A Compositional Framework for Quantitative Online Monitoring over Continuous-time Signals

Konstantinos Mamouras(✉), Agnishom Chattopadhyay, and Zhifu Wang

Rice University, Houston, TX 77005, USA
{mamouras, agnishom, zfwang}@rice.edu

Abstract. We investigate online monitoring algorithms over dense-time and continuous-time signals for properties written in metric temporal logic (MTL). We consider an abstract algebraic semantics based on complete lattices, which subsumes the Boolean (qualitative) semantics and the real-valued robustness (quantitative) semantics. Our semantics also extends to truth values that are partially ordered and allows the modeling of uncertainty in satisfaction. We propose a compositional approach for the construction of online monitors based on a class of infinite-state deterministic signal transducers that (1) are allowed to produce the output signal with some bounded delay relative to the input signal, and (2) do not introduce unbounded variability in the output signal. A key ingredient of our monitoring framework is a novel efficient algorithm for sliding-window aggregation over dense-time signals.

Keywords: Online monitoring · Signal temporal logic (STL) · Quantitative semantics · Cyber-physical systems (CPS) · Transducers.

1 Introduction

Metric temporal logic (MTL) [38] and signal temporal logic (STL) [41] are extensions of linear temporal logic (LTL) that have been widely used for specifying properties over the execution traces of cyber-physical systems (CPS). These traces are commonly represented as dense-time or continuous-time signals. Both MTL and STL have been extensively used as specification formalisms in the context of *monitoring*, where a system trace of finite duration is examined to determine whether it satisfies the desired temporal specification.

Our focus here is on *online* monitoring, where the system trace is presented incrementally, i.e., in a streaming fashion. This contrasts to the setting of offline monitoring, where the system trace is available in its entirety at the beginning of the computation. We choose MTL as the specification formalism, and we consider its interpretation over signals whose domain is the set of rational numbers (dense time) or the real numbers (continuous time). Our goal is to provide a unifying semantic and algorithmic framework that encompasses (1) the traditional Boolean semantics and the associated monitoring with qualitative (i.e., Boolean) verdicts, and (2) the real-valued quantitative semantics for MTL (also called *robustness* semantics) and the corresponding quantitative online monitors.

There is a wealth of proposals for quantitative semantics for MTL, such as [27,23,3]. We consider here the *spatial* robustness semantics of Fainekos and Pappas [26,27]. This uses the set of the extended real numbers, denoted by $\mathbb{R}^{\pm\infty} = \mathbb{R} \cup \{-\infty, \infty\}$, as the domain of truth values. A positive number indicates truth, a negative number indicates falsity, and zero is ambiguous. Disjunction (resp., existential quantification) is interpreted as max (resp., supremum), and conjunction (resp., universal quantification) is interpreted as min (resp., infimum). Two quantitative semantic notions are considered in [27]. The first one is the *robustness degree* $\mathsf{degree}(\varphi, \mathbf{x})$ of a signal $\mathbf{x}$ w.r.t. a formula φ, which is defined in a global way using distances between signals. This is the primary semantics, as it captures the intuitive idea of the degree of satisfaction using distances. The second notion is the *robustness estimate* $\rho(\varphi, \mathbf{x})$ of a formula φ w.r.t. a trace $\mathbf{x}$, which is defined by induction on the structure of φ. As the name suggests, the robustness estimate approximates the robustness degree; it is, in fact, an under-approximation (see Theorem 13 in page 4268 of [27]). The robustness estimate of [27] has been used in prior work on online monitoring [20,19], as it is amenable to efficient evaluation. For this reason, we will be using here the robustness estimate, not the robustness degree.

The robustness semantics of [27] can be generalized to other notions of quantitative truth values, as has already been done in [18] using an algebraic semantics based on bounded distributive lattices (where "join"/sup/$\sqcup$ generalizes max and "meet"/inf/$\sqcap$ generalizes min). The algebraic framework of [18] was developed for discrete-time signals only, since the considered class of lattices supports only finitary suprema and infima. For this reason, it is not appropriate for interpreting temporal formulas over dense-time or continuous-time signals. The semantics of [18] has been generalized further in [45] by considering semirings as truth domains, again in the context of discrete-time signals.

In this paper, we consider the class of *complete lattices*, infinitary algebraic structures of the form $(V, \bigsqcup, \bigsqcap)$, where $\bigsqcup$ is an arbitrary join/supremum operation (which models disjunction, existential quantification) and $\bigsqcap$ is an arbitrary meet/infimum operation (which models conjunction, universal quantification). The class of complete lattices contains $\mathbb{B} = \{\bot, \top\}$ (the Boolean values), and the lattice $(\mathbb{R}^{\pm\infty}, \sup, \inf)$ of extended real numbers. The lattice of intervals with join given by $\bigsqcup_i [a_i, b_i] = [\sup_i a_i, \sup_i b_i]$ and meet given by $\bigsqcap_i [a_i, b_i] = [\inf_i a_i, \inf_i b_i]$ is an especially interesting example, as it can be used to model *uncertainty* in the truth value: an element $[a, b]$ indicates that the truth value lies somewhere within this interval.

Using the algebraic quantitative semantics described in the previous paragraph, we introduce a compositional framework for online monitoring over dense-time and continuous-time signals. In order to ensure compositionality, we consider monitors that are infinite-state deterministic signal transducers. A key difference from other approaches is that our monitors do not require the input and output to be perfectly synchronized, but they can compute with some delay (or negative delay). That is, it is possible that the output signal falls behind the input signal (positive delay), or that the output signal is ahead of the input signal (negative delay). We distinguish those monitors where the delay is bounded and fixed throughput the computation. More specifically, we introduce a typing judgment $\texttt{f} : \mathsf{delay} = d$, where $d \in \mathbb{R}$, which says that the monitor $\texttt{f}$ has a fixed bounded delay d during the entire course of the computation. This concept has been explored in [47] for discrete-time signal transducers. Another key feature of our approach is that we distinguish monitors that do not introduce unbounded variability. More specifically, we use a typing judgment $\{\mathsf{ivar} = k\}\texttt{f}\{\mathsf{ovar} = \ell\}$ to indicate that if the monitor $\texttt{f}$ receives an input signal whose variability (number of value changes per time unit) is bounded above by k, then the variability of its output signal is bounded above by ℓ. The two properties of *bounded delay* and *bounded signal variability* are essential for constructing efficient monitors.

The monitoring of temporal formulas written in MTL (with unbounded past-time and bounded future-time connectives) can be reduced to a small number of computational primitives. An important fact is that we need two distributivity laws for lattices. Using the distributivity of finite meets over arbitrary joins (resp., finite joins over arbitrary meets) we show that the monitoring of the connective $\mathsf{S}_{[a,b]}$ (resp., the dual connective $\bar{\mathsf{S}}_{[a,b]}$) can be reduced to an online aggregation over a sliding window. For every MTL formula, we construct an online monitor by composing the following basic monitors: (1) $\texttt{map}(op)$, which applies the function op pointwise, (2) $\texttt{aggr}(init, op)$, which performs a running aggregation, (3) $\texttt{emit}(v, dt)$, which emits an initial signal prefix with value v and duration dt, (4) $\texttt{ignore}(dt)$, which removes an initial prefix of duration dt from the input signal, and (5) $\texttt{wnd}(dt, 1_{\otimes}, \otimes)$, which performs an associative aggregation $\otimes$ over a sliding window of duration dt. Monitors are composed using two *dataflow combinators*: (1) serial composition $\texttt{f} \gg \texttt{g}$, and (2) parallel composition $\texttt{par}(\texttt{f}, \texttt{g})$. The space efficiency of the monitors hinges on the preservation of bounded delay and bounded variability. The time efficiency relies on a novel sliding-window aggregation algorithm with $O(1)$ amortized time-per-item. The algorithm achieves this efficiency by maintaining partial aggregates of the window and reusing them as much as possible as the window slides forward.

We provide an implementation of our monitoring framework in Rust. Our experiments show that our monitors scale reasonably well and they compare favorably against the monitoring tool Reelay [52]. We chose Reelay for comparison because (1) it supports dense-time traces as input, (2) it uses a temporal semantics for specifications that is consistent with ours, and (3) it is implemented in a low-overhead compiled language (C++).

2 Algebraic Semantics with Complete Lattices

In this section, we present a quantitative semantics for MTL that uses complete lattices for the truth values. Using algebraic reasoning, we show that the temporal connectives of MTL can be rewritten into equivalent forms that suggest a simple approach for online monitoring. In particular, we show later in Proposition 4 that some distributivity laws are needed to deal with the “Since” temporal connective and its dual. Using the distributivity of finite meets over arbitrary joins (resp., finite joins over arbitrary meets) we can reduce the monitoring of $\mathsf{S}_{[a,b]}$ (resp., its dual $\bar{\mathsf{S}}_{[a,b]}$) to a sliding-window join (resp., meet). This suggests the class of (co)infinitely distributive complete lattices as an appropriate algebraic generalization of the Boolean and real-valued semantic domains.

A lattice is a partial order in which every two elements have a least upper bound and a greatest lower bound. We will use an equivalent algebraic definition. A *lattice* $(V, \sqcup, \sqcap)$ is a set V together with associative and commutative binary operations $\sqcup$ and $\sqcap$, called *join* and *meet* respectively, that satisfy the *absorption laws*, i.e, $x \sqcup (x \sqcap y) = x$ and $x \sqcap (x \sqcup y) = x$ for all $x, y \in V$. Define the relation $\leq$ as follows: $x \leq y$ iff $x \sqcup y = y$ for all $x, y \in A$. The relation $\leq$ is a partial order. It also holds that $x \leq y$ iff $x \sqcap y = x$. A lattice V is said to be *bounded* if there exists a *bottom* element $\bot \in V$ and a *top* element $\top \in V$ such that $\bot \sqcup x = x$ and $x \sqcap \top = x$ (equivalently, $\bot \leq x \leq \top$) for every $x \in V$. Let V be a bounded lattice. It is easy to check that $x \sqcup \top = \top$ and $x \sqcap \bot = \bot$ for every $x \in V$. A lattice V is said to be *distributive* if $x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z)$ and $x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z)$ for all $x, y, z \in V$.

Example 1. Consider the two-element set $\mathbb{B} = \{\top, \bot\}$ of Boolean values, where $\top$ represents truth and $\bot$ represents falsity. The set $\mathbb{B}$, together with disjunction as join and conjunction as meet, is a bounded and distributive lattice. The set $\mathbb{T} = \{\bot, ?, \top\}$ can be endowed with bounded lattice structure in a unique way so that $\bot \leq ? \leq \top$. It can be easily verified that $\mathbb{T}$ is distributive. The structure $\mathbb{T}$ is used to give a *three-valued* interpretation of formulas (? is inconclusive).

