Efficient Offline Monitoring for Dynamic Metric Temporal Logic

Konstantinos Mamouras[0000−0003−1209−7738]

Rice University, Houston TX 77005, USA
mamouras@rice.edu

Abstract. We propose an efficient offline monitoring algorithm for properties written in DMTL (Dynamic Metric Temporal Logic), a temporal formalism that combines MTL (Metric Temporal Logic) with regular expressions. Our algorithm has worst-case running time that is is polynomial in the size of the temporal specification and linear in the length of the input trace. In particular, our monitoring algorithm needs time $O(m^3 \cdot n)$, where m is the size of the DMTL formula and n in the length of the input trace.

Keywords: offline monitoring · metric temporal logic · regular expressions · dynamic logic · nondeterministic automata

1 Introduction

Monitoring is a lightweight verification technique for checking at runtime that a program or system behaves as desired. It has proved to be effective for evaluating the correctness of the behavior of complex systems, where static verification is computationally intractable. This includes cyber-physical systems (CPSs) that consist of both computational and physical processes. A *monitor* is a program that observes the execution trace of the system and emits values that indicate events of interest or other actionable information.

It is common to specify monitors using special-purpose formalisms such as variants of temporal logic [65], regular expressions [45], and other domain-specific programming languages [13]. In the context of cyber-physical systems, logics that are interpreted over signals are frequently used. This includes Metric Temporal Logic (MTL) [47] and Signal Temporal Logic (STL) [51].

Linear Temporal Logic (LTL) [65] cannot express all regular properties and is therefore expressively weaker than regular expressions. MTL inherits this lack of expressiveness. Several extensions of LTL have been considered in order to make it expressively complete for the class of regular properties [77,21].

We focus here on properties that are interpreted over discrete-time signals and are specified using an extension of MTL with regular expressions. We call this formalism ***Dynamic Metric Temporal Logic*** or DMTL. Its syntax is based on the dynamic modalities of Dynamic Logic [67] and it is similar to LDL (Linear Dynamic Logic) [21]. DMTL provides the dynamic temporal connectives $\langle r|_I$ (past diamond) and $|r\rangle_I$ (future diamond), where r is a regular expression and I is an interval that specifies the domain of temporal quantification.

Main contribution. We propose a novel algorithm for the efficient offline monitoring of DMTL (Theorem 5) with unrestricted past-time and future-time dynamic temporal connectives. Our algorithm goes beyond existing algorithms by considering a more expressive specification language that fuses metric temporal logic and regular expressions. Efficient monitoring algorithms for MTL have been considered before [71,41] (as well as in the setting of quantitative semantics [56]), but the fusion of MTL with regular expressions poses challenges that cannot be addressed by prior approaches. Monitoring for MDL (Metric Dynamic Logic, which is essentially the same formalism as what we call DMTL here) has been considered in [14] and [68]. These works propose algorithms whose time complexity is at least exponential in the size of the temporal specification.

In order to obtain our monitoring algorithm, we devise specialized data structures and algorithms for dealing with the dynamic temporal connectives $\langle r|_I$ and $|r\rangle_I$. These algorithms are expressed using the NFA for an appropriate abstraction of the regular expression r. The main challenge is dealing with the intervals I, which are succinctly represented in binary notation. The key feature of our algorithm is that it needs time-per-item that is polynomial in the size of the specification φ (in fact, $O(|\varphi|^3)$) and constant in the size of the input signal. Each past-time (resp., future-time) dynamic temporal connective is handled with a left-to-right (resp., right-to-left) pass over the trace.

2 Dynamic Metric Temporal Logic

We start this section by presenting the syntax of DMTL (Dynamic Metric Temporal Logic), a logical formalism that fuses metric temporal logic and regular expressions. The syntax that we use is based on LDL (Linear Dynamic Logic) [21] with the time interval annotations of MTL (Metric Temporal Logic) [47]. We interpret DMTL over discrete signals that can be either finite or infinite. In the case of finite signals we use a truncated semantics, in the spirit of [30]. We consider a qualitative (Boolean) semantics (Definition 4), which is given in terms of the satisfaction relation $\models$.

For integers $i, j \in \mathbb{Z}$ we define the intervals $[i,j] = \{n \in \mathbb{Z} \mid i \leq n \leq j\}$ and $[i,\infty) = \{n \in \mathbb{Z} \mid i \leq n\}$. For a set I of integers and $n \in \mathbb{Z}$, define $n + I = \{n + i \mid i \in I\}$ and $n - I = \{n - i \mid i \in I\}$.

For an alphabet D, we write D^* for the set of all finite strings over D. We denote by ε the empty string. For subsets of strings $A, B \subseteq D^*$, we define $A \cdot B = \{uv \mid u \in A \text{ and } v \in B\}$. We also define the *n-fold concatenation* A^n as follows: $A^0 = \{\varepsilon\}$ and $A^{n+1} = A^n \cdot A$.

The two-element set of Boolean values is $\mathbb{B} = \{0, 1\}$. Given a set A, a function $p : A \to \mathbb{B}$ represents a subset of A.

Definition 1 (Regular Expressions). Regular expressions $\mathsf{RExp}(D)$ are given by the following grammar:

$$r, r_1, r_2 ::= \varepsilon \mid p \mid r_1 + r_2 \mid r_1 \cdot r_2 \mid r^*,$$

where $p : D \to \mathbb{B}$ is an atomic predicate. Every regular expression r is interpreted as a subset $\mathcal{L}(r) \subseteq D^*$ as follows:

$$\mathcal{L}(\varepsilon) = \{\varepsilon\} \qquad \mathcal{L}(r_1 + r_2) = \mathcal{L}(r_1) \cup \mathcal{L}(r_2) \quad \mathcal{L}(r^*) = \bigcup_{n \geq 0} \mathcal{L}(r)^n$$
$$\mathcal{L}(p) = \{u \in D \mid p(u) = 1\} \qquad \mathcal{L}(r_1 \cdot r_2) = \mathcal{L}(r_1) \cdot \mathcal{L}(r_2)$$

We say that r *denotes* the language $\mathcal{L}(r)$.

The set $\mathsf{Mat}(m, n, A)$ consists of all matrices with m rows and n columns whose entries are elements of the set A. So, $\mathsf{Mat}(1, n)$ consists of row vectors of size n. Similarly, $\mathsf{Mat}(m, 1)$ consists of column vectors of size m. When no confusion arises, we sometimes identify $\mathsf{Mat}(1, 1, A)$ with A. For a matrix $M : \mathsf{Mat}(m, n, A)$ and integer indexes i and j with $0 \leq i < m$ and $0 \leq j < n$, we write $M(i, j) : A$ for the entry of M at the i-th row and j-th column. We write $\mathsf{Mat}(m, n)$ as abbreviation for $\mathsf{Mat}(m, n, \mathbb{B})$. That is, $\mathsf{Mat}(m, n)$ is the set of Boolean matrices with m rows and n columns.

Definition 2. A *nondeterministic finite automaton* (NFA) over D is a tuple $\mathcal{A} = (Q, \mathsf{init}, \Delta, \mathsf{fin})$, where Q is a finite set of n states, $\mathsf{init} : \mathsf{Mat}(1, n)$ is the initialization (row) vector, $\Delta : \mathsf{Mat}(n, n, D \to \mathbb{B})$ is the *transition matrix*, and $\mathsf{fin} : \mathsf{Mat}(n, 1)$ is the finalization (column) vector. The automaton $\mathcal{A}$ denotes a function $[\![\mathcal{A}]\!] : D^* \to \mathbb{B}$, given as follows:

$$[\![\mathcal{A}]\!](a_1 a_2 \dots a_n) = \mathsf{init} \cdot \Delta[a_1] \cdot \Delta[a_2] \cdots \Delta[a_n] \cdot \mathsf{fin},$$

where $\Delta[a] : \mathsf{Mat}(n, n)$ and $\Delta[a](i, j) = \Delta(i, j)(a)$ for all indexes i, j.

Every regular expression r can be converted into an equivalent nondeterministic automaton $\mathcal{A}_r$ with $|r|$ states, which means that $[\![\mathcal{A}_r]\!]$ is the characteristic function of the language $\mathcal{L}(r) \subseteq D^*$. That is, for every $u \in D^*$, we have that: $[\![\mathcal{A}_r]\!] = 1$ iff $u \in \mathcal{L}(r)$. We will be using this fact freely in Section 3 to describe the monitoring algorithm for DMTL.

We will consider a temporal formalism interpreted over *traces* that are finite or infinite sequences of *data items* from a set D. We write D^* (resp., D^+) for the set of all finite (resp., non-empty finite) sequences over D, and $D^\omega = \omega \to D$ for the set of all infinite sequences over D, where ω is the first infinite ordinal (i.e., the set of natural numbers). We also define $D^\infty = D^* \cup D^\omega$. We write ε for the empty sequence and $|u|$ for the length of a trace, where $|u| = \omega$ if u is infinite. A finite sequence $u \in D^*$ can be viewed as a function from $[0, |u| - 1]$ to D, that is, $u = u(0)u(1) \dots u(|u| - 1)$.

