Monitoring Algorithms for Metric Temporal Logic Specifications

Prasanna Thati Grigore Roşu

Department of Computer Science
University of Illinois at Urbana Champaign, USA
{thati,grosu}@cs.uiuc.edu

December 2003

Abstract

Program execution traces can be so large in practical testing and monitoring applications that it would be very expensive, if not impossible, to store them for detailed analysis. Monitoring execution traces *without storing* them, can be a nontrivial matter for many specification formalisms, because complex formulae may require a considerable amount of information about the past. *Metric temporal logic* (MTL) is an extension of propositional linear temporal logic with discrete-time-bounded temporal operators. In MTL, one can specify time limits within which certain temporal properties must hold, thus making it very suitable to express real-time monitoring requirements. In this paper, we present monitoring algorithms for checking timestamped execution traces against formulae in MTL or certain important sublogics of it. We also present lower bounds for the monitoring problem, showing that the presented algorithms are asymptotically optimal.

1 Introduction

Runtime verification and *monitoring* have been proposed as lightweight formal verification methods [13] with the explicit goal of checking systems against their formal requirements while they execute. In most monitoring applications, execution traces are available only *incrementally* and they are *much larger* than the formulae against which they are checked. Storing an entire execution trace and then performing the formal analysis by having random access to the trace is very expensive and sometimes even impossible. For example, the monitor may lack resources, e.g., if it runs within an embedded system, or the monitor may be expected to react promptly when its requirements are violated, in order for the system to safely take a recovery or a shutdown action.

In this paper, we adopt the position that a *monitoring algorithm* does not store execution traces, but rather *consumes* the events as they are received from the monitored program. The problem of checking execution traces against temporal specifications is known to have very simple and efficient algorithms for several temporal logics, as shown for example in [19], but most of these algorithms assume that the entire execution trace is available beforehand, so they violate the assumptions for a monitoring algorithm.

In this paper, we investigate monitoring algorithms for the *metric temporal logic (MTL)* [1,15] and its sublogics. MTL is an extension of propositional linear temporal logic (LTL) that can refer to discrete-timed properties, and its models are timestamped state-sequences, thus making it an appealing formalism for expressing monitoring requirements in real-time systems. Besides the propositional operators, MTL allows future and past time linear temporal operators which are bounded by *discrete-time intervals*. For example, $\phi\mathcal{U}_{[3,7]}\psi$ states that ψ should hold between 3 and 7 time units from now, and until then ϕ should hold. One or both of the ends of an interval can be 0 or ∞. LTL can be seen as a special case of MTL where every interval is $[0, \infty)$. As introduced in [1], MTL also provides *congruences* that allow one to state that a formula should hold periodically with respect to an absolute time. We call these *absolute congruences* and support them in our MTL specifications as well, but in addition we introduce a novel variant that we call *relative congruence*. Relative congruences allow one to refer to moments that occur periodically starting with the *current* time.

We first present a general MTL monitoring algorithm based on the idea of transforming the MTL formula as each time-stamped observation (or event, for short) is received from the monitored program. The underlying principle of the algorithm is "resolve the past and derive the future". By "resolving the past" we mean that the MTL formula is transformed into an equivalent formula with the property that it has no past time operator rooted subformulae which are not guarded by other temporal operators. By "deriving the future" we mean that the MTL formula is transformed into a new MTL formula with the property that the current formula holds before processing the newly received event if and only if the derived formula holds after processing the event. We show that this MTL monitoring algorithm runs in space $O(m2^m)$ and takes time $O(m^3 2^{3m})$ for processing each event, where m equals $|\underline{\phi}|$ plus the sum of all the numeric constants occurring in ϕ, and $\underline{\phi}$ is ϕ with all the timing subscripts dropped. The reader may note that although exponential, these bounds are *independent* of the size of execution trace which is typically much larger than the formula being monitored[1]. We also show that the algorithm has better bounds for certain sublogics of MTL, including LTL. In fact, the bounds for past and future time LTL match the previously best known monitoring algorithms for these logics [11,12]. Finally, we derive lower bounds for monitoring MTL and its sublogics, which show that our algorithm is close to optimal.

In the interest of space, proofs of all the claims have been moved to appendix.

[1] If the integer constants in ϕ are represented in binary notation, then the bounds are doubly exponential on $|\phi|$

Related Work. MTL was introduced in [1], where its complexity of expressiveness is investigated. MTL is just one amongst a variety of extensions of linear temporal logics for specifying real-time systems (see [2] for a survey). Our idea of deriving an MTL formula with an observed event is an adaptation of the classical *tableaux* construction for temporal logics [21,9], where formulas in the current state represent constraints on the remainder of the input trace and are systematically propagated from the current state to the next. Drusinski [6] implements monitors for MTL formulae in his commercial Temporal Rover system, but the implementation and algorithmic details of this implementation are not available.

Java PathExplorer (JPaX) [10] is a NASA runtime verification system providing monitoring algorithms for past and future time LTL. MTL non-trivially generalizes LTL, and the motivation for generalizing the LTL monitoring algorithms to MTL is clear - one would often like to monitor not only *qualitative* specifications such as those that can be expressed in LTL, but also *quantitative* specifications that refer to timing constraints. The algorithms we present, when used on LTL specifications, are as efficient or more efficient than the corresponding specialized algorithms in JPaX. Eagle [4,3] is a fix-point based logic formalism designed around and for JPaX, combining temporal aspects and data, thus allowing one to define temporal operators and support time; however, its generality and lack of complexity analysis makes it hard to compare with our approach.

The complexity of checking a path against temporal formulas has been discussed in the context of "model-checking a path" in [19], but metric temporal logic was not covered there. We describe a dynamic programming based procedure in the style of [19], but argue that it is not a monitoring procedure because it has to store the entire execution trace. A tableaux based-simply exponential method to detect "bad prefixes" for a subset of LTL formulae is presented in [8]. We show that our general algorithm, when used on LTL formulae, not only has a better complexity than the algorithm in [8], but also works on any LTL formula, including both future and past operators. Using alternating automata in monitoring is also an appealing approach, started with [7] for LTL, but it is not clear how easily it can be used in the context of timed sequences of events.

2 Metric Temporal Logic

In this section, we briefly recap MTL; the reader is referred to [1] for more details. Given a finite set P of propositions, the set of MTL formulas is inductively defined as follows.

$$\phi \ := \ \mathit{true} \mid \mathit{false} \mid p \mid \phi_1 \wedge \phi_2 \mid \phi_1 \oplus \phi_2 \mid \circ_I \phi \mid \phi_1 \mathcal{U}_I \phi_2 \mid \odot_I \phi \mid \phi_1 \mathcal{S}_I \phi_2$$

where $p \in P$, and I is one of the following:

(1) An *interval* of the non-negative real line whose left and right end-points

are natural numbers or ∞. For a number n, the expression $\pm I \pm n$ denotes the interval $\{\pm y \pm n \mid y \in I\} \cap [0, \infty)$.
(2) A *relative congruence* expression $\approx_d c$ for integers $d \geq 2$ and $c \geq 0$. $y \in \approx_d c$ denotes $y = c \textsf{ mod } d$, and $\pm I \pm n$ the set $\{y \mid y = \pm c \pm n \textsf{ mod } d\}$.
(3) An *absolute congruence* expression $=_d c$ for integers $d \geq 2$ and $c \geq 0$. The expression $y \in =_d c$ denotes $y = c \textsf{ mod } d$ and $\pm I \pm n$ the set $\{y \mid y = c \textsf{ mod } d\}$.

We use exclusive disjunction instead of negation to simplify certain technicalities in the Section 3.