The set $\mathbb{R}$ of real numbers, together with min as meet and max as join, is a distributive lattice. However, $(\mathbb{R}, \max, \min)$ is not a bounded lattice. It is commonplace to adjoin the elements ∞ and $-\infty$ to $\mathbb{R}$ so that they serve as the top and bottom element respectively. The structure $(\mathbb{R}^{\pm\infty}, \max, \min, -\infty, \infty)$ is a bounded distributive lattice. We interpret the max-min lattice $\mathbb{R}^{\pm\infty}$ as degrees of truth, where positive means true and negative means false.

A *complete lattice* is a partially ordered set V in which all subsets have both a supremum (join) and an infimum (meet). For a subset $S \subseteq V$, the join is denoted by $\bigsqcup S$ and the meet is denoted by $\bigsqcap S$. Notice that $\bigsqcup \emptyset$ is the bottom element of V and $\bigsqcap \emptyset$ is the top element of V. We say that V is *infinitely distributive* if $x \sqcap (\bigsqcup_{i \in I} y_i) = \bigsqcup_{i \in I} (x \sqcap y_i)$ for every index set I (finite meets distribute over arbitrary joins). We say that V is *co-infinitely distributive* if $x \sqcup (\bigsqcap_{i \in I} y_i) = \bigsqcap_{i \in I} (x \sqcup y_i)$ for every index set I (finite joins distribute over arbitrary meets). We will say that V is *(co)infinitely distributive* if it is both infinitely and co-infinitely distributive. The lattices $\mathbb{B}$ and $\mathbb{R}^{\pm\infty}$ are complete and (co)infinitely distributive.

Example 2 (Uncertainty). We will consider now an example of quantitative semantics that goes beyond linear orders, and therefore it cannot be directly handled by prior monitoring frameworks based on truth values from $\mathbb{B}$ or $\mathbb{R}^{\pm\infty}$.

Suppose we want to identify a notion of quantitative truth values in situations where we interpret formulas over a signal $\mathbf{x}(t)$ that is not known with perfect accuracy, but we can put an upper and lower bound on each sample, i.e., $a \leq$

$$\begin{array}{rlrl}
\rho(\varphi \vee \psi, \mathbf{x}, t) &= \rho(\varphi, \mathbf{x}, t) \sqcup \rho(\psi, \mathbf{x}, t) & \rho(\varphi \wedge \psi, \mathbf{x}, t) &= \rho(\varphi, \mathbf{x}, t) \sqcap \rho(\psi, \mathbf{x}, t) \\
\rho(\mathsf{P}_I \varphi, \mathbf{x}, t) &= \bigsqcup_{u \in t-I,\, u \in \mathrm{dom}(\mathbf{x})} \rho(\varphi, \mathbf{x}, u) & \rho(\mathsf{H}_I \varphi, \mathbf{x}, t) &= \bigsqcap_{u \in t-I,\, u \in \mathrm{dom}(\mathbf{x})} \rho(\varphi, \mathbf{x}, u) \\
\rho(\mathsf{F}_I \varphi, \mathbf{x}, t) &= \bigsqcup_{u \in t+I,\, s \in \mathrm{dom}(\mathbf{x})} \rho(\varphi, \mathbf{x}, u) & \rho(\mathsf{G}_I \varphi, \mathbf{x}, t) &= \bigsqcap_{u \in t+I,\, u \in \mathrm{dom}(\mathbf{x})} \rho(\varphi, \mathbf{x}, u)
\end{array}$$

$$\begin{aligned}
\rho(\varphi \mathbin{\mathsf{S}_I} \psi, \mathbf{x}, t) &= \textstyle\bigsqcup_{u \in t-I,\, u \in \mathrm{dom}(\mathbf{x})} \Big(\rho(\psi, \mathbf{x}, u) \sqcap \bigsqcap_{v \in (u,t]} \rho(\varphi, \mathbf{x}, v) \Big) \\
\rho(\varphi \mathbin{\bar{\mathsf{S}}_I} \psi, \mathbf{x}, t) &= \textstyle\bigsqcap_{u \in t-I,\, u \in \mathrm{dom}(\mathbf{x})} \Big(\rho(\psi, \mathbf{x}, u) \sqcup \bigsqcup_{v \in (u,t]} \rho(\varphi, \mathbf{x}, v) \Big) \\
\rho(\varphi \mathbin{\mathsf{U}_I} \psi, \mathbf{x}, t) &= \textstyle\bigsqcup_{u \in t+I,\, u \in \mathrm{dom}(\mathbf{x})} \Big(\bigsqcap_{v \in [t,u)} \rho(\varphi, \mathbf{x}, v) \sqcap \rho(\psi, \mathbf{x}, u) \Big) \\
\rho(\varphi \mathbin{\bar{\mathsf{U}}_I} \psi, \mathbf{x}, t) &= \textstyle\bigsqcap_{u \in t+I,\, u \in \mathrm{dom}(\mathbf{x})} \Big(\bigsqcup_{v \in [t,u)} \rho(\varphi, \mathbf{x}, v) \sqcup \rho(\psi, \mathbf{x}, u) \Big)
\end{aligned}$$

Fig. 1. Quantitative semantics for MTL based on complete lattices.

$\mathbf{x}(t) \leq b$. For example, suppose that we know that $99.9 \leq \mathbf{x}(0) \leq 100.1$ and we want to evaluate the atomic predicate $p =$ "$x \geq 99$" at time 0. The truth value can be taken to be the interval [0.9, 1.1] in this case, since there is uncertainty in the distance of signal value from the threshold.

In order to model this kind of uncertainty, we consider the set $\mathcal{I}(\mathbb{R}^{\pm\infty})$ of intervals of the form $[a, b]$ with $a \leq b$ and $a, b \in \mathbb{R}^{\pm\infty}$. An interval $[a, b] \subseteq \mathbb{R}^{\pm\infty}$ can be thought of as an uncertain truth value (it can be any one of those contained in $[a, b]$). For an arbitrary family of intervals $[a_i, b_i]$ we define $\bigsqcup_i [a_i, b_i] = [\sup_i a_i, \sup_i b_i]$ and $\bigsqcap_i [a_i, b_i] = [\inf_i a_i, \inf_i b_i]$. The structure $(\mathcal{I}(\mathbb{R}^{\pm\infty}), \bigsqcup, \bigsqcap)$ is a (co)infinitely distributive complete lattice.

The lattice $\mathcal{I}(\mathbb{R}^{\pm\infty})$ is a partial order and therefore does not fit in existing monitoring frameworks that consider only linear orders (e.g., the max-min lattice $\mathbb{R}^{\pm\infty}$ of the extended reals and the associated sliding-max/min algorithms).

Let T be the ***time domain***. This can be chosen to be either $\mathbb{Q}_{\geq 0}$, the set of nonnegative rational numbers, or $\mathbb{R}_{\geq 0}$, the set of nonnegative real numbers.

An A-valued *infinite signal* is a function $\mathbf{x} : T \to A$. We write $\mathsf{ISig}(A)$ to denote the set of all A-valued infinite signals. An A-valued *finite signal* is a function $\mathbf{x} : [0, t) \to A$ or $\mathbf{x} : [0, t] \to A$, where $t \in T$. We denote the set of all A-valued finite signals by $\mathsf{FSig}(A)$. We write $\mathsf{Sig}(A) = \mathsf{FSig}(A) \cup \mathsf{ISig}(A)$. The *duration* of a finite signal $\mathbf{x} : [0, t) \to A$ or $\mathbf{x} : [0, t] \to A$ is $|\mathbf{x}| = t$. The *duration* of an infinite signal $\mathbf{x} : T \to A$ is $|\mathbf{x}| = \infty$. The empty signal is $\varepsilon : \emptyset \to A$.

We will consider formulas of Metric Temporal Logic (MTL) interpreted over signals with domain T. We consider a set D of signal values, a complete lattice V whose elements represent quantitative truth values, and *unary quantitative predicates* $p : D \to V$. We write $\mathbb{1}, \mathbb{0} : D \to V$ for the predicates given by $\mathbb{1}(d) = \top$ and $\mathbb{0}(d) = \bot$ for every $d \in D$. The set $\mathsf{MTL}(D, V)$ of ***temporal formulas*** is built from the atomic predicates $p : D \to V$ using the Boolean connectives $\vee$ and $\wedge$, the unary temporal connectives P_I, H_I, F_I, G_I, and the binary temporal connectives S_I, $\bar{\mathsf{S}}_I$, U_I, $\bar{\mathsf{U}}_I$, where I is an interval of the form $[s, t]$ or $[t, \infty)$ with $s, t \in T$. For every temporal connective $X \in \{\mathsf{P}, \mathsf{H}, \mathsf{S}, \bar{\mathsf{S}}, \mathsf{F}, \mathsf{G}, \mathsf{U}, \bar{\mathsf{U}}\}$, we write X_t as an abbreviation for $X_{[t,t]}$ and X as an abbreviation for $X_{[0,\infty)}$.

$$\begin{array}{lll}
\mathsf{P}_{[a,\infty)}\varphi \equiv \mathsf{P}_a \mathsf{P}_{[0,\infty)}\varphi & \mathsf{H}_{[a,\infty)}\varphi \equiv \mathsf{H}_a \mathsf{H}_{[0,\infty)}\varphi & \varphi \, \mathsf{S}_{[a,\infty)} \, \psi \equiv \mathsf{P}_a(\varphi \, \mathsf{S}_{[0,\infty)} \, \psi) \wedge \mathsf{H}_{[0,a)}\varphi \\
\mathsf{P}_{[a,b]}\varphi \equiv \mathsf{P}_a \mathsf{P}_{[0,b-a]}\varphi & \mathsf{H}_{[a,b]}\varphi \equiv \mathsf{H}_a \mathsf{H}_{[0,b-a]}\varphi & \varphi \, \mathsf{S}_{[a,b]} \, \psi \equiv \mathsf{P}_a(\varphi \, \mathsf{S}_{[0,b-a]} \, \psi) \wedge \mathsf{H}_{[0,a)}\varphi \\
\mathsf{F}_{[a,b]}\varphi \equiv \mathsf{F}_b \mathsf{P}_{[0,b-a]}\varphi & \mathsf{G}_{[a,b]}\varphi \equiv \mathsf{G}_b \mathsf{H}_{[0,b-a]}\varphi & \varphi \, \mathsf{U}_{[a,b]} \, \psi \equiv \mathsf{G}_{[0,a)}\varphi \wedge \mathsf{F}_a(\varphi \, \mathsf{U}_{[0,b-a]} \, \psi)
\end{array}$$

Fig. 2. Equivalences between temporal formulas.