Definition 3 (Syntax). Let D be a set of data items. The set $\mathsf{DMTL}(D)$ of ***temporal formulas*** is built from atomic predicates $p : D \to \mathbb{B}$ using the Boolean connectives $\neg$ and $\vee$, and the unary dynamic temporal connectives $\langle r|_I$ and $|r\rangle_I$, where r is a regular expression and I is an interval of the form $[a, b]$ or $[a, \infty)$ with $0 \leq a \leq b < \omega$. More precisely, the formulas and regular expressions of DMTL are defined by mutual induction according to the following grammar:

$$\varphi, \varphi_1, \varphi_2 ::= p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi\langle r|_I \mid |r\rangle_I \varphi \qquad \text{[formulas]}$$
$$r, r_1, r_2 ::= \varepsilon \mid \varphi? \mid r_1 + r_2 \mid r_1 \cdot r_2 \mid r^* \qquad \text{[regular expressions]}$$

$$
\begin{aligned}
u,[i,j] \models \varepsilon &\iff i = j \\
u,[i,j] \models \varphi? &\iff j = i+1 \text{ and } u,i \models \varphi \\
u,[i,j] \models r_1 + r_2 &\iff u,[i,j] \models r_1 \text{ or } u,[i,j] \models r_2 \\
u,[i,j] \models r_1 \cdot r_2 &\iff u,[i,k] \models r_1 \text{ and } u,[k,j] \models r_2 \text{ for some } k \text{ with } i \le k \le j \\
u,[i,j] \models r^* &\iff i = j \text{ or there is a decomposition } \mathcal{S} \text{ of } [i,j] \text{ such that} \\
&\qquad\quad u,[k,\ell] \models r \text{ for every interval } [k,\ell] \text{ in } \mathcal{S} \\
\\
u,i \models p &\iff p(u(i)) = 1 \\
u,i \models \neg\varphi &\iff u,i \not\models \varphi \\
u,i \models \varphi_1 \vee \varphi_2 &\iff u,i \models \varphi_1 \text{ or } u,i \models \varphi_2 \\
u,i \models |r\rangle_I \varphi &\iff \text{there is } j < |u| \text{ with } j \in i + I \text{ s.t. } u,[i,j] \models r \text{ and } u,j \models \varphi \\
u,i \models \varphi\langle r|_I &\iff \text{there is } j \ge 0 \text{ with } j \in i - I \text{ such that} \\
&\qquad\quad u,j \models \varphi \text{ and } u,[j+1,i+1] \models r
\end{aligned}
$$

Fig. 1. Boolean semantics for DMTL.

We write $\mathsf{DMTL}(D)$ and $\mathsf{DRExp}(D)$ for the set of formulas and regular expressions respectively that the grammar above defines.

For a temporal connective $X \in \{\langle r|, |r\rangle\}$, we write X_a as an abbreviation for $X_{[a,a]}$ and X as an abbreviation for $X_{[0,\infty)}$. The usual temporal connectives P (sometime in the past), S (since), F (sometime in the future) and U (until) can be considered as the following abbreviations:

$$\mathsf{P}_I\varphi = \varphi\langle \top^*|_I \quad \mathsf{F}_I\varphi = |\top^*\rangle_I\varphi \quad \varphi\,\mathsf{S}_I\,\psi = \psi\langle(\varphi?)^*|_I \quad \varphi\,\mathsf{U}_I\,\psi = |(\varphi?)^*\rangle_I\psi$$

where $\top : D \to \mathbb{B}$ is the predicate that always returns 1, i.e., $\top(u) = 1$ for every $u \in D$. Conjunction $\wedge$ can be defined in terms of $\neg$ and $\vee$. The duals $|r]_I$ and $[r|_I$ of $|r\rangle_I$ and $\langle r|_I$ respectively can also be seen as abbreviations: $|r]_I\varphi = \neg|r\rangle_I\neg\varphi$ and $\varphi[r|_I = \neg((\neg\varphi)\langle r|_I)$.

A *decomposition* of an interval $[i,j]$ (where $i \le j$) is a nonempty finite sequence of intervals $[i_1,j_1],[i_2,j_2],\ldots,[i_n,j_n]$ with $i_1 = i$, $j_n = j$, and $j_k = i_{k+1}$ for every $k = 1,2,\ldots,n-1$.

For a trace u and an interval $[i,j]$ with $0 \le i \le j \le |u|$, we write $u[i..j] = u(i)u(i+1)\ldots u(j-1)$ for the substring of u at location $[i,j]$. In particular, we have that $u[i..i] = \varepsilon$ and $u[i..i+1] = u(i)$.

Definition 4 (Boolean Semantics). We interpret the formulas in $\mathsf{DMTL}(D)$ over traces from D^∞ and at specific time points. The regular expressions of DMTL are interpreted over time intervals. The Boolean semantics involves two *satisfaction relations*, which are defined in Fig. 1.

- For a regular expression r, a trace $u \in D^\infty$ and two positions $0 \le i \le j \le |u|$, we write $u,[i,j] \models r$ when r is true in u at the interval $[i,j]$.
- For a formula $\varphi \in \mathsf{DMTL}(D)$, a trace $u \in D^\infty$ and a position $0 \le i < |u|$, we write $u,i \models \varphi$ when φ is true in u at position i.

$$\rho(\varepsilon, u, [i,j]) = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}$$

$$\rho(\varphi?, u, [i,j]) = \begin{cases} \rho(\varphi, u, i), & \text{if } j = i+1 \\ 0, & \text{otherwise} \end{cases}$$

$$\rho(r_1 + r_2, u, [i,j]) = \rho(r_1, u, [i,j]) \sqcup \rho(r_2, u, [i,j])$$

$$\rho(r_1 \cdot r_2, u, [i,j]) = \bigsqcup_{i \le k \le j} \Big(\rho(r_1, u, [i,k]) \sqcap \rho(r_2, u, [k,j])\Big)$$

$$\rho(r^*, u, [i,j]) = \bigsqcup_{i \le k_1 \le \cdots \le k_n \le j} \Big(\rho(r, u, [i,k_1]) \sqcap \rho(r, u, [k_1,k_2]) \sqcap \cdots \sqcap \rho(r, u, [k_n,j])\Big)$$

$$\rho(p, u, i) = p(u(i))$$

$$\rho(\neg\varphi, u, i) = \neg\rho(\varphi, u, i)$$

$$\rho(\varphi \vee \psi, u, i) = \rho(\varphi, u, i) \sqcup \rho(\psi, u, i)$$

$$\rho(\langle r \rangle_I \varphi, u, i) = \bigsqcup_{j \in i+I,\, j < |u|} \Big(\rho(r, u, [i,j]) \sqcap \rho(\varphi, u, j)\Big)$$

$$\rho(\varphi \langle r|_I, u, i) = \bigsqcup_{j \in i-I,\, j \ge 0} \Big(\rho(\varphi, u, j) \sqcap \rho(r, u, [j+1, i+1])\Big)$$

Fig. 2. Some properties of the interpretation function for DMTL.

We define the *(formula) interpretation function* $\rho : \mathsf{DMTL}(D) \times D^\infty \times \omega \to \mathbb{B}$, where $\rho(\varphi, u, i)$ is defined when $0 \le i < |u|$, as follows:

$$\rho(\varphi, u, i) = 1, \text{ if } u, i \models \varphi \qquad \rho(\varphi, u, i) = 0, \text{ if } u, i \not\models \varphi$$

For a formula φ and a trace $u \in D^\infty$, then we write $\rho(\varphi, u)$ to denote the trace v with $|v| = |u|$ given by $v(i) = \rho(\varphi, u, i)$ for every $0 \le i < |u|$. Similarly, we define the *(regular expression) interpretation function* $\rho : \mathsf{DRExp}(D) \times D^\infty \times (\mathbb{N} \times \mathbb{N}) \to \mathbb{B}$, where $\rho(r, u, [i,j])$ is defined when $0 \le i \le j \le |u|$, as follows:

$$\rho(\varphi, u, [i,j]) = 1, \text{ if } u, [i,j] \models r \qquad \rho(\varphi, u, [i,j]) = 0, \text{ if } u, [i,j] \not\models r$$

Notice that $u, i \models \varphi$ iff $u, [i, i+1] \models \varphi?$. Another way to describe this property is as follows: $\rho(\varphi, u, i) = \rho(\varphi?, u, [i, i+1])$.

The set $\mathbb{B} = \{0, 1\}$ of Boolean values is a lattice. We write $\sqcap$ for the meet operation (i.e., conjunction) and $\sqcup$ for the join operation (i.e., disjunction).

Fig. 2 shows some properties that the interpretation function of Definition 4 satisfies. These properties are an immediate consequence of the definition of the satisfaction relation $\models$ from Fig. 1.

From DMTL to Regular Expression Matching. Let $r \in \mathsf{DRExp}(D)$ be a regular expression of DMTL. The expression r may contain subexpressions of the form $\varphi?$, where φ is a temporal formula. Let $\varphi_0, \varphi_1, \ldots, \varphi_{K-1}$ be an enumeration of the maximal subexpressions of the form $\varphi?$ that r contains. We will see now

how to reduce the interpretation of $\varphi_K\langle r|$ to the interpretation of a pure regular expression.

Define $\varphi_K\langle r|$ and $r' = \pi_K \cdot r[\pi_i/\varphi_i?]$, where $r[\pi_i/\varphi_i?]$ results from r by replacing each $\varphi_i?$ by π_i. Each $\pi_i : \mathbb{B}^{K+1} \to \mathbb{B}$ is the i-th projection function. We call r' the *abstraction* of the formula $\varphi_K\langle r|$.

This abstraction operation is similar to the "oracle-projection" operation used in [55] to deal with regular expressions that contain lookaround assertions.

