionally there are the past time operators $\ominus\varphi$ (previously) which checks that φ did hold in the previous state and $\varphi_1 \,\mathcal{S}\, \varphi_2$ (since) which enforces that φ_2 did hold at some point in the past and φ_1 held at every state since then. In order to keep following automaton constructions and proofs compact we defined only a minimal fragment of LTL. Other common LTL and Past LTL operators $\Box$ (globally), $\Diamond$ (finally), $\mathcal{R}$ (release), $\boxminus$ (globally in the past), $\diamondminus$ (once in the past) and $\mathcal{S}_w$ (weak since) can be expressed by the operators included above (see [19]).

Given an infinite word $w \in \Sigma^\omega$, where the alphabet is $\Sigma = 2^{AP}$, and given an atomic proposition $a \in AP$ we use $w(t)$ to reference the letter of w at position t. We write $p \models w(t)$ whenever $p \in w(t)$. Given a finite word $s \in \Sigma^*$ and a finite or an infinite word $w \in \Sigma^* \cup \Sigma^\omega$ we use sw for the concatenation of s followed by w. s is a *prefix* of w, denoted by $s \sqsubseteq w$ iff $w = sw'$ for some $w' \in \Sigma^* \cup \Sigma^\omega$.

A pointed word (w, t) is a pair consisting of a word w and a position $t \in \mathbb{N}^+$. We define the semantics of LTL associating pointed words to formulas as follows:

$$\begin{array}{lll}
(w,t) \models tt & & \\
(w,t) \models p & \text{iff} & p \models w(t) \\
(w,t) \models \neg\varphi & \text{iff} & (w,t) \not\models \varphi \\
(w,t) \models \varphi_1 \wedge \varphi_2 & \text{iff} & (w,t) \models \varphi_1 \text{ and } (w,t) \models \varphi_2 \\
(w,t) \models \bigcirc\varphi & \text{iff} & (w,t+1) \models \varphi
\end{array}$$

$$
\begin{array}{lll}
(w,t) \models \varphi_1 \,\mathcal{U}\, \varphi_2 & \text{iff} & \text{for some } t' \geq t.(w,t') \models \varphi_2 \text{ and} \\
 & & \quad \text{for all } t \leq t'' < t'.(w,t'') \models \varphi_1 \\
(w,t) \models \ominus \varphi & \text{iff} & (t > 1 \text{ and } (w,t-1) \models \varphi) \text{ or} \\
 & & \quad (t = 1 \text{ and } (w,1) \models \varphi) \\
(w,t) \models \varphi_1 \,\mathcal{S}\, \varphi_2 & \text{iff} & \text{for some } 1 \leq t' \leq t.(w,t') \models \varphi_2 \text{ and} \\
 & & \quad \text{for all } t' < t'' \leq t.\ (w,t'') \models \varphi_1
\end{array}
$$

Besides Full LTL we will investigate the following fragments of Full LTL:

- **Future LTL** is Full LTL restricted to the operators $\neg, \wedge, \bigcirc, \mathcal{U}$.
- **Past LTL** is Full LTL restricted to the operators $\neg, \wedge, \ominus, \mathcal{S}$.
- **Past LTL with Bounded Future**, which is Past LTL with additionally the operator $\bigcirc$.

2.1 Well-known Monitor constructions

We revise the well-known monitor constructions for Future LTL, Past LTL and Past LTL with Bounded Future.

Future LTL. The standard monitor construction for LTL is described in [3] and it is called the LTL_3 construction. This construction aims at deciding the question whether $(w,1) \models \varphi$. Therefore it iteratively receives $w \in \Sigma^\omega$, letter by letter, and calculates the verdicts $\top$ (meaning $(w,1) \models \varphi$), $\bot$ (meaning $(w,1) \not\models \varphi$) or ? (meaning a proper answer cannot be given for the prefix received up to now). Formally the output of the monitor after consumption of a finite prefix $s \sqsubseteq w$ is

- $\top$ iff $\forall w' \in \Sigma^\omega.(sw',1) \models \varphi$
- $\bot$ iff $\forall w' \in \Sigma^\omega.(sw',1) \not\models \varphi$
- ? otherwise

In other words the verdict domain $\mathbb{B}_3 = \{\top, \bot, ?\}$ encodes the set of possible outcomes of a monitoring question which are still possible after having processed prefix s of w: $\top = \{tt\}, \bot = \{ff\}, ? = \{tt, ff\}$. We will use these symbols with these meanings also for the further monitoring approaches throughout this paper. The LTL_3 construction works as follows. First, the formula φ and its negation $\neg\varphi$ are transformed into Alternating Büchi Automata by the standard LTL translation [15], which are then further transformed to Nondeterministic Büchi Automata (NBA). The monitor generation preprocesses each NBA determining which states are empty, (i.e. those states from which it is not possible to access accepting states infinitely often) which are removed from the automaton, resulting in an incomplete NFA, which is then determinized. Finally, the monitor is constructed as the product monitor of the resulting automata for φ and $\neg\varphi$. In those state pairs where the state of the negated formula is empty the monitor casts the verdict $\top$ as the formula cannot be breached anymore for the received prefix. In those pairs where the state of the non-negated monitor is