We assume that the integer constants that occur in a formula are encoded in binary format. We interpret MTL formulas over *finite timed state sequences.* A timed state sequence $\rho = (\pi, \tau)$ is a pair consisting of a finite sequence π of states $\pi_i \subseteq P$, and a finite sequence of natural numbers τ with $|\pi| = |\tau|$ and $\tau_i \leq \tau_{i+1}$ for each i. Define $|\rho| = |\pi|$. Intuitively, a sequence ρ represents a timed execution of a system and is understood as follows: at time τ_i the system was observed to be in state π_i. Let $\pi[i, j]$ denote $\pi_i \pi_{i+1} \dots \pi_j$, and similarly for $\tau[i, j]$, and let $\rho[i, j] = (\pi[i, j], \tau[i, j])$. Given a timed state sequence ρ and a position $1 \leq i \leq |\rho|$, we define what it means for (ρ, i) to satisfy a formula ϕ, written $(\rho, i) \models \phi$, as follows:

$(\rho, i) \models \mathit{true}$ is always true
$(\rho, i) \models \mathit{false}$ is always false
$(\rho, i) \models p$ iff $p \in \pi_i$
$(\rho, i) \models \phi_1 \wedge \phi_2$ iff $(\rho, i) \models \phi_1$ and $(\rho, i) \models \phi_2$
$(\rho, i) \models \phi_1 \oplus \phi_2$ iff exactly one of $(\rho, i) \models \phi_1$ and $(\rho, i) \models \phi_2$ holds
$(\rho, i) \models \circ_I \phi$ iff $i < |\rho|$, $(\rho, i+1) \models \phi$, and $\tau_{i+1} \in \tau_i + I$
$(\rho, i) \models \phi_1 \mathcal{U}_I \phi_2$ iff $(\rho, j) \models \phi_2$ for some $j \geq i$ with $\tau_j \in \tau_i + I$ and $(\rho, k) \models \phi_1$ for all $i \leq k < j$
$(\rho, i) \models \odot_I \phi$ iff $i > 1$, $(\rho, i-1) \models \phi$, and $\tau_{i-1} \in \tau_i - I$
$(\rho, i) \models \phi_1 \mathcal{S}_I \phi_2$ iff $(\rho, j) \models \phi_2$ for some $j \leq i$ with $\tau_j \in \tau_i - I$ and $(\rho, k) \models \phi_1$ for all $j < k \leq i$

We write $\rho \models \phi$ as shorthand for $(\rho, 1) \models \phi$. Note that intervals and relative congruences express timing constraints *relative* to the "current" time, while absolute congruences refer to the absolute time. For example, at position i, $\circ_{[m,n]}\mathit{true}$ holds if $\tau_{i+1} - \tau_i \in [m, n]$, and $\circ_{\approx_d c}\mathit{true}$ holds if $\tau_{i+1} - \tau_i = c \textsf{ mod } d$, while $\circ_{=_d c}\mathit{true}$ holds if $\tau_{i+1} = c \textsf{ mod } d$. MTL as originally defined in [1] contains only absolute congruences as primitives, but we introduce relative congruences since they naturally arise in many specifications. The following are some useful abbreviations:

$$\neg\phi = \mathit{true} \oplus \phi \qquad \phi_1 \vee \phi_2 = \phi_1 \oplus \phi_2 \oplus (\phi_1 \wedge \phi_2) \qquad \Diamond_I \phi = \mathit{true}\, \mathcal{U}_I \phi$$

$$\Box_I \phi = \neg \Diamond_I \neg \phi \qquad \diamondminus_I \phi = \mathit{true}\, \mathcal{S}_I \phi \qquad \text{æ}_I \phi = \neg \diamondminus_I \neg \phi$$

We write $\mathcal{U}$ for $\mathcal{U}_{[0,\infty)}$, $\mathcal{U}_{\leq m}$ for $\mathcal{U}_{[0,m]}$, $\mathcal{U}_{>m}$ for $\mathcal{U}_{(m,\infty)}$, $\mathcal{U}_m$ for $\mathcal{U}_{[m,m]}$, and similarly for the other temporal operators. Note that the standard LTL falls as a degenerate sublogic of MTL where only the interval $[0, \infty)$ is allowed, which amounts to "ignoring" the timestamps in execution traces.

Recursive definitions of satisfaction typically lead to efficient dynamic programming based algorithms for checking membership of a trace in the set of traces defined by a formula [19]. An equivalent recursive definition of the semantics above can be easily devised:

$$\begin{array}{ll}
(\rho, i) \vDash \phi_1 \mathcal{U}_I \phi_2 & \text{iff } 0 \in I \text{ and } (\rho, i) \vDash \phi_2, \text{ or } i < |\rho| \text{ and } (\rho, i) \vDash \phi_1 \text{ and} \\
& \quad (\rho, i+1) \vDash \phi_1 \mathcal{U}_{I'} \phi_2 \text{ where } I' = I - \tau_{i+1} + \tau_i \\
(\rho, i) \vDash \phi_1 \mathcal{S}_I \phi_2 & \text{iff } 0 \in I \text{ and } (\rho, i) \vDash \phi_2, \text{ or } i > 1 \text{ and } (\rho, i) \vDash \phi_1 \text{ and} \\
& \quad (\rho, i-1) \vDash \phi_1 \mathcal{S}_{I'} \phi_2 \text{ where } I' = I - \tau_i + \tau_{i-1}
\end{array}$$

An efficient *dynamic programming* algorithm for testing $(\rho, i) \vDash \phi$ follows naturally: allocate a table d of size $|\rho| \times |\phi| \times c$ of bits, where c is the largest integer constant occurring in ϕ. The idea is that $d(i, j, c)$ is 1 if and only if (ρ, i) satisfies the formula ψ that is obtained from the jth subformula of ϕ by subtracting c from the interval at the root of the subformula (if any). By carefully traversing the table d, one can fill it in time linear on its size. See [19] for related algorithms for other temporal logics. However, such an algorithm is *highly undesirable* in the context of monitoring, because it not only requires the entire trace to be stored, which is intolerable while monitoring very long executions, but it also is not online in nature.

3 Monitoring MTL Formulae over Finite Traces

In this section, we present our main monitoring algorithm for MTL.

3.1 Resolving the Past and Deriving the Future

We define two mutually recursive formula transformations, one for past and one for future. The transformation $[\rho, i]\phi$ resolves all the top-level past-time operators in ϕ according to the events until the i^{th} one in ρ, i.e. according to the events observed so far. The resulting formula is an equivalent formula that does not contain any unguarded past-time operators, i.e. every top-level temporal operator is a future-time operator (see Lemma 3.2). The transformation $\phi\{\rho, i\}$ derives the formula ϕ with respect to the i^{th} event in ρ, so that the resulting formula holds after the event if and only if ϕ holds before the event (see Lemma 3.2).

Definition 3.1 Let ρ be a timed state sequence, and $1 \leq i \leq |\rho|$. We define

$$\begin{array}{ll}
[\rho, i]\mathit{true} = \mathit{true} & [\rho, i]\mathit{false} = \mathit{false} \\
[\rho, i]p = p \in \pi_i & [\rho, i](\phi_1 \wedge \phi_2) = ([\rho, i]\phi_1) \wedge ([\rho, i]\phi_2) \\
[\rho, i](\phi_1 \oplus \phi_2) = ([\rho, i]\phi_1) \oplus ([\rho, i]\phi_2) & [\rho, i]\circ_I \phi = \circ_I \phi \\
[\rho, i](\phi_1 \mathcal{U}_I \phi_2) = \phi_1 \mathcal{U}_I \phi_2 & [\rho, i]\odot_I \phi = \text{if } i = 1 \text{ or } \tau_{i-1} \notin \tau_i - I \\
& \qquad \text{then } \mathit{false} \text{ else } [\rho, i](\phi\{\rho, i-1\})
\end{array}$$

$$[\rho,i](\phi_1\mathcal{S}_I\phi_2) = \left(\begin{array}{l}\text{if } 0\in I \text{ then } [\rho,i]\phi_2\\ \text{else } \mathit{false}\end{array}\right) \vee \left(\begin{array}{l}\text{if } i=1 \text{ then } \mathit{false}\\ \text{else } [\rho,i](\phi_1 \wedge (\phi_1\mathcal{S}_{I'}\phi_2)\{\rho,i-1\})\end{array}\right)$$

