Online Signal Monitoring with Bounded Lag

Konstantinos Mamouras and Zhifu Wang

***Abstract*—An essential approach for guaranteeing the safety of a cyber-physical system is to monitor its execution in real time. The execution trace of such a system typically consists of one or more signals, and a key computational task for safety monitoring is the online processing of these signals in order to identify events that need to be acted upon in a timely manner. There are several existing proposals for the specification of signal monitors: temporal logics, reactive languages, and dataflow formalisms. A shared feature of most of these proposals is that they describe online signal transformations that are causal. The causality requirement enables a real-time implementation, where the input and output signals are perfectly synchronized.**

We propose a new specification formalism for signal monitors that relaxes the causality restriction and allows the output to depend on a bounded amount of future input. It follows that an online implementation of such a monitor must have a certain amount of lag in the computation. We introduce a formal framework for signal transformations that allow bounded lag (the output has fallen behind the input) and bounded lead (the output is running ahead of the input), and we propose a type discipline for classifying these transformations according to their lead/lag. We show that this typed framework provides a modular approach for succinctly specifying: (1) monitors for temporal properties that involve both past and bounded-future connectives, and (2) complex signal processing computations, such as those arising in the monitoring of physiological signals in medical devices. We have implemented the proposed specification formalism and we have compared it against state-of-the-art tools for the online monitoring of temporal properties: MonPoly, StreamLAB, Aerial, and Reelay.

***Index Terms*—Cyber-physical systems, online monitoring, runtime verification, Metric Temporal Logic (MTL), Signal Temporal Logic (STL), quantitative properties, data streams, transducers, automata.**

I. Introduction

As a motivating application domain for this work, consider implantable medical devices, such as cardiac pacemakers and implantable cardioverter defibrillators (ICDs). A pacemaker is meant to detect an abnormal heart rhythm by continuously monitoring the electrical heart signal of the patient and deliver therapy upon detection in order to restore a normal rhythm. The response of the pacemaker has to be in real time in order to fulfill its purpose of patient treatment.

Cyber-physical systems, such as the one described previously, require algorithms for the *online monitoring* of signals. These algorithms must run in real time using a small amount of resources such as memory space and time to process each sample. One approach for developing such online algorithms is to use a low-level programming language. Another common approach is to use a logical specification formalism, such as Metric Temporal Logic (MTL) [1] or Signal Temporal Logic (STL) [2], which can then be compiled to an executable online monitor. There is a large body of work on monitoring various classes of properties: past-time temporal [3], past-time first-order temporal [4], metric first-order temporal (past-time and bounded future-time) [5], and timed regular [6]. Quantitative variants of Boolean monitoring have also been considered: using a robustness semantics for temporal logic [7], [8], [9], [10], [11], [12], and using domain-specific languages for stream processing [13], [14].

Closely related to online monitoring are the languages for programming reactive systems: (1) the synchronous languages [15], [16], [17], [18] have been used successfully for specifying embedded controllers, (2) the formalisms for synchronous dataflow [19], [20] have been used for specifying signal processing systems, and (3) the languages for reactive programming [21], [22], [23] are useful for developing event-driven applications.

Our goals here overlap with those of the aforementioned works: we want to provide language support for conveniently specifying complex but efficient monitors for signals that arise in cyber-physical systems. We are interested in properties that are not only Boolean, but also encompass the calculation of statistical measures (e.g., averages) and signal processing computations (e.g., low-pass and high-pass filters).

A key difference from most prior work is that we focus on relaxing the notion of *causal* online signal computations, by allowing the output at a timepoint t to depend on a bounded amount of future input, e.g., over the interval $[t, t+u]$. This implies that the computation of the output happens with some bounded lag: the monitor has to wait until all the relevant input is seen before it can emit its output. This relaxation of causality is useful for several practical computations. Consider, for example, the detection of the peaks in the ECG signal. In order to correctly identify that a peak is present at time t, the monitor has to first receive some of the input ahead of t in order to witness the falling slope after the peak.

We introduce a new formal framework for specifying online signal transformations that are not necessarily causal, but are instead only required to have a bounded lag (the amount by which the output has fallen behind the input) and a bounded lead (the amount by which the output is running ahead of the input). This class of transformations lies strictly between the causal transformations (where the output is completely synchronized with the input, e.g., as is typical in synchronous languages) and the asynchronous signal transformations of Kahn [24] (where there is no requirement regarding if and when output is emitted). We call these transformations ***lead/lag-bounded transductions***. For brevity,

The authors are with the Department of Computer Science, Rice University, Houston, TX, 77005 (e-mail: mamouras@rice.edu; zfwang@rice.edu).

Manuscript received April 17, 2020; revised June 17, 2020; accepted July 6, 2020. This article was presented in the International Conference on Embedded Software 2020 and appears as part of the ESWEEK-TCAD special issue.

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

we also call them ℓ-bounded transductions. We introduce a corresponding model of online computation, which we call *lead/lag-bounded transducer* (or ℓ-bounded transducer). Both transductions (semantic objects) and transducers (machine models) are classified using ***types*** of the form $(A, B, [m, n])$, where A is the type of the input samples, B is the type of the output samples, and $[m, n]$ is an interval of the integers. The lead/lag of the transformation must fall within the interval $[m, n]$. In other words, these lead/lag types indicate the allowable *skew* between the output and the input. In Section VII we discuss the relationship between lead/lag types and other notions of types for synchronous computations, such as the *clocks* of Lustre [15], [25] and its extensions (e.g., Lucid Synchrone [26], [27] and Lucy-n [28]), which in turn are also related to types that characterize the input/output *rate* of a transformation [19], [20].

The ℓ-bounded transducers and tranductions can be composed using several ***combinators***: (1) the dataflow combinators of serial composition, parallel composition, and feedback composition; (2) tranformations such as map, running aggregation (fold), and sliding windows; (3) the *ignore* combinator that disregards a prefix of the signal; and (4) the *emit* combinator that generates output without consuming input. These combinators give rise to a domain-specific language for describing ℓ-bounded transformations. Every ℓ-bounded transformation that is specified using only these combinators satisfies a crucial ***efficiency guarantee***: it can be implemented with an online monitor that uses constant memory space, i.e., space that is independent of the length of the signal seen so far.

In order to illustrate the usability of our typed framework of ℓ-bounded transformations, we use it to prototype a constant-space monitoring algorithm for MTL with past-time and bounded future-time temporal connectives. The monitor for a formula of this logic is encoded as an expression that only involves the combinators of the previous paragraph. This gives rise to a ***compositional monitoring*** algorithm: the monitor for a composite formula (e.g., $\varphi \mathsf{U}_{[a,b]} \psi$) is given by a construction on the monitors for the immediate subformulas (e.g., φ and ψ). This is different from existing non-compositional approaches (e.g., the one described in [9]), where a global analysis of the formula takes place initially in order to pre-allocate a table whose width is at least as large as the amount of lookahead (into the future) that the monitor requires.

As a case study, we use our framework to prototype a complex quantitative monitor for ***ECG peak detection***. Peak detection algorithms [29], [30], [31], [32] are a crucial part of software that runs on implantable medical devices such as pacemakers and defibrillators [33], [34]. This monitor detects the peaks in the ECG signal (which correspond to heart beats) using a complex online algorithm that (1) removes high frequencies with a low-pass filter, (2) calculates the slope of the signal using derivatives, (3) computes the length of the signal curve over a window centered around each timepoint, and (4) identifies the peaks with a decision rule that uses the calculated quantities.

We provide an ***implementation*** of ℓ-bounded transductions and their combinators using the Rust programming language. In order to evaluate the efficiency of the implementation we compare our prototype MTL monitor against the state-of-the-art tools MonPoly [35], StreamLAB [36], Aerial [37], and Reelay [38]. We chose these tools for comparison because they provide support for MTL monitoring. The experimental results show that our monitor performs well on several microbenchmarks and on the recently proposed Timescales benchmark [39].

TABLE I
TRANSDUCTIONS OF STREAMS AND SIGNALS.

Category & Description
$\mathsf{SF}(A, B)$: signal functions of type $A^\omega \to B^\omega$
$\mathsf{CSF}(A, B) \subseteq \mathsf{SF}(A, B)$: causal signal functions
$\mathsf{IMT}(A, B)$: (incremental) Mealy transductions, i.e., functions $A^+ \to B$
$\mathsf{MT}(A, B)$: (cumulative) Mealy transductions, i.e., monotone and length-preserving functions $A^+ \to B^+$
$\mathsf{IST}(A, B)$: (incremental) stream transductions, i.e., functions $A^* \to B^*$
$\mathsf{ST}(A, B)$: (cumulative) stream transductions, i.e., monotone functions of type $A^* \to B^*$
$\mathsf{KT}(A, B)$: Kahn transductions, i.e., ω-continuous functions of type $A^\infty \to B^\infty$, where $A^\infty = A^* \cup A^\omega$
$\mathsf{ST}_\Diamond(A, B) \subseteq \mathsf{ST}(A, B)$: stream transductions that satisfy the progress property
$\mathsf{KT}_\Diamond(A, B) \subseteq \mathsf{KT}(A, B)$: Kahn transductions that satisfy the progress property
$\mathsf{ST}_\mathsf{b}(A, B) \subseteq \mathsf{ST}_\Diamond(A, B)$: stream transductions with bounded lead/lag, $\mathsf{ST}_\mathsf{b}(A, B, [m, n]) \subseteq \mathsf{ST}_\mathsf{b}(A, B)$ contains those with lead in $[m, n]$

II. TRANSDUCTIONS OF SIGNALS

This section is an exploration of various denotational semantic models for online signal transformations. The classes of models that we consider are summarized in Table I. Most of the definitions in the table are provided to motivate and give context for our proposal, but it is sufficient for the reader to only remember ST_b (ℓ-bounded stream transductions).

We identify the class of lead/lag-bounded (ℓ-bounded) transductions as an extension of causal (synchronous) signal transformations that allows a bounded amount of lookahead into the future. The class of ℓ-bounded transductions is contained in the class of Kahn transductions [24], which allows complete asynchrony between input and output.

Let A be a set. A *signal* over A is a function $A^\omega = \omega \to A$, where $\omega = \{0, 1, 2, \ldots\}$ is the set of natural numbers. In other words, a signal over A is an infinite sequence over A. We will be using the common notation $i < \omega$ to mean $i \in \omega$. A ***signal function*** is an element of $\mathsf{SF}(A, B) = A^\omega \to B^\omega$. For $i < \omega$ we define the equivalence relation $\sim_i$ on A^ω as follows: $u \sim_i v$ iff $u(j) = v(j)$ for all $j \leq i$. That is, $u \sim_i v$ iff u, v agree on all timepoints up to (and including) i. A signal function $f : \mathsf{SF}(A, B)$ is *causal* if $u \sim_i v$ implies $f(u)(i) = f(v)(i)$ for every $i < \omega$ and all $u, v \in A^\omega$. Intuitively, f is causal if at every timepoint i the output of f depends only on the input values at timepoints $j \leq i$. We write $\mathsf{CSF}(A, B)$ for the set of signal functions of $\mathsf{SF}(A, B)$ that are causal.

The operation of *serial composition*, denoted $\gg$, is meaningful in the context of various kinds of signal transformations. For signal functions, we take $\gg$ to be composition of functions: $(f \gg g)(u) = g(f(u))$ for every $u \in A^\omega$, $f : \mathsf{SF}(A, B)$

and $g : \mathsf{SF}(B, C)$. We write $\mathsf{id}_A^{\mathsf{SF}} : \mathsf{SF}(A, A)$ for the identity function. SF is a typed monoid, i.e., a category [40].

In a causal signal function $f : A^\omega \to B^\omega$ the output at some timepoint i depends only on the finite prefix of the input signal from 0 up to i. So, we can describe the same transformation with a function $H(f) : A^+ \to B$, where A^+ is the set of all nonempty finite sequences over A. The idea is that for an input signal $u \in A^\omega$ the output $f(u)(i)$ is equal to $H(f)(u^{\leq i})$, where $u^{\leq i}$ is the prefix of u from the beginning up to timepoint i. We say that an element of $\mathsf{IMT}(A, B) = A^+ \to B$ is a ***Mealy transduction*** (in incremental form). For $f : A^+ \to B$, we define $\mathit{lift}(f)(a_0 a_1 \dots a_n) = f(a_0) f(a_0 a_1) \cdots f(a_0 a_1 \dots a_n)$ for every $a_0 a_1 \dots a_n \in A^+$. Serial composition is then given by $(f \gg g)(u) = g(\mathit{lift}(f)(u))$ for every $u \in A^+$, $f : \mathsf{IMT}(A, B)$ and $g : \mathsf{IMT}(B, C)$. The identity transduction $\mathsf{id}_A^{\mathsf{IMT}} : \mathsf{IMT}(A, A)$ sends $a_1 \dots a_n \in A^+$ to a_n. It can be easily checked that CSF and IMT are isomorphic. So, causal signal transformations can be described by Mealy transductions, which involve only the finite prefixes of the signals.

A transduction in incremental form gives the output increment when the last item of an input prefix is consumed. For a function $f : \mathsf{IMT}(A, B)$ we see that $g = \mathit{lift}(f) : A^+ \to B^+$ satisfies the following: (1) g is monotone, i.e., $u \leq v$ implies $g(u) \leq g(v)$ for all $u, v \in A^+$; and (2) g is length-preserving, i.e., $|f(u)| = |u|$ for every $u \in A^+$. We write $\mathsf{MT}(A, B)$ to denote the set of monotone and length-preserving functions of type $A^+ \to B^+$. An element of $\mathsf{MT}(A, B)$ is said to be a *Mealy transduction* in cumulative form. Serial composition is composition of functions, and $\mathsf{id}_A^{\mathsf{IMT}} : \mathsf{MT}(A, A)$ is the identity function. The mapping *lift* witnesses the isomorphism between the categories IMT and MT.