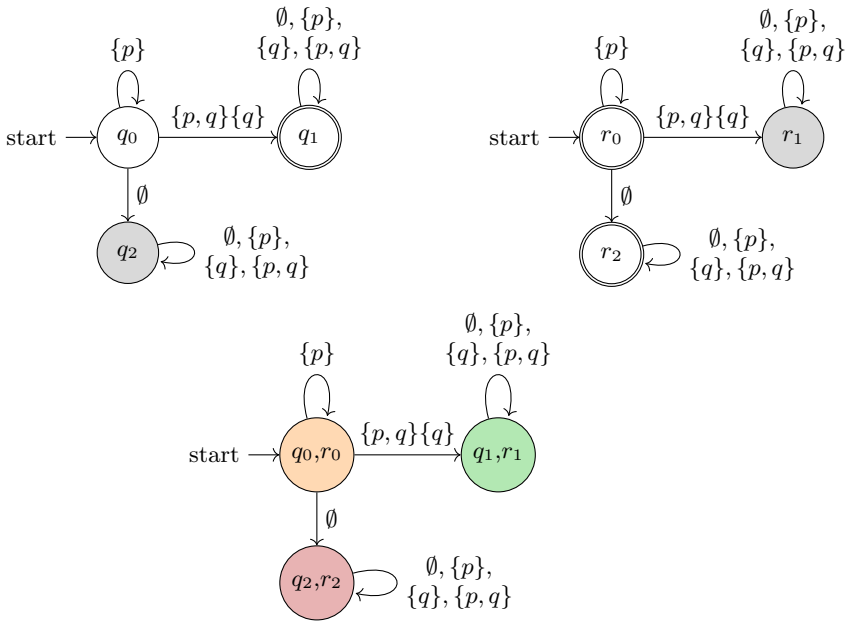

Fig. 1. Monitor for $\varphi = p\,\mathcal{U}\,q$ with the LTL$_3$ construction [3]. Up left: NBA of φ, Up right: NBA of $\neg\varphi$, bottom: Monitor generated from the product automaton. Empty states of the NBAs are marked gray. States of the monitor are marked according to their outputs: Orange for ?; green for $\top$, red for $\bot$.

empty $\bot$ is cast, as the formula cannot be satisfied anymore. Otherwise, when both states are non-empty ? is printed. An example of the monitor construction can be found in Fig. 1.

These monitors are called *impartial* because the monitor does not output a final verdict ($\top$ or $\bot$) as long as the corresponding LTL formula is not satisfied or violated for all extensions of the observed word. These monitors are also *anticipatory* because the monitor yields the final verdict at the first moment at which all extensions of the consumed prefix satisfy the verdict.

Past LTL The standard monitor construction for Past LTL [13] also receives the word letter by letter but evaluates $(w,t) \models \varphi$ at every step t, printing the outcome. In [13] the monitor is described as an imperative program that uses an array as central data structure which stores the current value (true or false) for every sub-formula of the given Past LTL formula φ. The program then receives the input word letter by letter and calculates bottom-up the new value of each sub-expression at the new instant, ultimately producing the verdict for the root formula. The size of the mentioned array data structure is the number of sub-formulas of a Past LTL formula and hence finite, so the imperative monitor can

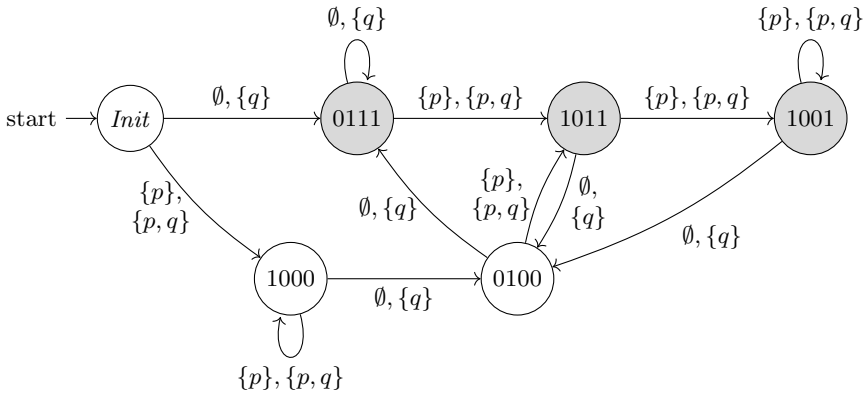

Fig. 2. Monitor for $p\,\mathcal{S}\ominus\neg p$. The states symbolize the current valuations of the sub-formulas $(p, \neg p, \ominus\neg p, p\,\mathcal{S}\ominus\neg p)$(1=true,0=false). States where the formula is satisfied for the current position are marked gray.

directly be seen as Moore machine. The state of the Moore machine is given by the current value of all sub-formulas.

Consider the Past LTL formula $p\mathcal{S}\ominus\neg p$ over $AP = \{p, q\}$. The state is given by a 4-tuple containing the current evaluation of the sub formulas $(p, \neg p, \ominus\neg p, p\mathcal{S}\ominus\neg p)$. The value of the first two entries is dependent on the current input letter. The value of the third entry is the last value of the second entry and the value of the last entry is true iff it was true before and the first entry is true or if the third entry is true. The resulting automaton is depicted in Fig. 2.

Past LTL with Bounded Future Note that introducing a next operator may make the evaluation of φ at a certain timestamp dependent on the input at a later position. We can statically determine an upper bound $\mathrm{ND}(\varphi)$ of future states which are required for evaluation of a formula φ by counting the maximal depth of nested nexts adjusted by the number of surrounding previous operators (*next depth*) in the syntax tree of the formula:

- $\mathrm{ND}(tt) = \mathrm{ND}(p) = 0$ for $p \in AP$
- $\mathrm{ND}(\neg\varphi) = \mathrm{ND}(\varphi)$
- $\mathrm{ND}(\varphi_1 \wedge \varphi_2) = \mathrm{ND}(\varphi_1\,\mathcal{S}\,\varphi_2) = \max\{\mathrm{ND}(\varphi_1), \mathrm{ND}(\varphi_2)\}$
- $\mathrm{ND}(\ominus\varphi) = \mathrm{ND}(\varphi) - 1$ and $\mathrm{ND}(\bigcirc\varphi) = \mathrm{ND}(\varphi) + 1$

For example for $\varphi = \bigcirc((\bigcirc p)\mathcal{S}\ominus q)\wedge\bigcirc p$ we have $\mathrm{ND}(\varphi) = 2$, because (depending on the evaluation of the since) it may be necessary to know at most two states of the word in advance. For $\varphi' = \bigcirc\ominus q$ we get $\mathrm{ND}(\varphi') = 0$ because φ' is only dependent on the current position of the word.

