Monitoring Metric First-order Temporal Properties

DAVID BASIN, FELIX KLAEDTKE, SAMUEL MÜLLER, and
EUGEN ZĂLINESCU, ETH Zurich

Runtime monitoring is a general approach to verifying system properties at runtime by comparing system events against a specification formalizing which event sequences are allowed. We present a runtime monitoring algorithm for a safety fragment of metric first-order temporal logic that overcomes the limitations of prior monitoring algorithms with respect to the expressiveness of their property specification languages. Our approach, based on automatic structures, allows the unrestricted use of negation, universal and existential quantification over infinite domains, and the arbitrary nesting of both past and bounded future operators. Furthermore, we show how to use and optimize our approach for the common case where structures consist of only finite relations, over possibly infinite domains. We also report on case studies from the domain of security and compliance in which we empirically evaluate the presented runtime monitoring algorithm. Taken together, our results show that metric first-order temporal logic can serve as an effective specification language for expressing and monitoring a wide variety of practically relevant system properties.

Categories and Subject Descriptors: D.2.1 [Software Engineering]: Requirements/Specifications—languages; D.2.4 [Software Engineering]: Software/Program Verification—validation, formal methods; D.2.5 [Software Engineering]: Testing and Debugging—monitors, tracing; D.4.6 [Operating Systems]: Security and Protection; F.3.1 [Mathematical Logic and Formal Languages]: Mathematical Logic—temporal logic; J.1 [Computer Applications]: Administrative Data Processing—business, law

General Terms: Security, Verification

Additional Key Words and Phrases: Runtime Verification, Security Policies

1. INTRODUCTION

Runtime monitoring is an approach to verifying system properties at execution time. The system's behavior is abstracted to a trace consisting of a sequence of states or events at some level of abstraction and an online algorithm is used to check whether the trace satisfies a given property. Runtime monitoring has numerous applications such as monitoring properties of safety-critical programs or auditing system logs to verify compliance with security policies.

The properties that are verified by runtime monitors are typically requirements on the occurrences and ordering of system actions, possibly with quantitative timing restrictions. For example, every request must, within some given time bound, eventually be followed by an acknowledgement. Such requirements are naturally expressed in temporal logics and algorithms have been developed for monitoring system behavior with respect to properties specified in different temporal logics. See for example [Barringer et al. 2004; Barringer et al. 2010; Bauer et al. 2011; Chomicki 1995; Roger and Goubault-Larrecq 2001; Roșu and Havelund 2005; Sistla and Wolfson 1995]. Algorithmically, monitors are realized from specifications as some kind of automaton, which reads events, updates state information, and reports violations upon their detection.

There is the standard trade-off between the expressiveness of a property specification language and the complexity both of constructing monitors from specifications and of updating a monitor's state during runtime. Existing monitoring algorithms are often quite restrictive in the properties they can handle. Typically, either the temporal dimension or the data dimension of the property specification language is restricted in some way. For instance, only temporal operators that refer to the past are handled,
the range of data items is restricted to finite domains, or universal and existential quantification over data is not supported. Such restrictions limit the scope of runtime monitoring techniques. For example, in application areas like the automated compliance checking of IT systems and business processes with respect to security policies, monitoring techniques must account for a potentially unbounded number of agents and data items.

In this article, we present a runtime monitoring approach for an expressive safety fragment of metric first-order temporal logic (MFOTL) that overcomes most of the limitations of previously presented runtime monitoring approaches with respect to their expressive power. The fragment consists of formulas of the form $\Box \Phi$, where $\Phi$ is bounded, that is, its temporal operators refer only finitely into the future. The standard temporal operator $\Box$ ("generally") requires that $\Phi$ must hold at every time point. Temporal past and bounded future operators can be arbitrary nested in $\Phi$. There are also no restrictions on the quantification of variables, which range over an infinite domain, or on the use of negation in $\Phi$. We rely here on finite-state automata as data structures to represent and manipulate infinite but regular sets [Kesten et al. 2001].

In a nutshell, our monitoring algorithm works as follows. Given an MFOTL formula $\Box \Phi$ over a signature $S$, where $\Phi$ is bounded, we first transform $\Phi$ into a first-order formula $\hat{\Phi}$ over an extended signature $\hat{S}$, obtained by augmenting $S$ with auxiliary predicates for each temporal subformula in $\Phi$. The monitoring algorithm then incrementally processes a temporal structure $(\bar{D}, \bar{\tau})$ over $S$, which is a sequence $\bar{D}$ of automatic structures [Khoussainov and Nerode 1995; Blumensath and Grädel 2004], that is, first-order structures with regular relations and their associated time stamps $\bar{\tau}$. For each time point $i$, it determines those elements in $(\bar{D}, \bar{\tau})$ that violate $\Phi$. This is achieved by incrementally constructing a collection of automata that finitely represent the possibly infinite but regular interpretations of the auxiliary predicates at the time point $i$ and by evaluating the transformed first-order formula $\neg \hat{\Phi}$ over an extended structure over the signature $\hat{S}$ at $i$. In doing so, the monitoring algorithm discards information not required for evaluating $\neg \hat{\Phi}$ at the current and future time points.

This algorithm can be seen as an extension of Chomicki’s [1995] algorithm, developed for checking temporal integrity constraints of databases. The extensions are with respect to the monitorable fragment of MFOTL. The use of automatic structures allows the unrestricted use of negation and quantification. The presented monitoring algorithm also handles additional temporal operators, namely, bounded future operators, which can be used to formulate requirements that events do or must not occur within some finite time bound.

We also show how to adapt our monitoring algorithm to the common case where the relations that change over time are finite. In this case, finite tables, as in relational databases, provide an alternative to automata to store the interpretations of the predicates at each time point. However, to effectively evaluate the transformed first-order formula $\neg \hat{\Phi}$ at a time point, additional assumptions, which restrict the use of negation and quantification, are needed to guarantee the finiteness of the relations calculated during its evaluation. The restrictions are similar to those for database query evaluation, see [Abiteboul et al. 1995]. Furthermore, we show that under the additional, realistic restriction that time increases after at most a fixed number of time points, our incremental construction ensures that the monitoring algorithm requires only polynomial space in the cardinality of the data appearing in the processed prefix of the monitored temporal structure. Finally, we experimentally evaluate the monitoring algorithm for this case. For several security policies, we conduct experiments that measure the performance of our prototype implementation MonPoly [Basin et al. 2012]. Our results indicate that monitoring system behavior with respect to complex
properties formalized in MFOTL is feasible in practice. We also provide evidence that MFOTL is well suited for formalizing a large variety of security policies.

Overall, we see our contributions as follows. First, the presented monitoring algorithm handles a substantially more expressive temporal logic than previous algorithms. Second, for the restricted setting where relations are finite, we show how to efficiently implement the monitoring algorithm by using techniques from relational databases. We also provide upper bounds on the space consumed by our monitoring algorithm with respect to the cardinality of the data appearing in the processed prefix of a monitored temporal structure. Finally, our work shows how to effectively combine ideas from different, but related areas, including database theory, model checking, and model theory, and to apply them to relevant practical problems in runtime verification.

The remainder of this article is structured as follows. In Section 2, we define MFOTL and fix notation and terminology. In Section 3, we present our monitoring algorithm based on automatic structures and afterwards, in Section 4, we address the important case where relations are finite. In Section 5, we analyze and optimize the memory usage of our monitoring algorithm. In Section 6, we report on case studies. In Section 7, we discuss related work and, finally, in Section 8, we draw conclusions.

2. METRIC FIRST-ORDER TEMPORAL LOGIC

In this section, we introduce metric first-order temporal logic (MFOTL), which extends propositional metric temporal logic [Koymans 1990; Alur and Henzinger 1992] in a standard way. In the forthcoming sections, we present and evaluate methods for monitoring requirements formalized within MFOTL.

2.1. Syntax and Semantics

Let $I$ be the set of nonempty intervals over $\mathbb{N}$. We often write an interval in $I$ as $[b, b') := \{a \in \mathbb{N} \mid b \leq a < b'\}$, where $b, b' \in \mathbb{N}$ and $b < b'$. A signature $S$ is a tuple $(C, R, \iota)$, where $C$ is a finite set of constant symbols, $R$ is a finite set of predicates disjoint from $C$, and the function $\iota : R \rightarrow \mathbb{N}$ assigns each predicate $r \in R$ an arity $\iota(r)$. In the following, let $S = (C, R, \iota)$ be a signature and $V$ a countably infinite set of variables, assuming $V \cap (C \cup R) = \emptyset$.

**Definition 2.1.** The (MFOTL) formulas over the signature $S$ are inductively defined as follows:

(i) For $t, t' \in V \cup C$, $t \approx t'$ is a formula.

(ii) For $r \in R$ and $t_1, \ldots, t_{\iota(r)} \in V \cup C$, $r(t_1, \ldots, t_{\iota(r)})$ is a formula.

(iii) For $x \in V$, if $\phi$ and $\psi$ are formulas then $(\neg \phi), (\phi \lor \psi), \text{and } (\exists x. \phi)$ are formulas.

(iv) For $I \in I$, if $\phi$ and $\psi$ are formulas then $(\bullet_{I} \phi), (\circ_{I} \phi), (\phi S_{I} \psi)$, and $(\phi U_{I} \psi)$ are formulas, where $\bullet_{I}, \circ_{I}, S_{I}$, and $U_{I}$ are temporal operators, called previous, next, since, and until, respectively.

To define the semantics of MFOTL, we need the following notions. A **structure** $\mathcal{D}$ over the signature $S$ consists of a domain $|\mathcal{D}| \neq \emptyset$ and interpretations $c^{\mathcal{D}} \in |\mathcal{D}|$ and $r^{\mathcal{D}} \subseteq |\mathcal{D}|^{\iota(r)}$, for each $c \in C$ and $r \in R$. A temporal structure over the signature $S$ is a pair $(\mathcal{D}, \vec{\tau})$, where $\mathcal{D} = (D_0, D_1, \ldots)$ is a sequence of structures over $S$ and $\vec{\tau} = (\tau_0, \tau_1, \ldots)$ is a sequence of non-negative integers, with the following properties:

1. The sequence $\vec{\tau}$ is monotonically increasing, that is, $\tau_i \leq \tau_{i+1}$, for all $i \geq 0$. Moreover, $\vec{\tau}$ makes progress: for every $\tau \in \mathbb{N}$, there is some index $i \geq 0$ such that $\tau_i > \tau$.
2. $\mathcal{D}$ has constant domains, that is, $|D_i| = |D_{i+1}|$, for all $i \geq 0$. We denote the domain by $|\mathcal{D}|$. 

3. Each constant symbol $c \in C$ has a rigid interpretation, that is, $c^{\mathcal{D}_i} = c^{\mathcal{D}_{i+1}}$, for all $i \geq 0$. We denote $c$'s interpretation by $c^\mathcal{D}$.

We also call the elements in the sequence $\bar{\tau}$ time stamps and the indices of the elements in the sequences $\mathcal{D}$ and $\bar{\tau}$ time points.

A valuation is a mapping $v : V \to |\mathcal{D}|$. For a valuation $v$, the variable vector $\bar{x} = (x_1, \ldots, x_n)$, and $\bar{d} = (d_1, \ldots, d_n) \in |\mathcal{D}|^n$, we write $v[\bar{x} \to \bar{d}]$ for the valuation that maps $x_i$ to $d_i$, for $1 \leq i \leq n$, and the other variables' valuation is unaltered. We abuse notation by applying a valuation $v$ also to constant symbols $c \in C$, with $v(c) = c^\mathcal{D}$.

**Definition 2.2.** Let $(\mathcal{D}, \bar{\tau})$ be a temporal structure over the signature $S$, with $\mathcal{D} = (\mathcal{D}_0, \mathcal{D}_1, \ldots)$ and $\bar{\tau} = (\tau_0, \tau_1, \ldots)$, $\phi$ a formula over $S$, $v$ a valuation, and $i \in \mathbb{N}$. We define the relation $(\mathcal{D}, \bar{\tau}, v, i) \models \phi$ inductively as follows:

\[
\begin{align*}
(\mathcal{D}, \bar{\tau}, v, i) \models t \approx t' & \quad \text{iff} \quad v(t) = v(t') \\
(\mathcal{D}, \bar{\tau}, v, i) \models r(t_1, \ldots, t_{i(r)}) & \quad \text{iff} \quad (v(t_1), \ldots, v(t_{i(r)})) \in r^{\mathcal{D}_i} \\
(\mathcal{D}, \bar{\tau}, v, i) \models (\neg \psi) & \quad \text{iff} \quad (\mathcal{D}, \bar{\tau}, v, i) \not\models \psi \\
(\mathcal{D}, \bar{\tau}, v, i) \models (\psi \lor \psi') & \quad \text{iff} \quad (\mathcal{D}, \bar{\tau}, v, i) \models \psi \lor (\mathcal{D}, \bar{\tau}, v, i) \models \psi' \\
(\mathcal{D}, \bar{\tau}, v, i) \models (\exists \psi) & \quad \text{iff} \quad (\mathcal{D}, \bar{\tau}, v, i) \models \psi, \text{ for some } d \in |\mathcal{D}| \\
(\mathcal{D}, \bar{\tau}, v, i) \models (\psi_1 \land \psi_2) & \quad \text{iff} \quad i > 0, \tau_i - \tau_{i-1} \in I, \text{ and } (\mathcal{D}, \bar{\tau}, v, i - 1) \models \psi \\
(\mathcal{D}, \bar{\tau}, v, i) \models (\psi_1 \approx \psi_2) & \quad \text{iff} \quad \tau_{i+1} - \tau_i = I \text{ and } (\mathcal{D}, \bar{\tau}, v, i + 1) \models \psi \\
(\mathcal{D}, \bar{\tau}, v, i) \models (\psi S_I \psi') & \quad \text{iff} \quad \text{for some } j \leq i, \tau_i - \tau_j \in I, (\mathcal{D}, \bar{\tau}, v, j) \models \psi', \\
& \quad \text{and } (\mathcal{D}, \bar{\tau}, v, k) \models \psi, \text{ for all } k \in \mathbb{N} \text{ with } j < k \leq i \\
(\mathcal{D}, \bar{\tau}, v, i) \models (\psi U_I \psi') & \quad \text{iff} \quad \text{for some } j \geq i, \tau_j - \tau_i \in I, (\mathcal{D}, \bar{\tau}, v, j) \models \psi', \\
& \quad \text{and } (\mathcal{D}, \bar{\tau}, v, k) \models \psi, \text{ for all } k \in \mathbb{N} \text{ with } i \leq k < j 
\end{align*}
\]

Note that the temporal operators are augmented with intervals and a formula of the form $(\psi S_I \psi)$, $(\psi U_I \psi')$, or $(\psi S_I \psi')$ is only satisfied in $(\mathcal{D}, \bar{\tau})$ at the time point $i$ if it is satisfied within the bounds given by the interval $I$ of the respective temporal operator, which are relative to the current time stamp $\tau_i$.

### 2.2. Terminology and Notation

We introduce notation and terminology, which is mostly standard. As syntactic sugar, we use standard Boolean connectives like $(\phi \land \psi) := (\neg (\neg \phi) \lor (\neg \psi))$ and $(\phi \to \psi) := (\neg \phi) \lor \psi$, the universal quantifier $(\forall x. \phi) := (\neg (\exists x. \neg \phi))$, and the temporal operators $(\psi S_I \phi) := (\text{true} S_I \phi), (\psi U_I \phi) := (\neg (\psi S_I \neg \phi)), (\psi F_I \phi) := (\text{true} U_I \phi)$, and $(\psi \Box_I \phi) := (\neg (\psi F_I \neg \phi))$, where $I \in \mathbb{I}$ and true abbreviates $c \equiv c$, for some constant symbol $c \in C$, assuming without loss of generality that $C$ is nonempty. Non-metric variants of the temporal operators are easily defined, for example, $(\psi \Box I \phi) := (\psi [0, \infty) \phi)$.

We use standard conventions concerning operators’ binding strength to omit parentheses. For example, $\neg$ binds stronger than $\land$, which binds stronger than $\lor$, which in turn binds stronger than $\exists$. Moreover, Boolean operators bind stronger than temporal ones.

We denote the set of free variables in a formula $\phi$ by $\text{free}(\phi)$. To fix the ordering of the free variables in $\phi$, we also say that $\phi$ has the vector of free variables $\bar{x} = (x_1, \ldots, x_n)$, where $\text{free}(\phi) = \{x_1, \ldots, x_n\}$. We call formulas of the form $t \approx t'$ and $r(t_1, \ldots, t_{i(r)})$ atomic, and formulas with no temporal operators first-order. A formula $\phi$ is bounded if the interval $I$ of every temporal operator $U_I$ occurring in $\phi$ is finite. The main connective of a non-atomic formula is the operator (that is, Boolean operator, quantifier, or temporal operator) at the root of the formula’s syntax tree. A formula that has a temporal operator as its main connective is a temporal formula. For a formula $\phi$, we
define the set of \( \phi \)'s top-level temporal subformulas as

\[
\text{tsub}(\phi) := \begin{cases} 
\{ \psi \} & \text{if } \phi = \neg \psi \text{ or } \phi = \exists x. \psi, \\
\text{tsub}(\psi) \cup \text{tsub}(\psi') & \text{if } \phi = \psi \lor \psi', \\
\{ \phi \} & \text{if } \phi \text{ is a temporal formula,} \\
\emptyset & \text{otherwise.} 
\end{cases}
\]

For example, for \( \phi := (\bullet \alpha) \lor (\circ \beta) \ S_{[1,9]} \gamma \), we have \( \text{tsub}(\phi) = \{ (\bullet \alpha), (\circ \beta) S_{[1,9]} \gamma \} \). The set of direct subformulas of \( \phi \) is defined as

\[
\text{dsub}(\phi) := \begin{cases} 
\{ \psi \} & \text{if } \phi = \neg \psi, \ \phi = \exists x. \psi, \ \phi = \bullet \psi, \text{ or } \phi = \circ I \psi, \\
\{ \psi, \psi' \} & \text{if } \phi = \psi \lor \psi', \ \phi = \psi S I \psi', \ \text{or } \phi = \psi U I \psi', \\
\emptyset & \text{otherwise.} 
\end{cases}
\]

For a formula \( \phi \) with the vector of free variables \( \bar{x} = (x_1, \ldots, x_n) \), we define the set of satisfying elements at time point \( i \in \mathbb{N} \) in the temporal structure \( (\bar{D}, \bar{r}) \) as

\[
\phi(\bar{D}, \bar{r}, i) := \{ \bar{d} \in |\bar{D}|^n \mid (\bar{D}, \bar{r}, v[\bar{x} \mapsto \bar{d}], i) \models \phi, \text{ for some valuation } v \}.
\]

If \( \phi \) is first-order, then \( \phi(\bar{D}, \bar{r}, i) \) only depends on the structure \( \bar{D}_i \) and we just write \( \phi^{\bar{D}_i} \) in this case. Similarly, we just write \( (\bar{D}_i, v) \models \phi \), for first-order formulas \( \phi \), since \( (\bar{D}_i, \bar{r}, v, i) \models \phi \) only depends on the structure \( \bar{D}_i \) and the valuation \( v \).

