Almost Event-Rate Independent Monitoring of Metric Dynamic Logic

David Basin, Srđan Krstić, and Dmitriy Traytel

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

Abstract. Linear temporal logic (LTL) and its quantitative extension metric temporal logic (MTL) are standard languages for specifying system behaviors. Regular expressions are an even more expressive formalism in the non-metric setting and several extensions of LTL, including the recently proposed linear dynamic logic (LDL), offer regular-expression-like constructs. We extend LDL with past operators and quantitative features. The resulting *metric dynamic logic* (MDL) offers the quantitative temporal conveniences of MTL while increasing its expressiveness. We develop and evaluate an online monitoring algorithm for MDL whose space-consumption is *almost event-rate independent*—a notion that characterizes monitors that scale to high-velocity event streams.

1 Introduction

Runtime monitoring is a well-established paradigm for verifying system properties by comparing system events against a specification formalizing which event sequences are allowed. Numerous monitoring algorithms have been developed for both the *online* setting, where events are monitored in real-time, as they occur, and for the *offline* setting, where the monitor analyzes events stored in logs. In this paper we address the *online monitoring problem*: Given a stream of events and a property formalized in a formal specification language, identify all the points in the stream that violate the property.

A standard specification language is linear temporal logic (LTL) with past and future temporal operators, which express qualitative temporal constraints like "*A* must follow (or be preceded by) *B*." Metric temporal logic (MTL) extends LTL to formulate quantitative temporal constraints like "*A* must be followed by *B* within an hour." There are numerous semantics for MTL and we work here with a discrete, point-based time model (Section 2). This model faithfully captures the imprecision of physical clocks and is algorithmically easier to handle than the dense, interval-based model [8].

We recently developed a monitor for MTL based on dynamic programming that scales to high-volume and high-velocity event streams [5]. The central notion in this work is *almost event-rate independence* (Section 2): the monitor's space consumption may only depend logarithmically on the number of events per time-unit. This requirement is stronger than the traditionally used trace-length independence, as it accounts for a high velocity of events. Future temporal constraints make it hard for a monitor to be event-rate independent. This is because the monitor may be unable to output verdicts for events, since the verdicts may depend on future events and the number of such events can exceed the event rate. To overcome this problem, our monitor outputs *equivalence*

verdicts in addition to the standard Boolean verdicts. With an equivalence verdict, the monitor indicates that it does not know the exact verdict for an event, but it knows that the verdict will be equivalent to the verdict of another (also presently unknown) event.

LTL falls short of expressing all regular languages. For example, one cannot express that some event occurs at every other position in a stream. This limitation is often a problem in practice [29] and carries over to the point-based semantics of MTL [9]. A realistic example that cannot be expressed in MTL is that, within the next day, an action is approved and executed and the approval event happens before the execution event.

To overcome this limitation, researchers have developed numerous, more expressive extensions of LTL and MTL by adding regular-expression-like constructs to the language [30]. This resulted in specification languages such as the industrially standardized property specification language [29] (PSL), regular linear temporal logic [22, 23] (RLTL), and more recently linear dynamic logic [14] (LDL). We survey these and other languages in Section 6. All of them lack some of the features that make MTL an attractive choice: either its support for past operators or its quantitative features.

Our contributions in this paper are as follows. First, we propose metric dynamic logic (MDL), an extension of LDL with past operators and quantitative features (Section 3). Second, we develop, implement, and optimize an almost event-rate independent monitoring algorithm for MDL that substantially extends our earlier algorithm for MTL and Antimirov's results on partial derivatives of regular expressions [3] (Section 4). Finally, we empirically evaluate our algorithm and show that it outperforms state-of-the-art monitoring tools for MTL and timed regular expressions (Section 5).

2 Metric Temporal Logic and Event-Rate Independence

We present our discrete, point-based time model [8] below, briefly introduce metric temporal logic (MTL) [21], and the notion of event-rate independence for monitors [5].

Let Σ be a set of atomic propositions. An *event* is a pair (τ, π), where $\tau \in \mathbb{N}$ is a *time-stamp* and $\pi \subseteq \Sigma$ is a set of propositions that are true at that event. An *event stream* is an infinite sequence of events $\rho = \langle(\tau_0, \pi_0), (\tau_1, \pi_1), (\tau_2, \pi_2), \ldots\rangle$ with monotonically increasing time-stamps: $\tau_i \leq \tau_{i+1}$ for all $i \in \mathbb{N}$. We write Δ_i for the non-negative time-stamp difference $\tau_{i+1} - \tau_i$. We call the indices in ρ *time-points*, i.e., event (τ_i, π_i) occurs at time-point i. Moreover, we require that time progresses: for all time-stamps τ there exists a time-point i with $\tau_i > \tau$. The *event rate* $\mathsf{er}_\rho(\tau)$ of a stream ρ at time-stamp τ is the number of time-points with that time-stamp, i.e., $\mathsf{er}_\rho(\tau) = |\{i \mid \tau_i = \tau\}|$.

Metric temporal logic (MTL) [21] is a commonly used language to formulate properties of event streams. Its syntax is given by the grammar

$$\varphi = p \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc_I \varphi \mid \bullet_I \varphi \mid \varphi \,\mathcal{S}_I\, \varphi \mid \varphi \,\mathcal{U}_I\, \varphi,$$

where $p \in \Sigma$ and $I \in \mathbb{I}$. Here, $\mathbb{I}$ denotes the set of non-empty intervals over $\mathbb{N}$. We write $[a, b]$ for the interval $\{x \in \mathbb{N} \mid a \leq x \leq b\}$, where $a \in \mathbb{N}$, $b \in \mathbb{N} \cup \{\infty\}$, and $a \leq b$. For an interval I and $n \in \mathbb{N}$, we define $I - n$ to be the interval $\{x - n \mid x \in I\} \cap \mathbb{N}$ and I^- to be the set of intervals $\{I - n \mid n \in \mathbb{N}\}$, which is always finite. Along with the standard Boolean operators, MTL includes the past temporal operators $\bullet_I$ (*previous*) and $\mathcal{S}_I$ (*since*) and the future temporal operators $\bigcirc_I$ (*next*) and $\mathcal{U}_I$ (*until*), which may be nested freely. A

formula is interpreted with respect to a fixed event stream ρ at a time-point i. As this is standard, we only show the $\mathcal{U}_I$ case (omitting the fixed event stream ρ):

$$i \models \varphi \,\mathcal{U}_I\, \psi \text{ iff } j \models \psi \text{ for some } j \geq i \text{ with } \tau_j - \tau_i \in I \text{ and } k \models \varphi \text{ for all } i \leq k < j.$$

A (*traditional online*) *monitor* for MTL takes as input an MTL formula φ and events from ρ, one at a time, and outputs a stream of *Boolean verdicts* $\langle 0 \models \varphi, 1 \models \varphi, 2 \models \varphi, \ldots \rangle$. Note that for future formulas (i.e., those containing $\mathcal{U}_I$ or $\bigcirc_I$), the monitor cannot in general output the verdict at time-point i before it has seen some of the events at future time-points $j > i$. Taking this to the extreme, for the formula $p \,\mathcal{U}_{[0,\infty]}\, q$ with atomic propositions p and q and the event stream ρ with $\tau_i = i$ and $\pi_i = \{p\}$ for all i, the monitor can never output a verdict at time-point 0 because it always sees only a finite prefix of the stream. Moreover, even with bounded future intervals, the number of events that a monitor might need to wait for is unbounded, since non-increasing time-stamps are allowed in our model. It follows that the memory consumption of any monitor depends on the event rate as it must allocate at least one bit for each time-point at which it has not yet produced an output.

