Multi-Head Monitoring of Metric Temporal Logic

Martin Raszyk, David Basin, Srđan Krstić, and Dmitriy Traytel

Institute of Information Security, Department of Computer Science, ETH Zürich, Switzerland

Abstract. We present a novel approach to the offline monitoring of specifications expressed in metric temporal logic (MTL). Our monitoring algorithm exploits multiple one-way reading heads that traverse a trace sequentially. We present both theoretical and practical results that show this substantially improves upon the state-of-the-art. In particular, our algorithm is the first offline monitoring algorithm for MTL with past and bounded-future temporal operators that is almost trace-length independent and outputs a trace of Boolean verdicts denoting the monitored formula's satisfaction at every position in the input trace. In addition, our algorithm's worst-case space complexity is linear in the formula size, while previous algorithms were exponential. Moreover, we compare our implementation of the algorithm with another almost trace-length independent tool that outputs non-standard verdicts to achieve this space complexity. Our tool used less memory and runs significantly faster, for example yielding a 10-fold improvement on average on random formulas, while producing better output.

1 Introduction

Monitoring (or runtime verification) is the process of verifying system properties by analyzing system events against a specification formalizing which event sequences constitute the intended system behavior. Monitoring algorithms (or monitors) can be classified based on how they interact with the monitored system [11]. A core distinction is *when* the events are monitored. *Online monitors* process events at runtime, as they occur during system execution. Whereas *offline monitors* process them after the system has stopped running. Offline monitors are often seen as special cases of their online counterparts. Indeed, most monitoring algorithms are created independently of whether they will be used online or offline: they process one event at a time, in order of appearance.

There is however an important distinction between the two classes of monitors. Online monitors sequentially analyze a potentially unbounded stream of events and, due to the nature of streams, each event can be read only once. If an event is needed for subsequent analysis, an online monitor must keep it in memory. An online monitor's computation can be viewed as reading the stream with a single one-way reading head that moves forward only, updating the monitor's state, and producing output. Offline monitors, in contrast, analyze finite sequences of events, called traces, typically stored in files. An offline monitor is thus equipped with a reading head without movement constraints. Indeed, offline monitors may, effectively, have multiple heads (corresponding to indices in the trace) that read from multiple locations simultaneously. We note though that there are good reasons to process events in order, even in offline monitoring. First, the raw performance of sequential reads outperforms random access reads

due to prefetching and other low-level file system and hardware specifics [15]. Second, we can delete parts of the trace processed by all reading heads and append new events at the end, and thus, effectively emulate online monitoring. Hence, we will exploit the multiple heads, not the ability to read in both directions.

Our thesis in this paper is that by exploiting multiple (but finitely many) one-way reading heads that traverse a trace in order (Section 3), offline monitoring differs from, and improves upon, online monitoring. The ability to read each input multiple (but finitely many) times allows us to obtain strictly better complexity results. We establish this result for monitoring specifications expressed in metric temporal logic (MTL) [19] (Section 2). Moreover, we obtain the first offline almost trace-length independent [3] monitoring algorithm for MTL with past and bounded-future that outputs a trace of Boolean verdicts denoting the formula's satisfaction at every position in the input trace. Here, "almost" denotes a logarithmic dependence of the monitor's space complexity on the trace's length, stemming from the need to store the length in binary representation.

Our main contributions are: (i) a multi-head monitoring algorithm for MTL (Section 4); (ii) its correctness and complexity analysis (Section 5); and (iii) an implementation (available at [20]) and evaluation (Section 6). Both our complexity analysis and evaluation show significant improvements over the state-of-the-art.

Related Work An MTL formula's satisfaction is defined (Section 2) with respect to a position in a trace, which is an infinite word. A monitor takes as input a finite prefix of this trace. Monitoring is sometimes understood as the task of computing a *single Boolean verdict* denoting whether the formula is satisfied at the first position in the trace. For such monitors, there exist trace-length independent algorithms [8, 13, 17, 24]. These are algorithms whose space complexity is independent of the length of the finite prefix. This property is highly desirable because it distinguishes the monitors that can handle large traces from those that cannot. In contrast, we consider monitors that output *entire traces* of Boolean verdicts. That is, rather than outputting that a trace violates a formula, the monitor outputs every position where such a violation occurs. This output provides more insight into why and when the property was violated. There are trace-length independent algorithms for past-only LTL based on dynamic programming [16] and Thati and Roşu's interval-shifting [24] allows one to extend these results to past-only MTL.

Roşu and Havelund [22] develop a dual trace-length independent dynamic programming offline monitor for future-only LTL that traverses the trace backwards. Their idea is generalized by Sanchez [23], who proposes to alternate forward and backward traversals in the context of stream runtime verification [9, 12, 14], pioneered by Lola [10]. He claims to obtain trace-length independence for well-formed Lola specifications, which can express LTL with past and future. However, his complexity analysis appears to gloss over intermediate streams, which are as large as the input trace and store the results of backward passes to be reused in later forward passes. For LTL, this corresponds to assuming that verdicts for subformulas are available to evaluate a temporal formula, e.g., an until formula, without counting the memory used to store this information.

Basin et al. [2] develop almost trace-length independent algorithms for MTL (in fact, *almost event-rate independent*, which is a stronger property that is desirable for online monitoring) with past and future temporal operators. To achieve almost trace-length independence, they mix non-standard equivalence verdicts with standard Boolean ones.

$(\rho, i) \models p$ iff $p \in \Gamma_i$
$(\rho, i) \models \neg\varphi$ iff $(\rho, i) \not\models \varphi$
$(\rho, i) \models \varphi_1 \vee \varphi_2$ iff $(\rho, i) \models \varphi_1$ or $(\rho, i) \models \varphi_2$
$(\rho, i) \models \bullet_I \varphi$ iff $i > 0$ and $\tau_i - \tau_{i-1} \in I$ and $(\rho, i-1) \models \varphi$
$(\rho, i) \models \circ_I \varphi$ iff $\tau_{i+1} - \tau_i \in I$ and $(\rho, i+1) \models \varphi$
$(\rho, i) \models \varphi_1 \mathsf{S}_I \varphi_2$ iff $(\rho, j) \models \varphi_2$ for some $j \leq i$ with $\tau_i - \tau_j \in I$ and $(\rho, k) \models \varphi_1$ for all $j < k \leq i$
$(\rho, i) \models \varphi_1 \mathsf{U}_J \varphi_2$ iff $(\rho, j) \models \varphi_2$ for some $j \geq i$ with $\tau_j - \tau_i \in J$ and $(\rho, k) \models \varphi_1$ for all $i \leq k < j$

Fig. 1: Semantics of an MTL formula for a stream $\rho = \langle(\tau_i, \Gamma_i)\rangle_{i\in\mathbb{N}}$ and a time-point i

Equivalences can be resolved to Boolean verdicts but this (trace-length dependent) task is offloaded to the monitor's user. Our algorithm outputs standard Boolean verdicts and is almost trace-length independent. Moreover, their algorithm's space complexity is doubly exponential in the formula size, whereas ours is linear (Section 5.2).

In an independent line of work, we show how multiple reading heads can be leveraged to eliminate non-determinism from functional finite-state transducers [21].

2 Metric Temporal Logic

We recap the discrete-time point-based semantics of metric temporal logic (MTL) and refer to Basin et al. [6] for a comprehensive comparison with other semantics.

Let $\mathbb{T} = \mathbb{N}$ be the set of time-stamps, $\mathbb{I}_{\mathit{fin}}$ the set of non-empty finite intervals over $\mathbb{T}$, and $\mathbb{I}_\infty$ the set of infinite intervals over $\mathbb{T}$. We write elements of $\mathbb{I}_{\mathit{fin}}$ as $[l, r]$, where $l, r \in \mathbb{T}$, $l \leq r$, and $[l, r] = \{x \in \mathbb{T} \mid l \leq x \leq r\}$. Similarly, elements of $\mathbb{I}_\infty$ are written $[l, \infty]$ and denote $\{x \in \mathbb{T} \mid l \leq x\}$. Let $\mathbb{I} = \mathbb{I}_{\mathit{fin}} \cup \mathbb{I}_\infty$ be the set of all (non-empty) intervals over $\mathbb{T}$.

MTL formulas over a finite set of atomic predicates $P \neq \varnothing$ are defined inductively:

$$\varphi = p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \bullet_I \varphi \mid \circ_I \varphi \mid \varphi_1 \mathsf{S}_I \varphi_2 \mid \varphi_1 \mathsf{U}_J \varphi_2,$$

where $p \in P$, $I \in \mathbb{I}$, and $J \in \mathbb{I}_{\mathit{fin}}$. This minimal syntax includes Boolean operators and the temporal operators (*previous*) $\bullet_I$, (*since*) S_I, (*next*) $\circ_I$, and (*until*) U_J. We employ the usual syntactic sugar for additional Boolean constants and operators $\mathit{true} = p \vee \neg p$, $\mathit{false} = \neg\mathit{true}$, $\varphi \wedge \psi = \neg(\neg\varphi \vee \neg\psi)$, and temporal operators (*once*) $\blacklozenge_I \varphi = \mathit{true}\, \mathsf{S}_I\, \varphi$, (*historically*) $\blacksquare_I \varphi = \neg\blacklozenge_I \neg\varphi$, (*eventually*) $\lozenge_J \varphi = \mathit{true}\, \mathsf{U}_J\, \varphi$, and (*always*) $\square_J \varphi = \neg\lozenge_J \neg\varphi$.