Let $u \in D^*$ be a finite trace and i be a time instance with $0 \leq i < |u|$. The main observation is that $u, i \models \varphi_K\langle r|$ iff $\tau[0..i+1] \in \mathcal{L}(r')$, where $\tau = \rho(\varphi_0, u) \times \cdots \times \rho(\varphi_{K-1}, u) \times \rho(\varphi_K, u)$.

3 Efficient Monitoring

Given a temporal property φ and a trace u, the *monitoring problem* asks to compute $\rho(\varphi, u)$ (see Definition 4). In other words, the output of monitoring is a tape $v : \mathbb{B}^*$ with length $|v| = |u|$ so that $v(i) = \rho(\varphi, u, i)$ for every $0 \leq i < |u|$.

DMTL monitoring can be reduced to a small set of computational primitives. In order to efficiently monitor formulas that involve the temporal connectives $\langle r|_I$ and $|r\rangle_I$, we provide specialized algorithms that are presented later in Fig. 3, Fig. 4, Fig. 5, and Fig. 6. The base expressions in each r are of the form $\varphi?$, where φ is a temporal formula.

The temporal connective $\langle r|_I$ is handled with a left-to-right pass over the input trace. For the formula $\varphi_K\langle r|_I$, let $\varphi_0?, \varphi_1? \ldots, \varphi_{K-1}?$ be an enumeration of the maximal temporal subexpressions $\varphi_i?$ that appear in r. Evaluating the formula $\varphi_K\langle r|_I$ over a trace $u \in D^\infty$ is the same as evaluating the formula $(\varphi_K\langle r|_I)[\pi_i/\varphi_i]$ that results from $\varphi_K\langle r|_I$ by replacing each φ_i by π_i, over

$$\tau = \rho(\varphi_0, u) \times \rho(\varphi_1, u) \times \cdots \times \rho(\varphi_{K-1}, u) \times \rho(\varphi_K, u) : (\mathbb{B}^{K+1})^*.$$

We write $\pi_i : A_0 \times A_1 \times \cdots \times A_K \to A_i$ for the i-th projection. So, the monitoring of $\varphi_K\langle r|_I$ can be performed in two stages: (1) monitoring each of the $\varphi_0, \varphi_1, \ldots, \varphi_{K-1}, \varphi_K$ formulas, and (2) propagating the output signals from the first stage into a monitor for the formula $\pi_K\langle r[\pi_i/\varphi_i]|_I$. Notice that $r[\pi_i/\varphi_i]$ is a pure regular expression, that is, there is no nesting of regular and temporal operators.

The key observation from the previous paragraph is that the original formula $\varphi_K\langle r|_I$ is true in u (original trace) at time instant i iff the rewritten formula $\pi_K\langle r[\pi_i/\varphi_i]|_I$ is true in the trace τ (output tapes from maximal temporal subformulas) at time instant i. In turn, this is equivalent to the pure regular expression $r' = \pi_K \cdot r[\pi_i/\varphi_i?]$ matching over the appropriate interval (determined by I)

The temporal connective $|r\rangle_I$ is handled with a right-to-left pass over the input trace. The computation is completely symmetric to the past-time case, so we omit the discussion.

```
// D = 𝔹^{K+1} is the type of input data items
// n = # states of the automaton 𝒜 = (Q, init, Δ, fin) : NFA(D) for p · r
Mat(1, n) vec ← 𝟘_{Mat(1,n)}  // row vector for automaton configuration
// Invariant for the configuration row vector vec:
// Suppose that the input history is [x_0, x_1, ..., x_{N-1}] ∈ D*.
// Then, vec = ⨆_{i=0}^{N-1} (init · Δ[x_i] · Δ[x_{i+1}] ··· Δ[x_{N-1}]).
Function Next(D x):
    vec ← vec ⊔ init // re-initialize:  a new automaton execution is spawned
    // the automaton takes a transition:
    vec ← vec · Δ[x]  // vector-matrix multiplication
    return vec · fin // emit output value
```

Fig. 3. Monitor for the formula $\varphi\langle r|_{[0,\infty)}$.

3.1 Monitor for the formula $p\langle r|_{[0,\infty)}$

Based on the previous discussion, we will assume from now on that a formula of the form $\varphi\langle r|_I$ contains only atomic predicates in r that are projections (i.e., π_i for some index i). We also assume that φ is a projection atomic predicate.

Since r is a pure regular expression (that is, every occurrence of the ? operator is applied to atomic propositions), the monitoring of the formula $p\langle r|_{[0,\infty)}$ amounts to matching the regular expression $p? \cdot r$. As shown in Fig. 3, the monitoring algorithm constructs the automaton $\mathcal{A} = (Q, \mathsf{init}, \Delta, \mathsf{fin}) : \mathsf{NFA}(D)$ for $p? \cdot r$ and simulates its execution. Let u be a finite trace and $\ell = |u| \geq 1$ be its length. We have that:

$$\begin{aligned}
\rho(p\langle r|_{[0,\infty)}, u, \ell - 1) &= \bigsqcup_{i=0}^{\ell-1} \Big(\rho(p, u, i) \sqcap \rho(r, u, [i+1, \ell]) \Big) \\
&= \bigsqcup_{i=0}^{\ell-1} \Big(\rho(p? \cdot r, u, [i, \ell]) \Big) \\
&= \bigsqcup_{i=0}^{\ell-1} \Big(\mathsf{init} \cdot \Delta[u(i)] \cdots \Delta[u(i_1)] \cdots \Delta[u(\ell-1)] \cdot \mathsf{fin} \Big) \\
&= X_\ell \cdot \mathsf{fin}, \text{ where} \\
X_0 = \mathbb{0}_{\mathsf{Mat}(1,n)} \quad &\text{and} \quad X_{i+1} = (X_i + \mathsf{init}) \cdot \Delta[u(i)]
\end{aligned}$$

For example, for the trace $u = u_0 u_1 u_2$, we would have

$$\begin{aligned}
X_3 &= (((\mathbb{0} + \mathsf{init}) \cdot \Delta[u_0] + \mathsf{init}) \cdot \Delta[u_1] + \mathsf{init}) \cdot \Delta[u_2] \\
&= \mathsf{init} \cdot \Delta[u_0] \cdot \Delta[u_1] \cdot \Delta[u_2] + \mathsf{init} \cdot \Delta[u_1] \cdot \Delta[u_2] + \mathsf{init} \cdot \Delta[u_2].
\end{aligned}$$

The algorithm maintains a row vector *vec* that represents the configuration of the automaton. At every step, the initialization row vector is added to *vec* (which essentially means that a new thread of execution is spawned at that moment) and then *vec* is multiplied with the transition matrix $\Delta[x]$, where x is the current data item. The output is $vec \cdot \mathsf{fin} \in \mathbb{B}$, where fin is the finalization column vector of the automaton. This algorithm uses space $O(n)$ to store *vec* and time-per-item $O(n^2)$ to perform a vector-matrix multiplication, where n is the number of states of $\mathcal{A}$.

```
// D = 𝔹^{K+1} is the type of input data items
// n = # states of the automaton 𝒜 = (Q, init, Δ, fin) : NFA(D) for p · r
// State for UpdateWnd_a:  buf, wnd, m, z, agg.
[D; a] buf ← [nil; a]  // input buffer:  fill array of size a with nil items
// window -- fill array of size a with identity matrices:
[Mat(n, n); a] wnd ← [𝟙_{Mat(n,n)}; a]
Nat m ← 0  // size of new block
Mat(n, n) z ← 𝟙_{Mat(n,n)}  // aggregate of new block:  identity matrix
Mat(n, n) agg ← 𝟙_{Mat(n,n)}  // initial overall aggregate:  identity matrix
Function UpdateWnd_a(D x):
    buf[m] ← x  // new item is placed on the buffer
    wnd[m] ← Δ[x]  // evict oldest matrix, replace with Δ[x]
    m ← m + 1  // new block enlarged
    z ← z · Δ[x]  // update aggregate for new block:  matrix multiplication
    if m = a then  // the new block is full
        for i ← a − 2 to 0 do  // convert new block to old block
            wnd[i] ← wnd[i] · wnd[i + 1]  // matrix multiplication
        m ← 0  // empty new block
        z ← 𝟙_{Mat(n,n)}  // identity matrix
    agg ← wnd[m] · z  // update overall aggregate:  matrix multiplication

// State for Next:
Nat ℓ ← 0  // counter for number of items consumed so far (up to a items)
Mat(1, n) vec ← 𝟘_{Mat(1,n)}  // row vector for automaton configuration
Function Next(D x):
    D old ← buf[m]  // oldest item gets evicted from the input buffer
    UpdateWnd_a(x)  // update the window data structure with current item
    if ℓ < a then  // no item evicted from the input buffer
        ℓ ← ℓ + 1  // increment ℓ
        return 0  // the formula is false
    else  // ℓ = a:  item evicted from the input buffer
        vec ← vec ⊔ init  // re-initialize:  a new automaton execution is spawned
        vec ← vec · Δ[old]  // automaton transition:  vector-matrix multiplication
        return vec · agg · fin  // emit output value
```

Fig. 4. Monitor for the formula $p\langle r|_{[a,\infty)}$, where $a \geq 1$.

