Unifying the Time-Event Spectrum for Stream Runtime Verification[⋆]

Felipe Gorostiaga[1,2,3], Luis Miguel Danielsson[1,2], and César Sánchez[1]

[1] IMDEA Software Institute, Spain
[2] Universidad Politécnica de Madrid (UPM), Spain
[3] CIFASIS, Argentina
{felipe.gorostiaga,luismiguel.danielsson,cesar.sanchez}@imdea.org

Abstract. We study the spectra of time-event and of synchronous-asynchronous models of computation for runtime verification, in particular in the context of stream runtime verification (SRV). Most runtime verification formalisms do not involve a notion of time, either by having inputs at all instants (like LTL or Lola) or by reacting to external events in an event-driven fashion (like MOP). Other formalisms consider notions of real-time, ranging from the collection and periodic processing of events to complex computations of the times at which events exist or are produced (like TeSSLa or Striver). Also, some monitoring languages assume that all inputs and outputs change values at once (synchronous), while others allow changes independently (asynchronous).
In this paper we present a unifying view of the event-time and synchronous-asynchronous dimensions in the general setting of SRV. We first prove that the Striver event-based asynchronous language can execute synchronous untimed specifications (written in Lola), and empirically show that this simulation is efficient. We then prove that Lola can simulate real-time Striver monitors under the assumption of the existence of temporal backbones and study two cases: (1) Purely event-driven or when reactions can be precomputed (for example periodic intervals), which results in an efficient simulation but restricted to a fragment. (2) When the time has a minimum quantum: which allows full expressivity but the performance is greatly affected, particularly for sparse input streams.

1 Introduction

Runtime verification (RV) is a dynamic technique for software quality assurance that consists of generating a monitor from a formal specification, that then inspects a single trace of execution of the system under analysis. Stream runtime verification, pioneered by Lola [7], defines monitors by declaring the dependencies between output streams of results and input streams of observations. In this

⋆ This work was funded in part by the Madrid Regional Government under project "S2018/TCS-4339 (BLOQUES-CM)", by Spanish National Project "BOSCO (PGC2018-102210-B-100)", and by FPU Grant FPU18/04362 granted by the Spanish Ministry of Science, Innovation and Universities.

paper we study different models of streams and how the corresponding languages compare to each other in terms of expressivity and efficiency.

Motivated by the counterparts in static verification, early approaches for RV specification languages were based on temporal logics [3, 9, 15], regular expressions [20], timed regular expressions [1], rules [2], or rewriting [18]. SRV is a more expressive formalism that goes beyond Boolean verdicts, like in logical techniques, to allow specifying the collection of statistics and the generation of richer (non-Boolean) verdicts. Examples include counting events, specifying robustness or generating models or quantitative verdicts. See [7, 8, 12, 14] for examples illustrating the expressivity of SRV languages. Some SRV formalisms consider streams to be sequences of raw data (as in LTL propositions), so the data observed in different streams at the same index in their sequences are considered to have occurred at the same time. In this regard, stream sequences are *synchronized* and thus we say that formalisms following this paradigm are *synchronous* SRV formalisms. Examples of synchronous formalisms include Lola [7], LTL, regular-expressions, Mision-time LTL [17], Functional Reactive Programming (FRP) [11] and systems like Copilot [16].

On the other hand, new formalisms have been proposed that consider streams to be sequences of events formed by data values that are time-stamped with the time at which the data is produced (either observed or generated). In this paradigm, streams can be of different length, and the only condition is that the time-stamps are monotonically increasing. As a result, the same position of different streams are not necessarily time-correlated. In this regard, we can say that stream sequences are *asynchronous*, and thus we say that formalisms following this paradigm are *asynchronous* SRV formalisms. Examples of asynchronous SRV formalisms include RTLola [13], Striver [14] and TeSSLa [5].

Synchronous SRV formalisms are best suited for cases when data is periodically gathered for every input stream at the same time from the system under analysis. Asynchronous formalisms are best suited for situations when data on the input streams can be received at unpredictable moments—when something of interest happens—and results can be calculated at any time, not only when an event is observed. By these characteristics, we say that synchronous SRV formalisms are *sample based*, while asynchronous SRV formalisms are *event based*.

In this paper we will use Lola and Striver to show how the semantics of a synchronous SRV formalism can be mimicked by an asynchronous SRV formalism and vice versa. As a corollary, the languages subsumed by each formalism can be automatically translated to the other under the conditions of our results. We also study the impact on efficiency of each approach, and the different alternatives to deal with the loss in performance.

The example specifications and empirical evaluation are based on the real-world data in the dataset Orange4Home [6], which comprises the recording of activities of a single person in an instrumented apartment over the span of four consecutive weeks of work days. This dataset was studied previously in RV using an Execution History Encoding (EHE) in [10].

Contributions and structure. Section 2 contains the preliminaries. Section 3 contains the main contribution: the comparison of two formalisms from different paradigms, and the proof that they are equally expressive. We also describe different alternatives to translate a Striver specification to Lola, which are empirically evaluated in Section 4. Finally, Section 5 concludes.

2 Preliminaries

We recall now Stream Runtime Verification (SRV) briefly (see [7] and the tutorial [19]). The fundamental idea of SRV, pioneered by Lola [7] is to cleanly separate the temporal dependencies from the individual operations to be performed at each step, which leads to the generalization of monitoring algorithms for logics to the computation of richer values.

A *value stream* is a sequence of values from a domain[4]. In this paper we use *sequences* to refer to value streams to distinguish them from event streams. We can refer to the value at the n-th position in a sequence z writing $z(n)$. For example, the sequence $\mathit{co2} = [350, 360, 289, 320, 330]$ contains samples of the level of CO2 in the air (measured in parts-per-million). In this sequence $\mathit{co2}(0) = 350$ and $\mathit{co2}(2) = 289$.

An *event stream* is a succession of events (t, d) where d is a value from a value domain (as in sequences) and t is a time-stamp. Time-stamps are elements of a *temporal domain* (for example $\mathbb{R}$, $\mathbb{Q}$, $\mathbb{Z}$), a set whose elements are totally ordered. The interpretation of the time domain is a global clock, which is common to all the streams in a monitor. The time-stamps in the events of a legal event-stream are monotonically increasing. Nothing prevents a temporal domain from being used as a value domain of some stream, and in fact it is common to define streams that compute and store the passage of time. Given an element t in the temporal domain of an event stream r, we use $r(t)$ to refer to the value with time-stamp t in r. For example, the event-stream $\mathit{tv\_status} = \{(1.5, \mathit{off}), (4.0, \mathit{on}), (6.0, \mathit{off}), (7.5, \mathit{on}), (8.0, \mathit{off})\}$ indicates when a television is turned *on* or *off*. The event $(4.0, \mathit{on})$ in *tv_status* or the fact that $\mathit{tv\_status}(4.0) = \mathit{on}$, indicate that the TV is switched *on* at time 4.0. We will use $z, w \dots$ for sequences and $s, r, \dots$ for event streams. Also, we use t to denote a value of the time domain, and n to range over sequence indices.

Given a positive number N we use $[N]$ for the set $\{0, \dots, N-1\}$. Given a sequence z, we also use $[z]$ for the set of indices of the sequence $[z] = \{0 \dots |z|-1\}$. For example, $[\mathit{co2}] = \{0, 1, 2, 3, 4\}$. Given an event stream s, $\mathit{dom}(s)$ is the set of elements in the temporal domain which have an associated value for s. For example, $\mathit{dom}(\mathit{tv\_status}) = \{1.5, 4.0, 6.0, 7.5, 8.0\}$.