$$\text{where } I' = I - \tau_i + \tau_{i-1}$$

$$\begin{array}{ll}
\mathit{true}\{\rho,i\} = \mathit{true} & \mathit{false}\{\rho,i\} = \mathit{false}\\
p\{\rho,i\} = p \in \pi_i & (\phi_1\wedge\phi_2)\{\rho,i\} = (\phi_1\{\rho,i\})\wedge(\phi_2\{\rho,i\})\\
(\phi_1\oplus\phi_2)\{\rho,i\} = (\phi_1\{\rho,i\})\oplus(\phi_2\{\rho,i\}) & (\odot_I\phi)\{\rho,i\} = ([\rho,i]\odot_I\phi)\{\rho,i\}\\
(\phi_1\mathcal{S}_I\phi_2)\{\rho,i\} = ([\rho,i](\phi_1\mathcal{S}_I\phi_2))\{\rho,i\} &
\end{array}$$

$$(\circ_I\phi)\{\rho,i\} = \text{if } i=|\rho| \text{ or } \tau_{i+1}\notin\tau_i+I \text{ then } \mathit{false} \text{ else } \phi$$

$$(\phi_1\mathcal{U}_I\phi_2)\{\rho,i\} = \left(\begin{array}{l}\text{if } 0\in I \text{ then } \phi_2\{\rho,i\}\\ \text{else } \mathit{false}\end{array}\right) \vee \left(\begin{array}{l}\text{if } i=|\rho| \text{ then } \mathit{false}\\ \text{else } (\phi_1\{\rho,i\}\wedge(\phi_1\mathcal{U}_{I'}\phi_2))\end{array}\right)$$

$$\text{where } I' = I - \tau_{i+1} + \tau_i$$

From now on we adopt the convention that the operators $[\rho,i]\cdot$ and $\cdot\{\rho,i\}$ bind weaker than all the logical connectives. E.g., $[\rho,i]\phi_1\mathcal{S}_I\phi_2$ denotes $[\rho,i](\phi_1\mathcal{S}_I\phi_2)$, and $\phi_1\mathcal{S}_I\phi_2\{\rho,i\}$ denotes $(\phi_1\mathcal{S}_I\phi_2)\{\rho,i\}$.

Let $\mathcal{F}(\phi)$ be the set of all subformulae of ϕ that are either rooted at a temporal operator or are atomic propositions. Let $\hat{\mathcal{F}}(\phi)$ be the set of formulas in $\mathcal{F}(\phi)$ which have an occurrence in ϕ that is not guarded by a temporal operator, i.e. formulas in $\mathcal{F}(\phi)$ that are at the "top-level". Let $\underline{\phi}$ denote the formula obtained by dropping all the intervals in ϕ (i.e., implicitly replacing every interval with $[0,\infty)$). For instance, for $\phi = p_1\mathcal{U}_I(p_2\wedge p_3)$, we have $\mathcal{F}(\phi) = \{p_1,p_2,p_3,\phi\}$, $\hat{\mathcal{F}}(\phi)=\{\phi\}$, and $\underline{\phi} = p_1\mathcal{U}(p_2\wedge p_3)$. Let

$$\begin{aligned}
\mathcal{F}^+(\phi) &= \mathcal{F}(\phi)\ \cup\ \{\phi_1\mathcal{U}_{I'}\phi_2 \mid \phi_1\mathcal{U}_I\phi_2\in\mathcal{F}(\phi), I' = I-n \textit{ for some } n\}\\
\mathcal{F}^-(\phi) &= \mathcal{F}(\phi)\ \cup\ \{\phi_1\mathcal{S}_{I'}\phi_2 \mid \phi_1\mathcal{S}_I\phi_2\in\mathcal{F}(\phi), I' = I-n \textit{ for some } n\}\\
\mathcal{F}^\pm(\phi) &= \mathcal{F}^+(\phi)\cup\mathcal{F}^-(\phi)
\end{aligned}$$

The following lemma states certain properties of the formula transformations in Definition 3.1, that we informally claimed earlier in this section.

Lemma 3.2 *For a timed state sequence ρ and $1\le i\le|\rho|$,*

(i) $\mathcal{F}([\rho,i]\phi)\subseteq\mathcal{F}^+(\phi)$. *Further, if ϕ is rooted at a past time temporal operator then $\mathcal{F}([\rho,i]\phi)\subseteq\mathcal{F}^+(\phi)\setminus\phi$.*

(ii) *Every formula in $\hat{\mathcal{F}}([\rho,i]\phi)$ is rooted at a future time temporal operator.*

(iii) $(\rho,i)\vDash\phi$ *if and only if* $(\rho,i)\vDash[\rho,i]\phi$.

(iv) $\mathcal{F}(\phi\{\rho,i\})\subseteq\mathcal{F}^+(\phi)$. *Further, if ϕ is rooted at a past time temporal operator then $\mathcal{F}(\phi\{\rho,i\})\subseteq\mathcal{F}^+(\phi)\setminus\phi$.*

(v) *$\mathcal{F}(\phi\{\rho, |\rho|\})$ is empty, i.e. $\phi\{\rho, |\rho|\}$ is equivalent to true or false.*

(vi) *For $i < |\rho|$, $(\rho, i) \vDash \phi$ if and only if $(\rho, i+1) \vDash \phi\{\rho, i\}$, and $(\rho, |\rho|) \vDash \phi$ if and only if $\phi\{\rho, |\rho|\}$.* □

3.2 Canonical Forms

While transforming the MTL formulae after every event, it is crucial to keep the size of the transformed formulae small. An important component of our monitoring algorithm is a procedure which keeps formulae in a canonical form that is guaranteed not to grow larger than exponential in size of the original formula. Moreover, the formula representations can be updated also in simple exponential time with the size of the original formula. As explained below, the correctness of this procedure is based on a result by Hsiang [14], regarding propositional calculus as a Boolean ring by reducing propositions to canonical forms consisting of exclusive disjunction of conjunctions. The encoding of propositions that follows is specialized for the particular operations required by our main monitoring algorithm. Whether BDDs [5] or other more standard encodings, such as CNF or DNF, can also be viable possibilities in our monitoring framework, as well as the viceversa, namely whether our encoding can outperform the others in some situations, are definitely issues deserving further investigation. However, for the time being we prefer the Boolean ring encoding presented next because it relieves us from dealing with negations and, more importantly, it allows very simple and efficient implementations of several propositional operations, including a non-trivial substitution.

Let $\mathcal{P} = \{p_1, p_2, ..., p_m\}$ be a set of "parameters", and let $Prop^{\oplus\wedge}(\mathcal{P})$ be the set of *$\oplus\wedge$-canonical propositions over symbols in* $\mathcal{P} \cup \{true, false\}$. By $\oplus\wedge$-canonical it is meant canonical modulo the associativity and commutativity equations of $\oplus$ and $\wedge$, using the other equations below as *rewriting rules*:

$$
\begin{array}{llll}
(1) & (\phi_1 \wedge \phi_2) \wedge \phi_3 = \phi_1 \wedge (\phi_2 \wedge \phi_3) & (2) & \phi_1 \wedge \phi_2 = \phi_2 \wedge \phi_1 \\
(3) & \phi \wedge true = \phi & (4) & \phi \wedge \phi = \phi \\
(5) & (\phi_1 \oplus \phi_2) \oplus \phi_3 = \phi_1 \oplus (\phi_2 \oplus \phi_3) & (6) & \phi_1 \oplus \phi_2 = \phi_2 \oplus \phi_1 \\
(7) & \phi \oplus false = \phi & (8) & \phi \oplus \phi = false \\
(9) & (\phi_1 \oplus \phi_2) \wedge \phi_3 = (\phi_1 \wedge \phi_3) \oplus (\phi_2 \wedge \phi_3) & &
\end{array}
$$

Let E be the set of equations above. Since $\oplus$ and $\wedge$ are commutative and associative, we can unambiguously write expressions such as $\phi_1 \oplus \ldots \oplus \phi_n$ and $\phi_1 \wedge \ldots \wedge \phi_n$, or $\oplus_{i=1}^n \phi_i$ and $\wedge_{i=1}^n \phi_i$, respectively. Due to the Church-Rosser and termination of the AC-rewriting system above [14], it is not hard to see that $\oplus\wedge$-canonical forms have unique forms $\oplus_{i \in I} \wedge_{j \in J_i} C_{ij}$, where

- $C_{ij} \in \mathcal{P} \cup \{true, false\}$ for all $i \in I$ and $j \in J_i$;
- for each $i \in I$, the elements C_{ij} form a *set*, that is, $C_{ij} \neq C_{ik}$ for $j \neq k$;

- the sets $\{C_{ij} \mid j \in J_i\}$ also form a set;
- *true* $\notin \{C_{ij} \mid j \in J_i\}$ except when $|J_i| = 1$, and if this is the case then i is the only index in I with this property;
- *false* $\notin \{C_{ij} \mid j \in J_i\}$ except when $|J_i| = 1$ and $|I| = 1$.