We interpret the formulas in $\mathsf{MTL}(D,V)$ over traces from $\mathsf{Sig}(D)$ and at specific time points. For the *interpretation function* $\rho : \mathsf{MTL}(D,V) \times \mathsf{Sig}(D) \times T \to V$, the value $\rho(\varphi, \mathbf{x}, t)$ is defined when $t \in \mathrm{dom}(\mathbf{x})$. The base case is $\rho(p, \mathbf{x}, t) = p(\mathbf{x}(t))$ and the rest are shown in Fig. 1. We say that the formulas φ and ψ are *equivalent*, and we write $\varphi \equiv \psi$, if $\rho(\varphi, \mathbf{x}, t) = \rho(\psi, \mathbf{x}, t)$ for every $\mathbf{x} \in \mathsf{Sig}(D)$ and $t \in \mathrm{dom}(\mathbf{x})$. For every formula φ and every interval I, it holds that $\mathsf{P}_I\varphi \equiv \mathbb{1} \, \mathsf{S}_I \, \varphi$, $\mathsf{H}_I\varphi \equiv \mathbb{0} \, \bar{\mathsf{S}}_I \, \varphi$, $\mathsf{F}_I\varphi \equiv \mathbb{1} \, \mathsf{U}_I \, \varphi$, and $\mathsf{G}_I\varphi \equiv \mathbb{0} \, \bar{\mathsf{U}}_I \, \varphi$. So, the temporal connectives $\mathsf{P}_I, \mathsf{H}_I, \mathsf{F}_I, \mathsf{G}_I$ can be defined as abbreviations in terms of $\mathsf{S}_I, \bar{\mathsf{S}}_I, \mathsf{U}_I, \bar{\mathsf{U}}_I$.

Lemma 3. Let D be a set of data items and V be a complete lattice. The identities of Fig. 2 hold for all formulas $\varphi, \psi \in \mathsf{MTL}(D,V)$.

The identities of Fig. 2 are shown using the axioms of complete lattices. The identities below can reduce the monitoring of $\mathsf{S}_{[a,b]}/\bar{\mathsf{S}}_{[a,b]}$ to $\mathsf{P}_{[a,b]}/\mathsf{H}_{[a,b]}$.

$$\varphi \, \mathsf{S}_{[0,b]} \, \psi \equiv \mathsf{P}_{[0,b]}\psi \wedge (\varphi \, \mathsf{S} \, \psi) \tag{1}$$
$$\varphi \, \mathsf{S}_{[a,b]} \, \psi \equiv \mathsf{P}_{[a,b]}\psi \wedge (\varphi \, \mathsf{S}_{[a,\infty)} \, \psi) \tag{2}$$
$$\varphi \, \bar{\mathsf{S}}_{[0,b]} \, \psi \equiv \mathsf{H}_{[0,b]}\psi \vee (\varphi \, \bar{\mathsf{S}} \, \psi) \tag{3}$$
$$\varphi \, \bar{\mathsf{S}}_{[a,b]} \, \psi \equiv \mathsf{H}_{[a,b]}\psi \vee (\varphi \, \bar{\mathsf{S}}_{[a,\infty)} \, \psi) \tag{4}$$

Earlier occurrences of this idea are found in [25] (for the Boolean semantics) and in [22] (for the real-valued quantitative semantics), where the authors consider the future-time form $\varphi \, \mathsf{U}_{[a,b]} \, \psi \equiv \mathsf{F}_{[a,b]}\psi \wedge (\varphi \, \mathsf{U}_{[a,\infty)} \, \psi)$. Prior work on efficient monitoring [19] uses an algorithm based on it. Specifically, [19] uses a sliding-max algorithm [39], which can be applied to the lattice $\mathbb{R}^{\pm\infty}$ and other similar linear orders, but is not applicable to partial orders.

Proposition 4. Let D be a set and V be a complete lattice. Then, we have:
(1) If V is infinitely distributive, then the identities (1) and (2) hold.
(2) If V is co-infinitely distributive, then the identities (3) and (4) hold.

Proposition 4 suggests the class of (co)infinitely distributive complete lattices as an appropriate algebraic generalization of $\mathbb{R}^{\pm\infty}$ for efficient quantitative online monitoring, as the monitoring of $\mathsf{S}_{[a,b]}$ and $\bar{\mathsf{S}}_{[a,b]}$ can be reduced to sliding aggregations (for which we present an efficient algorithm later in Fig. 7).

3 Monitors

In this section, we define the class of transducers that we will use for online monitoring. We consider infinite-state deterministic signal transducers. The transducers that we use operate on representations of *piecewise constant* signals, which are alternating sequences of points and open (left-open and right-open) segments. Our transducers are allowed to have output that is not perfectly synchronized with the input, that is, the output can either fall behind or run ahead of the input. We distinguish those transducers that have a bounded and fixed delay and we use a typing judgment $\texttt{f} : \mathsf{delay} = d$ to indicate that the transducer $\texttt{f}$ has fixed delay d. We also distinguish those transducers that do not introduce unbounded variability into the output signal. More specifically, we use a typing judgment of the form $\{\mathsf{ivar} = k\}\texttt{f}\{\mathsf{ovar} = \ell\}$ to indicate that if the monitor $\texttt{f}$ receives input with variability at most k then it will produce output with variability at most ℓ.

Let A be a set. We define the set $\mathsf{Item}(A) = \{\mathsf{Pt}(a) \mid a \in A\} \cup \{\mathsf{Seg}(a, dt) \mid a \in A \text{ and } dt \in T\}$ of *data items*. A data item is either a *point* of the form $\mathsf{Pt}(a)$, where $a \in A$, or an *open segment* of the form $\mathsf{Seg}(a, dt)$, where $a \in A$ and $dt \in T$ is a time delta. When no confusion arises we write a instead of $\mathsf{Pt}(a)$, and a^{dt} instead of $\mathsf{Seg}(a, dt)$. We also consider $\mathsf{PCSig}(A) = \mathsf{Pt}(A) \cdot (\mathsf{Seg}(A, T) \cdot \mathsf{Pt}(A))^* \cdot (\{\varepsilon\} \cup \mathsf{Seg}(A, T)) \subseteq \mathsf{Item}(A)^*$, the set of alternating point-segment sequences of data items that start with a point. An element of $\mathsf{PCSig}(A)$ represents a finite piecewise constant signal. We will use the term *trace* to refer to elements of $\mathsf{Item}(A)^*$ in order to differentiate them from the signals that they represent. For a trace $\mathbf{x}$, we write $|\mathbf{x}| \in \mathbb{N}$ to denote its *length*, that is, the number of items that is contains. We write $\mathsf{dur}(\mathbf{x}) \in T$ to denote its *duration*, that is, the total amount of time that it spans. More formally, $\mathsf{dur}(\varepsilon) = 0$, $\mathsf{dur}(\mathbf{x}a) = \mathsf{dur}(\mathbf{x})$ and $\mathsf{dur}(\mathbf{x}a^{dt}) = \mathsf{dur}(\mathbf{x}) + dt$ for every $\mathbf{x} \in \mathsf{Item}(A)^*$, $a \in A$ and $dt \in T$.

We define the ***variability*** of a trace $\mathbf{x} \in \mathsf{Item}(A)^*$ as the maximum number of items that fall within any one time interval of unit duration. For example, the variability of the trace $a\,b^1\,c\,d^1$ is 3, and the variability of the trace $a\,b^{0.5}\,c\,d^{0.5}e\,f^{0.5}$ is 5. Intuitively, the variability is the maximum number of times that the value of the signal can change within any one unit interval.

Let A and B be sets. A ***monitor*** of type $\mathsf{M}(A, B)$ is a state machine $\texttt{f} = (\mathsf{St}, \mathsf{init}, \mathsf{o}, \mathsf{next}, \mathsf{out})$, where St is a set of *states*, $\mathsf{init} \in \mathsf{St}$ is the *initial state*, $\mathsf{o} \in \mathsf{Item}(B)^*$ is the *initial output*, $\mathsf{next} : \mathsf{St} \times \mathsf{Item}(A) \to \mathsf{St}$ is the *transition function*, and $\mathsf{out} : \mathsf{St} \times A \to \mathsf{Item}(B)$ is the *output function*. The monitor denotes the transduction $[\![\texttt{f}]\!] : \mathsf{Item}(A)^* \to \mathsf{Item}(B)^*$. We require additionally that a monitor respects the representation of piecewise constant signals, that is: $[\![\texttt{f}]\!](\mathbf{x}) \in \mathsf{PCSig}(B)$ for every $\mathbf{x} \in \mathsf{PCSig}(A)$. In other words, if the input stream is an alternating sequence of points and segments, then so is the output stream.

In Fig. 3 we give several examples of simple monitors that can be used as building blocks. The monitor $\texttt{map}(op)$ applies the function $op : A \to B$ elementwise. The monitor $\texttt{aggr}(b, op)$ applies a running aggregation to the input trace that is specified by the initial aggregate $b \in B$ and the aggregation function $op : B \times A \to B$ (similar to the fold combinator used in functional programming). The monitor $\texttt{emit}(v, t)$ emits a (left-closed, right-open) segment with duration $t \in T$ and value $v \in A$ upon initialization and then echoes the input trace. The monitor $\texttt{ignore}(t)$ discards the initial (left-closed, right-open) signal segment of duration $t \in T$ and proceeds to echo the rest of the signal. The monitor $\texttt{wnd}(\varDelta, 1_\otimes, \otimes)$ (described later in Fig. 6 and Fig. 7 with pseudocode)

$$\begin{aligned} &\mathtt{map}(op) : \mathsf{M}(A,B) \\ &\mathsf{St} = \mathtt{Unit} \\ &\mathsf{init} = \mathtt{u} \\ &\mathsf{o} = \varepsilon \\ &\mathsf{next}(s,a) = s \\ &\mathsf{next}(s,a^{dt}) = s \\ &\mathsf{out}(s,a) = op(a) \\ &\mathsf{out}(s,a^{dt}) = op(a)^{dt} \end{aligned} \qquad \begin{aligned} &\mathtt{aggr}(b,op) : \mathsf{M}(A,B) \\ &\mathsf{St} = B \\ &\mathsf{init} = b \\ &\mathsf{o} = \varepsilon \\ &\mathsf{next}(s,a) = op(s,a) \\ &\mathsf{next}(s,a^{dt}) = op(s,a) \\ &\mathsf{out}(s,a) = op(s,a) \\ &\mathsf{out}(s,a^{dt}) = op(s,a)^{dt} \end{aligned} \qquad \begin{aligned} &\mathtt{aggrV}(b,op) : \mathsf{M}(A,B) \\ &\mathsf{St} = B \\ &\mathsf{init} = b \\ &\mathsf{o} = \varepsilon \\ &\mathsf{next}(s,a) = op(s,a) \\ &\mathsf{next}(s,a^{dt}) = op(s,a) \\ &\mathsf{out}(s,a) = s \\ &\mathsf{out}(s,a^{dt}) = op(s,a)^{dt} \end{aligned}$$

$$\begin{aligned} &\mathtt{emit}(v,t) : \mathsf{M}(A,A) \\ &\mathsf{St} = \mathtt{Unit} \\ &\mathsf{init} = \mathtt{u} \\ &\mathsf{o} = \langle v, v^{t} \rangle \\ &\mathsf{next}(s,x) = s \\ &\mathsf{out}(s,x) = x \end{aligned} \qquad \begin{aligned} &\mathtt{ignore}(t) : \mathsf{M}(A,A) \\ &\mathsf{St} = T \\ &\mathsf{init} = 0 \\ &\mathsf{o} = \varepsilon \\ &\mathsf{next}(s,a) = s \\ &\mathsf{next}(s,a^{dt}) = s + dt \end{aligned} \qquad \begin{aligned} &\mathsf{out}(s,a) = \varepsilon, \text{ if } s < t \\ &\mathsf{out}(s,a) = a, \text{ if } t \le s \\ &\mathsf{out}(s,a^{dt}) = \varepsilon, \text{ if } s + dt \le t \\ &\mathsf{out}(s,a^{dt}) = a^{dt-(t-s)}, \text{ if } s < t < s + dt \\ &\mathsf{out}(s,a^{dt}) = a^{dt}, \text{ if } t \le s \end{aligned}$$