### 2.3. Examples

Before presenting our monitoring method, we give several examples of using MFOTL for formalizing system requirements.

**Example 2.3.** Consider an approval policy for publishing business reports within a company, namely, any report must be approved prior to its publication. For the ease of exposition, we restrict ourselves here to this very simple policy. In Section 6, we consider more realistic security policies and their formalization in MFOTL.

We assume that the events for publishing and approving reports are logged in relations, which are, for instance, easily obtained from a log stream that records publish and approval events in an IT system. Specifically, for each time point \( i \in \mathbb{N} \), we have the unary relations \( PUBLISH_i \) and \( APPROVE_i \) such that (i) \( f \in PUBLISH_i \) iff report \( f \) is published at time \( i \) and (ii) \( f \in APPROVE_i \) iff report \( f \) is approved at time \( i \). Observe that there can be multiple approvals at the same time point for different reports. Furthermore, every time point \( i \) has a time stamp \( \tau_i \in \mathbb{N} \).

The corresponding temporal structure \( (\bar{D}, \bar{r}) \) with \( \bar{D} = (\bar{D}_0, \bar{D}_1, \ldots) \), a sequence of logged publishing and approval events, and \( \bar{r} = (\tau_0, \tau_1, \ldots) \), a sequence of time stamps, is as follows. The predicates in \( \bar{D}'s \) signature are \( publish \) and \( approve \), both of arity 1. The domain of \( \bar{D} \) consists of all possible report names. For example, if a report can be uniquely identified by a non-negative number, then we can assume that \( |\bar{D}| \) equals \( \mathbb{N} \). The \( i \)th structure in \( \bar{D} \) is time-stamped with \( \tau_i \) and contains the relations \( PUBLISH_i \) and \( APPROVE_i \).

We express the policy by the MFOTL formula

\[
\Box \forall f. \ publish(f) \to \bullet \ approve(f).
\]

The following formula formalizes an additional constraint. Namely, an approval is only valid for at most 10 time units:

\[
\Box \forall f. \ publish(f) \to \bullet_{[0,11]} \ approve(f).
\]
Note that in this last formula we speak of time units when measuring the time difference \( \tau_j - \tau_i \) between the time stamps \( \tau_i \) and \( \tau_j \) of two time points \( i \) and \( j \), with \( i \leq j \). The interpretation of a time unit within a system depends on the granularity with which time is tracked. For instance, if the system-time stamps each time point with the current date, that is, year, month, and day, then the smallest possible time unit is a day. If time stamps additionally contain the time of the day, then we could choose hours, minutes, or seconds as time units. In subsequent examples, the meaning of time units is either clear from the context or irrelevant.

**Example 2.4.** The following two examples are simple but typical properties arising in system verification.

The property “whenever the set variable in stores an element \( x \), then within 5 time units \( x \) must be contained in the set variable out” can be formalized by \( \Box \forall x. \text{in}(x) \rightarrow \diamond_{[0,6)} \text{out}(x) \). The property “the value of the integer variable \( v \) increases by 1 in each step from an initial value 0 until it becomes 5, and then it stays constant” can be formalized as \( \Box(\neg(\bullet \text{true}) \rightarrow v(0)) \land (\exists i. v(i) \land i < 5 \rightarrow \diamond v(i + 1)) \land (v(5) \rightarrow \diamond v(5)) \).

We assume that the relations for the predicate \( \alpha \) are singletons so that they model the values of an integer variable during the execution of a program, and that \( \prec \) is a binary predicate represented in infix notation and interpreted as expected.

3. Monitoring

To effectively monitor requirements given as MFOTL formulas, we restrict both the formulas and the temporal structures under consideration. We discuss these restrictions in Section 3.1 and describe monitoring in Sections 3.2–3.6.

3.1. Restrictions

Let \((A, \bar{r})\) be a temporal structure over the signature \( S = (C, R, \iota) \) and \( \Psi \) the formula expressing the property for monitoring. We make the following restrictions on \( \bar{A} \) and \( \Psi \).

First, we require \( \Psi \) to be of the form \( \Box \Phi \), where \( \Phi \) is bounded. It follows that \( \Psi \) describes a safety property [Alpern and Schneider 1985; Henzinger 1992]. Note, however, there are safety properties expressible in MFOTL that do not have this syntactic form [Chomicki and Niwiński 1995]. This is in contrast to propositional linear-time temporal logic, where every \( \omega \)-regular safety property can be expressed as a formula \( \Box \beta \), where \( \beta \) contains only past operators [Lichtenstein et al. 1985].

Second, we require that each structure in \( \bar{A} \) is automatic [Khoussainov and Nerode 1995]. Roughly speaking, this means that each structure can be finitely represented by a collection of finite-state automata over finite words. We briefly recall some background on automatic structures [Khoussainov and Nerode 1995; Blumensath and Grädel 2004], where we assume familiarity with basic automata theory.

Let \( \Sigma \) be an alphabet and \( \# \) a symbol not in \( \Sigma \). The convolution of the words \( w_1, \ldots, w_k \in \Sigma^* \), where each \( w_i = w_{i1} \cdots w_{i\ell_i} \), is the word

\[
w_1 \otimes \cdots \otimes w_k := \begin{bmatrix}
w'_1 \vdots \vdots
\end{bmatrix}
\cdots
\begin{bmatrix}\w'_k \vdots \vdots
\end{bmatrix}
\in \left( (\Sigma \cup \{\#\})^k \right)^*,
\]

where \( \ell = \max\{\ell_1, \ldots, \ell_k\} \) and \( w'_{ij} = w_{ij} \) for \( j \leq \ell_i \) and \( w'_{ij} = \# \) otherwise. The padding symbol \( \# \) is used to ensure that the words have the same length.

**Definition 3.1.** Let \( \bar{A} \) be a structure over the signature \( S = (C, R, \iota) \).

(i) The structure \( \bar{A} \) is **automatic** if there is a regular language \( \mathcal{L}_{|\bar{A}|} \subseteq \Sigma^* \) and a surjective function \( \nu : \mathcal{L}_{|\bar{A}|} \to |\bar{A}| \) such that the language \( \mathcal{L}_\nu := \{u \otimes v \mid u, v \in \mathcal{L}_{|\bar{A}|}\} \) is...
\( L_{\nu} \) with \( \nu(u) = \nu(v) \) is regular and, for each relation \( r^A \subseteq |A|^r \) with \( r \in R \), the language \( L_r := \{ w_1 \otimes \cdots \otimes w_{|r|} \mid w_1, \ldots, w_{|r|} \in L_{\nu} \} \) with \( \nu(w_1), \ldots, \nu(w_{|r|}) \in r^A \) is regular.

(ii) An automatic representation of the automatic structure \( A \) consists of (1) the function \( \nu : L_{|A|} \to |A| \), (2) a family of words \( (w_c)_{c \in C} \) with \( w_c \in L_{|A|} \) and \( \nu(w_c) = c^A \), for all \( c \in C \), and (3) automata \( A_{|A|} \), \( A_{\leq} \), and \( A_r \), for \( r \in R \), that recognize the languages \( L_{|A|} \), \( L_{\leq} \), and \( L_r \), for \( r \in R \), respectively. Note that the automata \( A_{|A|} \) and \( A_r \), for \( r \in R \), read the components of the convolution of a representative of an element \( a \in |A|^k \) synchronously. In the following, we assume that for an automatic structure, we always have an automatic representation for it at hand.

(iii) A relation \( r \subseteq |A|^k \) is regular if the language \( \{ u_1 \otimes \cdots \otimes u_k \mid u_1, \ldots, u_k \in L_{|A|} \} \) with \( \nu(u_1), \ldots, \nu(u_k) \in r \) is regular.

In addition to requiring that each structure is automatic, we also require that \( \mathcal{D} \) has a constant domain representation. This means that the domain of each \( D_i \) is represented by the same regular language \( L_{|D_i|} \) and each word in \( L_{|D_i|} \) represents the same element in \( |D_i| \). In other words, each automatic representation of the \( D_i \)'s has the same function \( \nu : L_{|D_i|} \to |D_i| \).

Finally, we assume that \( |D| = \mathbb{N} \) and that there is a binary predicate \( \prec \) in \( R \) that is interpreted as the standard ordering relation \( < \) on \( \mathbb{N} \). This assumption is without loss of generality whenever the function \( \nu \) is injective, that is, every element in \( |D| \) has only one representative in \( L_{|D|} \), see Lemma 3.2 below. Furthermore, note that every automatic structure has an automatic representation in which the function \( \nu \) is injective [Khoussainov and Nerode 1995].

**Lemma 3.2.** Let \( A \) be an automatic structure with an infinite domain that has an automatic representation in which each element is uniquely represented. There is an ordering \( \prec_\nu \) on \( |A| \) such that \( (|A|, \prec_\nu) \) is isomorphic to \( (\mathbb{N}, <) \).

**Proof.** Let \( A \) be an automatic structure represented by an injective function \( \nu : L_{|A|} \to |A| \) and the respective automata for the domain, the equality, and its relations. Without loss of generality, assume that the representation \( L_{|A|} \) of \( A \)'s domain is over the alphabet \( \Sigma \) which is linearly ordered by \( \prec_{\text{alph}} \). We lift \( \prec_{\text{alph}} \) to linearly order the elements in \( \Sigma^* \). For \( w, w' \in \Sigma^* \), we define \( w \prec_\nu w' \) iff \( |w| < |w'| \), or \( |w| = |w'| \) and \( w \prec_{\text{lex}} w' \), where \( |w| \) denotes the length of a word \( u \in \Sigma^* \) and \( \prec_{\text{lex}} \) is the lexicographical ordering on \( \Sigma^* \) with respect to the ordering \( \prec_{\text{alph}} \) on the alphabet \( \Sigma \).

It is easy to see that \( \prec_\nu \) can be recognized by an automaton by reading the letters of words \( w \) and \( w' \) synchronously. That is, the language \( L := \{ w \otimes w' \mid w \prec_\nu w' \} \) is regular. We can use \( \prec_\nu \) to order the elements in \( |A| \). For \( a, b \in |A| \), we define \( a \prec_\nu b \) iff \( \nu^{-1}(a) \prec_\nu \nu^{-1}(b) \), which is equivalent to \( \nu^{-1}(a) \otimes \nu^{-1}(b) \in L \). Obviously, the ordering \( \prec_\nu \) is regular and \( (|A|, \prec_\nu) \) is isomorphic to \( (\mathbb{N}, <) \). \( \Box \)

**Remark 3.3.** We state some properties of automatic structures that we need later. First, some basic arithmetic operations are first-order definable in the structure \( (\mathbb{N}, <) \) and thus regular. In particular, the successor relation \( \text{succ} := \{ (x, y) \in \mathbb{N}^2 \mid y = x + 1 \} \) is regular, since the formula \( x \prec y \land \neg \exists z \cdot x \prec z \land z \prec y \) defines it. It is also easy to see that the set \( \{ (x, y) \in \mathbb{N}^2 \mid x + d \leq y \} \) is regular, for any \( d \in \mathbb{N} \). Finally, for a first-order formula \( \phi \) and an automatic structure \( A \), we can effectively construct an automaton that represents the set \( \phi^A \). This follows from the closure properties of regular languages.
3.2. Overview of the Monitoring Algorithm

In the remainder of this section, let \((\hat{D}, \hat{r})\) be a temporal structure over the signature \(S = (C, R, \iota)\) and let \(\Box \Phi\) be an MFOTL formula with the restrictions from Section 3.1.

To monitor the formula \(\Box \Phi\) over a temporal structure \((\hat{D}, \hat{r})\), we incrementally build a sequence of structures \(\hat{D}_0, \hat{D}_1, \ldots\) over an extended signature \(\hat{S}\). The extension depends on the temporal subformulas of \(\Phi\). For each time point \(i\), we determine the elements that violate \(\Phi\) by evaluating a transformed, first-order formula \(\neg \hat{\Phi}\) over \(\hat{D}_i\). Observe that for a temporal subformula with a future operator as its main connective, we usually cannot yet carry out this evaluation at time point \(i\). The monitoring algorithm therefore maintains a queue of unevaluated formulas and evaluates them when enough time has elapsed.

In the following, we first describe in Section 3.3 how we extend \(S\) and transform \(\Phi\). Afterwards, we explain in Section 3.4 how we incrementally build the relations of the extended structures \(\hat{D}_i\). In Section 3.5, we give an example to illustrate the transformation and constructions of the relations in the \(\hat{D}_i\)'s. Finally, in Section 3.6, we present our monitoring algorithm and prove its correctness.

3.3. Signature Extension and Formula Transformation

In addition to the predicates in \(R\), the extended signature \(\hat{S}\) contains an auxiliary predicate \(p_\phi\) for each temporal subformula \(\phi\) of \(\Phi\). For subformulas of the form \(\beta S I_1 \gamma\) and \(\beta U I_1 \gamma\), we introduce additional auxiliary predicates, which store information that allows us to incrementally update the auxiliary relations. Specifically, let \(\hat{S} := (\hat{C}, \hat{R}, \hat{r})\) be the signature with \(\hat{C} := C\) and

\[
\hat{R} := R \cup \{p_\phi \mid \phi \text{ is a temporal subformula of } \Phi\} \cup \{r_\phi \mid \phi \text{ is a temporal subformula of } \Phi \text{ with main connective } S_I \text{ or } U_I\} \cup \{s_\phi \mid \phi \text{ is a temporal subformula of } \Phi \text{ with main connective } U_I\},
\]

where \(p_\phi, r_\phi, s_\phi \notin C \cup R \cup \nu\). The arities of the predicates in \(\hat{R}\) are as follows: For a predicate \(r \in R\), let \(\hat{i}(r) := i(r)\). If \(\phi\) is a temporal subformula of \(\Phi\) with \(n\) free variables, then \(\hat{i}(p_\phi) := n\), and \(\hat{i}(r_\phi) := \hat{i}(s_\phi) := n + 1\), if \(r_\phi\) and \(s_\phi\) exist.

We transform the MFOTL formula \(\hat{\Phi}\) over the signature \(\hat{S}\) into the first-order formula \(\phi\) over the extended signature \(\hat{S}\) as follows. For a subformula \(\phi\) of \(\Phi\), we define

\[
\hat{\phi} := \begin{cases} 
\phi & \text{if } \phi \text{ is an atomic formula,} \\
\neg \hat{\psi} & \text{if } \phi = \neg \psi, \\
\hat{\psi} \lor \hat{\psi}' & \text{if } \phi = \psi \lor \psi', \\
\exists y. \hat{\psi} & \text{if } \phi = \exists y. \psi, \\
p_\phi(\hat{x}) & \text{if } \phi \text{ is a temporal formula with the vector } \hat{x} = (x_1, \ldots, x_n) \text{ of free variables.}
\end{cases}
\]

This formula transformation has the following properties, which are easily shown by induction over the formula structure.

**Lemma 3.4.** Let \(\hat{D}_0, \hat{D}_1, \ldots\) be structures over the signature \(\hat{S}\) that extend the \(D_i\)'s, that is, \(|\hat{D}_i| = |D_i|\), \(c^{\hat{D}_i} = c^{D_i}\), and \(r^{\hat{D}_i} = r^{D_i}\), for all \(c \in C\) and \(r \in R\). For every subformula \(\phi\) of \(\Phi\) and for all \(i \in \mathbb{N}\), the following properties hold:

(i) If \(p^\hat{D}_i = p^D_i\) for all \(\psi \in \text{tsub}(\phi)\), then \(\hat{\phi}^{\hat{D}_i} = \phi^{D_i}\).

(ii) If \(p^\hat{D}_i\) is regular for all \(\psi \in \text{tsub}(\phi)\), then \(\hat{\phi}^{\hat{D}_i}\) is regular.
3.4. Incremental Extended Structure Construction

In this subsection, we show how to construct the extended structures \( \hat{F}_i \) incrementally, in particular, the relations for the auxiliary predicates. Their instantiations are computed recursively both over time and over the formula structure, where evaluations of subformulas may also be needed from future time points. We later show that this is well defined and can be evaluated incrementally.

For \( i \in \mathbb{N} \), \( c \in C \), and \( r \in R \), we define \( c^{\hat{F}_i} := c^{F_i} \) and \( r^{\hat{F}_i} := r^{F_i} \). We present the auxiliary relations for each type of temporal operator separately. Throughout this subsection, let \( i \in \mathbb{N} \) and let \( \alpha \) be a temporal subformula of \( \Phi \). Furthermore, for the ease of exposition and without loss of generality, we assume that the direct subformulas of \( \alpha \) have the vector \( \bar{x} = (x_1, \ldots, x_n) \) of free variables.