Notice that $|I| \leq 2^m$ and $|J_i| \leq m$. Because of the above, one can regard any $\oplus\wedge$-canonical form as a *set of sets* of elements in $\mathcal{P}$. In order for this to work, we need to adopt the standard convention that $\wedge_{j\in\emptyset}$ is *true*, and that $\oplus_{i\in\emptyset}$ is *false*.

We next describe how $\oplus\wedge$-canonical propositions over parameters in $\mathcal{P} = \{p_1, p_2, ..., p_m\}$ can be encoded on 2^m bits, and also how several common operations on propositions encoded this way can be performed efficiently. Since $\oplus\wedge$-canonical propositions can be seen as sets of sets of at most m elements, we start by encoding each subset P of $\mathcal{P}$ by a sequence of m bits b with the property that $b[j] = 1$ if and only if $p_j \in P$. Now each b corresponds to a number between 0 and 2^{m-1}, which allows us to assign exactly 2^m bits to any $\oplus\wedge$-canonical proposition ϕ; the idea being that the i^{th} bit is 1 if and only if the set corresponding to the binary m-bit representation of i corresponds to one of the conjuncts of ϕ. A sequence of 2^m zeros encodes the formula *false*; if ϕ is of the form *true* $\oplus\, \phi'$, then the bit corresponding to $i = 0$ in the 2^m-bit representation of ϕ is 1. In particular, the proposition true is encoded as 1, regarded as a 2^m-bit number.

Let us next define corresponding bitwise transformations for the various operations on $\oplus\wedge$-canonical propositions. From now, due to the one-to-one correspondence, we make no distinction between a $\oplus\wedge$-canonical proposition and its binary representation. Therefore, in particular, we say $\phi[i] = 1$ if and only if ϕ contains the conjunct formed with the corresponding propositions in the binary representation of i.

Exclusive disjunction. For $\oplus\wedge$-canonical propositions ϕ and ψ, the binary representation of the canonical form of $\phi \oplus \psi$, is nothing but the bitwise `xor` operation applied to the binary representations of ϕ and ψ. Indeed, each bit in the binary representation corresponds to a set of propositions forming a corresponding conjunct, and by equations (8) and (7), the same set cannot appear twice in a normal form. This simple procedure takes time $O(m2^m)$, because one also needs to increment the m-bit counter traversing the two 2^m-bit sequences.

Conjunction. For $\oplus\wedge$-canonical propositions ϕ and ψ, we claim that the following $O(m2^{2m})$ procedure calculates the binary representation of the $\oplus\wedge$-canonical form of their conjunction in ξ:

```
$\xi = 0$
for $i, j = 0$ to $2^m - 1$
    $k = binary(i)$ or $binary(j)$
    $\xi[k] = \xi[k]$ xor $(\phi[i]$ and $\psi[j])$
```

The operators `or`, `xor` and `and` above are bitwise, and $binary(i)$ is the binary representation of i. If i and j are already in binary representation then the increments of the for loop, and the calculation of k and $\xi[k]$, take time $O(m)$. To keep the notation simple, we ambiguously let $\phi \wedge \psi$ also denote the 2^m-bit ξ calculated by the procedure above.

Other boolean operators. One can define other boolean operations as well. For example, $\neg\phi$ can be calculated in constant time, by `xor`-ing the first bit of ϕ (the one corresponding to *true*) with 1. Similarly, $\phi \wedge p_k$, for some $p_k \in \mathcal{P}$, can be calculated like in the general conjunction $\phi \wedge \psi$, but with the optimization that since $j = 2^k$ is the only bit in ψ that is a 1, the conjunction can be computed in time $O(m2^m)$. Finally, since standard disjunction $\phi \vee \psi$ reduces to $\phi \oplus \psi \oplus (\phi \wedge \psi)$, it can be computed in time $O(m2^{2m})$.

Substitution. A very frequent operation on propositions that we will use, is that of applying a *substitution*. More precisely, suppose that $T\colon [1, m] \to [0, 2^m - 1]$ is a map assigning to parameters $p_j \in \mathcal{P}$, abstracted by their index, a $\oplus\wedge$-canonical proposition in binary representation. Now given another $\oplus\wedge$-canonical proposition in binary representation, say ϕ, the problem is to efficiently calculate the proposition obtained by the substitution given by T to the formula ϕ, after putting it in $\oplus\wedge$-canonical form. The following code running in time $O(m^2 2^{3m})$ calculates the binary representation of this proposition in ξ:

```
$\xi = 0$
for $i = 1$ to $2^m - 1$
    if $\phi[i]$ then $\gamma = 1$ (as a $2^m$-bit number)
                  for $j = 1$ to $m$
                      if $binary(i)[j]$ then $\gamma = \gamma \wedge T[j]$
                  $\xi = \xi \oplus \gamma$
```

The outer loop and conditional traverse all conjuncts of ϕ; then the inner loop and conditional traverse all the propositions occurring in a conjunct, and apply them the substitution incrementally, propagating the $\oplus$ bottom-up, due to the distributivity rule (9). Finally, the newly obtained proposition γ which is in $\oplus\wedge$-canonical form, needs to be merged with the already existing similar propositions obtained for different i. Let $\mathsf{subst}(\phi, T)$ be the ξ calculated above.

All the above allow us to state the following important result:

Theorem 3.3 *$\oplus\wedge$-canonical propositions over parameters in $\mathcal{P} = \{p_1, p_2, ..., p_m\}$ can be stored in space 2^m such that the operations of exclusive disjunction, conjunction and substitution, run in time $O(m2^m)$, $O(m2^{2m})$ and $O(m^2 2^{3m})$, respectively.* □

3.3 Monitoring MTL Formulas

The MTL monitoring algorithm can be now relatively easily defined, following the mutually recursive formula transformation relations in Definition 3.1, and

```
1    monitor($\phi, \rho$)
2        allocate $R[1 \ldots m]$, $D[1 \ldots m]$
3        for $i = 1$ to $|\rho|$ do
4            for $j = 1$ to $m$ do $R[j]$ = resolve(formula($j$), $R$, $D$, $\rho$, $i$)
5            for $j = 1$ to $m$ do $D[j]$ = derive(formula($j$), $R$, $D$, $\rho$, $i$)
6            $\phi$ = subst($\phi$, $D$)
7            if $\phi$ = false or $\phi$ = true then break
8        return $\phi$
9    end monitor
```

Fig. 1. The MTL monitoring algorithm over finite timed state sequences.

taking advantage of the 2^m-bit representations of propositions in $\oplus\wedge$-canonical forms and the efficient implementation of basic propositional operations. Our algorithm is essentially a *dynamic programming* algorithm that implements the recursive relations in Definition 3.1.

Given a formula ϕ in $\oplus\wedge$-canonical form, which one can accomplish off-line, using a procedure like Hsiang's [14], let $m = |\mathcal{F}^{\pm}(\phi)|$. Note that $m \leq |\underline{\phi}| + \Sigma_\phi$, where Σ_ϕ is the sum of all the numeric constants associated to each occurrence of $\mathcal{U}_I$ and $\mathcal{S}_I$ in ϕ as follows: if $I = [m, n]$ then n; if $I = [m, \infty]$ then m; if $I = \approx_d c$ then d; 0 otherwise. For each $\psi \in \mathcal{F}^{\pm}(\phi)$ assign a unique integer $1 \leq \mathsf{index}(\psi) \leq m$ s.t. whenever $\psi_1 \in \mathcal{F}^{\pm}(\psi_2)$ then $\mathsf{index}(\psi_1) \leq \mathsf{index}(\psi_2)$. For $1 \leq i \leq m$ let $\mathsf{formula}(i)$ return ψ such that $\mathsf{index}(\psi) = i$.