If we think of a Mealy transduction $f : \mathsf{MT}(A, B)$ operationally, it describes a transformation that consumes the input stream item by item and produces exactly one output item for every consumed input item. If this restriction is lifted to allow an arbitrary (but finite) number of output items per input item, we need to consider a *stream transduction*, which is a monotone function $g : A^* \to B^*$. We write $\mathsf{ST}(A, B)$ to denote the set of all such functions. As for MT, serial composition is composition of functions, and the identity $\mathsf{id}_A^{\mathsf{ST}} : \mathsf{ST}(A, A)$ is the identity function. For the incremental viewpoint, we define $\mathsf{IST}(A, B) = A^* \to B^*$ and $\mathit{lift}(f)(a_1 \dots a_n) = f(\varepsilon) \cdot f(a_1) \cdots f(a_1 \dots a_n)$ for every $f : \mathsf{IST}(A, B)$ and $a_1 \dots a_n \in A^*$. Serial composition is given by $(f \gg g)(u) = g(\mathit{lift}(f)(u))$ for every $u \in A^*$, $f : \mathsf{IST}(A, B)$ and $g : \mathsf{IST}(B, C)$. The identity $\mathsf{id}_A^{\mathsf{IST}} : \mathsf{IST}(A, A)$ is given by $\mathsf{id}_A^{\mathsf{IST}}(\varepsilon) = \varepsilon$ and $\mathsf{id}_A^{\mathsf{IST}}(a_1 \dots a_n) = a_n$. The mapping *lift* is an isomorphism from IST to ST. The isomorphism from ST to IST is given by the *differentiation* function, denoted ∂, which sends a function $f : \mathsf{ST}(A, B)$ to a function $\partial(f) : \mathsf{IST}(A, B)$, where $\partial(f)(\varepsilon) = f(\varepsilon)$ and $\partial(f)(ua) = f(u)^{-1} f(ua)$.

If the restriction of finite output per input item is lifted, we need to consider the set $A^\infty = A^* \cup A^\omega$ of finite and infinite words over A. Notice that A^∞ is an ω-CPO: it is partially ordered by the prefix relation, and every countable chain has a supremum. A *Kahn transduction* is a function $f : A^\infty \to B^\infty$ which is ω-continuous: $f(\sup_{i<\omega} x_i) = \sup_{i<\omega} f(x_i)$ for every chain $x_0 \leq x_1 \leq x_2 \leq \dots$ in A^*. We denote the set of all these functions by $\mathsf{KT}(A, B)$. Informally, a Kahn transduction can describe computations where the consumption of a single input item can trigger the production of an infinite sequence of output items. As in ST, serial composition is composition of functions and $\mathsf{id}_A^{\mathsf{KT}} : \mathsf{KT}(A, A)$ is the identity function.

	item	input history	output increment $f : \mathsf{IST}(A, B)$	output history $g : \mathsf{ST}(A, B)$
time ↓		ε	$f(\varepsilon)$	
	a_0	a_0	$f(a_0)$	$g(a_0)$
	a_1	$a_0\, a_1$	$f(a_0\, a_1)$	$g(a_0\, a_1)$
	a_2	$a_0\, a_1\, a_2$	$f(a_0\, a_1\, a_2)$	$g(a_0\, a_1\, a_2)$
	$\dots$			

Fig. 1. The transductions $f : \mathsf{IST}(A, B)$ and $g : \mathsf{ST}(A, B)$ with $f = \partial(g)$ and $g = \mathit{lift}(f)$ describe the same transformation.

The *extension* of a stream transduction $f : \mathsf{ST}(A, B)$ to a Kahn transduction $\mathsf{ext}(f) : \mathsf{KT}(A, B)$ is defined as follows: $\mathsf{ext}(f)(a_0 a_1 \dots) = \sup_{i<\omega} f(a_0 \dots a_i)$. This is well-defined because f is monotone and therefore $f(a_0)$, $f(a_0 a_1)$, $\dots$ is a chain. The mapping ext embeds ST into KT.

One problem with stream transductions and Kahn transductions is that they describe computations that can potentially stop producing output. We are interested here in computations over unbounded signals, which describe (in the limit) the transformation of an infinite input signal to an infinite output signal. We call this requirement the ***progress property***. It is formulated easily for Kahn transductions as follows: a function $f : \mathsf{KT}(A, B)$ satisfies the progress property if $f(u) \in B^\omega$ for every $u \in A^\omega$. Intuitively, this says that f never stops producing output while processing the input signal $u \in A^\omega$. This property says nothing about when the output is produced, only that there is no point at which f stops producing output. We write $\mathsf{KT}_\Diamond(A, B)$ to denote the subset of $\mathsf{KT}(A, B)$ that contains those transductions that satisfy the progress property. The operation of serial composition preserves the progress property, and every $\mathsf{id}_A^{\mathsf{KT}}$ satisfies the progress property. So, $\mathsf{KT}_\Diamond$ is a subcategory of KT. A stream transduction $f : \mathsf{ST}(A, B)$ is said to satisfy the progress property if $\mathsf{ext}(f) : \mathsf{KT}(A, B)$ satisfies the progress property. $\mathsf{ST}_\Diamond$ is the subcategory of ST that consists of the stream transductions satisfying the progress property.

Mealy transductions correspond to signal transformations where the input and output streams are perfectly synchronized: upon receipt of the input item for timepoint t, the output item for timepoint t is produced. Stream transductions relax this synchronicity requirement: they allow the output to fall behind relative to the input or to run ahead of it. When the output falls behind we say that the computation has *lag*, and when the output runs ahead we say that the computation has *lead*. The main class of transductions that we consider here are the ones with bounded lead and lag, since they are very useful in the specification of signal processing algorithms. We define the *lead* of a transduction $f : \mathsf{ST}(A, B)$ at $u \in A^*$ to be the integer $\mathsf{lead}(f)(u) = |f(u)| - |u|$. Similarly, the *lag* of a transduction $f : \mathsf{ST}(A, B)$ at $u \in A^*$ is defined as the

integer $\mathsf{lag}(f)(u) = |u| - |f(u)|$. So, $\mathsf{lead}(f)$ and $\mathsf{lag}(f)$ are are functions of type $A^* \to \mathbb{Z}$. We also define $\mathsf{Lead}(f) = \{\mathsf{lead}(f)(u) \mid u \in A^*\}$. We say that f is ***lead/lag-bounded*** or ***ℓ-bounded*** when $\mathsf{Lead}(f)$ is finite. We write $\mathsf{ST}_\mathsf{b}(A, B)$ for the set of all ℓ-bounded stream transductions. If $f : \mathsf{ST}_\mathsf{b}(A, B)$ and $g : \mathsf{ST}_\mathsf{b}(B, C)$, then it also holds that $f \gg g : \mathsf{ST}_\mathsf{b}(A, C)$. It follows that ST_b is a subcategory of $\mathsf{ST}_\Diamond$.

We think of each $\mathsf{ST}_\mathsf{b}(A, B)$ as a type that can be refined according to the magnitude of the lead/lag. We write $f : \mathsf{ST}_\mathsf{b}(A, B, [m, n])$, where $m \leq n$ are integers, when $\mathsf{Lead}(f) \subseteq [m, n]$. The following hold: $\mathsf{ST}_\mathsf{b}(A, B, [m, n]) \subseteq \mathsf{ST}_\mathsf{b}(A, B) \subseteq \mathsf{ST}_\Diamond(A, B)$.

Lemma 1. If $f : \mathsf{ST}_\mathsf{b}(A, B, [m, n])$ and $g : \mathsf{ST}_\mathsf{b}(B, C, [o, p])$ then $f \gg g : \mathsf{ST}_\mathsf{b}(A, C, [m + o, n + p])$.

Consider a function $f : \mathsf{ST}_\mathsf{b}(A, B, [m, n])$. We claim that $m \leq 0$ implies $n \geq 0$. Equivalently, the interval $[m, n]$ must contain at least one non-negative number. In other words, the set $\mathsf{ST}_\mathsf{b}(A, B, [m, n])$ is empty when $m, n < 0$. This is because $\mathsf{lead}(f)(\varepsilon) = |f(\varepsilon)| - |\varepsilon| = |f(\varepsilon)| \geq 0$. So, when $\mathsf{ST}_\mathsf{b}(A, B, [m, n])$ is nonempty then $[m, n]$ should intersect $\mathbb{N}$. Equivalently, we require that $n \geq 0$. We define a ***lead interval*** to be a subset $[m, n] = \{i \in \mathbb{Z} \mid m \leq i \leq n\}$ of the integers with $m \leq n$ and $n \geq 0$. If $[m, n]$ and $[o, p]$ are lead intervals, then so is $[m + o, n + p]$.

Example 2. Consider the transductions $f, g, h, k : \mathsf{ST}(\mathbb{N}, \mathbb{N})$, all of which represent the identity function on infinite streams. We write $\partial(f), \partial(g), \partial(h), \partial(k) : \mathsf{IST}(\mathbb{N}, \mathbb{N})$ for the corresponding incremental versions, which are given as follows:

$$\partial(f)(x_0 \dots x_n) = x_n$$

$$\partial(g)(x_0 \dots x_n) = \begin{cases} \varepsilon, & \text{if } n \text{ is even} \\ x_{n-1}x_n, & \text{if } n \text{ is odd} \end{cases}$$

$$\partial(h)(x_0 \dots x_n) = \begin{cases} n/2, & \text{if } n \text{ is even} \\ \varepsilon, & \text{otherwise} \end{cases}$$

$$\partial(k)(x_0 \dots x_n) = \begin{cases} \log_2(n), & \text{if } n \text{ is a power of 2} \\ \varepsilon, & \text{otherwise} \end{cases}$$

The following table illustrates f, g, h, k with an example:

input:	0	1	2	3	4	5	6	7	8
f output:	0	1	2	3	4	5	6	7	8
g output:		0 1		2 3		4 5		6 7	
h output:	0		1		2		3		4
k output:		0	1		2				3

We observe the following: (1) f is a Mealy transduction; (2) g has bounded lead/lag, in particular $g : \mathsf{ST}_\mathsf{b}(\mathbb{N}, \mathbb{N}, [-1, 0])$, but it is not a Mealy transduction; (3) h and k both satisfy the progress property, but they do not have bounded lead/lag.

The denotational models that we have considered in this section describe the transformation of discrete signals, which are typically uniformly sampled. These models can be varied and/or extended in several ways. It is possible to consider timed traces [41] to represent non-uniformly sampled discrete signals and dense-time (or continuous-time) signals [2]. The online transformation of signals can be presented in the broader context of data stream processing. A generalization of the notion of data streams to partial orders is considered in [42], [43]. An algebraic semantic framework that encompasses several concrete stream models (including discrete and continuous signals) is proposed in [44].

III. Transducers with Delay

In this section we introduce a class of transducers (automata) that compute the lead/lag-bounded transductions of section II. These transducers are more general than Mealy machines [45] and Moore machines [46]. They are more similar to the so-called sequential transducers [47], [48], but there are two key differences: (1) we do not restrict attention to finite-state transducers, and (2) we consider a type discipline for ensuring that the transducers have bounded lead/lag.

Definition 3 (Stream Transducer). Let A and B be sets. A *stream transducer* of type $\mathsf{SA}(A, B)$ is a deterministic transducer $G = (\mathsf{St}, \mathsf{init}, \mathsf{o}, \mathsf{next}, \mathsf{out})$, where St is a set of *states*, $\mathsf{init} \in \mathsf{St}$ is the *initial state*, $\mathsf{o} \in B^*$ is the *initial output*, $\mathsf{next} : \mathsf{St} \times A \to \mathsf{St}$ is the *transition function*, $\mathsf{out} : \mathsf{St} \times A \to B^*$ is the *output function*. The transition function extends to $\mathsf{gnext} : \mathsf{St} \times A^* \to \mathsf{St}$, given by $\mathsf{gnext}(s, \varepsilon) = s$ and $\mathsf{gnext}(s, ua) = \mathsf{next}(\mathsf{gnext}(s, u), a)$. Similarly, the output function extends to $\mathsf{gout} : \mathsf{St} \times A^* \to B^*$, given by $\mathsf{gout}(s, \varepsilon) = \varepsilon$ and $\mathsf{gout}(s, ua) = \mathsf{gout}(s, u) \cdot \mathsf{out}(\mathsf{gnext}(s, u), a)$. The *denotation* of G is the stream transduction $[\![G]\!] : \mathsf{ST}(A, B)$ given by $[\![G]\!](\varepsilon) = \mathsf{o}$ and $[\![G]\!](ua) = [\![G]\!](u) \cdot \mathsf{out}(\mathsf{gnext}(\mathsf{init}, u), a)$. We say that G *implements* the transduction f if $[\![G]\!] = f$.

Let A, B be sets and $G = (\mathsf{St}, \mathsf{init}, \mathsf{o}, \mathsf{next}, \mathsf{out}) : \mathsf{SA}(A, B)$ be a stream transducer. A ***lead labeling*** for G is a pair $\langle \lambda, \mu \rangle$ of functions $\lambda, \mu : \mathsf{St} \to \mathbb{Z}$ such that
(LL1) $\lambda(s) \leq \mu(s)$ for every $s \in \mathsf{St}$,
(LL2) $\lambda(\mathsf{next}(s, a)) \leq \lambda(s) + |\mathsf{out}(s, a)| - 1$ for every s, a,
(LL3) $\mu(s) + |\mathsf{out}(s, a)| - 1 \leq \mu(\mathsf{next}(s, a))$ for all s, a, and
(LL4) $\lambda(\mathsf{init}) \leq |\mathsf{o}| \leq \mu(\mathsf{init})$.
For integers $m \leq n$, we say that $\langle \lambda, \mu \rangle$ is a $[m, n]$-lead labeling if $m \leq \lambda(s)$ and $\mu(s) \leq n$ for all $s \in \mathsf{St}$.

Lemma 4 (Lead Labeling). Let $G : \mathsf{SA}(A, B)$ be a transducer. If G has a lead labeling $\langle \lambda, \mu \rangle$ then $\lambda(\mathsf{gnext}(\mathsf{init}, u)) \leq \mathsf{lead}([\![G]\!])(u) \leq \mu(\mathsf{gnext}(\mathsf{init}, u))$ for every $u \in A^*$.