Streams and sequences are typed using arbitrary (interpreted) multi-sorted first-order theories. A type has a collection of symbols used to construct expressions, together with an interpretation of these symbols. The domain of the types is the set of values to be used as data values in sequences and streams,

[4] Even though for past-only specifications the results can be extended to infinite sequences, we use here finite sequences as in [7].

and the interpretation of the symbols is used to evaluate ground expressions. A Lola specification describes monitors by declaratively specifying the relation between output sequences (verdicts) and input sequences (observations). Similarly, a Striver specification describes the relation between output event-streams and input event-streams. We describe these formalisms separately.

2.1 Lola

Syntax Given a set Z of (typed) stream variables, the set of *stream expressions* consists of (1) offsets $v[k, d]$ where v is a stream variable of type D, k is an integer number and d a value from D, and (2) function applications $f(t_1, \ldots, t_n)$ using constructors f from the theories to previously defined terms. Stream variables represent value streams (a.k.a sequences). The intended meaning of expression $v[-1, \mathit{false}]$ is the value of sequence v in the previous position of the trace (or *false* if there is no such previous position, that is, at the beginning). The particular case for an offset with $k = 0$ requires no default value as the index is guaranteed to be within the range of the sequence. Therefore, we will use $v[\mathit{now}]$ for a 0 offset expression. We assume that all theories have a constructor `if · then · else ·` that given an expression of type Bool and two expressions of type D constructs a term of type D. We use $\mathit{Term}_D(Z)$ for the set of stream expressions of type D constructed from variables from Z (and drop Z if clear from the context).

Definition 1 (Lola Specification). *A Lola specification $\varphi\langle I, O\rangle$ consists of a set $I = \{x_1, \ldots, x_m\}$ of input stream variables, a set $O = \{y_1, \ldots, y_n\}$ of output stream variables, and a set of defining equations, $y_i = e_i(x_1, \ldots, x_m, y_1, \ldots, y_n)$ one per output variable $y_i \in O$, where every e_i is an expression from $\mathit{Term}_D(I \cup O)$, and D is the type of y_i.*

A specification describes the relation between input sequences and output sequences. We will use v for an arbitrary variable (where x_i and y_j refer to input and output stream variables respectively).

Example 1. The specification "*the mean level of CO2 in the air in the last 3 instants*", can be expressed as follows, where `denom` calculates the number of instants that are taken into account:

```
input  num co2
output num denom := min(3, denom[-1|0]+1)
output num mean:=(co2[-2|0]+co2[-1|0]+co2[now])/denom[now]
```

□

Semantics An input valuation ρ contains one sequence ρ_x of length L for each input stream variable x, of values of the domain of the type of x. Note that $\rho_x(n)$ is the value at position n of sequence ρ_x (with $0 \leq n < L$). We call ρ_x a valuation of x, and ρ_I the collection of valuations of the set of stream variables I. The intended meaning of a Lola specification is to associate sequences to output stream variables (of the same length L) that satisfy the equations in the specification. Formally, this semantics are defined denotationally as follows. Given a valuation ρ (of all variables in $I \cup O$) the *evaluation* $[\![e]\!]_\rho$ of a term e is a sequence of length L of values of the type of e defined as follows:

– If e is $v[i,c]$ then $[\![v[i,c]]\!]_\rho(j) = [\![v]\!]_\rho(j+i)$ if $0 \leq j+i < L$, and c otherwise.
– If e is $f(e_1,\ldots,e_k)$ then $[\![f(e_1,\ldots,e_k)]\!]_\rho(j) = f([\![e_1]\!]_\rho(j),\ldots,[\![e_k]\!]_\rho(j))$

Note that in particular, $[\![v[\mathit{now}]]\!]_\rho(j) = \rho_v(j)$.

Definition 2 (Evaluation Model). *A valuation $\rho = (\rho_I, \rho_O)$ satisfies a Lola specification φ whenever for every output variable y_i, $[\![y_i]\!]_\rho = [\![e_i]\!]_\rho$. In this case we say that σ is an evaluation model of φ and write $(\sigma_I, \sigma_O) \vDash \varphi$.*

These semantics capture when a candidate valuation is an evaluation model, but the intention of a Lola specification is to compute the unique output sequences given input sequences. A *dependency graph* D_φ of a specification $\varphi\langle I, O\rangle$ is a weighted multi-graph (V, E) whose vertices are the stream variables $V = I \cup O$, and E contains a directed weighted edge $v \xrightarrow{k} y$ whenever $v[k,d]$ is a sub-term in the defining equation of y. If a dependency graph D_φ contains no cycles with 0 weight then the specification is called *well-formed*. Note that well-formedness is equivalent to stating that all cycles in a given maximal strongly connected component (MSCC) M of the dependency graph are positive, or all cycles of M are negative. Well-formedness guarantees that for every ρ_I there is a unique ρ_O such that $(\rho_I, \rho_O) \vDash \varphi$. Essentially, this is because acyclicity guarantees that the value of a sequence at a given position does not depend on itself. A well-formed Lola specification has a unique evaluation model for each input valuation ρ_I and we write $\rho_O = \varphi(\rho_I)$ for this unique output valuation.

Another important concept is the *evaluation graph* which given a length L contains one vertex v_j for every stream variable v and position k. There is an edge from $v_j \rightarrow y_{j+k}$ whenever there is an edge $v \xrightarrow{k} y$ in the dependency graph. For example, if the defining equation of y contains $x[-1,d]$ then y_{16} points to x_{15} in all the evaluation graphs with $L \geq 16$. In well-formed specifications there are no cycles in any evaluation graph, which enables us to reason by induction on evaluation graphs. See [7, 19] for details of these definitions as well as online and offline monitoring algorithms for Lola specifications.

2.2 Striver

Syntax The syntax of Striver is:

$$\alpha ::= \{c\} \mid r.\textbf{ticks} \mid \textbf{delay}\ \epsilon\ s \mid \alpha \cup \alpha \qquad (\textit{tick-expr})$$

$$\tau_x ::= x\texttt{<\~{}}\tau \mid x\texttt{<<}\tau \mid x\texttt{>\~{}}\tau \mid x\texttt{>>}\tau \qquad \tau ::= \textbf{t} \mid \tau_z \text{ for } z \in Z \qquad (\textit{offset-expr})$$

$$E := d \mid x(\tau_x) \mid \mathbf{f}(E_1,\ldots,E_k) \mid \tau \mid \texttt{-out} \mid \texttt{+out} \mid \textbf{notick} \qquad (\textit{value-expr})$$

There are three kinds of expressions:

– *Ticking Expressions*, which define those instants at which a stream may contain a value. Here $c \in \mathbb{T}$, $\epsilon \in \mathbb{T}^+$ are constants (with $\epsilon \neq 0$), r is a stream variable, and $\cup$ is used for the union of instants.

– *Offset Expressions*: which allow fetching time instants at which streams contain values. The expression **t** represents the current instant. The expression x<<τ is used to refer to the previous instant at which x ticked in the past of τ (or $\perp_{\texttt{-out}}$ if there is not such an instant). The expression x<~τ also considers the present as a candidate instant. Analogously, the intended meaning of x>>τ is to refer to the next instant strictly in the future of τ at which x ticks (or $\perp_{\texttt{+out}}$ if there is not such an instant). The expression x>~τ also considers the present as a candidate.
– *Value Expressions*, which compute values. Here, d is a constant of type D, x is a stream variable of type D and **f** is a function symbol of return type D. Note that in $x(\tau_x)$ the value of stream x is fetched at an offset expression indexed by x, which captures the ticking points of x and guarantees the existence of an event if the point is within the time boundaries. Expressions **t** and τ_x build expressions of sort $\mathbb{T}_{out}$. The three additional constants `-out`, `+out` and **notick** allow reasoning (using equality) about accessing both ends of the streams, or not generating an event at a ticking candidate instant.