MTL formulas are interpreted over *streams*, which are infinite sequences of events. An event has the form (τ_i, Γ_i), where the time-stamp $\tau_i \in \mathbb{T}$ is a nonnegative integer and $\Gamma_i \subseteq P$ is a subset of atomic predicates. We denote the set of events by $\mathbb{E} = \mathbb{T} \times 2^P$. We further assume that the sequence of time-stamps $\langle\tau_i\rangle_{i\in\mathbb{N}}$ is monotonically (non-strictly) increasing, i.e., $\tau_i \leq \tau_{i+1}$, for all $i \in \mathbb{N}$, and *unbounded*, i.e., for every $\tau \in \mathbb{T}$, there is an index (time-point) $i \in \mathbb{N}$ such that $\tau_i \geq \tau$. A finite prefix $\rho_{\leq i} = \rho_0 \ldots \rho_i$ of an event stream ρ is called a *trace*. Figure 1 shows the standard semantics of MTL formulas.

We define $\mathit{reach}([l, r]) = r$ if $r < \infty$, and $\mathit{reach}([l, r]) = l - 1$ otherwise. For the S_I operator (and analogously for the U_J operator), $\mathit{reach}(I)$ is the maximum value $\tau_i - \tau_j$ (see Figure 1) that our monitor stores during its evaluation. We define the set $\mathsf{SF}(\varphi)$ of all formula φ's subformulas as usual (including φ) and the *size* $|\varphi| = |\mathsf{SF}(\varphi)|$. We define the formula φ's *temporal size* $\|\varphi\|$ to be the sum of $|\varphi|$ and $\mathit{reach}(I)$ for all intervals I in φ.

3 Multi-Head Monitoring

The computation of an online monitor on an infinite event stream can be viewed as reading the stream with a *single* head moving forwards only (i.e., a *one-way* reading head), updating the monitor's state, and optionally producing an output, i.e., some verdicts. A multi-head monitor extends an online monitor with *multiple* one-way reading heads.

Definition 1. *A* multi-head monitor *is a tuple* $M = (P, V, \kappa, Q, q_0, \delta)$*, where* P *is a non-empty finite set of atomic predicates,* V *is a verdict alphabet,* $\kappa \in \mathbb{N}$ *is the number of one-way (reading) heads,* Q *is a set of states,* $q_0 \in Q$ *is an initial state, and* $\delta : Q \times \mathbb{E}^\kappa \to Q \times \{0,1\}^\kappa \times (V \cup \{\bot\})$ *is a step function that maps a state and the events read by the heads to a new state, offsets that indicate which heads to advance, and an optional verdict.*

This computational model in principle allows for an infinite set of states, which may seem unreasonable in practice. Rather than a priori restricting the model, we provide a space bound for our multi-head monitoring algorithm in Section 5 (Theorem 3).

The *configuration* of a multi-head monitor M consists of the current state and the reading heads' positions. Formally, the set of configurations is $\mathbb{C} = Q \times \mathbb{N}^\kappa$. Let $\rho[\overline{p}] \in \mathbb{E}^\kappa$ denote the events read from the stream ρ by the heads positioned at $\overline{p} \in \mathbb{N}^\kappa$. The *computation* on a stream ρ is an infinite sequence of configurations $\langle (q_i, \overline{p}^i) \rangle_{i \in \mathbb{N}}$ that starts with the initial configuration $(q_0, \overline{0})$ and in which any two consecutive configurations $(q_i, \overline{p}^i)$, $(q_{i+1}, \overline{p}^{i+1})$ satisfy $\delta(q_i, \rho[\overline{p}^i]) = (q_{i+1}, \overline{p}^{i+1} - \overline{p}^i, v_i')$, for some $v_i' \in V \cup \{\bot\}$. Finally, the *output* of a computation on a stream is the concatenation of the verdicts $v_i' \neq \bot$, i.e., a (potentially empty) sequence $\langle v_i \rangle_{i<n}$, with $v_i \in V$ and $n \in \mathbb{N} \cup \{\infty\}$.

For the remainder of this paper, let $V = \mathbb{T} \times \mathbb{B}$ be the verdict alphabet, that is, a verdict $v = (t, b)$ is a Boolean value $b \in \mathbb{B} = \{\mathsf{tt}, \mathsf{ff}\}$ time-stamped by $t \in \mathbb{T}$. Note that, with this choice of V, the step function returns at most one time-stamped Boolean verdict at a time, which simplifies the algorithm's presentation and complexity analysis.

An output $\langle (t_k, b_k) \rangle_{k<n}$ of a computation on a stream ρ is *sound* with respect to an MTL formula Φ if and only if the time-stamps t_k correspond to the time-stamps on the prefix $\rho_{<n}$ and the Boolean values b_k reflect the formula's semantics at the time-points k. Formally, the output $\langle (t_k, b_k) \rangle_{k<n}$ of the computation on a stream $\rho = \langle (\tau_i, \Gamma_i) \rangle_{i \in \mathbb{N}}$ is sound if and only if the following predicate $\mathsf{SOUND}_\Phi(\rho, \langle (t_k, b_k) \rangle_{k<n})$ holds:

$$\mathsf{SOUND}_\Phi(\rho, \langle (t_k, b_k) \rangle_{k<n}) \equiv \forall k.\ k < n \to (\tau_k = t_k \wedge ((\rho, k) \vDash \Phi \iff b_k)).$$

In the subsequent text, we use $[m]_0$ for the set $\{0, \ldots, m\}$, and $[m]$ for the set $\{1, \ldots, m\}$.

4 Multi-Head Monitoring of Metric Temporal Logic

We structure our multi-head monitor for a given formula Φ into two procedures. The first, recursive procedure follows Φ's structure (Section 4.1). In doing so, it recursively runs a separate multi-head monitor instance for each direct subformula of Φ. The procedure may invoke the step function of an instance multiple times until a verdict is produced. This may be necessary when future subformulas are monitored. The verdicts already produced by other instances are cached. Once every instance has produced a verdict, the second procedure is invoked. It combines the verdicts for Φ's direct subformulas into a verdict for Φ based on the semantics of Φ's top-level operator (Section 4.2).

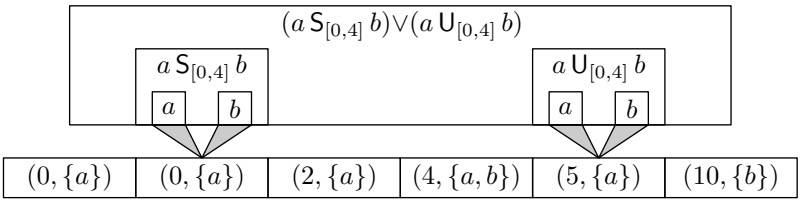

Fig. 2: Multi-head monitor's structure

In general, a monitor running many multi-head (sub)monitors is itself a multi-head monitor, inheriting reading heads from its submonitors. All such reading heads can move independently and asynchronously. In our monitor, the number of reading heads for a formula Φ equals the number of atomic subformulas in Φ.

Example 1. Consider the formula $\Phi = (a\,\mathsf{S}_{[0,4]}\,b) \vee (a\,\mathsf{U}_{[0,4]}\,b)$, which has four atomic subformulas corresponding to two occurrences of each of a and b. Figure 2 shows the monitor's structure as well as a snapshot of the four reading heads' positions while monitoring a trace. The reading heads are depicted as the gray triangles. In the current snapshot, the monitor for Φ has just produced a verdict for the first position. To do so, it needed the two corresponding verdicts for its two direct subformulas $a\,\mathsf{S}_{[0,4]}\,b$ and $a\,\mathsf{U}_{[0,4]}\,b$. The submonitor for $a\,\mathsf{S}_{[0,4]}\,b$ could deliver the verdict after reading the first position (and then advancing its heads to the second position). In contrast, the submonitor for $a\,\mathsf{U}_{[0,4]}\,b$ could only produce a verdict for the first position in this trace after reading the fourth position (and then advancing its heads to the fifth position).

4.1 First Procedure: Recursively Running the Multi-Head Monitors

To formally construct our multi-head monitor for an MTL formula Φ over a finite set of atomic predicates P^Φ, we perform a case distinction on the structure of Φ.

Atomic Predicate Let $\Phi = p$, where $p \in P^\Phi$. The monitor checks whether p is included in an event. For this, a single reading head suffices and no state is needed. Formally, $\kappa^\Phi = 1$, $Q^\Phi = \{\bot\}$, and $q_0^\Phi = \bot$. The step function is $\delta^\Phi(\bot, (\tau, \Gamma)) = (\bot, 1, (\tau, p \in \Gamma))$.