In earlier work [5], we developed an MTL monitor that is robust against increasing event rates. Our *almost event-rate independent* monitor differs from traditional monitors in its output: additionally to outputting Boolean verdicts, it may output *equivalence verdicts* between time-points $i \equiv j$. Such an equivalence means that the Boolean verdicts at both i and j are presently unknown (since they depend on future events). However, independent of the future events, they are guaranteed to be equal. Moreover, the monitor may output verdicts out of order with respect to time-points. For the above example, for $p \,\mathcal{U}_{[0,\infty]}\, q$ the monitor would forever output equivalence verdicts $\langle 0 \equiv 1, 0 \equiv 2, \ldots \rangle$. These changes in how the monitor outputs verdicts are crucial and result in monitor's space requirement being *almost* event-rate independent. Namely, while reading a sequence of time-points with identical time-stamps, the monitor keeps the current value of the time-stamp, as well as the offset between the current and the first time-point with the same time-stamp. The monitor therefore requires space only logarithmic in the event-rate, i.e. the number of bits needed to encode the above-mentioned offset.

3 Metric Dynamic Logic

In this section, we introduce metric dynamic logic (MDL). This logic extends LDL [14] with past temporal operators and time intervals associated with temporal formulas. MDL's syntax is defined by the following grammar, where $p \in \Sigma$ denotes an atomic proposition, and $I \in \mathbb{I}$ denotes a non-empty interval.

$$\psi = p \mid \neg\psi \mid \psi \vee \psi \mid \langle r \rangle_I \, \psi \mid \psi \, {}_I\langle r \rangle \qquad\qquad r = \star \mid \psi? \mid r + r \mid r \cdot r \mid r^*$$

Aside from Boolean operators, MDL contains dynamic modalities like the metric future diamond operator $\langle r \rangle_I \, \varphi$ which expresses that formula φ is true at some future time-point with a time difference bounded by the interval I and that the regular expression r matches the portion of the event stream from the current point up to that future time-point. The past diamond operator $\varphi \, {}_I\langle r \rangle$ expresses the same property about a past time-point. Regular expressions in MDL match portions of the event stream, i.e., words over 2^Σ. The expression $\star$ matches any character and $\varphi?$ matches the empty word starting

at time-point i if the formula φ holds at i. Moreover, $+$, $\cdot$, and * are the standard alternation, concatenation, and (Kleene) star operators. The semantics of formulas and regular expressions is defined by mutual induction. A formula is interpreted over a fixed event stream $\rho = \langle(\tau_i, \pi_i)\rangle_{i\in\mathbb{N}}$ and a position $i \in \mathbb{N}$. The semantics of a regular expression r is given by a relation $\mathcal{R}(r) \subseteq \mathbb{N} \times \mathbb{N}$ that contains pairs of time-points (i, j) with $i \leq j$ such that the sequence $\pi_i, \ldots, \pi_j$ (or $\pi_j, \ldots, \pi_i$ for past) from the fixed ρ matches r.

$$
\begin{aligned}
i &\models p \text{ iff } p \in \pi_i & \mathcal{R}(\star) &= \{(i, i+1) \mid i \in \mathbb{N}\}\\
i &\models \neg\varphi \text{ iff } i \not\models \varphi & \mathcal{R}(\varphi?) &= \{(i, i) \mid i \models \varphi\}\\
i &\models \varphi_1 \vee \varphi_2 \text{ iff } i \models \varphi_1 \text{ or } i \models \varphi_2 & \mathcal{R}(r+s) &= \mathcal{R}(r) \cup \mathcal{R}(s)\\
i &\models \langle r\rangle_I\, \varphi \text{ iff } j \models \varphi \text{ for some } j \geq i & \mathcal{R}(r \cdot s) &= \{(i, k) \mid \exists j.\ (i, j) \in \mathcal{R}(r)\\
&\quad \text{with } \tau_j - \tau_i \in I \text{ and } (i, j) \in \mathcal{R}(r) & &\qquad \text{and } (j, k) \in \mathcal{R}(s)\}\\
i &\models \varphi\, {}_I\langle r\rangle \text{ iff } j \models \varphi \text{ for some } j \leq i & \mathcal{R}(r^*) &= \{(i, i) \mid i \in \mathbb{N}\} \cup\\
&\quad \text{with } \tau_i - \tau_j \in I \text{ and } (j, i) \in \mathcal{R}(r) & &\quad \{(i_0, i_k) \mid \exists i_1, \ldots, i_{k-1}.\ (i_j, i_{j+1}) \in \mathcal{R}(r)\\
& & &\qquad \text{for all } 0 \leq j < k\}
\end{aligned}
$$

We employ the usual syntactic sugar for additional Boolean constants and operators: $\mathit{true} = p \vee \neg p$, $\mathit{false} = \neg\mathit{true}$, and $\varphi \wedge \psi = \neg(\neg\varphi \vee \neg\psi)$. The ability to arbitrarily nest the negation operator allows us to define the metric future and past box operators $[r]_I\, \varphi$ and $\varphi\, {}_I[r]$ as $\neg(\langle r\rangle_I\, \neg\varphi)$ and $\neg(\neg\varphi\, {}_I\langle r\rangle)$, respectively. We use the abbreviations $\langle\varphi\rangle_I\, \psi$ and $\psi\, {}_I\langle\varphi\rangle$ for $\langle\varphi? \cdot \star\rangle_I\, \psi$ and $\psi\, {}_I\langle\star \cdot \varphi?\rangle$. We perform the same implicit *cast* of a formula φ to the regular expression $\varphi? \cdot \star$ in the context of a future regular expression (or $\star \cdot \varphi?$ in the context of a past regular expression) for any formula that occurs as an argument to one of the $+$, $\cdot$, and * constructors. For example, $\langle\varphi^*\rangle_I\, \psi$ abbreviates $\langle(\varphi? \cdot \star)^*\rangle_I\, \psi$.

For an MDL formula φ, let $\mathsf{SF}(\varphi)$ denote the set of its subformulas defined as usual. Note that $\varphi \in \mathsf{SF}(\varphi)$. We extend this set to the set of *interval-skewed subformulas* $\mathsf{ISF}(\varphi) = \mathsf{SF}(\varphi) \cup \{\psi\, {}_J\langle r\rangle \mid \psi\, {}_I\langle r\rangle \in \mathsf{SF}(\varphi), J \in I^-\} \cup \{\langle r\rangle_J\, \psi \mid \langle r\rangle_I\, \psi \in \mathsf{SF}(\varphi), J \in I^-\}$, which contains all temporal formulas with the same structure as existing temporal subformulas of φ, except with intervals shifted by constants.

Theorem 1. *For every MTL formula there exists an equivalent MDL formula.*

Proof. We prove this constructively by defining a syntactic translation ξ:

$$
\xi(p) = p; \quad \xi(\varphi \vee \psi) = \xi(\varphi) \vee \xi(\psi); \quad \xi(\neg\varphi) = \neg\xi(\varphi); \quad \xi(\bigcirc_I\, \varphi) = \langle\star\rangle_I\, \xi(\varphi);
$$
$$
\xi(\varphi\, \mathcal{U}_I\, \psi) = \langle\xi(\varphi)^*\rangle_I\, \xi(\psi); \quad \xi(\bullet_I\, \varphi) = \xi(\varphi)\, {}_I\langle\star\rangle; \quad \xi(\varphi\, \mathcal{S}_I\, \psi) = \xi(\psi)\, {}_I\langle\xi(\varphi)^*\rangle.
$$

Given an MTL formula φ and a fixed stream ρ, one can prove that $\forall i.\ i \models \xi(\varphi) \Leftrightarrow i \models \varphi$ by induction on the structure of φ. (Note that we overload the notation for satisfiability $\models$ for both logics.) We show the proof only for $\mathcal{U}_I$. The other cases follow similarly.

$$
\begin{aligned}
& i \models \xi(\varphi\, \mathcal{U}_I\, \psi) \overset{\text{def. } \xi}{\Leftrightarrow} i \models \langle\xi(\varphi)^*\rangle_I\, \xi(\psi) \overset{\text{cast}}{\Leftrightarrow} i \models \langle(\xi(\varphi)? \cdot \star)^*\rangle_I\, \xi(\psi)\\
& \overset{\text{def. } \models}{\Leftrightarrow} j \models \xi(\psi) \text{ for some } j \geq i \text{ with } \tau_j - \tau_i \in I \text{ and } (i, j) \in \mathcal{R}((\xi(\varphi)? \cdot \star)^*)\\
& \overset{\text{IH } \psi}{\Leftrightarrow} j \models \psi \text{ for some } j \geq i \text{ with } \tau_j - \tau_i \in I \text{ and } (i, j) \in \mathcal{R}((\xi(\varphi)? \cdot \star)^*)\\
& \overset{\text{def. } \mathcal{R}}{\Leftrightarrow} j \models \psi \text{ for some } j \geq i \text{ with } \tau_j - \tau_i \in I \text{ and } k \models \xi(\varphi) \text{ for all } i \leq k < j\\
& \overset{\text{IH } \varphi}{\Leftrightarrow} j \models \psi \text{ for some } j \geq i \text{ with } \tau_j - \tau_i \in I \text{ and } k \models \varphi \text{ for all } i \leq k < j\\
& \overset{\text{def. } \models}{\Leftrightarrow} i \models \varphi\, \mathcal{U}_I\, \psi
\end{aligned}
$$