Fig. 3. Basic building blocks for constructing temporal quantitative monitors.

performs an aggregation, given by the associative function $\otimes : A \times A \to A$, over a sliding window of time duration Δ. The value $1_{\otimes}$ is a left and right identity for $\otimes$. We combine monitors using the operations

$$\frac{\mathtt{f} : \mathsf{M}(A,B) \qquad \mathtt{g} : \mathsf{M}(B,C)}{\mathtt{f} \gg \mathtt{g} : \mathsf{M}(A,B)} \qquad \frac{\mathtt{f} : \mathsf{M}(A,B) \qquad \mathtt{g} : \mathsf{M}(A,C)}{\mathtt{par}(\mathtt{f},\mathtt{g}) : \mathsf{M}(A, B \times C)}$$

serial composition $\gg$ and *parallel composition* $\mathtt{par}$. In the serial composition $\mathtt{f} \gg \mathtt{g}$ the output signal of $\mathtt{f}$ is propagated as input signal to $\mathtt{g}$. In the parallel composition $\mathtt{par}(\mathtt{f},\mathtt{g})$ the input signal is copied to two concurrently executing monitors $\mathtt{f}$ and $\mathtt{g}$ and their output signals are combined. Both combinators $\gg$ and $\mathtt{par}$ are given by variants of the product construction on state machines. In the case of $\mathtt{par}$ the output traces of $\mathtt{f}$ and $\mathtt{g}$ may not be synchronized (one may be ahead of the other), which requires buffering in order to properly align them. This amount of buffering is bounded when the input signal and the monitors satisfy the conditions that ensure bounded variability of their outputs. A construction similar to the one for $\mathtt{par}$ is described in [47] (in a discrete-time setting). Some of the basic monitors of Fig. 3 are similar to queries of the StreamQL language [37], which has been proposed for the processing of streaming time series.

Monitors and Delay. Let $\mathtt{f} : \mathsf{M}(A,B)$ be a monitor. We define the *delay* of the monitor $\mathtt{f}$ at $\mathbf{x} \in \mathsf{PCSig}(A)$ to be the signed time duration $\mathsf{delay}(\mathtt{f})(\mathbf{x}) = \mathsf{dur}(\mathbf{x}) - \mathsf{dur}(\mathtt{f}(\mathbf{x}))$. We say that $\mathtt{f}$ has a fixed (positive) delay d if $\mathsf{delay}(\mathtt{f})(\mathbf{x}) = \mathsf{dur}(\mathbf{x})$ when $\mathsf{dur}(\mathbf{x}) \le d$ and $\mathsf{delay}(\mathtt{f})(\mathbf{x}) = d$ when $\mathsf{dur}(\mathbf{x}) > d$. We indicate this by writing $\mathtt{f} : \mathsf{delay} = d$. Similarly, we say that $\mathtt{f}$ has a fixed (negative) delay $-d$ if $\mathsf{delay}(\mathtt{f})(\mathbf{x}) = -d$ for every $\mathbf{x}$. We indicate this by writing $\mathtt{f} : \mathsf{delay} = -d$.

$$\{\mathsf{ivar}=k\}\texttt{map}(op)\{\mathsf{ovar}=k\}$$
$$\{\mathsf{ivar}=k\}\texttt{aggr}(b,op)\{\mathsf{ovar}=k\}$$
$$\{\mathsf{ivar}=k\}\texttt{emit}(v,t)\{\mathsf{ovar}=k+1\}$$
$$\{\mathsf{ivar}=k\}\texttt{ignore}(t)\{\mathsf{ovar}=k\}$$
$$\{\mathsf{ivar}=k\}\texttt{wnd}(\mathit{\Delta},1_{\otimes},\otimes)\{\mathsf{ovar}=ck\}$$

$$\frac{\{\mathsf{ivar}=k\}\texttt{f}\{\mathsf{ovar}=\ell\} \qquad \{\mathsf{ivar}=\ell\}\texttt{g}\{\mathsf{ovar}=m\}}{\{\mathsf{ivar}=k\}\texttt{f}\gg\texttt{g}\{\mathsf{ovar}=m\}}$$

$$\frac{\{\mathsf{ivar}=k\}\texttt{f}\{\mathsf{ovar}=\ell\} \qquad \{\mathsf{ivar}=k\}\texttt{g}\{\mathsf{ovar}=m\}}{\{\mathsf{ivar}=k\}\texttt{par}(\texttt{f},\texttt{g})\{\mathsf{ovar}=\ell+m\}}$$

Fig. 4. Typing judgments for the preservation of finite variability.

All the monitors defined in Fig. 3 have a fixed (positive or negative) delay. Moreover, the combinators $\gg$ and `par` preserve this property.

$$\texttt{map}(op):\mathsf{delay}=0 \qquad \texttt{aggr}(b,op):\mathsf{delay}=0 \qquad \texttt{emit}(v,t):\mathsf{delay}=-t$$
$$\texttt{ignore}(t):\mathsf{delay}=t \qquad \texttt{wnd}(\mathit{\Delta},1_{\otimes},\otimes):\mathsf{delay}=0$$

$$\frac{\texttt{f}:\mathsf{delay}=s \qquad \texttt{g}:\mathsf{delay}=t}{\texttt{f}\gg\texttt{g}:\mathsf{delay}=s+t} \qquad \frac{\texttt{f}:\mathsf{delay}=s \qquad \texttt{g}:\mathsf{delay}=t}{\texttt{par}(\texttt{f},\texttt{g}):\mathsf{delay}=\max(s,t)}$$

This means that any monitor built from the basic ones (monitors of Fig. 3 and Fig. 7) using serial and/or parallel composition has fixed delay.

Monitors and Input/Output Variability. We are especially interested in monitors that do not introduce unbounded variability in their output. For a monitor $\texttt{f}:\mathsf{M}(A,B)$ we write the typing judgment $\{\mathsf{ivar}=k\}\texttt{f}\{\mathsf{ovar}=\ell\}$ to indicate that for every input trace $\mathbf{x}\in\mathsf{PCSig}(A)$ with variability at most k, the ouput trace $\texttt{f}(\mathbf{x})$ of the monitor has variability at most ℓ. In other words, this says that the monitor does not introduce unbounded variability.

Lemma 5. The typing judgments of Fig. 4 hold.

None of the monitors of Fig. 3 introduces unbounded variability. Moreover, the combinators $\gg$ and `par` preserve this property. The typing judgments of Fig. 4 imply that every monitor built from the basic ones (Fig. 3) using $\gg$ and `par` preserves the bounded variability of the input signal.

Bounded memory footprint. Notice that $\texttt{map}(op)$ and $\texttt{emit}(v,t)$ are stateless, which means that they need no memory. The monitor $\texttt{aggr}(b,op)$ needs one memory location to store the running aggregate. The monitor $\texttt{ignore}(t)$ needs one memory location for a clock that records the amount of time that has passed since the start of the computation. The sliding-window monitor $\texttt{wnd}(\mathit{\Delta},1_{\otimes},\otimes)$ needs $2\cdot\mathit{\Delta}\cdot\mathit{Var}$ memory locations, where Var is the variability of the input trace, for the buffers *bufL*, *bufR*, *bufL_agg* used by the sliding window algorithm (see Fig. 6 and Fig. 7 later). The combinator $\gg$ does not require additional memory. The combinator `par`, on the other hand, needs buffers that can store pending input from either input channel. Consider the monitoring $\texttt{par}(\texttt{f}_1,\texttt{f}_2)$ with

$$\texttt{f}_1:\mathsf{delay}=d_1 \qquad \{\mathsf{ivar}=k\}\texttt{f}_1\{\mathsf{ovar}=\ell_1\}$$
$$\texttt{f}_2:\mathsf{delay}=d_2 \qquad \{\mathsf{ivar}=k\}\texttt{f}_2\{\mathsf{ovar}=\ell_2\}.$$

If $d_2 \geq d_1$ (the second channel is behind the first channel), then we need a buffer of size $\lceil d_2 - d_1 \rceil \cdot \ell_1$ for buffering the first channel. If $d_1 \geq d_2$ (the first channel is behind the second channel), then we need a buffer of size $\lceil d_1 - d_2 \rceil \cdot \ell_2$ for buffering the second channel.

Notice that both bounded delay and bounded variability are crucial for putting a bound of the size of buffers used by `par` and `wnd`.

4 MTL Monitoring

In this section, we will see how temporal formulas are translated into monitors using the combinators of Sect. 3. Since we focus in this paper on online monitoring, we restrict attention to the ***future-bounded*** fragment of MTL, where the future-time temporal connectives are bounded. That is, every U_I connective is of the form $\mathsf{U}_{[a,b]}$ for $a \leq b < \infty$ (and similarly for F_I, G_I, $\bar{\mathsf{U}}_I$).

For an infinite input signal $\mathbf{x}$, the output of the monitor for the time instant t should be $\rho(\varphi, \mathbf{x}, t)$, but the monitor has to compute it by observing only a finite prefix of $\mathbf{x}$. In order for the output value of the monitor to agree with the standard temporal semantics over infinite traces we may need to delay an output item until some part of the future input is seen. For example, in the case of $\mathsf{F}_1 p$ we need to wait for one time unit: the output at time t is given after the input item at time $t+1$ is seen. In other words, the monitor for $\mathsf{F}_1 p$ has a *delay* (the output is falling behind the input) of one time unit. Symmetrically, we can allow monitors to emit output early when the correct value is known. For example, the output value for $\mathsf{P}_1 p$ is $\bot$ in the beginning and the value at time t is already known from time $t-1$. So, we also allow monitors to have negative delay (the output is running ahead of the input). The function $\mathsf{dl} : \mathsf{MTL} \to T$ gives the amount of delay required to monitor a formula. It is defined by $\mathsf{dl}(p) = 0$ and

$$\begin{aligned} \mathsf{dl}(\varphi \wedge \psi) &= \max(\mathsf{dl}(\varphi), \mathsf{dl}(\psi)) & \mathsf{dl}(\varphi \, \mathsf{S}_{[a,b]} \, \psi) &= \max(\mathsf{dl}(\varphi), \mathsf{dl}(\psi)) - a \\ \mathsf{dl}(\varphi \, \mathsf{S}_{[a,\infty)} \, \psi) &= \max(\mathsf{dl}(\varphi), \mathsf{dl}(\psi)) - a & \mathsf{dl}(\varphi \, \mathsf{U}_{[a,b]} \, \psi) &= \max(\mathsf{dl}(\varphi), \mathsf{dl}(\psi)) + b. \end{aligned}$$

$\mathtt{TL}(\varphi)$ is a signal transducer. If $\mathsf{dl}(\varphi) = 0$, the $\mathtt{TL}(\varphi)$ is transducer where the input and output signals are perfectly synchronized. If $\mathsf{dl}(\varphi) > 0$, then $\mathtt{TL}(\varphi)$ emits no output for the first $\mathsf{dl}(\varphi)$ time units and then behaves like a synchronized transducer. If $\mathsf{dl}(\varphi) < 0$, then $\mathtt{TL}(\varphi)$ emits a signal prefix of duration $\mathsf{dl}(\varphi)$ upon initialization and continues to behave like synchronized transducer.