Recursive Formula Let the top-level operator of Φ be op, where $\mathsf{op} \in \{\neg/1, \vee/2, \bullet_I/1, \bigcirc_I/1, \mathsf{S}_I/2, \mathsf{U}_J/2\}$ and the number next to an operator op denotes its arity $\eta(\mathsf{op})$. For $i \in [\eta(\mathsf{op})]$, let φ_i be a direct subformula of Φ and $M^{\varphi_i} = (P^{\varphi_i}, V, \kappa^{\varphi_i}, Q^{\varphi_i}, q_0^{\varphi_i}, \delta^{\varphi_i})$ be its multi-head monitor, which we construct recursively.

At this point, we abstract the operator evaluation procedure (Section 4.2). Specifically, we assume that each operator op comes with a set of states C^{op}, an initial state c_0^{op}, and a step function $s^{\mathsf{op}} : C^{\mathsf{op}} \times \mathbb{T} \times \mathbb{B}^{\eta(\mathsf{op})} \to C^{\mathsf{op}} \times V^*$. The step function is applied to a state, a time-stamp, and a tuple of Boolean values coming from the recursive invocations of the submonitors. It returns the new state and a list of verdicts.

Evaluating the operator op does not require any reading heads. Thus the number of heads κ^Φ of M^Φ is the sum of the numbers of heads κ^{φ_i} of M^{φ_i} for all Φ's direct subformulas φ_i. In particular, for each M^{φ_i}, there is a subset of M^Φ's heads belonging to M^{φ_i}.

$\delta^{\Phi} : Q^{\Phi} \times \mathbb{E}^{\kappa^{\Phi}} \to Q^{\Phi} \times \{0,1\}^{\kappa^{\Phi}} \times (V \cup \{\bot\})$

1 $\delta^{\Phi}((qs, bs, c^{\mathsf{op}}, vs), es) =$ **match** vs **do**
2 **case** $v \cdot vs' \Rightarrow$ **return** $((qs, bs, c^{\mathsf{op}}, vs'), \overline{0}, v)$
3 **case** $\varepsilon \Rightarrow$
4 let i be the smallest index such that $bs[i] = \bot$
5 let es^i be the events from es read by heads belonging to M^{φ_i}
6 $(q^{\varphi_i}, ms^i, v) := \delta^{\varphi_i}(q^{\varphi_i}, es^i)$
7 **match** v **do**
8 **case** $(\tau, b) \Rightarrow$ $bs[i] := b$
9 **if** $i = \eta(\mathsf{op})$ **then** $(c^{\mathsf{op}}, vs) := s^{\mathsf{op}}(c^{\mathsf{op}}, \tau, bs)$
10 $bs := \overline{\bot}$ **fi**
11 **case** $\bot \Rightarrow$ **od**
12 let ms extend offsets ms^i (from M^{φ_i}) with zero offsets (from M^{φ_j}, $j \neq i$)
13 **return** $((qs, bs, c^{\mathsf{op}}, vs), ms, \bot)$ **od**

Fig. 3: Recursive monitor

Figure 3 shows our multi-head monitor M^{Φ} for the formula Φ. Its state, $(qs, bs, c^{\mathsf{op}}, vs)$, stores a tuple qs whose i-th component is a submonitor's state q^{φ_i} and a tuple bs of optional Boolean values that cache the latest unprocessed verdict produced by M^{φ_i}. Moreover, the state of M^{Φ} stores a state c^{op} for evaluating the operator op as well as a list vs of time-stamped Boolean verdicts produced by evaluating the operator op, but not yet output. As long as vs is non-empty, the monitor outputs verdicts from it (line 2). Otherwise, the monitor repeatedly triggers its submonitors to produce a verdict (lines 3–13). Note that a submonitor does not need to produce a verdict upon each evaluation of its step function (line 7). Once M^{Φ} knows the Boolean verdicts for all subformulas, it can apply the operator-specific computation (line 9), which may return some new verdicts vs for Φ. Afterwards, it resets bs to eventually continue the subformulas' evaluation.

The initial state of M^{Φ} consists of a tuple qs, whose i-th component is $q_0^{\varphi_i}$, a tuple bs consisting of $\bot$, the initial state c_0^{op} for evaluating the operator op, and an empty list ε of time-stamped Boolean verdicts.

4.2 Second Procedure: Evaluating a Top-Level Operator

We describe how to evaluate each individual MTL operator by defining its set of states C^{op}, initial state c_0^{op}, and the step function $s^{\mathsf{op}} : C^{\mathsf{op}} \times \mathbb{T} \times \mathbb{B}^{\eta(\mathsf{op})} \to C^{\mathsf{op}} \times V^*$.

Negation and Disjunction For Boolean operators $\mathsf{op} \in \{\neg, \vee\}$, no state is needed and the step function simply combines its Boolean inputs. We have $C^{\mathsf{op}} = \{\bot\}$, $c_0^{\mathsf{op}} = \bot$, and

$$s^{\neg}(\bot, \tau, b) = (\bot, (\tau, \neg b)),$$
$$s^{\vee}(\bot, \tau, b_1, b_2) = (\bot, (\tau, b_1 \vee b_2)).$$

Previous and Next For $\mathsf{op} = \bullet_I$, the state stores the time-stamped Boolean value, denoting the operator's subformula's satisfaction at a previous time-point, if the current time-point is not the initial time-point. The state for $\mathsf{op} = \bigcirc_I$ is the same, only that no Boolean

value is stored. More formally, $C^{\bullet_I} = (\mathbb{T} \times \mathbb{B}) \cup \{\bot\}$ and $C^{\bigcirc_I} = \mathbb{T} \cup \{\bot\}$. The initial state is $c_0^{\mathsf{op}} = \bot$ in both cases. For $\bullet_I$, the step function propagates the stored Boolean value to its output if the time-stamp constraints given by I are satisfied. For $\bigcirc_I$, the step function also checks the constraints. If the check passes, it outputs the given Boolean input denoting the satisfaction of the operator's subformula at the current time-point as the verdict for the previous time-point (whose time-stamp is stored in $c^{\bigcirc_I}$). The initial state must be treated specially in both cases. Formally, the step function is defined as follows:

$$s^{\bullet_I}(c,\tau,b) = \begin{cases} ((\tau,b),(\tau,b' \wedge (\tau - \tau' \in I))) & \text{if } c = (\tau',b'), \\ ((\tau,b),(\tau,\mathsf{ff})) & \text{if } c = \bot, \end{cases}$$

$$s^{\bigcirc_I}(c,\tau,b) = \begin{cases} (\tau,(\tau',b \wedge (\tau - \tau' \in I))) & \text{if } c = \tau', \\ (\tau,\varepsilon) & \text{if } c = \bot. \end{cases}$$

Since and Until These temporal operators are more complex. For the since operator, the state keeps a history of *satisfaction witnesses* that correspond to the time-points j in the MTL semantics of the since operator for the operator's satisfaction at the current time-point i (Figure 1). For the until operator, the state keeps a history of *satisfaction candidates* that correspond to the time-points i, for which a future time-point could become the time-point j in the MTL semantics of the until operator (Figure 1). Instead of storing the time-points explicitly, we represent them by a list of zeros interleaved with (positive) time-stamp differences between the successive time-points. For example, consider the case where the monitor for the since operator has processed the following eight time-points j with the time-stamps τ_j and the corresponding Boolean values b_1^j and b_2^j.

j	0	1	2	3	4	5	6	7
τ_j	4	8	10	10	11	11	11	14
b_1^j	tt	tt	tt	tt	tt	tt	tt	tt
b_2^j	tt	ff	tt	tt	tt	ff	ff	ff

The satisfaction witnesses here are the time-points 0, 2, 3, 4, marked in gray above. Our history represents them with the list $[0, 4, 2, 0, 0, 1, 0, 3]$, where every zero corresponds to a satisfaction witness and the other numbers show only the *non-zero* time-stamp differences between the successive time-points. Note that the history can be obtained from the list of all time-stamp differences $[4, 2, 0, 1, 0, 0, 3]$ by dropping all zero time-stamp differences $[4, 2, 1, 3]$ and then inserting a zero for each satisfaction witness at the corresponding position $[0, 4, 2, 0, 0, 1, 0, 3]$. Crucially for our space complexity analysis, we store such lists using a *run-length encoding*, where we compress subsequences $\tau, \ldots, \tau$ to the pair (τ, n), where n is length of the subsequence. We use a shorthand notation for run-length encoded lists, e.g. writing 0420^2103 for the above list.

The state stores also the time-stamp (and, for the since operator, the Boolean value, too, similarly to $\bullet_I$) at a previous time-point, if the current time-point is not the initial time-point. Formally, $C^{\mathsf{S}_I} = ((\mathbb{T} \times \mathbb{B}) \cup \{\bot\}) \times \mathbb{T}^*$ and $C^{\mathsf{U}_J} = (\mathbb{T} \cup \{\bot\}) \times \mathbb{T}^*$. The initial state is $c_0^{\mathsf{op}} = (\bot, \varepsilon)$ for both operators.

To define the step functions, we first define the following operations on the lists of time-stamp differences. $\mathsf{SUM}(ts)$ is the sum of all time-stamp differences in the list ts.