Note that if for a formula $\mathrm{ND}(\varphi) = k$ is positive, then the formula $\varphi' = \ominus^k\varphi$ ($\ominus^k$ is used as syntactic sugar for a composition of k previous operators) which expresses that φ was true k steps in the past can be rewritten to a Past LTL

formula, without next operator. Therefore the next operators can be moved inside the other operators (which is possible for all given operators without changing the semantics of the formula) until a sub-formula $\ominus\bigcirc\psi$ is contained, which can then be substituted by ψ. For $\varphi = \bigcirc((\bigcirc p)\mathcal{S}\ominus(q))\wedge\bigcirc p$ from above one could construct $\ominus^2\varphi = \ominus^2(\bigcirc((\bigcirc p)\mathcal{S}\ominus q)\wedge\bigcirc p) = \ominus(\ominus\bigcirc((\bigcirc p)\mathcal{S}\ominus q)\wedge\ominus\bigcirc p) = \ominus(((\bigcirc p)\,\mathcal{S}\,\ominus q)\wedge p) = ((p\,\mathcal{S}\,\ominus^2 q)\wedge\ominus p)$. This observation allows to transform a Past LTL formula with bounded future φ into a Past LTL formula $\varphi' = \ominus^{\mathrm{ND}(\varphi)}\varphi$ and build the corresponding monitor with the algorithm described previously. Due to the equivalence $(w,t) \models \varphi' \iff (w, t - \mathrm{ND}(\varphi)) \models \varphi$ the resulting monitor still produces the output sequence $(w,1) \models \varphi, (w,2) \models \varphi \ldots$ but with a $\mathrm{ND}(\varphi)$ offset to receiving the corresponding input letters.

3 Initial and Recurrent Monitoring

Note that the monitor construction for Past LTL differs from the construction for Future LTL. The LTL_3 monitor attempts to answer the same question $(w,1) \models \varphi$ at each step. We call this *initial monitoring*. The Past LTL monitor instead continuously answers a different question, i.e. if the formula is satisfied from the current position $t \in \mathbb{N}^+$ $((w,t) \models \varphi)$. We call this concept *recurrent monitoring*. It makes sense especially for Past LTL monitoring, where the monitor can always give the conclusive verdict $\top$ or $\bot$ for the current state (in difference to LTL_3). In general, recurrent monitoring has advantages for the monitoring process, because it checks a property with respect to a certain position in the word. Hence a breach of the LTL property is related to a specific position in the trace and not for the whole trace in general. More importantly, the monitor can also recover from errors at previous positions and continue monitoring the trace after detection of a violation.

Consider for example a robot system and a property that states whether the robot is not too close to any objects. The intention of this monitor is to be able to react (or perhaps to later inspect log data). This problem is better cast as a recurrent monitoring problem, where the monitor raises an alarm at all positions where the robot does not satisfy a property.

We now investigate the opportunities of recurrent monitoring more thoroughly. First we define initial and recurrent monitoring formally. The monitoring problem is characterized by a function $\omega : \Sigma^* \to \mathbb{B}_3$ from finite prefixes received by the monitor to the usual $\mathbb{B}_3 = \{\top, \bot, ?\}$. Recall that $\top = \{tt\}, \bot = \{ff\}, ? = \{tt, ff\}$.

Definition 1 (Initial LTL monitoring). *Given an LTL specification φ, the following function $\omega_\varphi^{init} : \Sigma^* \to \mathbb{B}_3$ is called the* initial LTL monitoring function*:*

$$\omega_\varphi^{init}(s) = \{(sw,1) \models \varphi \mid w \in \Sigma^\omega\}$$

The initial LTL monitoring problem deals with providing the set of possible verdicts for $(sw,1) \models \varphi$ given a finite prefix s. The set is naturally the set of the verdicts for all possible infinite completions of the given prefix. Note that

for two finite words $s_1 \sqsubseteq s_2$, $\omega_\varphi^{init}(s_1) \supseteq \omega_\varphi^{init}(s_2)$ holds by definition, i.e. when calculating ω_φ^{init} repeatedly on growing traces, the set of verdicts gets refined as the observed prefix gets longer.

We define next *recurrent monitoring* as the problem where the property is checked at the position up to which the monitored trace has received events.

Definition 2 (Recurrent LTL monitoring). *Given an LTL specification φ, the following function $\omega_\varphi^{rec} : \Sigma^* \to \mathbb{B}_3$ is called the* recurrent monitoring function*:*

$$\omega_\varphi^{rec}(s) = \{(sw, |s|) \models \varphi \mid w \in \Sigma^\omega\}$$

Note that Def. 2 differs from Def. 1 since $\models$ is checked at position $|s|$ currently received by the monitor, which is the traditional approach for monitoring Past LTL. For Past LTL, only states from the past are necessary for the evaluation and hence after receiving s it is always possible to cast a certain verdict ($\top$, $\bot$). However this is not the case for Future LTL or Past LTL with bounded future, where the recurrent verdict for position $|s|$ may then yield the uncertain verdict $\{tt, ff\}$ (a.k.a. ?). We propose an extension of the recurrent monitoring where the verdict that the monitor must compute is shifted by a constant offset

Definition 3 (Recurrent LTL monitoring with constant offset). *Given an LTL specification φ and $k \in \mathbb{Z}$, the* recurrent k-offset monitoring function *$\omega_\varphi^{rec,k} : \Sigma^* \to \mathbb{B}_3$ is:*

$$\omega_\varphi^{rec,k}(s) = \{(sw, |s| + k) \models \varphi \mid w \in \Sigma^\omega\}$$

Note that the recurrent LTL monitoring function from Def. 2 is equivalent to the 0-offset LTL monitoring function from Def. 3.

Another degree of generalization of the recurrent monitoring results if we require the monitor to be able to return the best possible answer about any position that cannot be predicted upfront (that is, the *monitored state* is fully independent from the *monitoring state*).

Definition 4 (Random Access Recurrent LTL monitoring). *Given an LTL specification φ, the* random access recurrent monitoring function *$\omega_\varphi : \Sigma^* \to \mathbb{N}^+ \to \mathbb{B}_3$ is:*

$$\omega_\varphi(s)(i) = \{(sw, i) \models \varphi \mid w \in \Sigma^\omega\}$$

This definition is a generalization of all previous definitions as i can be fixed with the parameter $1, |s|, |s| + k$ to receive the previous monitoring functions.

All previous definitions indeed are not restricted to any fragments of LTL. However, it is trivial to perform initial monitoring for Past LTL and in general it is useless to do 0-offset recurrent monitoring for arbitrary Future LTL formulas. On the other hand, it makes sense to apply k-offset or random recurrent monitoring to Past LTL with bounded future or even for full LTL, tolerating sometimes ? verdicts, depending on the property and the chosen offset.