3.4.1. Previous Operator. If the formula \( \alpha \) is of the form \( \cdot_i \beta \) with \( I \in \mathbb{I} \), we define

\[
p^{\alpha}_{\hat{F}_i} := \begin{cases} \hat{\beta}^{\hat{F}_{i-1}} & \text{if } i > 0 \text{ and } \tau_i - \tau_{i-1} \in I, \\ \emptyset & \text{otherwise.} \end{cases}
\]

Intuitively, a tuple \( \bar{a} \) is in \( p^{\alpha}_{\hat{F}_i} \) if \( \bar{a} \) satisfies \( \beta \) at the previous time point \( i - 1 \) and the difference of the two successive time stamps is in the interval \( I \) given by the metric temporal operator \( \cdot_i \).

**Lemma 3.5.** Let \( \alpha = \cdot_i \beta \). The relation \( p^{\alpha}_{\hat{F}_0} \) is regular and \( p^{\alpha}_{\hat{F}_0} = \alpha^{(\hat{F}, \tau, 0)} = \emptyset \). For \( i > 0 \), if the relations \( p^{\alpha}_{\hat{F}_{i-1}} \) are regular and \( p^{\alpha}_{\hat{F}_{i-1}} = \phi^{(\hat{F}, \tau, i-1)} \) for all \( \phi \in \text{tsub}(\beta) \), then the relation \( p^{\alpha}_{\hat{F}_i} \) is regular and \( p^{\alpha}_{\hat{F}_i} = \alpha^{(\hat{F}, \tau, i)} \).

**Proof.** For \( i = 0 \), the lemma obviously holds. For \( i > 0 \), the regularity of \( p^{\alpha}_{\hat{F}_i} \) follows from the assumption that the relations \( p^{\alpha}_{\hat{F}_{i-1}} \) are regular and Lemma 3.4(ii). The equality of the two sets follows from Lemma 3.4(i) and the semantics of the temporal operator \( \cdot_i \). \( \square \)

3.4.2. Next Operator. If the formula \( \alpha \) is of the form \( \circ_i \beta \) with \( I \in \mathbb{I} \), we define

\[
p^{\alpha}_{\hat{F}_i} := \begin{cases} \hat{\beta}^{\hat{F}_{i+1}} & \text{if } \tau_{i+1} - \tau_i \in I, \\ \emptyset & \text{otherwise.} \end{cases}
\]

The following lemma is proved similarly to Lemma 3.5.

**Lemma 3.6.** Let \( \alpha = \circ_i \beta \). If the relations \( p^{\alpha}_{\hat{F}_{i+1}} \) are regular and \( p^{\alpha}_{\hat{F}_{i+1}} = \phi^{(\hat{F}, \tau, i+1)} \) for all \( \phi \in \text{tsub}(\beta) \), then the relation \( p^{\alpha}_{\hat{F}_i} \) is regular and \( p^{\alpha}_{\hat{F}_i} = \alpha^{(\hat{F}, \tau, i)} \).

3.4.3. Since Operator. Before we give the construction details for the metric since operator, we first consider its non-metric variant. Note that we could directly define the relation \( p^{\hat{F}_i}_{\beta \circ_\gamma} \) as

\[
\bigcup_{j \leq i} (\hat{\beta}^{\hat{F}_j} \cap \bigcap_{k < j \leq i} \hat{\beta}^{\hat{F}_{j-k}}).
\]

However, this construction has the drawback that at each time point \( i \), we recompute the unions of intersections for \( j \leq i \). Instead, we use the following construction, which reflects that \( \beta \circ \gamma \) is logically equivalent to \( \gamma \lor \beta \land (\beta \circ \gamma) \): For \( i \geq 0 \), we define

\[
p^{\alpha}_{\hat{F}_i} := \hat{\gamma}^{\hat{F}_i} \cup \begin{cases} \emptyset & \text{if } i = 0, \\ \hat{\beta}^{\hat{F}_i} \cap p^{\alpha}_{\hat{F}_{i-1}} & \text{if } i > 0. \end{cases}
\]
This construction is *incremental* in the sense that it only depends on the relations in $\mathcal{D}_i$, for which the corresponding predicates occur in the subformulas $\beta$ or $\hat{\gamma}$, and on the auxiliary relation $p^{\mathcal{D}_i}_{i;\beta,\gamma}$, when $i > 0$. In particular, it does not depend on relations in $\mathcal{D}_j$ for $j < i - 1$.

Now assume that the formula $\alpha$ is of the form $\beta S_I \gamma$ with $I = [b, b')$. To incorporate the timing constraint given by the interval $I$, we first incrementally construct the auxiliary relations for the predicate $r_\alpha$, similar to the above definition for the non-metric case. We define $r^{\mathcal{D}_i}_\alpha$ as the union of a set $N$ containing the new elements and a set $U$ containing the updated tuples. That is, $N$ contains the tuples that are obtained from data at the time point $i$ and $U$ contains the updated tuples from the time points $j$ with $j < i$ and $\tau_i - \tau_j < b'$. Formally, $r^{\mathcal{D}_i}_\alpha := N \cup U$, where $N := \hat{\gamma}^{\mathcal{D}_i} \times \{0\}$, $U := \emptyset$ if $i = 0$, and for $i > 0$,

$$U := \{ (\bar{a}, y) \mid \bar{a} \in \hat{\beta}^{\mathcal{D}_i}, \ y < b', \ \text{and} \ (\bar{a}, y') \in r^{\mathcal{D}_{i-1}}_\alpha \text{ with } y' = y - \tau_i + \tau_{i-1} \}.$$ 

Intuitively, a pair $(\bar{a}, y)$ is in $r^{\mathcal{D}_i}_\alpha$ if $\bar{a}$ satisfies $\alpha$ at the time point $i$ independent of the lower bound $b$, where the “age” $y$ indicates how long ago the formula $\alpha$ was satisfied by $\bar{a}$. If $\bar{a}$ satisfies $\gamma$ at the time point $i$, it is added to $r^{\mathcal{D}_i}_\alpha$ with the age $0$. For $i > 0$, we also update the tuples $(\bar{a}, y) \in r^{\mathcal{D}_{i-1}}_\alpha$ when $\bar{a}$ satisfies $\beta$ at time point $i$, that is, the age is adjusted by the difference of the time stamps $\tau_{i-1}$ and $\tau_i$ in case the new age is less than $b'$. Otherwise it is too old to satisfy $\alpha$ and the updated tuple is not included in $r^{\mathcal{D}_i}_\alpha$.

Finally, we obtain the auxiliary relation $p^{\mathcal{D}_i}_\alpha$ from $r^{\mathcal{D}_i}_\alpha$ by checking whether the age of a tuple in $r^{\mathcal{D}_i}_\alpha$ is old enough:

$$p^{\mathcal{D}_i}_\alpha := \{ \bar{a} \mid (\bar{a}, y) \in r^{\mathcal{D}_i}_\alpha, \text{ for some } y \geq b \}.$$ 

Observe that as in the non-metric case above, the definition of the relation $r^{\mathcal{D}_i}_\alpha$ only depends on the relation $r^{\mathcal{D}_{i-1}}_\alpha$ when $i > 0$, and on the relations in $\mathcal{D}_i$ for which the corresponding predicates occur in the subformulas $\beta$ or $\hat{\gamma}$. Furthermore, the arithmetic constraint $y' = y - \tau_i + \tau_{i-1}$ used in the above definition of $r^{\mathcal{D}_i}_\alpha$ for $i > 0$ is first-order definable in $\mathcal{D}_i$ as $\tau_i - \tau_{i-1}$ is a constant value (see Remark 3.3). From this it follows that $r^{\mathcal{D}_i}_\alpha$ is regular and thus also $p^{\mathcal{D}_i}_\alpha$. The details are given in the following lemma.

**Lemma 3.7.** Let $\alpha = \beta S_{[b,b')} \gamma$. Under the assumption that the relations $p^{\mathcal{D}_j}_\phi$ are regular and $p^{\mathcal{D}_j}_\phi = \phi^{(\mathcal{D},i)}$, for all $j \leq i$ and $\phi \in \text{tsub}(\beta) \cup \text{tsub}(\gamma)$, the following properties hold:

(i) The relation $r^{\mathcal{D}_i}_\alpha$ is regular and for all $\bar{a} \in \mathbb{N}^n$ and $y \in \mathbb{N},$

$$(\bar{a}, y) \in r^{\mathcal{D}_i}_\alpha \iff \text{there is a } j \text{ with } 0 \leq j \leq i \text{ such that } y = \tau_i - \tau_j < b', \ \bar{a} \in \hat{\gamma}^{(\mathcal{D},i)} \text{, and } \bar{a} \in \beta^{(\mathcal{D},i)} \text{, for all } k \text{ with } j < k \leq i.$$ 

(ii) The relation $p^{\mathcal{D}_i}_\alpha$ is regular and $p^{\mathcal{D}_i}_\alpha = \alpha^{(\mathcal{D},i)}$.

**Proof.** Property (ii) follows immediately from (i) and the definition of $p^{\mathcal{D}_i}_\alpha$. We prove (i) by induction over $i$.

**Base case $i = 0$:** The set $r^{\mathcal{D}_0}_\alpha$ is regular, since it can be defined by the formula

$$\psi(\bar{x}, y) := \hat{\gamma}(\bar{x}) \land \neg \exists z. \text{ succ}(z, y).$$

Note that, by assumption, the relations for the predicates occurring in $\hat{\gamma}$ are regular.
The equivalence for \( i = 0 \) follows from the definition of \( r_{\alpha}^{D_1} \), from the assumption, and from Lemma 3.4. Note that \( \tau_i - \tau_{i-1} < b' \), since in the definition of the syntax of MFOTL, we require that \( I \neq \emptyset \). Hence, \( b' > 0 \).

**Step case \( i > 0 \):** We first show that \( r_{\alpha}^{D_1} \) is regular. Similar to the base case, it follows that the set \( N = \bar{\gamma}^{D_1} \times \{0\} \) is regular. The set \( U = \{(a, y) \mid a \in \beta^{D_1}, y < b', \text{ and } (a, y') \in r_{\alpha}^{D_1-1} \text{ with } y' = y - \tau_i + \tau_{i-1} \} \) is also regular. If \( b' \neq \infty \), it can be expressed by the formula

\[
\psi(x, y) := \bar{\beta}(x) \land y < b' \land \exists y'. \psi'(x, y') \land y' + (\tau_i - \tau_{i-1}) \approx y,
\]

where \( \psi' \) is the formula that defines \( r_{\alpha}^{D_1-1} \), which is regular by the induction hypothesis. Note that \( b' \) and \( \tau_i - \tau_{i-1} \) are constant values and not variables. If \( b' = \infty \), we omit the conjunct \( y < b' \). Since \( r_{\alpha}^{D_1} \) is defined as the union of \( N \) and \( U \), we conclude that \( r_{\alpha}^{D_1} \) is regular.

In the following, we show the step case for the second conjunct of (i).

\((\Rightarrow)\) If the tuple \((a, y)\) is in \( N \), then the conjunct is obviously true. Assume that \((a, y) \in U \). By definition, there is a tuple \((a, y') \) in \( r_{\alpha}^{D_1-1} \) such that \( y' = y - \tau_i + \tau_{i-1} \). By the induction hypothesis, there is an integer \( j \) with \( 0 \leq j \leq i-1 \) such that \( y' = \tau_{i-1} - \tau_j < b' \), \( a \in \bar{\gamma}^{(D, \tau, j)} \), and \( a \in \beta^{(D, \tau, k)} \) for all \( k \) with \( j < k \leq i-1 \). It follows that \( y = y' + \tau_i - \tau_{i-1} = \tau_i - \tau_j \). From the assumption, we conclude that \( a \in \beta^{(D, \tau, k)} \) for all \( k \) with \( j < k \leq i \).

\((\Leftarrow)\) If \( j = i \), it follows that \( y = 0 \). From the assumption and the definition of \( r_{\alpha}^{D_1} \), it follows that \((a, 0) \in r_{\alpha}^{D_1} \). Assume that \( j < i \). By the induction hypothesis, \((a, y') \in r_{\alpha}^{D_1-1} \) with \( y' = y - (\tau_i - \tau_{i-1}) \). From the definition of \( r_{\alpha}^{D_1} \) and the assumption, we conclude that \((a, y) \in r_{\alpha}^{D_1} \). \( \square \)

### 3.4.4. Until Operator.

We now address the bounded future-time operator \( U_I \) with \( I = [b, b') \subseteq I \) and \( b' \in \mathbb{N} \). Assume that the formula \( \alpha \) is of the form \( \beta \cup_I \gamma \). Let \( \ell_i := \max \{ j \in \mathbb{N} \mid \tau_{i+j} - \tau_i < b' \} \) be the *lookahead offset* at time point \( i \). For convenience, we additionally define \( \ell_{i-1} := 0 \). As with the since operator, we could directly define \( p_{\alpha}^{D_1} \) as

\[
\bigcup_{0 \leq j \leq \ell_i \atop \tau_{i+j} - \tau_i \geq b} (\bar{\gamma}^{D_1+i+j} \cap \bigcap_{0 \leq j' < j} \bar{\beta}^{D_1+i+j'}).
\]

However, we instead define the relation \( p_{\alpha}^{D_1} \) in terms of the incrementally-built auxiliary relations \( r_{\alpha}^{D_1} \) and \( s_{\alpha}^{D_1} \). We show next how to initialize and update these relations.

Intuitively, the relation \( r_{\alpha}^{D_1} \) contains the tuple \((a, j)\) if \( a \) satisfies \( \beta \) at the time points \( i + j, \ldots, i + \ell_i \). The relation \( s_{\alpha}^{D_1} \) contains the tuple \((a, j, j')\) if \( j \leq j' \leq \ell_i \), \( a \) satisfies \( \gamma \) at time point \( i + j' \) and \( \beta \) at the time points \( i + j, \ldots, i + j' - 1, \) and the timing constraint \( \tau_{i+j'-1} - \tau_i \geq b \) is fulfilled. Note that the timing constraint \( \tau_{i+j'} - \tau_i < b' \) also holds since we only look at time points up to \( i + \ell_i \).

We define \( r_{\alpha}^{D_1} \) as the union of a set \( N_r \) for the new elements and a set \( U_r \) for the updated tuples. That is, \( N_r \) contains the tuples that are obtained from data at the time points \( i + \ell_i - 1, \ldots, i + \ell_i \) and \( U_r \) contains the updated tuples from the time points \( i, \ldots, i + \ell_i - 1 \). Formally, these two sets are defined as follows:

\[
N_r := \{(a, j) \mid \ell_{i-1} \leq j \leq \ell_i \text{ and } a \in \bar{\beta}^{D_1+i+k}, \text{ for all } k \text{ with } j \leq k \leq \ell_i\}
\]
and $U_r := \emptyset$ if $i = 0$, and
\[
U_r := \{ (a, j \supset 1) \mid (a, j) \in r_{\beta_i}^{D_{i-1}} \text{ and } (a, \ell_{i-1}) \in N_r \},
\]
for $i > 0$, where $\supset$ is the subtraction on the non-negative integers, that is, $x \supset y := \max\{0, x - y\}$.

Analogously to $r_{\beta_i}^{D_i}$, we define the relation $s_{\beta_i}^{D_i}$ as the union of the sets $N_s$, $U_s$, and $E_s$. $N_s$ contains the tuples that are new in the sense that they are obtained from data at the time points $i, i + \ell_{i-1}, \ldots, i + \ell_i$. $U_s$ contains the updated data from the time points $i, i + \ell_{i-1} + 1$. $E_s$ contains the data from the time points $i, i + \ell_{i-1} - 1$ that can be extended to the new time points $i + \ell_{i-1}, \ldots, i + \ell_i$. Formally, we define
\[
N_s := \{ (a, j \supset 1) \mid \ell_i \leq j \leq j' \leq \ell_i, a \in \gamma^{D_{i+j'}}, \tau_i + j' - \tau_i \geq b, \text{ and } a \in \hat{\beta}^{D_{i+k}}, \text{ for all } k \text{ with } j \leq k < j' \}
\]
and $U_s := E_s := \emptyset$ if $i = 0$. For $i > 0$, we define
\[
U_s := \{ (a, j \supset 1, j' \supset 1) \mid (a, j, j') \in s_{\beta_i}^{D_{i-1}} \text{ and } \tau_i + (j' \supset 1) - \tau_i \geq b \}
\]
and, with the help of $s_{\beta_i}^{D_{i-1}}$ and $N_s$, we define
\[
E_s := \{ (a, j \supset 1, j') \mid (a, j) \in r_{\beta_i}^{D_{i-1}} \text{ and } (a, \ell_{i-1}, j') \in N_s \}.
\]
Finally, with the relation $s_{\beta_i}^{D_{i-1}}$ at hand, we define
\[
p_{\alpha}^{D_i} := \{ \bar{a} \mid (a, 0, j') \in s_{\beta_i}^{D_i}, \text{ for some } j' \geq 0 \}.
\]

**Lemma 3.8.** Let $\alpha = \beta \cup_{(b, b')} \gamma$ with $b' \in \mathbb{N}$. Under the assumption that the relations $p_{\alpha}^{D_k}$ are regular and $p_{\alpha}^{D_k} = \phi^{(D, \bar{r}, k)}$, for all $k \leq i + \ell_i$ and $\phi \in \text{tsub}(\beta) \cup \text{tsub}(\gamma)$, the following properties hold:

(i) The relation $r_{\alpha}^{D_i}$ is regular and for all $a \in \mathbb{N}$ and $j \in \mathbb{N}$,
\[
(a, j) \in r_{\alpha}^{D_i} \iff a \in \beta^{(D, \bar{r}, i+k)}, \text{ for all } k \text{ with } j \leq k \leq \ell_i.
\]

(ii) The relation $s_{\alpha}^{D_i}$ is regular and for all $a \in \mathbb{N}$ and $j, j' \in \mathbb{N}$,
\[
(a, j, j') \in s_{\alpha}^{D_i} \iff \begin{cases} j \leq j', \tau_i + j' - \tau_i \in [b, b'), a \in \gamma^{(D, \bar{r}, i+j')}, \text{ and } a \in \beta^{(D, \bar{r}, i+k)}, \text{ for all } k \text{ with } j \leq k < j'. 
\end{cases}
\]

(iii) The relation $p_{\alpha}^{D_i}$ is regular and $p_{\alpha}^{D_i} = \alpha^{(D, \bar{r}, i)}$.

