




General Anticipatory Monitoring for Temporal Logics on Finite Traces^{*}

Hannes Kallwies¹, Martin Leucker¹, and César Sánchez²

¹ University of Lübeck, Lübeck, Germany
{kallwies,leucker}@isp.uni-luebeck.de

² IMDEA Software Institute, Madrid, Spain
cesar.sanchez@imdea.org

Abstract. Runtime Verification studies how to check a run of a system against a formal specification, typically expressed in some temporal logic. A monitor must produce a verdict at each step that is sound with respect to the specification. It is often the case that a monitor must produce a ? verdict and wait for more observations. On the other hand, sometimes a verdict is inevitable but monitoring algorithms wait to produce the verdict, because it seemingly depends on future inputs. Anticipation is the property of a monitor to immediately produce inevitable verdicts, which has been studied for logics on infinite traces.

Monitoring problems depend on the logic and on the semantics that the monitor follows. In initial monitoring, at every instant the monitor answers whether the specification holds for the observed trace from the initial state. In recurrent monitoring, the monitor answers at every instant whether the specification holds at that time.

In this paper we study anticipatory monitoring for temporal logics on finite traces. We first show that many logics on finite traces can be reduced linearly to Boolean Lola specifications and that initial monitoring can be reduced to recurrent monitoring for Lola. Then we present an algorithm with perfect anticipation for recurrent monitoring of Boolean Lola specifications, which we then extend to exploit assumptions and tolerate uncertainties.

1 Introduction

In this paper we study the anticipatory recurrent monitoring problem for runtime verification of temporal logics on finite traces. We provide a general solution and extend it to handle assumptions and uncertainties.

Runtime verification (RV) is a lightweight formal dynamic verification technique analyzing single executions of systems wrt. given correctness properties.

^{*} This work was funded in part by PRODIGY Project (TED2021-132464B-I00) funded by MCIN/AEI/10.13039/501100011033/ and the European Union NextGenerationEU/PRTR, and by a research grant from Nomadic Labs and the Tezos Foundation.

RV has been studied both in theory and practical applications [25,1]. The starting point is a formal specification of the property to monitor. A common specification language is Linear-time Temporal Logic (LTL) [30] which was originally introduced for infinite runs. Since in monitoring the sequence of observations at any point in time is necessarily finite, LTL has been adapted to finite traces, including infinite extensions of the finite prefix seen so far [4], limiting the logic to use only the next-operator [22], or finite version of LTL [26], strong and weak versions of LTL [14] or the so-called mission time LTL [32]. These monitoring approaches attempt to answer the *initial monitoring problem*: whether the trace at the initial position satisfies the property. Monitoring ongoing executions requires to emit a verdict for every event observed, so an uncertain verdict “?” is temporarily produced if the trace observed is not yet guaranteed to be only extendable into a model (verdict *tt*) or only extendable into a counter-model (verdict *ff*). Consider for example $\Box(p \rightarrow \Diamond q)$ (globally a p implies that there was once a q). The monitor emits ? until the first q or p is observed. The monitor emits *tt* if q happens no later than the first p , and *ff* if p happens strictly before the first q . In initial monitoring, once a certain verdict (*tt,ff*) is produced it remains fixed.

The seminal work by Havelund and Rosu [18] considers an alternative approach. Starting from specifications of past LTL formulas, the monitors in [18] produce at instant i a fresh verdict about whether the property holds at i , thus recurrently producing potentially different outcomes. We call this variant *recurrent monitoring*. As the current position is shifted with every new observation, recurrent monitoring performs a different evaluation at every instant. When recurrently monitoring $\Box(p \rightarrow \Diamond q)$, the monitor emits a *ff* for each p that is before the first q , then recovers and starts emitting ? attempting to see a q before the next p (moving then to *tt*). These two approaches are unified in [19], separating the monitoring time at which the questions are answered from the time at which the verdict is referring to.

In recurrent monitoring, the output for the specification at time i is either produced, or a “?” is cast and the concrete verdict for time i is never cast. An alternative family of formalisms for runtime verification are stream-based runtime verification (SRV), pioneered by Lola [12], which produce one output stream value for each input position (delaying if necessary the production of the outcome of the monitor for input i until a later instant). We call this variant the *universal monitoring* because the monitors ultimately produce (sooner or later or even at the end of the trace) all verdicts for all positions. Even though the common use of SRV is to encode recurrent monitoring problems for past (or at least bounded future) specifications future universal monitoring can be performed at the price of (1) unbounded resources and (2) only guaranteeing all verdicts at the end of the trace. Modern SRV systems (both synchronous and asynchronous) including RTLola [7], Lola2.0 [15], CoPilot [29], TeSSLa [11] and Striver [17] follow this approach. In summary,

- *initial monitoring* attempts to answer, at every instant t , whether the observed trace satisfies the specification if evaluated at time 0.

- *recurrent monitoring* attempts to answer, at every instant t , whether the observed trace satisfies the specification if evaluated at t .
- *universal monitoring* attempts to answer, as soon as possible and for every position t , whether the observed trace satisfies the specification at t .

It is desirable that a monitor produces a verdict as soon as possible when this verdict is inevitable, a feature called anticipation. For example, a naive monitor for the initial monitoring for $(\text{XX } \textit{false})$ would wait two steps until *false* is encountered to produce *ff* as verdict. An anticipatory monitor would immediately produce the correct verdict *ff* at time 0. As another example, $\text{X}(p \rightarrow (\text{X } \textit{false}))$ would require to check whether p holds at the first instant (producing *ff* if p holds and *tt* if p does not hold), under perfect anticipation. Anticipation has been solved for LTL on ω -words [5] where all (infinite) futures are explored at every step and deciding an outcome when the opposite is impossible. Anticipation guarantees that equivalent specifications produce the same outputs for the same inputs and at the same times.

The contributions of this paper are the following. We consider many logics for finite traces (in Section 2) and translate them into SRV language Lola on Boolean streams (in Section 3), and show that initial monitoring can be reduced to recurrent monitoring for Lola. Section 4 gives a recurrent monitoring algorithm with perfect anticipation for Lola, and extends it to exploit assumptions and tolerate uncertainties. We have implemented our approach in a prototype tool and report on a preliminary empirical evaluation (Section 5).

2 Temporal Logics on Finite Traces

Preliminaries. We use \mathbb{Z} for the set of integers and $\mathbb{T} = \{0 \dots N - 1\}$ for the natural numbers from 0 to $N - 1$. Given a set of propositions AP the alphabet $\Sigma = 2^{AP}$ consists of subsets of atomic propositions. A word σ is an element of Σ^+ , and $|\sigma|$ is the length of σ . We say that a natural number i is an index or position of a word σ whenever $0 \leq i < |\sigma|$. Given a word σ and an index i , we use $\sigma(i)$ for the letter at position i , (σ, i) is called a “pointed word”, and (σ, i, j) is called a “segment”. A *basic expression* is a Boolean combination of elements from AP , defined as follows:

$$\beta ::= \textit{true} \mid a \mid \beta \wedge \beta \mid \beta \vee \beta \mid \neg\beta$$

where $a \in AP$ is an atomic proposition. Given a letter s from Σ , $s \models p$ is defined as $s \models a$ whenever $a \in s$, and the usual definitions for Boolean operators.