Lemma 4 says that a lead labeling $\langle \lambda, \mu \rangle$ of a transducer G provides lower bounds (with λ) and upper bounds (with μ) for the lead of the transduction that G implements.

Lemma 5 (Lead/Lag Boundedness). Let A, B be sets and $m \leq n$ be integers. The denotation of a stream transducer $G : \mathsf{SA}(A, B)$ belongs to $\mathsf{ST}_\mathsf{b}(A, B, [m, n])$ iff G has a $[m, n]$-lead labeling $\langle \lambda, \mu \rangle$.

Lemma 5 says that the existence of a bounded lead labeling for a transducer G is equivalent to G implementing a transduction with bounded lead/lag. This allows us to prove the latter semantic property with a simple annotation of the transducer. We write $G : \mathsf{SA}(A, B, [m, n])$ when G has a $[m, n]$-lead labeling, and we say that it is *lead/lag bounded*. We write $\mathsf{SA}(A, B, n)$ as an abbreviation for $\mathsf{SA}(A, B, [n, n])$.

$$\mathsf{id} : \mathsf{ST_b}(A, A, 0) \qquad \mathsf{id}(u) = u$$

$$\frac{op : A \to B}{f = \mathsf{map}(op) : \mathsf{ST_b}(A, B, 0)} \qquad \begin{aligned}\partial(f)(\varepsilon) &= \varepsilon \\ \partial(f)(ua) &= op(a)\end{aligned}$$

$$\frac{in : A \to B \qquad op : B \times A \to B}{f = \mathsf{aggr}(in, op) : \mathsf{ST_b}(A, B, 0)} \qquad \begin{aligned}\partial(f)(\varepsilon) &= \varepsilon \\ \partial(f)(ua) &= fold(\mathsf{init}, op, ua)\end{aligned}$$

$$\frac{\text{integer } n \geq 1}{f = \mathsf{wnd}(n) : \mathsf{ST_b}(A, A^n, [-(n-1), 0])} \qquad \begin{aligned}\partial(f)(u) &= \varepsilon, \text{ if } |u| < n \\ \partial(f)(uv) &= v, \text{ if } |v| = n\end{aligned}$$

$$\frac{\text{integer } n \geq 1}{f = \mathsf{ignore}(n) : \mathsf{ST_b}(A, A, [-n, 0])} \qquad \begin{aligned}f(u) &= \varepsilon, \text{ if } |u| < n \\ f(uv) &= v, \text{ if } |u| = n\end{aligned}$$

$$\frac{n : \mathbb{N} \qquad val : B}{f = \mathsf{emit}(n, val) : \mathsf{ST_b}(A, B, n)} \qquad f(u) = val^n \cdot u$$

$$\frac{f : \mathsf{ST_b}(A, B, [m, n]) \qquad g : \mathsf{ST_b}(B, C, [o, p])}{\begin{aligned}f \gg g &: \mathsf{ST_b}(A, C, [m + o, n + p] \\ (f \gg g)(u) &= g(f(u))\end{aligned}}$$

$$\frac{f : \mathsf{ST_b}(A, B, [m, n]) \qquad g : \mathsf{ST_b}(A, C, [o, p])}{\begin{aligned}h = \mathsf{par}(f, g) &: \mathsf{ST_b}(A, B \times C, [\min(m, o), \min(n, p)]) \\ h(u) &= zip(f(u), g(u))\end{aligned}}$$

$$\frac{f : \mathsf{ST_b}(A \times B, B, [m, n]) \qquad m \geq 1}{\begin{aligned}g = \mathsf{loop}(f) &: \mathsf{ST_b}(A, B, [m, n]) \\ g(u) &= lfp(\tau_u), \text{ where } \tau_u(v) = f(zip(u, v)) \text{ for } v \in B^*\end{aligned}}$$

Fig. 2. Combinators for lead/lag-bounded transductions.

Example 6 (Sliding Window). We define the stream transducer $\mathtt{wnd}(n) : \mathsf{SA}(A, A^n)$ that partitions the input into overlapping segments of n elements, as shown below:

input:	0	1	2	3	4	5
$\mathtt{wnd}(2)$ output:		[0,1]	[1,2]	[2,3]	[3,4]	[4,5]
$\mathtt{wnd}(3)$ output:			[0,1,2]	[1,2,3]	[2,3,4]	[3,4,5]

The idea is to maintain a buffer that remembers the last $n-1$ elements. So, we define the transducer as follows:

$$\begin{aligned}\mathtt{wnd}(n) &= (\mathsf{St}, \mathsf{init}, \mathsf{o}, \mathsf{next}, \mathsf{out}) : \mathsf{SA}(A, A^n) \\ \mathsf{St} &= \textstyle\bigcup_{i=0}^{n-1} A^i, \ \mathsf{init} = [] \text{ and } \mathsf{o} = \varepsilon \\ \mathsf{next}(s, b) &= s \cdot [b], \text{ if } |s| < n - 1 \\ \mathsf{next}([a] \cdot s, b) &= s \cdot [b], \text{ if } |s| = n - 2 \\ \mathsf{out}(s, b) &= \varepsilon, \text{ if } |s| < n - 1 \\ \mathsf{out}(s, b) &= s \cdot [b], \text{ if } |s| = n - 1\end{aligned}$$

Now, we will define a $[-(n-1), 0]$-labeling $\langle \lambda, \mu \rangle$ for the transducer. We put $\lambda(s) = \mu(s) = -|s|$ for every $s \in \mathsf{St}$. Observe that $\lambda(\mathsf{next}(s, a)) = \lambda(s) + |\mathsf{out}(s, a)| - 1$ for every $s \in \mathsf{St}$ and $a \in A$. So, $\mathtt{wnd}(n) : \mathsf{SA}(A, A^n, [-(n-1), 0]$.

IV. Combinators

In this section we introduce a collection of combinators for stream transductions (semantic objects) and stream transducers (model of computation). These combinators constitute a domain-specific language (DSL) for programming signal transductions of bounded lead/lag. This language satisfies a key efficiency guarantee: every transducer that is defined in it requires a constant amount of memory space and a constant amount of computation time to process each input sample.

$$\mathtt{id} : \mathsf{SA}(A, A, 0) \qquad \frac{op : A \to B}{\mathtt{map}(op) : \mathsf{SA}(A, B, 0)}$$

$$\frac{in : A \to B \qquad op : B \times A \to B}{\mathtt{aggr}(in, op) : \mathsf{SA}(A, B, 0)} \qquad \frac{\text{integer } n \geq 1}{\mathtt{wnd}(n) : \mathsf{SA}(A, A^n, [-(n-1), 0])}$$

$$\frac{n : \mathbb{N}}{\mathtt{ignore}(n) : \mathsf{SA}(A, A, [-n, 0])} \qquad \frac{n : \mathbb{N} \qquad val : B}{\mathtt{emit}(n, val) : \mathsf{SA}(A, B, n)}$$

$$\frac{\mathtt{f} : \mathsf{SA}(A, B, [m, n]) \qquad \mathtt{g} : \mathsf{SA}(B, C, [o, p])}{\mathtt{f} \gg \mathtt{g} : \mathsf{SA}(A, C, [m + o, n + p])}$$

$$\frac{\mathtt{f} : \mathsf{SA}(A, B, [m, n]) \qquad \mathtt{g} : \mathsf{SA}(A, C, [o, p])}{\mathtt{par}(\mathtt{f}, \mathtt{g}) : \mathsf{SA}(A, B \times C, [\min(m, o), \min(n, p)])}$$

$$\frac{\mathtt{f} : \mathsf{SA}(A \times B, B, [m, n]) \qquad m \geq 1}{\mathtt{loop}(\mathtt{f}) : \mathsf{SA}(A, B, [m, n])}$$

Fig. 3. Combinators for lead/lag-bounded transducers.

In Figure 2 we show several combinators on stream transductions. The $\mathsf{map}(op)$ combinator describes a signal transformation that applies the function $op : A \to B$ elementwise. The $\mathsf{aggr}(in, op)$ combinator is a running aggregation, which is specified by the initialization function $in : A \to B$ (applied to the first element) and the aggregation function $op : B \times A \to B$. The definition uses the fold operation:

$$\begin{aligned}fold &: (A \to A) \times (B \times A \to B) \times A^+ \to B \\ fold(in, op, a) &= in(a), \text{ for } a \in A \\ fold(in, op, ua) &= op(fold(in, op, u), a)\end{aligned}$$

The $\mathsf{wnd}(n)$ combinator describes the transformation that partitions the input signal into a sequence of overlapping windows, so that the i-th output is a list of the last n elements from i to $i + n - 1$. The $\mathsf{ignore}(n)$ combinator skips the first n elements and echoes the rest of the input. The $\mathsf{emit}(n, val)$ combinator emits n copies of val at the beginning and then continues to echo the input. The serial composition combinator $\gg$ is used to stream the output of one transduction as input to another one. The parallel composition combinator describes the simultaneous application of two transformations. In order to define this, we consider the function $zip : A^* \times B^* \to (A \times B)^*$, which is given by $zip(a_1 \ldots a_m, b_1 \ldots b_n) = \langle a_1, b_1 \rangle \ldots \langle a_k, b_k \rangle$ where $k = \min(m, n)$. It follows that $|zip(u, v)| = \min(|u|, |v|)$ for all $u \in A^*$ and $v \in B^*$. For the feedback combinator loop, we consider a transduction $f : \mathsf{ST_b}(A \times B, B, [m, n])$ with $m \geq 1$ and we want to define $\mathsf{loop}(f) : \mathsf{ST_b}(A, B)$ as a solution to the equation $\mathsf{par}(\mathsf{id}, g) \gg f = g$. We can rewrite this equivalently as $f(zip(u, g(u))) = g(u)$ for every $u \in A^*$. The function $\tau_u : B^* \to B^*$, given by $\tau_u(v) = f(zip(u, v))$, has a least fixpoint $lfp(\tau_u) = \sup_{i<\omega} v_i$ where $v_0 = \varepsilon$ and $v_{i+1} = \tau_u(v_i)$.

Figure 3 contains the corresponding combinators for stream transducers, i.e., the implementations. In Figure 4 we show the implementation of the ***serial composition*** combinator $(\mathtt{f}_1 \gg \mathtt{f}_2)$ using a product construction on $\mathtt{f}_1$ and $\mathtt{f}_2$. The idea is that the output of $\mathtt{f}_1$ is propagated as input to $\mathtt{f}_2$. The ***parallel composition*** combinator $\mathtt{par}(\mathtt{f}_1, \mathtt{f}_2)$ is implemented with a modified product construction on $\mathtt{f}_1$ and $\mathtt{f}_2$, as shown in Figure 5. Observe that the state space is

$$\begin{aligned}
\mathtt{f}_1 &= (\mathsf{St}_1, \mathsf{init}_1, \mathsf{o}_1, \mathsf{next}_1, \mathsf{out}_1) : \mathsf{SA}(A, B)\\
\mathtt{f}_2 &= (\mathsf{St}_2, \mathsf{init}_2, \mathsf{o}_2, \mathsf{next}_2, \mathsf{out}_2) : \mathsf{SA}(B, C)\\
(\mathtt{f}_1 \gg \mathtt{f}_2) &= (\mathsf{St}_1 \times \mathsf{St}_2, \mathsf{init}, \mathsf{o}, \mathsf{next}, \mathsf{out}) : \mathsf{SA}(A, C)\\
\mathsf{init} &= \langle \mathsf{init}_1, \mathsf{gnext}_2(\mathsf{init}_2, \mathsf{o}_1)\rangle \text{ and } \mathsf{o} = \mathsf{o}_2 \cdot \mathsf{out}_2(\mathsf{init}_2, \mathsf{o}_1)\\
\mathsf{next}(\langle s_1, s_2\rangle, a) &= \langle \mathsf{next}_1(s_1, a), \mathsf{gnext}_2(s_2, \mathsf{out}_1(s_1, a))\rangle\\
\mathsf{out}(\langle s_1, s_2\rangle, a) &= \mathsf{out}_2(s_2, \mathsf{out}_1(s_1, a))
\end{aligned}$$

Fig. 4. Implementation of the $\gg$ combinator.

$$\begin{aligned}
\mathtt{f}_1 &= (\mathsf{St}_1, \mathsf{init}_1, \mathsf{o}_1, \mathsf{next}_1, \mathsf{out}_1) : \mathsf{SA}(A, B_1)\\
\mathtt{f}_2 &= (\mathsf{St}_2, \mathsf{init}_2, \mathsf{o}_2, \mathsf{next}_2, \mathsf{out}_2) : \mathsf{SA}(A, B_2)\\
\mathtt{par}(\mathtt{f}_1, \mathtt{f}_2) &= (\mathsf{St}, \mathsf{init}, \mathsf{o}, \mathsf{next}, \mathsf{out}) : \mathsf{SA}(A, B_1 \times B_2)\\
\mathsf{St} &= \mathsf{St}_1 \times \mathsf{St}_2 \times B_1^* \times B_2^*\\
\mathsf{init} &= \langle \mathsf{init}_1, \mathsf{init}_2, \mathit{remn}_1(\mathsf{o}_1, \mathsf{o}_2), \mathit{remn}_2(\mathsf{o}_1, \mathsf{o}_2)\rangle\\
\mathsf{o} &= \mathit{zip}(\mathsf{o}_1, \mathsf{o}_2)\\
\mathsf{next}(\langle s_1, s_2, u_1, u_2\rangle, a) &= \langle t_1, t_2, w_1, w_2\rangle\\
\mathsf{out}(\langle s_1, s_2, u_1, u_2\rangle, a) &= \mathit{zip}(u_1 v_1, u_2 v_2)\\
t_1 = \mathsf{next}(s_1, a) &\text{ and } t_2 = \mathsf{next}(s_2, a)\\
v_1 = \mathsf{out}(s_1, a) &\text{ and } v_2 = \mathsf{out}(s_2, a)\\
w_1 = \mathit{remn}_1(u_1 v_1, u_2 v_2) &\text{ and } w_2 = \mathit{remn}_2(u_1 v_1, u_2 v_2)
\end{aligned}$$

