General Anticipatory Runtime Verification*

Raik Hipler[1], Hannes Kallwies[1], Martin Leucker[1], and César Sánchez[2]

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

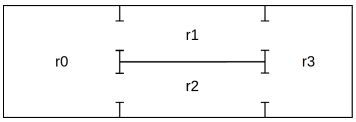
Abstract. Runtime verification is a technique for monitoring a system's behavior against a formal specification. Monitors must produce verdicts that are sound with respect to the specification. Anticipation is the ability to immediately produce verdicts when the monitor can confidently predict the inevitability of the verdict.
Stream runtime verification is a specialized form of runtime verification tailored to the monitoring and verification of data streams. In this paper we study anticipatory monitoring for stream runtime verification. More specifically, we present an algorithm with anticipation for monitoring of Lola specifications, which we then extend to exploit assumptions and tolerate uncertainties. As perfect anticipation is in general not computable, we use techniques from abstract interpretation, especially widening, to approximate anticipatory monitoring verdicts. Finally, we report on three empirical cases studies using a prototype implementation of a symbolic instantiation of our approach.

1 Introduction

In its simplest definition, *runtime verification (RV)* [26] solves the word problem: whether a certain property (for example, expressed as an LTL formula) is satisfied for a system run, given the run or a prefix of it. In recent years, advanced RV paradigms have emerged, such as *stream runtime verification (SRV)*, extending the traditional notion of runtime verification. First, SRV allows computations and outputs over arbitrary data domains, not only atomic Boolean propositions and verdicts like for LTL. Second, they specify "point-wise properties", which assign outputs to every position of the trace (instead of a single verdict for the trace as a whole). This is especially useful to identify points in a trace, e.g. an error location.

Fig. 1 shows an SRV specification in the pioneering formalism Lola [9] (see Section 2), which we will use as a running example. The scenario models a

* 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—by DECO Project (PID2022-138072OB-I00)—funded by MCIN/AEI/10.13039/501100011033 and by the ESF+—and by a research grant from Nomadic Labs and the Tezos Foundation.



In $r_{i\in\{0,1,2,3\}}$ **: Bool;** e**: Real**

Def $err \;:= \neg r_0 \wedge (e < 5)$
Def $Ferr \;:= err[now] \vee Ferr[+1 \mid false]$

Fig. 1: Example Lola specification and room map for a vacuum cleaner robot.

vacuum cleaner robot in a house with four rooms, connected by open doors. The charging station is located in room r_0. We want to check the following property: "*The robot may not enter rooms if its battery is not charged enough to be able to reach the base station.*" The Lola specification for this property defines four input streams of type Boolean $r_0, \ldots, r_3$ and one stream e of type real. A stream is a sequence of data values over time. The input streams originate from the robot system and are incrementally passed to the monitor. The values (events) of streams $r_0, \ldots, r_3$ encode the current location of the robot while e contains the battery charge (between 0% and 100%). The Lola specification defines a Boolean output stream *err* that defines the error: the robot is not in room 0 and its battery has run below 5%. Output streams contain events in synchrony with input streams, so the values at each input stream instant produce a value of *err*, revealing whether the system has run into an error. That is, this specification is a point-wise property. The specification also defines *Ferr*, which is true if either *err* is true now or in the future (by referring to *Ferr* at the next instant, with default value false at the trace end).

Monitoring can be performed *online* or *offline*. In offline monitoring the input trace is completely known upfront, for example as a log file. On the other hand, an online monitor receives the trace event by event while the observed system is running. In this paper we deal with online runtime verification. There is a significant difference between both kinds of monitoring when the specification contains future references (as stream *Ferr* above). Future references are not a problem in offline RV, because future input values can be easily accessed, but future values are unknown in online monitoring. In general there are two strategies for future references in online RV: (1) stalling calculations until all relevant input events are accessible [9]; (2) cast at each step an output as precise as possible with the information available (e.g. a set/interval of possible values). For a Boolean stream these outputs could be $\top = \{tt\}$, $\bot = \{ff\}$ or $? = \{tt, ff\}$ when both values are possible, depending on future inputs. This strategy is used in LTL_3 monitoring [2], but not for point-wise properties. The online monitoring of point-wise properties—while emitting the best possible sets of valuations—is called *perfect recurrent monitoring* in [21,20].

A stream being defined using future references does not necessarily imply that a ? verdict has to be cast. Consider *Ferr* above: if the value of *err* is true at some instant then *Ferr* is true now, independently of future events. Moreover, additional knowledge about the monitored system available in the form of *assumptions* [18,4,24] allows to reduce the set of possible valuations. Consider again our running example and assume that the robot consumes 3%

of energy when passing from one room to the next. We may conclude that room 0 is not reachable without dropping below 5% battery before (and thus *Ferr* is *true*), if the robot is in room 3 with an energy level below 8%. This kind of monitoring is called *anticipatory* [3].

In this paper we study the problem of anticipatory monitoring for Lola under assumptions and also uncertainties (missed or imprecise sensor values) in the input trace. While for a propositional logic, whether a prefix satisfies or violates a property in all continuations can be modeled using (Büchi) automata, whose emptiness can be effectively determined, the problem is more complex for richer domains. They require reasoning about satisfiability and validity in richer theories—which are computationally expensive or even undecidable—and require reasoning about all futures as finite formulas (instead of automata).

Related Work. Early RV research focused mostly on the monitoring of LTL [29] properties. The LTL_3 monitoring approach [2] was the first to consider anticipation, by reasoning about all possible trace continuations. More expressive RV formalisms were later introduced adding notions of time or complex data values in the traces. Examples include signal temporal logic (STL) [27], mission time LTL [31], Eagle [14] or metric first order temporal logic (MFOTL) [1]. A prominent class of extended RV approaches is SRV, pioneered by Lola [9], and later extended in asynchronous languages like RTLola [11,12], TeSSLa [6,22] and Striver [15,16]. Many RV formalisms can be encoded in Lola [20]. Recurrent monitoring [21] was first studied in [17] for past LTL and later extended with resets [4,5], and also for Lola [20]. The use of symbolic representations for monitoring (also to handle uncertainty) has recently been studied [10,34,4,5,13] and also applied to Lola [19]. Considering assumptions during monitoring was first proposed in [24] (under different wording) and later successfully adapted and extended [4,5,35,19]. The topic is theoretically studied in [18]. The approach that we present in this paper is based on the theory of abstract interpretation [8,7], which was used in RV to handle uncertainties in [25].

The works closest to this paper are [5] and [13] which study symbolic anticipatory LTL monitoring with linear arithmetic sub-formulas. The former [5] also considers uncertainties and assumptions.

In this paper we first introduce variations of the original Lola semantics: We give *monitoring semantics* which define the perfect monitoring results for uncertain stream prefixes. Based on this we define the *instant* and then (more importantly) *transformer semantics*, which also capture perfect monitoring outputs but discard unnecessary information about relations to all past and future events and can be deterministically computed. We then introduce a general abstraction framework for the effective computation of the transformer semantics and derive an efficient, anticipatory Lola monitoring algorithm. Provided with a sound or perfect abstraction for the stream values (e.g. one from the various literature on abstract interpretation) we present a general algorithm to monitor Lola specifications with future references. We give a criterion for the existence of perfect monitoring, and present a technique based on widening to produce a sound

monitor if perfect monitoring is impossible. Then, we instantiate our general framework for linear real arithmetic specifications using symbolic computation. Finally, we report on an empirical evaluation of a prototype implementation of our approach on three complex case studies.

Contributions. Compared to previous works (esp. [5] and [13]) the main contributions of our approach are:

- The anticipated monitor outputs may be of richer data types than Boolean.
- The monitor is able to produce arbitrarily many outputs per time step.
- Instead of unrolling a specification from the beginning to handle anticipation, we unroll from the back until an invariant is found which is then used to efficiently look ahead during the actual monitoring.
- If no perfect anticipation exists, we provide sound over-approximations instead.
- We are not restricted to symbolic reasoning but provide a general abstraction-based monitoring framework.

2 Lola Monitoring revisited

2.1 Recurrent Monitoring

Recurrent monitoring starts from a point-wise property, which assigns to every position of a trace a *valuation*. Traditionally, valuations are Boolean or other truth domains [33]. Here, we consider valuations from an arbitrary data domain.

Definition 1 (Point-wise property). *A* point-wise property $\mathcal{P}$ *of words of length* n *over domain* Γ *into domain* $\mathbb{D}$ *is a function* $\mathcal{P} : \Gamma^n \times \{1, 2, \ldots, n\} \to \mathbb{D}$.

In online monitoring of point-wise properties, the input $w \in \Gamma^n$ is not available at once but provided incrementally, and the monitor produces an output after each input letter. A monitor may output several possible values from $\mathbb{D}$, which in practice is encoded as an interval or ? (for all values). We identify a monitor with its characteristic function $\overline{M} : \Gamma^{\leq n} \to 2^{\mathbb{D}}$ which maps prefixes of inputs to sets of possible outputs. After the first k letters of the input, a recurrent monitor [21] tries to evaluate the corresponding property at position k. A *sound* recurrent monitor outputs a super set of the possible verdicts at the current instant (compatible with all possible future input continuations). The monitor is *perfect* if it casts exactly the set of possible property valuations.