The identities of Fig. 2 suggest that MTL monitoring can be reduced to a small set of computational primitives. The primitives of Sect. 3 are sufficient to specify the monitors, as shown in Fig. 5. We write $\pi_1 : A \times B \to A$ for the left projection and $\pi_2 : A \times B \to B$ for the right projection. Observe that the temporal connectives $X_{[0,\infty)}$ are encoded with `aggr` (running aggregation), whereas the temporal connectives $X_{(0,\infty)}$ are encoded with `aggrV` (a slight variant of running aggregation). The connectives P_a and H_a are encoded using `emit`. The connective $\mathsf{P}_{[0,a]}$ (resp., $\mathsf{H}_{[0,a]}$) is encoded using the sliding-window monitor `wnd` of Fig. 7, where the sliding aggregation is $\sqcup$ (resp., $\sqcap$). Similarly, the connectives $X_{[0,a)}$,

$$
\begin{aligned}
\mathtt{TL}(p) &= \mathtt{map}(p) \\
\mathtt{TL}(\varphi \vee \psi) &= \mathtt{par}(\mathtt{TL}(\varphi), \mathtt{TL}(\psi)) \gg \mathtt{map}(\sqcup) \\
\mathtt{TL}(\varphi \wedge \psi) &= \mathtt{par}(\mathtt{TL}(\varphi), \mathtt{TL}(\psi)) \gg \mathtt{map}(\sqcap) \\
\mathtt{TL}(\mathsf{P}_{[0,\infty)}\varphi) &= \mathtt{TL}(\varphi) \gg \mathtt{aggr}(\bot, \sqcup) \quad \text{and} \quad \mathtt{TL}(\mathsf{H}_{[0,\infty)}\varphi) = \mathtt{TL}(\varphi) \gg \mathtt{aggr}(\top, \sqcap) \\
\mathtt{TL}(\mathsf{P}_{(0,\infty)}\varphi) &= \mathtt{TL}(\varphi) \gg \mathtt{aggrV}(\bot, \sqcup) \quad \text{and} \quad \mathtt{TL}(\mathsf{H}_{(0,\infty)}\varphi) = \mathtt{TL}(\varphi) \gg \mathtt{aggrV}(\top, \sqcap) \\
\mathtt{TL}(\mathsf{P}_{a}\varphi) &= \mathtt{TL}(\varphi) \gg \mathtt{emit}(\bot, a) \quad \text{and} \quad \mathtt{TL}(\mathsf{H}_{a}\varphi) = \mathtt{TL}(\varphi) \gg \mathtt{emit}(\top, a) \\
\mathtt{TL}(\mathsf{P}_{[a,\infty)}\varphi) &= \mathtt{TL}(\mathsf{P}_a \mathsf{P}_{[0,\infty)}\varphi) \quad \text{and} \quad \mathtt{TL}(\mathsf{H}_{[a,\infty)}\varphi) = \mathtt{TL}(\mathsf{H}_a \mathsf{H}_{[0,\infty)}\varphi) \\
\mathtt{TL}(\mathsf{P}_{[0,b]}\varphi) &= \mathtt{wnd}(b, \bot, \sqcup) \quad \text{and} \quad \mathtt{TL}(\mathsf{H}_{[0,b]}\varphi) = \mathtt{wnd}(b, \top, \sqcap) \\
\mathtt{TL}(\mathsf{P}_{[a,b]}\varphi) &= \mathtt{TL}(\mathsf{P}_a \mathsf{P}_{[0,b-a]}\varphi) \quad \text{and} \quad \mathtt{TL}(\mathsf{H}_{[a,b]}\varphi) = \mathtt{TL}(\mathsf{H}_a \mathsf{H}_{[0,b-a]}\varphi) \\
\mathtt{TL}(\varphi \mathbin{\mathsf{S}} \psi) &= \mathtt{par}(\mathtt{TL}(\varphi), \mathtt{TL}(\psi)) \gg \mathtt{aggr}(\bot, \mathit{opS}) \\
&\mathit{opS} : V \times (V \times V) \to V, \text{ where } \mathit{opS}(s, \langle x, y \rangle) = (s \sqcap x) \sqcup y \\
\mathtt{TL}(\varphi \mathbin{\mathsf{S}_{[a,\infty)}} \psi) &= \mathtt{TL}(\mathsf{P}_a(\varphi \mathbin{\mathsf{S}} \psi) \wedge \mathsf{H}_{[0,a)}\varphi) \\
\mathtt{TL}(\varphi \mathbin{\mathsf{S}_{[0,b]}} \psi) &= \mathtt{TL}(\mathsf{P}_{[0,b]}\psi \wedge (\varphi \mathbin{\mathsf{S}} \psi)) \\
\mathtt{TL}(\varphi \mathbin{\mathsf{S}_{[a,b]}} \psi) &= \mathtt{TL}(\mathsf{P}_a(\varphi \mathbin{\mathsf{S}_{[0,b-a]}} \psi) \wedge \mathsf{H}_{[0,a)}\varphi) \\
\mathtt{TL}(\mathsf{F}_{a}\varphi) &= \mathtt{TL}(\varphi) \gg \mathtt{ignore}(a) \quad \text{and} \quad \mathtt{TL}(\mathsf{G}_{a}\varphi) = \mathtt{TL}(\varphi) \gg \mathtt{ignore}(a) \\
\mathtt{TL}(\mathsf{F}_{[a,b]}\varphi) &= \mathtt{TL}(\mathsf{F}_b \mathsf{P}_{[0,b-a]}\varphi) \quad \text{and} \quad \mathtt{TL}(\mathsf{G}_{[a,b]}\varphi) = \mathtt{TL}(\mathsf{G}_b \mathsf{H}_{[0,b-a]}\varphi) \\
\mathtt{TL}(\varphi \mathbin{\mathsf{U}_{[0,b]}} \psi) &= \mathtt{par}(\mathtt{TL}(\varphi), \mathtt{TL}(\psi)) \gg \mathtt{wnd}(b, 1_{\otimes_{\mathsf{U}}}, \otimes_{\mathsf{U}}) \gg \mathtt{map}(\pi_2) \gg \mathtt{ignore}(b) \\
\mathtt{TL}(\varphi \mathbin{\mathsf{U}_{[a,b]}} \psi) &= \mathtt{TL}(\mathsf{F}_a(\varphi \mathbin{\mathsf{U}_{[0,b-a]}} \psi) \wedge \mathsf{G}_{[0,a)}\varphi)
\end{aligned}
$$

Fig. 5. Online monitors for bounded-future MTL formulas.

$X_{(0,a]}$, $X_{(0,a)}$ can be encoded with a sliding aggregation that is a minor variant of the algorithm of Fig. 7 (the only difference is how the leftmost and rightmost points of the window are handled). Each connective of the form $X_{\langle a,b \rangle}$ is reduced to the connectives X_a and $X_{\langle 0,b-a \rangle}$. The "since" connectives $\mathsf{S}_{[a,\infty)}$, $\mathsf{S}_{[0,b]}$, $\mathsf{S}_{[a,b]}$ are reduced to other simpler temporal connectives. The future connectives F_a and G_a are encoded using $\mathtt{ignore}$. The connective $\mathsf{F}_{[a,b]}$ is encoded using F_b and $\mathsf{P}_{[0,b-a]}$, and similarly for $\mathsf{G}_{[a,b]}$. Finally, the "until" connective $\mathsf{U}_{[a,b]}$ is reduced to $\mathsf{U}_{[0,b-a]}$, which in turn is monitored using a sliding-window aggregation that we describe below. The connectives $\mathsf{U}_{[0,b)}$, $\mathsf{U}_{(0,b]}$, $\mathsf{U}_{(0,b)}$ are handled similarly.

Let $\mathbf{x} \in \mathsf{Sig}(D)$. If $\mathsf{dur}(\mathbf{x}) \geq t+a$ then $\rho(\varphi \mathbin{\mathsf{U}_{[0,a]}} \psi, \mathbf{x}, t) = \rho(\varphi \mathbin{\mathsf{U}} \psi, \mathbf{x}|_{[t,t+a]}, 0)$, where $\mathbf{x}|_{[t,t+a]}$ is the restriction of $\mathbf{x}$ to the interval $[t, t+a]$ (also translated so that the left endpoint is at 0). So, we can implement a monitor for the connective $\mathsf{U}_{[0,a]}$ by computing U over a window of duration exactly a time units.

Proposition 6 (Aggregation for Until). Let V be a (co)infinitely distributive complete lattice. For every piecewise constant trace $\mathbf{x} \in \mathsf{PCSig}(V \times V)$ whose underlying sequence of values is $\mathsf{val}(\mathbf{x}) = (x_0, y_0)(x_1, y_1) \ldots (x_n, y_n) \in (V \times V)^+$, the value $\rho(\pi_1 \mathbin{\mathsf{U}} \pi_2, \mathbf{x}, 0)$ can be written as an aggregate of the form $\pi_2((x_0, y_0) \otimes (x_1, y_1) \otimes \cdots \otimes (x_n, y_n))$.