We use `x(<t,d)` and `x(~t,d)` as syntactic sugar (mimicking `x[-1|d]` and `x[now]` from Lola) as follows

```
x(<t,d) ≝ if x<<t ==-out then d else x(x<<t)
x(~t,d) ≝ if x<~t ==-out then d else x(x<~t)
```

We define the duals `x(t>,d)` and `x(t~,d)` analogously.

Definition 3 (Striver Specification). *A Striver specification $\psi\langle I, O\rangle$ for input stream variables I and output stream variables O, consists of one value expression V_y and one ticking expression T_y for each $y \in O$ (where V_y is of the same type as y, plus the reserved constant* **notick***).*

As for Lola, Striver specifications are often given programmatically as illustrated in the following example.

Example 2. The property "*count for how long has the tv been on*", can be expressed as follows, where stream variable `tv_on` computes the result.

```
input TV_Status tv
ticks tv_on      := tv.ticks
define int tv_on := if tv(<t,off) == on
                    then tv_on(<t,0) + t - tv<<t else 0
```
□

Semantics The semantics of Striver are again defined denotationally. A valuation σ contains one event-stream σ_x for each stream variable in $x \in \{I \cup O\}$. The semantics use valuations to evaluate expressions:

– *Ticking Expressions.* The map $[\![.]\!]_\sigma$ assigns a set of instants to each ticking expression:
  • $[\![\{c\}]\!]_\sigma \overset{\text{def}}{=} \{c\}$ and $[\![a_1 \cup \cdots \cup a_k]\!]_\sigma \overset{\text{def}}{=} [\![a_1]\!]_\sigma \cup \cdots \cup [\![a_k]\!]_\sigma$,

  - $[\![r.\textbf{ticks}]\!]_\sigma \overset{\text{def}}{=} dom(\sigma_r)$, and
  - $[\![\textbf{delay}\ \ \epsilon\ s]\!]_\sigma$ contains the instants $t+v$ such that $(t,v) \in \sigma_s$, unless $|v| < |\epsilon|$ or $sign(v) \neq sign(\epsilon)$ or there is a $(t',v') \in \sigma_s$ with t' between t and $t+v$.
- *Offset Expressions*: $[\![.]\!]_\sigma$ provides, given an instant t, another instant in a valuation σ. In particular, $[\![\mathbf{t}]\!]_\sigma(t) \overset{\text{def}}{=} t$ is the current instant, and
  - $[\![x \texttt{ <<}e]\!]_\sigma$ is the previous instant at which x contains a value strictly before $[\![e]\!]_\sigma(t)$, or $\bot_{\texttt{-out}}$ if either there is no such instant, or if $[\![e]\!]_\sigma(t) = \bot_{\texttt{-out}}$. The expression $[\![x \texttt{ <\~{}}e]\!]_\sigma$ is similar but considers $[\![e]\!]_\sigma(t)$ as a candidate, and
  - $[\![x \texttt{ >>}e]\!]_\sigma$ is the dual of $[\![x \texttt{ <<}e]\!]_\sigma$, looking into the future and returning $\bot_{\texttt{+out}}$ in case it fails. Again, $[\![x \texttt{ >\~{}}e]\!]_\sigma$ is similar to $[\![x \texttt{ >>}e]\!]_\sigma$ but considers the instant $[\![e]\!]_\sigma(t)$ as a candidate.
- *Value Expressions*. The semantics are given in terms of t:
  - $[\![x(e)]\!]_\sigma(t)$ is v for $([\![e]\!]_\sigma(t), v) \in s$, or simply $[\![e]\!]_\sigma(t)$ if it is not an instant,
  - $[\![\mathbf{f}(E_1,\ldots,E_k)]\!]_\sigma(t) \overset{\text{def}}{=} f([\![E_1]\!]_\sigma(t),\ldots,[\![E_k]\!]_\sigma(t))$,
  - $[\![\tau_x]\!]_\sigma(t) \overset{\text{def}}{=} [\![\tau_x]\!]_\sigma(t)$, and $[\![\mathbf{c}]\!]_\sigma(t) \overset{\text{def}}{=} c$, for the constants in the domain and the reserved constants `-out`, `+out` and **notick**.

Evaluating expressions allows defining evaluation models, like in Lola, as those valuations that satisfy all equations (in this case ticking and value equations):

Definition 4 (Evaluation Model). *Given a valuation σ of variables $I \cup O$ the evaluation of the equations for stream $y \in O$ is:*

$$[\![\mathcal{T}_y,\, V_y]\!]_\sigma \overset{def}{=} \{(t,d) \mid t \in [\![\mathcal{T}_y]\!]_\sigma \text{ and } d = [\![V_y]\!]_\sigma(t) \text{ and } d \neq \bot_{\texttt{notick}}\}$$

An evaluation model is a valuation σ such that for every $y \in O$: $\sigma_y = [\![\mathcal{T}_y,\, V_y]\!]_\sigma$.

Similar definitions of dependency graph and well-formedness as the ones stated above for Lola can be given for Striver specifications. The well-formedness condition for Striver includes the condition for Lola (absence of zero-weight cycles). Additionally, well-formedness for Striver requires that closed paths in a given MSCC do not mix positive and negative edges. That is, cycles in a positive MSCC cannot contain negative edges (and cycles in a negative MSCC cannot contain positive edges). Again, we write $\sigma_O = \psi(\sigma_I)$ for the unique output valuation that corresponds to an input valuation. See [14] for details.

3 Time vs Event-Based Runtime Verification

In this section we study how Lola can simulate Striver and vice versa.

We start by introducing transformations between sequences and streams. Since events happen only at time instants, we reserve a special fresh constant $\bot$ (read as "none") to model the absence of an event in a sequence. We extend every value domain A into $A^\bot = A \cup \{\bot\}$. For example, the sequence

$[350, \perp, \perp, 360, \perp, 289, 320, 330, 382]$ is a sequence of $\mathbb{Z}^{\perp}$ values. A sequence of $A^{\perp}$ is called a *maybe sequence* of A values.

We say that a set τ of totally ordered elements with a minimum element such that $|\tau| \geq |[z]|$ *covers* z. When a subset τ of the temporal domain contains the set of time-stamps in a stream s, we say that τ *covers* s. If τ covers every stream in a valuation σ, we say that τ covers σ and we say that τ is a *temporal backbone* of σ. When ordered, τ can be seen as a sequence of increasing time-stamps.

Definition 5. *Let $\tau = \{t_0, t_1 \ldots\}$ be a temporal backbone that covers an event-stream s of sort A, and let z be a maybe sequence of type A (that is, a sequence of $A^{\perp}$ values). We say that s and z are equivalent for τ(and we write $s \equiv_{\tau} z$) whenever $|z| = |\tau|$, τ covers s and for all $n \in [z]$*

$$z(n) = \begin{cases} s(\tau(n)) & \textit{if } \tau(n) \in \textit{dom}(s) \\ \perp & \textit{otherwise} \end{cases}$$

Note that if $s \equiv_{\tau} z$ and $\textit{dom}(s) = \tau$ then $z(n) \neq \perp$ for any n, this is, z is a value sequence of type A when the backbone contains exactly the time-stamps of the events in s.