Definition 2 (Sound/perfect recurrent monitor). *Given a point-wise property* $\mathcal{P}$ *and a non-empty input prefix* $w \in \Gamma^{\leq n}$*, the set of possible verdicts after* w *is* $pos(w) = \{\mathcal{P}(wv, |w|) \mid v \in \Gamma^{n-|w|}\}$*. A recurrent monitor* M *for* $\mathcal{P}$ *is sound whenever for every* w*,* $\overline{M}(w) \supseteq pos(w)$*.* M *is perfect if* $\overline{M}(w) = pos(w)$*.*

2.2 Lola

A Lola specification defines a transformation from a tuple of input streams to a tuple of output streams. A finite stream of type $\mathbb{D}$ over a time domain $\mathbb{T} = \{0, 1, \ldots, t_{max}\}$ is a function $s : \mathcal{S}_{\mathbb{D}} := \mathbb{T} \to \mathbb{D}$ that assigns a data value to every instant in $\mathbb{T}$. In this work we fix t_{max} and thus $\mathbb{T}$. We use sequences to represent streams and their prefixes. Given $s = \langle 3, 4, 2 \rangle$ we use $s(0) = 3$, $s(1) = 4$, $s(2) = 2$.

A Lola specification [9] is given as an equation system, which defines output streams in terms of input and other output streams. The set of Lola expressions over a set of stream identifiers S, $Expr_S$, is recursively defined as

$$Expr_S := c \mid f(Expr_S, \ldots, Expr_S) \mid s[o|c]$$

where $s \in S$ is a stream identifier, c a constant value, f a function symbol, and $o \in \mathbb{Z}$ is an integer offset. A constant expression is interpreted as a stream with that constant value at all instants; a function application as the stream which results from the application of the function on the argument stream events at every instant. The operator $s[o|c]$, called the offset operator, describes a stream which carries the values of stream s, shifted o instants. To refer to past events o can be chosen to be negative. If the accessed instant does not exist because it is beyond the trace ends (beginning or end) the default value c is used instead. For offset operators with offset 0, the default value does not play a role, thus we use the notation $s[now]$ or simply s for $s[0|c]$ for arbitrary constant c.

Syntax. A Lola specification $\varphi = (I, S, E)$ is a 3-tuple where I is a finite set of input stream identifiers; S is finite set of output stream identifiers with $I \cap S = \emptyset$; $E : S \to Expr_{I \cup S}$ assigns a defining expression to every output steam. For the rest of the paper, we assume that specifications are *flat*, i.e. they only contain offsets $-1, 0, +1$. Every specification can be flattened by introducing additional streams and splitting greater offsets to a chain of ± 1 offsets.

Semantics. The formal semantics of a Lola specification $\varphi = (I, S, E)$ with input streams $I = \{i_1, \ldots, i_n\}$ and output streams $S = \{s_1, \ldots, s_m\}$ maps a tuple of concrete input streams to the corresponding tuple of concrete output streams as follows. Given a tuple of input streams $\Sigma = (\sigma_1, \ldots, \sigma_n)$ the semantics $[\![e]\!]_\Sigma \in \mathcal{S}_{\mathbb{D}}$ of an expression $e \in Expr_{I \cup S}$ of type $\mathbb{D}$ is:

- $[\![c]\!]_\Sigma(t) = c$
- $[\![f(e_1, \ldots, e_n)]\!]_\Sigma(t) = f([\![e_1]\!]_\Sigma(t), \ldots, [\![e_n]\!]_\Sigma(t))$
- $[\![i_j[o|c]]\!]_\Sigma(t) = \begin{cases} \sigma_j(t+o) & \text{if } t+o \in \mathbb{T} \\ c & \text{otherwise} \end{cases}$
- $[\![s_j[o|c]]\!]_\Sigma(t) = \begin{cases} [\![E(s_j)]\!](t+o) & \text{if } t+o \in \mathbb{T} \\ c & \text{otherwise} \end{cases}$

The semantics of φ, $[\![\varphi]\!] : \mathcal{S}_{\mathbb{D}_1} \times \cdots \times \mathcal{S}_{\mathbb{D}_n} \to \mathcal{S}_{\mathbb{D}'_1} \times \cdots \times \mathcal{S}_{\mathbb{D}'_m}$ is given as

$$[\![\varphi]\!](\Sigma) = ([\![E(s_1)]\!]_\Sigma, \ldots, [\![E(s_m)]\!]_\Sigma)$$

This Lola semantics is well-defined if the value of no stream event is dependent on itself. This is the case when the graph of the specification contains no self-loops, which can easily be checked [9]. We assume that all Lola specifications are well-defined. With $\mathbf{D} := \mathbb{D}_1 \times \cdots \times \mathbb{D}_n$ and $\mathbf{D}' := \mathbb{D}'_1 \times \cdots \times \mathbb{D}'_m$, the *induced pointwise property* of a specification φ is the function $\mathcal{P}_\varphi : \mathbf{D}^{t_{max}} \times \mathbb{T} \to \mathbf{D}'$ defined as

$$\mathcal{P}_\varphi(w, t) = (s_1(t), \ldots, s_m(t))$$

where $(s_1, \ldots, s_m) = [\![\varphi]\!](w)$. Thereby we implicitly understand w as a tuple of streams.

Assumptions. Assumptions are knowledge about system and environment [18], which allow to restrict the actual set of possible input and output traces. Consider again Fig. 1. First, the robot can only be in one room at a time, so exactly one of r_0, r_1, r_2, r_3 must be true at any instant. The map also limits the transitions, so if r_1 is true at some instant, only r_0, r_1, r_3 can be true at the next instant, but not r_2. We can also make assumptions about energy consumption (for example at least 3% of energy is used at every instant). We follow [19] and encode assumptions in Lola, using a special stream Λ which we assume to be true at every instant. The assumptions above are e.g. encoded as follows:

$$\begin{aligned}\textbf{Def } \Lambda := &(r_0[now] \leftrightarrow \neg(r_1[now] \vee r_2[now] \vee r_3[now])) \wedge \cdots \wedge \\ &(r_0[now] \rightarrow (r_0[1|tt] \vee r_1[1|tt] \vee r_2[1|tt])) \wedge \cdots \wedge \\ &(e[now] \leq e[-1|103] - 3)\end{aligned}$$

Given a specification φ with assumption Λ and a tuple of input streams $\Sigma = (\sigma_1, \ldots, \sigma_n)$ we write $\Sigma \models_\Lambda \varphi$ if $[\![\varphi]\!](\Sigma)$ yields an output that only contains tt events for Λ.

Recurrent Lola Monitoring. Based on Definition 2 we define a sound and perfect recurrent Lola monitor as a recurrent monitor for the induced point-wise property of a specification, taking assumptions into account.

Given a Lola specification φ over input data types $\mathbf{D}$ and given assumption Λ, the set of possible verdicts after a non-empty input prefix $w \in \mathbf{D}^{\leq t_{max}}$ is $pos_\varphi(w) = \{\mathcal{P}_\varphi(wv, |w| - 1) \mid wv \in \mathbf{D}^{t_{max}+1} \wedge wv \models_\Lambda \varphi\}$.

Definition 3 (Sound/perfect recurrent Lola monitor). *A recurrent Lola monitor M is:*

- sound *iff for every non-empty $w \in \boldsymbol{D}^{\leq t_{max}}$, $\overline{M}(w) \supseteq pos_\varphi(w)$.*
- perfect *iff for every non-empty $w \in \boldsymbol{D}^{\leq t_{max}}$, $\overline{M}(w) = pos_\varphi(w)$.*

Lola monitors receive input streams instant by instant and, per input, cast the set (or an over-approximation) of the possible output stream value tuples.

Several monitoring approaches can be reduced to recurrent monitoring by modification of the specification. For example, consider a Boolean stream b representing a property. The initial value of this property (the value of b at position 0) can iteratively be monitored by introduction of an additional stream

Def $s =$ **if** *first* **then** $b[now]$ **else** $s[-1|\mathit{ff}]$. Note that s at instant 0 takes the value of b and otherwise takes the previous value of s. A recurrent monitor for s outputs increasingly precise verdicts about the initial property b. This monitor simulates the typical initial monitor, for example for LTL_3 [2]. Recurrent Lola monitors further subsume monitoring with reset [4]; monitoring instants with a fixed offset of k to the current instant, or a fixed size window around the current instant; monitoring the distance to the next instant where a violation of a property occurs (see [21]) or counting of violations, etc. All these notions can be solved with recurrent monitoring by introducing additional streams in the specification.

Perfect recurrent monitoring requires reasoning about possible future continuations of a trace. This ability however, especially together with the presence of assumptions makes recurrent monitors very powerful. The vacuum cleaning robot example above could include the following four stream definitions:

$$\textbf{Def } \mathit{enter}_{i\in\{0,1,2,3\}} := r_i[+1|\mathit{false}] \wedge \neg \mathit{Ferr}[now]$$

Note that if a recurrent monitor yields the verdict $\bot = \{\mathit{ff}\}$ for one of these streams, entering the corresponding room will inevitably cause *Ferr* to be true, which means that the base station cannot be reached anymore with the remaining battery energy. On the other hand, the verdict $? = \{\mathit{tt}, \mathit{ff}\}$ implies that it is possible that *Ferr* is false when the corresponding room is entered. This way a higher level planning system an use the information that the monitor provides to steer and prevent the robot from going into rooms which will inevitably cause an error. If the robot always follows a path where ? verdicts are obtained it will eventually end up in room 0 if the battery level is critical. In this example anticipatory verdicts are possible if assumptions that are included in the specification reveal information about where the robot can drive and how much energy it consumes.