We define non-deterministic finite automata with a forward and backwards acceptance, in terms of segments of words. An ϵ -NFA over alphabet Σ is a tuple $(Q, q_0, \delta, \delta_\epsilon, F)$ where Q is a finite set of states, q_0 is the initial state, $F \subseteq Q$ is a set of final states, $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation and $\delta_\epsilon \subseteq Q \times Q$ is the epsilon transition relation. Given a word σ , two positions $0 \leq i, j < |\sigma|$ and an ϵ -NFA A , we say that A accepts (σ, i, j) in the *forward manner*, denoted $(\sigma, i, j) \models A$ if there is a sequence of states and positions $(q_0, i_0), (q_1, i_1) \dots (q_n, i_n)$ starting at q_0 such that (1) $i_0 = i$ and $i_n = j$; (2)

$q_n \in F$; and (3) for every $0 \leq k < n$, either $(q_k, \sigma(i_k), q_{k+1}) \in \delta$ and $i_{k+1} = i_k + 1$, or $(q_k, q_{k+1}) \in \delta_\epsilon$ and $i_{k+1} = i_k$.

Similarly, A accepts (σ, i, j) in the *backwards manner*, denoted $(\sigma, i, j) \models A^{-1}$ if there is a sequence of states and positions $(q_0, i_0), (q_1, i_1) \dots (q_n, i_n)$ starting at q_0 such that (1) $i_0 = i$ and $i_n = j$, (2) $q_n \in F$ and (3) for every $0 \leq k < n$, either $(q_k, \sigma(i_k), q_{k+1}) \in \delta$ and $i_{k+1} = i_k - 1$, or $(q_k, q_{k+1}) \in \delta_\epsilon$ and $i_{k+1} = i_k$.

2.1 Temporal Logics and Formalisms on Finite Traces

We now present several temporal logics over finite traces:

- LTL_f: an adaptation of LTL to finite traces [27,16], with past operators.
- RE: regular expressions [21,28] extended with past.
- RLTL_f: Regular Linear Temporal Logic (RLTL) [24,34,35] for finite paths.
- LDL_f: linear dynamic logic on finite traces [16].
- TRLTL_f: a slight variation of RLTL_f introduced in this paper.
- Lola: a stream runtime verification language [12].

For all these formalisms we use basic expressions over AP for individual observations obtained from the environment.

LTL_f. Manna and Pnueli [27] already studied how to adapt LTL from infinite traces to finite traces, by observing that one can adapt the next operator (X) into a new variant (weak next) $\tilde{X}\varphi$. Weak next is always true at the end of the trace in spite of the sub-formula φ , while $X\varphi$ is defined to be false at the end of the trace in spite of φ . These notions are dual to the corresponding past operators $\ominus\varphi$ (which is automatically true at the first position) and $\ominus\varphi$ (which is automatically false at the first position). The syntax of LTL_f is:

$$\varphi ::= \beta \mid \varphi \wedge \varphi \mid \neg\varphi \mid X\varphi \mid \tilde{X}\varphi \mid \varphi \mathcal{U} \varphi \mid \ominus\varphi \mid \ominus\varphi \mid \varphi \mathcal{S} \varphi$$

where β is a basic expression. The semantics of LTL_f associates traces $\sigma \in \Sigma^+$ with formulae as follows:

$$\begin{aligned} (\sigma, i) \models \beta & \quad \text{iff } \sigma(i) \models \beta \\ (\sigma, i) \models \varphi_1 \wedge \varphi_2 & \quad \text{iff } (\sigma, i) \models \varphi_1 \text{ and } (\sigma, i) \models \varphi_2 \\ (\sigma, i) \models \neg\varphi & \quad \text{iff } (\sigma, i) \not\models \varphi \\ (\sigma, i) \models X\varphi & \quad \text{iff } i + 1 < |\sigma| \text{ and } (\sigma, i + 1) \models \varphi \\ (\sigma, i) \models \tilde{X}\varphi & \quad \text{iff } i + 1 \geq |\sigma| \text{ or } (\sigma, i + 1) \models \varphi \\ (\sigma, i) \models \ominus\varphi & \quad \text{iff } 0 > i - 1 \text{ or } (\sigma, i - 1) \models \varphi \\ (\sigma, i) \models \ominus\varphi & \quad \text{iff } 0 \leq i - 1 \text{ and } (\sigma, i - 1) \models \varphi \\ (\sigma, i) \models \varphi_1 \mathcal{U} \varphi_2 & \quad \text{iff for some } j \geq i \text{ } (\sigma, j) \models \varphi_2 \text{ and for all } i \leq k < j, (\sigma, k) \models \varphi_1 \\ (\sigma, i) \models \varphi_1 \mathcal{S} \varphi_2 & \quad \text{iff for some } j \leq i \text{ } (\sigma, j) \models \varphi_2 \text{ and for all } j < k \leq i, (\sigma, k) \models \varphi_1 \end{aligned}$$

We use common derived operators like \vee as the dual of \wedge , \mathcal{R} as the dual of \mathcal{U} , \diamond (as *true* $\mathcal{U} \varphi$) and \square (as *false* $\mathcal{R} \varphi$). Likewise for past: \diamondleftarrow (as *true* $\mathcal{S} \varphi$). Note that $(\sigma, i) \models \varphi \mathcal{U} \psi$ if and only if $(\sigma, i) \models \psi \vee (\varphi \wedge X(\varphi \mathcal{U} \psi))$. Also $\neg X\varphi$ is equivalent to $\tilde{X}\neg\varphi$, $\neg\tilde{X}\varphi$ is equivalent to $X\neg\varphi$, $\neg\ominus\varphi$ is equivalent to $\ominus\neg\varphi$, $\neg\ominus\varphi$ is equivalent to $\ominus\neg\varphi$. The presented logic was later re-introduced in [16] and named LTL_f.

RE with Past. Regular expressions [21,28] is a classical formalism to express regular sets of finite words. The syntax of RE is:

$$\rho ::= \beta \mid \rho + \rho \mid \rho ; \rho \mid \rho^* \rho$$

where β is a basic expression. For convenience, we define the semantics of regular expressions using segments (as in [24]):

$$\begin{aligned} (\sigma, i, j) \models \beta & \quad \text{iff } \sigma(i) \models \beta \text{ and } j = i + 1 \\ (\sigma, i, j) \models x + y & \quad \text{iff } (\sigma, i, j) \models x \text{ or } (\sigma, i, j) \models y \\ (\sigma, i, j) \models x ; y & \quad \text{iff for some } k < |\sigma|, (\sigma, i, k) \models x \text{ and } (\sigma, k, j) \models y. \\ (\sigma, i, j) \models x^* y & \quad \text{iff either } (\sigma, i, j) \models y, \text{ or for some sequence } (i_0 = i, i_1, \dots, i_m), \\ & \quad (\sigma, i_k, i_{k+1}) \models x \text{ and } (\sigma, i_m, j) \models y \end{aligned}$$

We say that a finite word σ matches a regular expression ρ whenever $(\sigma, 0, |\sigma|) \models \rho$. In [34] past regular expressions were introduced in the context of regular linear temporal logic RLTL. The main idea is to define a new operator $\bar{\beta}$ for a basic expression β defined as $(\sigma, i, j) \models \bar{\beta}$ iff $\sigma(i) \models \beta$ and $j = i - 1$. Then, a pure past regular expression $\bar{\rho}$ is obtained from a regular expression ρ by replacing all basic expressions β with $\bar{\beta}$. Note that all basic steps in a pure future regular expression move forward and all basic steps in a past regular expression move backwards. It is crucial that we have first defined basic expressions (with \wedge , \vee and \neg) that work on single letters and we do not allow \wedge and \neg in regular expressions, to allow linear translations into richer logics.