□

Since MDL extends LDL, it can express any ω-regular language [14]. For instance, the property "the event a occurs at every even position" can be expressed as $[(\mathit{true} \cdot \mathit{true})^*]\,a$. This property cannot be expressed by any LTL or MTL formula. Similarly, the property "both b and c occur within the next two time-units and b occurs before c" cannot be expressed in MTL with point-based semantics [9]. It can, however, be expressed in MDL with the formula $\langle \mathit{true}^* \cdot b \cdot \mathit{true}^* \rangle_{[0,2]}\, c$.

4 The Monitoring Algorithm

Our almost event-rate independent monitor for MTL [5] uses dynamic programming in a way that extends the classic monitor for past-only LTL by Havelund and Roşu [18]. For MDL we follow the same approach. We recapitulate the algorithm in the simpler setting of MTL (Subsection 4.1) and afterwards we use incremental reasoning about dynamic modalities (Subsection 4.2) to extend the algorithm to MDL (Subsection 4.3). Finally, we present several important performance optimizations (Subsection 4.4).

4.1 An Almost Event-Rate Independent Monitor for MTL

At its core, the MTL monitor relies on an alternative recursive definition of the satisfiability of temporal formulas. For example, for a fixed stream ρ, $\mathcal{U}_I$'s definition reads:

$$i \models \varphi\, \mathcal{U}_I\, \psi \text{ iff } a = 0 \text{ and } i \models \psi, \text{ or } \Delta_i \leq b,\, i \models \varphi, \text{ and } i+1 \models \varphi\, \mathcal{U}_{I-\Delta_i}\, \psi, \qquad (1)$$

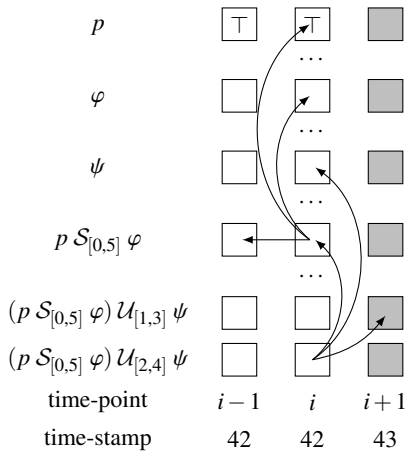
where $I = [a,b]$. The key observation is that the satisfiability of $\varphi\, \mathcal{U}_I\, \psi$ at time-point i is fully determined by the satisfiability of φ and ψ at the current time-point i and the satisfiability of the interval-skewed formula $\varphi\, \mathcal{U}_{I-\Delta_i}\, \psi$ at the next time-point $i+1$, along with some interval boundary checks. For $\mathcal{S}_I$, a symmetric characterization refers to an interval-skewed formula at the previous time-point $i-1$.

Figure 1 (left) illustrates these dependencies as arrows for verdicts at time point i for the formula $(p\, \mathcal{S}_{[0,5]}\, \varphi)\, \mathcal{U}_{[2,4]}\, \psi$ and its subformula $p\, \mathcal{S}_{[0,5]}\, \varphi$. Note that the future dependencies can, in general, only be resolved after having seen the event at time-point $i+1$. Our monitor treats such future dependencies symbolically as Boolean variables. To monitor the formula Φ, the algorithm stores a Boolean expression for each interval-skewed subformula $\varphi \in \mathsf{ISF}(\Phi)$ in an array **curr**, ordered such that, for any formula φ at index k, each of its proper subformulas occurs at a lower index $l < k$. We write Φ_k for the formula occurring at index k and sometimes use formulas synonymously as indices, e.g., by writing **curr**$[\varphi]$ for the **curr**'s entry at position k, given that $\varphi = \Phi_k$. We use Boolean expressions in negation normal form, defined inductively as:

$$\mathit{bexp} = \bot \mid \top \mid \mathsf{Var}\, \mathbb{N} \mid \neg \mathsf{Var}\, \mathbb{N} \mid \mathit{bexp} \wedge \mathit{bexp} \mid \mathit{bexp} \vee \mathit{bexp}.$$

Negation $\neg$ is applied to arbitrary Boolean expressions by pushing it down to the leaves.

With each arriving event, the array **curr** is updated following Equation 1 (and analogous equations). The variables in expressions at time-point i represent pointers into the monitor's array **curr** *after* processing the event at time-point $i+1$. Instead of using pointers to the past time-point $i-1$, the monitor directly uses the expressions from the array at time-point $i-1$ to build from them new expressions at time-point i.




```
now := −1
off := 0
hist := {}
prev := ⊥^n
curr := (Now ⊥)^n

step (τ, π) =
  Δ = τ − now
  for k = 0, …, n − 1
    prev[k] := eval_Δ(curr[k])
  hist := filter_verdicts
    ({((now, off), prev[Φ])} ∪
     {(j, subst (λk. prev[k]) c) | (j, c) ∈ hist})
  now := τ
  off := if Δ = 0 then off + 1 else 0
  for k = 0, …, n − 1
    curr[k] := progress (Φ_k, Δ, π)
```

Fig. 1: Example excerpt of the MTL monitor's state (left) and pseudocode (right)

For future formulas, there is an additional complication: before the monitor has seen the time-stamp at position $i+1$, it cannot know which of the interval-skewed future formulas to refer to. Therefore, we work with so called *future expressions* defined as $\mathit{fexp} = \mathsf{Now}\ \mathit{bexp} \mid \mathsf{Later}\ (\mathbb{N} \to \mathit{bexp})$, where the parameter of Later is the time-stamp difference between the time-points $i+1$ and i. The functions $\wedge_{\mathit{fexp}}$, $\vee_{\mathit{fexp}}$, $\neg_{\mathit{fexp}}$ lift Boolean operators to future expressions while propagating Boolean values eagerly, for example, by simplifying $\mathsf{Now}\ \top \vee_{\mathit{fexp}} x$ to $\mathsf{Now}\ \top$ and $\mathsf{Now}\ \bot \vee_{\mathit{fexp}} x$ to x. Given a time-stamp difference Δ, a future expression evaluates to a Boolean expression: $\mathsf{eval}_\Delta(\mathsf{Now}\ c) = c$ and $\mathsf{eval}_\Delta(\mathsf{Later}\ f) = f(\Delta)$.

We are now set to describe our almost event-rate independent monitor. Figure 1 (right) depicts its (OCaml-like) pseudocode. The monitor's state consists of five variables initialized as shown, where n is the number of interval-skewed subformulas of Φ. To denote their mutability, we write them in boldface. The variable **now** is the current time-stamp and together with it the natural number **off** identifies the current time-point. Note that we represent time-points as pairs (τ, k), where the second component is an offset into a block of time-points labeled with time-stamp τ. The history **hist** is a set of pairs of Boolean expressions and time-points (again stored as time-stamp-offset pairs). It contains all time-points at which no verdict was output so far, since the verdict depends on future events. The variable **curr** is the array of length n of future expressions for all interval-skewed subformulas at the current time-point. The variable **prev** is another array of length n of Boolean expressions that belong to the previous time-point. The monitor updates its state using the step function for each incoming event (τ, π).