3 Lola Recurrent Online Monitoring Semantics

We now introduce a novel Lola semantics for recurrent online monitoring. While the original semantics from Section 2 describes a relation between fully known input and output streams (i.e. an offline semantics), we now give a semantics that relates prefixes of input streams with partially known output streams. We base our definition on monitoring stream tuples (inspired by [32]) which represent a set of possible (complete and fully known) stream tuples:

Definition 4 (Monitoring stream tuple). *A* monitoring stream tuple *of n streams of types $\mathbb{D}_1, \ldots, \mathbb{D}_n$ is an element from $\mathcal{T}_{\mathbb{D}_1,\ldots,\mathbb{D}_n} := 2^{\mathcal{S}_{\mathbb{D}_1} \times \cdots \times \mathcal{S}_{\mathbb{D}_n}}$.*

We will use monitoring stream tuples in two ways: (1) to define input stream prefixes, which are only known up to a certain instant $t \in \mathbb{T}$; and (2) to encode uncertain input readings. (Note that the first case is a special case of the second, where all events after t are fully unknown.) The idea is that the monitoring stream tuple is the set of all complete and fully known input streams that are compatible with the (uncertain) input readings received so far.

Example 1. Consider again the robot example from Fig. 1 for $\mathbb{T} = \{0, 1, 2, 3, 4\}$ and where the received trace prefix is known up to instant 3. Assume that the robot started at room r_0 and moved to r_1 and then to r_3; then it is uncertain whether the robot remained in r_3 or moved back to r_1 again. Furthermore, the energy started at 100% and was reduced by 3% per step, but the sensor has an uncertainty of ±1%. This input would be encoded by the following monitoring stream tuple, where the streams follow the order r_0, r_1, r_2, r_3, e:

$$\begin{aligned} s = \{&(\langle tt, \mathit{ff}, \mathit{ff}, \mathit{ff}, r_0^4\rangle, \langle \mathit{ff}, tt, \mathit{ff}, r_1^3, r_1^4\rangle, \langle \mathit{ff}, \mathit{ff}, \mathit{ff}, \mathit{ff}, r_2^4\rangle, \langle \mathit{ff}, \mathit{ff}, tt, r_3^3, r_3^4\rangle, \\ &\langle e^0, e^1, e^2, e^3, e^4\rangle) \mid \\ &r_1^3 \leftrightarrow \neg r_3^3, e^0 \in [99, 101], e^1 \in [96, 98], e^2 \in [93, 95], e^3 \in [90, 92]\} \end{aligned}$$

Given a monitoring stream tuple $s \in \mathcal{T}_{\mathbb{D}_1 \times \cdots \times \mathbb{D}_n}$ we use $s(t)$ for $t \in \mathbb{T}$ to denote the set of all value tuples at position t. In the example above $s(3) = \{(\mathit{ff}, r_1^3, \mathit{ff}, r_3^3, e_3) \mid r_1^3 \leftrightarrow \neg r_3^3, e_3 \in [90, 92]\}$.

In this paper we restrict to "instant-wise uncertainty": our monitoring streams only encode uncertain values which are independent from the values at other instants. That is, we can encode that the robot is in room 3 iff it is not in room 0, but not that the robot is in room 3 if it was in room 0 in the previous instant. In many cases relations among instants can still be encoded as assumptions.

To simplify the definitions, for the rest of the paper we fix a Lola specification $\varphi = (I, S, E)$ with n input streams of type $\mathbb{D}_{1 \le i \le n}$ and m output streams of type $\mathbb{D}'_{1 \le i \le m}$. A monitoring stream tuple Σ for the input is then $\Sigma \in \mathcal{T}_{\mathbb{D}_1, \ldots, \mathbb{D}_n}$. We define the *monitoring semantics* of a Lola specification as the application of the standard Lola semantics on all streams from the input monitoring stream tuple.

Definition 5 (Lola monitoring semantics). *Let φ be a specification and Σ the monitoring stream tuple for the inputs. The monitoring semantics of φ, Σ is defined as:*

$$\begin{aligned} &[\![\varphi]\!]^{mon} : \mathcal{T}_{\mathbb{D}_1, \ldots, \mathbb{D}_n} \to \mathcal{T}_{\mathbb{D}_1, \ldots, \mathbb{D}_n, \mathbb{D}'_1, \ldots, \mathbb{D}'_m} \\ &[\![\varphi]\!]^{mon}(\Sigma) = \{(\sigma_1, \ldots, \sigma_n) \circ [\![\varphi]\!](\sigma_1, \ldots, \sigma_n) \mid (\sigma_1, \ldots, \sigma_n) \in \Sigma\} \end{aligned}$$

We handle assumptions by adding the condition $(\sigma_1, \ldots, \sigma_n) \models_\Lambda \varphi$ which restrict the input streams considered. The Lola monitoring semantics is closely related to a perfect recurrent Lola monitor: the output of a perfect recurrent Lola monitor after receiving input Σ at monitoring step t is $[\![\varphi]\!]^{mon}(\Sigma)(t)$. Receiving tuples $\Sigma_0, \Sigma_1, \Sigma_2 \ldots$ with growing information about input readings a monitor could compute $[\![\varphi]\!]^{mon}(\Sigma_0)$, $[\![\varphi]\!]^{mon}(\Sigma_1)$, $[\![\varphi]\!]^{mon}(\Sigma_2)$, ... and generate the outputs $[\![\varphi]\!]^{mon}(\Sigma_0)(0)$, $[\![\varphi]\!]^{mon}(\Sigma_1)(1)$, $[\![\varphi]\!]^{mon}(\Sigma_2)(2), \ldots$. This monitor, however, computes a monitoring stream tuple of all inputs and outputs so it contains information about all events of all streams, which makes semantics costly. Note that for recurrent monitoring we are actually only interested in the events at the current instant. Therefore, in the following we introduce a variation of the Lola monitoring semantics which produces sets of possible stream value combinations

(called *configurations*) for every instant, with no information relating different instants.

We first introduce some additional notation. Given a flat specification $\varphi = (I, S, E)$ for input stream types $\mathbb{D}_1, \ldots, \mathbb{D}_n$ and output stream types $\mathbb{D}'_1, \ldots, \mathbb{D}'_m$, we use $\mathbf{D}^\varphi = \mathbb{D}_1 \times \cdots \times \mathbb{D}_n \times \mathbb{D}'_1 \times \cdots \times \mathbb{D}'_m$ to denote the product of all stream types. Given $d \in \mathbf{D}^\varphi$ and $s \in I \cup S$ we use $d(s)$ to denote the entry of stream s in d. Elements from $2^{\mathbf{D}^\varphi}$, i.e. sets of stream value tuples, are called *configuration sets*. Given an expression $e \in \mathit{Expr}_{I \cup S}$ of type $\mathbb{D}$, the following three functions $[\![e]\!]^{\triangleright}_\varphi$ and $[\![e]\!]^{\triangleleft}_\varphi$ (with type $\mathbf{D}^\varphi \times \mathbf{D}^\varphi \to \mathbb{D}$), and $[\![e]\!]^{\bowtie}_\varphi$ (with type $\mathbf{D}^\varphi \times \mathbf{D}^\varphi \times \mathbf{D}^\varphi \to \mathbb{D}$) compute the value of e at the beginning, at the end and in the middle of the trace. $[\![e]\!]^{\triangleright}_\varphi$ receives the configuration for the current and subsequent instant, $[\![e]\!]^{\triangleleft}_\varphi$ receives the current and previous instant, and $[\![e]\!]^{\bowtie}_\varphi$ the configuration for the previous, current and subsequent instant. This semantics are:

$$\begin{aligned}
[\![d]\!]^{\bowtie}_\varphi(b, c, a) &= d \\
[\![f(e_1, \ldots, e_n)]\!]^{\bowtie}_\varphi(b, c, a) &= f([\![e_1]\!]^{\bowtie}_\varphi(b, c, a), \ldots, [\![e_n]\!]^{\bowtie}_\varphi(b, c, a)) \\
[\![s[-1|d]]\!]^{\bowtie}_\varphi(b, c, a) &= b(s) \\
[\![s[now]]\!]^{\bowtie}_\varphi(b, c, a) &= c(s) \\
[\![s[+1|d]]\!]^{\bowtie}_\varphi(b, c, a) &= a(s)
\end{aligned}$$

for constant $d \in \mathbb{D}$, stream identifier $s \in I \cup S$ and sub-expressions $e_1, \ldots, e_n \in \mathit{Expr}_{I \cup S}$. Here, b denotes the valuation at the previous instant, c at the current instant and a at the successor instant. The definitions for $[\![e]\!]^{\triangleright}_\varphi$ and $[\![e]\!]^{\triangleleft}_\varphi$ are analogous, but these use the default value for -1 and +1 references (resp.). Let $\varphi = (I, S, E)$ and let $S = \{s_1, \ldots, s_n\}$ be the output stream identifiers. We use

$$\begin{aligned}
[\![\varphi]\!]^{\triangleleft}(b, c) &= ([\![E(s_1)]\!]^{\triangleleft}_\varphi(b, c), \quad \ldots, [\![E(s_n)]\!]^{\triangleleft}_\varphi(b, c)) \\
[\![\varphi]\!]^{\triangleright}(c, a) &= ([\![E(s_1)]\!]^{\triangleright}_\varphi(c, a), \quad \ldots, [\![E(s_n)]\!]^{\triangleright}_\varphi(c, a)) \\
[\![\varphi]\!]^{\bowtie}(b, c, a) &= ([\![E(s_1)]\!]^{\bowtie}_\varphi(b, c, a), \ldots, [\![E(s_n)]\!]^{\bowtie}_\varphi(b, c, a))
\end{aligned}$$

to denote the application of the given functions on all defining expressions of φ.