3.2 Monitor for the formula $p\langle r|_{[a,\infty)}$ where $a \geq 1$

As shown in Fig. 4, the monitoring algorithm constructs the automaton $\mathcal{A} = (Q, \mathsf{init}, \Delta, \mathsf{fin}) : \mathsf{NFA}(D)$ for $p? \cdot r$ and simulates its execution. Let u be a finite trace and $\ell = |u|$ be its length. If $\ell \leq a$, then $\rho(p\langle r|_{[a,\infty)}, u, \ell-1) = 0$. If $\ell \geq a+1$, then we have:

$$\rho(p\langle r|_{[a,\infty)}, u, \ell - 1) =$$
$$\bigsqcup_{i=0}^{\ell-1-a} \Big(\rho(p, u, i) \sqcap \rho(r, u, [i+1, \ell])\Big) =$$
$$\bigsqcup_{i=0}^{\ell-1-a} \Big(\rho(p? \cdot r, u, [i, \ell])\Big) =$$
$$\bigsqcup_{i=0}^{\ell-1-a} \Big(\mathsf{init} \cdot \Delta[u(i)] \cdots \Delta[u(\ell-1-a)] \cdot \Delta[u(\ell-a)] \cdots \Delta[u(\ell-1)] \cdot \mathsf{fin}\Big) =$$

$$\left(\sum_{i=0}^{\ell-1-a} \mathsf{init} \cdot \Delta[u(i)] \cdots \Delta[u(\ell-1-a)]\right) \cdot \left(\Delta[u(\ell-a)] \cdots \Delta[u(\ell-1)]\right) \cdot \mathsf{fin}.$$

We see above that the value can be expressed as the product of a row vector $vec : \mathsf{Mat}(1, n)$, a matrix $agg : \mathsf{Mat}(n, n)$, and the finalization column vector $\mathsf{fin} : \mathsf{Mat}(n, 1)$. Intuitively, the row vector vec is the configuration of the automaton $\mathcal{A}$ after consuming all the input except for the last a data items. So, vec can be computed by executing the automaton $\mathcal{A}$ using input that is delayed by a steps. This delay of the input can be realized efficiently using a ring buffer. The matrix agg is the product of the transition matrices for the last a data items. This corresponds to a sliding-window computation, which means that at every step the window over which the product is computed shifts one step to the right.

The algorithm of Fig. 4 gives an efficient implementation of the ring buffer for computing vec and the algorithm for computing agg. The key data structures of the algorithm are buf, wnd, agg, m, and z. The arrays buf and wnd are split into a "new block" containing entries that correspond to the last m items and an "old block". Suppose that the last a input items are the following:

$$[x_0, x_1, \ldots, x_{a-m-1}, \underbrace{x_{a-m}, \ldots, x_{a-2}, x_{a-1}}_{\text{last } m \text{ input items}}], \text{ where } x_{a-1} \text{ is the last item.}$$

Then, the ring buffer buf (array of size a with elements of type D) has the following contents:

$$buf = [\ \underbrace{x_{a-m}, \ldots, x_{a-2}, x_{a-1}}_{\text{new block: last } m \text{ input items}}\ , \underbrace{x_0, x_1, \ldots, x_{a-m-1}}_{\text{old block}}].$$

The window wnd (array of size a with elements of type $\mathsf{Mat}(n, n)$) has the following contents:

$$wnd = [\underbrace{\Delta[x_{a-m}], \ldots, \Delta[x_{a-2}], \Delta[x_{a-1}]}_{\text{new block: } m \text{ matrices}},\ \underbrace{y_0, y_1, \ldots, y_{a-m-1}}_{\text{old block: } a-m \text{ matrices}}\ ], \text{ where}$$
$$y_i = \Delta[x_i] \cdot \Delta[x_{i+1}] \cdots \Delta[x_{a-m-1}] \text{ for every } i = 0, \ldots, a-m-1$$

The key invariant for the matrix $z : \mathsf{Mat}(n, n)$ is that it is always the product

$$z = \Delta[x_{a-m}] \cdots \Delta[x_{a-2}] \cdot \Delta[x_{a-1}]$$

of the transition matrices for the last m items. Finally, the matrix $agg : \mathsf{Mat}(n, n)$ is the product (aggregate) of the transition matrices for the entire window:

$$agg = y_0 \cdot z = \Delta[x_0] \cdots \Delta[x_{a-m-1}] \cdot \Delta[x_{a-m}] \cdots \Delta[x_{a-1}].$$

When a new item x arrives, we evict the aggregate y_0 corresponding to the oldest item x_0 and replace it by $\Delta[x]$. Thus, the new block is expanded with the additional matrix $\Delta[x]$ and therefore we also update the aggregates z and agg. When the new block becomes full (i.e., $m = a$) then we convert it to an old block by performing all partial products from right to left. This conversion requires

```
// D = 𝔹^{K+1} is the type of input data items
// n = # states of the automaton 𝒜 = (Q, init, Δ, fin) : NFA(D) for p · r
// s = b + 1 ≥ 1 is the size of the window
// State for PUpdateWndVM_s:  wndm, wndv, k, zm, zv, aggm, aggv.
[Mat(n, n); s] wndm ← [𝟙_{Mat(n,n)}; s]  // fill array of size s with identity matrices
[Mat(1, n); s] wndv ← [𝟘_{Mat(1,n)}; s]  // fill array of size s with zero row vectors
Nat k ← 0  // size of new block
Mat(n, n) zm ← 𝟙_{Mat(n,n)}  // matrix aggregate of new block:  identity matrix
Mat(1, n) zv ← 𝟘_{Mat(1,n)}  // vector aggregate of new block:  zero row vector
Mat(n, n) aggm ← 𝟙_{Mat(n,n)}  // initial matrix aggregate:  identity matrix
Mat(1, n) aggv ← 𝟘_{Mat(1,n)}  // initial vector aggregate:  zero row vector
Function PUpdateWndVM_s(D x):
    wndm[k] ← Δ[x]  // evict oldest matrix, replace with Δ[x]
    wndv[k] ← init · Δ[x]  // evict oldest vector, replace with init · Δ[x]
    k ← k + 1  // new block enlarged
    // update matrix aggregate for new block (matrix multiplication):
    zm ← zm · Δ[x]
    // update vector aggregate for new block (vector-matrix multiplication):
    zv ← zv · Δ[x] + init · Δ[x]
    if k = s then // the new block is full
        for i ← s − 2 to 0 do  // convert new block to old block
            wndm[i] ← wndm[i] · wndm[i + 1]  // matrix multiplication
            // vector-matrix multiplication:
            wndv[i] ← wndv[i] · wndm[i + 1] + wndv[i + 1]
        k ← 0  // empty new block
        zm ← 𝟙_{Mat(n,n)}  // identity matrix
        zv ← 𝟘_{Mat(1,n)}  // zero row vector
    // update overall matrix aggregate (matrix multiplication):
    aggm ← wndm[k] · zm
    // update overall vector aggregate (vector-matrix multiplication):
    aggv ← wndv[k] · zm + zv

Function Next(T x):
    PUpdateWndVM_{b+1}(x)  // update the data structure
    return aggv · fin  // emit output value
```

Fig. 5. Monitor for the formula $p\langle r|_{[0,b]}$.

$a - 1$ applications of matrix multiplication, but it is performed once every a items. So, the algorithm needs $O(n^3)$ amortized time-per-item and $O(a \cdot n^2)$ space to store the window of matrices.

Finally, notice that the updating of *vec* is very similar to the monitor of Fig. 3. The main difference is that the input items are delayed by a steps using the ring buffer *buf*.

3.3 Monitor for the formula $p\langle r|_{[0,b]}$

As shown in Fig. 5, the monitoring algorithm constructs the automaton $\mathcal{A} = (Q, \mathsf{init}, \Delta, \mathsf{fin}) : \mathsf{NFA}(D)$ for the regular expression $p? \cdot r$. Let u be a finite trace and $\ell = |u|$ be its length. We have:

$$\begin{aligned}\rho(p\langle r|_{[0,b]}, u, \ell-1) &= \bigsqcup_{i=\max(0,\ell-1-b)}^{\ell-1} \big(\rho(p,u,i) \sqcap \rho(r,u,[i+1,\ell-1])\big) \\ &= \Big(\sum_{i=\max(0,\ell-1-b)}^{\ell-1} \mathsf{init} \cdot \Delta[u(i)] \cdots \Delta[u(\ell-1)]\Big) \cdot \mathsf{fin}.\end{aligned}$$

We see above that the value can be expressed as the product of a row vector $aggv : \mathsf{Mat}(1,n)$ and the finalization column vector $\mathsf{fin} : \mathsf{Mat}(n,1)$. Intuitively, the row vector $aggv$ is the configuration of the automaton $\mathcal{A}$ (with re-initialization at every step) after consuming the last $s = b+1$ data items. It corresponds to a sliding-window aggregation, which means that at every step the window over which the aggregate is computed shifts one step to the right[1].