**Proof.** Property (iii) follows immediately from (ii) and the definition of $p_{\alpha}^{D_i}$. In the following, we prove (i) by induction over $i$. We omit the proof of (ii), which is similar to (i)'s proof.

**Base case** $i = 0$: Observe that $r_{\alpha}^{D_0} = N_r$. For each $j$ with $0 \leq j \leq \ell_0$, the set of the first components $\bar{a}$ of the tuples $(a, j)$ in $N_r$ is the finite intersection of regular sets. It follows that $N_r$ is the finite union of regular sets. The second conjunct of (ii) for $i = 0$ follows directly from the definition of $N_r$ and the assumption.

**Step case** $i > 0$: To show that $r_{\alpha}^{D_i}$ is regular, it suffices to show that $N_r$ and $U_r$ are regular. As in the base case we conclude that $N_r$ is regular. The regularity of $U_r$ follows from the induction hypothesis and the regularity of $N_r$. The second conjunct of (ii) for $i > 0$ follows straightforwardly from the induction hypothesis, the definitions of $N_r$ and $U_r$, and the assumption. □
that is, we use the formula
\[ Φ := \text{from Example 2.4.} \]
To determine which elements violate the specified property at which time points, we drop the outermost temporal operator \( \square \) and make \( x \) a free variable, that is, we use the formula \( Φ := in(x) \rightarrow □[0,6) out(x) \) for monitoring. In other words, for a given temporal structure \((\overline{D}, \overline{τ})\), the objective of the monitoring algorithm is to successively compute and output the sets \((-Φ)^{(\overline{D},\overline{τ},0)}, (-Φ)^{(\overline{D},\overline{τ},1)}, \ldots\).

Before presenting our monitoring algorithm, we illustrate the formula transformation and the constructions of the auxiliary relations with the formula

\[ \square \forall x. in(x) \rightarrow □[0,6) out(x) \]

from Example 2.4. To determine which elements violate the specified property at which time points, we drop the outermost temporal operator \( \square \) and make \( x \) a free variable, that is, we use the formula \( Φ := in(x) \rightarrow □[0,6) out(x) \) for monitoring. In other words, for a given temporal structure \((\overline{D}, \overline{τ})\), the objective of the monitoring algorithm is to successively compute and output the sets \((-Φ)^{(\overline{D},\overline{τ},0)}, (-Φ)^{(\overline{D},\overline{τ},1)}, \ldots\).

Since \( α := □[0,6) out(x) \) is the only temporal subformula of \( Φ \), the extended signature \( \overline{S} \) contains, in addition to the unary predicates \( in \) and \( out \), the unary predicate \( p_α \), the binary predicate \( r_s \), and the ternary predicate \( s_α \). Recall that \( □[0,6) out(x) \) is syntactic sugar for \( true U[0,6) out(x) \). The transformed formula \( \hat{Φ} \) is \( ¬in(x) \vee p_α(x) \).

We illustrate the incremental constructions of the auxiliary relations for the temporal formula \( α \) by considering the temporal structure \((\overline{D}, \overline{τ})\) in Figure 1, where \( a, b, c, \) and \( d \) are pairwise distinct elements in \(|\overline{D}| = N\). Since the incremental construction for the temporal operator \( U[0,6) \) assumes that the direct subformulas of \( α \) have the same vector of free variables, we add the conjunct \( x \approx x \) to the subformula \( true \).

At time point 0, the lookahead \( ℓ_0 \) is 3 because \( τ_3 - τ_0 < 0 \) and \( τ_4 - τ_0 = 6 \). The relation \( r_α^{D_0} \) is \( N \times \{0, 1, 2, 3\} \) and the relation \( s_α^{D_0} \) consists of the pairs \((a, j', j')\) with \( j \leq j' \leq ℓ_0 \) and \( a \in out^{D_0} \), that is, \( s_α^{D_0} = \{(b, 0, 2), (b, 1, 2), (b, 2, 2), (a, 0, 3), (a, 1, 3), (a, 2, 3), (a, 3, 3)\} \). The relation \( p_α^{D_0} = \{a, b\} \). When evaluating \( \hat{Φ} \) at time point 0, we obtain \( \hat{Φ}^{D_0} = \{in^{D_0} \cup p_α^{D_0} = N \setminus \{c\}\}. The violating elements at time point 0 are therefore \((-\hat{Φ})^{D_0} = \{c\} \).

At time point 1, the lookahead \( ℓ_1 \) is 2. Since \( ℓ_1 = ℓ_0 - 1 \), we need not consider any new time points. We obtain \( r_α^{D_1} \) and \( s_α^{D_1} \) from \( r_α^{D_0} \) and \( s_α^{D_0} \), respectively, by updating the relative indices in the tuples contained in \( r_α^{D_0} \) and \( s_α^{D_0} \), yielding \( r_α^{D_1} = N \times \{0, 1, 2\} \), \( s_α^{D_1} = \{(b, 0, 1), (b, 1, 1), (a, 0, 2), (a, 1, 2), (a, 2, 2)\} \), and \( p_α^{D_1} = \{a, b\} \). The set of violating elements at time point 1 is \((-\hat{Φ})^{D_1} = N \setminus ((N \setminus in^{D_1}) \cup p_α^{D_1}) = \{d\} \).

For the time point 2, we must also account for the new time point 4, since \( ℓ_2 = 2 \). We obtain the relation \( r_α^{D_2} = N \times \{0, 1, 2\} \) and the relation \( s_α^{D_2} = U_s \cup N_s \cup E_s \), with \( U_s = \{(b, 0, 0), (a, 0, 1), (a, 1, 1)\} \) by updating the indices of the tuples in \( s_α^{D_1} \), and \( N_s = \{(d, 2, 2)\} \) and \( E_s = \{(d, 0, 2), (d, 1, 2)\} \) by taking the additional structure at time point 4 into account. Furthermore, we get \( p_α^{D_2} = \{a, b, d\} \). The set of violating elements at time point 2 is \((-\hat{Φ})^{D_2} = N \setminus ((N \setminus in^{D_2}) \cup p_α^{D_2}) = ∅ \).

Obviously, the incremental construction for the bounded future operator \( □_I \) can be optimized. In particular, the auxiliary predicate \( r_s \) and its relations are superfluous in this case. Furthermore, the set \( E_s \) in an incremental construction and the first index \( j \) in the tuples \((a, j, j')\) of the relations for the auxiliary predicate \( s_α \) can be ignored.

![Fig. 1: A temporal structure.](image-url)
3.6. Monitoring Algorithm

Figure 2 presents our monitoring algorithm $M_\Phi$. To detect violations, $M_\Phi$ iteratively builds the relations of the extended structures $\mathcal{D}_0, \mathcal{D}_1, \ldots$ using the incremental constructions from Section 3.4. Without loss of generality, we assume that each temporal subformula occurs only once in $\Phi$. In the following, we describe $M_\Phi$’s operation.

$M_\Phi$ uses two counters $\ell$ and $i$. The counter $\ell$ is the index of the current element $(\mathcal{D}_\ell, \tau_\ell)$ in the input sequence $(\mathcal{D}_0, \tau_0), (\mathcal{D}_1, \tau_1), \ldots$, which is processed sequentially. Initially, $\ell$ is 0 and it is incremented with each loop iteration (lines 4–13). The counter $i$ is the index of the next time point $i$ (possibly in the past, from $\ell$’s point of view) for which we evaluate $\Phi$ over the extended structure $\mathcal{D}_i$. The evaluation is delayed until $\mathcal{D}_i$ is complete, that is, all the auxiliary relations are built (lines 8–11). Furthermore, $M_\Phi$ uses the list $^1 Q$ to ensure that the auxiliary relations of $\mathcal{D}_0, \mathcal{D}_1, \ldots$ are built at the right time: if $(\alpha, j, \emptyset)$ is an element of $Q$ at the beginning of a loop iteration, enough time has elapsed to build the auxiliary relations for the temporal subformula $\alpha$ of the structure $\mathcal{D}_i$. $M_\Phi$ initializes $Q$ in line 3. The function $\text{waitfor}$ identifies the subformulas that delay the formula evaluation:

$$
\text{waitfor}(\alpha) := \begin{cases} 
\text{waitfor}(\beta) & \text{if } \alpha = \neg \beta, \alpha = \exists x, \beta, \text{ or } \alpha = \bullet, \beta, \\
\text{waitfor}(\beta) \cup \text{waitfor}(\gamma) & \text{if } \alpha = \beta \lor \gamma \text{ or } \alpha = \beta S_I \gamma, \\
\{\alpha\} & \text{if } \alpha = O_I \beta \text{ or } \alpha = \beta U_I \gamma, \\
\emptyset & \text{otherwise.}
\end{cases}
$$

The list $Q$ is updated in line 12 before we increment $\ell$ in line 13 and start a new loop iteration. The update adds a new tuple $(\alpha, \ell + 1, \text{waitfor}(\alpha))$ to $Q$, for each temporal subformula $\alpha$ of $\Phi$, and it removes tuples of the form $(\alpha, j, \emptyset)$ from $Q$. Moreover, for tuples $(\alpha, j, S)$ with $S \neq \emptyset$, the set $S$ is updated using the functions $\text{waitfor}$ and $\text{update}$, accounting for the elapsed time to the next time point, that is, $\tau_{i+1} - \tau_i$. For a set of

$^1$We abuse notation by using set notation for lists. Moreover, we assume that $Q$ is ordered so that $(\alpha, j, S)$ occurs before $(\alpha', j', S')$, whenever $\alpha$ is a proper subformula of $\alpha'$, or $\alpha = \alpha'$ and $j < j'$. 
formulas $U$ and $t \in \mathbb{N}$, $update(U, t)$ is the set
\[
\{ \beta \mid \circ \gamma \in U \} \cup \{ \beta U_{[0, b-t-1]} \gamma \mid \beta U_{[b, b']}, \gamma \in U, \text{ with } b' - t > 0 \} \cup \\
\{ \beta \mid \beta U_{[b, b']} \gamma \in U \text{ or } \gamma U_{[b, b']} \beta \in U, \text{ with } b' - t \leq 0 \}.
\]

In line 7, we build the auxiliary relations for which enough time has elapsed, that is, the relations for $\alpha$ in $D_j$ with $(\alpha, j, \emptyset) \in Q$. To build the relations, we use the incremental constructions described earlier in this section. In lines 8–12, if all the relations of $D_i$ have been built, then $M_\Phi$ outputs the valuations violating $\Phi$ at time point $i$ together with the time stamp $\tau_i$. Furthermore, after each output, the extended structure $D_{i-1}$ is discarded (if $i > 0$) and $i$ is incremented.

Note that because $M_\Phi$ does not terminate, it is not an algorithm in the strict sense. However, it effectively determines the elements violating $\Phi$, for every time point.

**Theorem 3.9.** The monitoring algorithm $M_\Phi$ has the following properties:

(i) Whenever $M_\Phi$ executes line 9, then the output set is effectively computable, regular, and equals $(\Phi)^{(2, \bar{\tau}, i)}$.

(ii) For each $n \in \mathbb{N}$, $M_\Phi$ eventually sets the counter $i$ to $n$ by executing line 11.

**Proof.** For the proof, we index the program variable $Q$ of the monitoring algorithm $M_\Phi$ by the loop iteration when processing the given input sequence: for an integer $k \in \mathbb{N}$, $Q_k$ denotes the list when we enter the $(k+1)$st loop iteration. For example, $Q_0$ is the initialized list from line 3 and $Q_1$ is the list after the first update in line 12. Analogously, we index the counters $i$ and $\ell$ with $k \in \mathbb{N}$: $i_k$ and $\ell_k$ denote the counters’ values when entering the $(k+1)$st loop iteration. Note that $i_k = k$.

We start with some observations about the tuples stored in the list $Q$. Assume that $\alpha$ is a formula, $j \in \mathbb{N}$, and $S$ a set of formulas.

(1) For all $k \in \mathbb{N}$, we have $(\alpha, k, \text{waitfor}(\alpha)) \in Q_k$. This directly follows from the list $Q$’s initialization (line 3) and update (line 12).

(2) For all $k \in \mathbb{N}$, if $(\alpha, j, S) \in Q_k$ then $(\alpha, j, \emptyset) \in Q_{k'}$, for some $k' \geq k$. This follows from $Q$’s update (line 12), in particular from the application of the functions $\text{waitfor}$ and $\text{update}$, and because the sequence of time stamps $\tau_0, \tau_1, \ldots$ is monotonically increasing and progressing.

(3) For all $k \in \mathbb{N}$, if $(\alpha, j, S) \in Q_k$ then $j \geq i_k$. To see this, we first observe that $\ell_k \geq i_k$, for all $k \in \mathbb{N}$. It follows that the tuples that we add to the list $Q$ in line 3 and in line 12 before the $(k+1)$st loop iteration ends by incrementing the counter $\ell_k$ have a second component that is at least $i_k$. Furthermore, we observe that we only increment the counter $i$ (line 11) after all relations of $D_{i_k}$ have been built and after building the corresponding relations for a tuple in the list $Q$ in line 7, we remove it from $Q$ when updating the list in line 12.

From (1) and (2), it follows that for every temporal subformula $\alpha$ of $\Phi$ and $j \in \mathbb{N}$, we eventually execute line 7, where we build the auxiliary relations for $\alpha$ of the extended structure $D_j$. Hence for every value of the counter $i$, the while loop’s condition (line 8) eventually becomes true in some loop iteration and $i$ is eventually incremented (line 11). We conclude that the property (ii) holds.

We turn now to the property (i) of the theorem. We show that for each temporal subformula $\alpha$ of $\Phi$ and all $k, j \in \mathbb{N}$, if $(\alpha, j, \emptyset) \in Q_k$ then line 7 of the monitoring algorithm $M_\Phi$ can be executed. That is, the relations involved in the respective incremental construction (depending on $\alpha$’s main connective and given in Section 3.4) of the auxiliary relations for the temporal subformula $\alpha$ have been built earlier and have not yet been
discarded. From the respective lemma in Section 3.4, it follows that $p^\alpha_j = p^\alpha_{i\ell, j}$ and $p^\alpha_j$ is regular and effectively computable. Hence property (i) holds.

Relations are not discarded too early. To see this, assume $(\alpha, j, \emptyset) \in Q_k$, for some $j, k \in \mathbb{N}$. The relations necessary for executing line 7 of the monitoring algorithm $M_{\Phi}$ are from the extended structure $\hat{D}_{k-1}$ if $i_k > 0$ and subsequent extended structures. Since we have that $j \geq i_k$ by (3), none of these structures has been discarded yet by the execution of line 10 in some previous loop iteration.

It remains to prove that the relations are not built too late. We make a case split on $\alpha$'s main temporal connective, assuming $(\alpha, j, \emptyset) \in Q_k$, for some $j, k \in \mathbb{N}$.

**Case $\alpha = \bullet_j \beta$.** For $j = 0$, there is nothing to prove since $p^\alpha_0 = \emptyset$. For $j > 0$, the construction from Section 3.4.1 of the relation $p^\alpha_j$ uses at most the relations $r^{\hat{D}}_{j-1}$ with $r \in R$ and the auxiliary relations $p^\delta_{j-1}$ with $\delta \in tsub(\beta)$.

The relations of the predicates in $R$ have been carried over to the extended structure $\hat{D}_{j-1}$ by the execution of line 5 of the monitoring algorithm $M_{\Phi}$ in a previous loop iteration in which the counter $i$ had the value $j - 1$.

Assume $\delta \in tsub(\beta)$. There is an integer $k' \in \mathbb{N}$ with $k' \leq k$ such that $(\delta, j - 1, \emptyset) \in Q_{k'}$. This follows from the observation that $M_{\Phi}$ puts the tuple $(\delta, j - 1, waitfor(\delta))$ in the list $Q$ in the $j$th loop iteration and the tuple $(\alpha, j, waitfor(\alpha))$ in the $(j + 1)$st loop iteration. In each subsequent loop iteration, $M_{\Phi}$ updates the third component of each of these tuples until it becomes the empty set (line 12). By the functions $waitfor$ and $update$, we have that the third component $S$ of the tuple $(\alpha, j, S)$ does not become the empty set before the third component $S'$ of the tuple $(\delta, j - 1, S')$. Given the order of the elements in the list $Q$, it follows that monitoring algorithm $M_{\Phi}$ builds the relation $p^\delta_{j-1}$ before $p^\alpha_j$.

**Case $\alpha = \beta S_j \gamma$.** The construction from Section 3.4.3 of $p^\alpha_j$ is only based on the auxiliary relation $r^{\hat{D}}_j$. The given construction of $r^{\hat{D}}_j$ in turn uses at most the relations $r^{\hat{D}}_j$ with $r \in R$, the auxiliary relations $p^\delta_j$ with $\delta \in tsub(\beta) \cup tsub(\gamma)$, and the auxiliary relation $r^{\hat{D}}_{j-1}$ if $j > 0$. By a similar argumentation as given above for the temporal operator $\bullet_j$, it follows that the monitoring algorithm $M_{\Phi}$ builds all these relations before building $r^{\hat{D}}_j$.

**Case $\alpha = \odot_j \beta$.** The construction from Section 3.4.2 of $p^\alpha_j$ uses at most the relations $r^{\hat{D}}_{j+1}$ with $r \in R$ and the auxiliary relations $p^\delta_{j+1}$ with $\delta \in tsub(\beta)$. Because of the initialization (line 3) and the updates (line 13) of the list $Q$, we have that $(\alpha, j, \{\alpha\}) \in Q_j$ and $(\alpha, j, waitfor(\alpha)) \in Q_{j+1}$. It follows that $k \geq j + 1$. Thus, the monitoring algorithm $M_{\Phi}$ carries over the relations for the predicates in $R$ to the extended structure $\hat{D}_{j+1}$ before building the auxiliary relation $p^\alpha_j$. For $\delta \in tsub(\beta)$, we have that $(\delta, j + 1, waitfor(\delta)) \in Q_{j+1}$ with $waitfor(\delta) \subseteq waitfor(\beta)$. Analogously, as in the case for the temporal operator $\bullet_j$, we conclude that $M_{\Phi}$ builds $p^\delta_{j+1}$ before $p^\alpha_j$.