We can finally define an alternative fixed point semantics which can serve as the basis for recurrent monitoring.

Definition 6 (Lola instant semantics). *Let φ be a specification and Σ a monitoring stream tuple of the input streams. The instant semantics fixed point equation of φ, Σ is:*

$$\begin{aligned}
[\![\varphi]\!]^{inst}_\Sigma &: (2^{\boldsymbol{D}^\varphi})^{|\mathbb{T}|} \to (2^{\boldsymbol{D}^\varphi})^{|\mathbb{T}|} \\
[\![\varphi]\!]^{inst}_\Sigma(V) &= (V'_0, \ldots, V'_{t_{max}})
\end{aligned}$$

with

$$\begin{aligned}
V'_0 &= \{c \mid c = \sigma \circ [\![\varphi]\!]^{\triangleright}(c, a), \sigma \in \Sigma(0), a \in V(1)\} \\
V'_t &= \{c \mid c = \sigma \circ [\![\varphi]\!]^{\bowtie}(b, c, a), \sigma \in \Sigma(t), b \in V(t-1), a \in V(t+1)\} \\
V'_{t_{max}} &= \{c \mid c = \sigma \circ [\![\varphi]\!]^{\triangleleft}(b, c), \sigma \in \Sigma(t_{max}), b \in V(t_{max} - 1)\}.
\end{aligned}$$

The instant semantics *of φ is given as the greatest fixed point of $[\![\varphi]\!]^{inst}_\Sigma$ w.r.t. the point-wise $\subseteq$ order on the $(2^{\boldsymbol{D}^\varphi})^{|\mathbb{T}|}$ structure:*

$$\begin{aligned}
[\![\varphi]\!]^{inst} &: \mathcal{T}_{\mathbb{D}_1, \ldots, \mathbb{D}_n} \to (2^{\boldsymbol{D}^\varphi})^{|\mathbb{T}|} \\
[\![\varphi]\!]^{inst}(\Sigma) &= \nu([\![\varphi]\!]^{inst}_\Sigma)
\end{aligned}$$

The instant semantics fixed point equation takes a structure of configuration sets for every trace position, and returns a homogeneous structure consisting of the possible inputs and the semantics of the output stream expressions for the corresponding positions (based on the argument structure). Consequently, a fixed point of this equation is a solution of the Lola specification. We define the instant semantics as the greatest fixed point of the instant semantics fixed point equation. One structure is greater or equal than another if at every instant it contains at least the same configurations, i.e. is the point-wise application of $\subseteq$. Note that the instant semantics of φ is equivalent to the monitoring semantics with respect to the stream events at every instant, that is

$$\forall t \in \mathbb{T}.\, \nu([\![\varphi]\!]_{\Sigma}^{inst})(t) = \{T(t) \mid T \in [\![\varphi]\!]^{mon}(\Sigma)\}$$

Hence, this semantics can also be used as basis for recurrent monitoring. Computing this semantics, however, is rather complex—requiring a fixed point iteration—and it must be recomputed every time new inputs are received (since Σ changes). Therefore, we slightly adjust this semantics again. Instead of computing the possible value combinations (configurations sets) we now compute them *parametric in the values of the previous instant*, using the structure $(\mathbf{D}^{\varphi} \to 2^{\mathbf{D}^{\varphi}})^{|\mathbb{T}|}$ instead of $(2^{\mathbf{D}^{\varphi}})^{|\mathbb{T}|}$. We call the elements of this structure *transformers* as they transform the configurations from the previous instant to those of the current instant. Transformers receive a configuration $b \in \mathbf{D}^{\varphi}$ at $t \in \mathbb{T}$ and return the set of all possible configurations at $t+1 \in \mathbb{T}$, provided b.

Definition 7 (Lola transformer semantics). *Let φ be a specification and Σ an input stream tuple. The transformer semantics fixed-point equation of φ and Σ is given as:*

$$\begin{aligned} [\![\varphi]\!]_{\Sigma}^{tra} &: (\boldsymbol{D}^{\varphi} \to 2^{\boldsymbol{D}^{\varphi}})^{|\mathbb{T}|} \to (\boldsymbol{D}^{\varphi} \to 2^{\boldsymbol{D}^{\varphi}})^{|\mathbb{T}|} \\ [\![\varphi]\!]_{\Sigma}^{tra}(V) &= (V'_0, \ldots, V'_{t_{max}}) \end{aligned}$$

with

$$\begin{aligned} V'_0(b) &= \{c \mid c = \sigma \circ [\![\varphi]\!]^{\triangleright}(c, a), \sigma \in \Sigma(0), a \in V(1)(c)\} \\ V'_t(b) &= \{c \mid c = \sigma \circ [\![\varphi]\!]^{\bowtie}(b, c, a), \sigma \in \Sigma(t), a \in V(t+1)(c)\} \\ V'_{t_{max}}(b) &= \{c \mid c = \sigma \circ [\![\varphi]\!]^{\triangleleft}(b, c), \sigma \in \Sigma(t_{max})\}. \end{aligned}$$

The transformer semantics of φ *is the (only) fixed point of* $[\![\varphi]\!]_{\Sigma}^{tra}$*:*

$$\begin{aligned} [\![\varphi]\!]^{tra} &: \mathbb{T}_{\mathbb{D}_1, \ldots, \mathbb{D}_n} \to (2^{\boldsymbol{D}^{\varphi}})^{|\mathbb{T}|} \\ [\![\varphi]\!]^{tra}(\Sigma) &= \mu([\![\varphi]\!]_{\Sigma}^{tra}) \end{aligned}$$

This semantics is basically equivalent to the instant semantics except that V'_t is no longer dependent on $V(t-1)$, as the generated transformers are parameterized in the configuration of their previous instant. Therefore, b is now a

parameter of the single structure entries and a is still received from the argument structure of the fixed point equation, by applying the current configuration on the subsequent transformer ($V(t+1)(c)$).

This new semantics has several advantages for online monitoring. First, the fixed point of the upper semantics is unique and can (as opposed to monitoring and instant semantics) be deterministically computed from the back, as the single transformer elements only depend on the subsequent transformer. Second, this semantics can still conveniently be used for recurrent monitoring. One can mutually compute the current monitor state (i.e. the currently possible stream configurations) and the transformer to the subsequent instant and apply the current state on the transformer (see Section 5). However, one caveat is that computing with $(\mathbf{D}^\varphi \to 2^{\mathbf{D}^\varphi})^{|\mathbb{T}|}$ is complex, as it is unclear how to represent the elements in $\mathbf{D}^\varphi \to 2^{\mathbf{D}^\varphi}$ and in $2^{\mathbf{D}^\varphi}$. Furthermore, the recursively defined sets V_i' are hard to determine. Therefore, we introduce a framework for abstract computation of this semantics.

4 An Abstraction Framework for Lola Monitoring

We borrow concepts from abstract interpretation to efficiently implement the transformer semantics. The main element is an abstract domain which is a perfect representation (or a sound over-approximation) of the transformer or configuration set domain. An appropriate abstract domain must be easy to represent in memory and enable efficient computations.

We introduce two domains: A, whose elements abstract concrete configuration sets from Section 3, and $\tilde{A}$ that contains abstractions of the transformers. We require that $(A, \sqsubseteq^A)$ and $(\tilde{A}, \sqsubseteq^{\tilde{A}})$ are complete lattices, that is, partial orders where every subset has a least upper bound and a greatest lower bound. The relation $a \sqsubseteq^A b$ indicates that b over-approximates a, i.e. that every configuration represented by a is also represented by b. The same holds for $\sqsubseteq^{\tilde{A}}$. We demand the existence of functions:

$$\gamma^A : A \to 2^{\mathbf{D}^\varphi} \qquad \alpha^A : 2^{\mathbf{D}^\varphi} \to A$$
$$\gamma^{\tilde{A}} : \tilde{A} \to (\mathbf{D}^\varphi \to 2^{\mathbf{D}^\varphi}) \qquad \alpha^{\tilde{A}} : (\mathbf{D}^\varphi \to 2^{\mathbf{D}^\varphi}) \to \tilde{A}$$

which are able to translate from the concrete configuration set or transformer domain to the abstract counterpart and back. We require that these function pairs are Galois connections:

$$\forall a \in A, c \in 2^{\mathbf{D}^\varphi} : \alpha^A(c) \sqsubseteq^A a \leftrightarrow c \subseteq \gamma^A(a)$$
$$\forall a \in A, c \in (\mathbf{D}^\varphi \to 2^{\mathbf{D}^\varphi}) : \alpha^{\tilde{A}}(c) \sqsubseteq^{\tilde{A}} a \leftrightarrow c \trianglelefteq \gamma^{\tilde{A}}(a)$$

Here, $\trianglelefteq$ denotes the pointwise application of $\subseteq$ on all corresponding configurations sets where the functions from $(\mathbf{D}^\varphi \to 2^{\mathbf{D}^\varphi})$ map to. Galois connections ensure that a translation from the concrete to the abstract domain and back leads to an over-approximation, so abstract computations in the abstract domain produce sound monitor outputs.