$s^{\mathsf{S}_{[l,r]}} : C^{\mathsf{S}_{[l,r]}} \times \mathbb{T} \times \mathbb{B}^2 \to C^{\mathsf{S}_{[l,r]}} \times V^*$

1 $s^{\mathsf{S}_{[l,r]}}((v, ts), \tau, b_1, b_2) =$ **match** v **do**
2 **case** $(\tau', b) \Rightarrow$ $\beta := b$
3 **if** $\tau' < \tau \wedge ts \neq \varepsilon$ **then**
4 $ts := ts \cdot (\tau - \tau')$ **fi**
5 **case** $\bot \Rightarrow \beta :=$ ff **od**
6 $(\_, ts) := \mathsf{SPLIT}(ts, \lambda s.\ s \leq r)$
7 **if** $\neg b_1$ **then** $ts := \varepsilon$
8 $\beta :=$ ff **fi**
9 **if** b_2 **then** $ts := ts \cdot 0$ **fi**
10 **if** $r < \infty$ **then**
11 $\beta := (ts \neq \varepsilon \wedge \mathsf{SUM}(ts) \geq l)$
12 **else**
13 $(ds, ts) := \mathsf{SPLIT}(ts, \lambda s.\ s < l)$
14 **if** $ds \neq \varepsilon$ **then** $\beta :=$ tt **fi fi**
15 **return** $(((\tau, \beta), ts), (\tau, \beta))$

$s^{\mathsf{U}_{[l,r]}} : C^{\mathsf{U}_{[l,r]}} \times \mathbb{T} \times \mathbb{B}^2 \to C^{\mathsf{U}_{[l,r]}} \times V^*$

1 $s^{\mathsf{U}_{[l,r]}}((t, ts), \tau, b_1, b_2) =$ **match** t **do**
2 **case** $\tau' \Rightarrow$ **if** $\tau' < \tau \wedge ts \neq \varepsilon$ **then**
3 $ts := ts \cdot (\tau - \tau')$ **fi**
4 **case** $\bot \Rightarrow$ **od**
5 $vs := \varepsilon$
6 $(ds, ts) := \mathsf{SPLIT}(ts, \lambda s.\ s \leq r)$
7 $vs := vs \cdot \mathsf{RES}(ds, \tau - \mathsf{SUM}(ts),$ ff$)$
8 $ts := ts \cdot 0$
9 **if** b_2 **then**
10 $(ds, ts) := \mathsf{SPLIT}(ts, \lambda s.\ s < l)$
11 $vs := vs \cdot \mathsf{RES}(ds, \tau - \mathsf{SUM}(ts),$ tt$)$ **fi**
12 **if** $\neg b_1$ **then**
13 $vs := vs \cdot \mathsf{RES}(ts, \tau,$ ff$)$
14 $ts := \varepsilon$ **fi**
15 **return** $((\tau, ts), vs)$

Fig. 4: Step functions for $\mathsf{S}_{[l,r]}$ and $\mathsf{U}_{[l,r]}$

For a predicate π on $\mathbb{T}$, $\mathsf{SPLIT}(ts, \pi) = (ts_1, ts_2)$ partitions the list ts into two lists ts_1 and ts_2, such that the list ts_1 is as short as possible and the list ts_2 is either empty or starts with a zero and its sum $\mathsf{SUM}(ts_2)$ satisfies the predicate π.

Formally, we evaluate the since and until operators on a stream $\sigma = \langle(\tau_i, \overline{b}_1^i, \overline{b}_2^i)\rangle_{i \in \mathbb{N}}$, where $\overline{b}_1$ and $\overline{b}_2$ are the streams of Boolean values incrementally received by the operator's step function. After processing the time-point i, each time-point $j \leq i$ satisfying

$$\tau_i - \tau_j \leq reach([l,r]) \wedge \overline{b}_2^j \wedge \forall k.\ \left(j < k \leq i \Longrightarrow \overline{b}_1^k\right) \qquad \text{if } \mathsf{op} = \mathsf{S}_{[l,r]},$$
$$\tau_i - \tau_j \leq reach([l,r]) \wedge \forall k.\ \left(j \leq k \leq i \Longrightarrow \overline{b}_1^k \wedge \left(\overline{b}_2^k \Longrightarrow \tau_k - \tau_j < l\right)\right) \qquad \text{if } \mathsf{op} = \mathsf{U}_{[l,r]},$$

corresponds to a unique suffix $0 \cdot ts'$ of the list ts in the operator op's state satisfying $\mathsf{SUM}(ts') = \tau_i - \tau_j$. In particular, ts satisfies $\mathsf{SUM}(ts) \leq reach([l,r])$.

Figure 4 shows the definitions of the step functions. For the S_I operator, we proceed in four main steps. (1) We compute the current time-stamp difference and add it to the history (lines 3–4). (2) We drop the time-stamp differences that fall out of the interval I from the history (line 6). (3) We use the Boolean values denoting the subformula's satisfaction at the current point to update the history, starting with the since operator's first (left) subformula (lines 7–9). If the first subformula is not satisfied, all previous satisfaction witnesses of the since operator stored in the history are invalidated and must be dropped (line 7). If the second subformula is satisfied, we add a new satisfaction witness (line 9). (4) The output of the operator is in each step a single Boolean verdict denoting whether there is a satisfaction witness starting in the interval I (lines 2, 5, and 10–14; the latter distinguish between finite and infinite intervals). For the U_I operator, steps (1) (lines 2–3), (2) (line 6), and (3) (lines 9–14) are similar, except that the subformulas are checked in the reverse order starting from the until operator's second (right) subformula. The main difference is the way the until operator's evaluation resolves the satisfaction candidates to Boolean verdicts (lines 7, 11, and 13). This resolution uses

	$\varphi_1 = a\,\mathsf{S}_{[0,4]}\,b$		$\varphi_2 = a\,\mathsf{U}_{[0,4]}\,b$	
Trace	$c^{\mathsf{S}_{[0,4]}}$	Verdicts	$c^{\mathsf{U}_{[0,4]}}$	Verdicts
	$(\bot,\varepsilon)$		$(\bot,\varepsilon)$	
$(0,\{a\})$	$((0,\mathtt{ff}),\varepsilon)$	$(0,\mathtt{ff})$	$(0,0)$	ε
$(0,\{a\})$	$((0,\mathtt{ff}),\varepsilon)$	$(0,\mathtt{ff})$	$(0,0^2)$	ε
$(2,\{a\})$	$((2,\mathtt{ff}),\varepsilon)$	$(2,\mathtt{ff})$	$(2,0^220)$	ε
$(4,\{a,b\})$	$((4,\mathtt{tt}),0)$	$(4,\mathtt{tt})$	$(4,\varepsilon)$	$(0,\mathtt{tt})(0,\mathtt{tt})(2,\mathtt{tt})(4,\mathtt{tt})$
$(5,\{a\})$	$((5,\mathtt{tt}),01)$	$(5,\mathtt{tt})$	$(5,0)$	ε
$(10,\{b\})$	$((10,\mathtt{tt}),0)$	$(10,\mathtt{tt})$	$(10,\varepsilon)$	$(5,\mathtt{ff})(10,\mathtt{tt})$

Fig. 5: Example operator evaluation for the formulas $a\,\mathsf{S}_{[0,4]}\,b$ (left) and $a\,\mathsf{U}_{[0,4]}\,b$ (right).

the auxiliary function $\mathsf{RES}(ds,\tau,b)$ that maps every suffix $0\cdot ds'$ of the list ds to a time-stamped Boolean value $(\tau-\mathsf{SUM}(ds'),b)$. We omit its straightforward definition. Note that one step of the U_I operator's evaluation can produce several verdicts.

Example 1. We continue Example 1 from this section's start by considering a monitor for the formula $\Phi = (a\,\mathsf{S}_{[0,4]}\,b)\vee(a\,\mathsf{U}_{[0,4]}\,b)$. Figure 5 describes the monitor's execution steps on the trace from Figure 2. The trace is shown in column 1. A state q^Φ of the monitor consists of a pair of states q^{φ_1} and q^{φ_2} for its two submonitors for $\varphi_1 = a\,\mathsf{S}_{[0,4]}\,b$ and $\varphi_2 = a\,\mathsf{U}_{[0,4]}\,b$, a pair of optional Boolean values that cache the latest unprocessed verdict produced by the two submonitors, the state $\bot$ for Φ's top-level operator $\vee$, and a list of time-stamped Boolean verdicts produced by evaluating the operator $\vee$.

The state q^{φ_1} consists of a pair $(\bot,\bot)$ of submonitor states for atomic predicates a and b, a pair of optional Boolean values that cache the latest unprocessed verdict produced by the two submonitors, a state $c^{\mathsf{S}_{[0,4]}}$ for the top-level operator of φ_1 (column 2) and a list of time-stamped Boolean verdicts produced by evaluating the operator $\mathsf{S}_{[0,4]}$ (column 3), to be output. The length of this list is at most one since the step function $s^{\mathsf{S}_{[0,4]}}$ never produces more than one verdict at a time.

A state q^{φ_2} has a similar structure to a state q^{φ_1}. We show $c^{\mathsf{U}_{[0,4]}}$ for the top-level operator of φ_2 in column 4. Note that the length of the list of time-stamped Boolean verdicts produced by evaluating the operator $\mathsf{U}_{[0,4]}$ may be arbitrary (column 5).