The step function first computes the time-stamp difference Δ between τ and the previous time-stamp stored in **now**. It uses Δ to translate future expressions from **curr** to Boolean expressions and store them in **prev**, thereby discarding any old expression stored in **prev**. Next the history **hist** is updated. This step is the key to obtaining almost event-rate independence. The variables of all Boolean expressions stored in the history refer to the latest seen time-point. To maintain this invariant, we first update all expres-

sions in the history by substituting their variables (pointing to what used to be in **curr** before the call of step) with the actual Boolean expressions contained in **curr** (that is now stored in **prev**). The substitution is performed by the higher-order function subst, whose definition is omitted. Moreover, the expression $\mathbf{prev}[\Phi]$ is added as a new element to the history. Then, the function filter_verdicts (whose formal definition is omitted, too) performs two verdict output steps. First, it iterates over the history and removes (and outputs as Boolean verdicts) all Boolean expressions equivalent to $\top$ or $\bot$. Second, it finds all pairwise equivalent pairs of expressions from the history and for each such pair it removes the expression with the larger time-point from the history and outputs an equivalence verdict. By guaranteeing that only semantically different Boolean expressions in at most n variables are contained in the history, the monitor is almost event-rate independent, as only offset's size depends on the event rate. Note that if the second step were omitted, we would obtain a standard monitor that is event-rate dependent. Finally, after a trivial update of **now** and **off**, the progress function fills the **curr** array with new future expressions. We only show its definition for $\mathcal{U}_I$, following Equation 1.

$$\begin{aligned}\mathsf{progress}\ (\varphi\,\mathcal{U}_{[a,b]}\,\psi, \Delta, \pi) = (\mathsf{Now}\ (a=0) \wedge_{fexp} \mathbf{curr}[\psi]) \vee_{fexp} \\ \mathsf{Later}\ (\lambda\Delta'.\ \Delta' \leq b \wedge \mathsf{eval}_{\Delta'}(\mathbf{curr}[\varphi]) \wedge \mathsf{Var}\ (\varphi\,\mathcal{U}_{[a,b]-\Delta'}\,\psi))\end{aligned}$$

For other formulas, the update works similarly, for example, by inspecting the π argument for atomic propositions. Note that the parameter Δ is not used for $\mathcal{U}_I$, as it is the time-stamp difference between the current and previous time-point. In contrast, Δ' from the Later argument is the appropriate time-stamp difference for $\mathcal{U}_I$. It will be instantiated to a concrete value after the next event (including its time-stamp) is received. To refer to the satisfiability of $\varphi\,\mathcal{U}_{[a,b]-\Delta'}\,\psi$ at position $i+1$ as stipulated by Equation 1, we use the Var constructor applied to (the index of) $\varphi\,\mathcal{U}_{[a,b]-\Delta'}\,\psi$.

In the presented algorithm, only the progress function and the computation of interval-skewed subformulas are specific to MTL. In the following subsections we will replace these ingredients to obtain a monitor for MDL.

4.2 Derivatives of Dynamic Modalities

To build on ideas from the above algorithm for MDL, we need an alternative recursive definition of the past and future modalities that refer only to the $(i-1)$st and $(i+1)$st time-point. For a fixed stream ρ, the following characterization holds.

$$i \models \langle r \rangle_I\, \varphi \text{ iff } a=0,\ \varepsilon_i(r),\ \text{and } i \models \varphi,\ \text{or } \Delta_i \leq b \text{ and } i+1 \models \langle \delta_i(r) \rangle_{I-\Delta_i}\, \varphi \tag{2}$$

$$i \models \varphi\, {}_I\langle r \rangle \text{ iff } a=0,\ \varepsilon_i(r),\ \text{and } i \models \varphi,\ \text{or } \Delta_{i-1} \leq b \text{ and } i-1 \models \varphi\, {}_{I-\Delta_{i-1}}\langle \eth_i(r) \rangle \tag{3}$$

Here, $I=[a,b]$, $\varepsilon_i(r)$ is the Boolean denoting whether $(i,i) \in \mathcal{R}(r)$ (i.e., r matches the empty word), and $\delta_i(r)$ is the Brzozowski derivative [10] of the regular expression r (and $\eth_i(r)$ its symmetric counterpart). For plain regular expressions, the Brzozowski derivative $\delta_c(r)$ computes a regular expression whose language is the left quotient $\{w \mid cw \in L(r)\}$ of the input expression's language $L(r)$ by a given letter c. One may view the derivative as a deterministic automaton whose states are labeled by regular expressions, whereby reading c in a state r takes the automaton to $\delta_c(r)$. For MDL formulas, the time-point i takes the place of the given letter c and "reading c" means querying a

subformula's satisfaction at time-point i. The inductive definitions of ε, δ, and $\mathfrak{d}$ follow. They all are implicitly parameterized by the fixed ρ.

$$\begin{array}{rll}
\varepsilon_i(\star) = \bot & \delta_i(\star) = \top? & \mathfrak{d}_i(\star) = \top? \\
\varepsilon_i(\varphi?) = i \models \varphi & \delta_i(\varphi?) = \bot? & \mathfrak{d}_i(\varphi?) = \bot? \\
\varepsilon_i(r+s) = \varepsilon_i(r) \vee \varepsilon_i(s) & \delta_i(r+s) = \delta_i(r) + \delta_i(s) & \mathfrak{d}_i(r+s) = \mathfrak{d}_i(r) + \mathfrak{d}_i(s) \\
\varepsilon_i(r \cdot s) = \varepsilon_i(r) \wedge \varepsilon_i(s) & \delta_i(r \cdot s) = \delta_i(r) \cdot s + & \mathfrak{d}_i(r \cdot s) = r \cdot \mathfrak{d}_i(s) + \\
 & \quad \varepsilon_i(r)? \cdot \delta_i(s) & \quad \varepsilon_i(s)? \cdot \mathfrak{d}_i(r) \\
\varepsilon_i(r^*) = \top & \delta_i(r^*) = \delta_i(r) \cdot r^* & \mathfrak{d}_i(r^*) = r^* \cdot \mathfrak{d}_i(r)
\end{array}$$

The definition of δ is faithful to Brzozowski's original definition. Note that $\mathcal{R}(\top?) = \{(i,i) \mid i \in \mathbb{N}\}$, $\mathcal{R}(\bot?) = \{\}$, and $\varepsilon_i(r)? \cdot \delta_i(s)$ is equivalent to if $\varepsilon_i(r)$ then $\delta_i(s)$else $\bot?$, which is more commonly used to define Brzozowski derivatives. The equations for the right derivative $\mathfrak{d}$ are symmetric for the concatenation and star cases. Thereby, $\mathfrak{d}$ matches the regular expression from right to left. It is easy to verify that the Equations 2 and 3 hold for those definitions by structural induction on the regular expression r.

How can we integrate Equations 2 and 3 into our monitor? Since the equations refer to the satisfiability of formulas $\langle \delta_i(r) \rangle_{I-\Delta_i} \varphi$ and $\varphi_{I-\Delta_{i-1}} \langle \mathfrak{d}_i(r) \rangle$, those formulas must occur in our interval-skewed subformulas array. In other words, we must monitor $\langle \delta_i(r) \rangle_I \varphi$ simultaneously to $\langle r \rangle_I \varphi$ (and all their interval-skewed variants). But by the same reasoning, $\langle \delta_j(\delta_i(r)) \rangle_I \varphi$ must be monitored, too. Hence, we must monitor all formulas that can be reached by repeatedly computing the derivative of the original subexpressions. Fortunately, Brzozowski has proved that the set of expressions reachable by repeatedly taking derivatives is finite, provided that one rewrites expressions to a normal form with respect to associativity, commutativity, and idempotence (ACI) of the $+$ constructor. Unfortunately, the number of all such Brzozowski derivatives is exponential in the size of the initial expression r. This is hardly surprising, since regular expressions are exponentially more concise than deterministic automata and the set of all Brzozowski derivatives represents exactly the set of states of a deterministic automaton.