**Case $\alpha = \beta U_I \gamma$ with $I = [b, b')$.** The monitoring algorithm $M_{\Phi}$ postpones the constructions of the auxiliary relations $r^{\hat{D}}_{\alpha}, s^{\hat{D}}_{\alpha}$, and $p^\alpha_j$ for at least $k'$ loop iterations, for some $k' \in \mathbb{N}$ with $\tau_{j+k'} - \tau_j \geq b'$. This follows from the definition of the functions $waitfor$ and $update$ used for initializing and updating the list $Q$: we have that for all $k'' \in \mathbb{N}$ with $\tau_{j+k''} - \tau_j < b'$, there is some interval $I'$ such that $(\alpha, j, \{\beta U_I \gamma\}) \in Q_{j+k''}$. 


It follows that \( \tau_k - \tau_j \geq b' \). Thus, the relations for the predicates in \( R \) used in the construction given in Section 3.4.4 of \( r_{\alpha}^{D_j}, s_{\alpha}^{D_j} \), and \( p_{\alpha}^{D_j} \) have been carried over by the monitoring algorithm \( M_\Phi \) to the extended structures. Assume \( \delta \in tsub(\beta) \cup tsub(\gamma) \).

The monitoring algorithm \( M_\Phi \) postpones the construction of \( r_{\alpha}^{D_j}, s_{\alpha}^{D_j} \), and \( p_{\alpha}^{D_j} \) further until the auxiliary relations \( p_{\delta}^{D_j+k''} \), for all \( k'' \in \mathbb{N} \) with \( k'' \leq k' \) have been built. To see this, observe that for each such \( k'' \), we have that \( (\delta, j + k'', \text{waitfor}(\delta)) \in Q_{j+k''} \) and \( (\alpha, j, \text{waitfor}(\beta) \cup \text{waitfor}(\gamma)) \in Q_{j+k''} \) and \( \text{waitfor}(\delta) \subseteq \text{waitfor}(\beta) \cup \text{waitfor}(\gamma) \). □

4. MONITORING WITH FINITE RELATIONS

In this section, we shall assume that the relations that can change over time are finite. In this case, data structures and algorithms from relational databases provide an alternative to automata for implementing the monitoring algorithm \( M_\Phi \). When representing relations as finite tables, however, we inherit standard problems from database theory, which we illustrate in Section 4.1. Afterwards, in Section 4.2, we present a restricted class of formulas that \( M_\Phi \) can handle.

4.1. Example Revisited

The incremental constructions from Section 3.4 fail when the auxiliary relations are required to be finite. In particular, Lemmas 3.5–3.8 are invalid when replacing the word “regular” by “finite.” The constructed relations are still regular, but possibly infinite.

To illustrate some of the obstacles in monitoring with finite relations, consider again the formula \( \Box \forall x. \text{in}(x) \rightarrow \Diamond_{[0,6]} \text{out}(x) \) from Example 2.4. The relations for the predicates \( \text{in} \) and \( \text{out} \) can change over time and now we assume that they are finite at every time point. As in Section 3.5, for monitoring we drop the outermost temporal operator and the quantification. We also negate the formula since we want to detect violations. Moreover, we now push negation inwards as otherwise we could not evaluate the formula inductively over its structure, where intermediate results are stored in finite tables. If we push the negation all the way down to the predicates we obtain \( \Phi := \text{in}(x) \land \Box_{[0,6]} \neg \text{out}(x) \). Unfortunately, we cannot use \( \Phi \) for monitoring since the relation interpreting \( \neg \text{out}(x) \) is infinite. However, we can monitor the formula \( \text{in}(x) \land \neg \Diamond_{[0,6]} \text{out}(x) \). The auxiliary relations for the subformula \( \Diamond_{[0,6]} \text{out}(x) \) are always finite and, furthermore, although \( \neg \Diamond_{[0,6]} \text{out}(x) \) describes an infinite set, its conjunction with \( \text{in}(x) \) guarantees the finiteness of the result. In particular, if \( I \) and \( O \) are the finite sets of elements that satisfy \( \text{in}(x) \) and \( \Diamond_{[0,6]} \text{out}(x) \) at a time point \( i \in \mathbb{N} \), respectively, then \( I \setminus O \) is the set of elements that satisfy \( \text{in}(x) \land \neg \Diamond_{[0,6]} \text{out}(x) \) at time point \( i \).

There are often different syntactic alternatives available that yield monitorable formulas. Returning to \( \Phi = \text{in}(x) \land \Box_{[0,6]} \neg \text{out}(x) \), we can copy the conjunct \( \text{in}(x) \) into the temporal subformula. That is, we rewrite \( \Phi \) into the logically equivalent formula \( \Phi' := \text{in}(x) \land \Box_{[0,6]} \neg \text{out}(x) \land \Diamond_{[0,6]} \text{in}(x) \). Observe that at each time point, there are only finitely many elements that satisfy \( \Diamond_{[0,6]} \text{in}(x) \) and thus only finitely many that satisfy \( \neg \text{out}(x) \land \Diamond_{[0,6]} \text{in}(x) \). In fact, the relations for the auxiliary predicates for the temporal subformulas \( \Diamond_{[0,6]} \text{in}(x) \) and \( \Box_{[0,6]} \neg \text{out}(x) \land \Diamond_{[0,6]} \text{in}(x) \) of \( \Phi' \) are all finite.

As a second example, consider

\[
\Box \forall x. \forall y. \text{in}(x,y) \rightarrow \Diamond_{[0,5]} \text{out}(x) \land \neg \text{out}(y) \lor x \approx y,
\]

where \( \text{in} \) is a binary predicate. The formula states that the first component \( x \) of \( \text{in} \) must eventually be output (within the given bound) and the second component \( y \) must not simultaneously be output if \( y \) is different from \( x \). Observe that neither \( \Diamond_{[0,5]} \text{out}(x) \land \neg \text{out}(y) \lor x \approx y \) nor its negation is guaranteed to be ful-
filled by only finitely many elements. However, by rewriting, we obtain the formula
\[ \text{in}(x, y) \land \Box_{[0,5]} (\neg \text{out}(x) \lor \text{out}(y) \land \neg x \approx y) \land \Diamond_{[0,6]} \text{in}(x, y), \]
which is monitorable.

4.2. Monitorable Fragment

Throughout this section, we fix a signature \( S = (C, R, i) \). We distinguish in the following
between predicates whose corresponding relations are rigid over time and those
that are flexible, that is, their interpretations can change over time. Let \( F \subseteq R \) be the
set of flexible predicates. Let \((D, \bar{r})\) be a temporal structure with \( D = (D_0, D_1, \ldots) \). We
call \((D, \bar{r})\) a temporal database if (1) the domain \( |D| \) is countably infinite, (2) for each
\( r \in F \) and \( i \in \mathbb{N} \), the relation \( r^{D}_i \) is finite, and (3) for each \( r \in R \setminus F \) and \( i \in \mathbb{N} \), the
relation \( r^{D}_i \) is a decidable set and \( r^{D}_{i+1} = r^{D}_i \). We also assume in the following that
\( \mathbb{N} \cup \{\infty\} \subseteq |D| \) and that there is a binary predicate \(<\) in \( R \setminus F \), which is interpreted as
the standard ordering \(<\) on \( \mathbb{N} \) and \( n < \infty \), for all \( n \in \mathbb{N} \). Note that the finiteness assumption
on the relations interpreting the flexible predicates is more restrictive than the
regularity assumption in Section 3.1. In contrast, for the rigid predicates, we are
less restrictive. Furthermore, we do not fix the domain of a temporal structure \((D, \bar{r})\)
to \( \mathbb{N} \); instead we only require that \( N \) is included in \(|D|\).

4.2.1. Domain Independence. In database theory, the finiteness of queries can be guaran-
teed by restricting the range of variables to the so-called active domain, which is the
set of domain elements that occur in a table of the database or in the query itself. This
relativization is sound with respect to the first-order semantics for so-called domain
independent queries, see [Abiteboul et al. 1995]. The generalization to our temporal
setting is as follows.

Let \((D, \bar{r})\) be a temporal database, with \( \bar{D} = (D_0, D_1, \ldots) \) and \( \bar{r} = (r_0, r_1, \ldots) \). We
say that \( \ell \in \mathbb{N} \cup \{\infty\} \) is a lookahead at time point \( i \in \mathbb{N} \) for the formula \( \phi \) and \((D, \bar{r})\)
if \( \phi(D, \bar{r}, i) = \phi(D', \bar{r}', i) \), for all temporal databases \((D', \bar{r}')\), with \( D'_k = D_k \) and \( r'_k = r_k \),
for all \( k < \ell \). When \( \phi \) is bounded then there is always a lookahead \( \ell \in \mathbb{N} \) at \( i \) for \( \phi \) and
\((D, \bar{r})\), since bounded formulas refer only to finitely many time points in the future. For
\( D \subseteq |D|, \models_D \) denotes the relation \( \models \) defined in Definition 2.2, except that quantification
is relativized to the set \( D \). The active domain of \((D, \bar{r})\) and \( \ell \in \mathbb{N} \cup \{\infty\} \) is

\[
\text{adom}(D, \ell) := \{ c^{\bar{D}} \mid c \in C \} \cup \bigcup_{r \in F} \{ d_i \mid d_i \in |D| \} \quad \text{for some} \quad \{ d_1, \ldots, d_{\ell(r)} \} \in r^{D}_r \quad \text{and} \quad 1 \leq i \leq \ell(r) .
\]

The set \( \text{adom}(D, \ell) \) is finite if \( \ell \in \mathbb{N} \). Let \( v \) be some valuation. The formula \( \phi \) with free
variables \( \bar{x} = (x_1, \ldots, x_n) \) is domain independent if for all temporal databases \((D, \bar{r})\),
\( i \in \mathbb{N} \), and \( D', \bar{r}' \subseteq |D| \), it holds that

\[
\{ \bar{d} \in D^n \mid (D, \bar{r}, v[\bar{x} \mapsto \bar{d}], i) \models_D \phi \} = \{ \bar{d} \in D'^n \mid (D, \bar{r}, v[\bar{x} \mapsto \bar{d}], i) \models_{D'} \phi \},
\]

whenever \( \text{adom}(D, \ell) \subseteq D, D' \), where \( \ell \in \mathbb{N} \cup \{\infty\} \) is a lookahead at \( i \) for \( \phi \) and \((D, \bar{r})\).

For bounded formulas, domain independence obviously implies finiteness. However,
determining whether a formula is domain independent is undecidable. In fact, the
decision problem is already undecidable in the non-temporal setting [Di Paola 1969].
We therefore present in Section 4.2.2 a syntactically defined fragment of MFOTL that
guarantees finiteness and also domain independence when imposing additional re-
strictions on the atomic formulas with rigid predicates. With additional requirements
on the temporal subformulas, which we present in Section 4.2.3, formulas can be evalu-
ated inductively over their structure without restricting the range of variables explicitly
to the active domain as is done by Chomicki et al. [2001]. Restricting the range of
variables explicitly to the active domain produces a significant overhead when evaluating formulas, which grows with the size of the active domain over time.

4.2.2. Range Restriction. In the following, we assume that a formula’s bound variables are pairwise distinct and disjoint from the formula’s free variables. Furthermore, we treat the Boolean connectives ∧ and ∨ as a primitive. We label the subformulas of a formula \( \phi \), starting with the atomic formulas and propagate these labels to the root of \( \phi \)'s syntax tree. A labeling is a set of restriction facts, each of the form \( B \rightarrow h \), with \( B \subseteq V \) and \( h \in V \). Intuitively, the meaning of \( B \rightarrow h \) is that if the ranges of the variables in \( B \) are restricted, then the range of the variable \( h \) is restricted. The labeling rules are given in Figure 3, which we briefly explain in the following. The derivation in Figure 4 shows that the range of the variable \( x \) is restricted in the formula \( \in(x) \land \neg \circ_{[0,6]} \mathit{out}(x) \) from Section 4.1. Recall that \( \circ_{I} \phi \) abbreviates \( \mathit{true} \cup I \phi \), where \( \mathit{true} \) is syntactic sugar for \( c \equiv c \), for some \( c \in C \).

Atomic formulas are labeled by the empty set. Admissible restriction facts can always be added to a labeling of an atomic formula. A restriction fact \( \{y_1, \ldots, y_n\} \rightarrow x \) is admissible for the atomic formula \( \alpha \) if \( x, y_1, \ldots, y_n \in \mathit{free}(\alpha) \) and for every structure \( D \), all finite sets \( D_1, \ldots, D_n \subseteq |D| \), and every valuation \( v \) with \( v(y_1) \in D_1, \ldots, v(y_n) \in D_n \), there are only finitely many \( d \in |D| \) with \( (D, v[x \mapsto d]) \models \alpha \). We assume that we can determine whether a restriction fact \( B \rightarrow h \) is admissible for \( \alpha \). For example, restriction facts of the form \( \emptyset \rightarrow h \) are admissible for an atomic formula \( \forall(y_1, \ldots, t_n) \) if \( r \in F \) and \( h = t_i \), for some \( i \in \mathbb{N} \) with \( 1 \leq i \leq n \). The restriction fact \( \emptyset \rightarrow x \) is admissible for \( x \approx c \) when \( c \) is a constant symbol in \( C \) and \( \{y\} \rightarrow x \) is admissible for \( x \prec y \), since there are only finitely many non-negative integers that are smaller than \( y \).

A labeling \( L \) for a formula \( \phi \) can be simplified to \( L[L^*] \), where \( L^* := \{h \mid \emptyset \rightarrow h \in L\} \) and \( L[X] := \{B \mid X \rightarrow h \mid B \rightarrow h \in L\} \), where \( X \subseteq V \).

The labeling rule for the Boolean connective \( \neg \) removes all restriction facts from the labeling set. For the Boolean connectives \( \land \) and \( \lor \), we combine the labelings of the subformulas. The labeling rule for the existential quantifier syntactically restricts quantification to variables that are restricted. The labeling rules for the temporal operators \( \bullet_I \) and \( \circ_I \) propagate the labeling from the operator's subformula. The labeling rules for \( S_I \) and \( U_I \) only propagate the labeling of the second subformula.
A formula $\phi$ is X-range-restricted, with $X \subseteq \text{free}(\phi)$, if there is a derivation tree for $\phi : L$, for some labeling $L$ with $X \subseteq L^*$. If $X = \text{free}(\phi)$, we just say that $\phi$ is range-restricted. Note that the ranges of the quantified variables are restricted in $\emptyset$-range-restricted formulas.

**Lemma 4.1.** Let $\phi$ be a formula, $X \subseteq \text{free}(\phi)$, $(\mathcal{D}, \bar{\tau})$ a temporal database, and $i \in \mathbb{N}$. It is decidable whether $\phi$ is X-range-restricted. If $\phi$ is range-restricted and bounded then $\phi(\mathcal{D}, \tau, i)$ is finite. Furthermore, $\phi$ is domain independent if $\phi$'s range restriction can be shown by only using the labeling $\emptyset$ for atomic subformulas with rigid predicates.

**Proof.** To determine whether $\phi$ is X-range-restricted, we label the leaves of $\phi$’s syntax tree and propagate these to the root. It is sufficient to consider maximal labelings for the leaves, that is, if $L$ is a labeling of the atomic formula $\alpha$ and $B \rightarrow h$ is admissible for $\alpha$ then $B \rightarrow h \in L$ or there is a $B' \rightarrow h \in L$ with $B' \subseteq B$. Furthermore, we only propagate a labeling $L$ if it is simplified, that is, $L = L[L^*]$.

When $\phi$ is bounded, the finiteness of $\phi(\mathcal{D}, \tau, i)$ follows from the invariant that restriction facts in a derivation tree for $\phi : L$ are admissible. It is straightforward to show by structural induction that for every $\{y_1, \ldots, y_n\} \rightarrow x \in L$, all finite sets $D_1, \ldots, D_n \subseteq |\mathcal{D}|$ and every valuation $v$ with $v(y_1) \in D_1, \ldots, v(y_n) \in D_n$, there are only finitely many $d \in |\mathcal{D}|$ such that $(\mathcal{D}, \bar{\tau}, v[x \rightarrow d], i) \models \phi$.

If atomic formulas with rigid predicates are only labeled by $\emptyset$ in a derivation tree, then the range of all $\phi$’s variables must be restricted by atomic formulas with a flexible predicate or by $x = c$, with $c \in C$. Hence they only range over elements in the active domain. $\Box$

**4.2.3. Formula Evaluation.** Range-restricted first-order formulas with only flexible predicates can be translated to relational algebra expressions [Abiteboul et al. 1995]. They can therefore be efficiently evaluated. Their evaluation straightforwardly extends to range-restricted first-order formulas that also include rigid predicates. For illustration, consider the formula $p(y) \land \exists x. q(x, y) \lor x < y$, which is range-restricted under the assumption that $p$ and $q$ are flexible predicates. To evaluate this formula, we rewrite it to $p(y) \land \exists x. q(x, y) \lor p(y) \land x < y$ to restrict the range of the subformula $p(y) \land x < y$, and hence also $q(x, y) \lor p(y) \land x < y$. This rewritten formula can be inductively evaluated over its formula structure. In general, such rewriting combines formulas with unrestricted variables (for example, the variables in an atomic formula $\alpha$ with a rigid predicate or in a negated formula $\neg \phi$) with conjuncts that restrict the range of these variables.

In the following, we describe how and under which additional requirements the incremental constructions from Section 3.4 for the auxiliary relations for temporal subformulas can be carried out in a bottom-up manner, that is, inductively over the formula structure. Let $(\mathcal{D}, \bar{\tau})$ be a temporal database, with $\mathcal{D} = (\mathcal{D}_0, \mathcal{D}_1, \ldots)$ and $\bar{\tau} = (\tau_0, \tau_1, \ldots)$.