The since operator's step function produces a single verdict after each time-point and keeps the last time-stamped Boolean verdict as the first component of the operator state $c^{\mathsf{S}_{[0,4]}}$. The second component of $c^{\mathsf{S}_{[0,4]}}$ is the run-length encoded list of satisfaction witnesses. Since none of the first three time-points contains an atomic predicate b, the list of satisfaction witnesses is empty. The fourth time-point is a satisfaction witness, which is represented by a zero in the list of satisfaction witnesses. The fifth time-point is not a satisfaction witness, but the time-stamp difference to the previous time-point is appended to the list of satisfaction witnesses. At the last time-point, the fourth time-point falls out of the interval $[0,4]$ of the since operator and is removed from the list of satisfaction witnesses, which becomes empty. The last time-point is a new satisfaction witness itself; hence there is a zero in the list of satisfaction witnesses.

The until operator's step function may produce no verdict or multiple verdicts after a time-point. The first three time-points do not contain the atomic predicate b and thus become satisfaction candidates. Each of them corresponds to a zero in the run-length encoded list of satisfaction candidates, which is the state's second component. (The first

component is the time-stamp from the previously read time-point.) Since the time-stamp difference between the second and third time-stamp is nonzero, it is prepended before the last zero corresponding to the third satisfaction candidate. The fourth time-point contains an atomic predicate b, which resolves all satisfaction candidates, including the current time-point, to $\mathtt{tt}$ (as none of them falls out of the until operator's interval). After the fourth time-point, there are no satisfaction candidates. The fifth time-point is a new satisfaction candidate. The time-stamp difference between the last two time-points makes the fifth time-point fall out of the interval associated with the until operator and it is thus resolved to $\mathtt{ff}$. Since the last time-point contains the atomic predicate b, it is immediately resolved to $\mathtt{tt}$, and the list of satisfaction candidates becomes empty.

The monitor for the formula $\Phi = \varphi_1 \vee \varphi_2$ can only produce a verdict for the first time-point once it obtains the corresponding verdicts for its two subformulas. The submonitor for φ_1 outputs the verdict for the first time-point immediately after processing it (column 3). In contrast, the submonitor for φ_2 only outputs the verdict for the first time-point after processing the fourth time-point (column 5). After the verdict for the first time-point has been output, the remaining three verdicts returned by the operator step function for φ_2 are stored in the state q^{φ_2}. At this point, the reading heads are at the positions shown in Figure 2. The operator step function of φ_1 is then invoked at the second, third, and fourth time-point, making the reading heads of the submonitor for φ_1 catch up with the ones of the submonitor for φ_2. The three verdicts stored in the state q^{φ_2} are combined with the new ones obtained from the submonitor for φ_1. At the fifth time-point, the operator step function for φ_2 is invoked again without outputting any verdicts. After its next invocation, it outputs two verdicts for the last two time-points, which are then again similarly stored in φ_2 until the submonitor for φ_1 catches up.

5 Correctness and Complexity Analysis

We reverse the order compared to the previous section: We first prove the correctness of operator evaluation and then the recursive monitor's correctness. Afterwards we precisely bound the space complexity of representing the multi-head monitor's state.

5.1 Correctness of Operator Evaluation

To prove the correctness of MTL operator evaluation, we first formulate an invariant $\mathcal{I}(\mathsf{op}, \sigma_{\leq i}, c^{\mathsf{op}}, o^{\mathsf{op}})$ on the state c^{op} and output $o^{\mathsf{op}} \in V^*$ produced by evaluating an operator op that holds after applying the step function s^{op} on the finite stream prefix $\sigma_{\leq i} = \langle (\tau_k, \overline{b}^k_1, \ldots, \overline{b}^k_{\eta(\mathsf{op})}) \rangle_{k \leq i}$, which stores the Boolean values $\overline{b}^k_1, \ldots, \overline{b}^k_{\eta(\mathsf{op})}$ passed to op on the k-th invocation of its step function, starting from the initial state c^{op}_0.

We first define an auxiliary predicate $\mathsf{SOUND}_{\mathsf{op}}(\sigma_{\leq i}, o^{\mathsf{op}})$ that asserts the soundness of MTL operator evaluation. Assuming that the Boolean values from $\sigma_{\leq i}$ denote the verdicts of a formula's Φ direct subformulas, the operator has to produce the verdicts for Φ:

$$\begin{aligned}&\mathsf{SOUND}_{\mathsf{op}}(\langle (\hat{\tau}_k, \overline{\hat{b}}^k) \rangle_{k \leq i}, \langle (t_k, b_k) \rangle_{k<n}) \equiv \forall \rho\ \Phi\ \varphi_1\ \ldots\ \varphi_{\eta(\mathsf{op})}.\\ &\quad \Phi = \mathsf{op}(\varphi_1, \ldots, \varphi_{\eta(\mathsf{op})}) \wedge (\forall j \in [\eta(\mathsf{op})].\ \mathsf{SOUND}_{\varphi_j}(\rho, \langle (\hat{\tau}_k, \overline{\hat{b}}^k_j) \rangle_{k<i+1})) \Longrightarrow\\ &\quad \mathsf{SOUND}_{\Phi}(\rho, \langle (t_k, b_k) \rangle_{k<n}).\end{aligned}$$

For an empty stream prefix, the invariant asserts that the state is the initial state and no output has been produced: $\mathcal{I}(\mathsf{op}, \varepsilon, c^{\mathsf{op}}, o^{\mathsf{op}}) \equiv c^{\mathsf{op}} = c_0^{\mathsf{op}} \wedge o^{\mathsf{op}} = \varepsilon$.

Negation and Disjunction For $\mathsf{op} \in \{\neg, \vee\}$, the invariant asserts that sound output (second conjunct) has been produced for all time-points (from 0) up to i (first conjunct):

$$\mathcal{I}(\mathsf{op}, \sigma_{\leq i}, c^{\mathsf{op}}, o^{\mathsf{op}}) \equiv |o^{\mathsf{op}}| = i+1 \wedge \mathsf{SOUND}_{\mathsf{op}}(\sigma_{\leq i}, o^{\mathsf{op}}).$$

Previous and Next For $\mathsf{op} \in \{\bullet_I, \bigcirc_I\}$, the invariant asserts that sound output has been produced for all time-points up to i, or $i-1$, respectively, and that the state stores the time-stamp (and, for the $\bullet_I$ operator, also the Boolean value) from $\sigma_{\leq i}$ at i.

$$\begin{aligned}\mathcal{I}(\bullet_I, \sigma_{\leq i}, (t^{\mathsf{op}}, b^{\mathsf{op}}), o^{\mathsf{op}}) &\equiv |o^{\mathsf{op}}| = i+1 \wedge (\tau_i, \overline{b}_1^i) = (t^{\mathsf{op}}, b^{\mathsf{op}}) \wedge \mathsf{SOUND}_{\mathsf{op}}(\sigma_{\leq i}, o^{\mathsf{op}}) \\ \mathcal{I}(\bigcirc_I, \sigma_{\leq i}, t^{\mathsf{op}}, o^{\mathsf{op}}) &\equiv |o^{\mathsf{op}}| = i \wedge \tau_i = t^{\mathsf{op}} \wedge \mathsf{SOUND}_{\mathsf{op}}(\sigma_{\leq i}, o^{\mathsf{op}})\end{aligned}$$

In the following, let $I = [l, r]$. For the since and until operators, we use an auxiliary function CSUF that counts the number of suffixes of a list starting with a zero and having a fixed sum:

$$\mathsf{CSUF}(ts, \Delta) = |\{ts' \mid \exists w.\, ts = w \cdot 0 \cdot ts' \wedge \mathsf{SUM}(ts') = \Delta\}|.$$

Since The invariant for the since operator $\mathsf{op} = \mathsf{S}_I$ asserts that (i) sound output has been produced for all time-points up to i, (ii) the state stores the verdict at the time-point i, and (iii) the list stored in the state contains all satisfaction witnesses (Section 4):

$$\begin{aligned}\mathcal{I}(\mathsf{S}_I, \sigma_{\leq i}, (v^{\mathsf{op}}, ts), o^{\mathsf{op}}) \equiv |o^{\mathsf{op}}| = i+1 \wedge o_i^{\mathsf{op}} = v^{\mathsf{op}} \wedge \mathsf{SUM}(ts) \leq \mathit{reach}(I) \wedge \\ \forall \Delta \in [\mathit{reach}(I)]_0.\, \mathsf{WITS}(\sigma_{\leq i}, \Delta) = \mathsf{CSUF}(ts, \Delta) \wedge \mathsf{SOUND}_{\mathsf{op}}(\sigma_{\leq i}, o^{\mathsf{op}}),\end{aligned}$$

where the auxiliary function WITS counts the satisfaction witnesses:

$$\mathsf{WITS}(\sigma_{\leq i}, \Delta) = |\{j \in [i]_0 \mid \tau_i - \tau_j = \Delta \wedge \overline{b}_2^j \wedge \forall k.\, j < k \leq i \Longrightarrow \overline{b}_1^k\}|.$$

Until The invariant for the until operator $\mathsf{op} = \mathsf{U}_J$ is similar to the one for the since operator, but asserts that sound output has been produced for all time-points that do not have a corresponding suffix starting with a zero in the stored list ts:

$$\begin{aligned}\mathcal{I}(\mathsf{U}_J, \sigma_{\leq i}, (t^{\mathsf{op}}, ts), o^{\mathsf{op}}) \equiv |o^{\mathsf{op}}| = i+1 - \textstyle\sum_{\Delta=0}^{r} \mathsf{CSUF}(ts, \Delta) \wedge \tau_i = t^{\mathsf{op}} \wedge \\ \mathsf{SUM}(ts) \leq r \wedge \forall \Delta \in [r]_0.\, \mathsf{CANDS}(\sigma_{\leq i}, \Delta) = \mathsf{CSUF}(ts, \Delta) \wedge \mathsf{SOUND}_{\mathsf{op}}(\sigma_{\leq i}, o^{\mathsf{op}}),\end{aligned}$$

where the auxiliary function CANDS counts the satisfaction candidates (Section 4):

$$\mathsf{CANDS}(\sigma_{\leq i}, \Delta) = |\{j \in [i]_0 \mid \tau_i - \tau_j = \Delta \wedge \forall k \in \{j, \ldots, i\}.\, \overline{b}_1^k \wedge (\overline{b}_2^k \Longrightarrow \tau_k - \tau_j < l)\}|.$$

We now establish that the $\mathcal{I}$ is indeed an inductive invariant.