We now introduce an abstract notion of RV monitor. For the purposes of this paper, a monitor

- receives a system trace iteratively (either *online* or *offline*)
- maintains internally a state which represents the trace that has been received yet (*State part*)
- iteratively produces outputs (*Question answering part*)

Definition 5 (Monitor). *A monitor is a 6-tuple* $M = (\Sigma, \Omega, Q, q_0, \delta, \omega)$ *where*
- Σ *is a possibly infinite input alphabet.*
- Ω *is a possibly infinite output alphabet.*
- Q *is a possibly infinite state space.*
- $q_0 \in Q$ *is an initial state.*
- $\delta : Q \times \Sigma \to Q$ *is a transition function.*
- $\omega : Q \to \Omega$ *is an output function.*

We refer to the verdict of a monitor $M = (\Sigma, \Omega, Q, q_0, \delta, \omega)$ after the consumption of an input $s = a_1 \dots a_n \in \Sigma^*, a_i \in \Sigma$ as $\hat{\omega}(s) = \omega(\hat{\delta}(q_0, s))$ with $\hat{\delta} : Q \times \Sigma^* \to Q$ defined as $\hat{\delta}(q, \epsilon) = q, \hat{\delta}(q, a_1 a_2 \dots a_i) = \hat{\delta}(\delta(q_0, a_1), a_2 \dots a_i)$.

A monitor is essentially a Moore machine, except that input, output and state space are allowed to be infinite. Monitors with an infinite state spaces are common in Stream Runtime Verification [8,24] where the monitors are specified in terms of streams of arbitrary data types. Since monitors in Runtime Verification usually run for an arbitrary long time and resources are limited, it is crucial that their memory is independent of the trace length and can be bounded a-priori. The state maintained by a monitor depends on the inputs consumed (using sometimes knowledge about the system under analysis), but the monitor should not need to remember the whole trace. The output part of the monitor is tailored for the application. We call monitors, whose extended output function $\hat{\omega}$ is equal to one of the functions defined above, *initial*, *recurrent*, *k-offset recurrent* and *random access recurrent* monitors. Note that for *random access recurrent* monitors there is no straight-forward implementation that "prints" the output. One alternative is that the monitor serves as an question-answering device. Another, which we present next, is that the monitor provides abstract information about future positions.

4 Anticipatory Monitoring

It is often desirable to detect failures of the system under observation as early as possible. In initial monitoring for LTL_3 this boils down to raising the verdicts $\top$, $\bot$ as soon as all possible extensions lead to satisfaction (resp. violation). For recurrent monitoring there is also another dimension of anticipation. The output of a recurrent monitor is an evaluation of the pointed semantics of an LTL formula at increasing time instants. It is sometimes possible that a monitor that is asked to cast the verdict for $(w, t) \models \varphi$ after having received a prefix of length t, is also able to cast a verdict for the next step $(w, t+1) \models \varphi$ (or even further steps in the future).

While in recurrent monitoring the verdict indicates that the LTL formula is satisfied or violated exactly at the required time instant, the user is often

interested in knowing about a future violation as soon as possible. Consider for example a crash of a monitored robotic system. There one is not only interested that the monitor reports when a crash occurs, but also that it reports as soon as a crash is inevitable. Additionally, it may be very useful to know the number of steps in the future where there is surely no violation of the property.

Consider again $p\mathcal{S}\ominus\neg p$ and the corresponding monitor in Fig. 2. This monitor yields verdict for $(w,t) \models \varphi$ after having received t letters. When the monitor has received the prefix $\{p\}\{q\}$ the monitor is in state 0100 and yields $\bot = \{ff\}$, since $(\{p\}\{q\}\ldots,2) \not\models \varphi$. However at this position it is already inevitable that the output at the next step $(\{p\}\{q\}\ldots,3) \models \varphi$ is true. In Fig. 2 this can be seen as all possible successors of 0100 are accepting states. We seek monitors that not only generate information about the current verdict but also information about future verdicts. We define such anticipatory monitors as follows:

Definition 6 (Anticipatory Monitor). *Given a monitoring problem $f : \Sigma^* \to \mathbb{V}$ over an arbitrary verdict domain $\mathbb{V}$, a monitor $M = (\Sigma, \Omega, Q, q_0, \delta, \omega)$ with $\Omega = \mathbb{N}^+ \to 2^{\mathbb{V}}$ is called an* anticipatory monitor *for f whenever for all inputs $s \in \Sigma^*$ and positions $i \in \mathbb{N}^+$,*

$$\hat{\omega}(s)(i) \supseteq \{f(sr) \mid r \in \Sigma^i\}$$

If $=$ holds instead of $\supseteq$ then M is called a perfect anticipatory monitor for f.

Note that anticipatory monitoring is defined relative to a given monitoring function f. The anticipatory monitor computes functions that predict the future verdicts of the original monitor which are possible after the current observation. In practice, implementing an anticipatory monitor requires to represent concisely the output alphabet Ω and the function $\hat{\omega}$ that approximates f. One possibility is to predict only a fixed number of future states and to implicitly map all further instants to $\mathbb{V}$ (all verdicts are possible). Alternatively, we propose to compute the minimum number of future states which are guaranteed not to be $\top$ ($\top$ meaning a crash) and the maximum number of steps until the next $\top$ is guaranteed to happen. Note that such abstractions may lead to imperfect anticipatory monitors, but the information provided may be very useful.

4.1 Anticipatory Monitors from Recurrent Monitors

We now present an algorithm to produce an anticipatory monitor for the k-offset recurrent monitoring problem. Our monitor outputs intervals $(n,m) \in \mathbb{N}^\infty \times \mathbb{N}^\infty$ as an abstraction of the full output map. The interval indicates a lower and upper bound of letters that have to be received until the property is fulfilled. Even though we only handle the steps until the formula is fulfilled, the converse (providing the steps until the formula is violated) is analogous.