Fig. 5. Implementation of the par combinator.

$$\begin{aligned}
\mathtt{f} &= (\mathsf{St}, \mathsf{init}, \mathsf{o}, \mathsf{next}, \mathsf{out}) : \mathsf{SA}(A \times B, B)\\
\mathtt{loop}(\mathtt{f}) &= (\mathsf{St} \times B^+, \langle \mathsf{init}, \mathsf{o}\rangle, \mathsf{o}, \mathsf{next}', \mathsf{out}') : \mathsf{SA}(A, B)\\
\mathsf{next}'(\langle s, bv\rangle, a) &= \langle \mathsf{next}(s, \langle a, b\rangle), v \cdot \mathsf{out}(s, \langle a, b\rangle)\rangle\\
\mathsf{out}'(\langle s, bv\rangle, a) &= \mathsf{out}(s, \langle a, b\rangle)
\end{aligned}$$

Fig. 6. Implementation of the loop combinator.

defined as $\mathsf{St} = \mathsf{St}_1 \times \mathsf{St}_2 \times B_1^* \times B_2^*$. This is because the transducer has to properly align the output emitted by $\mathtt{f}_1$ and $\mathtt{f}_2$, and therefore it has to remember the additional output given by $\mathtt{f}_1$ when $\mathtt{f}_1$ runs ahead of $\mathtt{f}_2$, and similarly for the symmetric case when $\mathtt{f}_2$ runs ahead of $\mathtt{f}_1$. Every reachable state $\langle s_1, s_2, u_1, u_2\rangle$ satisfies the following invariant: $u_1 = \varepsilon$ or $u_2 = \varepsilon$. The functions $\mathit{remn}_1 : A^* \times B^* \to A^*$ and $\mathit{remn}_2 : A^* \times B^* \to B^*$ are used in Figure 5. For $u = a_1 \dots a_m \in A^*$ and $v = b_1 \dots b_n \in B^*$, we put $\mathit{remn}_1(u, v) = a_{k+1} \dots a_m$ and $\mathit{remn}_2(u, v) = b_{k+1} \dots b_n$ where $k = \min(m, n)$. In particular, $m \le n$ implies that $\mathit{remn}_1(u, v) = \varepsilon$, and $n \le m$ implies that $\mathit{remn}_2(u, v) = \varepsilon$. So, $\mathit{remn}_1(u, v) = \varepsilon$ or $\mathit{remn}_2(u, v) = \varepsilon$. In Figure 6 we give the implementation of the ***feedback composition*** combinator $\mathtt{loop}(\mathtt{f})$, where $\mathtt{f} : \mathsf{SA}(A \times B, B)$ is a transducer with two input channels (one of type A and one of type B). The idea is that the output of $\mathtt{f}$ is given as input to the second input channel of $\mathtt{f}$. In order for this to be well-defined, it is essential that $\mathtt{f}$ has lead in $[m, n]$ with $m \ge 1$.

Example 7 (FIR filtering). For an odd natural number $w = 2k + 1$ and coefficients $c_{-k}, \dots, c_{-1}, c_0, c_1, \dots, c_k$ we define the output signal by $y(n) = \sum_{i=-k}^{k} c_i \cdot x(n + i)$. We consider that $x(n) = 0$ for $n < 0$.

$$\frac{k \ge 0 \qquad \text{coefficients } \bar{c} = c_{-k}, \dots, c_{-1}, c_0, c_1, \dots, c_k}{\mathtt{FIR}(\bar{c}) : \mathsf{SA}(\mathbb{R}, \mathbb{R}, [-k, 0])}$$

An FIR filter can be used to smooth a signal. For example, $\mathtt{FIR}([0.1, 0.2, 0.4, 0.2, 0.1])$ is a weighted average around each point. We can express FIR in terms of emit, wnd and map:

$$\mathtt{FIR}([c_{-1}, c_0, c_1]) = \mathtt{emit}(1, 0) \gg \mathtt{wnd}(3) \gg \mathtt{map}(\mathit{op}),$$

where $\mathit{op} = (x_{-1}, x_0, x_1) \text{ -> } c_{-1}x_{-1} + c_0 x_0 + c_1 x_1$.

$$\frac{p : A \to \mathtt{Bool}}{\mathsf{atomic}(p) : \mathsf{MTL}(A)} \qquad \frac{\varphi, \psi : \mathsf{MTL}(A)}{\neg\varphi,\ \varphi \wedge \psi,\ \varphi \vee \psi : \mathsf{MTL}(A)}$$

$$\frac{\varphi : \mathsf{MTL}(A) \qquad \text{integers } 0 \le a \le b}{\mathsf{Y}\varphi,\ \mathsf{P}\varphi,\ \mathsf{H}\varphi,\ \mathsf{P}_{[a,\infty)}\varphi,\ \mathsf{H}_{[a,\infty)}\varphi,\ \mathsf{P}_{[a,b]}\varphi,\ \mathsf{H}_{[a,b]}\varphi : \mathsf{MTL}(A)}$$

$$\frac{\varphi, \psi : \mathsf{MTL}(A) \qquad \text{integers } 0 \le a \le b}{\varphi\ \mathsf{S}\ \psi,\ \varphi\ \mathsf{S}_{[a,\infty)}\ \psi,\ \varphi\ \mathsf{S}_{[a,b]}\ \psi : \mathsf{MTL}(A)}$$

$$\frac{\varphi, \psi : \mathsf{MTL}(A) \qquad \text{integers } 0 \le a \le b}{\mathsf{N}\varphi,\ \mathsf{N}^a\varphi,\ \mathsf{F}_{[a,b]}\varphi,\ \mathsf{G}_{[a,b]}\varphi,\ \varphi\ \mathsf{U}_{[a,b]}\ \psi : \mathsf{MTL}(A)}$$

Fig. 7. The syntax of typed metric temporal logic (MTL).

Theorem 8. The transducer combinators of Figure 3 implement the corresponding transduction combinators of Figure 2. Moreover, every implementation constructed with these combinators is an online algorithm that uses constant space (i.e., independent of the length of the input stream).

Proof. First, we show that each combinator id, $\mathtt{map}(\mathit{op})$, $\mathtt{aggr}(\mathit{in}, \mathit{op})$, $\mathtt{wnd}(n)$, $\mathtt{ignore}(n)$ and $\mathtt{emit}(n, \mathit{val})$ implements id, $\mathsf{map}(\mathit{op})$, $\mathsf{aggr}(\mathit{in}, \mathit{op})$, $\mathsf{wnd}(n)$, $\mathsf{ignore}(n)$ and $\mathsf{emit}(n, \mathit{val})$ respectively. Now, for serial composition it suffices to show the following: if $\mathtt{f} : \mathsf{SA}(A, B)$ implements $f : \mathsf{ST}(A, B)$ and $\mathtt{g} : \mathsf{SA}(B, C)$ implements $g : \mathsf{ST}(A, B)$, then $\mathtt{f} \gg \mathtt{g} : \mathsf{SA}(A, C)$ implements $f \gg g : \mathsf{ST}(A, C)$. Similar claims can be established for parallel composition and feedback composition.

We will show now that the implementation of each combinator requires a constant amount of space for the transducer memory. First, we observe that every transducer built with the combinators of Figure 3 is lead/lag-bounded. The combinators id and $\mathtt{map}(\mathit{op})$ do not require any memory, i.e., they are memoryless. The running aggregation $\mathtt{aggr}(\mathit{in}, \mathit{op})$ requires one memory location for storing the running aggregate. The windowing transducer $\mathtt{wnd}(n)$ requires $n - 1$ memory locations for the buffer of the last $n - 1$ elements. As seen in Figure 4, the serial composition $\mathtt{f} \gg \mathtt{g}$ does not require any additional memory locations. For the case of parallel composition $\mathtt{g} = \mathtt{par}(\mathtt{f}_1, \mathtt{f}_2)$, the argument relies crucially on the assumption that $\mathtt{f}_1$ and $\mathtt{f}_2$ are lead/lag-bounded. In this case, the skew between $\mathtt{f}_1 : \mathsf{SA}(A, B_1, [m_1, n_1])$ and $\mathtt{f}_2 : \mathsf{SA}(A, B_2, [m_2, n_2])$ is bounded above by $a = \max(|m_1 - n_2|, |m_2 - n_1|)$. So, g requires a buffer with at most a memory locations. For the case of feedback composition $\mathtt{loop}(\mathtt{f})$ with $\mathtt{f} : \mathsf{SA}(A \times B, B, [m, n])$, a buffer with at most n memory locations is required, in order to store the elements that are sent back to the input channel B and await to be matched with elements of the input channel A. □

V. Online Temporal Monitoring

In this section we show how our framework of lead/lag-bounded transducers (and the corresponding combinators) can be used to easily prototype an efficient monitoring algorithm

$$
\begin{aligned}
u,i \models \mathsf{atomic}(p) &\Leftrightarrow p(u(i)) = \mathtt{true}\\
u,i \models \neg\varphi &\Leftrightarrow u,i \not\models \varphi\\
u,i \models \varphi \wedge \psi &\Leftrightarrow u,i \models \varphi \text{ and } u,i \models \psi\\
u,i \models \varphi \vee \psi &\Leftrightarrow u,i \models \varphi \text{ or } u,i \models \psi\\
u,i \models \mathsf{Y}\varphi &\Leftrightarrow i \geq 1 \text{ and } u,i-1 \models \varphi\\
u,i \models \mathsf{P}\varphi &\Leftrightarrow u,j \models \varphi \text{ for some } j \in [0,i]\\
u,i \models \mathsf{H}\varphi &\Leftrightarrow u,j \models \varphi \text{ for every } j \in [0,i]\\
u,i \models \varphi\,\mathsf{S}\,\psi &\Leftrightarrow \text{there is } j \in [0,i] \text{ s.t. } u,j \models \psi\\
&\qquad \text{and } u,k \models \varphi \text{ for all } k \text{ with } j < k \leq i\\
u,i \models \mathsf{P}_{[a,\infty)}\varphi &\Leftrightarrow u,j \models \varphi \text{ for some } j \in [0,i-a]\\
u,i \models \mathsf{H}_{[a,\infty)}\varphi &\Leftrightarrow u,j \models \varphi \text{ for every } j \in [0,i-a]\\
u,i \models \varphi\,\mathsf{S}_{[a,\infty)}\,\psi &\Leftrightarrow \text{there is } j \in [0,i-a] \text{ s.t. } u,j \models \psi\\
&\qquad \text{and } u,k \models \varphi \text{ for all } k \text{ with } j < k \leq i\\
u,i \models \mathsf{P}_{[a,b]}\varphi &\Leftrightarrow u,j \models \varphi \text{ for some } j \in [i-b,i-a]\\
u,i \models \mathsf{H}_{[a,b]}\varphi &\Leftrightarrow u,j \models \varphi \text{ for every } j \in [i-b,i-a]\\
u,i \models \varphi\,\mathsf{S}_{[a,b]}\,\psi &\Leftrightarrow \text{there is } j \in [i-b,i-a] \text{ s.t. } u,j \models \psi\\
&\qquad \text{and } u,k \models \varphi \text{ for all } k \text{ with } j < k \leq i\\
u,i \models \mathsf{N}\varphi &\Leftrightarrow u,i+1 \models \varphi\\
u,i \models \mathsf{N}^a\varphi &\Leftrightarrow u,i+a \models \varphi\\
u,i \models \mathsf{F}_{[a,b]}\varphi &\Leftrightarrow u,j \models \varphi \text{ for some } j \in [i+a,i+b]\\
u,i \models \mathsf{G}_{[a,b]}\varphi &\Leftrightarrow u,j \models \varphi \text{ for every } j \in [i+a,i+b]\\
u,i \models \varphi\,\mathsf{U}_{[a,b]}\,\psi &\Leftrightarrow \text{there is } j \in [i+a,i+b] \text{ s.t. } u,j \models \psi\\
&\qquad \text{and } u,k \models \varphi \text{ for all } k \text{ with } i \leq k < j
\end{aligned}
$$

Fig. 8. The satisfaction relation for temporal formulas.

for Metric Temporal Logic (MTL) with past and future-bounded temporal connectives. The monitoring algorithm is specified in a modular way: the monitor for a composite formula is built from the monitors of its immediate subformulas by applying the combinators of section IV.

We consider a language of typed temporal formulas. We write $\varphi : \mathsf{MTL}(A)$ to indicate that φ is a temporal formula that is meant to be interpreted over traces in A^ω. The type $\mathsf{MTL}(A)$ is defined inductively in Figure 7. We use the abbreviation $X_a = X_{[a,a]}$, where X can be any of the temporal connectives $\mathsf{P}, \mathsf{H}, \mathsf{S}, \mathsf{F}, \mathsf{G}, \mathsf{U}$. The *satisfaction relation* $\models \subseteq A^\omega \times \omega \times \mathsf{MTL}(A)$ is defined by induction as shown in Figure 8. For a trace $u \in A^\omega$, an index $i < \omega$ and a formula φ, we write $u,i \models \varphi$ to denote that the formula φ is *satisfied* in u at index i. The satisfaction of the formulas $\mathsf{P}_a\varphi$ and $\mathsf{H}_a\varphi$ can be expressed as follows:

$$
\begin{aligned}
u,i \models \mathsf{P}_a\varphi &\Leftrightarrow (i \geq a \text{ and } u,i-a \models \varphi), \text{ and}\\
u,i \models \mathsf{H}_a\varphi &\Leftrightarrow (i < a \text{ or } u,i-a \models \varphi),
\end{aligned}
$$

because $\mathsf{P}_a = \mathsf{P}_{[a,a]}$, $\mathsf{H}_a = \mathsf{H}_{[a,a]}$, and $[a,a] = \{a\}$. We say that the formulas $\varphi, \psi : \mathsf{MTL}(A)$ are *equivalent*, and we write $\varphi \equiv \psi$, if for every trace $u \in A^\omega$ and index $i < \omega$ we have $u,i \models \varphi$ iff $u,i \models \psi$. The equivalences of Figure 9 can be proved from the definition of the satisfaction relation.