Lemma 1. *Let* op *be an MTL operator and* $\sigma = \langle(\tau_i, \overline{b}_1^i, \ldots, \overline{b}_{\eta(\mathsf{op})}^i)\rangle_{i \in \mathbb{N}}$. *Let* c_0^{op} *be the initial state for evaluating* op *as defined in Section 4. Let* $(c_{i+1}^{\mathsf{op}}, o_i^{\mathsf{op}}) = s^{\mathsf{op}}(c_i^{\mathsf{op}}, \tau_i, \overline{b}^i)$, *for all* $i \in \mathbb{N}$. *Then* $\mathcal{I}(\mathsf{op}, \sigma_{\leq 0}, c_1^{\mathsf{op}}, o_0^{\mathsf{op}})$ *holds. Moreover, for any* $i \geq 0$,

$$\mathcal{I}(\mathsf{op}, \sigma_{\leq i}, c_{i+1}^{\mathsf{op}}, o_{\leq i}^{\mathsf{op}}) \Longrightarrow \mathcal{I}(\mathsf{op}, \sigma_{\leq i+1}, c_{i+2}^{\mathsf{op}}, o_{\leq i+1}^{\mathsf{op}}).$$

Theorem 1. *Let* op *be an MTL operator,* $\sigma = \langle(\tau_i, \overline{b}_1^i, \ldots, \overline{b}_{\eta(\mathsf{op})}^i)\rangle_{i \in \mathbb{N}}$ *be a stream with unbounded time-stamps, and* i *be a time-point. Then there exists a time-point* j *such that, using the notation of Lemma 1,* $|o_{\leq j}^{\mathsf{op}}| > i$ *and* $\mathsf{SOUND}_{\mathsf{op}}(\sigma_{\leq j}, o_{\leq j}^{\mathsf{op}})$.

5.2 Correctness and Space Complexity of the Multi-Head Monitor

To prove the correctness of the multi-head monitor for an MTL formula Φ over its atomic predicates P^Φ, we formulate an invariant $\mathcal{I}(\Phi,\rho,q^\Phi,\overline{p},o^\Phi)$ on the state q^Φ and the produced output o^Φ that holds after applying the step function δ^Φ on the stream ρ starting from the initial state q_0^Φ with the reading heads positioned at $\overline{p}$.

Atomic Predicate Let $\Phi = p \in P^\Phi$. The invariant asserts that sound output has been produced for the first $\overline{p}_1$ time-points: $\mathcal{I}(\Phi,\rho,q^\Phi,\overline{p},o^\Phi) \equiv |o^\Phi| = \overline{p}_1 \wedge \mathsf{SOUND}_\Phi(\rho,o^\Phi)$.

Recursive Formula Let $\Phi = \mathsf{op}(\varphi_1,\ldots,\varphi_{\eta(\mathsf{op})})$ and M^Φ be its multi-head monitor. The invariant asserts that (i) the states of M^Φ for all the subformulas and the state of evaluating the operator op are obtained when the outputs of all the multi-head monitors for the subformulas are used to construct a trace for evaluating the operator op, and (ii) the list of time-stamped Boolean values stored in M^Φ's state contains exactly the values that have been output by the operator op, except those that have already been output by M^Φ.

$$
\begin{aligned}
&\mathcal{I}(\Phi,\rho,(\langle q^{\varphi_i}\rangle_{i=1}^{\eta(\mathsf{op})},\langle\beta_i\rangle_{i=1}^{\eta(\mathsf{op})},c^{\mathsf{op}},vs),\overline{p}^1\cdots\overline{p}^{\eta(\mathsf{op})},o^\Phi) \equiv\\
&\quad(\Phi=\mathsf{op}(\varphi_1,\ldots,\varphi_{\eta(\mathsf{op})}) \wedge \textstyle\bigwedge_{i=1}^{\eta(\mathsf{op})} |\overline{p}^i| = \kappa^{\varphi_i}) \Longrightarrow \exists n>0.\ \exists\langle(\tau_k,\overline{b}^k)\rangle_{k<n}.\ \forall i\in[\eta(\mathsf{op})].\\
&\qquad(\beta_i\neq\bot \Longrightarrow \beta_i = \overline{b}_i^{n-1} \wedge \mathcal{I}(\varphi_i,\rho,q^{\varphi_i},\overline{p}^i,\langle(\tau_k,\overline{b}^k)\rangle_{k<n})) \wedge\\
&\qquad(\beta_i=\bot \Longrightarrow \mathcal{I}(\varphi_i,\rho,q^{\varphi_i},\overline{p}^i,\langle(\tau_k,\overline{b}^k)\rangle_{k<n-1})) \wedge\\
&\qquad\exists i\in[\eta(\mathsf{op})].\ \beta_i=\bot \wedge \mathcal{I}(\mathsf{op},\langle(\tau_k,\overline{b}^k)\rangle_{k<n-1},c^{\mathsf{op}},o^\Phi\cdot vs)
\end{aligned}
$$

Lemma 2. *Let Φ be any MTL formula and $M^\Phi = (P^\Phi,V,\kappa^\Phi,Q^\Phi,q_0^\Phi,\delta^\Phi)$ be a multi-head monitor for Φ as defined in Section 4. Let ρ be a stream. Let $\langle(q_i^\Phi,\overline{p}^i)\rangle_{i\in\mathbb{N}}$ be the computation of M^Φ on the stream ρ. Moreover, let $\langle o_i^\Phi\rangle_{i\in\mathbb{N}}$ be the output of M^Φ. Then $\mathcal{I}(\Phi,\rho,q_1^\Phi,\overline{p}^1,o_0^\Phi)$ holds. Moreover, for any $i\geq 0$,*

$$\mathcal{I}(\Phi,\rho,q_{i+1}^\Phi,\overline{p}^{i+1},o_{\leq i}^\Phi) \Longrightarrow \mathcal{I}(\Phi,\rho,q_{i+2}^\Phi,\overline{p}^{i+2},o_{\leq i+1}^\Phi).$$

Theorem 2. *Let Φ be any MTL formula and ρ a fixed stream. Let i be an arbitrary time-point. Then there exists some n such that, using the notation of Lemma 2, $o_{\leq n}^\Phi = \langle(t_k,b_k)\rangle_{k<l}$ with $l>i$ and $\mathsf{SOUND}_\Phi(\rho,o_{\leq n}^\Phi)$.*

Finally, we bound the space complexity of storing a state of a multi-head monitor. Our analysis relies on all lists in the monitor's state being run-length-encoded.

Theorem 3. *Let Φ be a formula and $M^\Phi = (P^\Phi,V,\kappa^\Phi,Q^\Phi,q_0^\Phi,\delta^\Phi)$ be a multi-head monitor for Φ as defined in Section 4. Let ρ be an arbitrary stream. Let $\langle(q_i^\Phi,\overline{p}^i)\rangle_{i\in\mathbb{N}}$ be the computation of M^Φ on the stream ρ. Then q_i^Φ can be represented as a string over the alphabet $\{(;);,;\mathsf{ff};\mathsf{tt};\bot;\varepsilon;0;1\}$ of length at most $32\cdot\|\Phi\|\cdot(2+\log_2 i+\log_2\tau_i)$.*

For a trace $\sigma_{\leq i}$, our monitor thus requires $\mathcal{O}(\|\Phi\|\cdot\log(i\cdot\tau_i))$ space, i.e., the space requirement grows logarithmically with the trace length and the observed time-stamps. The dependence on the trace length might possibly be avoided by using *sensing* reading heads [18], i.e., by allowing the computational model to determine if any two reading heads are currently at the same position. The dependence on the observed time-stamps could also be avoided by computing bounded (by the largest constant occurring in an interval) time-stamp differences. This should be possible to achieve with a bounded number of additional reading heads. We leave these potential improvements as future work.