Definition 7 (Anticipatory (Recurrent) Interval Monitor). *Let φ be an LTL property. A monitor $M = (\Sigma, \mathbb{I}_\mathbb{N}, Q, q_0, \delta, \omega)$ with $\mathbb{I}_\mathbb{N} = \mathbb{N}^\infty \times \mathbb{N}^\infty$ is*

called k-offset anticipatory (recurrent) interval monitor *whenever for all inputs* s, $\hat{\omega}(s) = (n, m)$ *where*

$$n = \min pos_s$$
$$m = \max pref_s$$

and

$$pos_s = \{j \in \mathbb{N} \mid \text{ for some } w \in \Sigma^\omega, (sw, |s| + j + k) \models \varphi\}$$
$$pref_s = \{j \in \mathbb{N} \mid \text{ for some } w \in \Sigma^\omega, \text{ for all } i < j, (sw, |s| + i + k) \not\models \varphi\}$$

Note that n is the shortest sequence to a violation and m is the longest sequence without a violation. Also, $n \leq m$. For example, in the case of $\varphi = p \,\mathcal{S}\, \ominus\neg p$ and the input word $w = \{p\}\{p\}\{q\}\{p\}\{q\}\ldots$ an anticipatory interval monitor would output $(2,\infty)(2,\infty)(1,1)(0,0)(1,1)\ldots$. This means in the first state after receiving input $\{p\}$ it must take at least two further steps until φ is satisfied and it is also possible that φ will never be fulfilled. After receiving two further inputs $\{p\}$ and $\{q\}$ the output $(1,1)$ indicates that it is inevitable that in the next step the property will be fulfilled. Consequently after receiving a further letter we get $(0,0)$, meaning the property holds in the current state. In practical scenarios such a monitor helps detecting inevitable situations to undertake the right countermeasures (e.g. an emergency stop) before the failure occurs. Likewise, the knowledge that a breach of the property is impossible for a time horizon also helps in some scenarios allowing for example a robot to accelerate.

We can classify the meaning of an output interval (n, m) of an anticipatory recurrent monitor as follows:

$n = \infty$	$m = \infty$	φ will never be satisfied in the future
$n \in \mathbb{N}$	$m = \infty$	φ may be satisfied in the future but not before n steps
$n \in \mathbb{N}$	$m \in \mathbb{N}^+$	φ is inevitable, but not before n or after m steps
$n = 0$	$m = 0$	φ is satisfied in the current state

The anticipatory monitor M' with the described output behavior can be constructed directly from a given recurrent monitor M as follows. The state space and transition function of M' are taken without adjustments from those of M. The modified output function for M' is generated by a simple graph traversal from M: First, every state that was labeled with output $\top$ produces the output $(0,0)$. For the outputs of the other states a depth-first-search is performed. The output of such a state is then (n, m) where n is the minimum of the first interval component of all successor states plus 1, or 0, if the state is labeled with ? and m is the maximum of the second interval component plus 1. If a state is evaluated which is already on the DFS stack its output interval is (for the pending calculation) assumed to be (∞, ∞) since in this case an infinite non-$\top$-labeled loop exists in the monitor. A formalization of the algorithm in pseudo-code can be found in Fig. 3. The resulting monitor is an anticipatory recurrent interval monitor according to Def. 6.

Theorem 1. *Given a $k \in \mathbb{Z}$ offset recurrent monitor $M = (\Sigma, \mathbb{V}, Q, q_0, \delta, \omega)$ for specification φ the construction from Fig. 3 produces a k-offset anticipatory recurrent interval monitor for φ.*

Method dfs(q, stack)
  if $q \in$ stack **then**
    $\omega'(q) \leftarrow (\infty, \infty)$; *//only temporarily set*
  else if $\omega(q) = \top$
    $\omega'(q) \leftarrow (0, 0)$;
  else
    for each $q' \in succ(q)$ **do** dfs(q', stack $\cup\{q\}$); **end for**
    //get min/max of first/second component of all successor outputs
    $n \leftarrow$ **if** $\omega(q) =?$ **then** 0 **else** $\min_{q' \in succ(q)}\{\omega'(q').\_1\} + 1$;
    $m \leftarrow \max_{q' \in succ(q)}\{\omega'(q').\_2\} + 1$;
    $\omega'(q) \leftarrow (n, m)$;
  end if
End Method

for each $q \in Q$
  dfs(q, $\emptyset$)
end for

Fig. 3. Formalization of DFS-based algorithm for construction of the output function ω' of an anticipatory interval monitor $M' = (\Sigma, \Omega', Q', q_0', \delta', \omega')$ based on a given recurring monitor $M = (\Sigma, \Omega, Q, q_0, \delta, \omega)$, $succ(q) = \{\delta(q, a) | a \in \Sigma\}$.

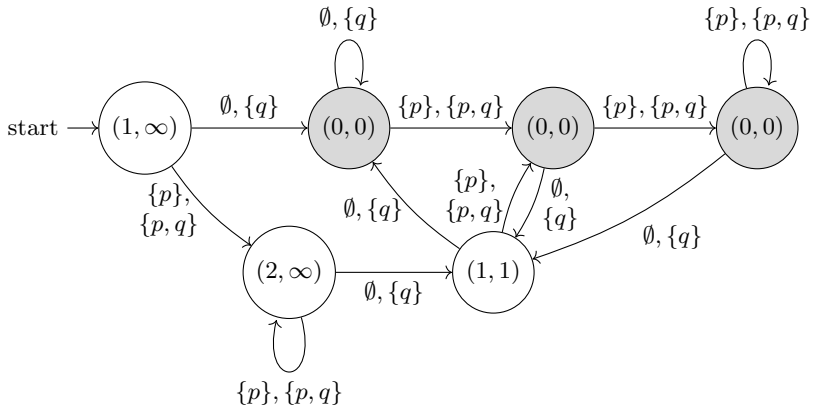

Fig. 4. Anticipatory monitor for $p\mathcal{S}\ominus\neg p$ based on the monitor from Fig. 2. The states are labeled with their corresponding interval outputs.

The result of an application of the algorithm on the recurrent monitor from Fig. 2 is depicted in Fig. 4. The output of this monitor matches the output trace described above. Note that the anticipatory monitor can also be used to answer the standard recurrent LTL monitoring problem: every state where the predictive monitor casts $(0, 0)$ is a state with verdict $\top$ in the original monitor.

5 Uncertainty and Assumptions

In this section we show how the anticipatory recurrent monitoring approach can be extended to handle uncertainty, in the sense that the content of some letters of the input word is unknown. We will also show how to exploit assumptions about the system to improve the monitoring process, where assumptions are invariants about the environment of the monitor and assumed to be always true.

5.1 Uncertainty

We model uncertain input events as subsets of Σ, which represent the set of possible inputs that actually happen. For example the input trace

$$\{\{p,q\},\{q\}\}\ \{\{p\}\}\ \{\emptyset,\{p\},\{q\},\{p,q\}\}\dots$$

encodes any trace where in the first step q holds but it is uncertain if p holds, in the second step p and not q holds (with total certainty) and where everything is possible in the third state (total uncertainty).

