Monitoring Metric First-order Temporal Properties

DAVID BASIN, ETH Zurich
FELIX KLAEDTKE, NEC Europe Ltd.
SAMUEL MÜLLER, Scandit AG
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 algorithms. 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.2.4 [**Software Engineering**]: Software/Program Verification—*validation, formal methods*; D.2.1 [**Software Engineering**]: Requirements/Specifications—*languages*; D.2.5 [**Software Engineering**]: Testing and Debugging—*monitors, tracing*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*temporal logic*; D.4.6 [**Operating Systems**]: Security and Protection; J.1 [**Computer Applications**]: Administrative Data Processing—*business, law*

General Terms: Security, Theory, Verification

Additional Key Words and Phrases: Runtime Verification, Temporal Databases, Automatic Structures, Security Policies, Compliance Checking

ACM Reference Format:
David Basin, Felix Klaedtke, Samuel Müller, and Eugen Zălinescu. Monitoring Metric First-order Temporal Properties. 201?. *J. ACM* V, N, Article A (January YYYY), 45 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

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

This work was partially done while the second and third authors were at ETH Zurich. We thank the Nokia Research Center, Switzerland for supporting parts of this work.

Authors' addresses: D. Basin and E. Zălinescu, ETH Zurich, Computer Science Department, Institute of Information Security, Universitätstraße 6, 8092 Zurich, Switzerland; F. Klaedtke, NEC Europe Ltd., Kurfürsten-Anlage 36, 69115 Heidelberg, Germany; S. Müller, Scandit AG, Limmatstraße 73, 8005 Zurich, Switzerland.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© YYYY ACM 0004-5411/YYYY/01-ARTA $15.00

DOI:http://dx.doi.org/10.1145/0000000.0000000

such as monitoring properties of safety-critical programs or tracking system events 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 acknowledgment. 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. 2010b; Bauer et al. 2011; Chomicki 1995; Roger and Goubault-Larrecq 2001; Roşu 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.

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 should 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 Φ is bounded, that is, its temporal operators refer only finitely into the future. The standard temporal operator $\Box$ ("generally") requires that Φ must hold at every time point. Temporal past and bounded future operators can be arbitrary nested in Φ. There are also no restrictions on the quantification of variables, which range over an infinite domain, or on the use of negation in Φ. We rely here on finite-state automata as data structures to represent and manipulate infinite but regular sets, for instance, as in [Henriksen et al. 1995] and [Kesten et al. 2001].

In a nutshell, our monitoring algorithm works as follows. Given an MFOTL formula $\Box\,\Phi$ over a signature S, where Φ is bounded, we first transform Φ 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 Φ. The monitoring algorithm then incrementally processes a temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$ over S, which is a sequence $\bar{\mathcal{D}}$ 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 $(\bar{\mathcal{D}}, \bar{\tau})$ that violate Φ. 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 used 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.

We implemented our monitoring algorithm for the two settings just described, regular relations and finite relations, each in a prototype tool, MonPoly-Reg and MonPoly-Fin, respectively. We evaluate both implementations on a number of realistic security policies, evaluating both their ability to monitor the policies and their runtime performance. Our experiments show that regular monitoring with MonPoly-Reg has the advantage that it can handle all formulas in our monitorable safety fragment of MFOTL directly, since there are no restrictions on the use of negation and quantification. In contrast, not every formula in this fragment can be handled by MonPoly-Fin and rewriting formulas, either by hand or by applying heuristics, is sometimes necessary. However, both tools are capable of handling a wide range of realistic policies. Our performance evaluation shows that for monitoring systems that produce large quantities of events, it is advantageous to use MonPoly-Fin. By using more efficient data structures for finite sets, monitoring using finite relations is several orders of magnitude more efficient than working with regular relations. Indeed, MonPoly-Fin's performance provides evidence that monitoring system behavior with respect to complex properties formalized in MFOTL is feasible in practice. Further validation of this hypothesis, where MonPoly-Fin has been applied to industrial case studies with non-synthetic data, is reported in [Basin et al. 2013].

Overall, we see our contributions as follows. First, our monitoring algorithm handles a 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 time and 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.

Parts of the work described here have been previously published in conference proceedings. A simplified account of the monitoring algorithm was first described in [Basin et al. 2008] and the MonPoly-Fin tool was presented in [Basin et al. 2012]. The current article provides full details of the algorithm and proofs, as well as a simpler and more general treatment of the finite relations case. The suitability of MFOTL for formalizing security policies and for monitoring IT systems was demonstrated in [Basin et al. 2010a; 2010b] along with an initial performance analysis. The performance analysis presented here is extended and uses a substantially improved implementation of our monitoring algorithm for the finite relations case and a new implementation with finite-state automata for regular relations.

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 space and time requirements 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 system requirements formalized in MFOTL.

