Symbolic Runtime Verification for Monitoring Under Uncertainties and Assumptions

Hannes Kallwies[1](✉), Martin Leucker[1], and César Sánchez[2]

[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 deals with the question of whether a run of a system adheres to its specification. This paper studies runtime verification in the presence of partial knowledge about the observed run, particularly where input values may not be precise or may not be observed at all. We also allow declaring assumptions on the execution which permits to obtain more precise verdicts also under imprecise inputs. We encode the specification into a symbolic formula that the monitor solves iteratively, when more observations are given. We base our framework on stream runtime verification, which allows to express temporal correctness properties not only in the Boolean but also in richer logical theories. While in general our approach requires to consider larger and larger sets of formulas, we identify domains (including Booleans and Linear Algebra) for which pruning strategies exist, which allow to monitor with constant memory (i.e. independent of the length of the observation) while preserving the same inference power as the monitor that remembers all observations. We empirically exhibit the power of our technique using a prototype implementation under two important cases studies: software for testing car emissions and heart-rate monitoring.

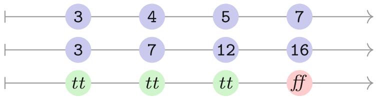
1 Introduction

We study runtime verification (RV) for imprecise and erroneous inputs, and describe a solution—called *Symbolic Runtime Verification*—that can exploit assumptions about the input and the system under analysis. RV is a dynamic verification technique in which a given run of a system is checked against a specification, typically a correctness property (see [1,13,23]). In *online monitoring* a monitor—synthesized from the specification—attempts to produce a verdict incrementally from the input trace. Originally, variants of LTL [26] tailored to finite runs were employed to formulate properties [3]. However, since

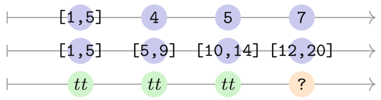
This work was funded in part by the Madrid Regional Government under project "S2018/TCS-4339 (BLOQUES-CM)" and by a research grant from Nomadic Labs and the Tezos Foundation.

$$\begin{aligned}&\texttt{In}\ \mathit{ld}\texttt{: Real}\\ \texttt{Def}\ \ \mathit{acc} &:= \mathit{acc}[-1|0] + \mathit{ld}[\mathit{now}] - \mathit{ld}[-3|0]\\ \texttt{Def}\ \ \mathit{ok} &:= (\mathit{acc}[\mathit{now}] \leq 15)\end{aligned}$$

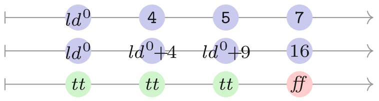
(a) A Specification



(b) Monitor run with perfect information



(c) Interval abstraction



(d) Symbolic monitor

Fig. 1. An example specification (a) and three monitors: (b) with perfect observability, (c) with an interval abstract domain, (d) a symbolic monitor developed in this paper. The symbolic monitor is enriched with the additional constraint that $1 \leq \mathit{ld}^0 \leq 5$.

RV requires to solve a variation of the word problem and not the harder model-checking problem, richer logics than LTL have been proposed that allow richer data and verdicts [10,14]. Lola [9] proposes *Stream Runtime Verification* (SRV) where monitors are described declaratively and compute output streams of verdicts from inputs streams (see also [12,21]). The development of this paper is based on Lola.

Example 1. Figure 1(a) shows a Lola specification with *ld* as input stream (the load of a CPU), *acc* as an output stream that represents the accumulated load, computed by adding the current value of *ld* and subtracting the third last value. Finally, *ok* checks whether *acc* is below 15. The expression $\mathit{acc}[-1|0]$ denotes the value of *acc* in the previous time point and 0 as default value if no previous time point exists.

Such a specification allows a direct evaluation strategy whenever values on the input streams arrive. If, for example, $\mathit{ld} = 3$ in the first instant, *acc* and *ok* evaluate to 3 and *tt*, respectively. Reading subsequently $4, 5, 7$ results in $7, 12, 16$ for *acc* and a violation is identified on stream *ok*. This is shown in Fig. 1(b).

A common obstacle in RV is that in practice sometimes input values are not available or not given precisely, due to errors in the underlying logging functionality or technical limitations of sensors. In Fig. 1(c) the first value on *ld* is not obtained (but we assume that the value of *ld* at instant 0 is between 1 and 5). One approach (followed in [22]) is to use interval arithmetic, which can be easily encoded as a domain in Lola. Even after reading precisely 4, 5 and 7, at time 4 the monitor cannot know for sure whether *ok* has been violated, as the interval $[12, 20]$ contains 15. If the unknown input on *ld* is denoted symbolically by ld^0 we still deduce that *ok* holds at time points 1 to 3. For time point 4, however, the symbolic representation $\mathit{acc}^4 = \mathit{acc}^3 + 7 - \mathit{ld}^0 = \mathit{ld}^0 + 9 + 7 - \mathit{ld}^0 = 16$ allows to infer that *ok* is clearly violated! This is shown in Fig. 1(d). □

Example 1 illustrates our first insight: *Symbolic Runtime Verification* is more precise than monitoring using abstract domains.

Since an infinite symbolic unfolding of the specification and all assumptions is practically infeasible, our online monitor unfolds the specification as time increases. We show that our monitor is both *sound* and *perfect* in the sense that it only produces correct verdicts and these verdicts are as precise as possible with the information provided. However, even with an incremental unfolding of the specification the symbolic monitor can grow as more unknowns and their inter-dependencies are introduced. For example, in the run in Fig. 1(d) as more unknown *ld* values are received, more variables ld^i will be added, which makes the size of the symbolic formula depend on the trace length. We show that for certain logical theories, the current verdict may be still be computed even after *summarizing* the history into a compact symbolic representation, whose size is independent of the trace length. For other theories, preserving the full precision requires an amount of memory that can grow with the trace length. More precisely, we show that for the theories of Booleans and of Linear Algebra, bounded symbolic monitors exist while this is not the case for the combined theory, which is our second insight.

We empirically validate our symbolic RV approach— including constant memory monitoring on long traces—using two realistic case studies: the Legal Driving Cycle [5,19] and an ECG heartbeat analysis [25,27] (following the Lola encoding from [11]). When intervals are given for unknown values, our method provides precise answers more often than previous approaches based on interval domains [22]. Especially in the ECG example, these methods are unable to recover once the input is unknown for even a short time, but our symbolic monitors recover and provide again precise results, even when the input was unknown for a larger period.

Related Work. Monitoring LTL for traces with mutations (errors) is studied in [17] where properties are classified according to whether monitors can be built that are resilient against the mutation. However, [17] only considers Boolean verdicts and does not consider assumptions. The work in [22] uses abstract interpretation to soundly approximate the possible verdict values when inputs contain errors for the SRV language TeSSLa [7].

Calculating and approximating the values that programs compute is central to static analysis and program verification. Two traditional approaches are symbolic execution [18] and abstract interpretation [8] which frequently require over-approximations to handle loops. In monitoring, a step typically does not contain loops, but the set of input variables (unlike in program analysis) grows. Also, a main concern of RV is to investigate monitoring algorithms that are guaranteed to execute with constant resources. Works that incorporate assumptions when monitoring include [6,15,20] but uncertainty is not considered in these works, and verdicts are typically Boolean. A symbolic approach for monitoring but in the setting of timed data automata and without constant memory guarantee (thus perfect) is also presented in [28]. Monitoring under assumptions in form of a linear hybrid automaton, and sampling uncertainty for a different kind of specifications and Boolean verdicts is also studied in [29]. Note that bounded model checking [4] also considers bounded unfoldings, but it does not solve the problem of building monitors of constant memory for successive iterations.