Figure 1 shows the pseudocode of the main monitoring algorithm. This procedure always keeps the formulas that it handles in 2^m-bit canonical form. These canonical forms will be over parameters $\mathcal{F}^{\pm}(\phi) = \{\psi_1, \ldots, \psi_m\}$ where $\mathsf{index}(\psi_i) = i$, and are encoded as described in Subsection 3.2. Note that the initial 2^m-bit representation of ϕ can be calculated in time $O(m2^m)$.

The monitoring procedure maintains two arrays, R and D, each of length m, which are updated by the loop in line 3, each time the next element in the observed timed sequence ρ is available. If $\mathsf{formula}(j) = \psi$ then after the i^{th} iteration $R[j]$ will be $[\rho, i]\psi$ and $D[j]$ will be $\psi\{\rho, i\}$. Further, $R[j]$ and $D[j]$ are kept in canonical form. We note that it is possible to use the same parameter set $\{\psi_1, \ldots, \psi_m\}$ for encoding the canonical representation of $R[j]$ and $D[j]$ because as a consequence of Lemma 3.2 $\mathcal{F}(D[j]), \mathcal{F}(R[j]) \subseteq \mathcal{F}^{\pm}(\phi)$. The arrays R and D are computed using two mutually recursive procedures - resolve and derive - shown in Figure 2. These follow Definition 3.1 and hence are self explanatory. Note that the computation of R in the current iteration uses D from the previous iteration, and the computation of D is the current iteration uses R from the current iteration. Thus, in each iteration R is updated before D.

Theorem 3.4 *The procedure* monitor(ϕ, ρ) *returns true iff* $\rho \models \phi$. *It takes space* $O(m2^m)$ *and time* $O(|\rho| m^3 2^{3m})$, *where* $m = |\mathcal{F}^{\pm}(\phi)| \leq |\underline{\phi}| + \Sigma_\phi$. □

```
resolve($\phi, R, D, \rho, i$)
    case $\phi$ of
      $p$                              :  $\psi = p \in \pi_i$
      $\odot_I \phi_1$                 :  if $i = 1$ or $\tau_{i-1} \notin \tau_i - I$ then $\psi =$ false
                                          else $\psi =$ subst(subst($\phi_1, D$), $R$)
      $\phi_1 \mathcal{S}_I \phi_2$    :  if $0 \in I$ then $\psi_2 =$ subst($\phi_2, R$) else $\psi_2 =$ false
                                          if $i = 1$ then $\psi_1 =$ false
                                          else $I' = I - \tau_i + \tau_{i-1}$
                                               $\psi_1 =$ subst($\phi_1, R$) $\wedge$ subst($D$[index($\phi_1 \mathcal{S}_{I'} \phi_2$)], $R$)
                                          $\psi = \psi_1 \vee \psi_2$
      $\circ_I \phi_1, \phi_1 \mathcal{U}_I \phi_2$ :  $\psi = \phi$
    return $\psi$
end resolve


derive($\phi, R, D, \rho, i$)
    case $\phi$ of
      $p$                              :  $\psi = p \in \pi_i$
      $\circ_I \phi_1$                 :  if $i = |\rho|$ or $\tau_{i+1} \notin \tau_i + I$ then $\psi =$ false else $\psi = \phi_1$
      $\phi_1 \mathcal{U}_I \phi_2$    :  if $0 \in I$ then $\psi_1 =$ subst($\phi_2, D$) else $\psi_1 =$ false
                                          if $i = |\rho|$ then $\psi_2 =$ false
                                          else $I' = I - \tau_{i+1} + \tau_i$
                                               $\psi_2 =$ subst($\phi_1, D$) $\wedge \phi_1 \mathcal{U}_{I'} \phi_2$
                                          $\psi = \psi_1 \vee \psi_2$
      $\odot_I \phi_1, \phi_1 \mathcal{S}_I \phi_2$ :  $\psi =$subst($R$[index($\phi$)], $D$)
    return $\psi$
end derive
```

Fig. 2. Resolving the past and deriving the future.

We end this section with a couple of observations. First, note that in the i^{th} iteration the monitoring procedure only access ρ_{i-1}, ρ_i and ρ_{i+1}, and thus we need not store the entire timed sequence observed. Second, the space requirement of the procedure can be further optimized by having entries in R for only those $\psi \in \mathcal{F}^{\pm}(\phi)$ that are rooted at past time operators. This is because the entries in R for atomic propositions coincide with the corresponding entries in D, and the entries for ψ rooted at future time operators contain ψ itself.

4 Stronger Performance Results for Sublogics of MTL

A more refined performance analysis of the monitoring algorithm for certain sublogics of MTL shows that the algorithm has much better performance over these sublogics in comparison to entire MTL. We consider two such sublogics - MTL with only past time operators, and LTL.

4.1 MTL with Only Past Time Operators

A large class of safety properties, often called *canonical safety* [18] properties, can be expressed compactly and naturally as a past time formula ϕ which has to hold at every moment in an execution trace. In MTL, this is the same as checking for $\Box\phi$ ($\Box_{[0,\infty)}\phi$). Such properties can be monitored very efficiently:

Theorem 4.1 *Suppose ϕ is an MTL formula with only past time operators. Then* $\mathsf{monitor}(\Box\phi, \rho)$ *takes time $O(|\rho|m)$ and space $O(m)$, where $m = |\mathcal{F}^{\pm}(\phi)|$.*□

The reader can check that monitoring MTL with only future time operators has the same complexity as MTL with both future and past time operators.

4.2 Linear Temporal Logics

The monitoring algorithm for MTL can be specialized to obtain an algorithm for LTL. Recall that LTL formulas can be seen as MTL formulas with only intervals of form $[0, \infty)$; although LTL formulas are interpreted over (untimed) state sequences. The monitoring algorithm can be specialized by simply dropping all references to time in the `resolve` and `derive` procedures.

As corollaries to Theorems 3.4 and 4.1 we get that LTL with both past and future time operators can be monitored in time $O(|\rho||\phi|^3 2^{3|\phi|})$ and space $O(|\phi|2^{|\phi|})$ (note that $\underline{\phi} = \phi$), while LTL with only past time operators can be monitored in time $O(|\rho||\phi|)$ and space $O(|\phi|)$. Indeed monitoring algorithms with the same complexity bounds are known for LTL with only future time operators [11] and LTL with only past time operators [12]. But the algorithm for LTL with both past and future time operators seems to be novel.

5 Exponential Lower Bounds for Space

We now derive some space lower bound results which show that the our monitoring algorithm for MTL and its sublogics is close to optimal.

5.1 Lower Bounds for MTL

Consider a monitoring scenario with only one proposition and hence only two states, say 0 and 1. For natural numbers k, n define the following language of finite timed sequences $\rho = (\pi, \tau)$:

$$\begin{aligned}\mathcal{L}_{k,n} = \{\rho \mid \tau_1 = \ldots = \tau_k,\ \tau_{i+k} = \tau_i + 1,\ \exists l \text{ s.t. } |\pi| = lkn, \text{ and}\\ \exists i < l \text{ s.t. } \pi[(i-1)kn + 1, ikn] = \pi[(l-1)kn + 1, lkn]\}\end{aligned}$$

$\mathcal{L}_{k,n}$ contains only those timed sequences whose length is a multiple of kn, and where time increases by one every k steps. Further, if the underlying state sequence is $w_1 \ldots w_m$, where each w_i is of length kn, then $w_m = w_i$ for some $i < m$.