Proposition 6 justifies the translation of $\mathsf{U}_{[0,b]}$ into the monitor shown in Fig. 5. Now, we will describe the data structure that performs the sliding aggregation, which is used in $\mathtt{wnd}(\Delta, 1_{\otimes}, \otimes)$. The implementation is shown in Fig. 6

```
// size = size($bufL$) + size($bufR$)
// Invariant: if size > 0 then size($bufL$) > 0.
$bufL \leftarrow []$   // empty left buffer (items)
$bufL\_agg \leftarrow []$   // empty left buffer (aggregates)
$bufR \leftarrow [\mathsf{Pt}(1_\otimes), \mathsf{Seg}(1_\otimes, \Delta)]$   // right buffer (items)
$aggR \leftarrow 1_\otimes$   // aggregate of right buffer
$agg \leftarrow 1_\otimes$   // initial overall aggregate
$dur \leftarrow \Delta$   // time duration of window
Reverse()   // restore the invariant
Function Reverse():
    // Called when size($bufL$) = 0 and size($bufR$) > 0.
    // This function restores the window invariant.
    $bufL \leftarrow bufR$   // move right buffer to left
    $bufR \leftarrow []$   // empty right buffer
    $aggR \leftarrow 1_\otimes$   // identity value
    $tmp\_agg \leftarrow 1_\otimes$   // running aggregate
    $bufL\_agg \leftarrow []$   // empty left buffer of aggregates
    for $i \leftarrow size(bufL) - 1$ to $0$ do   // calculate partial aggregates
        $tmp\_agg \leftarrow bufL[i].value \otimes tmp\_agg$   // new aggregate
        $bufL\_agg \leftarrow [tmp\_agg] \cdot bufL\_agg$   // prepend partial aggregate
    $agg \leftarrow bufL\_agg[0]$   // update overall aggregate
Function AddRight($x$):
    // item $x$ is either a point or a segment
    $bufR \leftarrow bufR \cdot [x]$   // add new item to the right
    $aggR \leftarrow aggR \otimes x.value$   // update right aggregate
    $agg \leftarrow bufL\_agg[0] \otimes aggR$   // update overall aggregate
    $dur \leftarrow dur + x.duration$   // update window duration
    // $dur$ does not change when adding a point: $\mathsf{Pt}(a).duration = 0$
Function AddLeft($x$):
    $tmp\_agg \leftarrow x.value \otimes bufL\_agg[0]$   // new partial aggregate
    $bufL \leftarrow [x] \cdot bufL$   // add new item to the left
    $bufL\_agg \leftarrow [tmp\_agg] \cdot bufL\_agg$   // prepend partial aggregate
    $agg \leftarrow bufL\_agg[0] \otimes aggR$   // update overall aggregate
    $dur \leftarrow dur + x.duration$   // update window duration
Function Remove():
    // remove oldest item from window
    $old \leftarrow bufL[0]$   // the oldest item
    $bufL \leftarrow tail(bufL)$   // remove oldest item from $bufL$
    $bufL\_agg \leftarrow tail(bufL\_agg)$   // remove corresponding aggregate
    if $size(bufL) = 0$ then
        Reverse()   // restore the invariant
    else // $size(bufL) > 0$
        $agg \leftarrow bufL\_agg[0] \otimes aggR$   // update overall aggregate
    $dur \leftarrow dur - old.duration$   // update window duration
```

Fig. 6. Auxiliary functions for the sliding-window aggregation algorithm of Fig. 7.

```
Function NextP(a):
    AddRight(Pt($a$))    // add new point to the right
    Emit(Pt($agg$))    // emit an output point
    Remove()    // remove oldest item (it should be a point)
Function NextS(a, dt):
    AddRight(Seg($a, dt$))    // add new segment to the right
    $over \leftarrow dur - \Delta$ // calculate extra duration
    while $over > 0$ do
        $old \leftarrow bufL[0]$    // the oldest item
        if $old = \mathsf{Pt}(a')$ then
            Emit(Pt($agg$))    // emit an output point
            Remove()    // remove oldest item (it should be a point)
        else if $old = \mathsf{Seg}(a', dt')$ then
            if $dt' \leq over$ then
                Emit(Seg($agg, dt'$))    // emit output segment
                Remove()    // remove old segment
            else // $dt' > over$
                Emit(Seg($agg, over$))    // emit output segment
                // modify oldest segment to reduce its duration by $over$
                $bufL[0] \leftarrow \mathsf{Seg}(a', dt' - over)$    // update
                $dur \leftarrow dur - over$    // update duration
                AddLeft(Pt($a'$))    // add a point back to the left
        $over \leftarrow dur - \Delta$ // recalculate extra duration
```

Fig. 7. Sliding aggregation over a continuous-time signal with wnd($\Delta, 1_\otimes, \otimes$).

(state, initialization of monitor, auxiliary funtions) and Fig. 7 (transition when a point or a segment is received). Suppose that the current window (of duration Δ) is $bufL \cdot bufR$, where $bufL = [x_1, x_2, \ldots, x_m]$ and $bufR = [x_{m+1}, \ldots, x_{m+n}]$. That is, the window is split into two buffers: *bufL* (left buffer) contains older elements, and *bufR* (right buffer) contains newer elements. We maintain a buffer of partial aggregates for the older elements: $bufL\_agg = [y_1, y_2, \ldots, y_m]$, where $y_i = x_i \otimes \cdots \otimes x_m$. We also maintain the aggregate $aggR = x_{m+1} \otimes \cdots \otimes x_{m+n}$ of the right buffer. So, the overall aggregate (for the entire window) is $agg = y_1 \otimes aggR$. When a new point $\mathsf{Pt}(a)$ arrives, we add it to the right buffer, we update *aggR* and *agg*, and we evict the oldest point from the window. When a new open segment $\mathsf{Seg}(a, dt)$ arrives, we add it to the right buffer, update *aggR*, *agg* and the current duration of the window, and then we evict as many old items as necessary in order to bring the window back to its desired duration Δ. Whenever the left buffer becomes empty, we convert the entire right buffer into a left buffer by performing all partial aggregations from right to left. We call this a “reversal” and it requires $O(n)$ applications of $\otimes$, where n is the size of window. If the variability of the input signal is bounded by a constant, then a reversal occurs only once every $\Theta(n)$ items. So, the algorithm needs $O(1)$ amortized time-per-item.

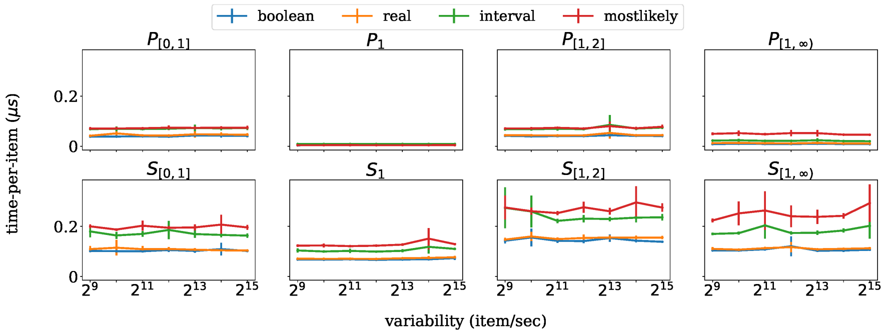

Fig. 8. Performance of our monitoring tool for various lattices of truth values.

Theorem 7. Let D be a set of signal values, V be a (co)infinitely distributive complete lattice, and $\varphi : \mathsf{MTL}(D, V)$ be a bounded-future formula. Assuming that the input signal has variability that is bounded by a constant, the monitor $\mathtt{TL}(\varphi) : \mathsf{M}(D, V)$ uses memory that is exponential in $|\varphi|$.

Proof. The algorithm needs memory that is exponential in the size of φ because of the connectives of the form $X_{[a,\infty)}$ and $X_{[a,b]}$. The monitor uses buffers of size proportional to a or $b - a$ (there is a multiplicative factor corresponding to variability). Since the constants a, b are written in binary notation, we need space that is exponential in the size.

Every temporal connective is implemented in $\mathtt{TL}(\varphi)$ as a sub-algorithm that uses constant amortized time-per-item. This hinges on the algorithm of Fig. 7, which is used for $X_{[0,b]}$ where $X \in \{\mathsf{P}, \mathsf{H}, \mathsf{S}, \mathsf{U}\}$. As discussed earlier, this sliding-window algorithm needs $O(1)$ amortized time-per-item.

5 Experiments

We have implemented the monitoring framework of Sect. 4 as a library in Rust, and we have compared our implementation with the monitoring tool Reelay [52]. We chose Reelay for the comparison because it supports dense-time traces and uses a semantics for temporal formulas that is consistent with ours. Additionally, Reelay is implemented as a C++ library, which makes the comparison with our Rust library more fair because both Rust and C++ are low-overhead compiled languages. We leave as future work the comparison with other monitoring tools (such as RTAMT [48], Breach [21], and S-TaLiRo [11]).

In our Rust implementation, we represent the values from the truth domain $\mathbb{R}^{\pm\infty}$ using 64-bit floating-point numbers. In Fig. 8, we show the performance of our tool when four different truth domains are used. We consider the lattice of Boolean values, the lattice $\mathbb{R}^{\pm\infty}$ of the extended real numbers, and the lattice $\mathcal{I}(\mathbb{R}^{\pm\infty})$ of intervals from Example 2. We also consider a variant of the lattice $\mathcal{I}(\mathbb{R}^{\pm\infty})$, labeled as "most-likely" in Fig. 8, which contains triples of the form

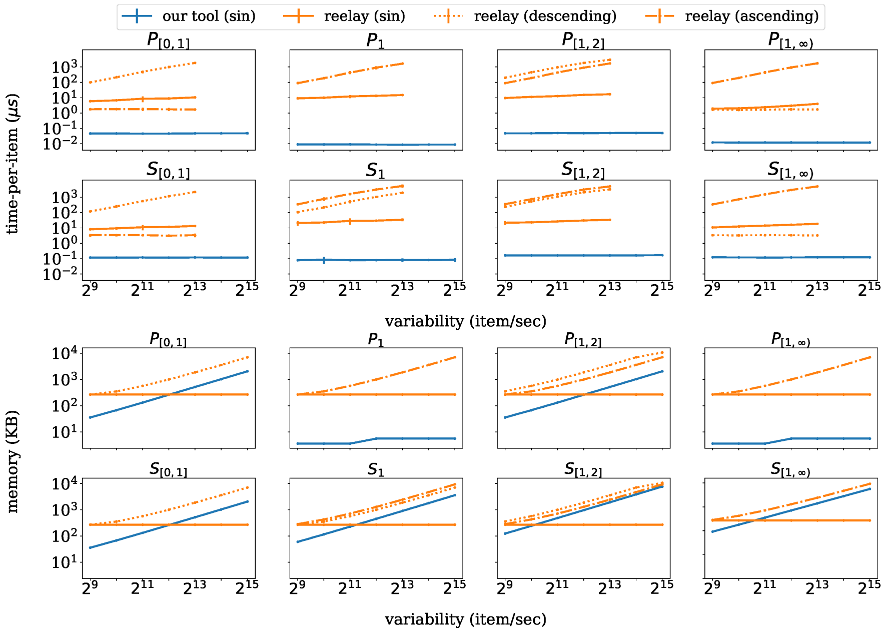

Fig. 9. Micro benchmarks w.r.t. different variability

$\langle a, m, b\rangle$ with $a \leq m \leq b$ with the interpretation that m is the most likely value and $[a, b]$ is the interval within which the value lies.

In Fig. 9, we show the time performance of the monitors with respect to the variability of the monitored signal (number of samples per time unit). We consider the formulas $X_{[0,1]}$, X_1, $X_{[1,2]}$, $X_{[1,\infty)}$ where $X \in \{\mathsf{P}, \mathsf{S}\}$. The time performance of our tool is independent of the specific signal being monitored, so we show the performance for only one kind of input signal (sinusoidal). The performance of Reelay, on the other hand, depends on the input signal. We therefore consider three different input signals: monotonically increasing, monotonically decreasing, and sinusoidal. It is desirable to have a monitoring algorithm which processes items at a fixed rate regardless of variability. We observe this behavior with our tool, and with Reelay in the case of sinusoidal input.