In summary, our contributions are: (1) A Symbolic Runtime Verification algorithm that dynamically unfolds the specification, collects precise and imprecise input readings, and instantiates assumptions generating a conjunction of formulas. This representation can be used to deduce verdicts even under uncertainty, to precisely recover automatically for example under windows of uncertainty, and even to anticipate verdicts. (2) A pruning method for certain theories (Booleans and Linear Algebra) that guarantees bounded monitoring preserving the power to compute verdicts. (3) A prototype implementation and empirical evaluation on realistic case studies.

Missing proofs, further examples and figures can be found in an extended version of this paper [16].

2 Preliminaries

We use Lola [9] to express our monitors. Lola uses first-order sorted theories to build expressions. These theories are interpreted in the sense that every symbol is both a constructor to build expressions, and an evaluation function that produces values from the domain of results from values from the domains of the arguments. All sorts of all theories that we consider include the $=$ predicate.

A synchronous stream s over a non-empty data domain $\mathbb{D}$ is a function $s : \mathcal{S}_{\mathbb{D}} := \mathbb{T} \to \mathbb{D}$ assigning a value of $\mathbb{D}$ to every element of $\mathbb{T}$ (timestamp). We consider infinite streams ($\mathbb{T} = \mathbb{N}$) or finite streams with a maximal timestamp $t_{\max}$ ($\mathbb{T} = [0 \ldots t_{\max}]$). For readability we denote streams as sequences, so $s = \langle 1, 2, 4 \rangle$ stands for $s : \{0, 1, 2\} \to \mathbb{N}$ with $s(0) = 1, s(1) = 2, s(2) = 4$. A Lola specification describes a transformation from a set of input streams to a set of output streams.

Syntax. A Lola specification $\varphi = (I, O, E)$ consists of a set I of typed variables that denote the input streams, a set O of typed variables that denote the output steams, and E which assigns to every output stream variable $y \in O$ a defining expression E_y. The set of expressions over $I \cup O$ of type $\mathbb{D}$ is denoted by $E_{\mathbb{D}}$ and is recursively defined as: $E_{\mathbb{D}} = c \mid s[o|c] \mid f(E_{\mathbb{D}_1}, ..., E_{\mathbb{D}_n}) \mid ite(E_{\mathbb{B}}, E_{\mathbb{D}}, E_{\mathbb{D}})$, where c is a constant of type $\mathbb{D}$, $s \in I \cup O$ is a stream variable of type $\mathbb{D}$, $o \in \mathbb{Z}$ is an offset and f a total function $\mathbb{D}_1 \times \cdots \times \mathbb{D}_n \to \mathbb{D}$ (*ite* is a special function symbol to denote if-then-else). The intended meaning of the offset operator $s[o|c]$ is to represent the stream that has at time t the value of stream s at $t + o$, and value c used if $t + o \notin \mathbb{T}$. A particular case is when the offset is $o = 0$ in which case c is not needed, which we shorten by $s[now]$. Function symbols allow to build terms that represent complex expressions. The intended meaning of the defining equation E_y for output variable y is to declaratively define the values of stream y in terms of the values of other streams.

Semantics. The semantics of a Lola specification φ is a mapping from input to output streams. Given a tuple of concrete input streams ($\Sigma = (\sigma_1, \dots, \sigma_n) \in \mathcal{S}_{\mathbb{D}_1} \times \dots \times \mathcal{S}_{\mathbb{D}_n}$) corresponding to input stream identifier $s_1, \dots, s_n$ and a specification φ the semantics of an expression $[\![\cdot]\!]_{\Sigma,\varphi} : E_{\mathbb{D}} \to \mathcal{S}_{\mathbb{D}}$ is iteratively defined as:

- $[\![c]\!]_{\Sigma,\varphi}(t) = c$
- $[\![s[o|c]]\!]_{\Sigma,\varphi}(t) = \begin{cases} \sigma_i(t+o) & \text{if } t+o \in \mathbb{T} \text{ and } s = s_i \in I \text{ (input stream)} \\ [\![e]\!]_{\Sigma,\varphi}(t+o) & \text{if } t+o \in \mathbb{T} \text{ and } E_s = e \text{ (output stream)} \\ c & \text{otherwise} \end{cases}$
- $[\![f(e_1, ..., e_n)]\!]_{\Sigma,\varphi}(t) = f([\![e_1]\!]_{\Sigma,\varphi}(t), \dots, [\![e_n]\!]_{\Sigma,\varphi}(t))$
- $[\![ite(e_1, e_2, e_3)]\!]_{\Sigma,\varphi}(t) = \begin{cases} [\![e_2]\!]_{\Sigma,\varphi}(t) & \text{if } [\![e_1]\!]_{\Sigma,\varphi}(t) = tt \\ [\![e_3]\!]_{\Sigma,\varphi}(t) & \text{if } [\![e_1]\!]_{\Sigma,\varphi}(t) = \mathit{ff} \end{cases}$

The semantics of φ is a map ($[\![\varphi]\!] : (\mathcal{S}_{\mathbb{D}_1} \times \dots \times \mathcal{S}_{\mathbb{D}_n}) \to (\mathcal{S}_{\mathbb{D}'_1} \times \dots \times \mathcal{S}_{\mathbb{D}'_m})$ defined as $[\![\varphi]\!](\sigma_1, ..., \sigma_n) = ([\![e'_1]\!]_{\Sigma,\varphi}, \dots, [\![e'_m]\!]_{\Sigma,\varphi})$. The evaluation map $[\![\cdot]\!]_{\Sigma,\varphi}$ is well-defined if the recursive evaluation above has no cycles. This acyclicity can be easily checked statically (see [9]).

In online monitoring monitors receive the values incrementally. The *very efficiently monitorable* fragment of Lola consists of specifications where all offsets are negative or 0 (without transitive 0 cycles). It is well-known that the very efficiently monitorable specifications (under perfect information) can be monitored online in a trace length independent manner. In the rest of the paper we also assume that all Lola specifications come with -1 or 0 offsets. Every specification can be translated into such a normal form by introducing additional streams (flattening).

In this paper we investigate online monitoring under uncertainty and assumptions for three special fragments of Lola (and the constraints for uncertain input readings and assumptions), depending on the data theories used:

- **Propositional Logic (Lola$_{\mathbb{B}}$):** The data domain of all streams is the Boolean domain $\mathbb{D} = \mathbb{B} = \{tt, \mathit{ff}\}$ and available functions are $\wedge, \neg$.
- **Linear Algebra (Lola$_{\mathcal{LA}}$):** The data domain of all streams are real numbers $\mathbb{D} = \mathbb{R}$ and every stream definition has the form $c_0 + c_1 * s_0[o_1|d_1] + \dots + c_n * s_n[o_n|d_n]$ where c_i, d_i are constants.
- **Mixed (Lola$_{\mathbb{B}/\mathcal{LA}}$):** The data domain is $\mathbb{B}$ or $\mathbb{R}$. Every stream definition is either contained in the Propositional Logic fragment extended by the functions $<, =$ for real variables or in the Linear Algebra fragment.

3 A Framework for Symbolic Runtime Verification

In this section we introduce a general framework for monitoring using symbolic computation, where the specification and the information collected by the monitor (including assumptions and precise and imprecise observations) are presented symbolically.