We now define two maps that transform sequences into event streams and vice versa. The map *tostream* takes a sequence z and a backbone τ and generates an event stream with underlying time-domain $\mathbb{T} \supseteq \tau$, provided that τ covers z. The map *toseq* takes an event stream s and a backbone τ and produces a sequence, provided that τ covers s. These maps are defined as follows:

$$\begin{aligned} \textit{tostream}(z, \tau) &\overset{\text{def}}{=} \{(\tau(n), z(n)) \mid n \in [z] \text{ and } z(n) \neq \perp\} \\ \textit{toseq}(s, \tau)(n) &\overset{\text{def}}{=} \begin{cases} s(\tau(n)) & \text{if } \tau(n) \in \textit{dom}(s) \\ \perp & \text{otherwise} \end{cases} \end{aligned}$$

Example 3. We show the transformations for *co2* and *tv_status* for a backbone $\tau \overset{\text{def}}{=} \{1.0, 1.5, 2.0, 2.5, 4.0, 4.5, 6.0, 7.0, 7.1, 7.2, 7.5, 8.0, 9.0\}$:

$$\begin{aligned} \textit{tostream}(\textit{co2}, \tau) &\overset{\text{def}}{=} \{(1.0, 350), (1.5, 360), (2.0, 289), (2.5, 320), (4.0, 330)\} \\ \textit{toseq}(\textit{tv\_status}, \tau) &\overset{\text{def}}{=} [\perp, \textit{off}, \perp, \perp, \textit{on}, \perp, \textit{off}, \perp, \perp, \perp, \textit{on}, \textit{off}, \perp] \end{aligned}$$ □

The following lemma relates *tostream* and *toseq*.

Lemma 1. *For every sequence z, event-stream s and backbone τ that covers s:*
- *$z \equiv_{[z]} \textit{tostream}(z, [z])$ and $z = \textit{toseq}(\textit{tostream}(z, [z]), [z])$*
- *$s \equiv_{\tau} \textit{toseq}(s, \tau)$ and $s = \textit{tostream}(\textit{toseq}(s, \tau), \tau)$.*

The previous definitions can be extended to collections of streams and to valuations as follows. Let V be a set of stream variables, σ be a collection of event-streams with one stream σ_v in σ for every $v \in V$, and let τ be a backbone that covers σ. Let ρ be a collection of $A^{\perp}$, with a sequence ρ_v for each variable v. We say that σ is equivalent to ρ for backbone τ over the streams V, and write $\sigma \equiv_{\tau}^{V} \rho$ whenever for all $v \in V$, $\sigma_v \equiv_{\tau} \rho_v$.

Fig. 1 shows the main result proven in the rest of this section.

$$\begin{tikzcd}
\rho_I \arrow[r, "\overset{\textit{Lola}}{\varphi}"{name=U}] \arrow[d, dashed, "\textit{tostream}"'] & \rho_O \arrow[d, dashed, leftrightarrow, "\equiv^O_\tau"] \\
\sigma_I \arrow[r, "\underset{\textit{Striver}}{\psi}"'{name=D}] & \sigma_{O'}
\arrow[from=U, to=D, dashed, "\textit{striver}"]
\end{tikzcd}
\qquad
\begin{tikzcd}
\sigma_I \arrow[r, "\overset{\textit{Striver}}{\psi}"{name=U}] \arrow[d, dashed, "\textit{toseq}"'] & \sigma_O \arrow[d, dashed, leftrightarrow, "\equiv^O_\tau"] \\
\rho_I \arrow[r, "\underset{\textit{Lola}}{\varphi}"'{name=D}] & \rho_{O'}
\arrow[from=U, to=D, dashed, "\textit{lola}"]
\end{tikzcd}$$

Fig. 1. Commutative diagrams for Theorems 1 and 2.

3.1 From Lola to Striver

We show now how to translate a synchronous specification (written in Lola) into an event-based specification (written in Striver) that generate equivalent outputs from equivalent inputs. Formally, we start from a well-formed Lola specification $\varphi\langle I, O\rangle$ and generate a well-formed Striver specification $\psi\langle I, O'\rangle$ with $O \subseteq O'$ (the equivalence will be restricted to $I \cup O$ as $O' \setminus O$ are auxiliary streams). We will show that for an evaluation model ρ of φ, if we choose $\tau = [\rho]$ (the instants of time $\{0 \dots |\rho - 1|\}$) as the time backbone, then for every evaluation model of ψ such that $\sigma \equiv^I_\tau \rho$ then $\sigma \equiv^O_\tau \rho$. This is, if the evaluation models coincide in their inputs with respect to the backbone τ, then the evaluation models coincide in the output streams of the Lola specification with respect to τ. We assume that the Lola specification is *flattened*, this is, the specification only contains stream accesses with offsets -1, 0 or 1. This has been proved to be feasible in [7].

Recall that well-formed Lola specifications require that each MSCC of the dependency graph has only positive cycles or only negative cycles. Additionally, well-formed Striver specifications also require that cycles in positive MSCCs have no negative edges and cycles in negative MSCCs have no positive edges. The reason is that in real-time, a single negative edge (corresponding to a future reference fetching an event in the past) can compensate for any number of positive edges and vice versa, and create a zero-weight cycle. Therefore, if we attempt to simply translate a Lola successor access $x[+1|d]$ as a Striver next event access $x(\mathtt{t>}, d)$, we may turn a well-formed Lola spec into an illegal Striver spec.

To overcome this issue, the translation proceeds in two stages. First, we translate the initial Lola specification into an equivalent Lola specification that does not contain positive edges in negative MSCCs or negative edges in positive MSCCs. This can be done for every Lola specification. In the second stage, we translate the resulting Lola specification into a Striver specification.

Eliminating mixed edges in MSCCs. We show now the translation for removing positive offsets from negative MSCCs. Removing negative offsets from positive MSCCs is dual. We introduce an auxiliary function $\mathit{exp}_M(e, k)$ that expands recursively the offsets within a negative MSCC M by the constant k. The expansion substitutes the definition of the referred stream making all offsets become non-negative:

$$\begin{aligned} exp_M(x[i|d],k) &\overset{\text{def}}{=} \begin{cases} x[i+k|d] & \text{if } x \notin M \text{ or } k+i \leq 0 \\ \begin{pmatrix} \texttt{if } \mathit{true}[k+i|\mathit{false}] \\ \texttt{then } exp_M(e_x, k+i) \\ \texttt{else } d \end{pmatrix} & \text{otherwise} \end{cases} \\ exp_M(d,i) &\overset{\text{def}}{=} d \\ exp_M(\mathbf{f}(E_1,\ldots,E_l),i) &\overset{\text{def}}{=} \mathbf{f}(exp_M(E_1,i),\ldots,exp_M(E_l,i)) \end{aligned}$$

Note that the recursive expansion terminates because each expansion corresponds to following an additional edge and all paths in the negative MSCC eventually either leave the MSCC, or make the path negative. Finally, we rewrite the term of every stream $x = e \in M$ as

$$\texttt{output x :=} exp_M(e_x, 0)$$

Example 4. Take for example the following Lola specification:

```
output x = y[1|999] + 1
output y = x[-2|5] * 2
```

The resulting specification after the expansion is:

```
x = ((if true[1|false] then x[-1|5] * 2) else 999) + 1
y = x[-2|5] * 2
```

The correctness of the translation is provided by the following lemma.