We define the *interpretation* of φ w.r.t. the trace $u \in A^\omega$, denoted $I(u,\varphi) : \mathtt{Bool}^\omega$, as shown in Figure 10 (past-time fragment) and Figure 11 (future-time fragment). We omit H and G from the figures, as they are dual to P and F respectively.

Lemma 9 (Interpretation). Let $\varphi : \mathsf{MTL}(A)$ be a formula, $u \in A^\omega$ be a trace, and $i < \omega$ be a timepoint. Then, $I(u,\varphi)(i) = \mathtt{true}$ iff $u,i \models \varphi$.

$$
\begin{aligned}
\varphi\,\mathsf{S}\,\psi &\equiv \psi \vee (\varphi \wedge \mathsf{Y}(\varphi\,\mathsf{S}\,\psi)) && (1)\\
\mathsf{P}_{[a,\infty)}\varphi &\equiv \mathsf{P}_a\mathsf{P}\varphi && (2)\\
\varphi\,\mathsf{S}_{[a,\infty)}\,\psi &\equiv \mathsf{P}_a(\varphi\,\mathsf{S}\,\psi) \wedge \mathsf{H}_{[0,a-1]}\varphi, \text{ for } a \geq 1 && (3)\\
\mathsf{P}_{[a,b]}\varphi &\equiv \mathsf{P}_a\mathsf{P}_{[0,b-a]}\varphi && (4)\\
\varphi\,\mathsf{S}_{[a,b]}\,\psi &\equiv \mathsf{P}_a(\varphi\,\mathsf{S}_{[0,b-a]}\,\psi) \wedge \mathsf{H}_{[0,a-1]}\varphi, \text{ for } a \geq 1 && (5)\\
\mathsf{F}_{[0,b]}\varphi &\equiv \mathsf{N}^b\mathsf{P}_{[0,b]}\varphi && (6)\\
\mathsf{F}_{[a,b]}\varphi &\equiv \mathsf{N}^a\mathsf{F}_{[0,b-a]}\varphi \equiv \mathsf{N}^b\mathsf{P}_{[0,b-a]}\varphi && (7)\\
\varphi\,\mathsf{U}_{[a,b]}\,\psi &\equiv \mathsf{N}^a(\varphi\,\mathsf{U}_{[0,b-a]}\,\psi) \wedge \mathsf{G}_{[0,a-1]}\varphi, \text{ for } a \geq 1 && (8)
\end{aligned}
$$

Fig. 9. Equivalences between temporal formulas.

$$
\begin{aligned}
I(u,\mathsf{atomic}(p))(i) &= p(u(i))\\
I(u,\neg\varphi)(i) &= \neg I(u,\varphi)(i)\\
I(u,\varphi\wedge\psi)(i) &= I(u,\varphi)(i) \wedge I(u,\psi)(i)\\
I(u,\varphi\vee\psi)(i) &= I(u,\varphi)(i) \vee I(u,\psi)(i)\\
I(u,\mathsf{Y}\varphi)(0) &= \mathtt{false}\\
I(u,\mathsf{Y}\varphi)(i+1) &= I(u,\varphi)(i)\\
I(u,\mathsf{P}\varphi)(i) &= \mathit{fold}(x \text{ -> } x, \vee, I(u,\varphi)^{\leq i})\\
I(u,\varphi\,\mathsf{S}\,\psi)(i) &= \mathit{fold}((x,y) \text{ -> } y, \mathit{opSince}, w_i), \text{ where}\\
w_i &= \mathit{zip}(I(u,\varphi)^{\leq i}, I(u,\psi)^{\leq i}) : (\mathtt{Bool}^2)^*\\
\mathit{opSince} &= (z,(x,y)) \text{ -> } y \vee (x \wedge z)\\
I(u,\mathsf{P}_a\varphi)(i) &= \mathtt{false}, \text{ if } i < a\\
I(u,\mathsf{P}_a\varphi)(i+a) &= I(u,\varphi)(i)\\
I(u,\mathsf{P}_{[a,\infty)}\varphi) &= I(u,\mathsf{P}_a\mathsf{P}\varphi)\\
I(u,\varphi\,\mathsf{S}_{[a,\infty)}\,\psi) &= I(u,\mathsf{P}_a(\varphi\,\mathsf{S}\,\psi) \wedge \mathsf{H}_{[0,a-1]}\varphi)\\
I(u,\mathsf{P}_{[0,b]}\varphi)(i) &= (\mathit{fold}(\mathit{in}_\mathsf{P}, \mathit{op}_\mathsf{P}, I(u,\varphi)^{\leq i}) \neq \bot)\\
\mathit{in}_\mathsf{P} &: \mathtt{Bool} \to \{\bot\} \cup [0,b]\\
\mathit{in}_\mathsf{P}(x) &= \bot, \text{ if } x = \mathtt{false}\\
\mathit{in}_\mathsf{P}(x) &= 0, \text{ if } x = \mathtt{true}\\
\mathit{op}_\mathsf{P} &: (\{\bot\} \cup [0,b]) \times \mathtt{Bool} \to \{\bot\} \cup [0,b]\\
\mathit{op}_\mathsf{P}(\bot,x) &= \mathit{in}_\mathsf{P}(x)\\
\mathit{op}_\mathsf{P}(k,x) &= 0, \text{ if } x = \mathtt{true}\\
\mathit{op}_\mathsf{P}(k,x) &= k+1, \text{ if } k < b \text{ and } x = \mathtt{false}\\
\mathit{op}_\mathsf{P}(b,x) &= \bot, \text{ if } x = \mathtt{false}\\
I(u,\varphi\,\mathsf{S}_{[0,b]}\,\psi)(i) &= (\mathit{fold}(\mathit{in}_\mathsf{S}, \mathit{op}_\mathsf{S}, w_i) \neq \bot)\\
w_i &= \mathit{zip}(I(u,\varphi)^{\leq i}, I(u,\psi)^{\leq i}) : (\mathtt{Bool}^2)^*\\
\mathit{in}_\mathsf{S} &: \mathtt{Bool}^2 \to \{\bot\} \cup [0,b]\\
\mathit{in}_\mathsf{S}(x,y) &= \bot, \text{ if } y = \mathtt{false}\\
\mathit{in}_\mathsf{S}(x,y) &= 0, \text{ if } y = \mathtt{true}\\
\mathit{op} &: (\{\bot\} \cup [0,b]) \times \mathtt{Bool}^2 \to \{\bot\} \cup [0,b]\\
\mathit{op}_\mathsf{S}(\bot,(x,y)) &= \mathit{in}_\mathsf{S}(x,y)\\
\mathit{op}_\mathsf{S}(k,(x,y)) &= 0, \text{ if } y = \mathtt{true}\\
\mathit{op}_\mathsf{S}(k,(x,y)) &= k+1, \text{ if } k < b,\ y = \mathtt{false} \text{ and } x = \mathtt{true}\\
\mathit{op}_\mathsf{S}(k,(x,y)) &= \bot, \text{ if } k < b,\ y = \mathtt{false} \text{ and } x = \mathtt{false}\\
\mathit{op}_\mathsf{S}(b,(x,y)) &= \bot, \text{ if } y = \mathtt{false}\\
I(u,\mathsf{P}_{[a,b]}\varphi) &= I(u,\mathsf{P}_a\mathsf{P}_{[0,b-a]}\varphi)\\
I(u,\varphi\,\mathsf{S}_{[a,b]}\,\psi) &= I(u,\mathsf{P}_a(\varphi\,\mathsf{S}_{[0,b-a]}\,\psi) \wedge \mathsf{H}_{[0,a-1]}\varphi)
\end{aligned}
$$

Fig. 10. Interpretation function: past-time formulas.

Lemma 9 implies that the formulas φ, ψ are equivalent iff $I(u,\varphi) = I(u,\psi)$ for every trace $u \in A^\omega$.

An ***online monitor*** for a formula $\varphi : \mathsf{MTL}(A)$ is a transducer $G : \mathsf{SA}(A, \mathtt{Bool})$ s.t. the extension $f = \mathsf{ext}(\llbracket G \rrbracket) : \mathsf{KT}(A, \mathtt{Bool})$ of its denotation satisfies $f(u) = I(u,\varphi)$ for

$$
\begin{aligned}
I(u, \mathsf{N}\varphi)(i) &= I(u,\varphi)(i+1)\\
I(u, \mathsf{N}^a\varphi)(i) &= I(u,\varphi)(i+a)\\
I(u, \mathsf{F}_{[0,b]}\varphi)(i) &= I(u, \mathsf{N}^b\mathsf{P}_{[0,b]}\varphi)\\
I(u, \varphi\ \mathsf{U}_{[0,b]}\ \psi)(i) &= (\mathit{fold}(\mathit{in}_{\mathsf{U}}, \mathit{op}_{\mathsf{U}}, w) = \mathsf{yes})\\
w &= \mathit{zip}(I(u,\varphi)^{[i,i+b]}, I(u,\psi)^{[i,i+b]}) : (\mathtt{Bool}^2)^*\\
\mathit{in}_{\mathsf{U}} &: \mathtt{Bool}^2 \to S,\ \text{where } S = \{\mathsf{yes}, \mathsf{wait}, \mathsf{no}\}\\
\mathit{in}_{\mathsf{U}}(x,y) &= \mathsf{yes},\ \text{if } y = \mathtt{true}\\
\mathit{in}_{\mathsf{U}}(x,y) &= \mathsf{wait},\ \text{if } y = \mathtt{false} \text{ and } x = \mathtt{true}\\
\mathit{in}_{\mathsf{U}}(x,y) &= \mathsf{no},\ \text{if } y = \mathtt{false} \text{ and } x = \mathtt{false}\\
\mathit{op}_{\mathsf{U}} &: S \times \mathtt{Bool}^2 \to S\\
\mathit{op}_{\mathsf{U}}(s,(x,y)) &= s,\ \text{if } s \in \{\mathsf{yes}, \mathsf{no}\}\\
\mathit{op}_{\mathsf{U}}(\mathsf{wait},(x,y)) &= \mathit{in}(x,y)\\
I(u, \mathsf{F}_{[a,b]}\varphi) &= I(u, \mathsf{N}^a\mathsf{F}_{[0,b-a]}\varphi) = I(u, \mathsf{N}^b\mathsf{P}_{[0,b-a]}\varphi)\\
I(u, \varphi\ \mathsf{U}_{[a,b]}\ \psi) &= I(u, \mathsf{N}^a(\varphi\ \mathsf{U}_{[0,b-a]}\ \psi) \wedge \mathsf{G}_{[0,a-1]}\varphi)
\end{aligned}
$$

Fig. 11. Interpretation function: bounded future-time formulas.

every trace $u \in A^\omega$. We describe a construction for efficient online MTL monitors in Figure 12. We omit the connectives $\wedge$, H and G because they are dual to $\vee$, P and F respectively.

Proposition 10 (Online Monitor). Let $\varphi : \mathsf{MTL}(A)$. The transducer $\mathtt{TL}(\varphi) : \mathsf{SA}(A, \mathtt{Bool})$, defined in Figure 12, has bounded lead/lag and is an online monitor for φ.

Proof. The proof is by induction on φ. It is immediate from the construction of Figure 12 that $\mathtt{TL}(\varphi)$ has bounded lead/lag. We focus on some representative cases and we leave the rest to the reader. Consider an arbitrary trace $u \in A^\omega$. Case $\mathsf{P}\varphi$:

$$
\begin{aligned}
w &= \mathsf{ext}(\llbracket \mathtt{TL}(\mathsf{P}\varphi) \rrbracket)(u)\\
&= (\mathsf{ext}(\llbracket \mathtt{TL}(\varphi) \rrbracket) \gg \mathsf{ext}(\mathsf{aggr}(x \dashrightarrow x, \vee)))(u)\\
&= \mathsf{ext}(\mathsf{aggr}(x \dashrightarrow x, \vee))(\mathsf{ext}(\llbracket \mathtt{TL}(\varphi) \rrbracket)(u))\\
&= \mathsf{ext}(\mathsf{aggr}(x \dashrightarrow x, \vee))(I(u,\varphi))
\end{aligned}
$$

and therefore $w(i) = \mathit{fold}(x \dashrightarrow x, \vee, v^{\leq i})$. This implies that $w = I(u, \mathsf{P}\varphi)$. For the case $\mathsf{P}_a\varphi$, we obtain similarly that

$$
w = \mathsf{ext}(\llbracket \mathtt{TL}(\mathsf{P}_a\varphi) \rrbracket)(u) = \mathsf{ext}(\mathsf{emit}(a, \mathtt{false}))(I(u,\varphi)).
$$

It follows that $w(i) = \mathtt{false}$ if $i < a$, and $w(i+a) = I(u,\varphi)(i)$ for every $i < \omega$. So, $w = I(u, \mathsf{P}_a\varphi)$. As before, for the case $\mathsf{N}^a\varphi$ we get that

$$
w = \mathsf{ext}(\llbracket \mathtt{TL}(\mathsf{N}^a\varphi) \rrbracket)(u) = \mathsf{ext}(\mathsf{ignore}(a))(I(u,\varphi)).
$$

This means that $w(i) = I(u,\varphi)(i+a)$ for every $i < \omega$, and therefore $w = I(u, \mathsf{N}^a\varphi)$. The rest of the cases are similar. □

We describe now our ***implementation*** and present an ***experimental evaluation***. From Theorem 8 we know that the online monitors of Figure 12 require a constant amount of space (in the size of the input stream). In order to examine empirically this efficiency guarantee, we have implemented the combinators of Figure 3 as an embedded domain-specific language in Rust and we have used it to specify the MTL monitors. We compare our implementation against the state-of-the-art tools MonPoly [35] (OCaml), StreamLAB [36] (Rust), Reelay [38] (C++) and Aerial [49] (OCaml). All tools are implemented in native code (i.e., no bytecode).