The algorithm of Fig. 5 gives an efficient implementation of the algorithm for computing $aggv$. The key data structures of the algorithm are $wndm$ (window of matrices), $wndv$ (window of row vectors), k, zm, zv, $aggm$ (overall matrix aggregate), and $aggv$ (overall vector aggregate). The arrays $wndm$ and $wndv$ are split between a "new block" containing entries that correspond to the last k items and an "old block". Suppose that the last s input items are the following:

$$[x_0, x_1, \ldots, x_{s-k-1}, \underbrace{x_{s-k}, \ldots, x_{s-2}, x_{s-1}}_{\text{last } k \text{ input items}}], \text{ where } x_{s-1} \text{ is the last item.}$$

The window $wndm$ (array of size s with elements of type $\mathsf{Mat}(n,n)$) has the following contents:

$$wndm = [\underbrace{\Delta[x_{s-k}], \ldots, \Delta[x_{s-2}], \Delta[x_{s-1}]}_{\text{new block: } k \text{ matrices}}, \underbrace{ym_0, ym_1, \ldots, ym_{s-k-1}}_{\text{old block: } s-k \text{ matrices}}], \text{ where}$$
$$ym_i = \Delta[x_i] \cdot \Delta[x_{i+1}] \cdots \Delta[x_{s-k-1}] \text{ for every } i = 0, \ldots, s-k-1$$

The window $wndv$ (array of size s with elements of type $\mathsf{Mat}(1,n)$) has the following contents:

$$wndv = [\underbrace{\mathsf{init} \cdot \Delta[x_{s-k}], \ldots, \mathsf{init} \cdot \Delta[x_{s-1}]}_{\text{new block: } k \text{ row vectors}}, \underbrace{yv_0, yv_1, \ldots, yv_{s-k-1}}_{\text{old block: } s-k \text{ row vectors}}], \text{ where}$$
$$yv_t = \textstyle\sum_{i=t}^{s-k-1} \big(\mathsf{init} \cdot \prod_{j=i}^{s-k-1} \Delta[x_j]\big) \text{ for every } t = 0, \ldots, s-k-1$$

The key invariants for the matrix $zm : \mathsf{Mat}(n,n)$ (new block matrix aggregate) and the row vector $zv : \mathsf{Mat}(1,n)$ (new block vector aggregate) are the following:

$$zm = \Delta[x_{s-k}] \cdots \Delta[x_{s-2}] \cdot \Delta[x_{s-1}] \qquad zv = \textstyle\sum_{i=s-k}^{s-1} \big(\mathsf{init} \cdot \prod_{j=i}^{s-1} \Delta[x_j]\big)$$

[1] Notice that this aggregation is not the same as the aggregation of Fig. 4, which is simply a product of transition matrices.

// $D = \mathbb{B}^{K+1}$ is the type of input data items
// $n =$ # states of the automaton $\mathcal{A} = (Q, \mathsf{init}, \Delta, \mathsf{fin}) : \mathsf{NFA}(D)$ for $p \cdot r$
// State for $\mathrm{UpdateWnd}_a$: buf, wnd, m, z, agg.
// State for $\mathrm{PUpdateWndVM}_s$ with $s = b - a + 1 \geq 1$:
// $wndm$, $wndv$, k, zm, zv, $aggm$, $aggv$.
Nat $\ell \leftarrow 0$ // counter for number of items consumed so far (up to a items)
Function Next(T x):
  D $old \leftarrow buf[m]$ // oldest item gets evicted from the input buffer
  // update the window data structure for the interval $[0, a-1]$:
  $\mathrm{UpdateWnd}_a(x)$
  if $\ell < a$ **then** // no item evicted from input buffer
    $\ell \leftarrow \ell + 1$ // increment ℓ
    return 0 // the formula is false
  else // $\ell = a$: item evicted from the input buffer
    // update the data structure for the interval $[a, b]$:
    $\mathrm{PUpdateWndVM}_{b-a+1}(old)$
    return $aggv \cdot agg \cdot \mathsf{fin}$ // emit output value

Fig. 6. Monitor for the formula $p\langle r|_{[a,b]}$, where $a \geq 1$.

Notice that zm is the product of transition matrices for last k items. Finally, the overall matrix aggregate $aggm : \mathsf{Mat}(n, n)$ and the overall vector aggregate $aggv : \mathsf{Mat}(1, n)$ satisfy the following invariants:

$$
\begin{aligned}
aggm &= ym_0 \cdot zm \\
&= \Delta[x_0] \cdots \Delta[x_{s-k-1}] \cdot \Delta[x_{s-k}] \cdots \Delta[x_{s-1}] \\
aggv &= yv_0 \cdot zm + zv \\
&= \left(\textstyle\sum_{i=0}^{s-k-1} \mathsf{init} \cdot \prod_{j=i}^{s-k-1} \Delta[x_j]\right) \cdot \prod_{j=s-k}^{s-1} \Delta[x_j] + \left(\textstyle\sum_{i=s-k}^{s-1} \mathsf{init} \cdot \prod_{j=i}^{s-1} \Delta[x_j]\right) \\
&= \left(\textstyle\sum_{i=0}^{s-k-1} \mathsf{init} \cdot \prod_{j=i}^{s-1} \Delta[x_j]\right) + \left(\textstyle\sum_{i=s-k}^{s-1} \mathsf{init} \cdot \prod_{j=i}^{s-1} \Delta[x_j]\right) \\
&= \textstyle\sum_{i=0}^{s-1} \left(\mathsf{init} \cdot \prod_{j=i}^{s-1} \Delta[x_j]\right)
\end{aligned}
$$

When a new item x arrives, we evict the aggregates ym_0, yv_0 corresponding to the oldest item x_0 and replace them by $\Delta[x]$ and $\mathsf{init} \cdot \Delta[x]$ respectively. Thus, the new block is expanded and therefore we also update the aggregates zm, zv, $aggm$, and $aggv$. When the new block becomes full (i.e., $m = s$) then we convert it to an old block. This conversion requires $s - 1$ applications of matrix multiplication and vector-matrix multiplication, but it is performed once every s items. So, the algorithm needs $O(n^3)$ amortized time-per-item and $O(s \cdot n^2)$ space to store the windows of matrices and vectors.

3.4 Monitor for the formula $p\langle r|_{[a,b]}$

As shown in Fig. 6, the monitoring algorithm constructs the automaton $\mathcal{A} = (Q, \mathsf{init}, \Delta, \mathsf{fin}) : \mathsf{NFA}(D)$ for the regular expression $p? \cdot r$. Let u be a finite trace and $\ell = |u|$ be its length. We have:

$$\begin{array}{l}
\rho(p\langle r|_{[a,b]}, u, \ell-1) = \\
\bigsqcup_{i=\max(0,\ell-1-b)}^{\ell-1-a} \big(\rho(p,u,i) \sqcap \rho(r,u,[i,\ell-1])\big) = \\
\bigsqcup_{i=\max(0,\ell-1-b)}^{\ell-1-a} \big(\mathsf{init} \cdot \Delta[u(i)] \cdots \Delta[u(\ell-1-a)] \cdot \Delta[u(\ell-a)] \cdots \Delta[u(\ell-1)] \cdot \mathsf{fin}\big) = \\
\Big(\sum_{i=\max(0,\ell-1-b)}^{\ell-1-a} \mathsf{init} \cdot \Delta[u(i)] \cdots \Delta[u(\ell-1-a)]\Big) \cdot \Big(\Delta[u(\ell-a)] \cdots \Delta[u(\ell-1)]\Big) \cdot \mathsf{fin}.
\end{array}$$

We see above that the value can be expressed as the product of a row vector $aggv : \mathsf{Mat}(1,n)$, a matrix $agg : \mathsf{Mat}(n,n)$, and the finalization column vector $\mathsf{fin} : \mathsf{Mat}(n,1)$. Intuitively, the row vector $aggv$ is the vector aggregate computed with the algorithm of Fig. 5 (procedure PUpdateWndVM$_s$ with size $s = b{-}a{+}1$). The difference here is that $aggv$ is delayed by a steps, which can be accomplished using a buffer of size a. The matrix agg is the product of the transition matrices for the last a data items. This corresponds to the sliding-window computation described in Fig. 4 (procedure UpdateWnd$_a$).

The online monitor of Fig. 6 includes the state needed for UpdateWnd$_a$(buf, wnd, m, z, agg) and the state needed for PUpdateWndVM$_s$ with size $s = b{-}a{+}1$ ($wndm$, $wndv$, k, zm, zv, $aggm$, $aggv$). Suppose that the last $b{+}1$ input items are the following:

$$[\underbrace{x_0, x_1, \ldots, x_{b-a}}_{b-a+1 \text{ input items}}, \underbrace{x_{b-a+1}, \ldots, x_{b-1}, x_b}_{\text{last } a \text{ input items}}], \text{ where } x_b \text{ is the last item.}$$

From the invariants for the procedures UpdateWnd$_a$ and PUpdateWndVM$_{b-a+1}$ we already know that the following hold:

$$aggv = \textstyle\sum_{i=0}^{b-a} \big(\mathsf{init} \cdot \prod_{j=i}^{b-a} \Delta[x_j]\big) \qquad agg = \Delta[x_{b-a+1}] \cdots \Delta[x_{b-1}] \cdot \Delta[x_b]$$

We conclude that $aggv \cdot agg \cdot \mathsf{fin} = \sum_{i=0}^{b-a} \big(\mathsf{init} \cdot \prod_{j=i}^{b} \Delta[x_j] \cdot \mathsf{fin}\big)$, which is the desired output value.

The algorithm needs $O(n^3)$ amortized time-per-item and $O(b \cdot n^2)$ space to store the windows of matrices and vectors.