Lemma 2. *Let φ be a Lola spec and φ' be the resulting specification after the defining equation e_x is replaced by $exp_M(e_x, 0)$. Then, φ and φ' are equivalent.*

The proof essentially proceeds by showing that given a candidate valuation $[\![e_x]\!]$, then $[\![e_x]\!] = [\![exp(e_x, 0)]\!]$ by structural induction on the expressions.

The application of exp_M guarantees that, given a well-formed Lola specification φ, the obtained equivalent Lola specification φ' satisfies that every MSCC in its dependency graph contains only positive edges or only negative edges in every cycle.

Translation to Striver. In the second stage of the translation, we define a function *striver* that generates a Striver value expression from the defining expression of a Lola stream.

$$\begin{aligned} striver(x[-1|d]) &\overset{\text{def}}{=} x(\texttt{<}\mathbf{t}, d) \\ striver(x[now]) &\overset{\text{def}}{=} \begin{cases} x(\texttt{\~{}}\mathbf{t}) & \text{if } x \text{ belongs to a non-positive MSCC} \\ x(\mathbf{t}\texttt{\~{}}) & \text{otherwise} \end{cases} \\ striver(x[1|d]) &\overset{\text{def}}{=} x(\mathbf{t}\texttt{>}, d) \\ striver(d) &\overset{\text{def}}{=} d \\ striver(\mathbf{f}(E_1,\ldots,E_k)) &\overset{\text{def}}{=} \mathbf{f}(striver(E_1),\ldots,striver(E_k)) \end{aligned}$$

For every output stream $x = e_x$ of type A in φ, we define its equivalent in ψ as:

```
ticks  x    := r.ticks
define a x  :=  striver(e_x)
```

where r is any input stream.

We use *striver*(φ) to refer to the Striver specification resulting by translating all output streams of φ as described above. We prove now that φ and *striver*(φ) are equivalent, so therefore Striver can simulate Lola specifications.

Theorem 1. *Let φ be a well-formed Lola specification and ρ_I a valuation of its inputs of length N. Let $\psi = \mathit{striver}(\varphi)$ be the Striver specification obtained by translating φ and let $\sigma_I = \mathit{tostream}(\rho_I, [N])$. Then, $\psi(\sigma_I) \equiv_{[N]} \varphi(\rho_I)$.*

Proof. Let φ be a Lola specification, and $\psi = \mathit{striver}(\varphi)$ its translation to Striver. Let ρ_I be an input valuation of φ of size N, and let $\tau = 0 \ldots N-1$, and let $\rho_O = \varphi(\rho_I)$. Let $\sigma_I = \mathit{tostream}(\sigma_I, \tau)$ be the corresponding input valuation for ψ and $\sigma_O = \psi(\sigma_I)$.

If $N = 0$, then $\rho_x = \langle\rangle$ and $\sigma_x = \{\}$ for every stream x in $I \cup O$.

We see now the case of $N > 0$. First, we observe that for the case of specs φ without mixed-edges the dependency graphs of φ and ψ are identical. We proceed by induction over a topological sort of the acyclic graph of MSCCs in the dependency graphs (the graph of MSCCs). By induction over $0 \ldots N-1$ for negative MSCCs (and by induction over $N-1 \ldots 0$ for positive MSCCs). Internally, we reason by induction over a topological sort of the MSCC with $\xrightarrow{-}$ and $\xrightarrow{+}$ edges removed.

Let x be a stream variable in a non-positive MSCC, and let $i \in \tau$. We know that $i \in \mathit{dom}(x)$ because $i \in \mathit{dom}(r)$, for any input stream r. Let $v = \rho_x(i)$ be the value at position i in σ_x. We consider the cases separately.

- The definition of x in φ is `x=v`, and thus the definition of the value of x in ψ is `x=v` and $\sigma_x(i) = v$, or
- The definition of x in φ is `x=y[now]`. Then, the corresponding definition of x in ψ is `x=y(~t)`. Since $\rho_y(i) = v$ then also $\sigma_y(i) = v$ (by induction hypothesis), and hence $\sigma_x(i) = v$
- The definition of x in φ is `x=y[-1|d]`, and thus the definition of the value of x in ψ is `x=y(<t,d)`. Then, either:
  - $i = 0$ and $d = v$, and $\sigma_x(i) = v$, or
  - $i > 0$ and $\rho_y(i-1) = v$, and $\sigma_y(i-1) = v$ (by induction hypothesis over i), and $\sigma_x(i) = v$.
- The definition of x in φ is `x=f(e1,...,ek)`, and thus the definition of the value of x in ψ is `x=f(e1',...,ek')`. In this case, we proceed by structural induction over the expression. The leaves fall within one of the previous cases. Since the arguments of every function are the same, they produce the same result. We apply this reasoning until we get to the topmost expression `f(e1',...,ek')`, where f is applied to the same arguments as in its Lola counterpart expression `f(e1,...,ek)`, and thus the result is v in both cases; and $\sigma'_x(i) = v$.

The proof for positive MSCCs is analogous. □

Example 5. Let φ be the specification from Example 1 and the sequence for *co2* in the preliminaries. The equivalent flattened specification is:

```
input  num co2
output num aux    := co2[-1|0]
output num denom  := min(3, denom [-1|0]+1)
output num mean  :=(aux[-1|0]+co2[-1|0]+co2[now])/denom[now]
```

The evaluation model for $\rho_{co2} = [350, 360, 289, 320, 330]$ is

$$\begin{aligned} \rho_{aux} &= [\quad 0,\ 350,\ 360,\ 289,\ 320] \qquad \rho_{denom} = [\ 1,\ 2,\ 3,\ 3,\ 3] \\ \rho_{mean} &= [\ 350,\ 355,\ 333,\ 323,\ 313] \end{aligned}$$

The translated Striver specification is:

```
input num co2

ticks aux          := co2.ticks
define num aux     := co2(<t,0)
ticks denom        := co2.ticks
define num denom   := min(3, denom(<t,0)+1)
ticks mean        := co2.ticks
define num mean  :=(aux(<t,0)+co2(<t,0)+co2(~t))/denom(~t)
```

And its evaluation model for $\tau \overset{\text{def}}{=} 0, \ldots, 4$ is:

$$\begin{aligned} \sigma_{co2} &= \{(0, 350), (1, 360), (2, 289), (3, 320), (4, 330)\} \\ \sigma_{aux} &= \{(0, 0), (1, 350), (2, 360), (3, 289), (4, 320)\} \\ \sigma_{denom} &= \{(0, 1), (1, 2), (2, 3), (3, 3), (4, 3)\} \\ \sigma_{mean} &= \{(0, 350), (1, 355), (2, 333), (3, 323), (4, 313)\} \end{aligned}$$

3.2 Striver to Lola

We show now how to translate a well-formed Striver specification $\psi\langle I, O\rangle$ to an equivalent well-formed Lola specification $\varphi\langle I, O'\rangle$ with the same input I and output stream variables O' (again $O \subseteq O'$ because O' contains some auxiliary stream variables). In this case we do not impose a temporal backbone τ. Instead, we will show the conditions that τ must meet for the translation to be correct. We reserve the word `notick` in the syntax of Lola to refer to the reserved constant $\perp$. To ease the translation we introduce a new Lola stream variable called *time* and assume that $toseq(\sigma, \tau)$ assigns $time(i) = \tau(i)$ for every instant i in σ.