We have used the profiling tool Valgrind [51] to analyze the memory consumption of the monitors. In Fig. 9, we show the peak memory usage of the monitors as a function of the variability of the input signal. For Reelay, we report the performance for three different signals. The memory consumption of our monitor is independent of the values of the input signal (but is dependent on the sampling), so we have only reported the performance for the sinusoidal input signal. For our monitor, we see that the memory consumption for $\mathsf{P}_{[0,1]}, \mathsf{P}_{[1,2]}, \mathsf{S}_{[0,1]}, \mathsf{S}_1, \mathsf{S}_{[1,2]}, \mathsf{S}_{[1,\infty)}$ increases linearly with variability. This is what we expect to observe because a larger signal variability leads to a larger number of elements for a window of fixed time duration, all of which need to be stored.

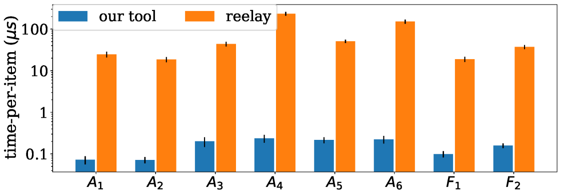

Fig. 10. Case studies from the automotive domain

For our monitor, the amount of memory allocated for P_1 and $\mathsf{P}_{[1,\infty)}$ is roughly constant. This is because the corresponding monitors do not allocate buffers. In the case of Reelay, we observe an increase in memory consumption for certain input signals. We also notice that Reelay uses at least 100 KB of memory, even for signals of low variability. We believe that this can be attributed to the complex interval-map data structures that Reelay uses from the Boost libraries [28].

We also consider two benchmarks from the automotive domain suggested in [33,34]. The system traces are generated from Simulink models using simulation. One of the benchmarks involves an automatic transmission system which has two input signals (a throttle and a brake) and three output signals: the gear sequence, the engine rotation speed (in rpm, denoted ω) and the vehicle speed (denoted v). We use a sawtooth wave of frequency 0.5 Hz for the throttle and a square wave of 0.1 Hz for the brake. We run the simulation for (a simulated time of) 300 seconds in Simulink and export the data for monitoring with our tool. The formulas that we consider are: $A_1 = \mathsf{H}_{[0,30]}(\omega < 4000)$, $A_2 = \mathsf{P}_{[0,45]}(v > 70)$, $A_3 = \mathsf{H}_{[27,57]}\mathsf{P}_{[0,13]}(v > 65)$, $A_4 = \mathsf{P}_{[60,100]}(v > 90) \to \mathsf{P}_{[70,100]}(\omega > 3000)$, $A_5 = \mathsf{H}_{[0,40]}(v < 100) \wedge \mathsf{H}_{[0,40]}(\omega < 4000)$, $A_6 = \mathsf{P}_{[0,40]}((v > 80) \to \mathsf{H}_{[0,40]}(\omega > 4000))$. The second benchmark involves a fuel control system which has a throttle and outputs the fuel flow rate (denoted λ) and the air-fuel ratio (denoted φ). We use a sawtooth wave as before for the throttle. The formulas that we consider are $F_1 = \mathsf{H}_{[0,49]}\mathsf{P}_{[0,1]}(\lambda > 0)$, $F_2 = \neg(\neg\mathsf{H}_{[0,1]}(\varphi < 1.0) \wedge \mathsf{P}_{[1,3]}(\varphi > 1.0))$. The experimental results for these two benchmarks are shown in Fig. 10.

All experiments were executed on a laptop with a 2.3 GHz Intel Core i7 10610 CPU with 16 GB of memory. Each reported value for time-per-item is the mean of 20 experiment trials. The whiskers in the plots indicate the standard deviation across all trials. Each reported value for memory consumption corresponds to one measurement, since the memory measurements are consistent across trials.

6 Related Work

Metric interval temporal logic (MITL) [5] was proposed as a restriction of MTL [38] in which non-singular intervals (i.e, intervals of the form $[a, a]$) were disallowed. Maler and Nickovic [41] proposed STL as an extension of MITL with the aim of monitoring properties of continuous signals. In that paper, STL was presented as a dense future-time logic with bounded intervals along with predicates over real-valued signals. An offline monitoring algorithm was also discussed with the assumption that the interpretation of each predicate has bounded variability (i.e, changes at most a constant number of times in each interval of fixed length). In [43], the models are restricted to signals whose time domain can be covered by left-closed right-open intervals. We consider a larger class of signals by representing our time domain in the form of a sequence of alternating points and open segments.

Fainekos and Pappas [27] defined a robustness semantics which quantifies the degree to which a given signal satisfies a specification. This semantics was generalized in [18] by using bounded distributive lattices for truth domains. The present paper employs a similar semantics, where complete lattices are used to accommodate dense and continuous time. The papers [35,45] consider two different algebraic semantics of temporal formulas using semirings, both of which only apply to the discrete-time setting. In [53], a dense-time online monitoring framework is presented with quantitative semiring-based semantics using weighted automata. In the frameworks given by [35] and [53], the semantics is based on shortest distances (i.e., standard semantics of weighted automata) as opposed to an inductive definition on formula structure like ours.

In [13,16] some generalizations of the Boolean semantics to finite lattices are considered in the context of runtime verification. It is worth noting that the standard algorithms used for Boolean semantics can be easily adopted to a semantics using finite lattices with a small number of elements. However, this is not the case with the infinite lattices, such as $(\mathbb{R}^{\pm\infty}, \sup, \inf)$, that we consider. The problem of *parametric identification* for STL [12] (where the syntax of STL is extended with symbolic parameters) is related to the problem of monitoring when the truth values are sets of possible parameter assignments/valuations. In this setting, the truth values form a complete lattice with union as join and intersection as meet. This suggests a relationship to our algebraic framework.

Timed automata [4] are a formalism for specifying real-time properties of systems. A discussion of the past and future fragments of MITL and their connection to timed automata can be found in [43]. The notion of a temporal tester is used in [42,31]. Temporal testers [49] are transducers which output the truth value of a temporal formula at each position. In these papers, the authors provide a compositional framework to construct testers from MITL formulas. We also consider a compositional transducer framework here, but our model of computation is more general and can support online quantitative monitoring that goes beyond temporal logic (e.g., general running and sliding-window aggregations with `aggr` and `wnd` respectively).

The line of work on SRV (Stream Runtime Verification) [50,32] is also relevant, because SRV languages can be used to encode quantitative monitoring algorithms. The stream-based specification language RTLola [30] provides a construct for aggregation over a sliding window. In contrast to our sliding windows, RTLola relies on the periodic partitioning and pre-aggregation along the time axis (an idea described earlier in [40]) in order to reduce the space requirements. So, the output signal can be viewed as a fixed-rate approximation of the desired sliding aggregation. This technique is therefore not suitable for implementing the temporal connectives (e.g., $\mathsf{P}_{[0,b]}$ and $\mathsf{H}_{[0,b]}$) of the logical formalism that we consider here. The StreamLAB tool [29], which is used for monitoring cyber-physical systems, uses RTLola as its specification language. Closely related to the aforementioned works on SRV are other formalisms and domain-specific languages for data stream processing. Quantitative regular expressions (QREs) [46] (see also [7] and [10]) have been used to express algorithms for medical monitoring [1,2]. The relationship between QREs and automata-theoretic models with registers is investigated in [8,9,6]. The synchronous languages [17,15,14] are based on Kahn's dataflow [36] and have been used for embedded controller design.

Originally, discussions involving offline monitoring, such as in [22] have only consisted of future-time connectives. This choice is made because the temporal formulas are interpreted at the beginning of the trace. In the context of online monitoring, however, different approaches have been taken towards future temporal connectives. While [20] assumes the availability of a predictor to interpret future connectives, [24] considers robustness intervals: the tightest intervals which cover the robustness of all possible extensions of the available trace prefix. The tool Reelay [52] uses only past-time temporal connectives. The tool RTAMT [48] *pastifies* a future-time formula by converting it into a past-time formula. The inductive definition of pastification is detailed in [44].

It was observed in [22] that the key ingredient for efficiently monitoring STL is an online algorithm for calculating the maximum/minimum over a sliding window. The commonly used algorithm [39] maintains a so-called monotonic wedge of values. In contrast, we use a more general algorithm, which applies to any associative aggregation (not only max/min) and does not require the domain of values to be totally ordered.

7 Conclusion

We have presented a new efficient algorithm for the online monitoring of MTL properties over dense-time and continuous-time signals. We have used an abstract algebraic semantics based on complete lattices satisfying certain infinitary distributivity laws, which can be instantiated to the widely-used Boolean (qualitative) and robustness (quantitative) semantics, as well as to other partially ordered truth values. Our monitoring framework is compositional in the sense that we construct monitors from formulas using a set of combinators on monitors. A key feature that enables compositionality and efficiency in our framework is the use of monitors that are deterministic signal transducers with associated typing judgments for ensuring that: (1) each monitor has a bounded and fixed delay, and (2) each monitor produces output of bounded variability given input of bounded variability. We have provided an implementation of our algebraic monitoring framework, and we have shown experimentally that our monitors scale reasonably well and are competitive against the tool Reelay [52].

Acknowledgments. This research was supported in part by US National Science Foundation award 2008096.