2.1. Syntax and Semantics

Let $\mathbb{I}$ be the set of nonempty intervals over $\mathbb{N}$. We often write an interval in $\mathbb{I}$ as $[b, b') := \{a \in \mathbb{N} \mid b \leq a < b'\}$, where $b \in \mathbb{N}$, $b' \in \mathbb{N} \cup \{\infty\}$, and $b < b'$. A *signature* S is a tuple (C, R, ι), where C is a finite set of constant symbols, R is a finite set of predicates disjoint from C, and the function $\iota : R \to \mathbb{N}$ assigns each predicate $r \in R$ an arity $\iota(r)$. Note that to simplify our account, signatures do not contain function symbols. This is without loss of generality, since for a function of arity $n \geq 1$, we can use an $n+1$-ary predicate to represent its graph. 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 $t_1, \dots, t_{\iota(r)} \in V \cup C$, $r(t_1, \dots, t_{\iota(r)})$ is a formula.
(iii) For $x \in V$, if ϕ and ψ are formulas then $(\neg\phi)$, $(\phi \vee \psi)$, and $(\exists x.\, \phi)$ are formulas.
(iv) For $I \in \mathbb{I}$, if ϕ and ψ are formulas then $(\bullet_I\, \phi)$, $(\bigcirc_I\, \phi)$, $(\phi \;\mathsf{S}_I\; \psi)$, and $(\phi \;\mathsf{U}_I\; \psi)$ are formulas.

The temporal operators $\bullet_I$ ("previous"), $\bigcirc_I$ ("next"), S_I ("since"), and U_I ("until") require the satisfaction of a formula within a particular time interval in the past or future. The subscript I of the operators specifies this time interval. To define their meaning and the semantics of the other connectives 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 $(\bar{\mathcal{D}}, \bar{\tau})$, where $\bar{\mathcal{D}} = (\mathcal{D}_0, \mathcal{D}_1, \dots)$ is a sequence of structures over S and $\bar{\tau} = (\tau_0, \tau_1, \dots)$ is a sequence of non-negative numbers, with the following properties.

1. The sequence $\bar{\tau}$ is monotonically increasing, that is, $\tau_i \leq \tau_{i+1}$, for all $i \geq 0$. Moreover, $\bar{\tau}$ makes progress, that is, for every $\tau \in \mathbb{N}$, there is some index $i \geq 0$ such that $\tau_i > \tau$.
2. $\bar{\mathcal{D}}$ has constant domains, that is, $|\mathcal{D}_i| = |\mathcal{D}_{i+1}|$, for all $i \geq 0$.
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 also call the elements in the sequence $\bar{\tau}$ *time stamps* and the indices of the elements in the sequences $\bar{\mathcal{D}}$ and $\bar{\tau}$ *time points*. Note that successive time points can have identical time stamps. However, by property (1), time cannot decrease and always eventually progresses. Furthermore, the relations $r^{\mathcal{D}_0}, r^{\mathcal{D}_1}, \dots$ in a temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$ corresponding to a predicate symbol $r \in R$ may change over time. In contrast, by properties (2) and (3), the interpretation of the constant symbols $c \in C$ and the domain of the $\mathcal{D}_i$s do not change. We denote them by $c^{\bar{\mathcal{D}}}$ and $|\bar{\mathcal{D}}|$, respectively. Temporal structures have a role for MFOTL similar to the role timed words have for propositional real-time logics like MTL and TPTL [Alur and Henzinger 1992; 1994]. However, instead of having at each time point a set of propositions, we have a structure interpreting the symbols given by the signature.

A *valuation* is a mapping $v : V \to |\bar{\mathcal{D}}|$. For a valuation v, the variable vector $\bar{x} = (x_1, \dots, x_n)$, and $\bar{d} = (d_1, \dots, d_n) \in |\bar{\mathcal{D}}|^n$, we write $v[\bar{x} \mapsto \bar{d}]$ 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 $v(c) = c^{\bar{\mathcal{D}}}$.

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

$$\begin{array}{lll}
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models t \approx t' & \text{iff} & v(t) = v(t') \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models r(t_1, \dots, t_{\iota(r)}) & \text{iff} & \big(v(t_1), \dots, v(t_{\iota(r)})\big) \in r^{\mathcal{D}_i} \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models (\neg\psi) & \text{iff} & (\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \psi \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models (\psi \vee \psi') & \text{iff} & (\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \psi \text{ or } (\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \psi' \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models (\exists x.\, \psi) & \text{iff} & (\bar{\mathcal{D}}, \bar{\tau}, v[x \mapsto d], i) \models \psi, \text{ for some } d \in |\bar{\mathcal{D}}| \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models (\bullet_I\, \psi) & \text{iff} & i > 0,\ \tau_i - \tau_{i-1} \in I, \text{ and } (\bar{\mathcal{D}}, \bar{\tau}, v, i-1) \models \psi \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models (\bigcirc_I\, \psi) & \text{iff} & \tau_{i+1} - \tau_i \in I \text{ and } (\bar{\mathcal{D}}, \bar{\tau}, v, i+1) \models \psi \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models (\psi \,\mathsf{S}_I\, \psi') & \text{iff} & \text{for some } j \leq i,\ \tau_i - \tau_j \in I,\ (\bar{\mathcal{D}}, \bar{\tau}, v, j) \models \psi', \\
 & & \text{and } (\bar{\mathcal{D}}, \bar{\tau}, v, k) \models \psi, \text{ for all } k \in \mathbb{N} \text{ with } j < k \leq i \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models (\psi \,\mathsf{U}_I\, \psi') & \text{iff} & \text{for some } j \geq i,\ \tau_j - \tau_i \in I,\ (\bar{\mathcal{D}}, \bar{\tau}, v, j) \models \psi', \\
 & & \text{and } (\bar{\mathcal{D}}, \bar{\tau}, v, k) \models \psi, \text{ for all } k \in \mathbb{N} \text{ with } i \leq k < j
\end{array}$$