The main idea of the translation is to create a defining expression for every stream variable $x \in O$ of type A, with ticking expression T_x and value expression V_x, as follows:

output A x := if $ticks(T_x)$ then $value(V_x)$ else notick

where $ticks(T_x)$ is a Boolean expression that is true whenever x has a value at the time corresponding to the instant, and $value(V_x)$ is an expression that computes the corresponding value.

We first define *ticks*, which given a *ticking expression* in Striver, returns a Boolean expression in Lola.

$$
\begin{aligned}
\mathit{ticks}(\{c\}) &\overset{\text{def}}{=} \texttt{time[now]==c} \\
\mathit{ticks}(x.\mathbf{ticks}) &\overset{\text{def}}{=} \texttt{x[now]!=notick} \\
\mathit{ticks}(x \cup y) &\overset{\text{def}}{=} \mathit{ticks}(x)\texttt{||}\mathit{ticks}(y) \\
\mathit{ticks}(\mathbf{delay}\ \epsilon\ x) &\overset{\text{def}}{=} \begin{cases} \texttt{delay\_eps\_x [-1|noalarm]==time[now]} & \text{if } \epsilon > 0 \\ \texttt{ndelay\_eps\_x [+1|noalarm]==time[now]} & \text{otherwise} \end{cases}
\end{aligned}
$$

where, for each x and ϵ used in an expression **delay** ϵ x, we add to $\mathit{lola}(\psi)$ the following stream variable delay_ϵ_x and ndelay_ϵ_x with defining expressions:

```
output Time ∪{noalarm}  delay_ε_x := if x[now]<ε then noalarm
       else if x[now] == notick then delay_ε_x [-1|noalarm]
       else x[now] + time[now]
output Time ∪{noalarm} ndelay_ε_x := if x[now]>ε then noalarm
       else if x[now] == notick then ndelay_ε_x [+1|noalarm]
       else x[now] + time[now]
```

Here, noalarm is a fresh value not in $\mathbb{T}$.

We now define the function *value*, which translates Striver value expressions into Lola expressions of the same type. We assume that the Striver specification is *flattened* so it does not contain nested offset expressions.[5]

$$
\begin{aligned}
\mathit{value}(x(x \texttt{<<}\mathbf{t})) &\overset{\text{def}}{=} \texttt{prev\_x[now]} & \mathit{value}(d) &\overset{\text{def}}{=} \texttt{d} \\
\mathit{value}(x(x \texttt{<\textasciitilde}\mathbf{t})) &\overset{\text{def}}{=} \texttt{preveq\_x[now]} & \mathit{value}(\texttt{-out}) &\overset{\text{def}}{=} \texttt{-out} \\
\mathit{value}(x(x \texttt{>>}\mathbf{t})) &\overset{\text{def}}{=} \texttt{succ\_x[now]} & \mathit{value}(\texttt{+out}) &\overset{\text{def}}{=} \texttt{+out} \\
\mathit{value}(x(x \texttt{>\textasciitilde}\mathbf{t})) &\overset{\text{def}}{=} \texttt{succeq\_x[now]} & \mathit{value}(\mathbf{notick}) &\overset{\text{def}}{=} \texttt{notick} \\
\mathit{value}(\mathbf{t}) &\overset{\text{def}}{=} \texttt{time[now]} & \mathit{value}(\mathbf{f}(E_1,\ldots)) &\overset{\text{def}}{=} \texttt{f(}\mathit{value}(E_1)\texttt{,...)}
\end{aligned}
$$

where, for every stream x, we define

```
preveq_x:=if x[now]!=notick then x[now] else preveq_x[-1|-out]
prev_x  :=preveq_x [-1|-out]
succeq_x:=if x[now]!=notick then x[now] else succeq_x[+1|+out]
succ_x  :=succeq_x [+1|+out]
```

Essentially, the new streams preveq_x search for the previous value in x that contains an actual value. The other auxiliary streams are analogous. Note that offsets are restricted to values, not times. A specification that contains offset expressions that access time can be translated to an equivalent one accessing values creating a Striver stream times_of_x. As a result, we will get a stream prev_times_of_x, along with the rest of the auxiliary streams in the translated Lola specification.

[5] An algorithm to get a flatten specification for a past-only Striver specification has been shown in [14].

We use $lola(\psi)$ for the Lola specification obtained by transforming every output stream variable in ψ as described above. Theorem 2 below captures whether the transformation gives an equivalent Lola specification, which depends on the temporal backbone being covering.

Theorem 2. *Let $\psi\langle I, O\rangle$ be a well-formed Striver specification, σ_I a valuation of the inputs of ψ, τ be a covering temporal backbone, and $\rho_I = toseq(\sigma_I, \tau)$. Let $\varphi = lola(\psi)$. Then, $\psi(\sigma_I) \equiv^O_\tau \varphi(\rho_I)$.*

Proof (sketch). The proof proceeds by complete induction on the evaluation graph of ψ for ρ (which is an acyclic graph). Essentially, if the equivalence does not hold there is a node (corresponding to a stream variable at a concrete position) that is minimal—in the sense that it violates the stated equivalence but all the lower nodes satisfy it—. Since in both cases the value of the node only depends on lower nodes with two expressions that guarantee the same results (given the values on the nodes they depend on), a contradiction is reached. □

Example 6. Let ψ be the specification of Example 2 and the sequence for *tv_status* from Section 2 The evaluation model is:

$$\begin{aligned}\sigma_{tv\_status} &= \{(1.5, \mathit{off}), (4.0, \mathit{on}), (6.0, \mathit{off}), (7.5, \mathit{on}), (8.0, \mathit{off})\}\\ \sigma_{tv\_on} &= \{(1.5, 0.0), (4.0, 0.0), (6.0, 2.0), (7.5, 0.0), (8.0, 0.5)\}\end{aligned}$$

The translated specification is:

```
input  TV_Status tv_status
output int tv_on :=
   if tv_status[now] != notick then
      if prev_tv_status[now] then
      prev_tv_on[now]+time[now]-prev_times_of_tv_status[now]
      else 0
   else notick
```

Its evaluation model for the covering backbone $\tau = \{1.5, 4.0, 6.0, 7.5, 8.0\}$ is, assuming $\rho_{tv\_status} = [\mathit{off}, \mathit{on}, \mathit{off}, \mathit{on}, \mathit{off}]$:

$$\rho_{tv\_on} = [0.0, 0.0, 2.0, 0.0, 0.5] \qquad \rho_{time} = [1.5, 4.0, 6.0, 7.5, 8.0]$$ □

3.3 Time Backbone Election

The main result of the previous section is Theorem 2, which establishes a translation from Striver to Lola, and a condition under which the translation is correct. Namely, that a temporal backbone is chosen in the translation satisfying that the sequence of times in the backbone contains all the instants where events may happen at runtime. We now describe three cases to compute a temporal backbone that satisfies the conditions of Theorem 2 and later in Section 4 evaluate the efficiency of the resulting monitors.

Full Time-Domain The first obvious choice is to use the minimum granularity of time that the monitoring system considers (this can be one millisecond, one second, etc. depending on the setting). In this case, $\mathbb{T}$ is a finite set (given a starting and finishing time) and choosing $\tau = \mathbb{T}$ guarantees trivially to cover all event-streams of any valuation of the Striver specification. We call the resulting Lola specification the *full-time translation*. As we will see, this approach becomes very inefficient if $|\mathbb{T}| \gg |\bigcup_{x \in I \cup O} dom(x)|$. We use *density* to refer to the ratio of instants at which there are events in a given valuation, and *sparsity* for how close together events are statistically. The less dense a valuation is, the more inefficient $lola(\psi)$ is compared to ψ when a full-time translation is used.