TRLTL_f. Regular Linear Temporal Logic (RLTL) [24] (see also [34,35]) extends the expressivity of LTL to all regular languages, introducing temporal operators that generalize both temporal operators from LTL and concatenation from regular expressions. The resulting logic has the same complexity as LTL and allows linear translations from both LTL and regular expressions. We now introduce a variation of RLTL for finite traces, called TRLTL_f where we also add the capability in the regular expression layer to test previously defined formulas. RLTL_f is TRLTL_f without the $\varphi?$ operator below, which does not add expressivity because RLTL_f without $\varphi?$ can already cover all regular languages. The resulting syntax has a regular expression layer (ρ) and a temporal layer (φ), where α is only used to decide whether the regular expression is interpreted forward or backwards in the trace.

$$\begin{aligned} \rho & ::= \rho + \rho \mid \rho ; \rho \mid \rho^* \rho \mid \beta \mid \varphi? & \alpha & ::= \rho \mid \bar{\rho} \\ \varphi & ::= true \mid \beta \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \neg \varphi \mid \alpha ; \varphi \mid \varphi \langle \alpha \rangle \varphi \end{aligned}$$

Note that the regular expression layer is extended with a “test operator” $\varphi?$ whose intention is to extend the language of atomic propositions with the capability to check previously defined expressions. We only introduce $\varphi?$ to obtain an immediate subsumption from LDL_f below. The semantics of the $\varphi?$ operator is $(\sigma, i, i) \models \varphi?$ iff $(\sigma, i) \models \varphi$.

The operator $\varphi | \alpha \rangle \rangle \varphi$ is called the power operator. The power expression $x | r \rangle \rangle y$ (read *x at r until y*) is built from three elements: y (the *attempt*), x (the *obligation*) and r (the *delay*). Informally, for $x | r \rangle \rangle y$ to hold, either the attempt holds, or the obligation is met and the whole expression evaluates successfully after the delay. In particular, for a power expression to hold, the obligation must be met after a finite number of delays. The power operator generalizes both Kleene repetition (x^*y is simply $true | x \rangle \rangle y$) and the LTL Until operator ($x \mathcal{U} y$ is simply $x | true \rangle \rangle y$). That is, conventional regular expressions can describe sophisticated delays with trivial obligations and escapes, while conventional LTL_f constructs allow complex obligations and escapes, but trivial one-step delays. The power operator extends the expressive power of LTL, for example, Wolper’s expression [36] “ p holds at even moments”—that cannot be expressed in LTL—is defined in $TRLTL_f$ as $\neg(true | true ; true \rangle \rangle \neg p)$, that is “it is not the case that after some sequence of $true ; true$, there is no p ”.

The completeness of $TRLTL_f$ with respect to regular languages is easily derived from the expressibility of regular expressions. Formally, the semantics of $TRLTL_f$ (\wedge , \vee and \neg are standard as in LTL_f above):

$$\begin{aligned} (\sigma, i) \models r ; \varphi & \quad \text{iff for some } j \ (\sigma, i, j) \models r \text{ and } (\sigma, j) \models \varphi \\ (\sigma, i) \models \varphi_1 | r \rangle \rangle \varphi_2 & \quad \text{iff for some sequence } (i_0 = i, i_1, \dots, i_m) : (\sigma, i_m) \models \varphi_2 \text{ and} \\ & \quad (\sigma, i_k, i_{k+1}) \models r \text{ and } (\sigma, i_k) \models \varphi_1 \text{ for every } k < m \end{aligned}$$

It is easy to see that LTL_f is subsumed by $TRLTL_f$ (using the linear translation from \mathcal{U}) because $X\varphi$ is $true ; \varphi$. Similarly, RE (for pure past expressions or pure future expressions) can also be expressed in $TRLTL_f$ using the linear translation for Kleene star.

LDL_f. Linear Dynamic Logic on finite traces (LDL_f) was introduced [16] as an extension of LTL_f to increase the expressivity to regular languages, inspired by dynamic logic. As $RLTL$, LDL_f considers a regular expression layer (extended with test) but restricts the temporal layer to a single “dynamic operator” $\langle \alpha \rangle \varphi$:

$$\varphi ::= \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \neg \varphi \mid \langle \alpha \rangle \varphi \mid [\alpha] \varphi$$

The semantics of the dynamic operator are precisely $\langle \alpha \rangle \varphi = \alpha ; \varphi$, while $[\alpha] \varphi$ is its dual $\neg(\alpha ; \neg \varphi)$. Therefore LDL_f can be translated linearly to $TRLTL_f$. Note that $TRLTL_f$ contains all operators from LDL_f and $RLTL_f$ to ease the translation from both. Since the expressive power of both LDL_f and $RLTL_f$ is the set of all regular languages they are equally expressive. We conjecture that one can have a linear translation from LDL_f into $RLTL_f$ and vice-versa, but the proof of this conjecture is out of the scope of this paper.

The following lemma summarizes our expressivity results.

Lemma 1. *For every LTL_f , RE, $RLTL_f$ and LDL_f expression there is an equivalent $TRLTL_f$ expression of linear size.*

It is well-known that RE, $RLTL_f$ (and therefore $TRLTL_f$) and LDL_f can express all regular languages. It is an open problem whether there is a *linear* translation from $RLTL$ into LDL and from $RLTL_f$ into LDL_f .

2.2 The Stream Runtime Verification Language Lola

In this section we recall the Lola stream runtime verification language [12]. A Lola specification describes a transformation from a set of input to output streams. Let \mathbb{D} be an arbitrary data domain (which essentially is a collection of types and constructor symbols, with their interpretations as values and functions). We denote by $\mathcal{S}_{\mathbb{D}} : \mathbb{T} \rightarrow \mathbb{D}$ the set of streams of type \mathbb{D} . In this paper we restrict ourselves to Boolean streams, i.e. domain $\mathbb{B} = \{tt, ff\}$ with the usual symbols *true*, *false* \wedge , \vee , \neg , etc.

The output streams of a Lola specification are defined by expressions over other stream identifiers. Given a set of Boolean stream variables S , the set of Lola expressions $Expr_S$ is:

$$Expr_S = true \mid false \mid s[o|c] \mid \neg Expr_S \mid Expr_S \wedge Expr_S \mid Expr_S \vee Expr_S$$

where $s \in S$ is a stream variable, $o \in \mathbb{Z}$ is an offset, and $c \in \{tt, ff\}$ is a Boolean constant. Thus a Lola expression is either a constant or the application of a \neg , \wedge , \vee . The intended meaning of $s[o, c]$ is the value of stream s , o time instants from the current position, using c as default value if this position does not exist (because the offset takes the position beyond the beginning or end of the trace). For example, $s[3, tt]$ represents the value of s three instants in the future, or tt if the trace end is reached. Note that an offset 0 references the current value of other streams, and the index is guaranteed to be legal after adding 0. Since in this case the default value is not necessary we use the alternatives $s[now]$ or s for $s[0, tt]$. Further we use *true*, *false* for the stream which has value tt resp. ff at all instances. In the following we assume Lola specifications to be in a so-called flat format, i.e. only the offsets $-1, 0, 1$ may be used. It is easy to see that every Lola specification can be transformed into a flat equivalent by introducing intermediate streams and splitting larger offsets in a sequence of $+1/-1$ offsets. This translation is linear (in the unary encoding of offsets).

A Lola specification $\varphi = (I, S, E)$ is given by I an ordered set of input stream identifiers, S an ordered set of output stream identifiers disjunct from I , and $E : S \rightarrow Expr_{S \cup I}$ a mapping which assigns to every output stream its defining expression. The semantics of a Lola specification $\varphi = (I = (i_1, \dots, i_n), S = (s_1, \dots, s_m), E)$ is a transformation from input to output streams: $\llbracket \varphi \rrbracket : (\mathcal{S}_{\mathbb{B}})^n \rightarrow (\mathcal{S}_{\mathbb{B}})^m$ with $\llbracket \varphi \rrbracket(\tau_1, \dots, \tau_n) = (\sigma_1, \dots, \sigma_m)$ such that $\sigma_i(t) = \llbracket E(s_i) \rrbracket(t)$ for all $i \in \{1, \dots, m\}$, $t \in \mathbb{T}$ where the semantics of the defining expression is given as follows (for $c \in \mathbb{B}$, $o \in \mathbb{Z}$, $e_1, e_2 \in Expr_{S \cup I}$, stream σ corresponding to identifier $s \in S \cup I$):

$$\begin{aligned} \llbracket true \rrbracket(t) &= tt & \llbracket \neg e_1 \rrbracket(t) &= \neg \llbracket e_1 \rrbracket(t) \\ \llbracket false \rrbracket(t) &= ff & \llbracket s[o|c] \rrbracket(t) &= \begin{cases} \sigma(t+o) & \text{if } t+o \in \mathbb{T} \\ c & \text{else} \end{cases} \\ \llbracket e_1 \wedge e_2 \rrbracket(t) &= \llbracket e_1 \rrbracket(t) \wedge \llbracket e_2 \rrbracket(t) \\ \llbracket e_1 \vee e_2 \rrbracket(t) &= \llbracket e_1 \rrbracket(t) \vee \llbracket e_2 \rrbracket(t) \end{aligned}$$

The semantics of φ is well defined if no stream instant is dependent on itself. We only allow Lola specifications where this is guaranteed (which can be statically checked [12,33]).