Note that the temporal operators are augmented with intervals and a formula of the form $(\bullet_I\, \phi)$, $(\bigcirc_I\, \phi)$, $(\phi \,\mathsf{S}_I\, \psi)$, or $(\phi \,\mathsf{U}_I\, \psi)$ is only satisfied in $(\bar{\mathcal{D}}, \bar{\tau})$ 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 τ_i. For instance, the formula $\bigcirc_I\, \phi$ is satisfied in a temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$ under valuation v at time point i if the elapsed time to the next time stamp in $\bar{\tau}$ is within the time interval I, that is, $\tau_{i+1} - \tau_i \in I$, and ϕ is satisfied at time point $i + 1$ in $(\bar{\mathcal{D}}, \bar{\tau})$ under v.

2.2. Terminology and Notation

We will make use of the following terminology and notation, most of which is standard; see, for example, the introductory textbook by Enderton [1972].

We denote the set of *free* variables in a formula ϕ by $\mathit{free}(\phi)$. To fix the ordering of the free variables in ϕ, we also say that ϕ has the vector of free variables $\bar{x} = (x_1, \dots, x_n)$, where $\mathit{free}(\phi) = \{x_1, \dots, x_n\}$. We call formulas of the form $t \approx t'$ and $r(t_1, \dots, t_{\iota(r)})$ *atomic*, and formulas with no temporal operators *first-order*. A formula ϕ is *bounded* if the interval I of every temporal operator U_I occurring in ϕ is finite. Likewise, we call the temporal operator U_I *bounded* if I 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 ϕ, we define the set of ϕ's *top-level* temporal subformulas as

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

For example, for $\phi := (\bullet_{[0,\infty)}\,\alpha)\vee((\bigcirc\,\beta)\mathsf{S}_{[1,9)}\gamma)$, we have $tsub(\phi) = \{(\bullet_{[0,\infty)}\,\alpha), ((\bigcirc\,\beta)\mathsf{S}_{[1,9)}\gamma)\}$. The set of *direct* subformulas of ϕ is defined as

$$dsub(\phi) := \begin{cases} \{\psi\} & \text{if } \phi = (\neg\psi),\ \phi = (\exists x.\,\psi),\ \phi = (\bullet_I\,\psi), \text{ or } \phi = (\bigcirc_I\,\psi),\\ \{\psi,\psi'\} & \text{if } \phi = (\psi \vee \psi'),\ \phi = (\psi\ \mathsf{S}_I\ \psi'), \text{ or } \phi = (\psi\ \mathsf{U}_I\ \psi'),\\ \emptyset & \text{otherwise.} \end{cases}$$

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

$$\phi^{(\bar{\mathcal{D}},\bar{\tau},i)} := \left\{\bar{d} \in |\bar{\mathcal{D}}|^n \;\middle|\; (\bar{\mathcal{D}}, \bar{\tau}, v[\bar{x} \mapsto \bar{d}], i) \models \phi, \text{ for some valuation } v\right\}.$$

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

As syntactic sugar, we use standard Boolean connectives like $(\phi\wedge\psi) := (\neg((\neg\phi)\vee(\neg\psi)))$ and $(\phi \to \psi) := ((\neg\phi) \vee \psi)$, the universal quantifier $(\forall x.\,\phi) := (\neg(\exists x.\,\neg\phi))$, and the temporal operators “once” $(\blacklozenge_I\,\phi) := (\mathit{true}\ \mathsf{S}_I\ \phi)$, “historically” $(\blacksquare_I\,\phi) := (\neg(\blacklozenge_I(\neg\phi)))$, “sometimes” $(\Diamond_I\,\phi) := (\mathit{true}\ \mathsf{U}_I\ \phi)$, and “always” $(\Box_I\,\phi) := (\neg(\Diamond_I(\neg\phi)))$, where $I \in \mathbb{I}$ and true is some valid formula with no free variables, for instance, $\exists x.\, x \approx x$. Non-metric variants of the temporal operators are easily defined, for example, $(\bullet\,\phi) := (\bullet_{[0,\infty)}\,\phi)$ and $(\Box\,\phi) := (\Box_{[0,\infty)}\,\phi)$. We use standard conventions concerning operators' binding strength to omit parentheses. For example, $\neg$ binds stronger than $\wedge$, which binds stronger than $\vee$, which in turn binds stronger than $\exists$. Moreover, Boolean operators bind stronger than temporal ones.