Lemma 5.1 *Any monitoring algorithm for $\mathcal{L}_{k,n}$ requires space $\Omega(2^{kn})$.* □

Now, we give an MTL formula $\phi_{k,n}$ that defines the language $\mathcal{L}_{k,n}$. The following 'macros' will be useful for this purpose.

$$\mathsf{tick} = \circ_1 \mathit{true} \qquad\qquad \mathsf{end} = \neg \circ \mathit{true}$$

$$\mathsf{startcell} = \circ_0^{k-1} \mathit{true} \qquad\qquad \mathsf{lastword} = \neg \Diamond_n \mathit{true}$$

where we write $\circ_0^{k-1}$ for a sequence of $\circ_0$ operators of length $k-1$. The predicate tick is true at a position if the time in the next position is exactly one more than the time in the current position, while end holds only at the last position in a timed sequence. The predicate $\mathsf{startcell}$ holds at a position only if the time does not advance in the next $k-1$ steps, while $\mathsf{lastword}$ holds at a position if the time in all the subsequent positions is at most $n-1$ units more than the time at the current position. The idea is that, since we are interested in the timed sequences where time increases only every k positions, $\mathsf{startcell}$ is true at positions that are one more than a multiple of k. In addition, since we are interested in sequences whose length is a multiple of kn, $\mathsf{lastword}$ is true only in the last kn positions. Define

$$\phi_{k,n} = \psi^1_{k,n} \wedge \psi^2_{k,n} \wedge \psi^3_{k,n}$$

where $\psi^i_{k,n}$ are defined as follows.

$$\psi^1_{k,n} = \circ_0^{k-1}(\mathsf{tick} \vee \mathsf{end}) \ \wedge \ \Box(\mathsf{tick} \rightarrow \circ_1 \circ_0^{k-1}(\mathsf{tick} \vee \mathsf{end}))$$

The predicate above expresses the condition that $\tau_1 = \ldots = \tau_k$ and $\tau_{i+k} = \tau_i + 1$.

$$\psi^2_{k,n} = \Box_{\approx_n 0} \Diamond_{n-1} \mathit{true}$$

The predicate $\psi^2_{k,n}$ in conjunction with $\psi^1_{k,n}$ expresses the condition that the length of the timed sequence is a multiple of kn.

$$\begin{aligned}\psi^3_{k,n} = \Diamond_{\approx_n 0}(\neg\mathsf{lastword} \ \wedge \ \mathsf{startcell} \ \wedge \\ \Box_{[0,n-1]}(\mathsf{startcell} \ \rightarrow \\ \wedge_{i=1}^{k}(\circ_0^{i-1}0 \rightarrow \Diamond_{\approx_n 0}(\mathsf{lastword} \ \wedge \ \mathsf{startcell} \ \wedge \ \circ_0^{i-1}0) \ \wedge \\ \circ_0^{i-1}1 \rightarrow \Diamond_{\approx_n 0}(\mathsf{lastword} \ \wedge \ \mathsf{startcell} \ \wedge \ \circ_0^{i-1}1))))\end{aligned}$$

The predicate $\psi^3_{k,n}$ in conjunction with $\psi^1_{k,n}$ and $\psi^2_{k,n}$ enforces the additional condition that $\rho[(l-1)kn+1, lkn] = \rho[(i-1)kn+1, ikn]$ for some $i < l$. Note the critical use of relative congruences in the above predicates instead of absolute congruences.

Theorem 5.2 *Let $\mathcal{A}$ be any monitoring algorithm for MTL.*

(i) *There is a formula ψ, to monitor which $\mathcal{A}$ requires space $\Omega(2^{\alpha c\sqrt{\frac{|\psi|}{}}})$, where c is the largest constant occurring in ψ, and α is a fixed constant.*

(ii) *There is a formula ψ, to monitor which $\mathcal{A}$ requires space $\Omega(2^{2^{\alpha|\psi|}})$ for a fixed constant α.* □

In the formula ψ of Theorem 5.2.1, let c be as in the statement. Then we have for $c > 1$

$$c\sqrt{|\underline{\psi}|} \ \geq\ \sqrt{(c+1)|\underline{\psi}|} \ \geq\ \sqrt{|\underline{\psi}| + \Sigma_\psi}$$

using the fact that $\Sigma_\psi \leq c|\underline{\psi}|$.

Finally, note that since $\phi_{k,n}$ contains only future time operators, the lower bounds established above also apply to MTL with only future time operators.

5.2 Lower Bounds for MTL with Intervals Only

We prove lower bounds for sublogics of MTL with no congruences (absolute or relative). We first prove a lower bound for MTL with only intervals of form $[0, \infty)$. Note that this will also give us a lower bound for LTL.

Consider a monitoring framework with only two atomic predicates and therefore only four possible states, say 0, 1, # and \$. For a natural number k, define $\mathcal{L}_k$ to be the set of all timed sequences (π, τ) such that

$$\pi \ \in\ \{\sigma\#w\#\sigma'\$w\sigma'' \mid w \in \{0,1\}^k \text{ and } \sigma, \sigma', \sigma'' \in \{0,1,\#\}^*\}$$

A similar language was previously used in several works [16,17,20] to prove lower bounds in model checking and in monitoring extended regular expressions.

Lemma 5.3 *Any monitoring algorithm for $\mathcal{L}_k$ requires space $\Omega(2^k)$.* □

Theorem 5.4 *Let $\mathcal{A}$ be any monitoring algorithm for MTL with only intervals of form $[0, \infty)$. There is a formula ϕ, to monitor which $\mathcal{A}$ requires space $\Omega(2^{\alpha\sqrt{|\phi|}})$ for a fixed constant α.* □

This lower bound can be improved for the sublogic of MTL with arbitrary intervals. Using the arguments similar to that in proof of Lemma 5.1 we can easily show that any monitoring algorithm would require space $\Omega(n)$ to monitor the formula $p \leftrightarrow (\neg(\Diamond_n true) \vee (\Diamond_n q))$. Thus, in general any monitoring algorithm would require space $\Omega(2^{\alpha\phi})$ to monitor a formula ϕ. The above happened because ϕ contains a constant that is exponentially larger than $|\phi|$. The following shows that even if the largest constant occurring in a formula is much smaller than the size of the formula, any monitor would still need an exponential space.

Theorem 5.5 *Suppose $\mathcal{A}$ is a monitoring algorithm for MTL with arbitrary intervals. There is a formula ϕ such that the largest constant occurring in it is smaller than $|\underline{\phi}|$ and $\mathcal{A}$ requires space $\Omega(2^{\alpha|\phi|/\log|\phi|})$, for a fixed constant α.* □

6 Conclusion and Future Work

A general monitoring algorithm for requirements expressed in metric temporal logic (MTL) has been presented, together with instantiations for various sublogics of MTL. It was shown that the algorithm is exponential in the number of temporal operators and atomic predicates, and in the sum of numeric constants in the original MTL formula, and also that the exponential bound cannot be avoided even for simple sublogics of MTL. The number of propositional operators, which often take most of the size of a specification, does not affect the complexity of our algorithms. Since MTL is an expressive and powerful logic for monitoring requirements, the presented novel and close to optimal algorithms can be used in practical runtime verification and testing tools, such as JPaX [10].

7 Acknowledgments

We are thankful to Koushik Sen for stimulating us in doing a better space analysis of the presented technique, thus improving the space requirement of our monitoring algorithm from our original rough $2^{O(m)}$ to the current $O(m2^m)$.

References

[1] R. Alur and T. Henzinger. Real time logics: complexity and expressiveness. In *Fifth annual symposium on logic in computer science*, pages 390–401. IEEE Computer Society Press, 1990.

[2] R. Alur and T. Henzinger. Logics and models of real time: A survey. In *Real Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*. Springer Verlag, 1992.

[3] H. Barringer, A. Goldberg, K. Havelund, and K. Sen. Rule-Based Runtime Verification. Pre-Print CSPP-24, University of Manchester, Department of Computer Science, August 2003.

[4] H. Barringer, A. Goldberg, K. Havelund, and K. Sen. Rule-based runtime verification. In *Proceedings of 5th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'04)*, Lecture Notes in Computer Science, 2004.

[5] R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691.

[6] Doron Drusinsky. The Temporal Rover and the ATG Rover. In *SPIN Model Checking and Software Verification*, volume 1885 of *Lecture Notes in Computer Science*, pages 323–330. Springer, 2000.