6 Implementation and Evaluation

We have implemented the multi-head monitor in a tool called HYDRA [20], consisting of roughly 1000 lines of C++ code. Our implementation mirrors the overall structure of the multi-head monitor presented here and consists of C++ classes for monitoring atomic predicates and formulas with various top-level operators (Section 4.1), which also implement the evaluation of the top-level operator (Section 4.2). HYDRA implements some optimizations that are omitted in the presentation for the sake of simplicity. Most importantly, all reads are implemented imperatively, which allows the multi-head monitor's step function to always return a verdict (as opposed to an optional verdict as in Section 4.1). Finally, all lists are encoded using run-length encoding (Section 5.2).

We empirically validate our space complexity analysis and demonstrate HYDRA's superior time complexity by answering the following four research questions:

RQ1: *How does* HYDRA *scale with respect to the trace length?*
RQ2: *How does* HYDRA *scale with respect to the (temporal) size of the formula?*
RQ3: *How does* HYDRA *perform on inputs that trigger worst-case space complexity for online monitors?*
RQ4: *How does* HYDRA *perform compared to the state-of-the-art monitoring tools?*

To answer the above questions, we perform four experiments measuring HYDRA's average-case and worst-case time and space usage. Our analysis also includes the state-of-the-art monitors AERIAL [7] and MONPOLY [4, 5]. We use AERIAL's SAFA mode in the average-case experiments, and its EXPR mode in the worst-case experiments, as this choice exhibits the best performance for AERIAL. We remark that MONPOLY is an online monitor producing Boolean verdicts for all positions in the trace and its space complexity can thus only be bounded by a linear function in the trace length [4], i.e., it is not trace-length independent.

The average-case traces are produced by a pseudorandom trace generator for a predefined event rate er [3]. Each trace contains events with 100 different time-stamps. The time-stamp differences are distributed uniformly in $[\Delta]$, for a predefined Δ; in our experiments, we use $\Delta = 4$. The atomic predicates are generated as follows: (i) independently with probability $1 - \frac{1}{\Delta \cdot er}$, an atomic predicate $p_0, \ldots, p_3$ is included; (ii) independently with probability $\frac{1}{2}$, an atomic predicate $p_4, \ldots, p_{15}$ is included.

The average-case formulas are produced by a pseudorandom formula generator for a predefined size and maximum interval bounds. A formula Φ of size $s > 0$ is generated as follows: (i) if $s = 1$, then $\Phi = p$, for an atomic predicate $p \in \{p_0, \ldots, p_{15}\}$ chosen uniformly at random; (ii) if $s = 2$, a top-level unary operator op is selected uniformly at random; (iii) if $s \geq 3$, a top-level operator op is selected as follows: with probability $\frac{1}{2}$, the until operator is chosen, otherwise, the top-level operator op is chosen uniformly at random among the five remaining operators. If the top-level operator op has an interval, then the interval is generated as follows: (i) with probability $\frac{1}{4}$, the interval $[0,0]$ is chosen; (ii) with probability $\frac{1}{4}$, an interval $[0,r]$ is chosen with r distributed uniformly in $[\Delta]$, or $[\Delta] \cup \{\infty\}$, for a predefined Δ (in our experiments, we use $\Delta = 16$); (iii) with probability $\frac{1}{2}$, an interval $[l,r]$ is chosen with $l \in [\Delta]$ and $r \in \{l, \ldots, \Delta\}$, or $r \in \{l, \ldots, \Delta\} \cup \{\infty\}$, distributed uniformly at random. Finally, if the top-level operator op is a unary

Experiment	Formula size	Family size	Trace length	Scaling factor
1	25	10	20 000–200 000	1
2	2–50	10	50 000	1
3	25	10	50 000	1–10

Fig. 6: Summary of the experimental setup.

operator, a pseudorandom subformula φ of size $s-1$ is generated recursively, and if op is a binary operator, two pseudorandom subformulas of sizes $s_1, s-1-s_1$ are generated recursively, where $s_1 \in [s-2]$ is chosen uniformly at random.

The first three experiments measure the average-case behavior of the tools. The first experiment assesses the impact of increasing the trace length on a family of pseudorandom formulas of a fixed size. The second experiment assesses the impact of increasing the formula size on a trace of fixed length. The third experiment assesses the impact of scaling the intervals of pseudorandom formulas and time-stamps of pseudorandom traces by a constant factor. The first three experiments are summarized in Figure 6.

To answer RQ3, we conduct a fourth experiment where we consider a family of formulas $\langle \Phi_n \rangle_{n \in \mathbb{N}}$ that exhibits the worst-case space complexity for online monitoring when restricted to produce a *single* Boolean verdict for the first time-point. The formula Φ_n is defined over the set of atomic predicates $P^{\Phi_n} = \{e, p_1, \ldots, p_n\}$:

$$\Phi_n = \bigcirc_{[1,1]} (\neg e \,\mathsf{U}_{[0,0]}\, (\neg e \wedge \textstyle\bigwedge_{i=1}^{n} (p_i \Rightarrow \Box_{[0,0]} (e \Rightarrow p_i)) \wedge \bigwedge_{i=1}^{n} (\neg p_i \Rightarrow \Box_{[0,0]} (e \Rightarrow \neg p_i)))) .$$

The family of traces for some fixed $n \in \mathbb{N}$ on which the space complexity of online monitoring for Φ_n becomes at least 2^n bits looks as follows: the first event is an empty event with a time-stamp τ_0, then for each subset $X \in \mathcal{X} \subseteq 2^{P^{\Phi_n} \setminus \{e\}}$ of atomic predicates without e, we include an event with the atomic predicates X and a time-stamp $\tau_0 + 1$. Next, for some $X \subseteq P^{\Phi_n}$, we include an event with the atomic predicates $X \cup \{e\}$ and a time-stamp $\tau_0 + 1$. Finally, we include an empty event with a time-stamp $\tau_0 + 3$, so that the trace uniquely determines the Boolean verdict for the first time-point.

Intuitively, for an online monitor to decide if Φ_n is satisfied at the first time-point of a trace from the family of traces, it must remember the exact subset $\mathcal{X}$ to check if the set X of atomic predicates, which eventually appear with the atomic predicate e, belongs to $\mathcal{X}$. As there are 2^{2^n} different sets $\mathcal{X}$, we derive a lower bound of 2^n bits to store $\mathcal{X}$.

We remark that the top-level next operator in the formula Φ_n is used to make the formula trivially false on the worst-case traces described above at all time-points but the first one (recall that all analyzed monitors produce a stream of Boolean verdicts at each position in the trace). In particular, monitoring Φ_n on a worst-case trace described above does not achieve the upper bound of $O(2^{2^n+n})$ on AERIAL's space complexity [7].

To benchmark the time complexity in the worst-case experiment, we use traces of fixed length obtained by repeating a worst-case trace for some fixed $n \in \mathbb{N}$ with an increasing base time-stamp τ_0.

We run our experiments on an Intel Core i7-8550U, 1.80GHz computer with 32 GB RAM. We measure the tools' total execution time and maximal memory usage with the Unix time command. Having thoroughly tested the tools' outputs separately, we discard any output during the experiments to exclude the impact of disk writes on performance.

Each run is repeated 5 times to minimize the impact of the execution environment. Each unfilled data point in our plots shows the average for the tool invocations with the same input parameters. We omit the negligible standard deviations. Each filled data point shows the average over a collection of a tool's data points with the same x-coordinate. We include trend lines over the filled data points in all our plots.

Figures 7 and 8 show the results of our three average-case and the worst-case experiments. The plots in Figure 7 show time (on the left) and memory (on the right) scalability of all the tools. The uppermost row in Figure 7 answers RQ1; it confirms that in the average case both HYDRA and AERIAL are almost trace-length independent, while MONPOLY's space usage increases linearly with the trace length. The plots also show HYDRA's modest increase in execution time with respect to the increasing trace length compared to the state-of-the-art tools (RQ4). Overall, our tool treats random formulas more consistently than the other tools, which is reflected by the trend lines which fit the measurements very well. The other four plots in Figure 7 show the scalability of the tools with respect to formula size and temporal size (RQ2). Only MONPOLY's memory usage is impacted by these parameters and HYDRA outperforms all the tools. Finally, Figure 8 confirms our analytical findings from Section 5 and shows that, in practice, when monitoring the family of formulas $\langle \Phi_n \rangle_{n \in \mathbb{N}}$, HYDRA's worst-case space complexity is asymptotically better than any state-of-the-art online monitoring tool. The experiments may be reproduced using an artifact available at [20].

7 Conclusion

We proposed multi-head monitoring as a novel approach to analyzing traces. A multi-head monitor reads an input trace simultaneously at multiple positions and its reading heads move asynchronously. Following this paradigm, we designed a monitor for metric temporal logic that outputs a stream of Boolean verdicts. Our monitor is sound and complete and it substantially improves upon all previous algorithms with this output format. We implemented our algorithm in a prototype tool, HYDRA, and demonstrated that it reliably outperforms other monitors, including those that produce less intelligible output.

Multi-head monitoring fills a middle-ground between offline and online monitoring. It requires the input trace to be stored on a disk, as usual in offline monitoring. However, as its reading heads move only in one direction, this allows us to (1) delete the parts of the trace that was processed by all reading heads, and (2) add new events at the end of the trace, mimicking online monitoring.