2.3. Examples

Before presenting our monitoring algorithm, 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, 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 $(\bar{\mathcal{D}}, \bar{\tau})$ with $\bar{\mathcal{D}} = (\mathcal{D}_0, \mathcal{D}_1, \dots)$, a sequence of logged publishing and approval events, and $\bar{\tau} = (\tau_0, \tau_1, \dots)$, a sequence of time stamps, is as follows. The predicates in $\bar{\mathcal{D}}$'s signature are $\mathit{publish}$ and $\mathit{approve}$, both of arity 1. The domain of $\bar{\mathcal{D}}$ 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 $|\bar{\mathcal{D}}|$ equals $\mathbb{N}$. The ith structure in $\bar{\mathcal{D}}$ is time-stamped with τ_i and contains the relations $PUBLISH_i$ and $APPROVE_i$.

We express the policy by the MFOTL formula

$$\Box\,\forall f.\, \mathit{publish}(f) \to \blacklozenge\, \mathit{approve}(f)\,.$$

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

$$\square \forall f.\, \mathit{publish}(f) \rightarrow \blacklozenge_{[0,11)}\, \mathit{approve}(f)\,.$$

Note that in this last formula we speak of *time units* when measuring the time difference $\tau_j - \tau_i$ between the time stamps τ_i and τ_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 illustrate 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 $\square \forall x.\, \mathit{in}(x) \rightarrow \lozenge_{[0,6)}\, \mathit{out}(x)$. 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 $\square(\neg(\bullet\, \mathit{true}) \rightarrow v(0)) \wedge (\exists i.\, v(i) \wedge i \prec 5 \rightarrow \bigcirc\, v(i+1)) \wedge (v(5) \rightarrow \bigcirc\, v(5))$. We assume that the relations for the predicate v 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 system 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 to 3.6.

3.1. Restrictions

Let $(\bar{\mathcal{D}}, \bar{\tau})$ be a temporal structure over the signature $S = (C, R, \iota)$ and let Ψ be the formula expressing the property to be monitored. We make the following restrictions.

First, we require Ψ to be of the form $\square\, \Phi$, where Φ is bounded. It follows that Ψ 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 ω-regular safety property can be expressed as a formula $\square\, \beta$, where β contains only past operators [Lichtenstein et al. 1985]. This restricted form allows us to check iteratively, at each time point, whether the specified property Φ is violated or satisfied. Furthermore, only finitely many time points must be considered when making each of these checks. Without this syntactic restriction, it is undecidable whether an MFOTL formula is violated at a time point, even when assuming that the formula describes a safety property [Chomicki and Niwiński 1995].

Second, we require that each structure in $\bar{\mathcal{D}}$ is automatic [Khoussainov and Nerode 1995] and each time stamp in $\bar{\tau}$ is a non-negative integer. This allows us to represent each structure by a finite collection of finite-state automata over finite words and each time stamp by a finite word. Note that the time stamps originate from a physical clock, which has limited precision. Using a dense time domain like the non-negative rationals would be unrealistic for monitoring.

Due to the closure properties of regular languages and Φ's boundedness restriction, we can compute finite-state automata representing the sets of satisfying valuations of Φ's subformulas at a time point, given additional restrictions concerning the representations of the structures' domains and the interpretations of the constants. Before introducing these additional restrictions, 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 Σ be a finite alphabet and $\#$ a symbol not in Σ. The *convolution* of the words $w_1, \dots, w_k \in \Sigma^*$, where $w_i = w_{i1} \cdots w_{i\ell_i}$ for each i with $1 \leq i \leq k$, is the word

$$w_1 \otimes \cdots \otimes w_k := \begin{bmatrix} w'_{11} \\ \vdots \\ w'_{k1} \end{bmatrix} \cdots \begin{bmatrix} w'_{1\ell} \\ \vdots \\ w'_{k\ell} \end{bmatrix} \in \left((\Sigma \cup \{\#\})^k \right)^*,$$

where $\ell = \max\{\ell_1, \dots, \ell_k\}$ 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. We use convolutions of words to encode tuples of domain elements, where each of the given words represents a domain element.

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

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

(ii) An *automatic representation* of the automatic structure $\mathcal{A}$ consists of (1) the function $\nu : \mathcal{L}_{|\mathcal{A}|} \to |\mathcal{A}|$, (2) a family of words $(w_c)_{c \in C}$ with $w_c \in \mathcal{L}_{|\mathcal{A}|}$ and $\nu(w_c) = c^{\mathcal{A}}$, for all $c \in C$, and (3) automata $\mathbf{A}_{|\mathcal{A}|}$, $\mathbf{A}_{\approx}$, and $\mathbf{A}_r$, for $r \in R$, that recognize the languages $\mathcal{L}_{|\mathcal{A}|}$, $\mathcal{L}_{\approx}$, and $\mathcal{L}_r$, for $r \in R$, respectively.

(iii) Given an automatic representation of $\mathcal{A}$, a relation $A \subseteq |\mathcal{A}|^k$ is *regular* if the language $\{u_1 \otimes \cdots \otimes u_k \mid u_1, \dots, u_k \in \mathcal{L}_{|\mathcal{A}|} \text{ with } (\nu(u_1), \dots, \nu(u_k)) \in A\}$ is regular.

Note that in the above Definition 3.1(ii), the automata $\mathbf{A}_{\approx}$ and $\mathbf{A}_r$, for $r \in R$, read the components of the convolution of a representative of an element $\bar{a} \in |\mathcal{A}|^k$ synchronously. In the following, we assume that for an automatic structure, we always have an automatic representation for it at hand.