References

1. Abbas, H., Alur, R., Mamouras, K., Mangharam, R., Rodionova, A.: Real-time decision policies with predictable performance. Proceedings of the IEEE, Special Issue on Design Automation for Cyber-Physical Systems **106**(9), 1593–1615 (2018). https://doi.org/10.1109/JPROC.2018.2853608
2. Abbas, H., Rodionova, A., Mamouras, K., Bartocci, E., Smolka, S.A., Grosu, R.: Quantitative regular expressions for arrhythmia detection. IEEE/ACM Transactions on Computational Biology and Bioinformatics **16**(5), 1586–1597 (2019). https://doi.org/10.1109/TCBB.2018.2885274
3. Akazaki, T., Hasuo, I.: Time robustness in MTL and expressivity in hybrid system falsification. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9207, pp. 356–374. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21668-3_21
4. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science **126**(2), 183–235 (1994). https://doi.org/10.1016/0304-3975(94)90010-8
5. Alur, R., Feder, T., Henzinger, T.A.: The benefits of relaxing punctuality. Journal of the ACM **43**(1), 116–146 (1996). https://doi.org/10.1145/227595.227602
6. Alur, R., Fisman, D., Mamouras, K., Raghothaman, M., Stanford, C.: Streamable regular transductions. Theoretical Computer Science **807**, 15–41 (2020). https://doi.org/10.1016/j.tcs.2019.11.018
7. Alur, R., Mamouras, K.: An introduction to the StreamQRE language. Dependable Software Systems Engineering **50**, 1–24 (2017). https://doi.org/10.3233/978-1-61499-810-5-1
8. Alur, R., Mamouras, K., Stanford, C.: Automata-based stream processing. In: ICALP 2017. Leibniz International Proceedings in Informatics (LIPIcs), vol. 80, pp. 112:1–112:15. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2017). https://doi.org/10.4230/LIPIcs.ICALP.2017.112
9. Alur, R., Mamouras, K., Stanford, C.: Modular quantitative monitoring. Proceedings of the ACM on Programming Languages **3**(POPL), 50:1–50:31 (2019). https://doi.org/10.1145/3290363
10. Alur, R., Mamouras, K., Ulus, D.: Derivatives of quantitative regular expressions. In: Aceto, L., Bacci, G., Bacci, G., Ingólfsdóttir, A., Legay, A., Mardare, R. (eds.) Models, Algorithms, Logics and Tools: Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday, LNCS, vol. 10460, pp. 75–95. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63121-9_4
11. Annapureddy, Y., Liu, C., Fainekos, G., Sankaranarayanan, S.: S-TaLiRo: A tool for temporal logic falsification for hybrid systems. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 254–257. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19835-9_21
12. Bakhirkin, A., Ferrère, T., Maler, O.: Efficient parametric identification for STL. In: HSCC 2018. pp. 177–186. ACM, New York, NY, USA (2018). https://doi.org/10.1145/3178126.3178132
13. Bauer, A., Leucker, M., Schallhart, C.: Comparing LTL semantics for runtime verification. Journal of Logic and Computation **20**(3), 651–674 (2010). https://doi.org/10.1093/logcom/exn075
14. Benveniste, A., Le Guernic, P., Jacquemot, C.: Synchronous programming with events and relations: The SIGNAL language and its semantics. Science of Computer Programming **16**(2), 103–149 (1991). https://doi.org/10.1016/0167-6423(91)90001-E

15. Berry, G., Gonthier, G.: The Esterel synchronous programming language: Design, semantics, implementation. Science of Computer Programming **19**(2), 87–152 (1992). https://doi.org/10.1016/0167-6423(92)90005-V
16. Bonakdarpour, B., Fraigniaud, P., Rajsbaum, S., Rosenblueth, D.A., Travers, C.: Decentralized asynchronous crash-resilient runtime verification. In: Desharnais, J., Jagadeesan, R. (eds.) CONCUR 2016. Leibniz International Proceedings in Informatics (LIPIcs), vol. 59, pp. 16:1–16:15. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2016). https://doi.org/10.4230/LIPIcs.CONCUR.2016.16
17. Caspi, P., Pilaud, D., Halbwachs, N., Plaice, J.A.: LUSTRE: A declarative language for real-time programming. In: POPL 1987. pp. 178–188. ACM, New York, NY, USA (1987). https://doi.org/10.1145/41625.41641
18. Chattopadhyay, A., Mamouras, K.: A verified online monitor for metric temporal logic with quantitative semantics. In: Deshmukh, J., Ničković, D. (eds.) RV 2020. LNCS, vol. 12399, pp. 383–403. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-60508-7_21
19. Deshmukh, J.V., Donzé, A., Ghosh, S., Jin, X., Juniwal, G., Seshia, S.A.: Robust online monitoring of signal temporal logic. Formal Methods in System Design **51**(1), 5–30 (2017). https://doi.org/10.1007/s10703-017-0286-7
20. Dokhanchi, A., Hoxha, B., Fainekos, G.: On-line monitoring for temporal logic robustness. In: Bonakdarpour, B., Smolka, S.A. (eds.) RV 2014. LNCS, vol. 8734, pp. 231–246. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11164-3_19
21. Donzé, A.: Breach, a toolbox for verification and parameter synthesis of hybrid systems. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 167–170. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14295-6_17
22. Donzé, A., Ferrère, T., Maler, O.: Efficient robust monitoring for STL. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 264–279. Springer, Heidelberg (2013)
23. Donzé, A., Maler, O.: Robust satisfaction of temporal logic over real-valued signals. In: Chatterjee, K., Henzinger, T.A. (eds.) FORMATS 2010. LNCS, vol. 6246, pp. 92–106. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15297-9_9
24. Dreossi, T., Dang, T., Donzé, A., Kapinski, J., Jin, X., Deshmukh, J.V.: Efficient guiding strategies for testing of temporal properties of hybrid systems. In: Havelund, K., Holzmann, G., Joshi, R. (eds.) NFM 2015. LNCS, vol. 9058, pp. 127–142. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-17524-9_10
25. D'Souza, D., Tabareau, N.: On timed automata with input-determined guards. In: Lakhnech, Y., Yovine, S. (eds.) FTRTFT 2004, FORMATS 2004. LNCS, vol. 3253, pp. 68–83. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30206-3_7
26. Fainekos, G.E., Pappas, G.J.: Robustness of temporal logic specifications. In: Havelund, K., Núñez, M., Roşu, G., Wolff, B. (eds.) FATES 2006, RV 2006. LNCS, vol. 4262, pp. 178–192. Springer, Heidelberg (2006). https://doi.org/10.1007/11940197_12
27. Fainekos, G.E., Pappas, G.J.: Robustness of temporal logic specifications for continuous-time signals. Theoretical Computer Science **410**(42), 4262–4291 (2009). https://doi.org/10.1016/j.tcs.2009.06.021
28. Faulhaber, J.: Boost library documentation: Interval container library. https://www.boost.org/doc/libs/1_76_0/libs/icl/doc/html/index.html (2021), [Online; accessed August 20, 2021]

29. Faymonville, P., Finkbeiner, B., Schledjewski, M., Schwenger, M., Stenger, M., Tentrup, L., Torfah, H.: StreamLAB: Stream-based monitoring of cyber-physical systems. In: Dillig, I., Tasiran, S. (eds.) CAV 2019. LNCS, vol. 11561, pp. 421–431. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4_24
30. Faymonville, P., Finkbeiner, B., Schwenger, M., Torfah, H.: Real-time stream-based monitoring. CoRR **abs/1711.03829** (2017), http://arxiv.org/abs/1711.03829
31. Ferrère, T., Maler, O., Ničković, D., Pnueli, A.: From real-time logic to timed automata. Journal of the ACM **66**(3), 19:1–19:31 (2019). https://doi.org/10.1145/3286976
32. Gorostiaga, F., Sánchez, C.: Striver: Stream runtime verification for real-time event-streams. In: Colombo, C., Leucker, M. (eds.) RV 2018. LNCS, vol. 11237, pp. 282–298. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03769-7_16
33. Hoxha, B., Abbas, H., Fainekos, G.E.: Benchmarks for temporal logic requirements for automotive systems. In: Frehse, G., Althoff, M. (eds.) ARCH@CPSWeek 2014, 2015. EPiC Series in Computing, vol. 34, pp. 25–30. EasyChair (2014). https://doi.org/10.29007/xwrs
34. Hoxha, B., Bach, H., Abbas, H., Dokhanchi, A., Kobayashi, Y., Fainekos, G.: Towards formal specification visualization for testing and monitoring of cyber-physical systems. In: International Workshop on Design and Implementation of Formal Tools and Systems. DIFTS 2014 (2014)
35. Jakšić, S., Bartocci, E., Grosu, R., Ničković, D.: An algebraic framework for runtime verification. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **37**(11), 2233–2243 (2018). https://doi.org/10.1109/TCAD.2018.2858460
36. Kahn, G.: The semantics of a simple language for parallel programming. Information Processing **74**, 471–475 (1974)
37. Kong, L., Mamouras, K.: StreamQL: A query language for processing streaming time series. Proceedings of the ACM on Programming Languages **4**(OOPSLA), 183:1–183:32 (2020). https://doi.org/10.1145/3428251
38. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-Time Systems **2**(4), 255–299 (1990). https://doi.org/10.1007/BF01995674
39. Lemire, D.: Streaming maximum-minimum filter using no more than three comparisons per element. CoRR **abs/cs/0610046** (2006), http://arxiv.org/abs/cs/0610046
40. Li, J., Maier, D., Tufte, K., Papadimos, V., Tucker, P.A.: No pane, no gain: Efficient evaluation of sliding-window aggregates over data streams. SIGMOD Record **34**(1), 39–44 (2005). https://doi.org/10.1145/1058150.1058158
41. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Lakhnech, Y., Yovine, S. (eds.) FTRTFT 2004, FORMATS 2004. LNCS, vol. 3253, pp. 152–166. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30206-3_12
42. Maler, O., Nickovic, D., Pnueli, A.: Real time temporal logic: Past, present, future. In: Pettersson, P., Yi, W. (eds.) FORMATS 2005. LNCS, vol. 3829, pp. 2–16. Springer, Heidelberg (2005). https://doi.org/10.1007/11603009_2
43. Maler, O., Nickovic, D., Pnueli, A.: From MITL to timed automata. In: Asarin, E., Bouyer, P. (eds.) FORMATS 2006. LNCS, vol. 4202, pp. 274–289. Springer, Heidelberg (2006). https://doi.org/10.1007/11867340_20
44. Maler, O., Ničković, D., Pnueli, A.: On synthesizing controllers from bounded-response properties. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 95–107. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73368-3_12

45. Mamouras, K., Chattopadhyay, A., Wang, Z.: Algebraic quantitative semantics for efficient online temporal monitoring. In: Groote, J.F., Larsen, K.G. (eds.) TACAS 2021. LNCS, vol. 12651, pp. 330–348. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72016-2_18
46. Mamouras, K., Raghothaman, M., Alur, R., Ives, Z.G., Khanna, S.: StreamQRE: Modular specification and efficient evaluation of quantitative queries over streaming data. In: PLDI 2017. pp. 693–708. ACM, New York, NY, USA (2017). https://doi.org/10.1145/3062341.3062369
47. Mamouras, K., Wang, Z.: Online signal monitoring with bounded lag. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (2020). https://doi.org/10.1109/TCAD.2020.3013053
48. Ničković, D., Yamaguchi, T.: RTAMT: Online robustness monitors from STL. In: Hung, D.V., Sokolsky, O. (eds.) ATVA 2020. LNCS, vol. 12302, pp. 564–571. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59152-6_34
49. Pnueli, A., Zaks, A.: On the Merits of Temporal Testers, LNCS, vol. 5000, pp. 172–195. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69850-0_11
50. Sánchez, C.: Online and offline stream runtime verification of synchronous systems. In: Colombo, C., Leucker, M. (eds.) RV 2018. LNCS, vol. 11237, pp. 138–163. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03769-7_9
51. The Valgrind Developers: Valgrind: An instrumentation framework for building dynamic analysis tools. https://valgrind.org/ (2021), [Online; accessed August 20, 2021]
52. Ulus, D.: The Reelay monitoring tool. https://doganulus.github.io/reelay/ (2020), [Online; accessed August 20, 2020]
53. Waga, M.: Online quantitative timed pattern matching with semiring-valued weighted automata. In: André, É., Stoelinga, M. (eds.) FORMATS 2019. LNCS, vol. 11750, pp. 3–22. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-29662-9_1