3.5 Overall Monitoring Algorithm

The algorithm proceeds in a bottom-up manner, computing the output $\rho(\psi, u)$ for every subformula ψ of φ. The base case of an atomic formula $p : D \to \mathbb{B}$ is trivial. The Boolean connectives negation $\neg$ and disjunction $\vee$ are easy to handle. The "box" connectives $|r]_I$ and $[r|_I$ are encoded using negation the "diamond" connectives. We discussed earlier in Sections 3.1, 3.2, 3.3 and 3.4 how to handle the past diamond connective $\langle r|_I$ with 4 different algorithms for each form of the time interval I: $[0,\infty)$, $[a,\infty)$ with $a \geq 1$, $[0,b]$, and $[a,b]$ with $a \geq 1$. All these

algorithms proceed with a single left-to-right pass over the input trace (and the output tapes of the maximal strict subformulas).

The case of the *future* diamond connective $|r\rangle_I$ is completely symmetric to the case $\langle r|_I$. The difference is that the future connective requires a *right-to-left* pass over the input.

Theorem 5. For a DMTL formula φ and a finite input trace u, the monitoring problem can be solved in time $O(|\varphi|^3 \cdot \ell)$, where ℓ is the length of the trace u.

Proof. The main arguments for the correctness of the overall monitoring algorithm (i.e., the invariants for the data structures) have already been provided in Sections 3.1, 3.2, 3.3 and 3.4. We have also seen for every case of the formulas of the form $\varphi = \psi\langle r|_I$ that the amortized complexity is $O(|\varphi|^3)$ per data item. Since the amortized per-item running time is an average over the entire input stream, the overall running time is $O(|\varphi|^3 \cdot \ell)$ *in the worst case*. The complexity analysis for formulas of the form $\varphi = |r\rangle_I\psi$ is analogous. The remaining cases of Boolean connectives are straightforward. □

The space requirements of the algorithm are exponential in the size of the specification φ because of the succinct representation of the time intervals I.

The polynomial complexity bound of Theorem 5 is interesting because the DMTL formalism combines regular expressions with metric temporal connectives that express a kind of "counting". The operator $\{m, n\}$ of *counting* (or *bounded repetition*) in regular expressions is similar. The pattern $r\{m, n\}$ describes the repetition of r from m to n times. Bounded repetition makes regular expressions exponentially more succinct and thus raises algorithmic challenges. See, for example, [36] for relevant results and pointers to relevant literature. Recent works [72,46,48,75] use automata with counters or bit vectors for efficient matching. For some special cases of intervals I, more efficient algorithms may apply [35].

The bottom-up monitoring algorithm described in this section performs left-to-right (resp., right-to-left) passes over the input trace for past-time (resp., future-time) temporal connectives. For the special case where there are only past-time or only future-time connectives, it would be possible to perform a single pass (similarly to [55] for lookaround assertions). In this case, it seems that the algorithm could be conveniently described using streaming dataflow constructs (see, e.g., [44,49,70,15,54]), as is done in [60] for MTL.

As a final remark, it may be possible to use the algorithms of this section in the context of quantitative formalisms and query languages for streaming data [19,32]. For example, Quantitative Regular Expressions (QREs) [59,6,9] and associated automata-theoretic models with registers [7,8,5] could be extended with "counting" features similar to DMTL's connectives. QREs have been used to express complex online detection algorithms for medical monitoring [3,4]. Extending their syntax with "counting" features may provide convenience of expression, thus making them suitable for more applications.

4 Related Work

The syntax of our temporal formalism DMTL is based on Dynamic Logic [67,37], and also on the more recent work on ForSpec [10], PSL [28,1], SystemVerilog [74,17,2], and LDL [21]. Pnueli and Zaks [64,66] consider runtime verification for PSL using a class of transducers that they call temporal testers. Morin-Allory and Borrione [62] use PVS to prove the correctness of monitors that are synthesized from PSL specifications. Das et al. [20] present an approach for synthesizing SystemVerilog assertions in hardware. Armoni et al. [11] use deterministic automata to monitor temporal specifications. Eisner [27] discusses the use of PSL in dynamic and runtime verification. Boulé and Zilic [16] use an automata-based approach for the synthesis of assertion checkers from PSL properties. Morin-Allory et al. [63] validate several rewrite rules for PSL properties using PVS. Javaheri et al. [43] studies the synthesis in hardware of PSL regular expressions. Eisner and Fisman [29] discuss extensions to temporal logic that are included in SVA and/or PSL and the semantic issues that they raise. Witharana et al. [76] present both pre-silicon and post-silicon assertion-based approaches for hardware validation. MDL (Metric Dynamic Logic) is considered in [14] and [68].

The monitoring of LTL properties with only future-time temporal connectives is considered in [38]. The monitoring of LTL with only past-time temporal connectives ("past-time LTL") is studied in [39] and [40]. The main observation is that the semantics of past-time LTL can be defined recursively so that the monitoring algorithm only needs to look one step backwards. Markey and Schnoebelen [61] discuss the complexity of offline monitoring for LTL (with both future-time and past-time temporal connectives) specifications. Finkbeiner and Sipma [34] propose monitoring algorithms that make use of alternating automata. The online monitoring of LTL with both past-time and future-time temporal connectives is discussed in [71] (Section 4.2 on "Linear Temporal Logics").

Thati and Roşu [71] study online monitoring for MTL with past-time ("since") and future-time ("until") temporal connectives. For the special case of past-time MTL, the online monitoring algorithm has better time and space complexity. Reinbacher et al. [69] consider an online monitoring algorithm for past-time MTL specifications and its FPGA realization. The FPGA implementation of STL monitors is considered in [41].

Basin et al. [14] propose an online MDL monitoring algorithm over traces with timestamped data items. In order to deal with the unavoidable unbounded lookahead needed for temporal connectives such as $\mathsf{U}_{[0,\infty)}$, the proposed algorithm produces non-standard output, which does not only consist of Boolean values. Raszyk et al. [68] consider multi-head monitoring of MDL. A multi-head monitor has multiple pointers, which are called reading heads, in the system trace. The proposed MDL monitoring algorithm requires space that is exponential in the formula size, but it is independent of the constants in the time intervals I that annotate the temporal connectives.

There are various approaches for interpreting future-time connectives in the context of online monitoring. For example, [23] assumes the availability of a predictor to interpret future connectives, and [26] considers robustness intervals:

the tightest intervals which cover the robustness for all possible extensions of the available trace. Reelay [73] uses only past-time connectives. The transducer-based framework of [60] can be used to monitor temporal properties which depend on bounded future input by allowing some bounded delay in the output.

While this paper focuses on qualitative (Boolean) semantics, it may be possible to generalize the techniques to quantitative interpretations of the specification language. There is a large body of work on monitoring for quantitative interpretations of temporal formalisms. We discuss some related works below.

Fainekos and Pappas [31] define the robustness degree of satisfaction in terms of the distance of the signal from the set of desirable ones (or its complement). They also suggest an under-approximation of the robustness degree which can be effectively monitored. This is called the *robustness estimate* and is defined by induction on MTL/STL formulas, by interpreting conjunction (resp., disjunction) as inf (resp., sup) over the set $\mathbb{R}^{\pm\infty}$ of the extended real numbers.

In [42], the authors study a generalization of the robustness degree by considering idempotent semirings of real numbers. They also propose an online monitoring algorithm that uses symbolic weighted automata. While this approach computes the precise robustness degree in the sense of [31], the construction of the relevant automata incurs a doubly exponential blowup if one considers STL specifications. In [18], it is observed that an extension of the robustness estimate to bounded distributive lattices can be effectively monitored, but this is explored for the more limited formalism of past-time MTL. The paper [56] considers an algebraic semantics based on semirings, which is too abstract to define a metric notion of signed distance and robustness degree. Semirings are also used in [12], where the authors consider a spatio-temporal logic. Complete lattices are used in [57,58], where the focus is on online monitoring over continuous-time signals.

A key ingredient for the efficient monitoring of STL is a streaming algorithm for sliding-window maximum [25,22]. The tool Breach [24], which is used for the falsification of temporal specifications over hybrid systems, uses the sliding-maximum algorithm of [50].

The compositional construction of automata-based monitors from temporal specifications has also been considered in [52,53,33].

5 Conclusion

We have studied the problem of offline monitoring for DMTL, a formalism that combines MTL with regular expressions. We propose the first polynomial-time algorithm for offline monitoring. More specifically, our algorithm has time complexity $O(m^3 \cdot n)$, where m is the size of the DMTL formula and n is the length of the input trace. Other approaches for monitoring that translate the specification into a single automaton cannot achieve our polynomial time complexity bound, since the resulting automaton can be of at least exponential size.

Acknowledgments. This research was supported in part by the US National Science Foundation award CCF 2319572.