With the size of the array exponential in the size of the input formula, we would still obtain an almost event-rate independent monitor, but not one that is very time-efficient. We can do better by resorting to nondeterministic automata, which are as concise as regular expressions. The equivalent of the Brzozowski derivative for nondeterministic automata are Antimirov's partial derivatives of regular expressions [3]. Instead of computing only one successor expressions, a partial derivative computes a set of expressions, analogous to the transition function of a nondeterministic automaton. The partial derivative ∂ and its symmetric counterpart $\mathfrak{d}$ are defined inductively as follows.

$$\begin{array}{rl}
\partial_i(\star) = \{\top?\} & \mathfrak{d}_i(\star) = \{\top?\} \\
\partial_i(\varphi?) = \{\} & \mathfrak{d}_i(\varphi?) = \{\} \\
\partial_i(r+s) = \partial_i(r) \cup \partial_i(s) & \mathfrak{d}_i(r+s) = \mathfrak{d}_i(r) \cup \mathfrak{d}_i(s) \\
\partial_i(r \cdot s) = \partial_i(r) \odot s \cup \varepsilon_i(r)? \odot \partial_i(s) & \mathfrak{d}_i(r \cdot s) = r \odot \mathfrak{d}_i(s) \cup \varepsilon_i(s)? \odot \mathfrak{d}_i(r) \\
\partial_i(r^*) = \partial_i(r) \odot r^* & \mathfrak{d}_i(r^*) = r^* \odot \mathfrak{d}_i(r)
\end{array}$$

Here, $\odot$ lifts $\cdot$ to sets of expressions. Overloading notation we have $r \odot X = \{r \cdot s \mid s \in X\}$ and $X \odot r = \{s \cdot r \mid s \in X\}$.

Partial derivatives enjoy nice properties: the sum of all expressions in $\partial_i(r)$ is equivalent to $\delta_i(r)$. Moreover, the number of different expressions reachable from r by repeated application of the partial derivative is bounded by $n+1$, where n is r's size [3]. In other words, partial derivatives convert a regular expression of size n into a nondeterministic automaton of size $n+1$. The states of this automaton are labeled by the $n+1$ reachable expressions, and these are exactly the ones our monitor will need to keep track of to follow the following partial derivative variant of Equations 2 and 3.

$$i \models \langle r \rangle_I \varphi \text{ iff } a = 0, \varepsilon_i(r), \text{ and } i \models \varphi, \text{ or } \Delta_i \leq b \text{ and } \bigvee_{s \in \partial_i(r)} i+1 \models \langle s \rangle_{I-\Delta_i} \varphi \quad (4)$$

$$i \models \varphi \,{}_I\langle r \rangle \text{ iff } a = 0, \varepsilon_i(r), \text{ and } i \models \varphi, \text{ or } \Delta_i \leq b \text{ and } \bigvee_{s \in \mathfrak{G}_i(r)} i-1 \models \varphi \,{}_{I-\Delta_i}\langle s \rangle \quad (5)$$

Those equations follow by structural induction on r, using the distributivity of the diamond operators over $+$, i.e., $i \models \langle r+s \rangle_I \varphi \Leftrightarrow i \models \langle r \rangle_I \varphi \vee \langle s \rangle_I \varphi$.

We must know which regular expressions to keep track of before actually running the monitor. We can overapproximate this set by replacing $\varepsilon_i(\varphi?)$ with $\top$ instead of $i \models \varphi$. Then both ε and ∂ become independent of the fixed ρ and i. The number of expressions in the approximation is still bounded by the size of the original expression (+1).

4.3 An Almost Event-Rate Independent Monitor for MDL

The recursive equations are a useful blueprint. However, we cannot use the ε and partial derivative operations directly, since they rely on the satisfiability of subformulas that are arguments of _?. But the monitor might not know at time-point i whether some subformula φ is satisfied, since φ could refer to the future. However, the monitor does know the symbolic future expression $\mathbf{curr}[\varphi]$ denoting φ's satisfiability at i. This knowledge allows us to compute the ε symbolically as a future expression:

$$\begin{array}{lll} \varepsilon(\star) = \mathsf{Now}\ \bot & \varepsilon(r+s) = \varepsilon(r) \vee_{fexp} \varepsilon(s) & \\ \varepsilon(\varphi?) = \mathbf{curr}[\varphi] & \varepsilon(r \cdot s) = \varepsilon(r) \wedge_{fexp} \varepsilon(s) & \varepsilon(r^*) = \mathsf{Now}\ \top \end{array}$$

Unlike previous definitions of ε, this definition does not depend on any fixed stream ρ.

For the symbolic version of partial derivatives, an additional complication arises. The above definition of ∂ computes a set of expressions and relies on the Boolean verdicts of certain subformulas. When we work with future expressions, we do not know for sure whether to include the partial derivatives of s when computing the partial derivatives $r \cdot s$, since $\varepsilon(r)$ is not a Boolean value but a future expression. Therefore, the derivative's result must be something like a decision tree with sets of regular expressions as leaves. Equations 4 and 5 illustrate, however, that in fact we are not interested in expressions as such, but rather in expressions wrapped into some fixed past or future diamond operators and ultimately the satisfiability of the resulting formulas. Satisfiability queries are much easier to represent using our machinery (as future expressions) than decision tree with sets of regular expressions as leaves. Using continuation-passing-style programming, we obtain the symbolic partial derivative ∂ (and the symmetric $\mathfrak{G}$) that computes a future expression corresponding to $\bigvee_{s \in \partial_i(r)} i+1 \models \langle s \rangle_{I-\Delta_i} \varphi$. The function ∂ takes two arguments: a regular expression r and a continuation function κ that is supposed to wrap a regular expressions in a past or future diamond operator and create a variable pointing to the corresponding formula in the $(i+1)$st time-point.

$$
\begin{aligned}
\partial\,(\star, \kappa) &= \kappa(\top?) & \mathfrak{G}\,(\star, \kappa) &= \kappa(\top?)\\
\partial\,(\varphi?, \kappa) &= \mathsf{Now}\ \bot & \mathfrak{G}\,(\varphi?, \kappa) &= \mathsf{Now}\ \bot\\
\partial\,(r+s, \kappa) &= \partial\,(r, \kappa) \vee_{fexp} \partial\,(s, \kappa) & \mathfrak{G}\,(r+s, \kappa) &= \mathfrak{G}\,(r, \kappa) \vee_{fexp} \mathfrak{G}\,(s, \kappa)\\
\partial\,(r \cdot s, \kappa) &= \partial\,(r, \lambda t.\ \kappa\,(t \cdot s)) \vee_{fexp} & \mathfrak{G}\,(r \cdot s, \kappa) &= \mathfrak{G}\,(s, \lambda t.\ \kappa\,(r \cdot t)) \vee_{fexp}\\
&\quad (\varepsilon(r) \wedge_{fexp} \partial\,(s, \kappa)) & &\quad (\varepsilon(s) \wedge_{fexp} \mathfrak{G}\,(r, \kappa))\\
\partial\,(r^*, \kappa) &= \partial\,(r, \lambda t.\ \kappa\,(t \cdot r^*)) & \mathfrak{G}\,(r^*, \kappa) &= \mathfrak{G}\,(r, \lambda t.\ \kappa\,(r^* \cdot t))
\end{aligned}
$$

Observe how the continuation is altered in the concatenation and star cases. The standard partial derivative first calculates recursively the set $\partial_i(r)$ before concatenating s to each expression in $\partial_i(r)$. Here, we extend the continuation κ to perform the concatenation via $\lambda t.\ \kappa\,(t \cdot s)$ at the leaves of the recursion tree.