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

Finally, we assume that $|\bar{\mathcal{D}}| = \mathbb{N}$ 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 ν is injective, that is, every element in $|\bar{\mathcal{D}}|$ has only one representative in $\mathcal{L}_{|\bar{\mathcal{D}}|}$, see Lemma 3.2 below. Furthermore, note that every automatic structure has an automatic representation in which the function ν is injective [Khoussainov and Nerode 1995].

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

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

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

Remark 3.3. We state some properties of automatic structures that we need later. First, for a first-order formula ϕ and an automatic structure $\mathcal{A}$, we can effectively construct an automaton that represents the set $\phi^{\mathcal{A}}$. This follows from the closure properties of regular languages and hence $\phi^{\mathcal{A}}$ is regular. Second, some basic arithmetic operations are first-order definable in the structure $(\mathbb{N}, <)$ and thus regular. In particular, the successor relation $\mathrm{succ} := \{(x, y) \in \mathbb{N}^2 \mid y = x + 1\}$ is regular, since the formula $x \prec y \wedge \neg\exists z.\, x \prec z \wedge z \prec y$ 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 $d \in \mathbb{N}$.

3.2. Overview of the Monitoring Algorithm

In the remainder of this section, let $(\bar{\mathcal{D}}, \bar{\tau})$ 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 $\Box\,\Phi$ over $(\bar{\mathcal{D}}, \bar{\tau})$, we incrementally build a sequence of structures $\hat{\mathcal{D}}_0, \hat{\mathcal{D}}_1, \ldots$ over an extended signature $\hat{S}$. The extension depends on the temporal subformulas of Φ. For each time point i, we determine the elements that violate Φ by evaluating a transformed, first-order formula $\neg\hat{\Phi}$ over $\hat{\mathcal{D}}_i$. 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 passed.

In the following, we first describe in Section 3.3 how we extend S and transform Φ. Afterwards, we explain in Section 3.4 how we incrementally build the relations of the extended structures $\hat{\mathcal{D}}_i$. In Section 3.5, we give an example and 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_ϕ for each temporal subformula ϕ of Φ. For subformulas of the form $\beta\,\mathsf{S}_I\,\gamma$ and $\beta\,\mathsf{U}_I\,\gamma$, we introduce additional auxiliary predicates, which store information that allows us to incrementally update the auxiliary relations. Specifically, let $\hat{S} := (\hat{C}, \hat{R}, \hat{\iota})$ be the signature with $\hat{C} := C$ and

$$\begin{aligned}\hat{R} := R \cup {} & \{p_\phi \mid \phi \text{ is a temporal subformula of } \Phi\} \cup \\ & \{r_\phi \mid \phi \text{ is a temporal subformula of } \Phi \text{ with main connective } \mathsf{S}_I \text{ or } \mathsf{U}_I\} \cup \\ & \{s_\phi \mid \phi \text{ is a temporal subformula of } \Phi \text{ with main connective } \mathsf{U}_I\}\,,\end{aligned}$$

where $p_\phi, r_\phi, s_\phi \notin C \cup R \cup V$. The arities of the predicates in $\hat{R}$ are as follows: For a predicate $r \in R$, let $\hat{\iota}(r) := \iota(r)$. If ϕ is a temporal subformula of Φ with n free variables, then $\hat{\iota}(p_\phi) := n$, and $\hat{\iota}(r_\phi) := n + 1$ and $\hat{\iota}(s_\phi) := n + 3$, if r_ϕ and s_ϕ exist.

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

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

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

LEMMA 3.4. *Let $\hat{\mathcal{D}}_0, \hat{\mathcal{D}}_1, \dots$ be structures over the signature $\hat{S}$ that extend the $\mathcal{D}_i$s, that is, $|\hat{\mathcal{D}}_i| = |\mathcal{D}_i|$, $c^{\hat{\mathcal{D}}_i} = c^{\mathcal{D}_i}$, and $r^{\hat{\mathcal{D}}_i} = r^{\mathcal{D}_i}$, for all $c \in C$ and $r \in R$. For every subformula ϕ of Φ and for all $i \in \mathbb{N}$, the following properties hold:*

(i) If $p_\psi^{\hat{\mathcal{D}}_i} = \psi^{(\bar{\mathcal{D}},\bar{\tau},i)}$ for all $\psi \in tsub(\phi)$, then $\hat{\phi}^{\hat{\mathcal{D}}_i} = \phi^{(\bar{\mathcal{D}},\bar{\tau},i)}$.

(ii) If $p_\psi^{\hat{\mathcal{D}}_i}$ is regular for all $\psi \in tsub(\phi)$, then $\hat{\phi}^{\hat{\mathcal{D}}_i}$ is regular.

3.4. Incremental Extended Structure Construction

We now show how to construct the extended structures $\hat{\mathcal{D}}_i$ 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 $c^{\hat{\mathcal{D}}_i} := c^{\mathcal{D}_i}$ and $r^{\hat{\mathcal{D}}_i} := r^{\mathcal{D}_i}$. We present the auxiliary relations for each type of temporal operator separately. Throughout this subsection, let $i \in \mathbb{N}$ and let α be a temporal subformula of Φ. Furthermore, for ease of exposition and without loss of generality, we assume that the direct subformulas of α have the vector $\bar{x} = (x_1, \dots, x_n)$ of free variables.