3 Translating TRLTL_f to Lola

In this section, we describe how to translate TRLTL_f into Lola. More specifically, given a TRLTL_f formula φ , we derive a corresponding Boolean Lola specification \mathfrak{L}_φ with a distinguished stream for s_φ that is *true* at position i whenever the input word satisfies φ in position i . The input streams of the Lola specification are given by the atomic propositions. We will introduce one input stream variable t_p for each $p \in AP$. Given a word σ and an atomic proposition p , the input stream corresponding to t_p , $\tau_{t_p}(i)$, is true if and only if $p \in \sigma(i)$. Abusing notation, given a stream s and a set of streams S we use $s \cup S$ for $\{s\} \cup S$. We also use $(s = e) \cup E$ for $\{s = e\} \cup E$.

The idea of the translation is as follows. Given a TRLTL_f formula φ we create for each sub-formula ψ a fresh new stream s_ψ that captures the truth value of ψ at each position, depending on the truth value of the streams for its sub-expressions. For atomic propositions, the definition is immediate as it has to coincide with the corresponding proposition on the input word. Boolean combinations of subformulas translate to the corresponding Boolean combinations of the corresponding streams. The syntax for sequential operators take a regular expression (with its direction of the evaluation) followed by another formula. The regular expressions are first transformed linearly into their corresponding ϵ -NFA representation. Without loss of generality we assume that final states have no successor³. We will introduce a fresh stream variable for each state of the ϵ -NFA mimicking the evaluation of the automata followed by the continuing expression. Whenever a testing operator $\psi?$ is used within the regular expression, we refer to the stream variable s_ψ and take a transition in the NFA only if s_ψ is true at the current position. The power operator in TRLTL_f is translated similarly, according its unwinding law $\varphi|\alpha\rangle\psi \equiv \psi \vee (\varphi \wedge \alpha; \varphi|\alpha\rangle\psi)$.

Let us now formally describe the translation, which is given inductively by providing a transformer for each operator of TRLTL_f . Each transformer takes a subformula φ and delivers a pair $(s_\varphi, \mathfrak{L}_\varphi)$ where s_φ is the *distinguished* stream of the Lola specification \mathfrak{L}_φ . Therefore, we give the translation of φ as $(s_\varphi, \mathfrak{L}_\varphi)$ and only need to define \mathfrak{L}_φ as follows.

- For *true*, \mathfrak{L}_{true} is $(I, \{s_{true}\}, \{s_{true} = true\})$. For atomic propositions $p \in AP$, \mathfrak{L}_p is $(I, \{s_p\}, \{s_p = t_p[now]\})$. For \vee :

$$\mathfrak{L}_{\varphi \vee \psi} := (I, (s_{\varphi \vee \psi} \cup S_\varphi \cup S_\psi), (s_{\varphi \vee \psi} = s_\varphi \vee s_\psi) \cup E_\varphi \cup E_\psi)$$

Conjunction and negation can be processed in a similar manner. Basic expressions β are also inductively processed using conjunctions, disjunctions, complementation and atomic propositions.

- For $\alpha; \varphi$ with forward $\alpha = \rho$, let $(Q, q_0, \delta, \delta_e, F)$ be the ϵ -NFA accepting the language defined by α , obtained using standard constructions. In the regular

³ Every ϵ -NFA can be linearly transformed into such a representation by duplicating final states that have successors into two copies: one (final) with no successor and the other (non-final) with the successors.

expression, we treat any testing operator $\psi?$ as a single letter. To define $\mathfrak{L}_{\alpha;\varphi}$ we add a stream for each state of the automaton and one equation following the execution of the automaton. We use S_Q and E_Q for these sets of streams and equations. Let us first consider non-final states ($q \notin F$), for which the automaton in state q may choose a letter a to proceed to some next state (processing an input letter), may choose an ϵ -transition to proceed to some next state or may perform a check $\psi?$ (at the current input). The equation that we add for state $q \notin F$ is:

$$s_q = \bigvee_{(q,a,q') \in \delta} (s_a \wedge s_{q'}[+1|ff]) \vee \bigvee_{(q,q') \in \delta_\epsilon} s_{q'} \vee \bigvee_{(q,\psi?,q') \in \delta} (s_\psi \wedge s_{q'})$$

For final states $q \in F$, the formula φ has to be checked at the current state as the only possible continuation: $s_q = s_\varphi$. Finally, we add the equation for the distinguished stream $s_{\alpha;\varphi}$ as $s_{\alpha;\varphi} = s_{q_0}$, being true whenever a succesful run of the ϵ -NFA followed by the successful evaluation of φ is achieved by starting in the initial state.

$$\mathfrak{L}_{\alpha;\varphi} = (I, (s_{\alpha;\varphi} \cup S_Q \cup S_\varphi), (\{s_{\alpha;\varphi} = s_{q_0}\} \cup E_Q \cup E_\varphi))$$

- For $\alpha ; \varphi$ with backward $\alpha = \bar{\rho}$, we follow a similar construction, except that upon reading a letter the offset used to continue is -1 instead of $+1$. For $q \notin F$:

$$s_q = \bigvee_{(q,a,q') \in \delta} (s_a \wedge s_{q'}[-1|ff]) \vee \bigvee_{(q,q') \in \delta_\epsilon} s_{q'} \vee \bigvee_{(q,\psi?,q') \in \delta} (s_\psi \wedge s_{q'})$$

For final states $q \in F$, $s_q = s_\varphi$ and for the resulting specification

$$\mathfrak{L}_{\bar{\rho};\varphi} = (I, (s_{\bar{\rho};\varphi} \cup S_Q \cup S_\varphi), (\{s_{\bar{\rho};\varphi} = s_{q_0}\} \cup E_Q \cup E_\varphi))$$

- For $\varphi|\alpha\rangle\psi$ and a forward regular expression α , let $(s_\varphi, \mathfrak{L}_\varphi)$ be the translation of φ and $(s_\psi, \mathfrak{L}_\psi)$ the translation of ψ . Let also $(Q, q_0, \delta, \delta_\epsilon, F)$ be the ϵ -NFA for α . The equations for $s_{\varphi|\alpha}\psi$ follow the unwinding equivalence $\varphi|\alpha\rangle\psi \equiv \psi \vee (\varphi \wedge \alpha ; \varphi|\alpha)\psi$. For $\alpha ; (\varphi|\alpha)\psi$ we follow the construction for the sequential operator by adding streams for each state of the automaton and equations following the transitions. Non-final states are treated exactly as before. For final states $q \in F$ and for the distinguished stream:

$$s_q = s_{\varphi|\alpha}\psi \qquad s_{\varphi|\alpha}\psi = s_\psi \vee (s_\varphi \wedge s_{q_0})$$

Finally, $\mathfrak{L}_{\varphi|\alpha}\psi = (I, S_{\varphi|\alpha}\psi, E_{\varphi|\alpha}\psi)$ where

$$\begin{aligned} S_{\varphi|\alpha}\psi &= s_{\varphi|\alpha}\psi \cup S_Q \cup S_\varphi \cup S_\psi \\ E_{\mathfrak{L}_{\varphi|\alpha}\psi} &= (s_{\varphi|\alpha}\psi = s_\psi \vee (s_\varphi \wedge s_{q_0})) \cup E_Q \cup E_\varphi \cup E_\psi \end{aligned}$$

It is easy to see that the resulting Lola specification is linear in the length of the formula. The following result establishes the correctness of the translation, which can be formally shown by induction, following the inductive definition of the construction.