Finally, we define the progress function for MDL. The function takes as input a subformula φ, the time-stamp difference Δ between the current and the previous time-point, and the set of currently true atomic predicates π. Moreover, it assumes, that the array **prev** contains the Boolean expressions denoting the satisfiability at the previous time-point for all interval-skewed variants of φ and that the array **curr** contains the future expression denoting the satisfiability at the current time-point for all subformulas of φ. It computes a future expression denoting the satisfiability of φ at the current time-point.

$$
\begin{aligned}
&\mathsf{progress}\ (\varphi, \Delta, \pi) = \mathsf{case}\ \varphi\ \mathsf{of}\\
&\quad \mid p \qquad\quad \Rightarrow \mathsf{Now}\ (p \in \pi)\\
&\quad \mid \neg\psi \qquad \Rightarrow \neg_{fexp}\ \mathbf{curr}[\psi]\\
&\quad \mid \psi_1 \vee \psi_2 \ \Rightarrow \mathbf{curr}[\psi_1] \vee_{fexp} \mathbf{curr}[\psi_2]\\
&\quad \mid \langle r \rangle_{[a,b]}\ \psi \Rightarrow (\mathsf{Now}\ (a = 0) \wedge_{fexp} \varepsilon(r) \wedge_{fexp} \mathbf{curr}[\psi]) \vee_{fexp}\\
&\qquad\qquad \mathsf{Later}\ (\lambda\Delta'.\ \Delta' \leq b \wedge \mathsf{eval}_{\Delta'}(\partial\,(r, \lambda s.\ \mathsf{Now}\ (\mathsf{Var}\ (\langle s \rangle_{[a,b]-\Delta'}\ \psi)))))\\
&\quad \mid \psi\ _{[a,b]} \langle r \rangle \Rightarrow (\mathsf{Now}\ (a = 0) \wedge_{fexp} \varepsilon(r) \wedge_{fexp} \mathbf{curr}[\psi]) \vee_{fexp}\\
&\qquad\qquad (\mathsf{Now}\ (\Delta \leq b) \wedge_{fexp} \mathfrak{G}\,(r, \lambda s.\ \mathsf{subst}\ (\lambda k.\ \mathbf{curr}[k])\ \mathbf{prev}[\psi\ _{[a,b]-\Delta} \langle s \rangle]))
\end{aligned}
$$

Only the cases for the diamond operators are interesting. They implement Equations 4 and 5. The first disjunct is the same for both the future and past, since it covers the case when the regular expression matches the empty word. For the future diamond, the second disjunct is a Later future expression, since it does not know the time-stamp difference between the current and the next time-point. The argument to Later is the conjunction of the Boolean from the interval boundary test with the symbolic partial derivative ∂ (evaluated to a Boolean expression using the abstracted time-difference Δ'). The continuation κ wraps a given regular expression into a future diamond formula and creates a variable denoting the satisfiability of the resulting formula at the next time-point. For the past diamond, the second disjunct is a conjunction of the interval boundary test and the right derivative $\mathfrak{G}$. The continuation function for the latter wraps a given regular expression into a past diamond formula and retrieves the Boolean expression denoting the formula's satisfaction at the previous time-point from **prev**. The variables in this expression point to the current time-point. The function subst updates those variables to the next time-point by accessing **curr**.

Using this progress function in the algorithm shown in Subsection 4.1 results in our almost event-rate independent monitor for MDL.

Theorem 2. *Our monitor is sound: for an MDL formula Φ and any prefix of the event-stream ρ, whenever it outputs a Boolean verdict b at time-point i, then $i \models \varphi \Leftrightarrow b$ and whenever it outputs an equivalence verdict between time-points i and j, then $i \models \varphi \Leftrightarrow j \models \varphi$. Moreover, the monitor outputs each verdict as soon as it has seen enough events to compute the verdict and its space consumption is almost event-rate independent.*

Proof. For lack of space, we refer to the similar proof in our previous work for the MTL monitor [5, Section 5.3]. Since we have only modified the progress function, the only part that must be adjusted is the calculation given there in the proof of Lemma 2. □

To process an event, our monitor solves several NP-complete problem instances. However, the Boolean equivalences arising in practice are simple and tractable (Section 5).

4.4 Optimizations

AERIAL [1] is a concise OCaml implementation of our monitoring algorithms for MTL and MDL. In previous work [5], we reported on a PolyML implementation of the homonymous MTL monitor. The OCaml successor employs several optimizations, used in both logics, that substantially improve its performance for MTL.

Following Havelund and Rosu [18], our arrays store only expressions for temporal subformulas. The expressions for the Boolean connectives are computed on the fly accessing the **curr** array for temporal subformulas and are not stored.

A central operation in our monitor is the access to the **curr** and **prev** arrays based on a subformula's index. This raises the question of how to efficiently retrieve a subformula's index. Searching the array of subformulas is of course not an efficient option (although our previous PolyML implementation did just that). A standard more efficient solution would be to use a hash table, but some preliminary experiments showed that computing hashes of formulas very quickly becomes a bottleneck, too. Instead, AERIAL stores the indices for all subformulas directly in the formulas as annotations on the constructors. In the progress function, we then still need to compute the index of a formula based on the indices of its subformulas and the interval. However, the stored index allows us to avoid computing the indices of the subformulas recursively. For MTL it is easy to compute the exact position of a temporal formula $\varphi \,\mathcal{U}_I\, \psi$ based on this information by using a canonical order on subformulas as in $[\ldots, \varphi, \ldots, \psi, \ldots, \varphi \,\mathcal{U}_{I-2}\, \psi, \varphi \,\mathcal{U}_{I-1}\, \psi, \varphi \,\mathcal{U}_I\, \psi]$: the index of $\varphi \,\mathcal{U}_I\, \psi$ is just the index of ψ increased by b, where $I = [a, b]$. For MDL this is still problematic, since the derivatives are hard to align in a predictable way. We resort to memoizing the derivative functions ∂ and $\mathfrak{G}$ to compute a symbolic expression not only in the verdicts at the $(i+1)$st time-point but also in the verdicts at the ith time-point. Thereby, the search for indices happens only once during the initialization of the monitor and not in the progress function. The progress function merely needs to substitute the symbolic variables pointing to the ith time-point with the current values of **curr**.

Another crucial question is how to represent Boolean expressions. AERIAL offers the choice between two representations: the one reported in the paper and one based on binary decision diagrams (BDDs). For the former, it is important to keep the expressions small. To achieve this, we normalize expressions with respect to the associativity, commutativity, and idempotence of $\wedge$ and $\vee$, as well as Boolean tautologies such as

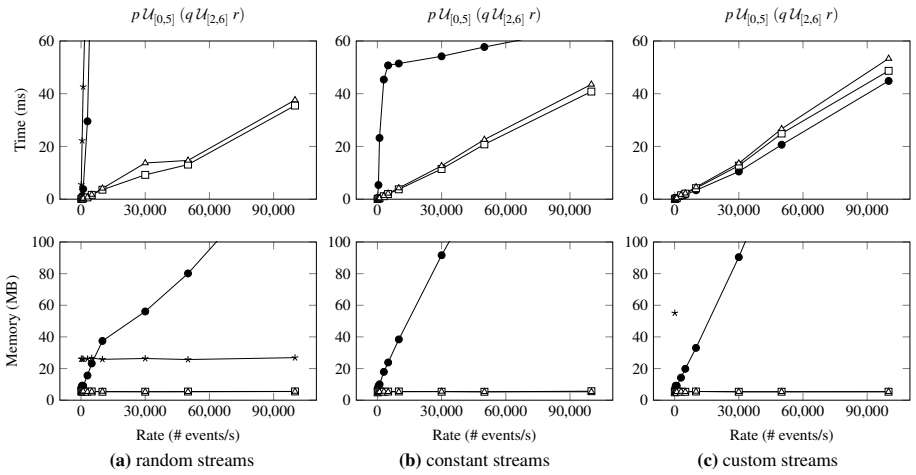


Fig. 2: Time (top) and memory (bottom) usage with respect to the event rate

$\top \wedge c = c$. Boolean expressions offer a low-cost substitution operation, but are expensive to check for equivalence. In fact, the equivalence check translates expressions into BDDs. The BDD version always works with the BDD representation thereby avoiding the costly translation. In contrast, the substitution operation becomes more expensive. In our experiments, the Boolean expression version outperformed the BDD version.