3.1 Symbolic Expressions

Consider a specification $\varphi = (I, O, E)$. We will use symbolic expressions to capture the relations between the different streams at different points in time. We introduce the *instant variables* x^t for a given stream variable $x \in I \cup O$ and instant $t \in \mathbb{T}$. The type of x^t is that of x. Considering Example 1, ld^3 represents the real value that corresponds to the input stream ld at instant 3 which is 7. The set of instant variables is $V = (I \cup O) \times \mathbb{T}$.

Definition 1 (Symbolic Expression). *Let φ be a specification and $\mathcal{A}$ a set of variables that contains all instant variables (that is $V \subseteq \mathcal{A}$), the set of symbolic expressions $\overline{\mathbb{D}}$ is the smallest set containing (1) all constants c and all symbols in $a \in \mathcal{A}$, (2) all expressions $f(t_1, \ldots, t_n)$ where f is a constructor symbol of type $\mathbb{D}_1 \times \cdots \times \mathbb{D}_n \to \mathbb{D}$ and t_i are elements of $\overline{\mathbb{D}}$ of type $\mathbb{D}_i$.*

We use $\mathit{Expr}^{\mathbb{D}}_{\varphi}(\mathcal{A})$ for the set of symbolic expressions of type $\mathbb{D}$ (and drop φ and $\mathcal{A}$ when it is clear from the context).

Example 2. Consider again Example 1. The symbolic expression $\mathit{acc}^3 + \mathit{ld}^4$, of type $\mathbb{R}$, represents the addition of the load at instant 4 and the accumulator at instant 3. Also, $\mathit{acc}^4 = \mathit{acc}^3 + \mathit{ld}^4$ is a predicate (that is, a $\mathbb{B}$ expression) that captures the value of acc at instant 4. The symbolic expression $\mathit{ld}^1 = 4$ corresponds to the reading of the value 4 for input stream ld at instant 1. Finally, $1 \leq \mathit{ld}^0 \wedge \mathit{ld}^0 \leq 5$ corresponds to the assumption at time 0 that ld has value between 1 and 5. $\square$

3.2 Symbolic Monitor Semantics

We define the symbolic semantics of a Lola specification $\varphi = (I, O, E)$ as the expressions that result by instantiating the defining equations E.

Definition 2 (Symbolic Monitor Semantics). *The map $[\![\cdot]\!]_{\varphi} : E_{\mathbb{D}} \to \mathbb{T} \to \mathit{Expr}^{\mathbb{D}}_{\varphi}$ is defined as $[\![c]\!]_{\varphi}(t) = c$ for constants, and*

- $[\![f(e_1, \ldots, e_n)]\!]_{\varphi}(t) = f([\![e_1]\!]_{\varphi}(t), \ldots, [\![e_n]\!]_{\varphi}(t))$
- $[\![s[o|c]]\!]_{\varphi}(t) = s^{t+o}$ *if $t + o \in \mathbb{T}$, or $[\![s[o|c]]\!]_{\varphi}(t) = c$ otherwise.*

The symbolic semantics of a specification φ is the map $[\![\cdot]\!]_{sym} : \mathbb{T} \to 2^{\mathit{Expr}^{\mathbb{B}}_{\varphi}}$ defined as $[\![\varphi]\!]^t_{sym} = \{y^t = [\![E_y]\!]_{\varphi}(t) \mid$ for every $y \in O\}$.

A slight modification of the symbolic semantics allows to obtain equations whose right hand sides only have input instant variables:

- $[\![s[o|c]]\!]_{\varphi}(t) = s^{t+o}$ if $t + o \in \mathbb{T}$ and $s \in I$
- $[\![s[o|c]]\!]_{\varphi}(t) = [\![E_s]\!](t + o)$ if $t + o \in \mathbb{T}$ and $s \in O$
- $[\![s[o|c]]\!]_{\varphi}(t) = c$ otherwise

We call this semantics the symbolic unrolled semantics, which corresponds to what would be obtained by performing equational reasoning (by equational substitution) in the symbolic semantics.

Example 3. Consider again the specification φ in Example 1. The first four elements of $[\![\varphi]\!]_{sym}$ are (after simplifications like $0 + x = x$ etc.):

0	1	2	3
$acc^0 = ld^0$ $ok^0 = acc^0 \leq 15$	$acc^1 = acc^0 + ld^1$ $ok^1 = acc^1 \leq 15$	$acc^2 = acc^1 + ld^2$ $ok^2 = acc^2 \leq 15$	$acc^3 = acc^2 + ld^3 - ld^0$ $ok^3 = acc^3 \leq 15$

Using the unrolled semantics the equations for ok would be, at time 0, $ok^0 = ld^0 \leq 15$, and at time 1, $ok^1 = ld^0 + ld^1 \leq 15$. In the unrolled semantics all equations contain only instant variables that represent inputs. □

Recall that the denotational semantics of Lola specifications in Sect. 2 maps every tuple of input streams into a tuple of output streams, that is $[\![\varphi]\!] : \mathcal{S}_{\mathbb{D}_1} \times \cdots \times \mathcal{S}_{\mathbb{D}_n} \rightarrow \mathcal{S}_{\mathbb{D}'_1} \times \cdots \times \mathcal{S}_{\mathbb{D}'_m}$. The symbolic semantics also has a denotational meaning even without receiving the input stream, defined as follows.

Definition 3 (Denotational semantics). *Let $\varphi = (I, O, E)$ be a specification with $I = (x_1, \ldots, x_n)$ and $O = (y_1, \ldots, y_m)$. The denotational semantics of a set of equations $E \subseteq Expr^{\mathbb{B}}_{\varphi}$, $[\![E]\!]_{den} \subseteq \mathcal{S}_{\mathbb{D}_1} \times \cdots \times \mathcal{S}_{\mathbb{D}_n} \times \mathcal{S}_{\mathbb{D}'_1} \times \cdots \times \mathcal{S}_{\mathbb{D}'_m}$ is:*

$$[\![E]\!]_{den} = \{(\sigma_1, \ldots, \sigma_n, \sigma'_1, \ldots, \sigma'_m) \mid \textit{ for every } e \in E \\ \{x_1^t = \sigma_1(t), \ldots, x_n^t = \sigma_n(t), y_1^t = \sigma'_1(t), \ldots, y_m^t = \sigma'_m(t)\} \models e\}$$

Using the previous definition, $[\![\bigcup_{i \leq t} [\![\varphi]\!]^i_{sym}]\!]_{den}$ corresponds to all the tuples of streams of inputs and outputs that satisfy the specification φ up to time t.

A Symbolic Encoding of Inputs, Constraints and Assumptions. Input readings can also be defined symbolically as follows. Given an instant t, an input stream variable x and a value v, the expression $x^t = v$ captures the precise reading of v at t on x. Imprecise readings can also be encoded easily. For example, if at instant 3 an input of value 7 for ld is received by a noisy sensor (consider a 1 unit of tolerance), then $6 \leq ld^3 \leq 8$ represents the imprecise reading.

Assumptions are relations between the variables that we assume to hold at all positions, which can be encoded as stream expressions of type $\mathbb{B}$. For example, the assumption that the load is always between 1 and 10 is $1 \leq ld[now] \leq 10$. Another example, $ld[-1|0] + 1 \geq ld[now]$ which encodes that ld cannot increase more than 1 per unit of time. We use A for the set of assumptions associated with a Lola specification φ (which are a set of stream expressions of type $\mathbb{B}$ over $I \cup O$).