Lemma 2 (Correctness of Translation). *Let φ be a TRLTL_f formula and $(s_\varphi, \mathcal{L}_\varphi)$ be the corresponding Lola specification. Let $\sigma \in \Sigma^+$ be a word. Then, for all $i \in \{0, \dots, |\sigma|\}$, $(\sigma, i) \models \varphi$ if and only if $s_\varphi(i) = tt$.*

4 General Anticipatory Monitoring

In this section we develop an anticipatory algorithm for the recurrent monitoring problem of Lola specifications. Then, we will extend our algorithm to support assumptions and uncertainties. Our algorithm can be used for the initial monitoring problem as well, because, given a Lola specification $(s, (I, S, E))$ that we would like to use for initial monitoring, we can create $(r, (I, S \cup \{r\}, E'))$ where

$$E' = E \cup \{r = \text{if false}[-1|tt] \text{ then } s[\text{now}] \text{ else } r[-1|ff]\}$$

We use $r\langle i \rangle$ to denote the value of stream r at timepoint i . It is easy to see that at each point in time $r\langle i \rangle = s\langle 0 \rangle$. Therefore answering the question, at position i , of whether $r\langle i \rangle$ is true or false, is equivalent to answering whether $s\langle 0 \rangle$ is true or false. Thus, in the case of Lola, recurrent monitoring subsumes initial monitoring.

In the rest of the section to simplify the definitions we assume that $\varphi = (I, S, E)$ is an arbitrary well-defined Lola specification. We start with a general definition of recurrent monitors for Lola specifications.

4.1 Recurrent Monitors as Moore Machines

We first define the class of monitors for a Lola specification as Moore machines. These monitors will receive as inputs the values of the input streams and produce, at each instant, as output one Boolean verdict (or ?) per output stream.

Definition 1 (Moore Machine Monitor). *Given a Lola specification $\varphi = (I, S, E)$ a Moore Machine for φ is a tuple $M_\varphi = (P, \Sigma, \Omega, p_0, \delta_m, \omega)$ where*

- P is a set of states and $p_0 \in P$ is the initial state;
- $\Sigma = 2^I$ is the input alphabet;
- $\Omega = S \rightarrow \{tt, ff, ?\}$ is the output alphabet, that encodes one verdict per output stream;
- $\delta_m : P \times \Sigma \rightarrow P$ is the transition function;
- $\omega : P \rightarrow \Omega$ is the verdict function.

A monitor M_φ for a Lola specification φ is *sound* for output stream s if after processing an input string u (of length i), it produces tt only if for all continuations of u , the value of $s\langle i \rangle$ is tt (analogous for ff). Note that a sound monitor must produce ? if both tt and ff can be the result for $s\langle i \rangle$ depending on the continuation. Note also that a sound monitor is allowed to produce ?, even if the verdict is definite (in the extreme case, a monitor that always produces ? is sound).

4.2 An Anticipatory Algorithm

In general Lola specifications may contain future offsets, which potentially make stream events dependent on other streams at later instants. While in offline monitoring this is not a problem, as the full input word is already accessible, this poses a difficulty for online monitoring. The traditional online monitoring algorithm for Lola [12] tackles this problem by stalling computations, delaying the production of verdicts until the required values are available. This algorithm does not produce a value even when it is inevitable.

In this section we present an alternative recurrent monitoring for Lola as follows. A translation from Lola to DFA is presented in [8], which captures whether a sequence of input and output stream values matches a given Lola specification. Based on this construction, we transform a Lola specification into a labeled transition system, from which we build a perfect anticipatory monitor.

We will define a nondeterministic transition system, where the states encode (1) valuations of all (input and output) streams at the current instant and (2) guesses of the valuations at the next position. A valuation $v \in 2^{I \cup S}$ encodes which inputs and outputs are true, so states are pairs of valuations $2^{I \cup S} \times 2^{I \cup S}$. To encode the end of the input trace we use the symbol \perp , so the states are elements of $2^{I \cup S} \times (2^{I \cup S} \cup \{\perp\})$. Finally we also add an initial state $\#$ where no input letter has been received and thus no stream has a valuation yet, resulting in a state space

$$Q_\varphi \stackrel{\text{def}}{=} \{\#\} \cup (2^{I \cup S} \times (2^{I \cup S} \cup \{\perp\}))$$

Example 1. Consider for example the LTL formula $\varphi = \Box p \wedge \Diamond \neg p$. Following the translation from the previous section with some trivial simplifications (like inlining constant streams), the corresponding Lola specification would have an input stream s_p and four defined streams

$$\begin{aligned} s_{\neg p} &= \neg s_p[\text{now}] & s_{\Box p} &= s_p[\text{now}] \wedge s_{\Box p}[+1 \mid tt] \\ s_{\Diamond \neg p} &= s_{\neg p}[\text{now}] \vee s_{\Diamond \neg p}[+1 \mid ff] & s_\varphi &= s_{\Box p}[\text{now}] \wedge s_{\Diamond \neg p}[\text{now}] \end{aligned}$$

Some possible states of the transition system include

$$q_1 = (\{s_{\neg p}, s_{\Diamond \neg p}\}, \{s_p, s_{\Box p}\}) \quad q_2 = (\emptyset, \emptyset) \quad q_3 = (\{s_p, s_{\Box p}, s_{\Diamond \neg p}, s_\varphi\}, \perp)$$

State q_1 encodes the situation where s_p and $s_{\Box p}$ are false and $s_{\neg p}$ and $s_{\Diamond \neg p}$ are true in the current instant. In the subsequent instant $s_{\Box p}$ and s_p are true but the other streams are false. Note that in fact only q_1 is compatible with the Lola specification, but the other states are a contradiction to the equations of the specification. State q_2 is a contradiction because s_p and $s_{\neg p}$ cannot be false at the same time. State q_3 is a contradiction because when s_p is true at the last position of the trace, $s_{\Diamond \neg p}$ has to be false. \square

Based on this encoding we build a nondeterministic transition system, where the transition relation maps state (u, v) to state (v, w) when the first component of the post state coincides the second component of the pre-state (unless $v =$

\perp). The second component w of the post-state is not determined and different transitions can make different guesses.

Our transition system will have $\Sigma = 2^I$ as input alphabet, determining which input streams are true and which are false. Given an input $b \in 2^I$ and an element $(v, v') \in Q$ we write $v(b)$ when v coincides with b in the truth value of all input streams (i.e. $v \cap I = b$). Given $v, v', v'' \in 2^{I \cup S}$ we further say $(v, v') \models^i E$ if substituting every 0 offset with the value corresponding to v and every +1 offset with those corresponding to v' makes all equations E in the Lola specification φ true for position 0. We say $(v, v', v'') \models E$ if substituting all 0 offset operators with the value according v' , all -1 offsets with the value according v and all +1 offsets with the value according v'' makes all equations in E are satisfied. Finally we write $(v, v') \models^f E$ if -1 offset operators replaced by values according to v and 0 offset operators by those according to v' makes the equations in E satisfied for the last instant before the trace end.