5 Evaluation

In our evaluation, we distinguish between two variants of our tool: AERIAL MDL and AERIAL MTL and we compare them with MONPOLY [6, 7], a state-of-the-art monitor for *metric first-order temporal logic* (*MFOTL*) and MONTRE [27, 28], a state-of-the-art matcher for *timed regular expressions* (*TRE*). We aim to answer the following questions:
Q1: *How does* AERIAL MDL *scale with respect to the event rate?*
Q2: *How does* AERIAL MDL *scale with respect to the size of the monitored formula?*
Q3: *Does* AERIAL MDL *perform better then state-of-the-art tools?*

We ran all our experiments on a 2.6 GHz quad-core Intel Core i7 processor with 16 GB RAM. We measure the tools' total execution time and maximal memory usage via the Unix time command. We use GNU Parallel [24] both to generate the streams and run the four tools. Our evaluation can be divided into two parts: analyzing the tools' scalability with respect to (1) the event rate and (2) the size of the monitored formula. In the first part, we monitor a fixed set of formulas ($\Diamond_{[0,5]} p$, $p\,\mathcal{U}_{[0,5]}\,q$, $p\,\mathcal{U}_{[0,5]}\,(q\,\mathcal{S}_{[2,6]}\,r)$, and $p\,\mathcal{U}_{[0,5]}\,(q\,\mathcal{U}_{[2,6]}\,r)$) over streams with an increasing event rate. In the second part, we monitor formulas of increasing size over a set of streams with a fixed event rate.

The finite prefixes of the synthetic streams used in the experiments span 100 time units, with the event rate (the number of time-points labeled with the same time-stamp) ranging from 100 to 100,000 on average ($\pm 10\%$) per stream. The streams contain three events, $\Sigma = \{p, q, r\}$, and their distribution depends on three different generation

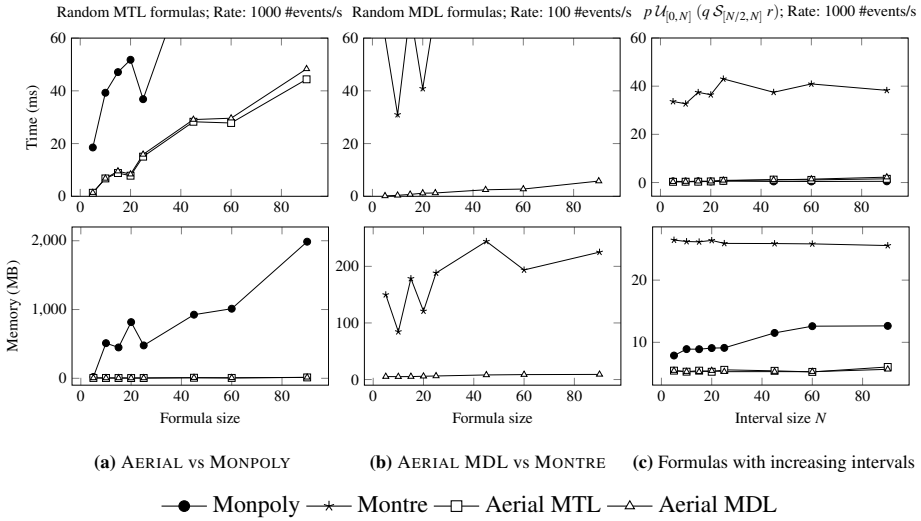
strategies: *random*, *constant*, and *custom*. The random generation strategy uses the uniform probability distribution for each event. Under the constant strategy, each stream has identical events at every time-point. Since $|\Sigma| = 3$, there are exactly eight distinct constant streams, including the stream with all empty time-points, and the stream with all events at all time-point. Constant streams are useful to test edge cases in the monitors' implementations and often trigger worst-case monitor execution time and memory usage. Finally, the custom generation strategy uses event probability distributions tailored to the particular formulas. For example, for the formula $\Diamond_{[0,5]} p$, the probability of p occurring is very small, which makes the tools wait longer before producing a verdict.

For each generation strategy and event rate, we generated eight different streams. We also converted each of these streams to the format supported by MONTRE. Since MONTRE supports only strictly monotonic time-stamps, our conversion simulated large event rates by increasing the time granularity of the MONTRE streams. Namely, time-points that share the same time-stamp in the original stream are converted into a sequence of time-points with time-stamps that strictly increase by one time unit. The granularity of the time unit in the converted stream is proportional to the event rate.

We have developed a random MTL and MDL formula generator parameterized by the formula's size using QCheck [2], a QuickCheck implementation for OCaml. For our evaluation, we measure the formula size simply as the number of subformulas and we separately check the scalability of the monitors with respect to different interval sizes in the formulas. Note that the tools can only be compared on commonly supported logical fragments. Propositional MTL with both future and past is the common fragment supported by AERIAL MDL, AERIAL MTL, and MONPOLY, while future MDL formulas in positive normal form belong to the common fragment of AERIAL MDL and MONTRE. To supply the correct input to each tool, the formula generator implements a translation from MTL to fragments of MDL, MFOTL, and a translation from MDL to TRE. The translation to TRE also scales the intervals appropriately to match the different time granularity of MONTRE-compliant streams. In contrast to monitors that report violations, MONTRE outputs all parts of the stream that match a TRE pattern. Hence, to properly compare the tools, we negate the formulas inputed to the other monitors. We generated ten arbitrary formulas for each formula size ranging from 5 to 100.

We set a timeout for each monitoring run to be 100 seconds, coinciding with the streams' time span. Moreover, we enforce the following disqualification scheme: If a tool times out for all the traces with the same event rate (or for all formulas with the same size) it will not be invoked for traces with larger event rates (or, respectively, for formulas with larger size). When computing the average values, a timeout counts as 100 seconds (although the actual run may take longer) and skews the curves to converge to the 100 second margin. Therefore, in our plots we only show average values below 50 seconds. The memory used before a timeout contributes to the average memory usage.

Figure 2 shows the results of the first part of the evaluation classified according to the stream generation strategy. We show the plots for formula $p\,\mathcal{U}_{[0,5]}\,(q\,\mathcal{U}_{[2,6]}\,r)$, which had least favorable outcome for AERIAL MDL. Each data point in the plots represents a value averaged over eight different streams with a fixed event rate. To answer Q1, we note that the space consumption of both versions of AERIAL is constant. As expected, the increasing memory consumption of other tools significant increases the overall pro-



(a) AERIAL vs MONPOLY **(b)** AERIAL MDL vs MONTRE **(c)** Formulas with increasing intervals

Fig. 3: Time (top) and memory (bottom) usage with respect to the formula size

cessing time. MONTRE was almost immediately disqualified in the case of constant and custom streams. To answer Q2, we note that, even for the largest formula, AERIAL MDL requires only 12MB of space compared to MONPOLY, which uses almost 2GB (see Figure 3a). These experiments were performed on random traces and random formulas and each data point is an average value over eight random traces and ten random formulas with the same size. During the experiment indicated in Figure 3b, MONTRE timed out 392 times which is over 50% of all its invocations. Figure 3c shows that all the tools are mostly unaffected by the size of the time interval N in the formula $p\,\mathcal{U}_{[0,N]}\,(q\,\mathcal{S}_{[N/2,N]}\,r)$. Finally, we observe that in all tests AERIAL MDL performs only marginally worse than AERIAL MTL, while supporting a strictly more expressive logic.

6 Related work

We survey temporal logics similar to MDL and we describe and compare monitoring algorithms for those logics most closely related to ours.

MTL is a well known temporal logic for specifying real-time properties. Thati and Roşu [25] provide an event-rate independent, dynamic programming monitoring algorithm for MTL based on derivatives of MTL formulas. However, their algorithm implements a non-standard semantics of MTL, truncated to finite traces. When handling future temporal operators, the algorithm outputs verdicts without looking at future events, which can potentially alter the verdicts. Computing verdicts this way defeats the purpose of (top-level) future operators: An *until* that is not satisfied at the current time-point, but only at the next one, is reported as a violation. Basin et al. [7, 8] introduce techniques to handle MTL and metric first-order temporal logic with bounded future operators, adhering to the standard non-truncated semantics for future formulas. Their monitor uses a queue to postpone the evaluation of a future formula to a time-point at which all needed information is present. Their algorithm stores in the worst case all

time-points while it waits and its space complexity is therefore linear in the event rate. Our previous work [5] is, up to now, the only almost event-rate independent monitoring algorithm for MTL with the standard semantics for future operators.