3.3 A Symbolic Runtime Verification Algorithm

Based on the previous definitions we develop our symbolic RV algorithm shown in Algorithm 1. Line 3 instantiates the new equations and assumptions from the specification for time t. Line 4 incorporates the set of input readings ψ^t (perfect or imperfect). Line 5 performs evaluations and simplifications, which is dependent on the particular theory. In the case of past-specifications with perfect information this step boils down to substitution and evaluation. Line 6 produces the output of the monitor. Again, this is application dependent. In the case of past specifications with perfect information the output value will be computed without delay and emitted in this step. In the case of $\mathbb{B}$ outputs with imperfect information, an SMT solver can be used to discard a verdict. For example, to determine the value of *ok* at time t, the verdict *tt* can be discarded if $\exists * .ok^t$ is UNSAT, and the verdict *ff* can be discarded if $\exists * .\neg ok^t$ is UNSAT. For richer domains specific reasoning can be used, like emitting lower and upper bounds or deducing the set of constraints. Finally, Line 7 eliminates constraints that will not be necessary for future deductions and performs variable renaming and summarization to restrict the memory usage of the monitor (see Sect. 4). For past specifications with perfect information, after step 5 every equation will be evaluated to $y^t = v$ and the pruning will remove from E all the values that will never be accessed again.

Alg. 1: Online Symbolic Monitor for φ

```
1 $t \leftarrow 0$ and $E \leftarrow \emptyset$;
2 while $t \in \mathbb{T}$ do
3     $E \leftarrow E \cup [\![\varphi]\!]^t_{sym} \cup [\![A^t]\!]_\varphi$;
4     $E \leftarrow E \cup \psi^t$;
5     Evaluate and Simplify;
6     Output;
7     Prune;
8     $t \leftarrow t + 1$ ;
```

The symbolic RV algorithm generalizes the concrete monitoring algorithm by allowing to reason about uncertain values, while it still obtains the same results and performance under certainty. Concrete RV allows to monitor with constant amount of resources specifications with bounded future references when inputs are known with perfect certainty.

Symbolic RV, additionally, allows to handle uncertainties and assumptions, because the monitor stores constraints (equations) that include variables that capture the unknown information, for example the unknown input values. We characterize a symbolic monitor as a step function $M : 2^{Expr}_\varphi \rightarrow 2^{Expr}_\varphi$ that transforms expressions into expressions. At a given instant t the monitor collects readings $\psi^t \in Expr_\varphi$ about the input values and applies the step function to the previous information and the new information. Given a sequence of input readings $\psi^0, \psi^1 \dots$ we use $M^0 = M(\psi^0)$ and $M^{i+1} = M(M^i \cup \psi^{i+1})$ for the sequence of monitor states reached by the repeated applications of M. We use $\Phi^t = \cup_{i \leq t}([\![\varphi]\!]^i_{sym} \cup [\![A^i]\!]_\varphi \cup \psi^i)$ for the formula that represents the unrolling of the specification and the current assumptions together with the knowledge about inputs collected up to t.

Definition 4 (Sound and Perfect monitoring). *Let φ be a specification, M a monitor for φ, $\psi^0, \psi^1 \dots$ a sequence of input observations, and $M^0, M^1 \dots$ the monitor states reached after repeatedly applying M. Consider an arbitrary predicate α involving only instant variables x^t at time t.*

– *M is sound if whenever $M^t \models \alpha$ then $\Phi^t \models \alpha$ for all $t \in \mathbb{T}$.*
– *M is perfect if it is sound and if $\Phi^t \models \alpha$ then $M^t \models \alpha$ for all $t \in \mathbb{T}$.*

Note that soundness and perfectness is defined in terms of the ability to infer predicates that only involve instant variables at time t, so the monitor is allowed to eliminate, rename or summarize the rest of the variables. It is trivial to extend this definition to expressions α that can use instant variables $x^{t'}$ with $(t-d) \leq t' \leq t$ for some constant d. If a monitor is perfect in this extended definition it will be able to answer questions for variables within the last d steps.

The version of the symbolic algorithm presented in Alg. 1 that never prunes (removing line 7) and computes at all steps Φ^t is a sound and perfect monitor. However, the memory that the monitor needs grows without bound if the number of uncertain items also grows without bound. In the next section we show that (1) trace length independent perfect monitoring under uncertainty is not possible in general, even for past only specifications and (2) we identify concrete theories, namely Booleans and Linear Algebra and show that these theories allow perfect monitoring with constant resources under unbounded uncertainty.

4 Symbolic Runtime Verification at Work

Example 4. Consider the Lola specification on the left, where the Real input stream ld indicates the current CPU load and the Boolean input stream usr_a indicates if the currently active user is user A. This specification checks whether the accumulated load of user A is at most 50% of the total accumulated load. Consider the inputs $ld = \langle ?, 10, 4, ?, ?, 1, 9, \dots \rangle$, $usr_a = \langle ff, ff, ff, tt, tt, tt, ff, \dots \rangle$ from 0 to 6. Also, assume that at every instant t, $0 \leq ld^t \leq 10$. At instant 6 our monitoring algorithm would yield the symbolic constraints $(acc^6 = 24 + ld^0 + ld^3 + ld^4)$ and $(acc_a^6 = 1 + ld^3 + ld^4)$ for acc^6 and acc_a^6, and the additional one $(0 \leq ld^0 \leq 10 \ \wedge 0 \leq ld^3 \leq 10 \ \wedge \ 0 \leq ld^4 \leq 10)$. An existential query to an SMT solver allows to conclude that ok^6 is true since acc_a^6 is at most 21 but then acc^6 is 44. However, every unknown variable from the input will appear in one of the constraints stored and will remain there during the whole monitoring process. □

$$\begin{aligned} acc &:= acc[-1|0] + ld[now] \\ acc_a &:= acc_a[-1|0] + ite(usr_a[now], \\ &\qquad ld[now], 0) \\ ok &:= acc_a[now] \leq 0.5 * acc[now] \end{aligned}$$

When symbolic computation is used in static analysis, it is not a common concern to deal with a growing number of unknowns as usually the number of inputs is fixed a-priori. In contrast, a goal in RV is to build online monitors that are trace-length independent, which means that the calculation time and memory consumption of a monitor stays below a constant bound and does not increase with the received number of inputs. In Example 4 above this issue can be tackled by rewriting the constraints as part of the monitor's pruning step using $n \leftarrow ld^0$, $m \leftarrow (ld^3 + ld^4)$ to obtain $(acc^t = 24 + n + m)$, $(acc_a^t = 1 + m)$ and $(0 \leq n \leq 10) \wedge (0 \leq m \leq 20)$. From the rewritten constraints it can still be deduced that $acc_a^6 \leq 0.5 * acc^6$. Note also that every instant variable in the specification only refers to previous instant variables. Thus for all $t \geq 7$, there is no direct reference to either ld^3 or ld^4. Variables ld^3 and ld^4 are, individually, no longer *relevant* for the verdict and it does not harm to denote $ld^3 + ld^4$ by a single variable m. We call this step of rewriting *pruning* (of non-relevant variables).