As future work, we would like to further reduce our monitors' space complexity and achieve true trace-length independence, without the theoretically annoying (albeit practically harmless) logarithmic dependence. Our preliminary results suggest that this can be achieved at the expense of using exponentially many reading heads, instead of linearly many as used in this paper. We also plan to extend our results beyond MTL, e.g., to timed regular expressions [1] or prove the impossibility of this extension.

Acknowledgments. We thank the anonymous reviewers for their valuable suggestions on earlier drafts of this paper, which helped us to improve the presentation. This research is supported by the Swiss National Science Foundation grant "Big Data Monitoring" (167162).

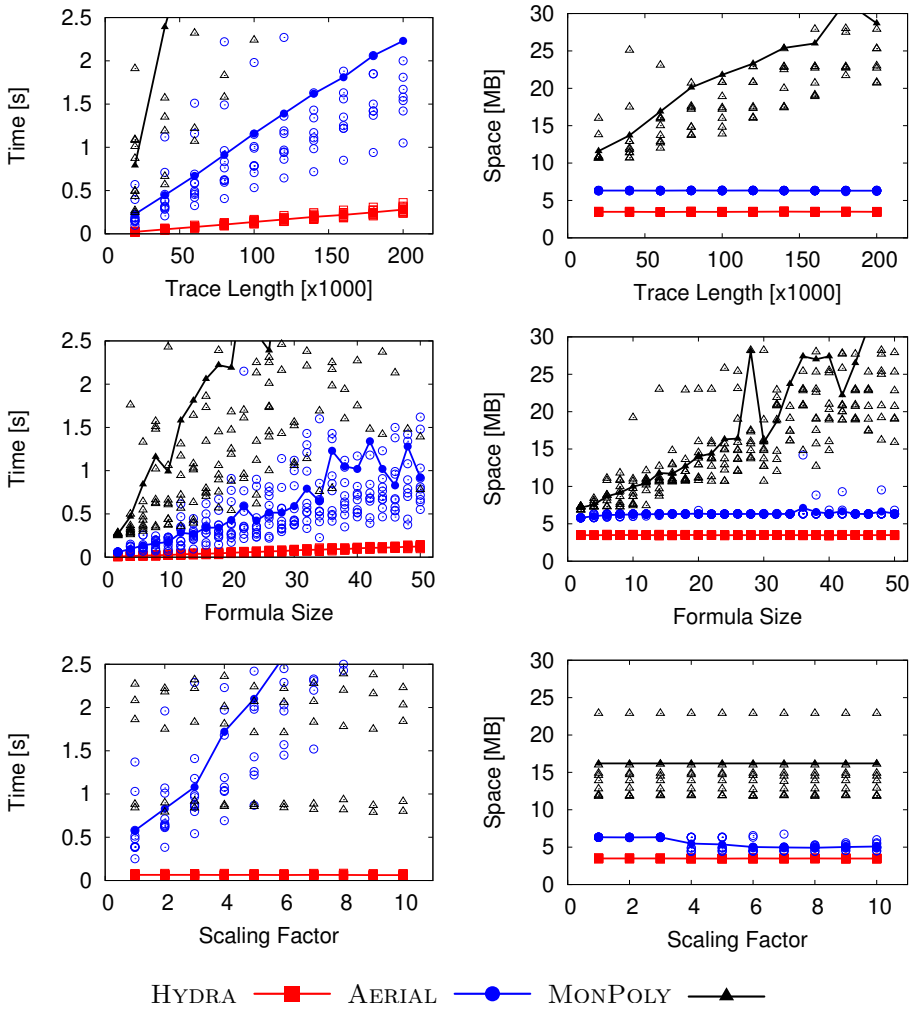

Fig. 7: Evaluation results for average-case behavior

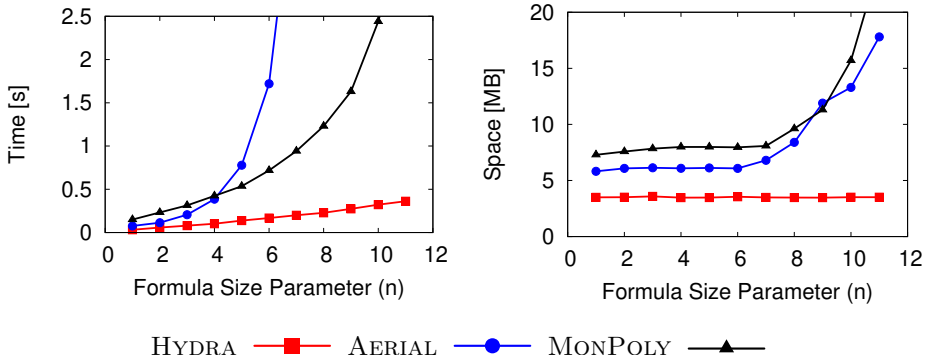

Fig. 8: Evaluation results for worst-case behavior

References

1. Asarin, E., Caspi, P., Maler, O.: Timed regular expressions. J. ACM **49**(2), 172–206 (2002)
2. Basin, D., Bhatt, B., Krstić, S., Traytel, D.: Almost event-rate independent monitoring. Form. Meth. Sys. Des. (2019)
3. Basin, D., Bhatt, B., Traytel, D.: Almost event-rate independent monitoring of metric temporal logic. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10206, pp. 94–112. Springer (2017)
4. Basin, D., Klaedtke, F., Müller, S., Zalinescu, E.: Monitoring metric first-order temporal properties. J. ACM **62**(2), 15 (2015)
5. Basin, D., Klaedtke, F., Zalinescu, E.: The MonPoly monitoring tool. In: Reger, G., Havelund, K. (eds.) RV-CuBES 2017. Kalpa Publications in Computing, vol. 3, pp. 19–28. EasyChair (2017)
6. Basin, D., Klaedtke, F., Zalinescu, E.: Algorithms for monitoring real-time properties. Acta Inf. **55**(4), 309–338 (2018)
7. Basin, D., Krstic, S., Traytel, D.: AERIAL: almost event-rate independent algorithms for monitoring metric regular properties. In: Reger, G., Havelund, K. (eds.) RV-CuBES 2017. Kalpa Publications in Computing, vol. 3, pp. 29–36. EasyChair (2017)
8. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. ACM Trans. Softw. Eng. Methodol. **20**(4), 14:1–14:64 (2011)
9. Convent, L., Hungerecker, S., Leucker, M., Scheffel, T., Schmitz, M., Thoma, D.: Tessla: Temporal stream-based specification language. In: Massoni, T., Mousavi, M.R. (eds.) SBMF 2018. LNCS, vol. 11254, pp. 144–162. Springer (2018)
10. D'Angelo, B., Sankaranarayanan, S., Sánchez, C., Robinson, W., Finkbeiner, B., Sipma, H.B., Mehrotra, S., Manna, Z.: LOLA: runtime monitoring of synchronous systems. In: TIME 2005. pp. 166–174. IEEE Computer Society (2005)
11. Falcone, Y., Krstić, S., Reger, G., Traytel, D.: A taxonomy for classifying runtime verification tools. In: Colombo, C., Leucker, M. (eds.) Runtime Verification. pp. 241–262. Springer, Cham (2018)
12. Faymonville, P., Finkbeiner, B., Schwenger, M., Torfah, H.: Real-time stream-based monitoring. CoRR **abs/1711.03829** (2017)
13. Finkbeiner, B., Sipma, H.: Checking finite traces using alternating automata. Formal Methods in System Design **24**(2), 101–127 (2004)
14. 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 (2018)
15. Gray, J., Shenoy, P.J.: Rules of thumb in data engineering. In: Lomet, D.B., Weikum, G. (eds.) ICDE 2000. pp. 3–10. IEEE Computer Society (2000)
16. Havelund, K., Roşu, G.: Synthesizing monitors for safety properties. In: Katoen, J., Stevens, P. (eds.) TACAS 2002. LNCS, vol. 2280, pp. 342–356. Springer (2002)
17. Ho, H., Ouaknine, J., Worrell, J.: Online monitoring of metric temporal logic. In: Bonakdarpour, B., Smolka, S.A. (eds.) RV 2014. LNCS, vol. 8734, pp. 178–192. Springer (2014)
18. Ibarra, O.H.: A note on semilinear sets and bounded-reversal multihead pushdown automata. Inf. Process. Lett. **3**(1), 25–28 (1974)
19. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-Time Syst. **2**(4), 255–299 (1990)
20. Raszyk, M., Basin, D., Krstić, S., Traytel, D.: HYDRA. `https://bitbucket.org/krle/hydra` (2019)
21. Raszyk, M., Basin, D., Traytel, D.: From Nondeterministic to Multi-Head Deterministic Finite-State Transducers. In: Baier, C., Chatzigiannakis, I., Flocchini, P., Leonardi, S. (eds.)

ICALP 2019. LIPIcs, vol. 132, pp. 127:1–127:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2019)
22. Roşu, G., Havelund, K.: Rewriting-based techniques for runtime verification. Automated Software Engineering **12**(2), 151–197 (2005)
23. 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 (2018)
24. Thati, P., Roşu, G.: Monitoring algorithms for metric temporal logic specifications. Electr. Notes Theor. Comput. Sci. **113**, 145–162 (2005)