For the incremental construction from Section 3.4.1 for a formula $\alpha = \bullet_I \beta$, we require that $\beta$ is range-restricted. Let $\bar{x}$ be the free variables of $\beta$ and let $I = [b, b')$ with $b \in \mathbb{N}$ and $b' \in \mathbb{N} \cup \{\infty\}$. The construction of the auxiliary relation $p^\mathcal{D}_{\alpha}$ is obvious for the time point $i = 0$. For $i > 0$, the range-restricted first-order formula

$$\hat{\beta}(\bar{x}) \land \neg(\tau_i - \tau_{i-1} < b) \land \tau_i - \tau_{i-1} < b'$$

More expressive first-order fragments can be translated to relational algebra expressions; see, for example, [Van Gelder and Topor 1991]. For the sake of readability and space, we restrict ourselves here to the simple fragment of range-restricted first-order formulas and its extension with temporal operators.
describes the tuples in $p^D_\alpha$ when interpreting the predicates in $\hat{\beta}$ by the relations of the structure at the previous time point. We assume here that the signature contains constant symbols for $b$, $b'$, and $\tau_i - \tau_{i-1}$. Thus, we can obtain $p^D_\alpha$, by evaluating this formula in a bottom-up manner over the structure at the previous time point. The incremental construction from Section 3.4.2 for $\alpha = \Box I \beta$ is done analogously, where we also assume that $\hat{\beta}$ is range-restricted.

For a formula $\alpha = \beta S_I \gamma$, we require that $\text{free}(\beta) \subseteq \text{free}(\gamma)$, $\hat{\beta}$ is $\emptyset$-range-restricted, and $\hat{\gamma}$ is range-restricted. With these requirements, the incremental construction from Section 3.4.3 of the auxiliary relation $r^D_\alpha$ is as follows. We omit the case where the time point $i$ is 0, since it is subsumed by the case where $i > 0$. Let $\bar{x}$ be the free variables of $\gamma$ and $I = \{b, b'\}$ with $b \in \mathbb{N}$ and $b' \in \mathbb{N} \cup \{\infty\}$. The tuples in the relation $r^D_\alpha$, are described by the range-restricted first-order formula

$$\left( \hat{\gamma}(\bar{x}) \land y \approx 0 \right) \lor \left( \exists y'. \hat{\beta}(\bar{x}) \land r_\alpha(\bar{x}, y') \land y \approx y' + \tau_i - \tau_{i-1} \land y < b' \right),$$

where the relations for the predicates in $\hat{\beta}$ and $\hat{\gamma}$ are taken from the structure of the current time point $i$ and the relation for $r_\alpha(\bar{x}, y)$ is taken from the previous time point $i-1$. We assume here that the signature contains constant symbols for 0, $\tau_i - \tau_{i-1}$, and $b'$, and that there is a rigid predicate in the signature for the function graph of addition over $\mathbb{N}$. Note that the subformula $\hat{\beta}(\bar{x}) \land r_\alpha(\bar{x}, y')$ is range-restricted, since $r_\alpha$ is a flexible predicate and, by assumption, $\hat{\beta}$ is $\emptyset$-range-restricted and $\text{free}(\beta) \subseteq \text{free}(\gamma)$. The range-restricted first-order formula $\exists y. r_\alpha(\bar{x}, y) \land \neg y \approx b$ describes the tuples in the auxiliary relation $p^D_\alpha$, where the predicate $r_\alpha$ is interpreted by the relation $r^D_\alpha$.

For the incremental construction from Section 3.4.4 for a formula $\alpha = \beta U_I \gamma$, we require that $\text{free}(\beta) \subseteq \text{free}(\alpha)$ and that $\hat{\beta}$ and $\hat{\gamma}$ are range-restricted. Similar to the above cases, although more involved, we can describe the auxiliary relations $r^D_\alpha$, $s^D_\alpha$, and $p^D_\alpha$ by range-restricted first-order formulas. We omit the details. In contrast to the case for the temporal operator $S_I$, we require that $\hat{\beta}$ is range-restricted and not just $\emptyset$-range-restricted. The reason is that the incremental construction involves the auxiliary relations for the predicate $r_\alpha$, which depend on $\beta$ and are not restricted by $\gamma$. However, for the important case of $\diamond I \gamma$, which is syntactic sugar for $\text{true} U_I \gamma$, it suffices that $\hat{\gamma}$ is range-restricted. The incremental construction can be easily optimized for this case so that it no longer relies on the auxiliary relations for $r_\alpha$. See also Section 5.2.

4.2.4. Monitoring. For monitoring, we do not explicitly restrict the range of variables to the active domain. Instead, we require a formula for the negated property that is range-restricted and its temporal subformulas satisfy the requirements for the incremental constructions stated in Section 4.2.3. In the following, we give heuristics to obtain such a monitorable formula $\Psi$ from the formula $\Box \Psi$, where $\Psi$ is logically equivalent to $\neg \Phi$. Our heuristics have proved to be effective in practice. We obtained monitorable formulas for most of the formulas that we encountered in our case studies. See Section 6.

First, we push negation in $\neg \Phi$ inwards by iteratively rewriting subformulas of the form $\neg \neg \psi$ to $\psi$, $\neg (\psi \lor \psi')$ to $\neg \psi \land \neg \psi'$, and $\neg (\psi \land \psi')$ to $\neg \psi \lor \neg \psi'$. If we have not succeeded yet, we try to rewrite the formula further by applying the rewrite rules in Figure 5. These rules aim to push a subformula $\alpha$ inwards, where it is assumed that $\alpha$ restricts the range of variables that are not restricted by $\beta$ and $\gamma$.

Furthermore, we can try to push negation over temporal operators to obtain monitorable formulas. For example, given a subformula $\neg \circ \alpha$, where $\hat{\alpha}$ is not range-restricted, we can first rewrite it to $\circ \neg \alpha$ and then push the negation into $\alpha$. When
treating the dual temporal operators “trigger” $T_I$ and “release” $R_I$ for $S_I$ and $U_I$, respectively, as primitives, we can push negation inwards even further. The incremental constructions for these temporal operators are similar to the ones in Section 3.4. However, the corresponding labeling rules, given in Figure 6, are more restrictive than their dual counterparts. The additional constraint $0 \in I$ of these rules stems from the fact that the temporal operators $T_I$ and $R_I$ implicitly quantify universally over time points. Formulas $\phi T_I \psi$ and $\phi R_I \psi$ are therefore trivially satisfied by all valuations if there are no time points that fulfill the metric constraints specified by the interval $I$. In such a degenerated case, $\phi T_I \psi$ and $\phi R_I \psi$ describe infinite sets. If $0 \in I$, this degenerate case does not occur, since the current time point fulfills the metric constraints. To remove the constraint $0 \in I$ from the labeling rules, we must additionally require that the time stamps in the sequence $\bar{\tau} = (\tau_0, \tau_1, \ldots)$ of a temporal database $(\bar{D}, \bar{\tau})$ are sufficiently dense. That is, for every time point $i \in \mathbb{N}$, there is a time point $j \in \mathbb{N}$ such that (1) $j \leq i$ and $\tau_i - \tau_j \in I$, if the temporal operator is $T_I$, and (2) $j \geq i$ and $\tau_j - \tau_i \in I$, if the temporal operator is $R_I$.

### 5. SPACE REQUIREMENTS

In this section, we analyze the space requirements of the monitoring algorithm $M_\Phi$ and present optimizations.

#### 5.1. Memory Usage

In the following, we assume that $\Psi$ is the formula used by the monitoring algorithm $M_\Phi$. When using automata to represent relations, $\Psi$ equals $\neg \Phi$. In the finite relation case, we obtain the monitorable formula $\Psi$, for example, by rewriting $\neg \Phi$ as described in Section 4.2.4. Note that in the latter case $\Psi$ must fulfill additional requirements to be monitorable. Since $M_\Phi$ iteratively processes the structures and time stamps in the temporal database $(\bar{D}, \bar{\tau})$, our upper bounds are given in terms of the processed prefix of $(\bar{D}, \bar{\tau})$, with $\bar{D} = (\bar{D}_0, \bar{D}_1, \ldots)$ and $\bar{\tau} = (\tau_0, \tau_1, \ldots)$. The largest and most relevant portion of $M_\Phi$’s memory usage is the space needed to store the relations of the extended structures $\bar{D}_0, \bar{D}_1, \ldots$.

We first establish an upper bound on the number of relations that are kept in memory. Recall that the values of $M_\Phi$’s counters $\ell$ and $i$ are at most the length of the processed prefix. Furthermore, we have that $i \leq \ell$, where we identify for readability the counter name with its value. We also observe that $M_\Phi$ stores in each loop iteration only the relations from the extended structures whose indices are between $\max\{0, i - 1\}$.
and $\ell$. Under the assumption that there are at most $m$ consecutive equal time stamps in $\tau$, the difference between $i$ and $\ell$ is bounded by $m \cdot s$, where $s$ is the sum of the upper bounds of the intervals of the future operators occurring in $\Psi$. Hence, the number of relations kept in memory in an iteration by $M_\Phi$ is in $O(m \cdot s \cdot k)$, where $k$ is $\Psi$’s length.

When representing relations by automata, the sizes of these automata are not predictable and we are not aware any upper bounds on their sizes other than non-elementary ones. Furthermore, their sizes depend on how domain elements are represented. Hence we instead focus on the case where relations are finite. A meaningful measurement for the representation size in this case is the cardinality of a relation. For a domain independent subformula $\alpha$ of $\Psi$, we have that every domain element that occurs in $p_\alpha^{D_j}$ is also in $\text{adom} (\overline{D}, \ell)$, where $j \in \mathbb{N}$ with $i \leq j \leq \ell$. It follows that the cardinality of an auxiliary relation for the predicates $p_\alpha$ is polynomially bounded by the cardinality of the active domain at the time point when $M_\Phi$ constructs it, where the degree of the polynomial is the number of the free variables in $\alpha$.

If $\alpha$ is of the form $\beta \cup I \gamma$, then the tuples in $r_\alpha^{D_j}$ and $s_\alpha^{D_j}$ are of the form $(\overline{a}, k)$ and $(\overline{a}, k')$, respectively, where the elements in $\overline{a}$ also occur in the active domain, and $k$ and $k'$ are from the set $\{0, \ldots, \ell - i\}$. We obtain upper bounds, which are larger by a factor of $m \cdot k$ and $(m \cdot k)^2$, respectively, than the polynomial bounds for the auxiliary relations for the predicate $p_\alpha$.

If $\alpha$ is of the form $\beta \cap I \gamma$, with $I = [b, b')$, then the tuples in $r_\alpha^{D_j}$ are of the form $(\overline{a}, y)$, where $y$ is from the set $\mathbb{N}$ if $b' = \infty$ and from the set $\{0, \ldots, b'\}$ if $b' \in \mathbb{N}$. When $b' \in \mathbb{N}$, the cardinality of $r_\alpha^{D_j}$ is at most $|p_\alpha^{D_j}| \cdot (b' + 1)$. To obtain a polynomial upper bound for the case where $b' = \infty$, we must optimize the incremental construction of the auxiliary relations for $r_{\beta \cap I \gamma}$ (see Section 5.2) so that the age of an element is the minimum of its actual age and the interval’s lower bound $b$. The additional factor is then $(b + 1)$.

5.2. Optimizations

In the following, we optimize the memory usage of our monitoring algorithm $M_\Phi$.

**Discarding Relations.** Some of the relations from the extended structures $\overline{D}_0, \overline{D}_1 \ldots$ can be discarded earlier, that is, before executing line 10 of $M_\Phi$ (Figure 2 on page 14) with the respective value of the counter $i$. However, we can only discard the relations that are not used when executing line 7 of $M_\Phi$ in subsequent loop iterations. For instance, if $\alpha$ in line 7 is of the form $\beta \cup I \gamma$, we can discard the auxiliary relations $p_\delta^{D_j}$ with $\delta \in \text{tsub} (\beta) \cup \text{tsub} (\gamma)$ directly after executing line 7. Moreover, if $j > 0$ we can also discard the auxiliary relation $r_\alpha^{D_j-\ell}$. We cannot discard $r_\alpha^{D_j}$ since the incremental construction in Section 3.4.3 uses $r_\alpha^{D_j}$ to build the relation $r_\alpha^{D_j+1}$. Finally, instead of checking in line 8 whether $D_j$ is complete, we check if $i \leq \ell$ and whether each relation has been built for which the corresponding predicate occurs in $\Phi$.

**Improving Incremental Constructions.** To minimize the size of the relations, we can optimize our incremental constructions by removing redundant data tuples from auxiliary relations. For instance, we can optimize the incremental construction for a formula $\alpha = \beta \cup I \gamma$ as follows. If $(\overline{a}, t), (\overline{a}, t') \in r_\alpha^{D_j}$ with $t, t' \in I$ and $t > t'$, then we can remove $(\overline{a}, t)$ from $r_\alpha^{D_j}$. Since $t, t' \in I$, both tuples satisfy the condition of our construction so that $\overline{a}$ is put into the relation $p_\alpha^{D_j}$. Moreover, if the updated version of $(\overline{a}, t)$ is in $r_\alpha^{D_j+1}$, then the updated version of $(\overline{a}, t')$ is also in $r_\alpha^{D_j+1}$, and we have that $t + \tau_{i+1} - \tau_i > t' + \tau_{i+1} - \tau_i$. Again, both updated tuples satisfy the condition such that $\overline{a}$
is put into the relation $p_{i+1}^\beta$. Similar optimizations apply to formulas $\alpha = \beta \cup I \gamma$. There
the auxiliary relations for the predicate $s_\alpha$ may contain redundant elements. Namely,
if $(\bar{a}, j_1, j_1') \in s_\alpha$ with $[j_1, j_1') \subseteq [j_2, j_2')$ then we can remove $(\bar{a}, j_1, j_1')$ from $s_\alpha$.

Another optimization is to tune the incremental constructions for certain kinds of
formulas. For instance, if $\alpha = \square I \gamma$, which is syntactic sugar for $true \cup I \gamma$, then we do
not need the auxiliary relations for $r_\alpha$ at all and instead of storing tuples of the form
$(\bar{a}, j, j')$ in the relations for $s_\alpha$, it suffices to store only $(\bar{a}, j')$. Furthermore, some of the
tuples can be removed. Namely, we can remove the tuple $(\bar{a}, j')$ if there is another tuple
$(\bar{a}, j'')$ in the relation, with $j'' > j'$. This can be seen by an argument similar to the one
we gave when optimizing relations that handle the temporal operator $S_I$.

Simplifying Formulas. The rewriting techniques given for past-only first-order tem-
poral logic by Chomicki and Toman [1995] can be extended to MFOTL. We can thereby
reduce the number of auxiliary relations created from an input formula and also de-
crease their arity. For example, by rewriting the formula $\exists x. \square_I \beta$ to $\square_I \exists x. \beta$
we reduce the arity of the predicates $p_{i+1}^\beta \exists x. \beta$ and $r_{i+1}^\beta \exists x. \beta$ by one; by rewriting the formula $\square_I \circ_I \beta$
to $\beta$ we decrease the number of auxiliary relations.

6. CASE STUDIES
In this section, we demonstrate that MFOTL is well suited for formalizing a wide vari-
ety of security policies including compliance policies and history-based access-control
policies. By evaluating the performance of a prototype implementation of our moni-
toring algorithm, we also demonstrate that monitoring IT systems with respect to such
policies is feasible in practice.

6.1. Formalization of Security Policies
We outline the steps we take when using MFOTL to formalize security policies:

(1) Fix a signature that describes the objects and events that are to be monitored.
(2) Specify the assumptions, if any, on the objects and events that all “well-formed”
systems should satisfy. These assumptions specify basic system requirements that
are prerequisites to formalizing security policies. For example, for systems imple-
menting role-based access control (RBAC) [Ferraiolo et al. 2001], one such well-
formedness assumption is that users can only be assigned to existing roles.
(3) Specify the security policy as formulas $\phi_1, \ldots, \phi_n$ in the MFOTL fragment for which
we can use the monitoring algorithm described in Sections 3 and 4.

The monitors for the formulas $\phi_1, \ldots, \phi_n$ can then be used either online to monitor
events as they occur or offline to read log files and report policy violations.

We illustrate these steps in the remainder of this subsection for three different poli-
cies. In Section 6.2, we report on the monitors’ performance.

6.1.1. Approval Requirements. Recall from Example 2.3 the policy that whenever a busi-
ness report is published, its publication must have been previously approved. The for-
malization $\square I \forall f. publish(f) \Rightarrow \diamond approve(f)$ from Example 2.3 is somewhat simplistic.
In realistic settings, we would also require, for example, that the person who publishes
the report must be an accountant and the person who approves the publication must
be the accountant’s manager. Moreover, the approval must happen within a given time
window, such as at most 10 days before the publication.

Before we give our MFOTL formalization of this refined policy, we point out that
flexible predicates like approving a report and being somebody’s manager are different
in the following respect. The act of approving a report is an event: it happens at a time
point and does not have a duration. In contrast, being someone’s manager describes a
state that has a duration. Since the semantics of MFOTL is point-based, it naturally captures events. Entities like system states have a duration and they do not have a direct counterpart in MFOTL. However, we can model such entities using start and finish events. The following formalization of the above security policy illustrates these two different kinds of entities and how we handle them. To distinguish between them, we use the terms event predicate and state predicate.

**Signature.** The signature consists of the unary predicates \( acc_s \) and \( acc_f \), and the binary predicates \( mgr_s, mgr_f, publish, \) and \( approve \). All of them are flexible predicates. Intuitively speaking, \( mgr_s(m,a) \) marks the time when \( m \) starts being \( a \)'s manager and \( mgr_f(m,a) \) marks the corresponding finishing time. Analogously, \( acc_s(a) \) and \( acc_f(a) \) mark the starting and finishing times when \( a \) is an accountant. With these markers, we can simulate state predicates in MFOTL. For example, the formula \( acc(a) := \neg acc_f(a) \land acc_s(a) \) holds at the time points where \( a \) is an accountant. It states that a starting event for \( a \) being an accountant has previously occurred and the corresponding finishing event has not occurred since then. Analogously, we use the formula \( mgr(m,a) := \neg mgr_f(m,a) \land mgr_s(m,a) \) for the state predicate that \( m \) is \( a \)'s manager.