Let $\mathcal{C}^t \subseteq \mathit{Expr}^{\mathbb{B}}_{\varphi}$ be the set of constraints maintained by the monitor that encode its knowledge about inputs and assumptions for the given specification. In general, pruning is a transformation of a set of constraints $\mathcal{C}^t$ into a new set $\mathcal{C}'^t$ requiring less memory, but still describing the same relations between the instant variables:

Definition 5 (Pruning strategy). *Let $\mathcal{C} \subseteq \mathit{Expr}^{\mathbb{B}}$ be a set of constraints over variables $\mathcal{A}$ and $\mathcal{R} = \{r_1, \dots, r_n\} \subseteq \mathcal{A}$ the subset of* relevant *variables. We use $|\mathcal{C}|$ for a measure on the size of $\mathcal{C}$. A* pruning strategy *$\mathcal{P} : 2^{\mathit{Expr}^{\mathbb{B}}} \to 2^{\mathit{Expr}^{\mathbb{B}}}$ is a transformation such that for all $\mathcal{C} \in \mathit{Expr}^{\mathbb{B}}$, $|\mathcal{P}(\mathcal{C})| \leq |\mathcal{C}|$. A Pruning strategy $\mathcal{P}$ is called*

- sound*, whenever for all $\mathcal{C} \subseteq \mathit{Expr}^{\mathbb{B}}$, $[\![\mathcal{C}]\!]_{\mathcal{R}} \subseteq [\![\mathcal{P}(\mathcal{C})]\!]_{\mathcal{R}}$,*
- perfect*, whenever for all $\mathcal{C} \subseteq \mathit{Expr}^{\mathbb{B}}$, $[\![\mathcal{C}]\!]_{\mathcal{R}} = [\![\mathcal{P}(\mathcal{C})]\!]_{\mathcal{R}}$,*

where $[\![\mathcal{C}]\!]_{\mathcal{R}} = \{(v_1, \dots, v_n) | (r_1 = v_1 \wedge \dots \wedge r_n = v_n) \models \mathcal{C}\}$ is the set of all value tuples for $\mathcal{R}$ that entail the constraint set $\mathcal{C}$. We say that the pruning strategy is constant *if for all $\mathcal{C} \subseteq \mathit{Expr}^{\mathbb{B}} : |\mathcal{P}(\mathcal{C})| \leq c$ for a constant $c \in \mathbb{N}$.*

A monitor that exclusively stores a set $\mathcal{C}^t$ for every $t \in \mathbb{T}$ is called a constant-memory monitor if there is a constant $c \in \mathbb{N}$ such that for all t, $|\mathcal{C}^t| \leq c$.

Previously we defined an online monitor M as a function that iteratively maps sets of constraints to sets of constraints. Clearly, the amount of information to maintain grows unlimited if we allow the monitor to receive constraints that contain information of an instant variable at time t at any other time t'. Consequently, we first restrict our attention to *atemporal monitors*, defined as those which receive proposition sets that only contain instant variables of the current instant of time. Atemporal monitors cannot handle assumptions like $ld[-1|0] \leq 1.1 * ld[\mathit{now}]$. At the end of this section we will extend our technique to monitors that may refer n instants to the past.

Theorem 1. *Given a specification φ and a constant pruning strategy $\mathcal{P}$ for $\mathit{Expr}^{\mathbb{B}}_{\varphi}$, there is an atemporal constant-memory monitor M_{φ} s.t.*

- *M_{φ} is sound if the pruning strategy is sound.*
- *M_{φ} is perfect if the pruning strategy is perfect.*

Yet we have not given a complexity measure for constraint sets. For our approach we use the number of variables and constants in the constraints, that is $|\mathcal{C}| = \sum_{\varphi \in \mathcal{C}} |\varphi|$ and $|c| = 1$, $|v| = 1$, $|f(e_1, \dots, e_n)| = |e_1| + \dots + |e_n|$, $|\mathit{ite}(e_1, e_2, e_3)| = |e_1| + |e_2| + |e_3|$ for a constant c and a variable v.

4.1 Application to Lola Fragments

We describe now perfect pruning strategies for Lola$_{\mathbb{B}}$ and Lola$_{\mathcal{LA}}$. For Lola$_{\mathbb{B}/\mathcal{LA}}$ we will show that no such perfect pruning strategy exists but present a sound and constant pruning strategy.

Lola$_{\mathbb{B}}$: First we consider the fragment Lola$_{\mathbb{B}}$ where all input and output streams, constants and functions are of type Boolean. Consequently, we assume constraints given to the monitor (input readings, assumptions) also only contain variables, constants and functions of type Boolean.

Example 5. Consider the following specification (where all inputs are uncertain, $\oplus$ denotes exclusive or) shown on the left. The unrolled semantics, shown on the right, indicates that *ok* is always true.

$$\begin{aligned} a &:= a[-1|\mathit{ff}] \oplus x[\mathit{now}] \\ b &:= b[-1|\mathit{tt}] \oplus x[\mathit{now}] \\ \mathit{ok} &:= a[\mathit{now}] \oplus b[\mathit{now}] \end{aligned}$$

0	1	2	3	...
x^0	$x^0 \oplus x^1$	$x^0 \oplus x^1 \oplus x^2$	$x^0 \oplus x^1 \oplus x^2 \oplus x^3$	...
$\neg x^0$	$\neg x^0 \oplus x^1$	$\neg x^0 \oplus x^1 \oplus x^2$	$\neg x^0 \oplus x^1 \oplus x^2 \oplus x^3$	...
tt	*tt*	*tt*	*tt*	...

However, the Boolean formulas maintained internally by the monitor are continuously increasing. Note that at time 1 the possible combinations of (a^1, b^1, ok^1) are (*ff*, *tt*, *tt*) and (*tt*, *ff*, *tt*), as shown below (left). By eliminating duplicates from this table we obtain another table with two columns which can be expressed by formulas over a single, fresh variable v^1 (as shown on the right). From this table we can directly infer the new formulas $a^1 = v^1$, $b^1 = \neg v^1$, $ok^1 = tt$, which preserve the condensed information that a^1 and b^1 are opposites. We can use these new formulas for further calculation. At time 2, $a^2 = v^1 \oplus x^2$, $b^1 = \neg v^1 \oplus x^2$ which we rewrite as $a^2 = v^2$, $b^1 = \neg v^2$ again concluding $ok^1 = tt$. This illustrates how the pruning guarantees a constant-memory monitor. Note that this monitor will be able to infer at every step that *ok* is *tt* even without reading any input. □

(x^0, x^1)	00	01	10	11
a^1	*ff*	*tt*	*tt*	*ff*
b^1	*tt*	*ff*	*ff*	*tt*
ok^1	*tt*	*tt*	*tt*	*tt*

v^1	0	1
a^1	*ff*	*tt*
b^1	*tt*	*ff*
ok^1	*tt*	*tt*

The strategy from the example above can be generalized to a pruning strategy. Let $\mathcal{R} = \{r_1, \ldots, r_m\}$ be the set of relevant variables (in our case the output variables s_i^t) and $\mathcal{V} = \{s_1, \ldots, s_n\} \cup \mathcal{R}$ all variables (in our case input variables and fresh variables from previous pruning applications). Let $\mathcal{C}$ be a set of constraints over $r_1, \ldots, r_m, s_1, \ldots, s_n$, which can be rewritten as a Boolean expression γ by conjoining all constraints.

The method generates a value table T which includes as columns all value combinations of $(v_1, \ldots, v_m)$ for $(r_1, \ldots, r_m)$ such that $(r_1 = v_1) \wedge \cdots \wedge (r_m = v_m) \models \gamma$. Then it builds a new constraint set $\mathcal{C}'$ with an expression $r_i = \psi_i(v_1, \ldots, v_k)$ for every $1 \leq i \leq m$ over k fresh variables, where the ψ_i are generated from the rows of the value table. The number of variables is $k = \lceil \log(c) \rceil$ with c being the number of columns in the table (i.e. combinations of r_i satisfying γ). This method is the Lola$_{\mathbb{B}}$ pruning strategy which is perfect. By Theorem 1 this allows to build a perfect atemporal constant-memory monitor for Lola$_{\mathbb{B}}$.