References

1. IEEE standard for Property Specification Language (PSL). IEEE Std 1850-2010 (Revision of IEEE Std 1850-2005) pp. 1–182 (2010). https://doi.org/10.1109/IEEESTD.2010.5446004
2. IEEE standard for SystemVerilog–unified hardware design, specification, and verification language. IEEE Std 1800-2023 (Revision of IEEE Std 1800-2017) pp. 1–1354 (2024). https://doi.org/10.1109/IEEESTD.2024.10458102
3. Abbas, H., Alur, R., Mamouras, K., Mangharam, R., Rodionova, A.: Real-time decision policies with predictable performance. Proceedings of the IEEE, Special Issue on Design Automation for Cyber-Physical Systems **106**(9), 1593–1615 (2018). https://doi.org/10.1109/JPROC.2018.2853608
4. Abbas, H., Rodionova, A., Mamouras, K., Bartocci, E., Smolka, S.A., Grosu, R.: Quantitative regular expressions for arrhythmia detection. IEEE/ACM Transactions on Computational Biology and Bioinformatics **16**(5), 1586–1597 (2019). https://doi.org/10.1109/TCBB.2018.2885274
5. Alur, R., Fisman, D., Mamouras, K., Raghothaman, M., Stanford, C.: Streamable regular transductions. Theoretical Computer Science **807**, 15–41 (2020). https://doi.org/10.1016/j.tcs.2019.11.018
6. Alur, R., Mamouras, K.: An introduction to the StreamQRE language. Dependable Software Systems Engineering **50**, 1–24 (2017). https://doi.org/10.3233/978-1-61499-810-5-1
7. Alur, R., Mamouras, K., Stanford, C.: Automata-based stream processing. In: Chatzigiannakis, I., Indyk, P., Kuhn, F., Muscholl, A. (eds.) Proceedings of the 44th International Colloquium on Automata, Languages, and Programming (ICALP 2017). Leibniz International Proceedings in Informatics (LIPIcs), vol. 80, pp. 112:1–112:15. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2017). https://doi.org/10.4230/LIPIcs.ICALP.2017.112
8. Alur, R., Mamouras, K., Stanford, C.: Modular quantitative monitoring. Proceedings of the ACM on Programming Languages **3**(POPL), 50:1–50:31 (2019). https://doi.org/10.1145/3290363
9. Alur, R., Mamouras, K., Ulus, D.: Derivatives of quantitative regular expressions. In: Aceto, L., Bacci, G., Bacci, G., Ingólfsdóttir, A., Legay, A., Mardare, R. (eds.) Models, Algorithms, Logics and Tools: Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday, LNCS, vol. 10460, pp. 75–95. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63121-9_4
10. Armoni, R., Fix, L., Flaisher, A., Gerth, R., Ginsburg, B., Kanza, T., Landver, A., Mador-Haim, S., Singerman, E., Tiemeyer, A., Vardi, M.Y., Zbar, Y.: The ForSpec temporal logic: A new temporal property-specification language. In: Katoen, J.P., Stevens, P. (eds.) TACAS 2002. LNCS, vol. 2280, pp. 296–311. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46002-0_21
11. Armoni, R., Korchemny, D., Tiemeyer, A., Vardi, M.Y., Zbar, Y.: Deterministic dynamic monitors for linear-time assertions. In: Havelund, K., Núñez, M., Roşu, G., Wolff, B. (eds.) FATES/RV 2006. LNCS, vol. 4262, pp. 163–177. Springer, Heidelberg (2006). https://doi.org/10.1007/11940197_11
12. Bartocci, E., Bortolussi, L., Loreti, M., Nenzi, L.: Monitoring mobile and spatially distributed cyber-physical systems. In: Proceedings of the 15th ACM-IEEE International Conference on Formal Methods and Models for System Design. pp. 146–155. MEMOCODE 2017, ACM, New York, NY, USA (2017). https://doi.org/10.1145/3127041.3127050

13. Bartocci, E., Deshmukh, J., Donzé, A., Fainekos, G., Maler, O., Ničković, D., Sankaranarayanan, S.: Specification-based monitoring of cyber-physical systems: A survey on theory, tools and applications. In: Bartocci, E., Falcone, Y. (eds.) Lectures on Runtime Verification: Introductory and Advanced Topics, LNCS, vol. 10457, pp. 135–175. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-75632-5_5
14. Basin, D., Krstić, S., Traytel, D.: Almost event-rate independent monitoring of metric dynamic logic. In: Lahiri, S., Reger, G. (eds.) RV 2017. LNCS, vol. 10548, pp. 85–102. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67531-2_6
15. Benveniste, A., Caspi, P., Edwards, S.A., Halbwachs, N., Le Guernic, P., de Simone, R.: The synchronous languages 12 years later. Proceedings of the IEEE **91**(1), 64–83 (2003). https://doi.org/10.1109/JPROC.2002.805826
16. Boulé, M., Zilic, Z.: Automata-based assertion-checker synthesis of PSL properties. ACM Transactions on Design Automation of Electronic Systems **13**(1), 4:1–4:21 (2008). https://doi.org/10.1145/1297666.1297670
17. Bustan, D., Korchemny, D., Seligman, E., Yang, J.: SystemVerilog Assertions: Past, present, and future SVA standardization experience. IEEE Design & Test of Computers **29**(2), 23–31 (2012). https://doi.org/10.1109/MDT.2012.2183336
18. Chattopadhyay, A., Mamouras, K.: A verified online monitor for metric temporal logic with quantitative semantics. In: Deshmukh, J., Ničković, D. (eds.) RV 2020. LNCS, vol. 12399, pp. 383–403. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-60508-7_21
19. D'Angelo, B., Sankaranarayanan, S., Sanchez, C., Robinson, W., Finkbeiner, B., Sipma, H.B., Mehrotra, S., Manna, Z.: LOLA: Runtime monitoring of synchronous systems. In: Proceedings of the 12th International Symposium on Temporal Representation and Reasoning (TIME 2005). pp. 166–174. IEEE, USA (2005). https://doi.org/10.1109/TIME.2005.26
20. Das, S., Mohanty, R., Dasgupta, P., Chakrabarti, P.: Synthesis of System Verilog Assertions. In: Proceedings of the Design Automation & Test in Europe Conference (DATE 2006). vol. 2, pp. 1–6. IEEE, USA (2006). https://doi.org/10.1109/DATE.2006.243776
21. De Giacomo, G., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: Twenty-Third International Joint Conference on Artificial Intelligence. pp. 854–860 (2013), https://www.ijcai.org/Proceedings/13/Papers/132.pdf
22. Deshmukh, J.V., Donzé, A., Ghosh, S., Jin, X., Juniwal, G., Seshia, S.A.: Robust online monitoring of signal temporal logic. Formal Methods in System Design **51**(1), 5–30 (2017). https://doi.org/10.1007/s10703-017-0286-7
23. Dokhanchi, A., Hoxha, B., Fainekos, G.: On-line monitoring for temporal logic robustness. In: Bonakdarpour, B., Smolka, S.A. (eds.) RV 2014. LNCS, vol. 8734, pp. 231–246. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11164-3_19
24. Donzé, A.: Breach, a toolbox for verification and parameter synthesis of hybrid systems. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 167–170. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14295-6_17
25. Donzé, A., Ferrère, T., Maler, O.: Efficient robust monitoring for STL. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 264–279. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_19
26. Dreossi, T., Dang, T., Donzé, A., Kapinski, J., Jin, X., Deshmukh, J.V.: Efficient guiding strategies for testing of temporal properties of hybrid systems. In:

Havelund, K., Holzmann, G., Joshi, R. (eds.) NFM 2015. LNCS, vol. 9058, pp. 127–142. Springer, Cham (2015)
27. Eisner, C.: PSL for runtime verification: Theory and practice. In: Sokolsky, O., Taşıran, S. (eds.) RV 2007. LNCS, vol. 4839, pp. 1–8. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-77395-5_1
28. Eisner, C., Fisman, D.: A Practical Introduction to PSL. Springer, Boston, MA (2007). https://doi.org/10.1007/978-0-387-36123-9
29. Eisner, C., Fisman, D.: Functional specification of hardware via temporal logic. In: Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.) Handbook of Model Checking, pp. 795–829. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-10575-8_24
30. Eisner, C., Fisman, D., Havlicek, J., Lustig, Y., McIsaac, A., Van Campenhout, D.: Reasoning with temporal logic on truncated paths. In: Hunt, W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 27–39. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45069-6_3
31. Fainekos, G.E., Pappas, G.J.: Robustness of temporal logic specifications for continuous-time signals. Theoretical Computer Science **410**(42), 4262–4291 (2009). https://doi.org/10.1016/j.tcs.2009.06.021
32. Faymonville, P., Finkbeiner, B., Schledjewski, M., Schwenger, M., Stenger, M., Tentrup, L., Torfah, H.: StreamLAB: Stream-based monitoring of cyber-physical systems. In: Dillig, I., Tasiran, S. (eds.) CAV 2019. LNCS, vol. 11561, pp. 421–431. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4_24
33. Ferrère, T., Maler, O., Ničković, D., Pnueli, A.: From real-time logic to timed automata. Journal of the ACM **66**(3), 19:1–19:31 (2019). https://doi.org/10.1145/3286976
34. Finkbeiner, B., Sipma, H.: Checking finite traces using alternating automata. Formal Methods in System Design **24**(2), 101–127 (2004). https://doi.org/10.1023/B:FORM.0000017718.28096.48
35. Ganardi, M., Hucke, D., König, D., Lohrey, M., Mamouras, K.: Automata theory on sliding windows. In: Niedermeier, R., Vallée, B. (eds.) Proceedings of the 35th Symposium on Theoretical Aspects of Computer Science (STACS 2018). Leibniz International Proceedings in Informatics (LIPIcs), vol. 96, pp. 31:1–31:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2018). https://doi.org/10.4230/LIPIcs.STACS.2018.31
36. Gelade, W.: Succinctness of regular expressions with interleaving, intersection and counting. Theoretical Computer Science **411**(31), 2987–2998 (2010). https://doi.org/10.1016/j.tcs.2010.04.036
37. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic. MIT Press (2000)
38. Havelund, K., Roşu, G.: Monitoring programs using rewriting. In: Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001). pp. 135–143. IEEE, USA (2001). https://doi.org/10.1109/ASE.2001.989799
39. Havelund, K., Roşu, G.: Synthesizing monitors for safety properties. In: Katoen, J.P., Stevens, P. (eds.) TACAS 2002. LNCS, vol. 2280, pp. 342–356. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46002-0_24
40. Havelund, K., Roşu, G.: Efficient monitoring of safety properties. International Journal on Software Tools for Technology Transfer **6**(2), 158–173 (2004). https://doi.org/10.1007/s10009-003-0117-6
41. Jakšić, S., Bartocci, E., Grosu, R., Kloibhofer, R., Nguyen, T., Ničković, D.: From signal temporal logic to FPGA monitors. In: 2015 ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE 2015). pp. 218–227. IEEE, USA (2015). https://doi.org/10.1109/MEMCOD.2015.7340489