Dynamic LTL [19] (DLTL) is one of the earliest attempts to extend LTL's until operator to express all ω-regular languages. Leucker and Sánchez [22] propose regular LTL (RLTL) that further improves on DLTL by allowing regular expressions to be nested arbitrarily as LTL subformulas. RLTL's power operator is also more suitable for extensions that can handle past. Sánchez and Leucker [23] extend RLTL with past operators and show that it can be translated into a 2-way alternating parity automaton with size linear in the size of the RLTL formula. But 2-way automata are not ideally suited for the online monitoring of high-velocity event streams, and removing bidirectionality incurs an exponential blowup [20]. Dax et al. [11] propose a similar extension of the property specification language [29] (PSL) with additional past operators, called regular temporal logic (RTL). They translate RTL formulas into nondeterministic Büchi automata with the worst-case size doubly exponential in the size of the RTL formula.

More recently, De Giacomo and Vardi [14] revisited this problem for the finite-trace semantics and introduced linear dynamic logic (LDL_f). It was inspired by the well-known propositional dynamic logic (PDL) [17], but its semantics closely resemble LTL. The authors do not discuss the past diamond operator and do not provide a monitoring algorithm. However, they do provide a general translation from LDL formulas to alternating finite state automata that employs partial derivatives in a similar fashion as our monitoring algorithm. De Giacomo et al. [12, 13] provide a direct translation from LDL_f formulas to nondeterministic automata that are more suitable for monitoring. Faymonville et al. [15, 16] propose an extension of LDL called parametric linear dynamic logic (PLDL) that can specify quantitative temporal constraints. However, PLDL does not support past operators and its point-based time model does not include time-stamps, but rather time is implicitly encoded in the time-points. In this work, we chose to extend LDL with metric features (as opposed to PSL, RTL, or RLTL) due to its convenient and elegant mutual nesting of logical formulas and regular expressions.

Asarin et al. [4] introduce *timed regular expressions* (TRE) and prove their equivalence to timed automata. Additionally, some of the authors propose offline [27] and online [28] pattern matching algorithms for TRE, implemented as an open source tool called MONTRE [26]. Although TRE was originally defined over both discrete point-based and dense interval-based time models, MONTRE assumes the latter model.

7 Conclusion

We have introduced metric dynamic logic (MDL), a new logic that combines the expressive power of regular expressions with the ability to reference the past and the future in both a quantitative and qualitative way, as in metric temporal logic. Moreover, we have extended our previous almost event-rate independent monitoring algorithm for MTL to support MDL. Our evaluation shows that our implementation of this monitor, AERIAL, outperforms other state-of-the-art monitoring tools.

As future work, we would like to extend the presented ideas to the first-order setting, where events may carry data and formulas may quantify over the data's domain.

Acknowledgment. Felix Klaedtke pointed us to a motivating example of a property not expressible in MTL. Bhargav Bhatt, Domenico Bianculli, and three anonymous reviewers provided helpful feedback on earlier drafts of this paper. Srđan Krstić is supported by the Swiss National Science Foundation grant Big Data Monitoring (167162). The authors are listed alphabetically.

References

[1] Aerial: An almost event-rate independent monitor for metric temporal logic. `https://bitbucket.org/traytel/aerial` (2017)
[2] QCheck: QuickCheck inspired property-based testing for OCaml. `https://github.com/c-cube/qcheck` (2017)
[3] Antimirov, V.: Partial derivatives of regular expressions and finite automaton constructions. Theor. Comput. Sci. 155(2), 291–319 (1996)
[4] Asarin, E., Caspi, P., Maler, O.: Timed regular expressions. J. ACM 49(2), 172–206 (2002)
[5] Basin, D., Bhatt, B., Traytel, D.: Almost event-rate indepedent monitoring of metric temporal logic. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10206, pp. 94–112. Springer (2017)
[6] Basin, D.A., Klaedtke, F., Müller, S., Pfitzmann, B.: Runtime monitoring of metric first-order temporal properties. In: Hariharan, R., Mukund, M., Vinay, V. (eds.) FSTTCS 2008. LIPIcs, vol. 2, pp. 49–60. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2008)
[7] Basin, D.A., Klaedtke, F., Müller, S., Zălinescu, E.: Monitoring metric first-order temporal properties. J. ACM 62(2), 15:1–15:45 (2015)
[8] Basin, D.A., Klaedtke, F., Zălinescu, E.: Algorithms for monitoring real-time properties. In: Khurshid, S., Sen, K. (eds.) RV 2011. LNCS, vol. 7186, pp. 260–275. Springer (2012)
[9] Bouyer, P., Chevalier, F., Markey, N.: On the expressiveness of TPTL and MTL. Inf. Comput. 208(2), 97–116 (2010)
[10] Brzozowski, J.A.: Derivatives of regular expressions. J. ACM 11(4), 481–494 (1964)
[11] Dax, C., Klaedtke, F., Lange, M.: On regular temporal logics with past. Acta Inf. 47(4), 251–277 (2010)
[12] De Giacomo, G., De Masellis, R., Grasso, M., Maggi, F.M., Montali, M.: LTLf and LDLf monitoring: A technical report. CoRR abs/1405.0054 (2014)
[13] De Giacomo, G., De Masellis, R., Grasso, M., Maggi, F.M., Montali, M.: Monitoring business metaconstraints based on LTL and LDL for finite traces. In: Sadiq, S.W., Soffer, P., Völzer, H. (eds.) BPM 2014. LNCS, vol. 8659, pp. 1–17. Springer (2014)
[14] De Giacomo, G., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: Rossi, F. (ed.) IJCAI-13. pp. 854–860. AAAI Press (2013)
[15] Faymonville, P., Zimmermann, M.: Parametric linear dynamic logic. In: Peron, A., Piazza, C. (eds.) Proceedings 5th GandALF 2014. EPTCS, vol. 161, pp. 60–73 (2014)
[16] Faymonville, P., Zimmermann, M.: Parametric linear dynamic logic. Inf. Comput. 253, 237–256 (2017)
[17] Fischer, M.J., Ladner, R.E.: Propositional dynamic logic of regular programs. J. Comput. Syst. Sci. 18(2), 194–211 (1979)
[18] 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)
[19] Henriksen, J.G., Thiagarajan, P.: Dynamic linear time temporal logic. Ann. Pure Appl. Logic 96(1), 187–207 (1999)
[20] Kapoutsis, C.A.: Removing bidirectionality from nondeterministic finite automata. In: Jedrzejowicz, J., Szepietowski, A. (eds.) MFCS 2005. LNCS, vol. 3618, pp. 544–555. Springer (2005)

[21] Koymans, R.: Specifying real-time properties with metric temporal logic. Real-Time Syst. 2(4), 255–299 (1990)
[22] Leucker, M., Sánchez, C.: Regular linear temporal logic. In: Jones, C.B., Liu, Z., Woodcock, J. (eds.) ICTAC 2007. LNCS, vol. 4711, pp. 291–305. Springer (2007)
[23] Sánchez, C., Leucker, M.: Regular linear temporal logic with past. In: Barthe, G., Hermenegildo, M.V. (eds.) VMCAI 2010. LNCS, vol. 5944, pp. 295–311. Springer (2010)
[24] Tange, O.: GNU Parallel - the command-line power tool. ;login: The USENIX Magazine 36(1), 42–47 (Feb 2011), http://www.gnu.org/s/parallel
[25] Thati, P., Roşu, G.: Monitoring algorithms for metric temporal logic specifications. Electr. Notes Theor. Comput. Sci. 113, 145–162 (2005)
[26] Ulus, D.: Montre: A tool for monitoring timed regular expressions. arXiv preprint arXiv:1605.05963 (2016)
[27] Ulus, D., Ferrère, T., Asarin, E., Maler, O.: Timed pattern matching. In: Legay, A., Bozga, M. (eds.) FORMATS 2014. LNCS, vol. 8711, pp. 222–236. Springer (2014)
[28] Ulus, D., Ferrère, T., Asarin, E., Maler, O.: Online timed pattern matching using derivatives. In: Chechik, M., Raskin, J.F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 736–751. Springer (2016)
[29] Vardi, M.Y.: From Church and Prior to PSL. In: Grumberg, O., Veith, H. (eds.) 25 Years of Model Checking - History, Achievements, Perspectives. LNCS, vol. 5000, pp. 150–171. Springer (2008)
[30] Wolper, P.: Temporal logic can be more expressive. Inform. Control 56(1/2), 72–99 (1983)