$$
\frac{p : A \to \mathtt{Bool}}{\mathtt{TL}(\mathsf{atomic}(p)) = \mathtt{map}(p) : \mathsf{SA}(A, \mathtt{Bool}, 0)}
$$

$$
\frac{\mathtt{TL}(\varphi) : \mathsf{SA}(A, \mathtt{Bool}, [m,n])}{\mathtt{TL}(\neg\varphi) = \mathtt{TL}(\varphi) \gg \mathtt{map}(\neg) : \mathsf{SA}(A, \mathtt{Bool}, [m,n])}
$$

$$
\frac{\mathtt{TL}(\varphi) : \mathsf{SA}(A, \mathtt{Bool}, [m,n]) \qquad \mathtt{TL}(\psi) : \mathsf{SA}(A, \mathtt{Bool}, [o,p])}{\begin{aligned}\mathtt{TL}(\varphi \vee \psi) = \mathtt{par}(\mathtt{TL}(\varphi), \mathtt{TL}(\psi)) \gg \mathtt{map}(\vee)\\ : \mathsf{SA}(A, \mathtt{Bool}, [\min(m,o), \min(n,p)])\end{aligned}}
$$

$$
\frac{\mathtt{TL}(\varphi) : \mathsf{SA}(A, \mathtt{Bool}, [m,n])}{\begin{aligned}
\mathtt{TL}(\mathsf{Y}\varphi) &= \mathtt{TL}(\varphi) \gg \mathtt{emit}(1, \mathtt{false}) : \mathsf{SA}(A, \mathtt{Bool}, [m+1, n+1])\\
\mathtt{TL}(\mathsf{P}\varphi) &= \mathtt{TL}(\varphi) \gg \mathtt{aggr}(x \dashrightarrow x, \vee) : \mathsf{SA}(A, \mathtt{Bool}, [m,n])\\
\mathtt{TL}(\mathsf{P}_a\varphi) &= \mathtt{TL}(\varphi) \gg \mathtt{emit}(a, \mathtt{false}) : \mathsf{SA}(A, \mathtt{Bool}, [m+a, n+a])\\
\mathtt{TL}(\mathsf{P}_{[a,\infty)}\varphi) &= \mathtt{TL}(\mathsf{P}_a\mathsf{P}\varphi) : \mathsf{SA}(A, \mathtt{Bool}, [m+a, n+a])\\
\mathtt{TL}(\mathsf{P}_{[0,b]}\varphi) &= \mathtt{TL}(\varphi) \gg \mathtt{aggr}(\mathit{in}_{\mathsf{P}}, \mathit{op}_{\mathsf{P}}, x \dashrightarrow x \neq \bot)\\
&\quad : \mathsf{SA}(A, \mathtt{Bool}, [m,n])\\
\mathtt{TL}(\mathsf{P}_{[a,b]}\varphi) &= \mathtt{TL}(\mathsf{P}_a\mathsf{P}_{[0,b-a]}\varphi) : \mathsf{SA}(A, \mathtt{Bool}, [m+a, n+a])
\end{aligned}}
$$

$$
\frac{\mathtt{TL}(\varphi) : \mathsf{SA}(A, \mathtt{Bool}, [m,n]) \qquad \mathtt{TL}(\psi) : \mathsf{SA}(A, \mathtt{Bool}, [o,p])}{\begin{aligned}
\mathtt{TL}(\varphi\ \mathsf{S}\ \psi) &= \mathtt{par}(\mathtt{TL}(\varphi), \mathtt{TL}(\psi)) \gg \mathtt{aggr}((x,y) \dashrightarrow y, \mathit{opSince})\\
&\quad : \mathsf{SA}(A, \mathtt{Bool}, [\min(m,o), \min(n,p)])\\
\mathtt{TL}(\varphi\ \mathsf{S}_{[a,\infty)}\ \psi) &= \mathtt{TL}(\mathsf{P}_a(\varphi\ \mathsf{S}\ \psi) \wedge \mathsf{H}_{[0,a-1]}\varphi)\\
&\quad : \mathsf{SA}(A, \mathtt{Bool}, [\min(m,o)+a, \min(n,p)+a])\\
\mathtt{TL}(\varphi\ \mathsf{S}_{[0,b]}\ \psi) &= \mathtt{par}(\mathtt{TL}(\varphi), \mathtt{TL}(\psi)) \gg \mathtt{aggr}(\mathit{in}_{\mathsf{S}}, \mathit{op}_{\mathsf{S}}, x \dashrightarrow x \neq \bot)\\
&\quad : \mathsf{SA}(A, \mathtt{Bool}, [\min(m,o), \min(n,p)])\\
\mathtt{TL}(\varphi\ \mathsf{S}_{[a,b]}\ \psi) &= \mathtt{TL}(\mathsf{P}_a(\varphi\ \mathsf{S}_{[0,b-a]}\ \psi) \wedge \mathsf{H}_{[0,a-1]}\varphi)\\
&\quad : \mathsf{SA}(A, \mathtt{Bool}, [\min(m,o)+a, \min(n,p)+a])
\end{aligned}}
$$

$$
\frac{\mathtt{TL}(\varphi) : \mathsf{SA}(A, \mathtt{Bool}, [m,n])}{\begin{aligned}
\mathtt{TL}(\mathsf{N}\varphi) &= \mathtt{TL}(\varphi) \gg \mathtt{ignore}(1) : \mathsf{SA}(A, \mathtt{Bool}, [m-1, n-1])\\
\mathtt{TL}(\mathsf{N}^a\varphi) &= \mathtt{TL}(\varphi) \gg \mathtt{ignore}(a) : \mathsf{SA}(A, \mathtt{Bool}, [m-a, n-a])\\
\mathtt{TL}(\mathsf{F}_{[a,b]}\varphi) &= \mathtt{TL}(\mathsf{N}^b\mathsf{P}_{[0,b-a]}\varphi) : \mathsf{SA}(A, \mathtt{Bool}, [m-b, n])
\end{aligned}}
$$

$$
\frac{\mathtt{TL}(\varphi) : \mathsf{SA}(A, \mathtt{Bool}, [m,n]) \qquad \mathtt{TL}(\psi) : \mathsf{SA}(A, \mathtt{Bool}, [o,p])}{\begin{aligned}
\mathtt{TL}(\varphi\ \mathsf{U}_{[0,b]}\ \psi) &= \mathtt{par}(\mathtt{TL}(\varphi), \mathtt{TL}(\psi)) \gg \mathtt{wnd}(b+1, \mathit{in}_{\mathsf{U}}, \mathit{op}_{\mathsf{U}})\\
&\quad : \mathsf{SA}(A, \mathtt{Bool}, [\min(m,o)-b, \min(n,p)])\\
\mathtt{TL}(\varphi\ \mathsf{U}_{[a,b]}\ \psi) &= \mathtt{TL}(\mathsf{N}^a(\varphi\ \mathsf{U}_{[0,b-a]}\ \psi) \wedge \mathsf{G}_{[0,a-1]}\varphi)\\
&\quad : \mathsf{SA}(A, \mathtt{Bool}, [\min(m,o)-b, \min(n,p)])
\end{aligned}}
$$

Fig. 12. Online monitor for bounded-future MTL formulas.

In Figure 13, we compare our approach (DSL) with MonPoly, StreamLAB, Reelay and Aerial on two sets of formulas, both of which contain a single past-time temporal connective (bounded or unbounded). The formulas for the top plot (P1 to P17) are: Y, P, $\mathsf{P}_{[0,1]}$, $\mathsf{P}_{[0,10]}$, $\mathsf{P}_{[0,100]}$, $\mathsf{P}_{[0,1000]}$, $\mathsf{P}_{[0,10000]}$, $\mathsf{P}_{[1,\infty)}$, $\mathsf{P}_{[10,\infty)}$, $\mathsf{P}_{[100,\infty)}$, $\mathsf{P}_{[1000,\infty)}$, $\mathsf{P}_{[10000,\infty)}$, $\mathsf{P}_{[1,2]}$, $\mathsf{P}_{[10,20]}$, $\mathsf{P}_{[100,200]}$, $\mathsf{P}_{[1000,2000]}$, $\mathsf{P}_{[10000,20000]}$. The formulas for the bottom plot (P1 to P16) are: S, $\mathsf{S}_{[0,1]}$, $\mathsf{S}_{[0,10]}$, $\mathsf{S}_{[0,100]}$, $\mathsf{S}_{[0,1000]}$, $\mathsf{S}_{[0,10000]}$, $\mathsf{S}_{[1,\infty)}$, $\mathsf{S}_{[10,\infty)}$, $\mathsf{S}_{[100,\infty)}$, $\mathsf{S}_{[1000,\infty)}$, $\mathsf{S}_{[10000,\infty)}$, $\mathsf{S}_{[1,2]}$, $\mathsf{S}_{[10,20]}$, $\mathsf{S}_{[100,200]}$, $\mathsf{S}_{[1000,2000]}$, $\mathsf{S}_{[10000,20000]}$. We observe that DSL is 10-200 times faster than MonPoly and StreamLAB, and 80-1300 times faster than Reelay. The performance of DSL, StreamLAB, Reelay and MonPoly is not affected by the size of intervals. Aerial performs well when the size of intervals is small, but its throughput decreases sharply as the size of the intervals grows. Aerial's space and time-per-element complexity is linear in the sum of the numeric constants and the formula size [50].

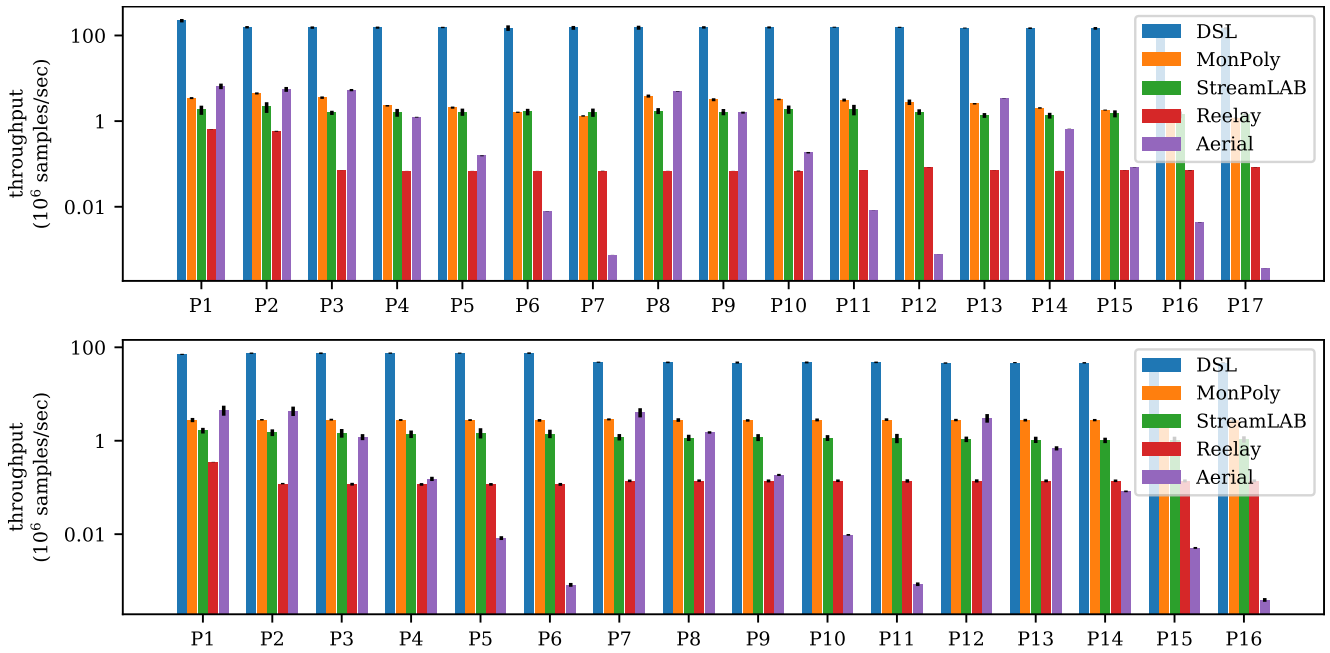

Fig. 13. Past-time microbenchmark: DSL, Reelay, MonPoly, Aerial.

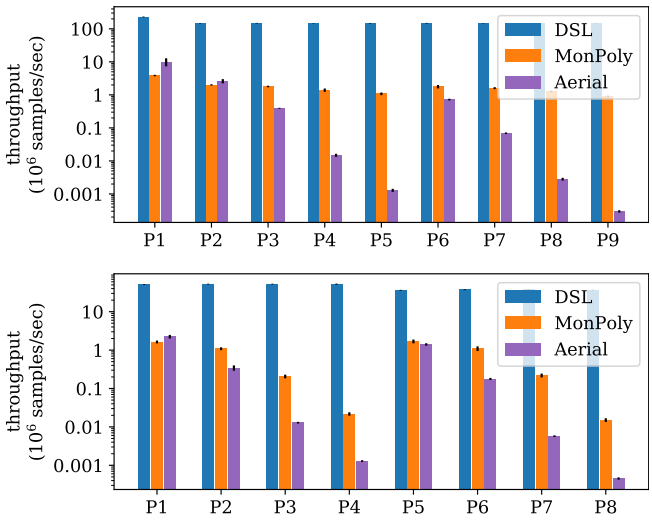

Fig. 14. Future-time microbenchmark: DSL, MonPoly, Aerial.