3.4.1. Previous Operator. If the formula α is of the form $\bullet_I\, \beta$ with $I \in \mathbb{I}$, we define

$$p_\alpha^{\hat{\mathcal{D}}_i} := \begin{cases} \hat{\beta}^{\hat{\mathcal{D}}_{i-1}} & \text{if } i > 0 \text{ and } \tau_i - \tau_{i-1} \in I, \\ \emptyset & \text{otherwise.} \end{cases}$$

Intuitively, a tuple $\bar{a}$ is in $p_\alpha^{\hat{\mathcal{D}}_i}$ if $\bar{a}$ satisfies β 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 $\bullet_I$.

LEMMA 3.5. *Let $\alpha = \bullet_I\, \beta$. The relation $p_\alpha^{\hat{\mathcal{D}}_0}$ is regular and $p_\alpha^{\hat{\mathcal{D}}_0} = \alpha^{(\bar{\mathcal{D}},\bar{\tau},0)} = \emptyset$. For $i > 0$, if the relations $p_\phi^{\hat{\mathcal{D}}_{i-1}}$ are regular and $p_\phi^{\hat{\mathcal{D}}_{i-1}} = \phi^{(\bar{\mathcal{D}},\bar{\tau},i-1)}$ for all $\phi \in tsub(\beta)$, then the relation $p_\alpha^{\hat{\mathcal{D}}_i}$ is regular and $p_\alpha^{\hat{\mathcal{D}}_i} = \alpha^{(\bar{\mathcal{D}},\bar{\tau},i)}$.*

PROOF. For $i = 0$, the lemma obviously holds. For $i > 0$, the regularity of $p_\alpha^{\hat{\mathcal{D}}_i}$ follows from the assumption that the relations $p_\phi^{\hat{\mathcal{D}}_{i-1}}$ 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 $\bullet_I$. □

3.4.2. Next Operator. If the formula α is of the form $\bigcirc_I \beta$ with $I \in \mathbb{I}$, we define

$$p_\alpha^{\hat{\mathfrak{D}}_i} := \begin{cases} \hat{\beta}^{\hat{\mathfrak{D}}_{i+1}} & \text{if } \tau_{i+1} - \tau_i \in I, \\ \emptyset & \text{otherwise.} \end{cases}$$

The following lemma is proved similarly to Lemma 3.5.

LEMMA 3.6. *Let $\alpha = \bigcirc_I \beta$. If the relations $p_\phi^{\hat{\mathfrak{D}}_{i+1}}$ are regular and $p_\phi^{\hat{\mathfrak{D}}_{i+1}} = \phi^{(\bar{\mathfrak{D}},\bar{\tau},i+1)}$ for all $\phi \in tsub(\beta)$, then the relation $p_\alpha^{\hat{\mathfrak{D}}_i}$ is regular and $p_\alpha^{\hat{\mathfrak{D}}_i} = \alpha^{(\bar{\mathfrak{D}},\bar{\tau},i)}$.*

3.4.3. Since Operator. Before we give the construction details for the metric since operator, we first consider its non-metric variant. Let α be a formula of the form $\beta \,\mathsf{S}\, \gamma$. Note that we could directly define the relation $p_\alpha^{\hat{\mathfrak{D}}_i}$ as

$$\bigcup_{j \le i} \big(\hat{\gamma}^{\hat{\mathfrak{D}}_j} \cap \bigcap_{j<k\le i} \hat{\beta}^{\hat{\mathfrak{D}}_{i-k}} \big) .$$

However, this construction has the drawback that at each time point i, we recompute the unions of intersections for $j \le i$. Instead, we use the following construction, which reflects that $\beta \,\mathsf{S}\, \gamma$ is logically equivalent to $\gamma \vee \beta \wedge \bullet(\beta \,\mathsf{S}\, \gamma)$: For $i \ge 0$, we define

$$p_\alpha^{\hat{\mathfrak{D}}_i} := \hat{\gamma}^{\hat{\mathfrak{D}}_i} \cup \begin{cases} \emptyset & \text{if } i = 0, \\ \hat{\beta}^{\hat{\mathfrak{D}}_i} \cap p_\alpha^{\hat{\mathfrak{D}}_{i-1}} & \text{if } i > 0. \end{cases}$$

This construction is *incremental* in the sense that it only depends on the relations in $\hat{\mathfrak{D}}_i$ for which the corresponding predicates occur in the subformulas $\hat{\beta}$ or $\hat{\gamma}$, and on the auxiliary relation $p_\alpha^{\hat{\mathfrak{D}}_{i-1}}$, when $i > 0$. In particular, it does not depend on relations in $\hat{\mathfrak{D}}_j$ for $j < i - 1$.