Definition 2 (Lola Nondeterministic Transition System). Let $\varphi = (I, S, E)$ be a well-defined Lola specification. The Lola nondeterministic transition system (LNTS) for φ is a tuple $\mathcal{T}_\varphi = (Q_\varphi, \Sigma, q_0, \delta)$, where $q_0 = \#$ and

$$\begin{aligned} - \delta(\#, b) &= \{(v, v') \mid v(b) \text{ and } (v, v') \models^i E\} \\ - \delta((u, v), b) &= \begin{cases} \{(v, v') \mid (u, v, v') \models E\} \cup \{(v, \perp) \mid (u, v) \models^f E\} & \text{if } v(b) \\ \emptyset & \text{otherwise} \end{cases} \end{aligned}$$

The transition relation first checks that the guessed successor v is compatible with inputs received and then guesses a new successor state v' that satisfies all equations E of the specification (including the possibility of guessing \perp denoting the end of the trace).

Given a tuple of input streams $\tau \in (\mathcal{S}_{\mathbb{B}})^{|I|}$ and a well-defined Lola specification φ , there is a unique state sequence $q_0, q_1, \dots, q_{|\tau|}$ such that $q_{i+1} \in \delta(q_i, \tau(i))$ and $q_{|\tau|} = (v, \perp)$. This follows from the fact that a well-defined Lola specification has a unique valuation (given the whole input sequence).

The LNTS \mathcal{T}_φ allows to define a sequence of stream valuations for the current time instant, which is consistent with the inputs received so far and with φ . To build an anticipatory monitor for φ we determinize \mathcal{T}_φ by applying the following two stages:

Removing Dead States. First we remove all states from the transition system from which a state with \perp in the second component is not reachable, which corresponds to a situation where a wrong guess was made and that cannot be completed, no matter of the future inputs. Dead states can be identified by a depth-first search in \mathcal{T}_φ starting at the initial state. We therefore limit the state space Q_φ to $\{q \in Q_\varphi \mid \exists w \in \Sigma^*, (q', \perp) \in \widehat{\delta}(q, w)\}$ where $\widehat{\delta}$ is defined as $\widehat{\delta}(q, \epsilon) = q$ and $\widehat{\delta}(q, aw) = \widehat{\delta}(\delta(q, a), w)$.

Example 2. Consider again the specification for $\varphi = \Box p \wedge \Diamond \neg p$ and the states $q_1 = (\{s_{\neg p}, s_{\Diamond \neg p}\}, \{s_p, s_{\Box p}\})$ and $q_2 = (\{s_p, s_{\Box p}, s_{\Diamond \neg p}, s_\varphi\}, \{s_p, s_{\Box p}, s_{\Diamond \neg p}, s_\varphi\})$. From s_1 the state $(\{s_p, s_{\Box p}\}, \perp)$ is reachable, because the sub-formula $s_{\Box p}$ is

satisfied at the trace end iff s_p is satisfied. Thus q_1 is alive. On the other hand q_2 is dead, because s_p , $s_{\square p}$ and $s_{\diamond \neg p}$ being true at some instant imply that these three streams (and thus also s_φ) are also true at the next instant. Consequently q_2 is the only possible successor of q_2 and especially $(\{s_p, s_{\square p}, s_{\diamond \neg p}, s_\varphi\}, \perp)$ is not a valid successor. \square

Determinize. Second, we use a power set construction to determinize the resulting transition system⁴. If the monitor is in a power state which corresponds to states in the original transition system that have different values for an output stream, then the monitoring output is ? for this stream. On the other hand, if all states in the power set agree on the valuation of an output stream, the monitor yields exactly this valuation. In particular, the distinguished stream receives either the valuation ? or a definite valuation tt or ff . Formally, the resulting monitor is defined as follows, where $s(q)$ denotes the current valuation of stream $s \in I \cup S$ in state q .

Definition 3 (Anticipatory Recurrent Lola monitor). Let $\varphi = (I, S, E)$ be a well-defined Lola specification and $\mathcal{T}_\varphi = (Q_\varphi, \Sigma, q_0, \delta)$ be the corresponding LNTS with dead states removed. The anticipatory recurrent Lola monitor for φ is the Moore Machine Monitor $M_\varphi : (P, \Sigma, \Omega, p_0, \delta_m, \omega)$ where $P = 2^{Q_\varphi}$, $p_0 = \{q_0\}$ and

$$\begin{aligned} - \delta_m(p, x) &= \{\delta(q, x) \mid q \in p\} \cap Q_\varphi \\ - \omega(p)(s) &= \begin{cases} tt & \text{if } s(q) = tt \text{ for all } q \in p \\ ff & \text{if } s(q) = ff \text{ for all } q \in p \\ ? & \text{otherwise} \end{cases} \end{aligned}$$

An *anticipatory* monitor is a sound monitor for all output streams that produces tt if and only if the evaluation of the output stream at this position is tt for all continuations, i.e. when the verdict is inevitable and the analogous for ff . Such a monitor thus only yields ? when both results are possible in different continuations.

The following result proves the correctness of our construction.

Theorem 1. Let $\varphi = (I, S, E)$ be a well-defined Lola specification and M_φ the monitor according to Definition 3. Then M_φ is an anticipatory recurrent monitor for φ .

The proof follows because M_φ only contains states that can lead to end states, so for every power state in M_φ there is a continuation of the input streams such that the equations in φ are satisfied. On the other hand the power set contains all non-dead states which are reachable from q_0 after processing the input received. A tt (resp. ff) verdict for a specific stream is cast if and only if all states agree on that valuation and thus there is no continuation of the currently received inputs compatible with a different verdict. It is easy to see that the size of M_φ is $2^{2^{\mathcal{O}(|\varphi|)}}$.

⁴ which can for performance reasons also be done on the fly while monitoring.

4.3 Assumptions

Assumptions are additional knowledge about the system and its environment and thus restrict the set of possible input sequences that may be passed to the monitor. A general way to formalize assumptions in Lola [20] is to introduce an output stream s_a expressing the assumption and assume the stream to be always true. We rule out those states where s_a is false, or states that inevitably lead to those states. We restrict the state space to states where the assumption stream is true $Q_a \stackrel{\text{def}}{=} \{(u, v) \in Q_\varphi \mid s_a \in u, \text{ and } v = \perp \text{ or } s_a \in v\}$ and refine the definition of alive states to $\text{alive}_a^\varphi = \{q \in Q_\varphi \mid \exists w \in \Sigma^*, (q', \perp) \in \widehat{\delta}_a(q, w)\}$ where $\widehat{\delta}_a$ is defined as $\widehat{\delta}_a(q, b) = \delta(q, b) \cap Q_a$ and $\widehat{\delta}_a(q, bw) = \widehat{\delta}_a(\widehat{\delta}_a(q, b), w)$, that is, $\widehat{\delta}_a$ is like $\widehat{\delta}$ but only considers successor states that satisfy the assumption. We define a recurrent Lola monitor with assumptions, which differs from the monitor of Definition 3 by considering an advanced set of dead states.

Definition 4 (Recurrent Lola monitor with Assumptions). *Let $\varphi = (I, S, E)$ be a well-defined Lola specification and let $\mathcal{T}_\varphi = (Q_\varphi, \Sigma, q_0, \delta)$ be the corresponding LNTS with dead states (not in alive_a^φ) removed. The anticipatory recurrent Lola monitor for φ under assumptions is the Moore machine $(P, \Sigma, \Omega, p_0, \delta_m, \omega)$ where $P = 2^{Q_a}$ and $p_0 = \{q_0\}$ and $\delta_m(p, b) = \{\delta(q, b) \mid q \in p\} \cap Q_a$ (and Σ, Ω and ω are as before).*

Note that $\omega(p)(s_a)$ is forced to be tt at all states and that the new Moore machine has fewer states compared to the previous construction.