Lemma 1. *The Lola$_{\mathbb{B}}$ pruning strategy is perfect and constant.*

Lola$_{\mathcal{LA}}$: The same idea used for Lola$_{\mathbb{B}}$ can be adapted to Linear Algebra.

Example 6. Consider the specification on the left. The main idea is that acc_a accumulates the load of CPU A (as indicated by ld_a), and similarly acc_b accumulates the load of CPU B (as indicated by ld_b). Then, $total$ keeps the average of ld_a and ld_b. The unrolled semantics is

$$\begin{aligned} acc_a &:= acc_a[-1|0] + ld_a[now] \\ acc_b &:= acc_b[-1|0] + ld_b[now] \\ total &:= total[-1|0] + \tfrac{1}{2}(ld_a[now]+ \\ &\qquad ld_b[now]) \end{aligned}$$

0	1	2	...
ld_a^0	$ld_a^0 + ld_a^1$	$ld_a^0 + ld_a^1 + ld_a^2$	...
ld_b^0	$ld_b^0 + ld_b^1$	$ld_b^0 + ld_b^1 + ld_b^2$	...
$\frac{1}{2}(ld_a^0 + ld_b^0)$	$\frac{1}{2}((ld_a^0 + ld_b^0) + (ld_a^1 + ld_b^1))$	$\frac{1}{2}((ld_a^0 + ld_b^0) + (ld_a^1 + ld_b^1) + (ld_a^2 + ld_b^2))$	...

Again, the formulas maintained during monitoring are increasing. The formulas at 0 cannot be simplified, but at 1, ld_a^0 and ld_a^1 have exactly the same influence on acc_a^1, acc_b^1 and $total$. To see this consider the calculation of $(acc_a^1, acc_b^1, total^1)$ as the matrix multiplication shown below on the left. The matrix in the middle just contains two linearly independent vectors. Hence the system of equations can be equally written as shown in the right, over two fresh variables u^1, v^1:

$$\begin{pmatrix} acc_a^1 \\ acc_b^1 \\ total^1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{pmatrix} * \begin{pmatrix} ld_a^0 \\ ld_b^0 \\ ld_a^1 \\ ld_b^1 \end{pmatrix} \qquad \begin{pmatrix} acc_a^1 \\ acc_b^1 \\ total^1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix} * \begin{pmatrix} u^1 \\ v^1 \end{pmatrix}$$

The rewritten formulas then again follow directly from the matrix. Repeating the application at all times yields:

0	1	2	...
ld_a^0	$ld_a^0 + ld_a^1 \equiv u^1$	$u^1 + ld_a^2 \equiv u^2$	...
ld_b^0	$ld_b^0 + ld_b^1 \equiv v^1$	$v^1 + ld_b^2 \equiv v^2$	...
$\frac{ld_a^0+ld_b^0}{2}$	$\frac{(ld_a^0+ld_b^0)+(ld_a^1+ld_b^1)}{2} \equiv \frac{u^1+v^1}{2}$	$\frac{(u^1+v^1)+(ld_a^2+ld_b^2)}{2} \equiv \frac{u^2+v^2}{2}$	...

which results in a constant monitor. □

This pruning strategy can be generalized as well. Let $\mathcal{R} = \{r_1, \ldots, r_m\}$ be a set of relevant variables (in our case the output variables s_i^t) and $\mathcal{V} = \{s_1, \ldots, s_n\} \cup \mathcal{R}$ be the other variables (in our case input variables or fresh variables from previous pruning applications). Let $\mathcal{C}$ be a set of constraints maintained by our monitoring algorithm which has to be fulfilled over $r_1, \ldots, r_m, s_1, \ldots, s_n$, which contains equations of the form $c = \sum_{i=1}^{m} c_{r_i} * r_i + \sum_{i=1}^{n} c_{s_i} * s_i + c'$ where c, c', c_{s_i}, c_{r_i} are constants.

If the equation system is unsolvable (which can easily be checked) we return $\mathcal{C}' = \{0 = 1\}$, otherwise we can rewrite it as shown on the left. The matrix N of this equation system has m rows and n columns. Let r be the rank of this matrix which is limited by $\min\{m,n\}$. Consequently an $m \times r$ matrix N' with $r \leq m$ exists with the same span as N and the system can be rewritten (without loosing solutions to $(r_1,\ldots,r_m)$). From this rewritten equation system a new constraint set $\mathcal{C}'$ can be generated which contains the equations from the system. We call this method the Lola$_{\mathcal{LA}}$ pruning strategy, which is perfect and constant.

$$\begin{pmatrix} r_1 \\ \vdots \\ r_m \end{pmatrix} = \begin{pmatrix} c_{1,1} & \ldots & c_{1,n} \\ & \vdots & \\ c_{m,1} & \ldots & c_{m,n} \end{pmatrix} * \begin{pmatrix} s_1 \\ \vdots \\ s_n \end{pmatrix} + \begin{pmatrix} o_1 \\ \vdots \\ o_m \end{pmatrix}$$

Lemma 2. *The Lola$_{\mathcal{LA}}$ pruning strategy is perfect and constant.*

Lola$_{\mathbb{B}/\mathcal{LA}}$. Consider the specification below (left) where i, a and b are input streams of type $\mathbb{R}$. Consider a trace where the values of stream i are unknown until time 2, but that we have the assumption $0 \leq i[now] \leq 1$. The unpruned symbolic expressions describing the values of x, y at time 2 would then be in matrix notation:

$$\begin{aligned} x &:= x[-1|0] + i[now] \\ y &:= 2 * y[-1|0] + i[now] \\ ok &:= (a[now] = x[now]) \wedge (b[now] = y[now]) \end{aligned}$$

$$\begin{pmatrix} x^2 \\ y^2 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 4 & 2 & 1 \end{pmatrix} * \begin{pmatrix} i^0 \\ i^1 \\ i^2 \end{pmatrix}$$

Since the assumption forces all i^j to be between 0 and 1 the possible set of value combinations x and y can take at time 2 is described by a polygon with 6 edges depicted in Fig. 2. Describing this polygon requires 3 vectors. It is easy to see that each new unknown input generates a new vector, which is not multiple of another. Hence for n unknown inputs on stream i the set of possible value combinations for (x^t, y^t) is described by a polygon with $2n$ edges for which a constraint set of size $\mathcal{O}(n)$ is required. This counterexample implies that for Lola$_{\mathbb{B}/\mathcal{LA}}$ there is no perfect pruning strategy. However, one can apply the following approximation: Given a constraint set $\mathcal{C}$ over $\mathcal{V} = \{s_1,\ldots,s_n\} \cup \mathcal{R}$ with $\mathcal{R} = \{r_1,\ldots,r_m\}$ the set of relevant variables.