Now assume that the formula α is of the form $\beta \,\mathsf{S}_I\, \gamma$ 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_α, similar to the above definition for the non-metric case. We define $r_\alpha^{\hat{\mathfrak{D}}_i}$ 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, $r_\alpha^{\hat{\mathfrak{D}}_i} := N \cup U$, where $N := \hat{\gamma}^{\hat{\mathfrak{D}}_i} \times \{0\}$, $U := \emptyset$ if $i = 0$, and for $i > 0$,

$$U := \big\{ (\bar{a}, t) \;\big|\; \bar{a} \in \hat{\beta}^{\hat{\mathfrak{D}}_i},\ t < b',\ \text{and } (\bar{a}, t') \in r_\alpha^{\hat{\mathfrak{D}}_{i-1}} \text{ with } t' = t - \tau_i + \tau_{i-1} \big\} .$$

Intuitively, a pair $(\bar{a}, t)$ is in $r_\alpha^{\hat{\mathfrak{D}}_i}$ if $\bar{a}$ satisfies α at the time point i independent of the lower bound b, where the "age" t indicates how long ago the formula α was satisfied by $\bar{a}$. If $\bar{a}$ satisfies γ at the time point i, it is added to $r_\alpha^{\hat{\mathfrak{D}}_i}$ with the age 0. For $i > 0$, we also update the tuples $(\bar{a}, t) \in r_\alpha^{\hat{\mathfrak{D}}_{i-1}}$ when $\bar{a}$ satisfies β at time point i, that is, the age is adjusted by the difference of the time stamps τ_{i-1} and τ_i in case the new age is less than b'. Otherwise it is too old to satisfy α and the updated tuple is not included in $r_\alpha^{\hat{\mathfrak{D}}_i}$.

Finally, we obtain the auxiliary relation $p_\alpha^{\hat{\mathfrak{D}}_i}$ from $r_\alpha^{\hat{\mathfrak{D}}_i}$ by checking whether the age of a tuple in $r_\alpha^{\hat{\mathfrak{D}}_i}$ is old enough:

$$p_\alpha^{\hat{\mathfrak{D}}_i} := \big\{ \bar{a} \;\big|\; (\bar{a}, t) \in r_\alpha^{\hat{\mathfrak{D}}_i}, \text{for some } t \ge b \big\} .$$

Observe that as in the non-metric case above, the definition of the relation $r_\alpha^{\hat{\mathfrak{D}}_i}$ only depends on the relation $r_\alpha^{\hat{\mathfrak{D}}_{i-1}}$ when $i > 0$, and on the relations in $\hat{\mathfrak{D}}_i$ for which the corresponding predicates occur in the subformulas $\hat{\beta}$ or $\hat{\gamma}$. Furthermore, the arithmetic constraint $t' = t - \tau_i + \tau_{i-1}$ used in the above definition of $r_\alpha^{\hat{\mathfrak{D}}_i}$ for $i > 0$ is first-order definable in $\hat{\mathfrak{D}}_i$ as $\tau_i - \tau_{i-1}$ is a constant value (see Remark 3.3). From this it follows that $r_\alpha^{\hat{\mathfrak{D}}_i}$ is regular and thus also $p_\alpha^{\hat{\mathfrak{D}}_i}$. The details are given in the following lemma.

LEMMA 3.7. *Let $\alpha = \beta \, \mathsf{S}_{[b,b')} \, \gamma$. Under the assumption that the relations $p_\phi^{\hat{\mathfrak{D}}_j}$ are regular and $p_\phi^{\hat{\mathfrak{D}}_j} = \phi^{(\bar{\mathfrak{D}},\bar{\tau},j)}$, for all $j \leq i$ and $\phi \in tsub(\beta) \cup tsub(\gamma)$, the following properties hold:*

(i) The relation $r_\alpha^{\hat{\mathfrak{D}}_i}$ is regular and for all $\bar{a} \in \mathbb{N}^n$ and $t \in \mathbb{N}$,

$$(\bar{a}, t) \in r_\alpha^{\hat{\mathfrak{D}}_i} \quad \textit{iff} \quad \begin{array}{l} \textit{there is a } j \textit{ with } 0 \leq j \leq i \textit{ such that } t = \tau_i - \tau_j < b' \,, \\ \bar{a} \in \gamma^{(\bar{\mathfrak{D}},\bar{\tau},j)} \,, \textit{ and } \bar{a} \in \beta^{(\bar{\mathfrak{D}},\bar{\tau},k)}, \textit{ for all } k \textit{ with } j < k \leq i \,. \end{array}$$

(ii) The relation $p_\alpha^{\hat{\mathfrak{D}}_i}$ is regular and $p_\alpha^{\hat{\mathfrak{D}}_i} = \alpha^{(\bar{\mathfrak{D}},\bar{\tau},i)}$.

PROOF. Property (ii) follows immediately from (i) and the definition of $p_\alpha^{\hat{\mathfrak{D}}_i}$. We prove (i) by induction over i.