We say that A is a *perfect configuration set abstraction* if for all $c \in 2^{\mathbf{D}^\varphi}$, $\gamma^A(\alpha^A(c)) = c$. Analogously $\tilde{A}$ is a *perfect transformer abstraction* if for all $c \in (\mathbf{D}^\varphi \to 2^{\mathbf{D}^\varphi})$, $\gamma^{\tilde{A}}(\alpha^{\tilde{A}}(c)) = c$.

Symbolic abstraction. We introduce now a perfect abstract transformer and configuration set abstract domain based on symbolic constraints, which will be later used for an anticipatory Lola monitoring algorithm in Section 6. For the symbolic abstraction we use symbolic constraints (i.e. quantifier-free first order logic expressions) that perfectly describe the relation among all possible values of a configuration or transformer.

We start with the symbolic representation of the configuration sets. We use a symbolic constraint where every stream value is represented by its own variable. For example, $C = \{(tt, 3), (ff, 5)\}$—for two streams b (of type bool) and r (of type real)—captures values that can either be tt and 3 or ff and 5, This configuration set can be expressed as $(b \to (r = 3)) \wedge (\neg b \to (r = 5))$. Our symbolic computation is restricted to those configuration sets which are symbolically representable, thus the theory of choice (e.g. Boolean algebra or linear real arithmetic) determines the capabilities of the monitor. We assume that the chosen algebra can encode all monitor inputs and operations in the specification.

The concretization function of a symbolic constraint ψ is:

$$\gamma(\psi) = \{v \in \mathbf{D}^\varphi \mid \big(\bigwedge_{s \in I \cup S} s = v(s) \big) \models \psi\}$$

Recall that $v(s)$ denotes the value of stream s in a configuration $v \in \mathbf{D}^\varphi$. We implicitly define α s.t. for any configuration set $C \in 2^{\mathbf{D}^\varphi}$, $\gamma(\alpha(C)) = C$. That is, every configuration set C has a canonical symbolic encoding. In the algorithm we only require α for translating uncertain input readings to symbolic representations. Note that by the given definition of α the symbolic domain is a perfect configuration set abstraction. Also note that while our symbolic domain is defined as abstraction of configuration sets over all streams, it is also possible to encode only sets of sub-configurations, e.g. only input stream values.

Consider for example Fig. 1 and the following configuration set $v = \{(ff, r_1^3, ff, r_3^3, e_3) \mid \neg(r_1^3 \leftrightarrow r_3^3), e_3 \in [90, 92]\}$, which represents the uncertain input for instant 3 from the example above. A symbolic representation of this configuration set is $\alpha(v) = \neg r_0 \wedge \neg r_2 \wedge \neg(r_1 \leftrightarrow r_3) \wedge (90 \leq e \leq 92)$.

We also encode transformers symbolically, extending the variables of our constraints to $I \cup S \cup \{s^{-1} \mid s \in I \cup S\}$, where s^{-1} represent the stream values at the previous instant in which the transformer is parametric. The corresponding concretization function for transformers is given as $\gamma(\psi) = \tau$ s.t.

$$\forall v \in \mathbf{D}^\varphi : \tau(v) = \{u \in \mathbf{D}^\varphi \mid \big(\bigwedge_{s \in I \cup S} ((s^{-1} = v(s)) \wedge (s = u(s))) \big) \models \psi\}.$$

Abstract Transformer Semantics Computation. We now present the computation of an alternative, abstract transformer semantics, related to the concrete semantics given in Definition 7. This semantics is computed in an $\tilde{A}^{|\mathbb{T}|}$

structure where each entry contains the abstract transformer for the corresponding trace position.

We fix an abstract transformer domain $\tilde{A}$ with translation functions $\gamma^{\tilde{A}} : \tilde{A} \to (\mathbf{D}^{\varphi} \to 2^{\mathbf{D}^{\varphi}})$ and $\alpha^{\tilde{A}} : (\mathbf{D}^{\varphi} \to 2^{\mathbf{D}^{\varphi}}) \to \tilde{A}$.

Definition 8 (Abstract Lola transformer semantics). *A fixed point equation for φ, Σ is called* abstract Lola transformer fixed point equation *if*

$$\begin{aligned}[\![\varphi]\!]^{\sharp}_{\Sigma} &: \tilde{A}^{|\mathbb{T}|} \to \tilde{A}^{|\mathbb{T}|} \\ [\![\varphi]\!]^{\sharp}_{\Sigma}(V) &= (\tau^0_{\varphi,\Sigma}(V(1)), \tau^1_{\varphi,\Sigma}(V(2)), \ldots, \tau^{t_{max}}_{\varphi,\Sigma})\end{aligned}$$

with $\tau^{t_{max}}_{\varphi,\Sigma} : \tilde{A}$ and $\tau^t_{\varphi,\Sigma} : \tilde{A} \to \tilde{A}$ for $t \in \{0, \ldots, t_{max} - 1\}$ s.t.

$$\begin{aligned}\tau^0_{\varphi,\Sigma}(V_1) &\sqsupseteq^{\tilde{A}} \alpha^{\tilde{A}}(b \mapsto \{c \mid c = \sigma \circ [\![\varphi]\!]^{\triangleright}(c, a) \mid \sigma \in \Sigma(0), a \in \gamma^{\tilde{A}}(V_1)(c)\}) \\ \tau^t_{\varphi,\Sigma}(V_{t+1}) &\sqsupseteq^{\tilde{A}} \alpha^{\tilde{A}}(b \mapsto \{c \mid c = \sigma \circ [\![\varphi]\!]^{\bowtie}(b, c, a) \mid \sigma \in \Sigma(t), a \in \gamma^{\tilde{A}}(V_{t+1})(c)\}) \\ \tau^{t_{max}}_{\varphi,\Sigma} &\sqsupseteq^{\tilde{A}} \alpha^{\tilde{A}}(b \mapsto \{c \mid c = \sigma \circ [\![\varphi]\!]^{\triangleleft}(b, c) \mid \sigma \in \Sigma(t_{max})\}).\end{aligned}$$

This corresponds to a computation in the abstract structure $\tilde{A}^{|\mathbb{T}|}$ where all the entries are over-approximations of the transformers of the concrete Lola transformer semantics. If the $\sqsupseteq^{\tilde{A}}$ relation in the above definitions is an equality then $[\![\varphi]\!]^{\sharp}_{\Sigma}$ is called a *perfect abstract Lola transformer fixed point equation*. We will later in Section 6 provide the abstract transformer constructors $\tau^t_{\varphi,\Sigma}$ for the symbolic abstract domain introduced above.

As in the concrete case, the abstract transformer fixed point equation above has a unique fixed point $\mu([\![\varphi]\!]^{\sharp}_{\Sigma})$, as it can be computed deterministically from back to front given a particular input Σ. We say that our abstract transformer semantics is sound in relation to the concrete semantics if for all $t \in \mathbb{T}$, $\mu([\![\varphi]\!]^{tra}_{\Sigma})(t) \subseteq \gamma^{\tilde{A}}(\mu([\![\varphi]\!]^{\sharp}_{\Sigma})(t))$ and perfect if $\mu([\![\varphi]\!]^{tra}_{\Sigma})(t) = \gamma^{\tilde{A}}(\mu([\![\varphi]\!]^{\sharp}_{\Sigma})(t))$. By properties of abstract interpretation the following holds:

Theorem 1. *Let $\mu([\![\varphi]\!]^{\sharp}_{\Sigma})$ be an abstract transformer semantics for φ. Then:*

- *$\mu([\![\varphi]\!]^{\sharp}_{\Sigma})$ is sound.*
- *$\mu([\![\varphi]\!]^{\sharp}_{\Sigma})$ is perfect if $[\![\varphi]\!]^{\sharp}_{\Sigma}$ is a perfect abstract Lola transformer fixed point equation and $\tilde{A}$ is a perfect transformer abstraction.*

This justifies that we can build a sound or perfect recurrent Lola monitor based on this abstract semantics. Consider the computation of the fixed point $\mu([\![\varphi]\!]^{\sharp}_{\top})$, where $\top$ is the maximal element in $\mathcal{T}_{\mathbb{D}_1,\ldots,\mathbb{D}_n}$ (i.e. the input monitoring stream tuple where no information about any input streams is available). The abstract transformer structure chosen for the abstract semantics has one significant advantage in terms of the computation of this fixed point: As soon as a single element in $S = \mu([\![\varphi]\!]^{\sharp}_{\top})$ repeats, all entries of the structure (except the one for instant 0) are known. This is because if $S(t) = S(t + k)$ for $k > 0$, $t \in \mathbb{T}$, then also $S(t - 1) = S(t + k - 1)$ are equal (as no input information is available with $\Sigma = \top$). Therefore, all entries in S can be filled up to instant 1 without new computations being required. Hence, $\mu([\![\varphi]\!]^{\sharp}_{\top})$ can be computed

back to front until the first instant at which $\mu([\![\varphi]\!]^\sharp_\top)(t) = \mu([\![\varphi]\!]^\sharp_\top)(t+k)$ occurs, and then the values at all instants are determined (except for the first entry). If the number of elements in the abstract domain $\tilde{A}$ is bounded by c, (e.g. Boolean specifications) then after at most c iterations a loop in $\mu([\![\varphi]\!]^\sharp_\top)$ is found. There are domains beyond Booleans for which finite perfect representations exist [13].