Input Timestamps Sometimes, it can be guaranteed that a Striver specification is purely event-driven. In other words, all output events of all valuations happen only at instants where there are input-events. The fragment of Striver specifications whose tick operators are restricted to **ticks** and U (i.e. { c } and **delay** are not used) is called *event-driven* and satisfies the following proposition.

Proposition 1. *Let ψ be an event-driven Striver specification, σ_I an input valuation and $\sigma_O = \psi(\sigma_I)$. Then, $\bigcup_{y \in O} dom(y) \subseteq \bigcup_{x \in I} dom(x)$.*

In other words, $\bigcup_{v \in I \cup O} dom(v) = \bigcup_{x \in I} dom(x)$ for event-driven specifications. As a result, we can incrementally define τ as the witnessed input timestamps, and an incremental online Lola engine will be correct. We call the corresponding Lola specification the *event-driven translation*. As we will see, this translation is very efficient regardless of the density or sparsity of the streams observed. However, unfortunately, this choice of backbone only supports a fragment of the Striver language.

Input-independent Timestamps A third translation considers the case under which one can statically determine that any valuation will only contain either time instants dictated by the input (event-driven) or a set of instants which may require time calculation but that does not depend on the input.

This happens, for example, when **delay** is used in a controlled way to define periodic clocks (that only depend on themselves in a recursive definition with **delay**), and not depending on instants in the input. Mixing event-driven and periodic clocks allows us to capture the assumptions of the RTLola real-time SRV tool [13]. We call the resulting Lola specification the *isochronous translation*.

4 Empirical Evaluation

In this section we report an empirical evaluation, executed on a MacBook Pro with a Dual Core Intel-i5 at 2.5GHz with 8GB of RAM running MacOS Catalina. The Lola monitors are generated using HLola, a Haskell implementation of Lola described in [4] and the Striver monitors are generated using HStriver, a similar infrastructure for Striver[6]. We evaluate empirically the following hypotheses:

- (H1) Lola can be simulated by Striver with little penalty in time.
- (H2) Striver can be simulated by Lola via a full-time translation, but the resource penalty can be very large, particularly for low density inputs.
- (H3) If the Striver specification is purely event-driven then the event-driven translation into Lola can simulate it very efficiently.
- (H4) If the Striver contains only event driven streams or periodic streams, it can also be efficiently simulated via an isochronous translation.
- (H5) In practice, embedded monitoring execution platforms can either enter idle mode immediately after processing an event or remain awake waiting for events to be received shortly. Resources for sparse inputs can be reduced by choosing an optimal patience time before entering idle mode.

To evaluate these hypotheses we have written a number of specifications in HLola and HStriver for properties over the Orange4Home data-set [6]. The translations were computed manually following the algorithms in Section 3. Consider for example S1: "*the person does not watch TV for longer than 3 hours a day*", which involves detecting the beginning and end of a TV watching session and computing the total TV watching time during a day. A second specification S2: "*the person does not watch TV more than 30 minutes more than the daily average in the past*", requires also computing and maintaining numerical calculations from previous days (note that this specification is not expressible in LTL). This specification also requires events to be generated on the fly at time instants that are neither input-driven nor periodic.

In the first experiment we assess (H1) starting from Lola specifications for S1 and S2, and translating them into Striver. We assume that the events are spaced roughly by one minute during a day and ignore the seconds, which is reasonable in our dataset.

In the second experiment we assess (H2) by re-implemening S1 and S2 in Striver to consider the time of the inputs for their computation. This makes the monitor more precise and allows it to report the excess of TV exactly at the moment the property is violated (at that time no input event occurs in general). The HLola implementation is slightly changed to expect events to be one second apart, making the HLola specification as precise as its HStriver counterpart, but also causing it to be much more inefficient.

In the third experiment we evaluate (H3) running a simpler event-driven Striver Boolean specification S3: "*there is some TV on in the house*" and the numeric S4: "*total TV time when TV is switched-off*", as well as the Lola equivalent that uses the timestamps of the TV events as a backbone.

In the fourth experiment we evaluate (H4) running a spec S5 that calculates the summary of TV time at the end of every hour.

[6] Both HLola and HStriver are open source available from http://github.com/imdea-software. All executions in this empirical evaluation are packaged as a docker container downloadable from http://hub.docker.io/imdea-software/rv2020/.

	Event throughput									
	2h38m		9h18m		1d8h46m		11d4h27m		25d2h6m	
	HStriver	HLola	HStriver	HLola	HStriver	HLola	HStriver	HLola	HStriver	HLola
Exp 1	6666	5050	7142	5577	7692	5900	8064	5830	8130	5279
Exp 2	4000	6407	7142	6377	7407	6264	7692	6145	7462	6149
Exp 3	6666	12900	11111	8933	11764	10650	11904	10432	12195	10528
Exp 4	4000	6550	10000	9083	10000	8438	12345	9182	12820	9376

Table 1. Experiments data

We calculate, for each experiment and a number of running traces, the average number of processed events per second. In each experiment we use a translator that generates the equivalent sequences and event-streams of varying time spans. The summary of the observations can be found in Table 1, Fig. 2, Fig. 3(a) and Fig. 3(b).

The results in Table 1 suggest that the event processing throughput is unaffected by the number of events/instants being processed (all specifications are trace-length independent), as predicted. Also, we observe that the event processing throughput for both HLola and HStriver are similar for the same experiment. Fig. 2 shows that the second experiment increases exponentially for Lola with respect to the trace length (note that the y axis has a logarithmic scale), but increases linearly with respect to the trace length in all other cases. The number of events processed by Striver is roughly twice as its Lola counterpart (except in experiment 2), since data from two different origins with the same timestamp accounts for one event in Lola, but Striver processes them separately.

Fig. 3(a) reports a variation of experiment 3 where the input sequence is padded with empty data to evaluate the impact on the performance of Lola. For reference, the blue line at 1.64 indicates the execution time of the corresponding HStriver specification. We can conclude that the execution time increases linearly with respect to the density of the input data, as expected.

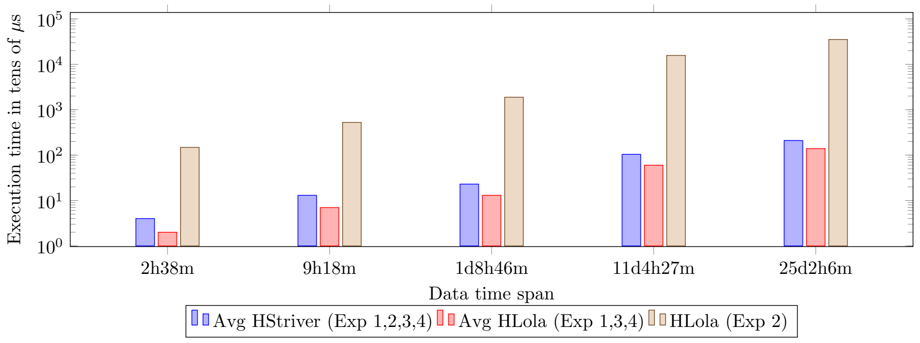

Fig. 2. Execution time

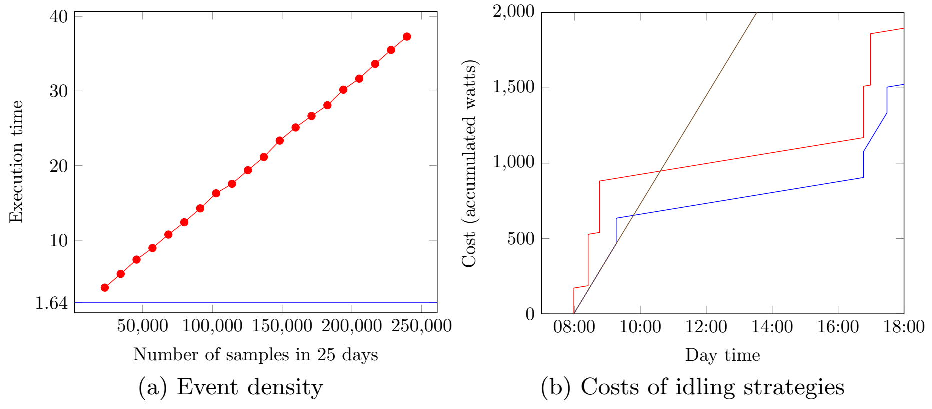

Fig. 3. Event density (left) and costs (right).