**Formalization.** Before we formalize the refined approval policy, we formally state the assumptions about the start and finish events in a temporal structure \((\bar{D}, \bar{\tau})\). These assumptions reflect the system requirement that these events are generated in a well-formed way. First, we assume that start and finish events do not occur at the same time point, since their ordering would then be unclear. Formally, for the start and finish events of being an accountant, we assume that \((\bar{D}, \bar{\tau})\) satisfies the formula

\[
\Box \forall a. \neg (acc_s(a) \land acc_f(a)). \tag{A1}
\]

In other words, we require that \( a \) cannot start and stop being an accountant at the same time point. Furthermore, we assume that every finish event is preceded by a matching start event and between two start events there is a finish event. Formally, for the start and finish events of being an accountant, we assume that \((\bar{D}, \bar{\tau})\) satisfies the formulas

\[
\Box \forall a. acc_f(a) \to \Box (\neg acc_f(a) \land acc_s(a)) \tag{A2}
\]

and

\[
\Box \forall a. acc_s(a) \to \neg \Box (\neg acc_f(a) \land acc_s(a)). \tag{A3}
\]

The assumptions for the predicates \( mgr_s \) and \( mgr_f \) are similar and we omit them.

Our formalization of the policy that whenever a report is published, it must be published by an accountant and the report must be approved by her manager within at most 10 time units prior to publication is now given by the formula

\[
\Box \forall a. \forall f. publish(a, f) \to acc(a) \land \Box_{[0, 11]} \exists m. mgr(m, a) \land approve(m, f). \tag{P1}
\]

Note that the state predicates \( acc \) and \( mgr \) can change over time and that such changes are accounted for in our MFOTL formalization of this security policy. In particular, at the time point where \( m \) approves the report \( f \), the formula (P1) requires that \( m \) is \( a \)'s manager. However, \( m \) need no longer be \( a \)'s manager when \( a \) publishes \( f \), although \( a \) must be an accountant at that time point.

**Remark 6.1.** Our approach of formalizing state predicates like \( acc \) and \( mgr \) in MFOTL using start and finish events generalizes to state predicates of any arity. For the sake of brevity, in the following we just introduce the predicate \( p \) of arity \( n \geq 1 \) and implicitly assume that the signature contains the corresponding \( n \)-ary predicates
\(p_a\) and \(p_f\). Moreover, we require that a given temporal structure satisfies the assumptions (A1) to (A3) for \(p\). Finally, we use \(p(x_1, \ldots, x_n)\) as an abbreviation of the formula 
\[
\neg p_f(x_1, \ldots, x_n) \lor p_a(x_1, \ldots, x_n).
\]

Note that under the assumptions (A1) to (A3) the semantics of a syntactically-defined state predicate like being an accountant \((\text{acc}(a) = \neg \text{acc}_l(a) \lor \text{acc}_r(a))\) does not necessarily capture the intuitive meaning of the corresponding state predicate when \(\text{acc}(a)\) occurs in the scope of a temporal operator with metric constraints. For example, consider the formula \(\text{\#}_{[3,4]} \text{acc}(a)\). Recall that \(\text{\#}_{[3,4]} \text{acc}(a)\) not only requires that \(a\) was previously an accountant, say at time point \(j\), it additionally requires that between the current time point \(i\) and the time point \(j\) exactly 3 time units have passed. As a result, even when there was a start event and no finish event for \(a\) being an accountant, the formula \(\text{\#}_{[3,4]} \text{acc}(a)\) is false at the current time point \(i\) for \(a\) when no previous time point \(j\) satisfies the timing constraint \(\tau_i - \tau_j = 3\). To avoid these non-intuitive aspects, we stipulate that a state predicate occurring in the scope of a temporal operator with metric constraints must be relativized by an event predicate [Basin et al. 2012] as, for example, the occurrence of \(\text{mgr}(m, a)\) in the formula (P1) with the event predicate \(\text{approve}(m, f)\).

6.1.2. Transaction Requirements. Our next example is a compliance policy for a banking system that processes customer transactions. The requirements stem from anti-money laundering regulations such as the Bank Secrecy Act [Department of the Treasury 1970] and the USA Patriot Act [107th Congress 2001].

**Signature.** We use the signature \((C, R, \iota)\), with \(C := \{\text{th}\}\), \(R := \{\prec\} \cup F\), \(F\) being the set \(\{\text{trans}, \text{auth}, \text{report}\}\) of flexible predicates, and \(\iota(\prec) := 2\), \(\iota(\text{trans}) := 3\), \(\iota(\text{auth}) := 2\), and \(\iota(\text{report}) := 1\). The ternary predicate \(\text{trans}\) represents the execution of a transaction of some customer transferring a given amount of money. The binary predicate \(\text{auth}\) denotes the authorization of a transaction by some employee. Finally, the unary predicate \(\text{report}\) represents the situation where a transaction is reported as suspicious.

**Formalization.** We assume that the constant \(\text{th}\) is interpreted as some natural number and that the rigid predicate \(\prec\) is interpreted as the standard ordering on the natural numbers.

We first formalize the requirement that executed transactions \(t\) of any customer \(c\) must be reported within at most 5 days if the transferred money \(a\) exceeds a given threshold \(\text{th}\):

\[
\Box \forall c. \forall t. \forall a. \text{trans}(c, t, a) \land \text{th} < a \rightarrow \Diamond_{[0, 6]} \text{report}(t).
\]

(P2)

Moreover, transactions that exceed the threshold must be authorized by some employee \(e\) before they are executed. A formalization of this requirement is given by the formula

\[
\Box \forall c. \forall t. \forall a. \text{trans}(c, t, a) \land \text{th} < a \rightarrow \text{\#}_{[2, 21]} \exists e. \text{auth}(e, t).
\]

(P3)

Here we require that the authorization takes place at least 2 days and at most 20 days before executing the transaction.

Our last requirement concerns the transactions of a customer that has previously made transactions that were classified as suspicious. Namely, every executed transaction \(t\) of a customer \(c\), who has within the last 30 days been involved in a suspicious transaction \(t'\), must be reported as suspicious within 2 days:

\[
\Box \forall c. \forall t. \forall a. \text{trans}(c, t, a) \land (\text{\#}_{[0, 31]} \exists a'. t \neq t' \land \text{trans}(c, t', a') \land \Diamond_{[0, 6]} \text{report}(t')) \rightarrow \\
\Diamond_{[0, 3]} \text{report}(t).
\]

(P4)
6.1.3. Separation of Duty. As a final example, we formalize different types of separation-of-duty (SoD) constraints. SoD is a security principle that aims to prevent fraud and errors by requiring multiple users to be involved in critical processes. SoD constraints are often stated on top of the standard model for role-based access control (RBAC). In a nutshell, RBAC controls access to resources by assigning users to sets of roles, where each role is associated with a set of permissions. A user acquires permissions by being assigned to one or more roles. In the context of RBAC, SoD constraints are usually specified in terms of mutually exclusive roles.

**Signature.** We first describe the signature for formalizing RBAC. It contains unary predicates for the state predicates $U, R, A, O, S$, binary predicates for the state predicates $UA, user, roles$, and a ternary predicate $PA$ for the state predicates $PA$. The unary predicates represent the sets of users $U$, roles $R$, actions $A$, objects $O$, and sessions $S$ in the RBAC system at a given time point. The predicates $UA$ and $PA$ represent the user-assignment relation $UA \subseteq U \times R$ and the permission-assignment relation $PA \subseteq R \times A \times O$ at a given time point. Furthermore, the predicate $user$ indicates a user’s sessions at a time point and $roles$ represents the roles that are active in a session at a time point. All these state predicates are flexible.

In order to formalize different SoD polices, our signature also contains the binary predicate $X$ and the ternary predicate $exec$. The intuitive meaning of these predicates is that $X(r, r')$ holds at those time points when the roles $r$ and $r'$ are mutually exclusive and $exec(s, a, o)$ holds when action $a$ is executed on object $o$ in session $s$. These predicates are also flexible.

**Formalization.** Before we formalize different SoD constraints, we state our assumptions, which reflect system requirements concerning the desired RBAC semantics of the predicates $U, R, A$, and so on. The formula (A4) requires that, at every time point, the predicate $UA$ is correctly typed, namely, it only always relates currently existing users with currently existing roles:

$$\Box \forall u. \forall r. UA(u, r) \rightarrow U(u) \land R(r).$$

(A4)

The formulas that ensure that the other predicates are correctly typed at each time point are similar and we omit them. Formulas (A5) to (A8) state that each running session is associated with exactly one user. In other words, the predicate $user$ represents a function from sessions to users that is constant over a session’s lifetime:

$$\Box \forall s. S_s(s) \rightarrow \exists u. U(u) \land user(s, u).$$

(A5)

$$\Box \forall s. \forall u. \forall u'. user(s, u) \land user(s, u') \rightarrow u \approx u'.$$

(A6)

$$\Box \forall s. \forall u. \forall u'. user(s, u) \land (\bigcirc user(s, u')) \rightarrow u \approx u'.$$

(A7)

and

$$\Box \forall s. \forall u. \forall u'. \neg (user_1(s, u) \land user_2(s, u')).$$

(A8)

The formula (A9) ensures that only those roles may be activated in a session that are presently assigned to the user associated with the session:

$$\Box \forall s. \forall r. roles_s(s, r) \rightarrow \exists u. user(s, u) \land UA(u, r).$$

(A9)

The formula (A10) expresses that actions can only be carried out on objects when the necessary credentials are available:

$$\Box \forall s. \forall a. \forall o. exec(s, a, o) \rightarrow \exists r. roles(s, r) \land PA(r, a, o).$$

(A10)

Finally, we assume that $X$ is irreflexive and symmetric at every time point. We omit the straightforward MFOTL formalization of this assumption.
We now turn to the formalization of the static and dynamic SoD constraints. Static SoD states that no user may be assigned to a pair of roles that are considered mutually exclusive. This is formalized by

$$\Box \forall r \forall r', X(r, r') \rightarrow \neg \exists u. U\bar{A}(u, r) \land U\bar{A}(u, r').$$  

(P5)

**Simple dynamic SoD** states that a user may be a member of any two exclusive roles as long as he does not activate them both in the same session. This is formalized by

$$\Box \forall r \forall r'. X(r, r') \rightarrow \neg \exists s. \text{roles}(s, r) \land (\neg S_f(s) \land \text{roles}(s, r')).$$  

(P6)

Recall that a session is always associated with the same user and that the user remains constant over the session’s lifetime. The formula (P7) formalizes object-based SoD, which states that a user may be a member of any two exclusive roles and may also activate them both within the same session, but he must not act upon the same object through both:

$$\Box \forall r \forall r'. X(r, r') \rightarrow \neg \exists s. \exists o. \big( \exists a. \text{exec}(s, a, o) \land \text{roles}(s, r) \land \text{PA}(r, a, o) \big) \land \big( \neg S_f(s) \land \exists a'. \text{exec}(s, a', o) \land \text{roles}(s, r') \land \text{PA}(r', a', o) \big).$$  

(P7)

This prohibits executing an action on an object whenever the same user has executed another action on the same object associated with a conflicting role in a single session.

6.2. Monitor Performance

In the following, we report on an experimental evaluation of the monitoring algorithm presented in the Sections 3 to 5. In our experiments, we monitored the formulas (P1) to (P4), formalizing the security policies described in the Sections 6.1.1 and 6.1.2, which we evaluated on synthetically generated log files. For the evaluation, we used version 1.1.1 of our monitoring tool MonPoly [Basin et al. 2012], which is an OCaml prototype implementation of the monitoring algorithm for finite relations, and a standard desktop computer with an Intel Core i5 2.67 GHz CPU and 8 GBytes of RAM. MonPoly’s source code and the scripts used in the experiments are publicly available at [http://projects.developer.nokia.com/MonPoly](http://projects.developer.nokia.com/MonPoly).

We assess MonPoly’s performance by carrying out experiments to answer the following three questions. (1) What is MonPoly’s run time and memory consumption with respect to an event rate? (2) What is the maximum event rate that MonPoly can handle online? (3) How does MonPoly’s performance compare with an off-the-shelf database management system (DBMS)?

Before describing the experiments conducted and the results obtained, we make the following two remarks. First, except for the formula (P4), MonPoly’s rewriter automatically obtains monitorable formulas. For (P4), the implemented heuristics fail and we had to manually rewrite the formula to guide MonPoly’s rewriter to obtain a monitorable formula. The second remark is on the generation of the log files. For simplicity, we restrict ourselves to relational structures with singleton relations, that is, there is exactly one event per time point in a generated log file. By default, a generated log file spans over 300 seconds. For the generation, we also fix the event rate, which is the average number of events per second. For each time stamp, the number of events is randomly chosen within ±10% of the fixed event rate. Moreover, the generated log files are such that the number of violations with respect to a policy is on average 5% of the number of events. We populate the log files by generating a stream of publish events, for (P1), and respectively trans events, for (P2) to (P4), with randomly generated parameters. We then generate and correlate the other events in such a way that the event and violation rates are respected.
6.2.1. Resource Consumption with Respect to the Event Rate. Figure 7 shows MonPoly’s resource consumption for formula (P4) for the event rates $1,200$, $1,600$, $2,000$, $2,400$, and $2,800$. For this experiment, we generated for each of these event rates, five log files as described above. The reported values, for each of the event rates, are the average over the five respective log files. We also conducted similar experiments for the other formulas (P1) to (P3). The results are similar to (P4), and are thus omitted.

We observe in the graph of run times (left-hand side of Figure 7) that the time needed to process the logs grows linearly with the time span of the log files, independently of the event rate. This shows that processing a time point does not depend on the size of the log file, but only on the amount of data present in the relevant time window. In our experiments, this amount is constant on average because the event rate is fixed and because the relevant time window also has a fixed size for the formulas (P2) to (P4), as the intervals labeling the temporal operators are bounded. For the formula (P1), even though the formula contains unbounded past operators, the amount of data in the relevant time window does not grow as time progresses because on average the sizes of the accountant and manager relations do not change over time (as for instance, new accountants come and old accountants go). Note that the run time on individual log files deviates from the average at most $5\%$ for the formulas (P2) to (P4) and at most $15\%$ for (P4).

We further observe in the graph of memory usage (right-hand side of Figure 7) that after a start-up phase the memory consumption stabilizes. In particular, as the total number of events grows, memory consumption does not grow.
Table I: Maximal event rate for online monitoring.

<table>
<thead>
<tr>
<th>formula</th>
<th>event rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>(P1)</td>
<td>1,050</td>
</tr>
<tr>
<td>(P2)</td>
<td>16,502</td>
</tr>
<tr>
<td>(P3)</td>
<td>162,140</td>
</tr>
<tr>
<td>(P4)</td>
<td>2,055</td>
</tr>
</tbody>
</table>

Figure 8 shows how MonPoly’s resource consumption varies with respect to the event rate, again for the formula (P4). We remark that the run time grows polynomially and memory consumption grows linearly. This is consistent with the complexity of the atomic operations on relations, in particular intersection, union, and join, which are the ones affected by the change in the event rate. We conducted the same experiments for the formulas (P1) to (P3). The graphs and the observations are similar to those observed for the formula (P4), with the exception of (P1) for which memory consumption is quadratic in the event rate. This behavior is due to size of one the intermediate relations (that is, the evaluation of the subformula \( \forall_{[0,11]} \exists m. \text{mgr}(m,a) \land \text{approve}(m,f) \)) being quadratic in the event rate. Furthermore, for formula (P3) the run times grow linearly with the event rate. This is because the handling of temporal operators labeled by intervals \( I \) with \( 0 \notin I \) is optimized. More precisely, for such operators it is possible to group auxiliary relations by time stamp, instead of by time point, thus iterating through a smaller number of indexes.

6.2.2. Maximal Event Rate for Online Monitoring. As we have seen in the previous subsection, MonPoly’s performance degrades as the event rate increases. In this experiment, we determine the maximal event rate for which the average time used to process one second of logged data is smaller than one second. In other words, we determine the maximal event rate that is less than or equal to the corresponding throughput, where a monitor’s throughput is defined as average number of events it processes in one second. This value thus roughly corresponds to the maximal event rate for which MonPoly can be used online.

We determine this maximal event rate by iteratively increasing the event rate and computing the throughput (as the run time divided by the time span, namely 300 seconds) at each iteration until the throughput is less than the event rate. The event rate is increased by the 50 events per second for (P1), 2,000 for (P2), 5,000 for (P3), and 100 for (P4). Table I lists the obtained event rates for each of the four formulas.

These numbers also show which policies are hard to monitor with MonPoly. As expected, (P4) is slower to monitor than (P2) or (P3) because (P4) contains more temporal operators. Monitoring for the formula (P1) is even slower as it contains the unbounded since operator, and for its evaluation the whole history must be taken into account. In particular, the size of the relevant time windows increases as time passes, while for the other policies it remains the same. The formula (P3) is faster to monitor than (P2) because of the previously mentioned optimization.

We also used MonPoly in a real-world case study [Basin et al. 2011]. There, the analyzed log file contains more than 200 million events (namely 218,778,681) representing logged data of approximately one year (namely 36,507,815 seconds). The average event rate is thus 6, with a peak of 3,964 events per second. Furthermore, there are 14 formulas in this case study, and the formulas’ largest time window is 30 days. Only two of them needed to be manually rewritten for monitoring. For each formula, the log file is processed in less than an hour. While we used MonPoly offline in this case study for reporting the policy violations, it could have been used online, since the lowest
throughput is approximately 60,771 events per second (computed as 218,778,681 events over 1 hour of run time), which is significantly larger than the average event rate.

6.2.3. Comparison with a DBMS. As a final experiment, we compare MonPoly with an off-the-shelf DBMS, namely PostgreSQL version 9.1.4 [PostgreSQL Global Development Group 2012]. For the comparison, we first generate SQL queries that are equivalent to the formulas (P1) to (P4). We then run MonPoly and PostgreSQL on synthetically generated log files and the corresponding databases, respectively.