Given a finite prefix $s \in \Sigma^*$ and $s' \in \left(2^\Sigma\right)^*$ we write $s \models s'$ whenever s is one possible concrete representation of s', i.e. $|s| = |s'|$ and $\forall_{1\leq i\leq |s|}.s(i) \in s'(i)$. We adjust our anticipatory recurrent monitor from Def. 7 to handle uncertain inputs.

Definition 8 (Uncertain Anticipatory Recurrent Monitor). *Let φ be an LTL property. A monitor $M = (2^\Sigma, \mathbb{I}_\mathbb{N}, Q, q_0, \delta, \omega)$ is called an* uncertain k-offset anticipatory recurrent monitor *if for all inputs $s \in (2^\Sigma)^*$, $\hat{\omega}(s) = (n, m)$ where,*

$$n = \min\{pos_u \mid \textit{for some } u \models s\}$$
$$m = \max\{pref_u \mid \textit{for some } u \models s\}$$

This definition extends anticipatory recurrent monitoring to the minimal and maximal distance to a $\top$-verdict over all possible concrete input words. Note that the definition is a more general version of Def. 7, which yields the same intervals when singleton sets (certain inputs) are provided.

The classical automata-theoretic approach to handle uncertainty is the power set construction, where a new monitor is built whose state space is the power set of the original monitor's state space. When an uncertain input is received the power set monitor changes to all possible successor states of the currently possible states. The main remaining detail is how the power set automaton can produce verdicts, i.e. how the intervals of the potential states of the original monitor can be combined. We show that it suffices to take the minimum and maximum of the interval bounds of the active states. This results in the following monitor construction.

Theorem 2. *Given φ and a k-offset anticipatory interval monitor $M = (\Sigma, \mathbb{I}_\mathbb{N}, Q, q_0, \delta, \omega)$ for φ, the monitor $M' = (2^\Sigma, \mathbb{I}_\mathbb{N}, 2^Q, \{q_0\}, \delta', \omega')$ with*

- *$\delta'(S, L) = \{\delta(s, l) | s \in S, l \in L\}$ for $S \in 2^Q, L \in 2^\Sigma$*

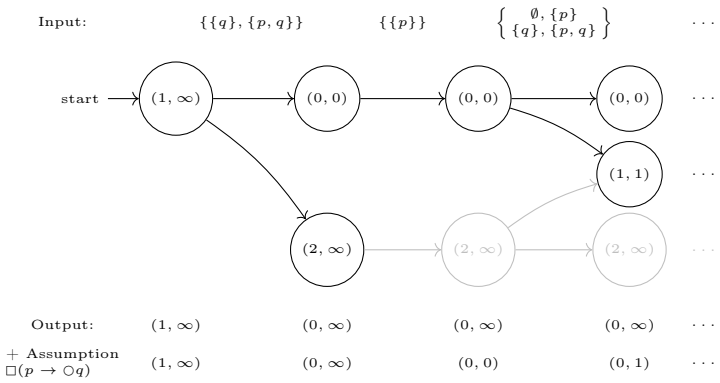

Fig. 5. Run of the anticipatory recurrent monitor from Fig. 4 with uncertain inputs. The states in which the monitor is potentially located in a time step and their outputs are drawn on top of each other. The assumption (see section 5.2) $\square(p \rightarrow \bigcirc q)$ eliminates the grey states and transitions and leads to more precise verdicts.

– $\omega'(S) = (\min\{a|(a,b) = \omega(s) \text{ for } s \in S\}, \max\{b|(a,b) = \omega(s) \text{ for } s \in S\})$
is an uncertain k-offset anticipatory recurrent monitor for φ.

A run of the recurrent anticipatory monitor from Fig. 4 for the uncertain input $\{\{p,q\},\{q\}\}\ \{\{p\}\}\ \{\emptyset,\{p\},\{q\},\{p,q\}\}\ldots$ from above is depicted in Fig. 5. The output is $(1,\infty)(0,\infty)(0,\infty)(0,\infty)\ldots$ i.e. the monitor detects that there could be three satisfactions of the property in the first three states, but depending on the real input there also could be none.

5.2 Assumptions

Another aspect with practical impact in RV is how to exploit knowledge about the system into the monitoring process. This information usually includes (partial) knowledge about the state the system is currently in and which properties (inputs to the monitor) may hold in the current and subsequent states. For example, consider the assumption $\square(p \rightarrow \bigcirc q)$, which states that every state where proposition p holds is succeeded by a state in which q holds. This assumption implies, for example, that the input word $\{p,q\}\{p\}\ldots$ will never be passed to the monitor. Since several input words or continuations are excluded, assumptions help to produce more precise verdicts and detect failures earlier. Note that, of course, one could also detect traces where an assumption is not met, indicating a severe error in the whole monitoring setting.

Especially in the presence of uncertainty in the inputs assumptions are very useful to produce more precise (anticipatory) verdicts and recover from uncertainty in the input. For example, in our assumption, observing p allows to conclude that q will follow, allowing to better anticipate. Also, not observing q

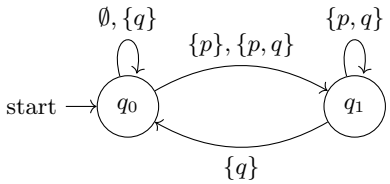

Fig. 6. Transition system corresponding to $\square(p \rightarrow \bigcirc q)$.

allows to deduce that p did not happen in the previous step, which reduces the uncertainty if the previous event was not properly received.

Additionally note that assumptions have to be explicitly handled by the monitoring algorithm, as they restrict the space of possible models, and hence cannot be encoded directly in the LTL formulas, for example as $\varphi' = \varphi_{ass} \rightarrow \varphi$ or $\varphi' = \varphi_{ass} \wedge \varphi$. Such an encoding would not allow the monitor to perform inferences about uncertain or future inputs, as it could never be sure if the assumption φ_{ass} actually holds or not.

A general way to represent assumptions and system invariants is by a Kripke structure or equivalently a transition system.