For abstract domains where $|\tilde{A}|$ is unbounded one can use a widening operator [8,7]. For example, using $\mu([\![\varphi]\!]^\sharp_\top)(t) \bigtriangledown \mu([\![\varphi]\!]^\sharp_\top)(t-1)$ instead of $\mu([\![\varphi]\!]^\sharp_\top)(t-1)$ in the fixed point computation where the operator $\bigtriangledown : \tilde{A} \times \tilde{A} \to \tilde{A}$ yields an over-approximation of the arguments by taking all unstable components of the abstractions directly to the extreme limits and thus enforcing a loop in $\mu([\![\varphi]\!]^\sharp_\top)$.

Based on these observations we build in the next section an efficient sound (or perfect) recurrent Lola monitoring algorithm.

5 Abstraction-Based Recurrent Lola Monitoring

We introduce our monitor construction based on the abstract structure from the previous section. At runtime the monitor receives information incrementally, so there is a sequence of extending input monitoring stream tuples $\Sigma_0, \Sigma_1, \ldots, \Sigma_{t_{max}}$ where in Σ_t all streams are fully unknown for instants larger than t and equal to Σ_{t-1} for instants smaller than t. Based on this observation we introduce the online monitoring algorithm Algorithm 1.

Algorithm 1 Abstract Lola monitoring algorithm

Compute (over-approximation of) $\mu([\![\varphi]\!]^\sharp_\top)$
$s^\sharp \leftarrow \top^A$
foreach $t \in \mathbb{T}$ **do**
  Read inputs for t
  Compute $\mu([\![\varphi]\!]^\sharp_{\Sigma_t})(t) = \tau^t_{\varphi,\Sigma_t}(\mu([\![\varphi]\!]^\sharp_\top)(t+1))$
  $s^\sharp \leftarrow \mu([\![\varphi]\!]^\sharp_{\Sigma_t})(t)(s^\sharp)$
  Output $\gamma(s^\sharp)$
end

The algorithm first determines $\mu([\![\varphi]\!]^\sharp_\top)$, which is not dependent on inputs and can thus be computed statically as part of the monitor synthesis (as described at the end of the previous section). Then, at runtime the monitor receives iteratively the (possibly uncertain) inputs for the current instant t and computes $\mu([\![\varphi]\!]^\sharp_{\Sigma_t})(t)$. By definition

$$\mu([\![\varphi]\!]^\sharp_{\Sigma_t})(t) = \tau^t_{\varphi,\Sigma_t}(\mu([\![\varphi]\!]^\sharp_{\Sigma_t})(t+1)).$$

However, $\mu([\![\varphi]\!]^\sharp_{\Sigma_t})(t+1) = \mu([\![\varphi]\!]^\sharp_\top)(t+1)$ because for all $t' > t$ no inputs are available yet. This can be taken from the pre-computed $\mu([\![\varphi]\!]^\sharp_\top)$, and hence

$\mu([\![\varphi]\!]^\sharp_{\Sigma_t})(t)$ can be efficiently determined by applying $\tau^t_{\varphi,\Sigma_t}$ once without requiring a full computation of the fixed point $\mu([\![\varphi]\!]^\sharp_{\Sigma_t})$ from the end.

Then, the algorithm applies the abstracted configuration set from the previous step, stored in $s^\sharp$ (of type A) on the computed transformer $\mu([\![\varphi]\!]^\sharp_{\Sigma_t})(t)$ and assigns the result to $s^\sharp$ again. In this manner $s^\sharp$ represents the monitor state: the set of possible stream configurations at the current instant t. Note that $s^\sharp$ is not available for $t = 0$ as there is no previous instant and thus also no monitor state. Yet $\mu([\![\varphi]\!]^\sharp_{\Sigma_0})(0)$ yields (by definition) a transformer which is independent of the predecessor argument. The concrete representation of $s^\sharp$ is $\gamma(s^\sharp)$ which consists of a set of possible value tuples for all streams, and serves as the monitor output. This output is perfect if and only if the chosen abstract domains A and $\tilde{A}$ are perfect configuration set and transformer abstractions and the abstract transformer semantics is also perfect (see Theorem 1).

The application of an abstract transformer $T \in \tilde{A}$ on a configuration set abstraction $s \in A$ is technically defined as $T(s) = \alpha^A(\{\gamma^{\tilde{A}}(T)(c) \mid c \in \gamma^A(s)\})$. Depending on the concrete abstractions there may be easier ways to achieve the application, for example using symbolic constraints, as we will see in the next section.

The size of $s^\sharp$ may grow over time, so for a constant-size monitor it may be necessary to find an over-approximation. In conclusion the following holds:

Theorem 2. *Let φ be a Lola specification let $\Sigma_0, \Sigma_1, \ldots, \Sigma_{t_{max}}$ be an extending sequence of input monitoring stream tuples where Σ_t contains the input readings for instant t. Algorithm 1 yields a sound recurrent Lola monitor and a perfect recurrent Lola monitor if $[\![\varphi]\!]^\sharp_\Sigma$ is a perfect abstract Lola transformer fixed point equation and $\tilde{A}$ is a perfect transformer abstraction.*

6 Symbolic Recurrent Lola Monitoring

yy We now show a symbolic monitoring strategy under assumptions that tolerates uncertainty, for linear real arithmetic Lola specifications based on the general framework from the previous section. This theory supports real and Boolean streams, and the common Boolean operations, additions, constant multiplications and comparisons among real streams.

We will use the symbolic abstract domain to symbolically represent configurations and transformers. For convenience we use instant variables formed by stream names with the corresponding instant in the exponent of the symbolic variables. For example, s^3 indicates the value of the event in stream s at instant 3. Abstractions of configuration sets only contain variables of a single instant, transformer abstractions those of the current and previous instant.

Example 2. Consider a specification with a single stream e of type real. The configuration set that states the value of e at instant 3 is between 90 and 92 (both inclusive) would be represented by the constraint $90 \leq e^3 \leq 92$. To express that the value of e at instant 4 is at least 3 less than the value one instant before, that is, the transformer $T(e) = \{e' \mid e' \leq e - 3\}$, we could use $e^4 \leq e^3 - 3$.

A perfect monitoring procedure requires—besides perfect abstract domains $\tilde{A}$ and A—perfect symbolic constructions for the transformers $\tau^t_{\varphi,\Sigma}$. This can be achieved in a straight forward manner as follows. To compute the transformer at instant t we take the symbolic representation of the subsequent transformer in the structure, and conjunct it with the symbolic instantiation of the specification at the current instant and the input readings for the current instant. This works because we required the input values of different instants to be independent of each other. For $t = 0$ or $t = t_{max}$ we use the default values.

Example 3. Consider again the specification from Fig. 1 (for this example without the parts added later like assumptions) and the situation where no inputs are known, which can be encoded by the symbolic constraint tt, and $\mathbb{T} = \{0, \ldots, 10\}$. The symbolic transformer $\tau^{t_{max}}_{\varphi,\top}$ for $t_{max} = 10$ is:

$$\mu([\![\varphi]\!]^\sharp_\top)(10) = \tau^{10}_{\varphi,\top} = (err^{10} = \neg r_0^{10} \wedge (e^{10} < 5)) \wedge (Ferr^{10} = err^{10})$$

and $\tau^9_{\varphi,\top}$ applied on $\mu([\![\varphi]\!]^\sharp_\top)(10)$ is

$$\begin{aligned}\mu([\![\varphi]\!]^\sharp_\top)(9) = \tau^9_{\varphi,\top}(\mu([\![\varphi]\!]^\sharp_\top)(10)) = (err^{10} = \neg r_0^{10} \wedge (e^{10} < 5)) \wedge (Ferr^{10} = err^{10})\\ \wedge (err^9 = \neg r_0^9 \wedge (e^9 < 5)) \wedge (Ferr^9 = err^9 \vee Ferr^{10}).\end{aligned}$$

Applying this strategy, the resulting formulas can grow and ultimately involve all instant variables from the current instant up to the trace end. Likewise instant variables from later instants are included, which are actually not allowed to be included in the transformers because their presence could prevent finding a repeated element in $\tilde{A}$ and result in full unrolling of the specification. The fully computed transformers would express relations among all the instant variables to the stream end. In contrast, our online monitoring only preserves the relation among the variables for the current and previous instant, so we search for an alternative representation of the formula above which is equivalent w.r.t. the instant variables at the current and previous time points. This is equivalent to existentially quantifying over the variables to be removed and apply quantifier elimination if it can be used.