The translation of MFOTL formulas into SQL queries is performed automatically in two steps. The first step embeds MFOTL into first-order logic. In the second step, first-order formulas are translated into relational algebra expressions, which are then written as SQL queries. The first step is briefly presented in the next paragraph, while the second step is standard [Abiteboul et al. 1995].

The embedding of MFOTL into first-order logic consists of (i) transforming signatures $S = (C, R, i)$ into new signatures $S'$ by increasing the arity of each predicate in $R$ by 2, adding a new predicate $\text{tpts}$ of arity 2, and predicates and function symbols for the standard arithmetic operations like $\leq$ and $-$. (ii) translating temporal structures over $S$ into structures over $S'$, and (iii) translating MFOTL formulas $\phi$ over $S$ into a first-order formulas $\phi'$ over $S'$. Given a temporal structure $(D, \tau)$, we build a structure $M$ with $\text{tpts}^M := \{(i, \tau_i) \mid i \in \mathbb{N}\}$ and $r^M := \{(i, \tau_i, a) \mid i \in \mathbb{N} \text{ and } a \in r^D\}$, for any $r \in R$. The translation of formulas is defined inductively over the formula structure. The translation of formulas whose main connective is not a temporal connective is straightforward, while for temporal formulas we encode the temporal constraints explicitly. For instance, we have $\Diamond_{t,b} \phi := \exists t'. \exists i'. \text{tpts}(i', t') \land i' \leq i \land b \leq t - t' \land t - t' < b' \land \phi$, where $b, b' \in \mathbb{N}$ and the free variables $i$ and $t$ represent the current time point and its time stamp. We thus have that $\bigcup_{i \in \mathbb{N}} \phi^{(D, \tau, i)} = (\exists i. \exists t. \text{tpts}(i, t) \land \phi^M)$. In the experiment, for each generated log file we construct a database. The construction follows the translation of (ii), except that we only consider a finite prefix of a temporal structure of length $\ell \in \mathbb{N}$. By restricting the time points $i \in \mathbb{N}$ to time points with $i < \ell$, we build the structure $M_{\ell,n}$ where the relations for the flexible predicates are finite.

We generate log files with the following event rates: 10 for (P1), 100 for (P4), and 1,000 for (P2) and (P3). For each formula, we iteratively generate a sequence of log files, the first log file having a time span of 300 seconds, and each subsequent log file having a time span twice as large as the previous one. Thus, the number of events in the log file at iteration $i$ is approximately $(300 \cdot 2^i) \cdot r$, where $r$ is the event rate. For each formula, at each iteration, we load the log file into a PostgreSQL database, following the translation described above. We then execute the SQL query obtained as above on this database, and also run MonPoly on the log file. Table II shows each tool's run times in seconds. Note that the run times for PostgreSQL do not include the times for loading a log file into a database.

We observe that MonPoly's run time doubles at each iteration. This behavior corresponds to the one illustrated in Figure 7. For PostgreSQL, the run-time growth rate is constant for a while, after which point the run time explodes. Note that when this explosion occurs, we observe that temporary files are created on disk, indicating that intermediary data does not fit in main memory. For the formulas (P2) and (P3), PostgreSQL is faster up to a point, while for the other two formulas, MonPoly is faster. Furthermore, indexing does not significantly influence PostgreSQL's run times.

In summary, MonPoly performs reasonably well even in an offline setting, where it may outperform a DBMS, in particular for complex policies. In an online setting, one clearly benefits from a specialized approach: after some time, the data processed no longer fits into main memory, which drastically reduces the performance of a DBMS.
Table II: Comparison of PostgreSQL's and MonPoly’s run times (in seconds). The symbol † means that the query execution did not finish within 6 hours.

<table>
<thead>
<tr>
<th>formula</th>
<th>time span</th>
<th>300</th>
<th>600</th>
<th>1200</th>
<th>2400</th>
<th>4800</th>
<th>9600</th>
<th>19200</th>
<th>38400</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>tool</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(P1)</td>
<td>PostgreSQL</td>
<td>12</td>
<td>47</td>
<td>188</td>
<td>747</td>
<td>2,985</td>
<td>11,912</td>
<td>47,727</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>MonPoly</td>
<td>0.1</td>
<td>0.2</td>
<td>0.4</td>
<td>0.8</td>
<td>1.6</td>
<td>3.2</td>
<td>6.3</td>
<td>†</td>
</tr>
<tr>
<td>(P2)</td>
<td>PostgreSQL</td>
<td>0.2</td>
<td>0.5</td>
<td>1.1</td>
<td>2.3</td>
<td>5.9</td>
<td>12</td>
<td>20</td>
<td>†</td>
</tr>
<tr>
<td></td>
<td>MonPoly</td>
<td>2.1</td>
<td>4.3</td>
<td>8.5</td>
<td>17</td>
<td>35</td>
<td>67</td>
<td>140</td>
<td>283</td>
</tr>
<tr>
<td>(P3)</td>
<td>PostgreSQL</td>
<td>0.2</td>
<td>0.6</td>
<td>1.1</td>
<td>2.3</td>
<td>5.9</td>
<td>9.9</td>
<td>22</td>
<td>†</td>
</tr>
<tr>
<td></td>
<td>MonPoly</td>
<td>1.5</td>
<td>3.1</td>
<td>6.1</td>
<td>12</td>
<td>24</td>
<td>48</td>
<td>97</td>
<td>217</td>
</tr>
<tr>
<td>(P4)</td>
<td>PostgreSQL</td>
<td>17</td>
<td>68</td>
<td>271</td>
<td>2,136</td>
<td>†</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MonPoly</td>
<td>0.9</td>
<td>1.7</td>
<td>3.3</td>
<td>6.7</td>
<td>13</td>
<td>26</td>
<td>53</td>
<td>107</td>
</tr>
</tbody>
</table>

7. RELATED WORK

Temporal logics are widely applicable in computing since they allow one to formally and naturally express system properties and handle them algorithmically. For instance, the propositional temporal logics LTL, CTL, and PSL are widely used in system verification, in particular, in model checking [Pnueli 1977; Clarke and Emerson 1982; Vardi 2009]. In the following, we focus on related work on monitoring algorithms that handle temporal logic specifications. We group these works with respect to their application areas.

Program Verification. Monitoring program executions has emerged as a light-weight alternative to software model checking [Havelund and Visser 2002]. Executions are represented as sequences of events obtained via an a priori instrumentation of the program’s source or binary code. The monitors are also often directly instrumented into the code. Many of the developed monitoring algorithms for program verification use a propositional temporal logic for specifying properties. For instance, monitoring algorithms exist for LTL and variants [Giannakopoulou and Havelund 2001; Finkbeiner and Sipma 2004] and for propositional real-time logics [D’Angelo et al. 2005; Bauer et al. 2011]. All these monitoring algorithms are based on either translating formulas into finite-state automata of some kind or on formula rewriting. When using finite-state automata, a monitor updates the automaton’s state when processing an event and it checks for violations depending on the automaton’s current state. When using rewriting, a formula is rewritten according to the current event, resulting in a formula that states the obligations that must be satisfied by the remainder of the execution [Havelund and Roșu 2004; Roșu and Havelund 2005].

Boolean propositions are often too coarse to express relationships between events with data values, in particular when the data values are not known in advance and their number cannot be fixed a priori. Various runtime verification algorithms overcome this limitation by handling specification languages with propositions that have parameters. The parameters are instantiated during an execution with the corresponding event data values. Monitors handling event parameters include EAGLE [Barringer et al. 2004], J-LO [Stolz and Bodden 2006], RuleR [Barringer et al. 2010], LogScope [Barringer et al. 2010], and TraceContract [Barringer and Havelund 2011]. The approach of JavaMOP [Meredith et al. 2012] for checking parametrized properties separates parameter binding from property checking. This leads to a monitoring framework, independent of the specification language used to express (non-parametrized) properties, in which the input trace is sliced at run-time and each slice is monitored independently. The underlying ideas of parametrized monitoring are presented by Roșu and Chen [2012]. A recent development appears in [Barringer et al. 2011].
Monitoring Metric First-order Temporal Properties

[2012], which uses so-called quantified event automata that generalize the parametric trace slicing approach. Here parameters can be explicitly quantified and the quantification ranges over the values that appear in the trace. However, specifying complex policies results in automata with many states and they can be difficult to understand and maintain. It is unclear if there is an equivalent declarative specification language.

In contrast to MFOTL, when writing parametrized properties, variables representing event parameters are free and there is no quantification. Variables are assigned to data values that appear at the current position of the input trace. Such binding has the flavor of variable binding under freeze quantification. This type of quantification was introduced by Alur and Henzinger [1994] for time variables to restrict timing constraints to simple and intuitive temporal patterns. It corresponds to a restricted form of classic first-order quantification. See also [Henzinger 1990]. To illustrate that parametrization is indeed more restrictive than quantification, consider the MFOTL formula $\square \forall x. p(x) \rightarrow \square_{[2,5]} \exists y. q(x, y)$. If $p(a)$ holds at the time point $i$ for some domain element $a$, then for each future time point $j \geq i + 1$ within the specified time window, there must be a domain element $b$ such that $q(a, b)$ holds. However, this domain element need not be the same at each time point $j$. In contrast, when replacing the existential quantifier with the freeze quantifier, or simply omitting it and using the parametrized monitoring semantics, then $y$ is bound to a domain element $b$ such that $q(a, b)$ holds at time point $i + 1$ and that binding does not change when inspecting time points $j > i + 1$. Furthermore, when using freeze quantification, parameter instantiations must be unique at a time point. For instance, it cannot be the case that for $a \neq a'$, both $p(a)$ and $p(a')$ hold at a time point $i$.

**Hardware Verification.** Dedicated monitoring algorithms have also been developed to check the real-time behavior of hardware components, where properties are specified in a real-time temporal logic. We refer to [Basin et al. 2012] for a comparison of the different underlying time models and their impact on monitoring. The restriction to a propositional temporal logic is not a limitation here, since one only needs to reason about Boolean or numeric signal values. In particular, Maler and Nickovic [2012] present an algorithm for monitoring continuous numeric signals, where properties are specified in a real-time logic that extends propositional metric temporal logic with numerical predicates on signal values. Reinbacher et al. [2012] present a specialized monitoring algorithm for discrete hardware systems that admits an efficient hardware realization.

**Security and Audit.** Linear-time temporal logics have been used to formalize regulations and usage-control policies. See, for instance, [Giblin et al. 2005; Zhang et al. 2005; Hilty et al. 2005]. Furthermore, Barth et al. [2006] and Dougherty et al. [2007] suggest using standard automata-based techniques to reason about security policies, in particular, privacy policies and policies with obligations. However, their focus is not on monitoring, but rather on finding appropriate models for expressing security policies.

Monitoring algorithms similar to the ones for program verification have been presented in [Dinesh et al. 2008; Maggi et al. 2011; Baresi et al. 2009; Baader et al. 2009]. The one in [Dinesh et al. 2008] uses a formula-rewriting approach, similar to EAGLE, for checking conformance of traces to regulations, where a regulation can refer to other regulations. Maggi et al. [2011] adapt the automata approach to detect violations of multiple constraints using a single automaton for monitoring the execution of business processes with respect to constraints expressed in LTL. Baresi et al. [2009] adapt the translation from LTL to alternating automata in order to monitor the interaction between web services with regard to properties expressed in a temporal assertion language. Baader et al. [2009] use a translation to Büchi automata to monitor temporal properties expressed in a variant of LTL, where propositions are replaced by axioms in
a description logic to express local properties of states that have a complex structure. Roger and Goubault-Larrecq [2001] present an automata-based monitoring algorithm for intrusion detection. Attack patterns are expressed in a specialized temporal logic with parametrized propositions. Common to all these monitoring algorithms is that properties are specified in a propositional linear-time temporal logic, where propositions are, in some cases, parametrized as previously explained.

In contrast, the monitoring algorithm by Hallé and Villemaire [2012] for monitoring data-aware contracts on XML-based message interactions between web services directly supports existential and universal quantification of variables. However, quantified variables must be guarded and only range over elements that appear at the current position of the input trace. This restriction guarantees that quantified variables range over finitely many data values. To illustrate the restriction imposed by guarded quantification, consider the MFOTL formula $\Box \forall x. p(x) \rightarrow \exists y. \Box_{[2,5]} q(x,y)$. The quantification over $x$ is guarded by the predicate $p(x)$, while the one over $y$ is not guarded. The data values for $y$ are not restricted to the data values that appear at the current time point $i$. The formulas (P1), (P5), and (P6) from Section 6.1 also use unguarded quantification. While we allow unrestricted quantification, for finite relations we require instead that formulas are range-restricted. Furthermore, in [Hallé and Villemaire 2012], quantification is handled algorithmically by explicit variable instantiation. The cost of handling quantification in this way is a polynomial of degree $k$, where $k$ is the maximum number of nested quantifiers. In contrast, in our setting for finite relations, the cost is a polynomial of small degree, depending on the implementation of the relation algebra operators, but not depending on the number of nested quantifiers. Finally, other differences to our monitoring algorithm are that the monitoring algorithm by Hallé and Villemaire [2012] does not handle past operators and future operators need not be bounded. Bauer et al. [2009] present a monitoring algorithm for checking history-based access-control policies, which are expressed in a temporal first-order logic with similar restrictions as the one by Hallé and Villemaire [2012]. In particular, quantifiers must be guarded and are also handled by variable instantiations.

In the context of checking privacy regulations, Garg et al. [2011] consider the problem of auditing incomplete log files, where policies are expressed in a first-order logic with guarded quantification and multiple truth values. The audit is performed by formula rewriting, where the formula obtained after rewriting contains only atoms whose truth value is unknown, due to incomplete data. Their algorithm is not well suited when data needs to be processed online. An adaptation of our monitoring algorithm for finite relations and multiple truth values to cope with incomplete log files, suitable for online monitoring, appears in [Basin et al. 2013].

**Databases.** Different runtime monitoring algorithms have been developed for checking temporal integrity constraints of databases and for specifying temporal database triggers. In fact, our monitoring algorithm shares many similarities with Chomicki’s [1995] monitoring algorithm. Our monitoring algorithm handles a richer specification language than Chomicki’s. For example, our monitoring algorithm supports bounded future operators and, when using automatic structures, no syntactic restrictions on the MFOTL formula to domain-independent queries are necessary. Furthermore, the incremental update constructions for the metric operators are simplified and optimized.

The monitoring algorithm by Lipeck and Saake [1987] relies on formula rewriting in disjunctive normal form and variable instantiations. It is more restrictive than Chomicki’s and ours: temporal operators and quantification cannot be nested and it only supports future operators. The two monitoring algorithms presented in [Sistla
and Wolfson 1995] do not handle the nesting of future and past operators. Their first algorithm handles only future operators and their second one handles only past operators. Furthermore, in both algorithms, quantification of variables is handled similar to parameter instantiation used in the monitoring algorithms for program verification.

**Data-stream Processing and Complex-event Processing.** Data-stream processing is concerned with the online analysis of rapidly evolving data streams, which are time-stamped sequences of relations. Analysis is performed by issuing continuous queries expressed in SQL-like languages [Arasu et al. 2006] extended with constructs for selecting portions of the data streams. Complex-event processing focuses on detecting temporal patterns in event streams, which are time-stamped sequences of tuples. Patterns are usually expressed using formalisms inspired by regular expressions, augmented with features to express event parameters, their correlations, and constraints on the time of event occurrences. Such patterns define so-called complex events from simple ones, and these can in turn be used in other patterns. We refer to [Cugola and Margara 2012] for a survey on data-stream processing and complex-event processing.

Our monitoring algorithm for finite relations can be seen as processing an input data stream, given as a temporal structure, and producing an output data stream, the sequence of satisfying valuations. However, in contrast with related work in stream processing, the specification languages and data and time models used by stream and event processors are not based on temporal logics, which makes a direct comparison difficult. It remains to be seen if and how we can benefit from work in these domains, in order to increase the scope and efficiency of our monitoring algorithm.

8. CONCLUSION

Runtime monitoring (also called runtime verification) has evolved over the past decade from a specialist topic into a field of its own merit with a wide range of algorithms, system integration techniques, and applications. Through examples from the domain of security and compliance, we have illustrated the usefulness of expressive specification languages in general, and MFOTL in particular. We provided a monitoring algorithm for a large safety fragment of MFOTL that handles temporal structures with infinite domains and regular relations. We also specialized it to the important case where the relations that change over time are finite. While the worst-case complexity of monitoring in this case is polynomial space in size of the data appearing in the processed prefix, we show experimentally that processing time and memory usage are moderate. Overall, our results show that MFOTL is an effective specification language for expressing and monitoring a wide variety of practically relevant system properties.

We emphasize though that our approach is not a panacea: there is no one silver bullet that covers all applications and handles all system properties equally well. For instance, policies outside the scope of MFOTL and the presented monitoring algorithm are those that involve aggregation operators like the maximum, sum, and average over a specified time window. Returning to our transaction-processing example from Section 6.1.2, a policy may require that the transactions of any customer must be reported within 5 days, if the customer has cumulatively transferred more than a given amount, say $10,000, within the last 30 days. Arithmetic functions over time windows are either not expressible in MFOTL at all or it is extremely cumbersome. Future work would be to add such features, which are common in stream processing, to MFOTL and to support them in monitoring.

Another area for future work concerns distributed and highly scalable monitoring. Many IT systems are composed of distributed, concurrently executing subsystems. Monitoring their compliance to policies is a major challenge. One fundamental question is how to soundly and effectively distribute the monitoring process for a given
global system property. Since monitors then only observe local system behavior, they may need to communicate with each other or cope with partial knowledge about the system’s global behavior. Another challenge is to scale-up to the amount of data that modern distributed IT systems process, which can be on the order of billions of actions per day or even per hour. To support such enormous quantities of data, large-scale parallelization of monitoring appears necessary.

REFERENCES


Monitoring Metric First-order Temporal Properties