Definition 9 (Transition System). *A* Transition System *over a finite input alphabet Σ is a tuple $\mathcal{T} = (Q, q_0, \delta)$ with*
- *Q a finite state space.*
- *$q_0 \in Q$ an initial state.*
- *$\delta \in Q \times Q \times \Sigma$ a transition relation.*

A transition system $\mathcal{T}$ describes a subset of valid inputs $[\![\mathcal{T}]\!] \subseteq \Sigma^\omega$. For all words $w \in [\![\mathcal{T}]\!]$ there is a path $q_0, q_1, \ldots$ in the system such that $(q_i, q_{i+1}, w(i+1)) \in \delta$ for all $i \in \mathbb{N}$. Hence a transition system can be used as a very general way to express assumptions. Every assumption given in LTL can be used to build a corresponding transition system [19] (for simplicity we consider only safety formulas as assumptions). The transition system corresponding to the formula $\square(p \rightarrow \bigcirc q)$ is depicted in Fig. 6.

We can further refine Def. 7 now to also support assumptions given in form of a transition system $\mathcal{T}$:

Definition 10 (Uncertain Anticipatory Recurrent Monitor with Assumptions). *Let φ be an LTL specification and let $\mathcal{T}$ be a transition system over 2^Σ. A monitor $M = (2^\Sigma, \mathbb{I}_\mathbb{N}, Q, q_0, \delta, \omega)$ is called an* uncertain k-offset anticipatory recurrent monitor under assumption $\mathcal{T}$ *whenever for every $s \in (2^\Sigma)^*$, $\hat{\omega}(s) = (n, m)$ where,*

$$n = \min\{pos_u^\mathcal{T} \mid \textit{for some } u \models s\}$$
$$m = \max\{pref_u^\mathcal{T} \mid \textit{for some } u \models s\}$$

and

$$pos_s^{\mathcal{T}} = \{j \in \mathbb{N} \mid \text{ for some } w \text{ s.t. } sw \in [\![\mathcal{T}]\!], (sw, |s| + j + k) \models \varphi\}$$
$$pref_s^{\mathcal{T}} = \{j \in \mathbb{N} \mid \text{ for some } w \text{ s.t. } sw \in [\![\mathcal{T}]\!], \text{ for all } i < j, (sw, |s| + i + k) \not\models \varphi\}$$

Def. 10 allows to only care about words which are also valid inputs according to the transition system. This definition is a further generalization of Def. 8 and both are equivalent for the uninformative transition system $[\![\mathcal{T}]\!] = (2^{\Sigma})^{\omega}$.

To exploit the transition system that encodes the assumption we make use of a product construction, where states are tuples of the original monitor's states and transition system states. We only preserve transitions which are allowed by the transition system.

Since the existence of assumptions improves the anticipatory capabilities of the resulting monitor we take care already at the generation of the anticipatory monitor. Given a recurrent monitor $M = (\Sigma, \mathbb{V}, Q, q_0, \delta, \omega)$ for a specification φ, and a transition system $\mathcal{T} = (Q^{\mathcal{T}}, q_0^{\mathcal{T}}, \delta^{\mathcal{T}})$ over 2^{Σ} we first construct the recurrent monitor under assumption $M^{\mathcal{T}} = (\Sigma, \mathbb{V} \cup \{\downarrow\}\}, Q \times Q^{\mathcal{T}} \cup \{q_\perp\}, (q_0, q_0^{\mathcal{T}}), \delta', \omega')$ with

$$\delta'(q, l) = \begin{cases} (\delta(q^M, l), {q'}^{\mathcal{T}}) & \text{if } q = (q^M, q^{\mathcal{T}}) \neq q_\perp \text{ and } (q^{\mathcal{T}}, {q'}^{\mathcal{T}}, l) \in \delta^{\mathcal{T}} \\ q_\perp & \text{otherwise} \end{cases}$$

$$\omega'(q) = \begin{cases} \omega(q^M) & \text{if } q = (q^M, q^{\mathcal{T}}) \neq q_\perp \\ \downarrow & \text{otherwise} \end{cases}$$

where $q_\perp$ serves as an error state and $\downarrow$ indicates the breach of an assumption.

Together with the constructions from the previous sections this monitor builds the basis for an uncertain recurrent anticipatory monitor under assumption.

Theorem 3. *The construction from Fig. 3 applied to a k-offset recurrent monitor under assumption $M^{\mathcal{T}}$ and the subsequent application of the construction from Th. 2 yields an uncertain k-offset anticipatory recurrent monitor under assumption $\mathcal{T}$.*

In the example from Fig. 5 considering the assumption leads to the monitor run in which the grey transitions do not exist anymore in the adjusted recurrent anticipatory monitor. This is because the second input is $\{p\}$ which implies that the first letter could not have been $\{p, q\}$, which had lead to the state labeled with $(2, \infty)$. The uncertain monitor is capable of removing successors that would violate the assumptions and determines more precise verdicts. In particular, this monitor can detect a satisfaction after receiving the second letter (output $(0, 0)$). Also, after receiving the third letter this monitor can conclude that the property is fulfilled either there or at the subsequent step (output $(0, 1)$).

6 Conclusion

In this paper we have studied the concept of recurrent monitoring where monitors produce verdicts for the property at all positions. This is a promising concept both theoretical and practical, particularly for Past LTL with bounded future, as it provides more information on the position in a trace where a property violation occurs and typically allows the monitor to recover afterwards.

To be able to detect situations of interest (e.g. crashes of the observed system) as early as possible we extended the concept with a notion of anticipation and proposed a monitor construction which gives estimates about the number of steps until the next situation of interest could occur, and if it is even inevitable. We presented constructions such that these monitors can further handle uncertainty in inputs, as well as assumptions about the system, and showed how these can lead to more precise verdicts.

In general solving the recurrent word problem for arbitrary (future and past) LTL requires unbounded memory. Future work includes studying useful bounded monitors that approximate this problem. Also, we would like to extend our monitoring notion, particularly under uncertainty and assumptions, to more complex recurrent monitoring settings, like Stream Runtime Verification. We also aim at implementation, particularly of an SRV engine, and an empirical evaluation on realistic case studies.