Example 4. Revisiting the previous example, real linear arithmetic quantifier elimination determines that $\mu([\![\varphi]\!]^\sharp_\top)(9)$ is

$$\begin{aligned}\mu([\![\varphi]\!]^\sharp_\top)(9) &= \exists r_0^{10}, err^{10}, Ferr^{10}, e^{10}.(err^{10} = \neg r_0^{10} \wedge (e^{10} < 5)) \wedge\\ &\quad (Ferr^{10} = err^{10}) \wedge (err^9 = \neg r_0^9 \wedge (e^9 < 5)) \ \wedge (Ferr^9 = err^9 \vee Ferr^{10})\\ &= (err^9 = \neg r_0^9 \wedge (e^9 < 5)) \wedge (err^9 \rightarrow Ferr^9)\end{aligned}$$

Following this strategy for $\mu([\![\varphi]\!]^\sharp_\top)(8)$:

$$\begin{aligned}\mu([\![\varphi]\!]^\sharp_\top)(8) &= \exists \neg r_0^9, err^9, Ferr^9, e^9.(err^9 = \neg r_0^9 \wedge (e^9 < 5)) \wedge\\ &\quad (err^9 \rightarrow Ferr^9) \wedge (err^8 = \neg r_0^8 \wedge (e^8 < 5)) \ \wedge (Ferr^8 = err^8 \vee Ferr^9)\\ &= (err^8 = \neg r_0^8 \wedge (e^8 < 5)) \wedge (err^8 \rightarrow Ferr^8)\end{aligned}$$

Thus $\mu([\![\varphi]\!]^\sharp_\top)(9)$ and $\mu([\![\varphi]\!]^\sharp_\top)(8)$ are (modulo instant variable timestamps) equal to each other and consequently also to $\mu([\![\varphi]\!]^\sharp_\top)(7), \ldots, \mu([\![\varphi]\!]^\sharp_\top)(1)$. Hence, after three computation steps $\mu([\![\varphi]\!]^\sharp_\top)$ is fully computed, independent of the concrete t_{max} (except for the entry at instant 0).

If the specification contains assumptions, we also add $\ldots \wedge \Lambda^t$ to each symbolic transformer. Unfortunately, quantifier elimination does not guarantee to reach a stabilized formula as above. Therefore, we propose the following three stage strategy for the computation of the initial fixed point, which may ultimately lead to an over-approximation of $\mu([\![\varphi]\!]^\sharp_\top)$:

1. Compute the elements of $\mu([\![\varphi]\!]^\sharp_\top)$ from back to front applying quantifier elimination for k steps.
2. If no repeating entry is found for l steps, the elements of $\mu([\![\varphi]\!]^\sharp_\top)$ are determined but besides variables of future instants all real variables are eliminated. For the current instant real variables' maximal and minimal bounds are determined based on the computed symbolic representation and added to the final symbolic representation (see [19]).
3. If still no repeating element is found, the strategy is applied again but with widening [7] on the bounds of two subsequent instants interval. For example, let $[a, b]$ be the previously computed interval and $[a', b']$ the new one. The lower widened interval bound is $-\infty$ if $a' < a$ and a otherwise. Dually, the upper widened interval bound is ∞ if $b' > b$ and b otherwise.

As all constraints over a fixed number of Boolean variables can be represented in a formula of constant length and the bounds of all real variables either stabilize or are brought to $\pm\infty$ by widening, it is guaranteed that a repeating element will be found in the third stage. Note that eliminating real variables and replacing their constraints with bounds leads to an over-approximation. The resulting transformer and monitor are still sound but not necessarily perfect.

For the monitoring we finally recompute $\mu([\![\varphi]\!]^\sharp_{\Sigma^t})(t)$ for each timestamp t. We do this analogously to the initial fixed point computation before, but also add the new input constraints $\alpha(\Sigma^t) \backslash \alpha(\Sigma^{t-1})$ (i.e. the input readings of the current instant).

When it comes to the application of the computed transformer to the current monitor state we can simply conjunct the constraints of the transformer and the current monitor state and again use quantifier elimination to eliminate the variables from the previous instant.

Example 5. Take again the transformer $\tau = (e^4 \leq e^3 - 3)$ from Example 2 and the monitoring state $s^\sharp = (90 \leq e^3 \leq 92)$ from above, we get $\tau(s) = \exists e^3.(e^4 \leq e^3 - 3) \wedge (90 \leq e^3 \leq 92)$. After application of quantifier elimination this would lead to state $s^\sharp = e^4 \leq 89$.

If the monitor state grows too large we can also apply the second stage of the above strategy to reduce its size at the cost of making the monitor state less precise. As a further optimization, note that from the first fixed point (for $\Sigma = \top$), we only need the relation between those variables that are referenced

by +1 offsets in the specification. Therefore, during quantifier elimination for the initial fixed point we can also remove all variables from the current instant which are not referenced in this way.

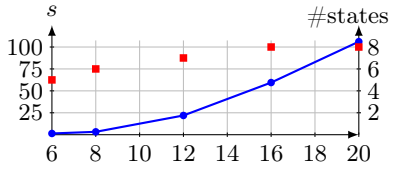
7 Empirical Evaluation

We developed a prototype for symbolic recurrent Lola monitoring in Scala using Z3 [28] as backend solver for symbolic reasoning and quantifier elimination. We evaluated our tool on three case studies running on a 64-bit Linux machine with an Intel Core i7-1365U CPU and 32GB of RAM.

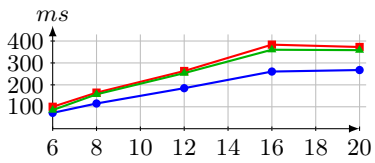
Path Planning. The first case study examines a variation of the vacuum cleaning robot from Fig. 1. The example was extended such that the output does not only specify whether a room can be safely entered but also with how much surplus or missing energy. This information could then be used to control the robot's behavior, e.g. switching to a power-saving mode, showing the advantages of a monitoring approach which is able to compute richer verdicts than just Booleans.

We analyzed the monitor's synthesis time (i.e. the time for the computation of the initial fixed point), and the monitor time per instant at runtime for a variable number of rooms by simulating a random walk according to the monitor's output. In this case study, the initial semantics could be fully determined without widening, as a repeating symbolic transformer element was found after a few computations. As Fig. 2a shows, the synthesis time grows non-linearly. This is because the backwards calculation becomes more expensive with longer paths. This can be remedied by simplifying the symbolic representation of formulas during computation. However, Z3 is rather optimized for satisfiability checks but not for simplifying symbolic constraints. We will explore the benefits of specialized simplifiers and further optimizations for reducing the synthesis time as future work.

More important than synthesis time is the execution time of the monitors. The average computation time per instant during the monitor execution, measured with different degrees of induced uncertainty, is shown in Fig. 2b. Runtime increases when uncertainty is introduced but the time-per-event is still small (384 ms) in the worst case.

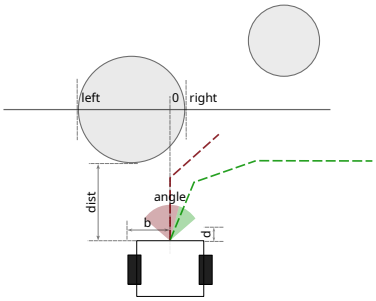


2a: Monitor synthesis per number of rooms. Synthesis time in seconds (●) and number of computed states (■).

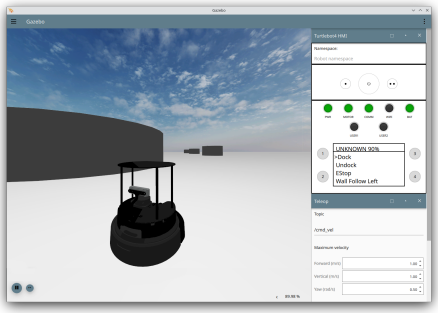


2b: Avg. runtime (ms) per instant for different room numbers. Full certain (●), 30% noisy (■) and 15% entirely unknown (▲).

Collision Avoidance. In the second case study a robot uses a Lola monitor to navigate through an area with obstacles. The robot receives a set of waypoints from the user and tries to follow them while avoiding the obstacles. The monitor receives as inputs the distance *dist* to the closest obstacle in front of the robot, as well as its leftmost and rightmost points *left* and *right*. The monitor outputs the possible steering angles to avoid collisions in the future (see Fig. 3a). Assumptions define parameters like the maximum possible steering angle and the bounding box of the robot (see b and d in Fig. 3a).



3a: Collision avoidance (scheme).



3b: Screenshot of the simulation in Gazebo.

The study was qualitatively evaluated by integrating our Lola monitoring tool with the robot operating system ROS [30] running on a turtlebot[3] inside the simulation environment Gazebo [23] (see Fig. 3b). The robot follows a user defined path, periodically calling the monitor for the closest obstacle in front obtaining safe steering angles, from which the robot chooses the one closest to the defined path. The monitor was able to steer the robot without collision with an uncertainty margin of up to 30%. We additionally extracted execution traces and evaluated the performance of the monitor offline. Fig. 4 shows the runtime per instant, which increases with growing input uncertainty due to the increasing complexity of the constraint states.

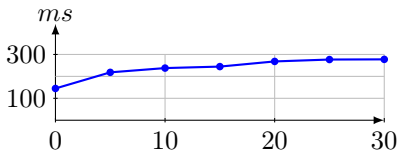


Fig. 4: Average runtime (ms) for uncertainty margins from 0% to 30%.

Program Monitoring. In the third case study we use our approach for traditional program monitoring. An excerpt of the monitored program is shown below on the left.