[7] B. Finkbeiner and H. Sipma. Checking finite traces using alternating automata. *Electronic Notes in Theoretical Computer Science*, 55(2), 2001.

[8] M. Geilen. On the construction of monitors for temporal logic properties. *Electronic Notes in Theoretical Computer Science*, 55(2), 2001.

[9] M.C.W. Geilen. An improved on-the-fly tableau construction for a real-time temporal logic. In *International Conference on Computer Aided Verification*, July 2003.

[10] K. Havelund and G. Roşu. Monitoring Java programs with Java PathExplorer. *Electronic Notes in Theoretical Computer Science*, 55(2), 2001.

[11] K. Havelund and G. Roşu. Monitoring programs using rewriting. In *Automated Software Engineering*. Institute of Electrical and Electronics Engineers Computer Society, 2001.

[12] K. Havelund and G. Roşu. Synthesizing monitors for safety properties. In *Tools and Algorithms for Construction and Analysis of Systems*, Lecture Notes in Computer Science 2280, pages 342–356, 2002.

[13] Klaus Havelund and Grigore Roşu. *Runtime Verification 2001*, volume 55 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science, 2001. Proceedings of a *Computer Aided Verification (CAV'01)* satellite workshop.

[14] Jieh Hsiang. Refutational Theorem Proving using Term Rewriting Systems. *Artificial Intelligence*, 25:255–300, 1985.

[15] R. Koymans. Specifying real-time properties with metric temporal logic. *Real Time Systems*, 2(4):255–299, 1990.

[16] O. Kupferman and M. Y. Vardi. Freedom, Weakness, and Determinism: From linear-time to branching-time. In *Proceedings of the IEEE Symposium on Logic in Computer Science*, pages 81–92, 1998.

[17] O. Kupferman and M. Y. Vardi. Model Checking of Safety Properties. In *Proceedings of the Conference on Computer-Aided Verification*, Lecture Notes in Computer Science, 1999.

[18] Zohar Manna and Amir Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer, New York, 1995.

[19] N. Markey and Ph. Schnoebelen. Model checking a path (preliminary report). In *14th Int. Conf. Concurrency Theory*, Lecture Notes in Computer Science 2761, pages 251–265. Springer, 2003.

[20] G. Roşu and M. Viswanathan. Testing extended regular language membership incrementally by rewriting. In *Rewriting Techniques and Applications*, Lecture Notes in Computer Science 2706, 2003.

[21] P. Wolper. *Synthesis of Communicating Processes from Temporal Logic Specifications*. PhD thesis, Stanford University, Dpeartment of Computer Science, 1982.

A Appendix

Proof of Lemma 3.2: We start with a few definitions. Define $\phi_1 \preceq \phi_2$ if

- $\mathcal{F}(\underline{\phi_1}) \subseteq \mathcal{F}(\underline{\phi_2})$, and
- $\mathcal{F}(\underline{\phi_1}) = \mathcal{F}(\underline{\phi_2})$ implies $|\phi_1| \leq |\phi_2|$.

Define $\phi_1 \simeq \phi_2$ if $\phi_1 \preceq \phi_2$ and $\phi_2 \preceq \phi_1$, and $\phi_1 \prec \phi_2$ if $\phi_1 \preceq \phi_2$ and $\phi_1 \not\simeq \phi_2$. Note that the relation $\prec$ is well-founded.

We prove all the six statements simultaneously by nested induction. The outer induction is on i, while the inner one is a Noetherian induction on the relation $\prec$ over formulas. For the outer induction, the base case is when $i = 1$. The arguments for the nested induction within this base case are similar to (and simpler than) those that arise within the induction step, and so we leave them to the reader. For the induction step, assume that all the six statements are true whenever $i < k$ (the outer induction hypothesis). We have to show that they are true for $i = k$. We now do a nested induction on $\prec$. The base case for this is when $\phi = \mathit{true}$ or $\phi = \mathit{false}$, and these are easy to check. For the induction step, we may assume that the six statements are true for all $i \leq k$ and $\psi \prec \phi$ (the inner induction hypothesis). We now consider each statement in turn.

1. We consider the case $\phi = \phi_1 \mathcal{S}_I \phi_2$ and $0 \in I$ and $i > 1$; the others are simpler.

$$[\rho, k]\phi_1 \mathcal{S}_I \phi_2 = [\rho, k](\phi_2 \vee (\phi_1 \wedge (\phi_1 \mathcal{S}_{I'} \phi_2)\{\rho, k-1\}))$$

where $I' = I - \tau_k + \tau_{k-1}$. Let $\psi = \phi_2 \vee (\phi_1 \wedge (\phi_1 \mathcal{S}_{I'} \phi_2)\{\rho, k-1\})$. Then $\mathcal{F}([\rho, k]\phi) = \mathcal{F}([\rho, k]\psi)$. From the outer induction hypothesis for statement 4, we have $\mathcal{F}(\phi_1 \mathcal{S}_{I'} \phi_2\{\rho, k-1\}) \subseteq \mathcal{F}^+(\phi_1 \mathcal{S}_{I'} \phi_2) \setminus \phi_1 \mathcal{S}_{I'} \phi_2 = \mathcal{F}^+(\phi) \setminus \phi$. Then it follows that $\mathcal{F}(\underline{\psi}) \subset \mathcal{F}(\underline{\phi})$, and hence $\psi \prec \phi$. Then by inner induction hypothesis for statement 1, we have $\mathcal{F}([\rho, k]\psi) \subseteq \mathcal{F}^+(\psi) = \mathcal{F}^+(\phi_1) \cup \mathcal{F}^+(\phi_2) \cup \mathcal{F}^+(\phi_1 \mathcal{S}_{I'} \phi_2\{\rho, i\}) = \mathcal{F}^+(\phi) \setminus \phi$, and the desired result follows.

2. The argument is similar to 1.

3. We again only consider the case $\phi = \phi_1 \mathcal{S}_I \phi_2$ and $0 \in I$ and $i > 1$; the others are simpler. Let ψ and I' be as in 1. Then

$$\begin{array}{lll}
(\rho, k) \models \phi_1 \mathcal{S}_I \phi_2 & \text{iff} & (\rho, k) \models \phi_2, \text{ or } (\rho, k) \models \phi_1 \text{ and } (\rho, k-1) \models \phi_1 \mathcal{S}_{I'} \phi_2 \\
 & \text{iff} & (\rho, k) \models \phi_2, \text{ or } (\rho, k) \models \phi_1 \text{ and } (\rho, k) \models (\phi_1 \mathcal{S}_{I'} \phi_2)\{\rho, k-1\}) \\
 & & \text{(using outer induction hypothesis for statement 6)} \\
 & \text{iff} & (\rho, k) \models \phi_2 \vee (\phi_1 \wedge (\phi_1 \mathcal{S}_{I'} \phi_2)\{\rho, k-1\}) \\
 & \text{i.e.} & (\rho, k) \models \psi
\end{array}$$

From the argument in 1, $\psi \prec \phi$. Then by inner induction hypothesis for statement 3 we have $(\rho, k) \models \psi$ iff $(\rho, k) \models [\rho, k]\psi$, which gives us the desired result.

4. We consider only $k < |\rho|$, of which we again consider only two subcases.

- $\phi = \phi_1 \mathcal{U}_I \phi_2$: Let $0 \in I$. Then

$$\phi_1 \mathcal{U}_I \phi_2\{\rho, k\} = \phi_2\{\rho, k\} \vee (\phi_1\{\rho, k\} \wedge (\phi_1 \mathcal{U}_{I'} \phi_2))$$

  where $I' = I - \tau_{i+1} + \tau_i$. Let $\psi = \phi_2\{\rho, k\} \vee (\phi_1\{\rho, k\} \wedge (\phi_1 \mathcal{U}_{I'} \phi_2))$. Then $\mathcal{F}(\phi\{\rho, k\}) = \mathcal{F}(\psi)$. Clearly, for $i = 1, 2$ we have $\mathcal{F}(\underline{\phi_i}) \subset \underline{\mathcal{F}(\phi)}$ and hence $\phi_i \prec \phi$. Then by the inner induction hypothesis for statement 4, we have $\mathcal{F}(\phi_i\{\rho, k\}) \subseteq \mathcal{F}^+(\phi_i) \subseteq \mathcal{F}^+(\phi)$. Also, $\mathcal{F}^+(\phi_1 \mathcal{U}_{I'} \phi_2) \subseteq \mathcal{F}^+(\phi_1 \mathcal{U}_I \phi_2)$. Then $\mathcal{F}(\psi) \subseteq \mathcal{F}^+(\phi)$, and the desired result follows.