References

1. Bartocci, E., Falcone, Y. (eds.): Lectures on Runtime Verification - Introductory and Advanced Topics, Lecture Notes in Computer Science, vol. 10457. Springer (2018). https://doi.org/10.1007/978-3-319-75632-5, `https://doi.org/10.1007/978-3-319-75632-5`
2. Bauer, A., Leucker, M., Schallhart, C.: Monitoring of real-time properties. In: Proc. of the 26th Int'l Conf. on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'06). LNCS, vol. 4337, pp. 260–272. Springer (2006). https://doi.org/10.1007/11944836_25
3. Bauer, A., Leucker, M., Schallhart, C.: Monitoring of real-time properties. In: FSTTCS. Lecture Notes in Computer Science, vol. 4337, pp. 260–272. Springer (2006)
4. Bauer, A., Leucker, M., Schallhart, C.: Comparing LTL semantics for runtime verification. J. Log. Comput. **20**(3), 651–674 (2010). https://doi.org/10.1093/logcom/exn075, `https://doi.org/10.1093/logcom/exn075`
5. Baumeister, J., Finkbeiner, B., Schirmer, S., Schwenger, M., Torens, C.: RTLola cleared for take-off: Monitoring autonomous aircraft. In: Proc. of 32nd Int'l Conf. on Computer-Aided Verification CAV'20. pp. 28–39. Springer (2020)
6. Cimatti, A., Tian, C., Tonetta, S.: Assumption-based runtime verification of infinite-state systems. In: Proc. of the 21st Int'l Conf. on Runtime Verification (RV'21). LNCS, vol. 12974, pp. 207–227. Springer (2021). https://doi.org/10.1007/978-3-030-88494-9_11

7. Convent, L., Hungerecker, S., Leucker, M., Scheffel, T., Schmitz, M., Thoma, D.: TeSSLa: Temporal stream-based specification language. In: Proc. of the 21th Brazilian Symp. on Formal Methods (SBMF'18). LNCS, vol. 11254, pp. 144–162. Springer (2018). https://doi.org/10.1007/978-3-030-03044-5_10
8. D'Angelo, B., Sankaranarayanan, S., Sánchez, C., Robinson, W., Finkbeiner, B., Sipma, H.B., Mehrotra, S., Manna, Z.: LOLA: runtime monitoring of synchronous systems. In: 12th International Symposium on Temporal Representation and Reasoning (TIME 2005), 23-25 June 2005, Burlington, Vermont, USA. pp. 166–174. IEEE Computer Society (2005). https://doi.org/10.1109/TIME.2005.26, https://doi.org/10.1109/TIME.2005.26
9. Eisner, C., Fisman, D., Havlicek, J., Lustig, Y., McIsaac, A., Campenhout, D.V.: Reasoning with temporal logic on truncated paths. In: Jr., W.A.H., Somenzi, F. (eds.) Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2725, pp. 27–39. Springer (2003). https://doi.org/10.1007/978-3-540-45069-6_3, https://doi.org/10.1007/978-3-540-45069-6_3
10. Faymonville, P., Finkbeiner, B., Schledjewski, M., Schwenger, M., Stenger, M., Tentrup, L., Torfah, H.: StreamLAB: Stream-based monitoring of cyber-physical systems. In: Proc. of the 31st Int'l Conf. on Computer-Aided Verification (CAV'19). LNCS, vol. 11561, pp. 421–431. Springer (2019). https://doi.org/10.1007/978-3-030-25540-4_24
11. Gorostiaga, F., Sánchez, C.: Stream runtime verification of real-time event streams with the Striver language. International Journal on Software Tools for Technology Transfer **23**, 157–183 (2021). https://doi.org/10.1007/s10009-021-00605-3
12. Havelund, K., Rosu, G.: Synthesizing monitors for safety properties. In: Katoen, J., Stevens, P. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, 8th International Conference, TACAS 2002, Held as Part of the Joint European Conference on Theory and Practice of Software, ETAPS 2002, Grenoble, France, April 8-12, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2280, pp. 342–356. Springer (2002). https://doi.org/10.1007/3-540-46002-0_24, https://doi.org/10.1007/3-540-46002-0_24
13. Havelund, K., Rosu, G.: Synthesizing monitors for safety properties. In: TACAS. Lecture Notes in Computer Science, vol. 2280, pp. 342–356. Springer (2002)
14. Henzinger, T.A., Saraç, N.E.: Monitorability under assumptions. In: Deshmukh, J., Nickovic, D. (eds.) Proc. of the 20th Int'l Conf. on Runtime Verification (RV'20). LNCS, vol. 12399, pp. 3–18. Springer (2020). https://doi.org/10.1007/978-3-030-60508-7_1
15. Kupferman, O., Vardi, M.Y.: Weak alternating automata are not that weak. ACM Transactions on Computational Logic **2**(3), 408–429 (2001)
16. Leucker, M.: Teaching runtime verification. In: Khurshid, S., Sen, K. (eds.) Runtime Verification - Second International Conference, RV 2011, San Francisco, CA, USA, September 27-30, 2011, Revised Selected Papers. Lecture Notes in Computer Science, vol. 7186, pp. 34–48. Springer (2011). https://doi.org/10.1007/978-3-642-29860-8_4, https://doi.org/10.1007/978-3-642-29860-8_4
17. Leucker, M.: Sliding between model checking and runtime verification. In: Qadeer, S., Tasiran, S. (eds.) Runtime Verification, Third International Conference, RV 2012, Istanbul, Turkey, September 25-28, 2012, Revised Selected Papers. Lecture Notes in Computer Science, vol. 7687, pp. 82–87. Springer (2012). https://doi.org/10.1007/978-3-642-35632-2_10, https://doi.org/10.1007/978-3-642-35632-2_10

18. Leucker, M., Schallhart, C.: A brief account of runtime verification. J. Logic Algebr. Progr. **78**(5), 293–303 (2009)
19. Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer-Verlag (1991)
20. Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems. Springer, New York (1992)
21. Perez, I., Dedden, F., Goodloe, A.: Copilot 3. Tech. Rep. NASA/TM–2020–220587, NASA Langley Research Center (April 2020)
22. Pnueli, A.: The temporal logic of programs. In: Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS-77). pp. 46–57. IEEE Computer Society Press, Providence, Rhode Island (Oct 31–Nov 2 1977)
23. Reinbacher, T., Rozier, K.Y., Schumann, J.: Temporal-logic based runtime observer pairs for system health management of real-time systems. In: Proc. 20th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14). LNCS, vol. 8413, pp. 357–372. Springer (2014). https://doi.org/10.1007/978-3-642-54862-8_24
24. Sánchez, C.: Online and offline stream runtime verification of synchronous systems. In: Proc. of the 18th Int'l Conf. on Runtime Verification (RV'18). LNCS, vol. 11237, pp. 138–163. Springer (2018). https://doi.org/10.1007/978-3-030-03769-7_9