[3] `https://www.turtlebot.com/`

```
1 x = getInput();
2 [...]
3 y = 0;
4 while (x > 0) {
5   x--;
6   y += 2;
7 }
8 assert y >= 15;
```

At the end of the program we wanted to ensure that the value of variable `y` which is previously computed in a while loop does exceed 15. We have created a Lola specification which receives the current variable values as input streams and the current program line. Furthermore the program behavior itself was encoded in a straight-forward manner as assumption in the Lola specification. With its anticipation capabilities the monitor was able to compute legal values for the variables at certain program positions s.t. the assertion at the end is satisfied. Thus, it was able to detect program failures at an early stage during program execution.

Since the valid variable values depend on the number of while loop executions in the program (and thus the remaining trace length), the initial transformer semantics computation of our approach did not find a repeating transformer. Consequently the widening strategy described above has been applied to yield a sound recurrent Lola monitor for the specification. In the particular example however the simple interval widening was still able to capture that before entering the while loop variable `x` has to be at least 8, yet some other variable connections have been over-approximated. Yet, when in line 1 an input was entered which ultimately lead to $\texttt{x} < 8$ in line 3 the monitor was able to detect the failure right there. Altogether this provides an illustrative example how the approach from this paper could be used for a mixture of static and dynamic program analysis, which in a large scale however would require more sophisticated widening techniques than in the current implementation.

8 Conclusion

In this paper we have studied general anticipatory monitoring of Lola specifications under uncertainties and assumptions. We have introduced a hierarchy of monitoring semantics and presented an abstraction based framework for monitoring, from which we developed a general sound or perfect online monitoring algorithm for Lola. This algorithm considers future continuations of the received input, provided an abstraction of stream data values. Finally, we have presented an instantiation of this algorithm based on a symbolic representation. and evaluated the approach in three practical scenarios. Due to Lola's universality, our theory can also serve a general framework for anticipatory monitoring of synchronous RV formalisms.

Future work includes a more efficient implementation, especially improving the simplification of the symbolic constraints applied during monitoring, and applications to other Lola fragments beyond linear arithmetic. We also plan to extend the approach to infinite traces and asynchronous SRV formalisms.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Basin, D.A., Harvan, M., Klaedtke, F., Zalinescu, E.: MONPOLY: monitoring usage-control policies. In: Proc. of the 2nd Int'l Conf on Runtime Verification (RV'11). LNCS, vol. 7186, pp. 360–364. Springer (2011). https://doi.org/10.1007/978-3-642-29860-8_27
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.: Comparing LTL semantics for runtime verification. J. Log. Comput. **20**(3), 651–674 (2010). https://doi.org/10.1093/logcom/exn075
4. Cimatti, A., Tian, C., Tonetta, S.: Assumption-based runtime verification with partial observability and resets. In: Pro. of the 19th International Conference on Runtime Verification (RV'19). LNCS, vol. 11757, pp. 165–184. Springer (2019). https://doi.org/10.1007/978-3-030-32079-9_10
5. 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
6. Convent, L., Hungerecker, S., Leucker, M., Scheffel, T., Schmitz, M., Thoma, D.: TeSSLa: Temporal stream-based specification language. In: Proc. of teh 21st Brazilian Symposium on Formal Methods: Foundations and Applications (SBMF'18). LNCS, vol. 11254, pp. 144–162. Springer (2018). https://doi.org/10.1007/978-3-030-03044-5_10
7. Cousot, P.: Principles of abstract interpretation. The MIT Press (2021)
8. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proc. of the 4th ACM Symp. on Principles of Programming Languages (POL'77). pp. 238–252. ACM (1977). https://doi.org/10.1145/512950.512973
9. 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 Symp. of Temporal Representation and Reasoning (TIME'05). pp. 166–174. IEEE Computer Society (2005). https://doi.org/10.1109/TIME.2005.26
10. Decker, N., Leucker, M., Thoma, D.: Monitoring modulo theories. Int. J. Softw. Tools Technol. Transf. **18**(2), 205–225 (2016). https://doi.org/10.1007/s10009-015-0380-3
11. 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) Part I. LNCS, vol. 11561, pp. 421–431. Springer (2019). https://doi.org/10.1007/978-3-030-25540-4_24
12. Faymonville, P., Finkbeiner, B., Schwenger, M., Torfah, H.: Real-time stream-based monitoring. CoRR **abs/1711.03829** (2017), http://arxiv.org/abs/1711.03829
13. Felli, P., Montali, M., Patrizi, F., Winkler, S.: Monitoring arithmetic temporal properties on finite traces. In: Proce. of the 37th AAAI Conference on Artificial Intelligence (AAAI'23). pp. 6346–6354. AAAI Press (2023). https://doi.org/10.1609/aaai.v37i5.25781

14. Goldberg, A., Havelund, K.: Automated runtime verification with Eagle. In: Proc. of the 3rd Int'l Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems, (MSVVEIS'05). INSTICC Press (2005)
15. Gorostiaga, F., Sánchez, C.: Striver: Stream runtime verification for real-time event-streams. In: Proc. of the 18th Int'l Conf. on Runtime Verification (RV'18). LNCS, vol. 11237, pp. 282–298. Springer (2018). https://doi.org/10.1007/978-3-030-03769-7_16
16. 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
17. Havelund, K., Rosu, G.: Synthesizing monitors for safety properties. In: Proc. of the 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
18. Henzinger, T.A., Saraç, N.E.: Monitorability under assumptions. In: 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
19. Kallwies, H., Leucker, M., Sánchez, C.: Symbolic runtime verification for monitoring under uncertainties and assumptions. In: Proc. of the 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
20. Kallwies, H., Leucker, M., Sánchez, C.: General anticipatory monitoring for temporal logics on finite traces. In: Proc. of the 23rd Int'l Conf. on Runtime Verification (RV'23). LNCS, vol. 14245, pp. 106–125. Springer (2023). https://doi.org/10.1007/978-3-031-44267-4_6
21. Kallwies, H., Leucker, M., Sánchez, C., Scheffel, T.: Anticipatory recurrent monitoring with uncertainty and assumptions. In: Proc. of the 22nd Int'l Conf. on Runtime Verification (RV'22). LNCS, vol. 13498, pp. 181–199. Springer (2022). https://doi.org/10.1007/978-3-031-17196-3_10
22. Kallwies, H., Leucker, M., Schmitz, M., Schulz, A., Thoma, D., Weiss, A.: TeSSLa - an ecosystem for runtime verification. In: Proc. of the 22nd Int'l Conf. on Runtime Verification (RV'22). LNCS, vol. 13498, pp. 314–324. Springer (2022). https://doi.org/10.1007/978-3-031-17196-3_20
23. Koenig, N.P., Howard, A.: Design and use paradigms for Gazebo, an open-source multi-robot simulator. In: Proc. of the 2004 IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems (IROS'04). vol. 3, pp. 2149–2154 vol.3. IEEE (2004). https://doi.org/10.1109/IROS.2004.1389727
24. 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
25. Leucker, M., Sánchez, C., Scheffel, T., Schmitz, M., Thoma, D.: Runtime verification for timed event streams with partial information. In: Proc. of the 19th Int'l Conf. on Runtime Verification (RV'19). LNCS, vol. 11757, pp. 273–291. Springer (2019). https://doi.org/10.1007/978-3-030-32079-9_16
26. Leucker, M., Schallhart, C.: A brief account of runtime verification. J. Log. Algebraic Methods Program. **78**(5), 293–303 (2009). https://doi.org/10.1016/j.jlap.2008.08.004
27. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Proc. of the Joint International Conferences on Formal Modelling and Analysis of

Timed Systems (FORMATS'04) and Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'04). LNCS, vol. 3253, pp. 152–166. Springer (2004). https://doi.org/10.1007/978-3-540-30206-3_12
28. de Moura, L.M., Bjørner, N.: Z3: An efficient SMT solver. In: Proc. of the 14th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08). LNCS, vol. 4963, pp. 337–340. Springer (2008). https://doi.org/10.1007/978-3-540-78800-3_24
29. 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
30. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.: ROS: an open-source robot operating system. In: Workshops at the IEEE Int'l Conf. on Robotics and Automation (ICRA'90). vol. 3 (2009)
31. 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
32. Schmitz, M.: Efficient implementation of stream transformations. Ph.D. thesis, University of Lübeck, Germany (2024), https://www.zhb.uni-luebeck.de/epubs/ediss3011.pdf
33. Shoham, S., Grumberg, O.: A game-based framework for CTL counterexamples and 3-valued abstraction-refinement. In: Proc. of the 15th Int'l Conf. on Computer Aided Verification (CAV'03). LNCS, vol. 2725, pp. 275–287. Springer (2003). https://doi.org/10.1007/978-3-540-45069-6_28
34. Waga, M., André, É., Hasuo, I.: Symbolic monitoring against specifications parametric in time and data. In: Proc. of the 31st Int'l Conf. on Computer-Aided Verification (CAV'19) Part I. LNCS, vol. 11561, pp. 520–539. Springer (2019). https://doi.org/10.1007/978-3-030-25540-4_30
35. Waga, M., André, É., Hasuo, I.: Model-bounded monitoring of hybrid systems. ACM Transactions on Cyber-Physical Systems **6:4**(30), 1–26 (2021). https://doi.org/10.1145/3529095