In Figure 14, we compare our approach (DSL) with MonPoly and Aerial on two sets of formulas, both of which contain a single (bounded) future-time temporal connective. The formulas for the top plot (P1 to P9) are: N, $\mathsf{F}_{[0,10]}$, $\mathsf{F}_{[0,100]}$, $\mathsf{F}_{[0,1000]}$, $\mathsf{F}_{[0,10000]}$, $\mathsf{F}_{[10,20]}$, $\mathsf{F}_{[100,200]}$, $\mathsf{F}_{[1000,2000]}$, $\mathsf{F}_{[10000,20000]}$. The formulas for the bottom plot (P1 to P8) are: $\mathsf{U}_{[0,10]}$, $\mathsf{U}_{[0,100]}$, $\mathsf{U}_{[0,1000]}$, $\mathsf{U}_{[0,10000]}$, $\mathsf{U}_{[10,20]}$, $\mathsf{U}_{[100,200]}$, $\mathsf{U}_{[1000,2000]}$, $\mathsf{U}_{[10000,20000]}$. We observe that DSL is 30-1500 times faster than MonPoly. As in the past-time microbenchmark, Aerial's performance depends on the constants that appear in the formulas.

Finally, we use the recently proposed Timescales benchmark [39] to compare the four tools (DSL, MonPoly, Reelay, Aerial) on a set of realistic properties for monitoring. The results

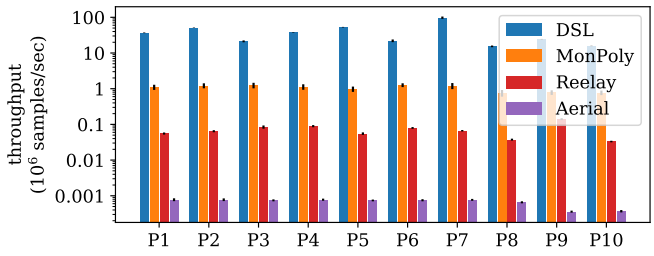

Fig. 15. Timescales benchm.: DSL, MonPoly, Reelay, Aerial.

are shown in Figure 15. DSL is 10-80 times faster than MonPoly, 50-350 times faster than Reelay and 20000-140000 times faster than Aerial. We have already discussed that Aerial performs poorly when large constants occur in the formulas.

All of our experiments were executed on a laptop with an Intel Core i7 10610U CPU clocked at 2.30 GHz and 16 GB of memory. Each throughput value that we report in Figure 13, 14 and 15 is the mean and standard deviation of 50 iterations of the experiment.

Overall, we observe that our Rust implementation of the typed framework of lead/lag-bounded transformations has performance that is competitive compared to state-of-the-art tools. We are not claiming that our implementation is better than the tools we compare against, since MonPoly and Reelay consider more expressive temporal formalisms than our variant of MTL. We leave for future work the development of a robust tool for our framework and a more extensive experimental evaluation.

VI. Case Study: Peak Detection in ECG Signal

In this section we consider the monitoring of cardiac (heart) signal for a patient. This signal is called an electrocardiogram (ECG). We will focus on the problem of *peak detection* in the ECG, which corresponds to the detection of the heartbeat. This

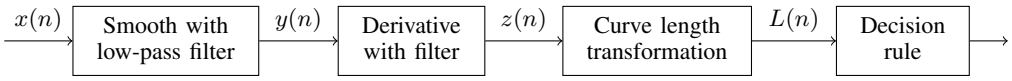

Fig. 16. The high-level structure of an online algorithm for peak detection in the ECG signal.

$$
\begin{aligned}
\texttt{smooth} &= \texttt{FIR}([0.1, 0.2, 0.4, 0.2, 0.1]) : \mathsf{SA}(\mathsf{V}, \mathsf{V}, [-n, 0]) \\
\texttt{FIR}([c_{-n}, \ldots, c_n]) &= \texttt{emit}(n, 0) \gg \texttt{wnd}(2n+1) \gg \\
&\quad \texttt{map}((x_{-n}, \ldots, x_n) \texttt{ -> } c_{-n}x_{-n} + \cdots + c_n x_n) \\
\texttt{slope} &= \texttt{FIR}([-0.5, 0, 0.5]) : \mathsf{SA}(\mathsf{V}, \mathsf{V}, [-1, 0]) \\
\texttt{length} &= \texttt{emit}(w, 0) \gg \\
&\quad \texttt{map}(x \texttt{ -> } sqrt(1 + x^2)) \gg \texttt{wnd}(2w+1) \gg \\
&\quad \texttt{map}((x_{-w}, \ldots, x_w) \texttt{ -> } \textstyle\sum_{i=-w}^{w} x_i) \\
\texttt{detect} &= (\mathsf{St}, \mathtt{A}, \varepsilon, \delta, \mathsf{out}) : \mathsf{SA}(\mathsf{VL}, \mathtt{Bool}, [-2w, 0]) \\
\mathsf{St} &= \{\mathtt{A}\} \cup \{\mathtt{C}(i) \mid i \in [1, 2w]\} \\
&\quad \cup \{\mathtt{B}(i, m, j) \mid i \in [1, 2w], m \in \mathsf{V}, j \in [1, i]\} \\
\delta(\mathtt{A}, (v, \ell)) &= \mathtt{A}, \text{ if } \ell < \mathsf{Threshold} \\
\delta(\mathtt{A}, (v, \ell)) &= \mathtt{B}(1, v, 1), \text{ if } \ell \geq \mathsf{Threshold} \\
\delta(\mathtt{B}(i, m, j), (v, \ell)) &= \mathtt{B}(i+1, m, j), \text{ if } v \leq m \text{ and } i < 2w \\
\delta(\mathtt{B}(i, m, j), (v, \ell)) &= \mathtt{B}(i+1, v, i+1), \text{ if } v > m \text{ and } i < 2w \\
\delta(\mathtt{B}(i, m, j), (v, \ell)) &= \mathtt{C}(j), \text{ if } i = 2w \\
\delta(\mathtt{C}(i), (v, \ell)) &= \mathtt{C}(i-1), \text{ if } i > 1 \\
\delta(\mathtt{C}(i), (v, \ell)) &= \mathtt{A}, \text{ if } i = 1 \\
\mathsf{out}(\mathtt{A}, (v, \ell)) &= \mathtt{false}, \text{ if } \ell < \mathsf{Threshold} \\
\mathsf{out}(\mathtt{A}, (v, \ell)) &= \varepsilon, \text{ if } \ell \geq \mathsf{Threshold} \\
\mathsf{out}(\mathtt{B}(i, m, j), (v, \ell)) &= \varepsilon, \text{ if } i < 2w \\
\mathsf{out}(\mathtt{B}(i, m, j), (v, \ell)) &= \mathtt{false}^{j-1} \cdot \mathtt{true} \cdot \mathtt{false}^{2w+1-j}, \text{ if } i = 2w \\
\mathsf{out}(\mathtt{C}(i), (v, \ell)) &= \mathtt{false} \\
\texttt{preprocess} &= \texttt{par}(\texttt{id}, \texttt{smooth} \gg \texttt{slope} \gg \texttt{length}) \\
\texttt{main} &= \texttt{preprocess} \gg \texttt{detect} : \mathsf{SA}(\mathsf{V}, \mathtt{Bool})
\end{aligned}
$$

Fig. 17. Peak detection for cardiac (ECG) signal.

problem is one of the most widely studied detection problems in the area of biomedical engineering [29], [30], [31], [32], as it forms the basis of many analyses over cardiac data. We will see that our domain-specific language can be used to easily specify the detection algorithm. Similar in spirit are the works [33] and [34], where the problem of arrhythmia detection is considered.

A simple algorithm for detecting the peaks consists of four stages: (1) smoothing the signal to eliminate high-frequency noise, (2) taking the derivative of the smoothed signal to calculate the slope, (3) computing a nonlinear curve length transformation, and (4) detecting the peaks using both the raw measurements and the curve lengths. This algorithm is described in [32] and its implementation is found in [51].

We show how to prototype this algorithm using our typed framework of ℓ-bounded transducers (section III) and their combinators (section IV). In Figure 17 we see that the stages for smoothing and slope calculation can be expressed with the `FIR` combinator (recall Example 7), which in turn is a pipeline of `emit`, `wnd`, `map`. For the curve length transformation, we consider a window of size $w = 65\,\text{msec} \cdot F$ (where F is the sampling frequency in Hz), centered at each timepoint. That is, w is the number of samples that correspond to 65 msec of the signal. The transducer `length` : $\mathsf{SA}(\mathsf{V}, \mathsf{L}, [-w, 0]$ describes this transformation. Notice that is has a maximum lag of w (lead $-w$), because it needs to see w samples ahead in order to calculate the curve length. The decision rule is given by a customized transducer `detect` : $\mathsf{SA}(\mathsf{VL}, \mathtt{Bool}, [-2w, 0])$, which takes as input the original signal and the signal of curve length values. Intuitively, `detect` alternates between three modes: (1) mode `A`, during which the transducer searches for a curve length value exceeding a threshold, (2) mode `B`, during which the transducer finds the maximum of the original signal from the point t that the threshold has crossed until $t + 2w$ (window of size 130 msec), and (3) mode `C`, which ensures that 130 msec pass from the detected peak until the transducer can enter the search mode (i.e., mode `A`) again. The lead interval for `detect` is $[-2w, 0]$, which is established by annotating the transducer with the lead labeling $\langle \lambda, \lambda \rangle$, where: $\lambda(\mathtt{A}) = 0$, $\lambda(\mathtt{B}(i, m, j)) = -i$, and $\lambda(\mathtt{C}) = 0$. It suffices to observe that $\lambda(\delta(s, x)) = \lambda(x) + |\mathsf{out}(s, x)| - 1$ for every $s \in \mathsf{St}$ and $x \in \mathsf{VL}$.

Finally, we use the ECG peak detector (`main` in Fig. 17) to define a transducer for monitoring the heart rate. Let F be the sampling frequency of the signal.

$$
\begin{aligned}
p &= \mathsf{atomic}(x \texttt{ -> } x) : \mathsf{MTL}(\mathtt{Bool}) \\
\bar{p} &= \mathsf{atomic}(x \texttt{ -> } \neg x) : \mathsf{MTL}(\mathtt{Bool}) \\
\varphi &= (p \rightarrow \mathsf{H}_{[1,\, 250\text{msec} \cdot F]} \bar{p}) : \mathsf{MTL}(\mathtt{Bool}) \\
\psi &= \mathsf{F}_{[1,\, 1000\text{msec} \cdot F]} p : \mathsf{MTL}(\mathtt{Bool}) \\
\texttt{mon} &= \texttt{par}(\texttt{MTL}(\varphi), \texttt{MTL}(\psi)) : \mathsf{SA}(\mathtt{Bool}, \mathtt{Bool}^2) \\
\texttt{top} &= \texttt{main} \gg \texttt{mon} : \mathsf{SA}(\mathsf{V}, \mathtt{Bool}^2)
\end{aligned}
$$

The transducer `top` checks whether the heart rate is too slow or too fast: φ checks that no interval is shorter than 250 ms (i.e., rate faster than 240 bpm), and ψ checks that no interval is longer than 1000 ms (i.e., rate slower than 60 bpm).

VII. Discussion

In this section, we discuss the relationship between our framework of ℓ-bounded transductions/transducers and the established synchronous language Lustre and its descendants.

In Lustre [15] a variable x represents a sequence of values (i.e., a signal or stream) and has a clock c associated with it, which specifies the time instants (over some discrete time domain) for which x is defined. These clocks are used to type-check expressions. For example, the expression $x + y$ is allowed only when x and y have the same underlying clock c. These clock compatibility checks ensure that combined signals are perfectly synchronized. This gives rise to a guarantee of efficient computation: no amount of buffering is required for storing elements. Lucid Synchrone [26], [27] extends Lustre with higher-order features, provides inference of the clock types [52], [53], and supports hierarchical state machines [54].

Other synchronous formalisms such as SDF [19] and CSDF [20] employ a notion of rate for the consumption and production of elements (e.g., a computational element produces 3

output elements for every 2 input elements that it consumes). SDF and CSDF also provide guarantees of efficiency, but they do so by inserting finite buffers between computing elements. This provides flexibility in the specification and relegates the issue of finding the appropriate buffer sizes to the compiler.

The work on n-synchronous Kahn networks [55] studies a relaxed model of synchrony for a Lustre-based clocked language that allows communication over bounded buffers. This setting requires reasoning about ultimately periodic clocks, which can be represented canonically in the form $u(v)^\omega$, where $u, v \in \{0,1\}^*$ and $(v)^\omega$ denotes the infinite repetition of v. The symbol 1 (resp., 0) indicates the presence (resp., absence) of a value at a given timepoint of the base clock.

Clock envelopes are introduced in [56] as an abstraction of (not necessarily periodic) clocks. They denote sets of clocks that almost have a fixed period except for a bounded amount of allowable jitter. Lucy-n [28] is the implementation of the n-synchronous model of [55], and it uses a variant of the clock abstraction of [56] for clock inference. Lucy-n extends Lustre with a buffering construct, which serves as a placeholder for a bounded communication buffer whose size is determined by the compiler. An extension of Lucy-n with a delay operator is considered in [57]. A more precise clock inference algorithm for Lucy-n is described in [58].

Observe in the proof of Theorem 8 that the transducer $\texttt{par}(\texttt{f}, \texttt{g})$ requires a buffer to accommodate the skew between the output of `f` and the output of `g`. The use of this buffer is similar in spirit to the bounded buffer insertion that is considered in SDF [19], CSDF [20], and Lucy-n [28]. Identifying a precise correspondence between our transducer model and Lucy-n is an interesting direction for future work. The main question is to clarify the relationship between the lead/lag types that we introduce here and the clocks of Lucy-n. A relevant concept is the domain/rate of Quantitative Regular Expressions [59], [14], [60] and related formalisms [61], [62], [63], [64], which is a kind of regular expression that specifies when output is emitted.

We conclude this section with the remark that synchronous languages such as Lustre [15] have been used to program monitors that run alongside synchronous systems. Such monitors check for safety violations and are called synchronous observers [65]. Relevant to this is the language Lutin [66] for expressing temporal properties, as well as the translation of regular expressions (which capture safety properties) to synchronous networks [67]. An interesting direction for future work is to provide a translation from our framework to Heptagon/BZR [68] in order to obtain low-level C code for deployment on embedded systems.