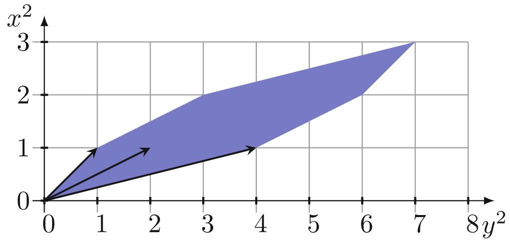


Fig. 2. Set of possible values of x^2 and y^2

1. Split the set of relevant variables into $\mathcal{R}_{\mathbb{B}}$ containing those of type Boolean and $\mathcal{R}_{\mathbb{R}}$ containing those of type Real.
2. For $\mathcal{R}_{\mathbb{B}}$ do the rewriting as for Lola$_{\mathbb{B}}$ obtaining $\mathcal{C}'_{\mathbb{B}}$.
3. For $\mathcal{R}_{\mathbb{R}}$ do the rewriting as for Lola$_{\mathcal{LA}}$ over $\mathcal{C}^{\mathcal{LE}}$ with $\mathcal{C}^{\mathcal{LE}} \subseteq \mathcal{C}$ being the set of all linear equations in $\mathcal{C}$, obtaining $\mathcal{C}'_{\mathbb{R}}$.
4. For all fresh variables v_i with $1 \leq i \leq k$ in $\mathcal{C}'_{\mathbb{R}}$ calculate a minimum bound l_i and maximum bound g_i (may be over-approximating) over the constraints $\mathcal{C} \cup \mathcal{C}'_{\mathbb{R}}$ and build $\mathcal{C}''_{\mathbb{R}} = \mathcal{C}'_{\mathbb{R}} \cup \{l_i \leq v_i \leq g_i | 1 \leq i \leq k\}$.
5. Return $\mathcal{C}' = \mathcal{C}'_{\mathbb{B}} \cup \mathcal{C}''_{\mathbb{R}}$

We call this strategy the Lola$_{\mathbb{B}/\mathcal{LA}}$ pruning strategy, which allows to build an atemporal (imperfect but sound) constant-memory monitor.

Lemma 3. *The Lola$_{\mathbb{B}/\mathcal{LA}}$ pruning strategy is sound and constant.*

Note that with the Lola$_{\mathbb{B}/\mathcal{LA}}$ fragment we can also support if-then-else expressions. A definition $s = ite(c, t, e)$ can be rewritten to handle s as an input stream adding assumption $(c \wedge s = t) \vee (\neg c \wedge s = e)$. After applying this strategy the specification is within the Lola$_{\mathbb{B}/\mathcal{LA}}$ fragment and as a consequence the sound (but imperfect) pruning algorithm from there can be applied.

4.2 Temporal Assumptions

We study now how to handle temporal assumptions. Consider again Example 4, but instead of the assumption $0 \leq ld[now] \leq 10$ take $0.9 * ld[-1, 0] \leq ld[now] \leq 1.1 * ld[-1, 100]$. In this case it would not be possible to apply the presented pruning algorithms. In the pruning process at time 1 we would rewrite our formulas in a fashion that they do not contain ld^1 anymore, but at time 2 we would receive the constraint $0.9 * ld^1 \leq ld^2 \leq 1.1 * ld^1$ from the assumption.

Pruning strategies can be extended to consider variables which may be referenced by input constraints at a later time as relevant variables, hence they will not be pruned. A monitor which receives constraint sets over the last l instants is called an l-lookback monitor. An atemporal monitor is therefore a 0-lookback monitor. For an l-lookback monitor the number of variables that are referenced at a later timestamp is constant, so our pruning strategies remain constant. Hence, the following theorem is applicable to our pruning strategies and as a consequence our solutions for atemporal monitors can be adapted to l-lookback monitors (for constant l).

Theorem 2. *Given a Lola specification φ and a constant pruning strategy $\mathcal{P}$ for $Expr_\varphi^{\mathbb{B}}$ there is a constant-memory l-lookback monitor M_φ such that*

- *M_φ is sound if the pruning strategy is sound.*
- *M_φ is perfect if the pruning strategy is perfect.*

5 Implementation and Empirical Evaluation

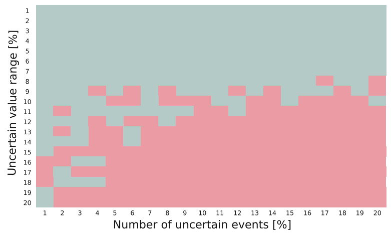
We have developed a prototype implementation of the symbolic algorithm for past-only Lola in Scala, using Z3 [24] as solver. Our tool supports Reals and Booleans with their standard operations, ranges (e.g. $[3, 10.5]$) and ? for unknowns. Assumptions can be encoded using the keyword `ASSUMPTION`.[1] Our tool performs pruning (Sect. 4.1) at every instant, printing precise outputs when possible. If an output value is uncertain the formula and a range of possible values is printed.

[1] Note that for our symbolic approach assumptions can indeed be considered as a stream specification of type Boolean which has to be true at every time instant.

We evaluated two realistic case studies, a test drive data emission monitoring [19] and an electrocardiogram (ECG) peak detector [11]. All measurements were done on a 64-bit Linux machine with an Intel Core i7 and 8 GB RAM. We measured the processing time of single events in our evaluation, for inputs from 0% up to 20% of uncertain values, resulting in average of 25 ms per event (emissions case study) and 97 ms per event (ECG). In both cases the runtime per event did not depend on the length of the trace (as predicted theoretically). The long runtime is in general due to using Z3 naively to deduce bounds of unknown variables, other methods/specialized tools should be investigated in the future. The longer runtime per event in the second case study is explained because of a window of size 100 which is unrolled to 100 streams. We discuss the two case studies separately.

Case Study #1: Emission Monitoring. The first example is a specification that receives test drive data from a car (including speed, altitude, NOx emissions,...) from [19]. The Lola specification is within Lola$_{\mathbb{B}/\mathcal{LA}}$ (with *ite*), and checks several properties, including `trip_valid` which captures if the trip was a valid test ride. The specification contains around 50 stream definitions in total. We used two real trips as inputs, one where the allowed NOx emission was violated and one where the emission specification was satisfied.

We injected uncertainty into the two traces by randomly selecting x% of the events and modifying the value to an uncertainty interval of $\pm y$% around the correct value. The figure on the left shows the result of executing this experiment for all integer combinations of x and y



between 1 and 20, for one trace. The green space represents the cases for which the monitor computed the valid answer and the red space the cases where the monitor reported unknown. In both traces, even with 20% of incorrect samples within an interval of ± 7% around the correct value the monitor was able to compute the correct answer. We also compared these results to the value-range approach, using interval arithmetic. However, the final verdicts do not differ here. Though the symbolic approach is able to calculate more precise intermediate results, these do not differ enough to obtain different final Boolean verdicts.

As expected, for fully unknown values and no assumptions, neither the symbolic nor the interval approaches could compute any certain verdict, because the input values could be arbitrarily large. However, in opposite to the interval approach, the symbolic approach allows adding assumptions (e.g. the speed or altitude does not differ much from the previous value). With this assumption, we received the valid result for `trip_valid` when up to 4% of inputs are fully uncertain. In other words, the capability of symbolic RV to encode physical dependencies as assumptions often allows our technique to compute correct verdicts in the presence of several unknown values.

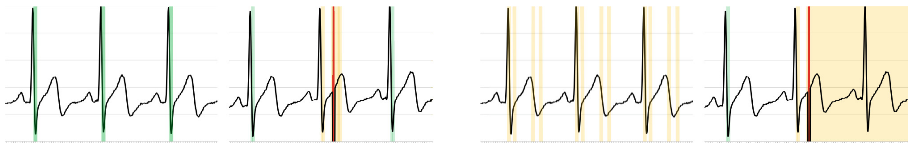


Fig. 3. ECG analysis. Left: symbolic approach, Right: value range approach. Green: certain heartbeats, Yellow: potential heartbeats, Red: bursts of unknown values. (Color figure online)

Case Study #2: Heart Rate Monitoring. Our second case study concerns the peak detection in electrocardiogram (ECG) signals [11]. The specification calculates a sliding average and stores the values of this convoluted stream in a window of size 100. Then it checks if the central value is higher than the 50 previous and the 50 next values to identifying a peak.