4.4 Uncertainties

We now extend our approach to tolerate uncertain inputs, where the value of some input is not known to be true or false. Instead of input alphabet Σ we consider 2^Σ as uncertain input alphabet where each letter encodes which certain input letters are possible at the current instant. Consider $AP = \{p, q\}$ and $\Sigma = 2^{AP}$, then input $\{\emptyset, \{p, q\}\} \in 2^\Sigma$ encodes that it is uncertain if p, q hold but it is known that they have the same value.

In the recurrent monitor with assumptions (Definition 4) we just extend the transition function such that from a set of specific states it transitions to all states which are reachable with one of the possible inputs:

Definition 5 (Recurrent Lola Monitor with Uncertainty and Assumptions). *Let $\varphi = (I, S, E)$ be a well-defined Lola specification and $\mathcal{T}_\varphi = (Q_\varphi, \Sigma, q_0, \delta)$ be the corresponding LNTS. The recurrent Lola monitor under uncertainty and assumptions for φ is the Moore machine $(P, 2^\Sigma, \Omega, p_0, \delta_m, \omega)$ where P, p_0, Ω and ω are as before and $\delta_m(p, B) = \{\delta(q, b) \mid q \in p, b \in B\} \cap Q_a$.*

Note that δ_m considers all possible inputs, potentially leading to more successors.

5 Anticipatory Monitoring in Action

We implemented the algorithm for anticipatory Lola monitoring from Section 4 in Scala⁵. The tool receives a Lola specification, calculates the set of empty states and then simulates the power set monitor on the fly as described in Section 4 with minor obvious optimizations. It supports assumptions and uncertainty.

We illustrate monitoring of the following TRLTL_f formula that includes past and future operators (encoded linearly):

$$\varphi = p \wedge \Box((\neg p); p \rightarrow \Diamond(q; p; p; p; (\neg q)^*; q))$$

The formula holds in every position where (1) p is true and (2) if for all subsequent positions matching $(\neg p); p$, the pattern $q; p; p; p; (\neg q)^*; q$ was present somewhere in the past.

Following Section 3 we manually transformed the formula into a Lola specification with two input streams, p and q , ten defined streams, seven future references and one past reference. A traditional universal Lola monitor [12] would only immediately yield the verdict ff at all positions where there is no p in the trace. All the locations where φ holds would only be reported after the whole trace is processed, because the \Box part of the formula introduces a future reference, so a monitoring algorithm without anticipation only resolves these streams once the end of the trace is reached.

We evaluated our anticipatory monitoring approach on three randomly generated traces of length 1000. For each position, first q was selected to be true with a probability of 66%. If q was false then p was set to true, otherwise p was set to true with a probability of 50%. Consequently there were no positions in the traces where p and q were simultaneously false.

We ran our monitor for each trace, one time with the additional assumption $\Box(p \vee q)$ in the specification and one time without. Further we executed the monitor under presence of the assumption, but total uncertain information about p (i.e. $p = ?$ was sent to the monitor at all instants). The numbers of tt resp. ff verdicts are depicted in the following table:

Trace	Offline Monitor		Rec. Ant. Monitor		+ Assumption		+ Uncertainty	
	tt	ff	tt	ff	tt	ff	tt	ff
1	650	350	644	349	646	349	311	0
2	651	349	647	339	649	339	307	0
3	659	341	655	337	656	337	286	0

The first column shows the number of positions where φ is satisfied, which corresponds to the output an offline monitor with full knowledge of the whole trace would yield. The recurrent anticipatory monitor is able to cast final verdicts as soon as it detects a sequence $q; p; p; p; (\neg q)^*; q$ in the trace, because from then

⁵ Tool and example are available on https://gitlab.isp.uni-luebeck.de/public_repos/anticipatory-recurrent-artifact

on $\diamond(q; p; p; p; (\neg q)^*; q)$ and thus $\Box((\neg p); p \rightarrow \diamond(q; p; p; p; (\neg q)^*; q))$ is satisfied. Hence the monitor only reports a few ? verdicts at the beginning of the trace.

When the assumption $\Box(p \vee q)$ is present, the recurrent monitor already yields final verdicts after receiving the sequence $q; p; p; p$, because from this moment on it can conclude that whenever the premises of the implication inside the globally operator, $(\neg p); p$, holds, then there is a $(\neg p)$ in the trace and consequently q holds at this position. This however implies that the trace also contains a sequence matching $q; p; p; p; (\neg q)^*; q$. This is why the recurrent monitor with assumption yields a slightly higher number of certain verdicts.

If p is fully uncertain, the monitor can not directly check anymore whether $q; p; p; p$ is contained in the trace. Yet, again using the assumption it can conclude that if $q; (\neg q); (\neg q); (\neg q)$ holds somewhere in the trace, then also $q; p; p; p$ holds there and thus $\Box((\neg p); p \rightarrow \diamond(q; p; p; p; (\neg q)^*; q))$ is satisfied from that instant on. Hence, from that position on, the monitor is able to cast *tt* whenever q is false, because at these instants p must be true. However, it cannot give verdicts at positions where q is true, including all positions where p is false, and thus never produces *ff*. Note that a traditional (online or offline) Lola monitor without the ability of handling assumptions would not be able to cast any certain verdicts under presence of the mentioned uncertainty.

We ran our examples on a Linux machine with 8GB RAM and Intel Core i7-8550 U (1.80GHz) CPU. The average time spent for the emptiness check was 2227 ms without assumption and 2156 ms with assumption. The processing time per event (without I/O handling) was on average, 2.22 ms without assumption, 1.84 ms with assumption and 5.60 ms under additional presence of uncertainty.

6 Final Remarks

Anticipation states that an online monitor should emit a precise verdict as soon as possible with the information received. This was first introduced for infinite traces [4,6] for LTL and for timed, event-clock extensions of LTL, and later generalized to all formalisms definable by Büchi automata in [13]. In [23] anticipation is made more precise if part of the underlying system is known. All these works consider only initial monitoring. In [18], recurrent monitoring was studied for past-only LTL always yielding a verdict for the current position in the trace. A generalization of the concept for future LTL formulas—but without an explicit construction—was studied in [19] together with assumption and uncertainties (meaning imprecise or missing inputs).

In recent years, temporal logics for finite traces have gained importance so it is a natural question how the concept of recurrent monitoring under uncertainties and assumptions materializes in the finite setting.

In this paper we addressed this question for the Boolean fragment of the SRV language Lola, which is a very general formalism encompassing many temporal logics on finite traces. We showed how many temporal logics on finite traces can be linearly translated into TRLTL_f , which we then translated into Lola. The logic TRLTL_f introduced here simply takes all operands from RLTL [24] and

LDL_f [16]. Since LDL_f and $RLTL_f$ are both expressive equivalent to regular languages they are equivalent in terms of expressive power. We are studying whether there are linear translations from LDL_f to $RLTL_f$ and vice-versa, following reductions from [31] in the context of Metric Dynamic Logic (MDL) [3,2]. MDL extends LDL_f with intervals and has the same expressive power.

We then presented a recurrent, anticipatory monitoring algorithm for Lola specifications, extended it to handle uncertainties and assumptions, and pointed out how to map initial monitoring into recurrent monitoring for Lola.

The approach most closely related to ours is [9], which also considers monitoring under uncertainties and assumptions. However, [9] is limited to LTL and assumptions given as fair Kripke structures only.

We restricted our algorithm to Boolean Lola, so it is a natural question how to deal with anticipation, assumptions, and uncertainties for specifications over arbitrary theories. While assumptions and uncertainties for non-Boolean theories are studied in [20], anticipation is not considered there. A solution for LTL extended with theories, presented in [10], is based on reduction to bounded model checking and thus not guaranteed to be perfect and not trace-length independent.