- $\phi = \odot_I \phi_1$: Let $\eta = [\rho, k]\phi$. We have $\mathcal{F}(\phi\{\rho, k\}) = \mathcal{F}(\eta\{\rho, k\})$. From 1 we have $\mathcal{F}(\eta) \subseteq \mathcal{F}^+(\phi) \setminus \phi$. Then $\mathcal{F}(\underline{\eta}) \subset \mathcal{F}(\underline{\phi})$ and hence $\eta \prec \phi$. Then by inner induction hypothesis for statement 4 we have $\mathcal{F}(\eta\{\rho, k\}) \subseteq \mathcal{F}^+(\eta) \subset \mathcal{F}^+(\phi) \setminus \phi$, from which the result follows.

5. The argument is similar to 4.

6. We consider only $k < |\rho|$, of which we again consider only two subcases.

- $\phi = \phi_1 \mathcal{U}_I \phi_2$ and $0 \in I$: Let $I' = I - \tau_{k+1} + \tau_k$. Then

  $(\rho, k) \vDash \phi \quad$ iff $\quad (\rho, k) \vDash \phi_2$, or $(\rho, k) \vDash \phi_1$ and $(\rho, k+1) \vDash \phi_1 \mathcal{U}_{I'} \phi_2$

  Now, clearly for $i = 1, 2$ we have $\mathcal{F}(\underline{\phi_i}) \subset \mathcal{F}(\underline{\phi})$ and hence $\phi_i \prec \phi$. Then using the inner induction hypothesis for statement 6 we have

$$\begin{array}{lll} (\rho, k) \vDash \phi & \text{iff} & (\rho, k+1) \vDash \phi_2\{\rho, k\}, \text{ or } (\rho, k+1) \vDash \phi_1\{\rho, k\} \text{ and} \\ & & (\rho, k+1) \vDash \phi_1 \mathcal{U}_{I'} \phi_2 \\ & \text{iff} & (\rho, k+1) \vDash \phi_2\{\rho, k\} \vee (\phi_1\{\rho, k\} \wedge \phi_1 \mathcal{U}_{I'} \phi_2) \end{array}$$

- $\phi = \odot_I \phi_1$: Let η be as in 4. Then from 3 we have $(\rho, k) \vDash \phi$ iff $(\rho, k) \vDash \eta$. From the argument in 4 we have that $\eta \prec \phi$. Then by the inner hypothesis for statement 6, $(\rho, k) \vDash \eta$ iff $(\rho, k+1) \vDash \eta\{\rho, k\}$, and the result follows. □

Proof of Theorem 3.4: Since the pseudocode of Figures 2 and 1 closely follows Definition 3.1, it is clear that the monitoring procedure is correct. The memory required by the monitoring procedure above is the memory to store R and D, that is, $O(m2^m)$. With respect to time complexity, $\mathsf{monitor}(\phi, \rho)$ calls m times the procedures `resolve` and `derive` at steps 4 and 5, which take longer than calculating the substitution at step 6. Each of the procedures `resolve` and `derive` make one, two, or three calls to the substitution procedure, so each call to them takes $O(m^2 2^{3m})$. Thus, the time complexity of $\mathsf{monitor}(\phi, \rho)$ is $O(|\rho| m^3 2^{3m})$. □

Proof of Theorem 4.1: We have $\texttt{index}(\Box\phi_0) = m$. By Lemma 3.2, we have that $R[i]$ is either *true* or *false* for $i < m$ and $R[m] = \Box\phi_0$. Similarly, $D[i]$ is *true* or *false* for $i < m$ and $D[m]$ is either *false* or $\Box\phi_0$. The monitoring procedure can be easily modified to not have entries $R[m]$ and $D[m]$. One can check that the procedure thus modified takes time $O(|\rho|m)$ and space $O(m)$.$\Box$

Proof of Lemma 5.1: The proof is by contradiction. Define an equivalence relation $\equiv$ on timed sequences whose length is a multiple of kn, as follows:

$$\rho_1 \equiv \rho_2 \;\text{ if }\; \{\rho_1[(i{-}1)kn{+}1, ikn] \mid ikn \leq |\rho_1|\} = \{\rho_2[(j{-}1)kn{+}1, jkn] \mid jkn \leq |\rho_2|\}$$

Note that there are $2^{2^{kn}}$ equivalence classes. Suppose there is a monitoring algorithm $\mathcal{A}$ that uses space less than 2^{kn}. Then by the pigeon hole principle there are two timed sequences $\rho_1 \not\equiv \rho_2$ s.t. the memory of $\mathcal{A}$ is the same after reading ρ_1 and ρ_2. Since $\rho_1 \not\equiv \rho_2$ there is a w s.t. $w \in \{\rho_1[(i-1)kn+1, ikn]\}$ and $w \notin \{\rho_2[(j-1)kn+1, jkn]\}$. But $\mathcal{A}$ gives the same answer on $\rho_1.w$ and $\rho_2.w$. $\Box$

Proof of Theorem 5.2:

(i) Take $\psi = \phi_{k,n}$. We have $|\underline{\phi_{k,n}}| = \Theta(k^2)$, and the largest constant in $\phi_{k,n}$ is n. By Lemma 5.1, $\mathcal{A}$ requires $\Omega(2^{kn})$ space, and the result follows.

(ii) Take $\psi = \phi_{1,n}$. We have $|\phi_{1,n}| = \Theta(\log n)$. By Lemma 5.1, $\mathcal{A}$ requires $\Omega(2^n)$ space, and the result follows. $\Box$

Proof of Lemma 5.3: Similar to proof of Lemma 5.1. $\Box$

Proof of Theorem 5.4: The following formula defines $\mathcal{L}_k$

$$\begin{aligned}\phi_k = {} & [(\neg\$)\ \mathcal{U}\ (\$ \ \wedge\ \circ\Box(\neg\$))] \ \wedge \\ & \quad \Diamond[\# \ \wedge\ \circ^{k+1}\# \ \wedge\ \textstyle\bigwedge_{i=1}^{k}((\circ^i 0 \ \wedge\ \Box(\$ \rightarrow \circ^i 0)) \ \vee \\ & \qquad\qquad\qquad\qquad (\circ^i 1 \ \wedge\ \Box(\$ \rightarrow \circ^i 1)))].\end{aligned}$$

Note that $|\phi_k| = \Theta(k^2)$ and hence $k = \sqrt{|\phi_k|}$. It now follows by Lemma 5.3.$\Box$

Proof of Theorem 5.5: Consider the language which contains exactly those $(\pi, \tau) \in \mathcal{L}_k$ (defined above) such that $\tau_{i+1} = \tau_i + 1$. Clearly, any monitoring algorithm for $\mathcal{L}_k$ would require $\Omega(2^k)$ space. Now, the following formula defines this language:

$$\begin{aligned}\phi_k = {} & [(\neg\$)\ \mathcal{U}\ (\$ \ \wedge\ \circ\Box(\neg\$))] \ \wedge \\ & \quad \Diamond[\# \ \wedge\ \Diamond_{k+1}\# \ \wedge\ \textstyle\bigwedge_{i=1}^{k}((\Diamond_i 0 \ \wedge\ \Box(\$ \rightarrow \Diamond_i 0)) \ \vee \\ & \qquad\qquad\qquad\qquad (\Diamond_i 1 \ \wedge\ \Box(\$ \rightarrow \Diamond_i 1)))].\end{aligned}$$

The size of this formula is $\Theta(k \log k)$, and the largest constant occurring in it is $k < |\underline{\phi_k}|$. One can show that $k = \Theta(|\phi| / \log|\phi|)$, and the result follows. $\Box$