We evaluated the specification against a ECG trace with 2700 events corresponding to 14 heartbeats. We integrated uncertainty into the data in two different ways. First, we modified $x\%$ percent of the events to uncertainty intervals of $\pm y\%$. Even if 20% of the values were modified with an error of ±20%, the symbolic approach returned the perfect result, while the interval approach degraded over time because of accumulated uncertainties (many peaks were incorrectly "detected", even under 5% of unknown values with a ±20% error—see front part of traces in Fig. 3). Second, we injected bursts of consecutive errors (? values) of different lengths into the input data. The interval domain approach lost track after the first burst and was unable to recover, while the symbolic approach returned some ? around the area with the bursts and recovered when new values were received (see Fig. 3).

We exploited the ability of symbolic monitors to handle assumptions by encoding that heartbeats must be apart from each other more than 160 steps (roughly 0.5 s), which increased the accuracy. In one example the monitor correctly detected a peak right after a burst of errors. The assumption allowed the monitor to infer values of certain variables from the knowledge that there is no heartbeat, which enabled in turn the detection of the next heartbeat. This is not possible if heartbeats that are not at least 160 steps apart are just filtered out.

6 Conclusion

We have introduced the concept of symbolic Runtime Verification to monitor in the presence of input uncertainties and assumptions on the system behavior. We showed theoretically and empirically that symbolic RV is more precise than monitoring with intervals, and have identified logical theories for which perfect symbolic RV can be implemented in constant memory. Future work includes: (1) to identify other logical theories and their combinations that guarantee perfect trace length independent monitoring; (2) to be able to anticipate verdicts ahead of time for rich data domains by unfolding the symbolic representation of the specification beyond, along the lines of [2,20,30] for Booleans;

Finally, we envision that symbolic Runtime Verification can become a general, foundational approach for monitoring that will allow to explain many existing monitoring approaches as instances of the general schema.

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. Bauer, A., Leucker, M., Schallhart, C.: Monitoring of real-time properties. In: Arun-Kumar, S., Garg, N. (eds.) FSTTCS 2006. LNCS, vol. 4337, pp. 260–272. Springer, Heidelberg (2006). https://doi.org/10.1007/11944836_25
3. Bauer, A., Leucker, M., Schallhart, C.: Comparing LTL semantics for runtime verification. J. Logic Comput. **20**(3), 651–674 (2010)
4. Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y.: Bounded model checking. In: Highly Dependable Soft., chap. 3, pp. 118–149, No. 58 in Advances in Computers 2003. Academic Press (2003)
5. Biewer, S., Finkbeiner, B., Hermanns, H., Köhl, M.A., Schnitzer, Y., Schwenger, M.: RTLola on board: testing real driving emissions on your phone. In: TACAS 2021. LNCS, vol. 12652, pp. 365–372. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72013-1_20
6. Cimatti, A., Tian, C., Tonetta, S.: Assumption-based runtime verification of infinite-state systems. In: Feng, L., Fisman, D. (eds.) RV 2021. LNCS, vol. 12974, pp. 207–227. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-88494-9_11
7. Convent, L., Hungerecker, S., Leucker, M., Scheffel, T., Schmitz, M., Thoma, D.: TeSSLa: temporal stream-based specification language. In: Massoni, T., Mousavi, M.R. (eds.) SBMF 2018. LNCS, vol. 11254, pp. 144–162. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03044-5_10
8. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: POPL, pp. 238–252. ACM (1977)
9. D'Angelo, B., et al.: LOLA: runtime monitoring of synchronous systems. In: Proceeding of TIME 2005, pp. 166–174. IEEE Computer Society (2005)
10. Decker, N., Leucker, M., Thoma, D.: Monitoring mod. theories. STTT **18**(2), 205–225 (2016)
11. Gorostiaga, F., Sánchez, C.: Nested monitors: monitors as expressions to build monitors. In: Feng, L., Fisman, D. (eds.) RV 2021. LNCS, vol. 12974, pp. 164–183. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-88494-9_9
12. Gorostiaga, F., Sánchez, C.: Stream runtime verification of real-time event streams with the Striver language. Int. J. Softw. Tools Technol. Transfer **23**(2), 157–183 (2021). https://doi.org/10.1007/s10009-021-00605-3
13. Havelund, K., Goldberg, A.: Verify your runs. In: Meyer, B., Woodcock, J. (eds.) VSTTE 2005. LNCS, vol. 4171, pp. 374–383. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69149-5_40
14. Havelund, K., Peled, D.: An extension of first-order LTL with rules with application to runtime verification. Int. J. Softw. Tools Technol. Transfer **23**(4), 547–563 (2021). https://doi.org/10.1007/s10009-021-00626-y

15. Henzinger, T.A., Saraç, N.E.: Monitorability under assumptions. In: Deshmukh, J., Ničković, D. (eds.) RV 2020. LNCS, vol. 12399, pp. 3–18. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-60508-7_1
16. Kallwies, H., Leucker, M., Sánchez, C.: Symbolic runtime verification for monitoring under uncertainties and assumptions. arXiv abs/2207.05678 (2022)
17. Kauffman, S., Havelund, K., Fischmeister, S.: What can we monitor over unreliable channels? STTT, 1–24 (2020). https://doi.org/10.1007/s10009-021-00625-z
18. King, J.C.: Symbolic execution and program testing. CACM **19**(7), 385–394 (1976)
19. Köhl, M.A., Hermanns, H., Biewer, S.: Efficient monitoring of real driving emissions. In: Colombo, C., Leucker, M. (eds.) RV 2018. LNCS, vol. 11237, pp. 299–315. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03769-7_17
20. Leucker, M.: Sliding between model checking and runtime verification. In: Qadeer, S., Tasiran, S. (eds.) RV 2012. LNCS, vol. 7687, pp. 82–87. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35632-2_10
21. Leucker, M., Sánchez, C., Scheffel, T., Schmitz, M., Schramm, A.: Tessla: runtime verification of non-synchronized real-time streams. In: SAC 2018, pp. 1925–1933. ACM (2018)
22. Leucker, M., Sánchez, C., Scheffel, T., Schmitz, M., Thoma, D.: Runtime verification for timed event streams with partial information. In: Finkbeiner, B., Mariani, L. (eds.) RV 2019. LNCS, vol. 11757, pp. 273–291. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32079-9_16
23. Leucker, M., Schallhart, C.: A brief account of runtime verification. J. Log. Algebraic Meth. Program. **78**(5), 293–303 (2009)
24. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
25. Pan, J., Tompkins, W.J.: A real-time QRS detection algorithm. IEEE Trans. Biomed. Eng. BME **32**(3), 230–236 (1985)
26. Pnueli, A.: The temporal logic of programs. In: FOCS 1977, pp. 46–57. IEEE (1977)
27. Sznajder, M., Łukowska, M.: Python Online and Offline ECG QRS Detector based on the Pan-Tomkins algorithm (2017)
28. Waga, M., André, É., Hasuo, I.: Symbolic monitoring against specifications parametric in time and data. In: Dillig, I., Tasiran, S. (eds.) CAV 2019. LNCS, vol. 11561, pp. 520–539. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4_30
29. Waga, M., André, É., Hasuo, I.: Model-bounded monitoring of hybrid systems. In: ICCPS, pp. 21–32. ACM (2021)
30. Zhang, X., Leucker, M., Dong, W.: Runtime verification with predictive semantics. In: Goodloe, A.E., Person, S. (eds.) NFM 2012. LNCS, vol. 7226, pp. 418–432. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28891-3_37