The problem of anticipatory general Lola monitoring can also be solved easily if the length of the trace is known a-priori using a bounded-model-checking approach, by unwinding the specification to the known length and using a symbolic tool (e.g. an SMT solver) to compute definite verdicts with anticipation. Future work includes anticipatory monitoring algorithms for richer data specifications (like numerical Lola specifications) without assuming a known bound on the length of the trace.

Future work further comprises an implementation of the translations from logics to Lola specifications that were described in this paper and a thorough empirical comparison to other monitoring approaches for these logics and Lola in practical scenarios.

Acknowledgements

We would like to thank the anonymous reviewers for the thorough analysis of the paper and their useful suggestions and future directions.

References

1. Bartocci, E., Falcone, Y. (eds.): Lectures on Runtime Verification - Introductory and Advanced Topics, LNCS, vol. 10457. Springer (2018). <https://doi.org/10.1007/978-3-319-75632-5>
2. Basin, D., Bhatt, B.N., Krstić, S., Traytel, D.: Almost event-rate independent monitoring. *Formal Methods in System Design* **54**, 449–478 (2019). <https://doi.org/10.1007/s10703-018-00328-3>
3. Basin, D.A., Krstić, S., Traytel, D.: Almost event-rate independent monitoring of metric dynamic logic. In: Proc. of the 17th Int'l Conf. on Runtime Verification

- (RV'17). LNCS, vol. 10548, pp. 85–102. Springer (2017). https://doi.org/10.1007/978-3-319-67531-2_6
4. 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
 5. 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>
 6. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. *ACM Trans. Softw. Eng. Methodol.* **20**(4), 14:1–14:64 (2011). <https://doi.org/10.1145/2000799.2000800>
 7. 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 (Part II). LNCS, vol. 12225, pp. 28–39. Springer (2020). https://doi.org/10.1007/978-3-030-53291-8_3
 8. Bozzelli, L., Sánchez, C.: Foundations of boolean stream runtime verification. *Theor. Comput. Sci.* **631**, 118–138 (2016). <https://doi.org/10.1016/j.tcs.2016.04.019>
 9. Cimatti, A., Tian, C., Tonetta, S.: Assumption-based runtime verification with partial observability and resets. In: Proc. of the 19th Int'l Conf. on Runtime Verification (RV'19). LNCS, vol. 11757, pp. 165–184. Springer (2019). https://doi.org/10.1007/978-3-030-32079-9_10
 10. 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
 11. 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
 12. 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: Proc. of the 12th Int'l Symposium on Temporal Representation and Reasoning (TIME'05). pp. 166–174. IEEE Computer Society (2005). <https://doi.org/10.1109/TIME.2005.26>, <https://doi.org/10.1109/TIME.2005.26>
 13. Dong, W., Leucker, M., Schallhart, C.: Impartial anticipation in runtime-verification. In: Proc. of the 6th Int'l Symp. on Automated Technology for Verification and Analysis (ATVA'08). LNCS, vol. 5311, pp. 386–396. Springer (2008). https://doi.org/10.1007/978-3-540-88387-6_33
 14. Eisner, C., Fisman, D., Havlicek, J., Lustig, Y., McIsaac, A., Campenhout, D.V.: Reasoning with temporal logic on truncated paths. In: Proc. of the 15th Int'l Conf. on Computer Aided Verification (CAV'03). LNCS, vol. 2725, pp. 27–39. Springer (2003). https://doi.org/10.1007/978-3-540-45069-6_3
 15. 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

16. Giacomo, G.D., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: Proc. of the 23rd Int'l Joint Conf. on Artificial Intelligence (IJ-CAI'13). pp. 854–860. IJCAI/AAAI (2013), <http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6997>
17. 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>
18. Havelund, K., Rosu, G.: Synthesizing monitors for safety properties. In: Proc. of 8th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02). LNCS, vol. 2280, pp. 342–356. Springer (2002). https://doi.org/10.1007/3-540-46002-0_24
19. Kallwies, H., Leucker, M., Sánchez, C., Scheffel, T.: Anticipatory recurrent monitoring with uncertainty and assumptions. In: Proc. of the 22nd Int'l Conference on Runtime Verification (RV'22). LNCS, vol. 13498, pp. 181–199. Springer (2022). https://doi.org/10.1007/978-3-031-17196-3_10
20. Kallwies, H., Leucker, M., Sánchez, C.: Symbolic runtime verification for monitoring under uncertainties and assumptions. In: Proc. of 20th Int'l Symp. on Automated Technology for Verification and Analysis (ATVA'22). LNCS, vol. 13505, pp. 117–134. Springer (2022). https://doi.org/10.1007/978-3-031-19992-9_8
21. Kleene, S.C.: Representation of events in nerve nets and finite automata. In: Shannon, C.E., McCarthy, J. (eds.) *Automata Studies*, vol. 34, pp. 3–41. Princeton University Press, Princeton, New Jersey (1956)
22. Leucker, M.: Teaching runtime verification. In: Proc. of the 2nd Int'l Conf. on Runtime Verification (RV'11). LNCS, vol. 7186, pp. 34–48. Springer (2011). https://doi.org/10.1007/978-3-642-29860-8_4
23. Leucker, M.: Sliding between model checking and runtime verification. In: Proc. of the 3rd Int'l Conf. on Runtime Verification (RV'12). LNCS, vol. 7687, pp. 82–87. Springer (2012). https://doi.org/10.1007/978-3-642-35632-2_10
24. Leucker, M., Sánchez, C.: Regular linear temporal logic. In: Proc. of the 4th Int'l Colloquium on Theoretical Aspects of Computing (ICTAC'07). LNCS, vol. 4711, pp. 291–305. Springer (2007). https://doi.org/10.1007/978-3-540-75292-9_20
25. Leucker, M., Schallhart, C.: A brief account of runtime verification. *J. Logic Algebr. Progr.* **78**(5), 293–303 (2009). <https://doi.org/10.1016/j.jlap.2008.08.004>
26. Manna, Z., Pnueli, A.: *The Temporal Logic of Reactive and Concurrent Systems*. Springer, New York (1992)
27. Manna, Z., Pnueli, A.: *Temporal Verification of Reactive Systems*. Springer-Verlag (1995)
28. McNaughton, R.F., Yamada, H.: Regular expressions and state graphs for automata. *IEEE Transactions on Electronic Computers* **9**, 39–47 (1960). <https://doi.org/10.1109/TEC.1960.5221603>
29. Perez, I., Dedden, F., Goodloe, A.: Copilot 3. Tech. Rep. NASA/TM–2020–220587, NASA Langley Research Center (April 2020)
30. Pnueli, A.: The temporal logic of programs. In: Proc. of the 18th IEEE Symp. on the Foundations of Computer Science (FOCS'77). pp. 46–57. IEEE Computer Society Press (1977). <https://doi.org/10.1109/SFCS.1977.32>
31. Raszyk, M.: *Efficient, Expressive, and Verified Temporal Query Evaluation*. Ph.D. thesis, ETH (2022). <https://doi.org/10.3929/ethz-b-000553221>
32. 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
33. 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
 34. Sánchez, C., Leucker, M.: Regular linear temporal logic with past. In: Proc. of the 11th Int'l Conf. on Verification, Model Checking, and Abstract Interpretation, (VMCAI'10). LNCS, vol. 5944, pp. 295–311. Springer (2010). https://doi.org/10.1007/978-3-642-11319-2_22
 35. Sánchez, C., Samborski-Forlese, J.: Efficient regular linear temporal logic using dualization and stratification. In: Proc. of the 19th Int'l Symp. on Temporal Representation and Reasoning (TIME'12). pp. 13–20. IEEE Computer Society (2012). <https://doi.org/10.1109/TIME.2012.25>
 36. Wolper, P.: Temporal logic can be more expressive. *Information and Control* **56**, 72–99 (1983). [https://doi.org/10.1016/S0019-9958\(83\)80051-5](https://doi.org/10.1016/S0019-9958(83)80051-5)