42. Jakšić, S., Bartocci, E., Grosu, R., Ničković, D.: An algebraic framework for runtime verification. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **37**(11), 2233–2243 (2018). https://doi.org/10.1109/TCAD.2018.2858460
43. Javaheri, F.N., Morin-Allory, K., Borrione, D.: Synthesis of regular expressions revisited: From PSL SEREs to hardware. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **36**(5), 869–882 (2017). https://doi.org/10.1109/TCAD.2016.2600241
44. Kahn, G.: The semantics of a simple language for parallel programming. Information Processing **74**, 471–475 (1974)
45. Kleene, S.C.: Representation of events in nerve nets and finite automata. In: Shannon, C.E., McCarthy, J. (eds.) Automata Studies, pp. 3–41. No. 34 in Annals of Mathematics Studies, Princeton University Press (1956)
46. Kong, L., Yu, Q., Chattopadhyay, A., Le Glaunec, A., Huang, Y., Mamouras, K., Yang, K.: Software-hardware codesign for efficient in-memory regular pattern matching. In: Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation. pp. 733–748. PLDI 2022, ACM, New York, NY, USA (2022). https://doi.org/10.1145/3519939.3523456
47. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-Time Systems **2**(4), 255–299 (1990). https://doi.org/10.1007/BF01995674
48. Le Glaunec, A., Kong, L., Mamouras, K.: Regular expression matching using bit vector automata. Proceedings of the ACM on Programming Languages **7**(OOPSLA1), 92:1–92:30 (2023). https://doi.org/10.1145/3586044
49. Lee, E.A., Messerschmitt, D.G.: Static scheduling of synchronous data flow programs for digital signal processing. IEEE Transactions on Computers **C-36**(1), 24–35 (Jan 1987). https://doi.org/10.1109/TC.1987.5009446
50. Lemire, D.: Streaming maximum-minimum filter using no more than three comparisons per element. CoRR **abs/cs/0610046** (2006), http://arxiv.org/abs/cs/0610046
51. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Lakhnech, Y., Yovine, S. (eds.) FTRTFT/FORMATS 2004. LNCS, vol. 3253, pp. 152–166. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30206-3_12
52. Maler, O., Nickovic, D., Pnueli, A.: Real time temporal logic: Past, present, future. In: Pettersson, P., Yi, W. (eds.) FORMATS 2005. LNCS, vol. 3829, pp. 2–16. Springer, Heidelberg (2005). https://doi.org/10.1007/11603009_2
53. Maler, O., Nickovic, D., Pnueli, A.: From MITL to timed automata. In: Asarin, E., Bouyer, P. (eds.) FORMATS 2006. LNCS, vol. 4202, pp. 274–289. Springer, Heidelberg (2006). https://doi.org/10.1007/11867340_20
54. Mamouras, K.: Semantic foundations for deterministic dataflow and stream processing. In: Müller, P. (ed.) ESOP 2020. LNCS, vol. 12075, pp. 394–427. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-44914-8_15
55. Mamouras, K., Chattopadhyay, A.: Efficient matching of regular expressions with lookaround assertions. Proceedings of the ACM on Programming Languages **8**(POPL), 92:1–92:31 (2024). https://doi.org/10.1145/3632934
56. Mamouras, K., Chattopadhyay, A., Wang, Z.: Algebraic quantitative semantics for efficient online temporal monitoring. In: Groote, J.F., Larsen, K.G. (eds.) TACAS 2021. LNCS, vol. 12651, pp. 330–348. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72016-2_18

57. Mamouras, K., Chattopadhyay, A., Wang, Z.: A compositional framework for quantitative online monitoring over continuous-time signals. In: Feng, L., Fisman, D. (eds.) RV 2021. LNCS, vol. 12974, pp. 142–163. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-88494-9_8
58. Mamouras, K., Chattopadhyay, A., Wang, Z.: A compositional framework for algebraic quantitative online monitoring over continuous-time signals. International Journal on Software Tools for Technology Transfer **25**(4), 557–573 (2023). https://doi.org/10.1007/s10009-023-00719-w
59. Mamouras, K., Raghothaman, M., Alur, R., Ives, Z.G., Khanna, S.: StreamQRE: Modular specification and efficient evaluation of quantitative queries over streaming data. In: Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 693–708. PLDI 2017, ACM, New York, NY, USA (2017). https://doi.org/10.1145/3062341.3062369
60. Mamouras, K., Wang, Z.: Online signal monitoring with bounded lag. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **39**(11), 3868–3880 (2020). https://doi.org/10.1109/TCAD.2020.3013053
61. Markey, N., Schnoebelen, P.: Model checking a path. In: Amadio, R., Lugiez, D. (eds.) CONCUR 2003. LNCS, vol. 2761, pp. 251–265. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45187-7_17
62. Morin-Allory, K., Borrione, D.: Proven correct monitors from PSL specifications. In: Proceedings of the Design Automation & Test in Europe Conference (DATE 2006). vol. 1, pp. 1–6. IEEE, USA (2006). https://doi.org/10.1109/DATE.2006.244079
63. Morin-Allory, K., Boulé, M., Borrione, D., Zilic, Z.: Validating assertion language rewrite rules and semantics with automated theorem provers. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **29**(9), 1436–1448 (2010). https://doi.org/10.1109/TCAD.2010.2049150
64. Pnueli, A., Zaks, A.: PSL model checking and run-time verification via testers. In: Misra, J., Nipkow, T., Sekerinski, E. (eds.) FM 2006. LNCS, vol. 4085, pp. 573–586. Springer, Heidelberg (2006). https://doi.org/10.1007/11813040_38
65. Pnueli, A.: The temporal logic of programs. In: Proceedings of the 18th Annual Symposium on Foundations of Computer Science (SFCS 1977). pp. 46–57. IEEE, USA (1977). https://doi.org/10.1109/SFCS.1977.32
66. Pnueli, A., Zaks, A.: On the merits of temporal testers. In: Grumberg, O., Veith, H. (eds.) 25 Years of Model Checking: History, Achievements, Perspectives, LNCS, vol. 5000, pp. 172–195. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69850-0_11
67. Pratt, V.R.: Semantical considerations on Floyd-Hoare logic. In: Proceedings of the 17th IEEE Annual Symposium on Foundations of Computer Science (SFCS 1976). pp. 109–121. IEEE, USA (1976). https://doi.org/10.1109/SFCS.1976.27
68. Raszyk, M., Basin, D., Traytel, D.: Multi-head monitoring of metric dynamic logic. In: Hung, D.V., Sokolsky, O. (eds.) ATVA 2020. LNCS, vol. 12302, pp. 233–250. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59152-6_13
69. Reinbacher, T., Függer, M., Brauer, J.: Real-time runtime verification on chip. In: Qadeer, S., Tasiran, S. (eds.) RV 2012. LNCS, vol. 7687, pp. 110–125. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35632-2_13
70. Stephens, R.: A survey of stream processing. Acta Informatica **34**(7), 491–541 (1997). https://doi.org/10.1007/s002360050095
71. Thati, P., Roşu, G.: Monitoring algorithms for metric temporal logic specifications. Electronic Notes in Theoretical Computer Science **113**, 145–162 (2005).

https://doi.org/10.1016/j.entcs.2004.01.029, proceedings of the Fourth Workshop on Runtime Verification (RV 2004)
72. Turoňová, L., Holík, L., Lengál, O., Saarikivi, O., Veanes, M., Vojnar, T.: Regex matching with counting-set automata. Proceedings of the ACM on Programming Languages **4**(OOPSLA), 218:1–218:30 (2020). https://doi.org/10.1145/3428286
73. Ulus, D.: The Reelay monitoring tool. https://doganulus.github.io/reelay/ (2020), [Online; accessed August 20, 2020]
74. Vijayaraghavan, S., Ramanathan, M.: A Practical Guide for SystemVerilog Assertions. Springer, Boston, MA (2006). https://doi.org/10.1007/b137011
75. Wen, Z., Kong, L., Le Glaunec, A., Mamouras, K., Yang, K.: BVAP: Energy and memory efficient automata processing for regular expressions. In: Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2. pp. 151–166. ASPLOS 2024, ACM, New York, NY, USA (2024). https://doi.org/10.1145/3620665.3640412
76. Witharana, H., Lyu, Y., Charles, S., Mishra, P.: A survey on assertion-based hardware verification. ACM Computing Surveys **54**(11s), 225:1–225:33 (2022). https://doi.org/10.1145/3510578
77. Wolper, P.: Temporal logic can be more expressive. Information and Control **56**(1–2), 72–99 (1983). https://doi.org/10.1016/S0019-9958(83)80051-5