Finally, we run an additional experiment to evaluate (H5), with a synthetic cost model that considers the energy of going idle and waking up (Fig. 3(b)). The brown line indicates the accumulated costs of a monitor that never goes idle. The red line represents a monitor that goes idle immediately after processing every event. The jumps in the red line correspond to the times at which an event was received. The blue line corresponds to a monitor that waits for half an hour after the last event processed to go idle. The outcome illustrates that waiting is favourable if the next event comes soon, while sleeping is preferred if the next event takes long to arrive.

5 Conclusions

We have studied the conditions under which synchronous monitoring and fully asynchronous monitoring can simulate each other, particularly in the context of stream runtime verification. Our first result is that every Lola specification can be efficiently simulated by Striver. The second result is the definition of a condition of the temporal backbone under which Lola can simulate Striver, via a general translation, leading to three practical translations: (1) the full-time translation that uses the minimum granularity of time, which is general but inefficient; (2) the event-driven translation, which is efficient but restricted to event-driven Striver specs; and (3) the asynchronous translation, a mixed approach that supports event-driven execution plus simple time-driven events like periodic clocks.

A simple analysis of the translations (from Lola to Striver and vice versa) presented in Section 3 reveals that the resulting specification is linear in the size of the original one, and that the algorithm takes linear time. Similar translations can be made for other SRV specification languages like TeSSLa and RTLola. Our empirical evaluation using the Orange4Home dataset illustrates the expressivity of the SRV languages used and allowed us to empirically confirm the predictions on the runtime efficiency of the corresponding monitors.

References

1. Eugene Asarin, Paul Caspi, and Oded Maler. Timed regular expressions. *J. ACM*, 49(2):172–206, 2002.
2. Howard Barringer, Allen Goldberg, Klaus Havelund, and Koushik Sen. Rule-based runtime verification. In *Proc. of the 5th Int'l Conf. on Verification, Model Checking and Abstract Interpretation (VMCAI'04)*, volume 2937 of *LNCS*, pages 44–57. Springer, 2004.
3. Andreas Bauer, Martin Leucker, and Chrisitan Schallhart. Runtime verification for LTL and TLTL. *ACM Transactions on Software Engineering and Methodology*, 20(4):14, 2011.
4. Martín Ceresa, Felipe Gorostiaga, and César Sánchez. Declarative stream runtime verification (hLola). Under submission, 2020.
5. Lukas Convent, Sebastian Hungerecker, Martin Leucker, Torben Scheffel, Malte Schmitz, and Daniel Thoma. TeSSLa: Temporal stream-based specification language. In *Proc. of the 21th Brazilian Symp. on Formal Methods (SBMF'18)*, volume 11254 of *LNCS*, pages 144–162. Springer, 2018.
6. Julien Cumin, Grgoire Lefebvre, Fano Ramparany, and James Crowley. A dataset of routine daily activities in an instrumented home. pages 413–425, 11 2017.
7. Ben D'Angelo, Sriram Sankaranarayanan, César Sánchez, Will Robinson, Bernd Finkbeiner, Henny B. Sipma, Sandeep Mehrotra, and Zohar Manna. LOLA: Runtime monitoring of synchronous systems. In *Proc. of the 12th Int'l Symp. of Temporal Representation and Reasoning (TIME'05)*, pages 166–174. IEEE CS Press, 2005.
8. Luis Miguel Danielsson and César Sánchez. Decentralized stream runtime verification. In Bernd Finkbeiner and Leonardo Mariani, editors, *Runtime Verification - 19th International Conference, RV 2019, Porto, Portugal, October 8-11, 2019, Proceedings*, volume 11757 of *Lecture Notes in Computer Science*, pages 185–201. Springer, 2019.
9. Cindy Eisner, Dana Fisman, John Havlicek, Yoad Lustig, Anthony McIsaac, and David Van Campenhout. Reasoning with temporal logic on truncated paths. In *Proc. of the 15th Int'l Conf. on Computer Aided Verification (CAV'03)*, volume 2725 of *LNCS*, pages 27–39. Springer, 2003.
10. Antoine El-Hokayem and Yliès Falcone. Bringing runtime verification home. In Christian Colombo and Martin Leucker, editors, *Runtime Verification - 18th International Conference, RV 2018, Limassol, Cyprus, November 10-13, 2018, Proceedings*, volume 11237 of *Lecture Notes in Computer Science*, pages 222–240. Springer, 2018.
11. Conal Eliot and Paul Hudak. Functional reactive animation. In *Proc. of ICFP'07*, pages 163–173. ACM, 1997.
12. Peter Faymonville, Bernd Finkbeiner, Sebastian Schirmer, and Hazem Torfah. A stream-based specification language for network monitoring. In *Proc. of the 16th Int'l Conf. on Runtime Verification (RV'16)*, volume 10012 of *LNCS*, pages 152–168. Springer, 2016.
13. Peter Faymonville, Bernd Finkbeiner, Maximilian Schwenger, and Hazem Torfah. Real-time stream-based monitoring. *CoRR*, abs/1711.03829, 2017.
14. Felipe Gorostiaga and César Sánchez. Striver: Stream runtime verification for real-time event-streams. In *Proc. of the 18th Int'l Conf. on Runtime Verification (RV'18)*, volume 11237 of *LNCS*, pages 282–298. Springer, 2018.

15. Klaus Havelund and Grigore Roşu. 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)*, volume 2280 of *LNCS*, pages 342–356. Springer-Verlag, 2002.
16. Lee Pike, Alwyn Goodloe, Robin Morisset, and Sebastian Niller. Copilot: A hard real-time runtime monitor. In *Proc. of the 1st Int'l Conf. on Runtime Verification (RV'10)*, volume 6418 of *LNCS*, pages 345–359. Springer, 2010.
17. Thomas Reinbacher, Kristin Y. Rozier, and Johann Schumann. 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)*, volume 8413 of *LNCS*, pages 357–372. Springer, 2014.
18. Grigore Roşu and Klaus Havelund. Rewriting-based techniques for runtime verification. *Automated Software Engineering*, 12(2):151–197, 2005.
19. César Sánchez. Online and offline stream runtime verification of synchronous systems. In *Proc. of the 18th Int'l Conf. on Runtime Verification (RV'18)*, volume 11237 of *LNCS*, pages 138–163. Springer, 2018.
20. Koushik Sen and Grigore Roşu. Generating optimal monitors for extended regular expressions. In Oleg Sokolsky and Mahesh Viswanathan, editors, *Electronic Notes in Theoretical Computer Science*, volume 89. Elsevier, 2003.