VIII. Conclusion

We have introduced the framework of lead/lag-bounded signal transformations, which relaxes the causality requirement and allows a bounded amount of lookahead into the future. Such transformations can be implemented with a bounded amount of lag. We classify the transformations with a type discipline that records the bounds on the allowable lead and/or lag of the computation. Our framework gives rise to a domain-specific language (DSL) for signal monitoring with a key efficiency guarantee: every monitor defined by the DSL can be executed using a bounded amount of space and processing time per input sample. We validate the usefulness of our proposal with two significant case studies: (1) the prototyping of an efficient monitor for MTL for past-time and bounded future-time temporal connectives, and (2) the specification of a complex algorithm for peak detection and heart rate monitoring over ECG signal. We have also implemented the proposed DSL and we have compared experimentally our MTL monitor against state-of-the-art tools.

Acknowledgment

The authors would like to thank the anonymous reviewers for their constructive comments and suggestions.

References

[1] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-Time Systems*, vol. 2, no. 4, pp. 255–299, 1990.

[2] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Proc. FORMATS/FTRTFT*. Springer, 2004, pp. 152–166.

[3] K. Havelund and G. Roşu, "Efficient monitoring of safety properties," *International Journal on Software Tools for Technology Transfer*, vol. 6, no. 2, pp. 158–173, 2004.

[4] K. Havelund, D. Peled, and D. Ulus, "First-order temporal logic monitoring with BDDs," *Formal Methods in System Design*, pp. 1–21, 2019.

[5] D. Basin, F. Klaedtke, S. Müller, and B. Pfitzmann, "Runtime monitoring of metric first-order temporal properties," in *Proc. FSTTCS*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2008, pp. 49–60.

[6] D. Ulus, T. Ferrère, E. Asarin, and O. Maler, "Online timed pattern matching using derivatives," in *Proc. TACAS*. Springer, 2016, pp. 736–751.

[7] Y. Annapureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan, "S-taliro: A tool for temporal logic falsification for hybrid systems," in *Proc. TACAS*. Springer, 2011, pp. 254–257.

[8] A. Donzé, O. Maler, E. Bartocci, D. Nickovic, R. Grosu, and S. Smolka, "On temporal logic and signal processing," in *Proc. ATVA*. Springer, 2012, pp. 92–106.

[9] A. Dokhanchi, B. Hoxha, and G. Fainekos, "On-line monitoring for temporal logic robustness," in *Proc. RV*. Springer, 2014, pp. 231–246.

[10] J. V. Deshmukh, A. Donzé, S. Ghosh, X. Jin, G. Juniwal, and S. A. Seshia, "Robust online monitoring of signal temporal logic," *Formal Methods in System Design*, vol. 51, no. 1, pp. 5–30, 2017.

[11] S. Jakšić, E. Bartocci, R. Grosu, T. Nguyen, and D. Ničković, "Quantitative monitoring of STL with edit distance," *Formal Methods in System Design*, vol. 53, no. 1, pp. 83–112, 2018.

[12] S. Jakšić, E. Bartocci, R. Grosu, and D. Ničković, "An algebraic framework for runtime verification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2233–2243, 2018.

[13] B. D'Angelo, S. Sankaranarayanan, C. Sanchez, W. Robinson, B. Finkbeiner, H. B. Sipma, S. Mehrotra, and Z. Manna, "LOLA: Runtime monitoring of synchronous systems," in *Proc. TIME*. IEEE, 2005, pp. 166–174.

[14] K. Mamouras, M. Raghothaman, R. Alur, Z. G. Ives, and S. Khanna, "StreamQRE: Modular specification and efficient evaluation of quantitative queries over streaming data," in *Proc. PLDI*. ACM, 2017, pp. 693–708.

[15] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, "The synchronous data flow programming language LUSTRE," *Proceedings of the IEEE*, vol. 79, no. 9, pp. 1305–1320, 1991.

[16] G. Berry and G. Gonthier, "The Esterel synchronous programming language: Design, semantics, implementation," *Science of Computer Programming*, vol. 19, no. 2, pp. 87–152, 1992.

[17] A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. L. Guernic, and R. de Simone, "The synchronous languages 12 years later," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 64–83, 2003.

[18] T. Bourke and M. Pouzet, "Zélus: A synchronous language with ODEs," in *Proc. HSCC*. ACM, 2013, pp. 113–118.

[19] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987.

[20] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete, "Cyclo-static dataflow," *IEEE Transactions on Signal Processing*, vol. 44, no. 2, pp. 397–408, 1996.

[21] C. Elliott and P. Hudak, "Functional reactive animation," in *Proc. ICFP*. ACM, 1997, pp. 263–273.

[22] P. Hudak, A. Courtney, H. Nilsson, and J. Peterson, "Arrows, robots, and functional reactive programming," in *AFP*. Springer, 2003, pp. 159–187.

[23] C. M. Elliott, "Push-pull functional reactive programming," in *Proc. Haskell*. ACM, 2009, pp. 25–36.

[24] G. Kahn, "The semantics of a simple language for parallel programming," *Information Processing*, vol. 74, pp. 471–475, 1974.

[25] P. Caspi, "Clocks in dataflow languages," *Theoretical Computer Science*, vol. 94, no. 1, pp. 125–140, 1992.

[26] M. Pouzet, *Lucid Synchrone, Version 3. Tutorial and Reference Manual*, Université Paris-Sud, LRI, 2006.

[27] P. Caspi, G. Hamon, and M. Pouzet, "Synchronous functional programming with Lucid Synchrone," in *Modeling and Verification of Real-Time Systems: Formalisms and Software Tools*. Wiley, 2008, pp. 207–247.

[28] L. Mandel, F. Plateau, and M. Pouzet, "Lucy-n: a n-synchronous extension of Lustre," in *Proc. MPC*. Springer, 2010, pp. 288–309.

[29] J. Pan and W. J. Tompkins, "A real-time QRS detection algorithm," *IEEE Transactions on Biomedical Engineering*, vol. BME-32, no. 3, pp. 230–236, 1985.

[30] P. S. Hamilton and W. J. Tompkins, "Quantitative investigation of QRS detection rules using the MIT/BIH arrhythmia database," *IEEE Transactions on Biomedical Engineering*, vol. BME-33, no. 12, pp. 1157–1165, 1986.

[31] B.-U. Köhler, C. Hennig, and R. Orglmeister, "The principles of software QRS detection," *IEEE Engineering in Medicine and Biology Magazine*, vol. 21, no. 1, pp. 42–57, 2002.

[32] W. Zong, G. B. Moody, and D. Jiang, "A robust open-source algorithm to detect onset and duration of QRS complexes," in *Proc. Computers in Cardiology*. IEEE, 2003, pp. 737–740.

[33] H. Abbas, R. Alur, K. Mamouras, R. Mangharam, and A. Rodionova, "Real-time decision policies with predictable performance," *Proceedings of the IEEE, Special Issue on Design Automation for Cyber-Physical Systems*, vol. 106, no. 9, pp. 1593–1615, 2018.

[34] H. Abbas, A. Rodionova, K. Mamouras, E. Bartocci, S. A. Smolka, and R. Grosu, "Quantitative regular expressions for arrhythmia detection," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 16, no. 5, pp. 1586–1597, 2019.

[35] D. Basin, F. Klaedtke, and E. Zalinescu, "The MonPoly monitoring tool," in *Proc. RV-CuBES*. EasyChair, 2017.

[36] P. Faymonville, B. Finkbeiner, M. Schledjewski, M. Schwenger, M. Stenger, L. Tentrup, and H. Torfah, "StreamLAB: Stream-based monitoring of cyber-physical systems," in *Proc. CAV*. Springer, 2019, pp. 421–431.

[37] D. Basin, S. Krstic, and D. Traytel, "AERIAL: Almost event-rate independent algorithms for monitoring metric regular properties," in *Proc. RV-CuBES*. EasyChair, 2017.

[38] D. Ulus, "The Reelay monitoring tool," https://doganulus.github.io/reelay/, 2020, [Online; accessed April 17, 2020].

[39] ——, "Timescales: A benchmark generator for MTL monitoring tools," in *Proc. RV*. Springer, 2019, pp. 402–412.

[40] S. MacLane, *Categories for the Working Mathematician*. Springer, 1972.

[41] E. Asarin, P. Caspi, and O. Maler, "Timed regular expressions," *Journal of the ACM*, vol. 49, no. 2, pp. 172–206, 2002.

[42] R. Alur, K. Mamouras, C. Stanford, and V. Tannen, "Interfaces for stream processing systems," in *Principles of Modeling: Essays Dedicated to Edward A. Lee on the Occasion of His 60th Birthday*. Springer, 2018, pp. 38–60.

[43] K. Mamouras, C. Stanford, R. Alur, Z. G. Ives, and V. Tannen, "Data-trace types for distributed stream processing systems," in *Proc. PLDI*. ACM, 2019, pp. 670–685.

[44] K. Mamouras, "Semantic foundations for deterministic dataflow and stream processing," in *Proc. ESOP*. Springer, 2020, pp. 394–427.

[45] G. H. Mealy, "A method for synthesizing sequential circuits," *The Bell System Technical Journal*, vol. 34, no. 5, pp. 1045–1079, 1955.

[46] E. F. Moore, "Gedanken-experiments on sequential machines," in *Automata Studies*. Princeton University Press, 1956, pp. 129–153.

[47] G. N. Raney, "Sequential functions," *Journal of the ACM*, vol. 5, no. 2, pp. 177–180, 1958.

[48] S. Ginsburg and G. F. Rose, "A characterization of machine mappings," *Canadian Journal of Mathematics*, vol. 18, pp. 381–388, 1966.

[49] S. Krstic and D. Traytel, "The aerial monitoring tool," https://bitbucket.org/traytel/aerial/src/master/, 2020, [Online; accessed June 17, 2020].

[50] D. Traytel, Private communication, June 2020.

[51] W. Zong and G. B. Moody, "wqrs: Single-lead QRS detector based on length transform," https://archive.physionet.org/physiotools/wfdb/app/wqrs.c, 2010, [Online; accessed April 17, 2020].

[52] P. Caspi and M. Pouzet, "Synchronous Kahn networks," in *Proc. ICFP*. ACM, 1996, pp. 226—238.

[53] J.-L. Colaço and M. Pouzet, "Clocks as first class abstract types," in *Proc. EMSOFT*. Springer, 2003, pp. 134–155.

[54] J.-L. Colaço, B. Pagano, and M. Pouzet, "A conservative extension of synchronous data-flow with state machines," in *Proc. EMSOFT*. ACM, 2005, pp. 173—182.

[55] A. Cohen, M. Duranton, C. Eisenbeis, C. Pagetti, F. Plateau, and M. Pouzet, "N-synchronous Kahn networks: A relaxed model of synchrony for real-time systems," in *Proc. POPL*. ACM, 2006, pp. 180–193.

[56] A. Cohen, L. Mandel, F. Plateau, and M. Pouzet, "Abstraction of clocks in synchronous data-flow systems," in *Proc. APLAS*. Springer, 2008, pp. 237–254.

[57] L. Mandel, F. Plateau, and M. Pouzet, "Static scheduling of latency insensitive designs with Lucy-n," in *Proc. FMCAD*, 2011, pp. 171–175.

[58] L. Mandel and F. Plateau, "Scheduling and buffer sizing of n-synchronous systems," in *Proc. MPC*. Springer, 2012, pp. 74–101.

[59] R. Alur, D. Fisman, and M. Raghothaman, "Regular programming for quantitative properties of data streams," in *Proc. ESOP*. Springer, 2016, pp. 15–40.

[60] R. Alur and K. Mamouras, "An introduction to the StreamQRE language," *Dependable Software Systems Engineering*, vol. 50, pp. 1–24, 2017.

[61] R. Alur, K. Mamouras, and C. Stanford, "Modular quantitative monitoring," *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, 2019.

[62] R. Alur, D. Fisman, K. Mamouras, M. Raghothaman, and C. Stanford, "Streamable regular transductions," *Theoretical Computer Science*, vol. 807, pp. 15–41, 2020.

[63] R. Alur, K. Mamouras, and C. Stanford, "Automata-based stream processing," in *Proc. ICALP*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, pp. 112:1–112:15.

[64] R. Alur, K. Mamouras, and D. Ulus, "Derivatives of quantitative regular expressions," in *Models, Algorithms, Logics and Tools: Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday*. Springer, 2017, pp. 75–95.

[65] N. Halbwachs, F. Lagnier, and P. Raymond, "Synchronous observers and the verification of reactive systems," in *Proc. AMAST*. Springer, 1994, pp. 83–96.

[66] P. Raymond, Y. Roux, and E. Jahier, "Lutin: A language for specifying and executing reactive scenarios," *EURASIP Journal on Embedded Systems*, vol. 2008, 2008.

[67] P. Raymond, "Recognizing regular expressions by means of dataflow networks," in *Proc. ICALP*. Springer, 1996, pp. 336–347.

[68] G. Delaval, H. Marchand, M. Pouzet, and E. Rutten, "Heptagon/BZR," http://heptagon.gforge.inria.fr/, 2020, [Online; accessed June 17, 2020].

Konstantinos Mamouras completed undergraduate studies in electrical and computer engineering at the National Technical University of Athens, Greece, received the M.Sc. degree in computer science from the Imperial College London, London, U.K., and the Ph.D. degree in computer science from Cornell University, Ithaca, NY, USA.

He is currently an Assistant Professor in the Department of Computer Science, Rice University, Houston, TX, USA. His research interests lie in the areas of programming languages and formal methods. His current research includes the design of domain-specific languages for processing real-time data, and the runtime verification of cyber-physical systems.

Zhifu Wang received the B.Sc. degree in computer science from Nanjing University, Nanjing, China, in 2019. She is currently pursuing the Ph.D. degree with the Department of Computer Science, Rice University, Houston, TX, USA. Her current research interests include programming languages and runtime verification.