Base case $i = 0$: The set $r_\alpha^{\hat{\mathfrak{D}}_0}$ is regular, since it can be defined by the formula

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

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 $r_\alpha^{\hat{\mathfrak{D}}_0}$, from the assumption, and from Lemma 3.4. Note that $\tau_i - \tau_i < b'$, 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 $r_\alpha^{\hat{\mathfrak{D}}_i}$ is regular. Similar to the base case, it follows that the set $N = \hat{\gamma}^{\hat{\mathfrak{D}}_i} \times \{0\}$ is regular. The set $U = \{(\bar{a}, t) \mid \bar{a} \in \hat{\beta}^{\hat{\mathfrak{D}}_i}, \ t < b', \text{ and } (\bar{a}, t') \in r_\alpha^{\hat{\mathfrak{D}}_{i-1}} \text{ with } t' = t - \tau_i + \tau_{i-1}\}$ is also regular. If $b' \neq \infty$, it can be expressed by the formula

$$\psi(\bar{x}, y) := \hat{\beta}(\bar{x}) \wedge y \prec b' \wedge \exists y'. \, \psi'(\bar{x}, y') \wedge y' + (\tau_i - \tau_{i-1}) \approx y \,,$$

where ψ' is the formula that defines $r_\alpha^{\hat{\mathfrak{D}}_{i-1}}$, 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 $y \prec b'$. Since $r_\alpha^{\hat{\mathfrak{D}}_i}$ is defined as the union of N and U, we conclude that $r_\alpha^{\hat{\mathfrak{D}}_i}$ is regular.

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

($\Rightarrow$) If the tuple $(\bar{a}, t)$ is in N, then the conjunct is obviously true. Assume that $(\bar{a}, t) \in U$. By definition, there is a tuple $(\bar{a}, t')$ in $r_\alpha^{\hat{\mathfrak{D}}_{i-1}}$ such that $t' = t - \tau_i + \tau_{i-1}$. By the induction hypothesis, there is an integer j with $0 \leq j \leq i{-}1$ such that $t' = \tau_{i-1} - \tau_j < b'$, $\bar{a} \in \gamma^{(\bar{\mathfrak{D}},\bar{\tau},j)}$, and $\bar{a} \in \beta^{(\bar{\mathfrak{D}},\bar{\tau},k)}$ for all k with $j < k \leq i - 1$. It follows that $t = t' + \tau_i - \tau_{i-1} = \tau_i - \tau_j$. From the assumption, we conclude that $\bar{a} \in \beta^{(\bar{\mathfrak{D}},\bar{\tau},k)}$ for all k with $j < k \leq i$.

($\Leftarrow$) If $j = i$, it follows that $t = 0$. From the assumption and the definition of $r_\alpha^{\hat{\mathfrak{D}}_i}$, it follows that $(\bar{a}, 0) \in r_\alpha^{\hat{\mathfrak{D}}_i}$. Assume that $j < i$. By the induction hypothesis, $(\bar{a}, t') \in r_\alpha^{\hat{\mathfrak{D}}_{i-1}}$ with $t' = t - (\tau_i - \tau_{i-1})$. From the definition of $r_\alpha^{\hat{\mathfrak{D}}_i}$ and the assumption, we conclude that $(\bar{a}, t) \in r_\alpha^{\hat{\mathfrak{D}}_i}$. □

3.4.4. Until Operator. We now address the bounded future-time operator U_I, with $I = [b, b') \in \mathbb{I}$ and $b' \in \mathbb{N}$. Assume that the formula α is of the form $\beta \mathsf{U}_I \gamma$. Let $\ell_i := \max\{j \in \mathbb{N} \mid \tau_{i+j} - \tau_i < b'\}$ be the *lookahead offset* at time point i. From the sequence $\bar{\tau}$, we can determine the lookahead offset ℓ_i by successively inspecting the time stamps $\tau_{i+1}, \tau_{i+2}, \ldots$ until we find a time stamp τ_k with $\tau_k - \tau_i \geq b'$. The lookahead offset is then $k - 1 - i$. This process terminates since, by assumption, U_I is bounded. Also note that this process inspects time points that are in the future with respect to the time point i. For convenience, we additionally define $\ell_{-1} := 0$.

As with the since operator, we could directly define $p_\alpha^{\hat{\mathcal{D}}_i}$ as

$$\bigcup_{\substack{0 \leq j' \leq \ell_i \\ \text{with } \tau_{i+j'} - \tau_i \geq b}} \Big(\hat{\gamma}^{\hat{\mathcal{D}}_{i+j'}} \cap \bigcap_{0 \leq j < j'} \hat{\beta}^{\hat{\mathcal{D}}_{i+j}}\Big).$$

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

Intuitively, the relation $r_\alpha^{\hat{\mathcal{D}}_i}$ contains the tuple $(\bar{a}, j)$ if $\bar{a}$ satisfies β at each of the time points $i + j, \ldots, i + \ell_i$. The relation $s_\alpha^{\hat{\mathcal{D}}_i}$ contains the tuple $(\bar{a}, j, j', t)$ if $j \leq j' \leq \ell_i$, $t = \tau_{i+j'} - \tau_i$, $\bar{a}$ satisfies γ at time point $i + j'$ and β at each of the time points $i + j, \ldots, i + j' - 1$, and the timing constraint $\tau_{i+j'} - \tau_i \geq b$ 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 $r_\alpha^{\hat{\mathcal{D}}_i}$ 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 $i, \ldots, i + \ell_{i-1} - 1$. Formally, these two sets are defined as follows:

$$N_r := \big\{(\bar{a}, j) \bigm| \ell_{i-1} \leq j \leq \ell_i \text{ and } \bar{a} \in \hat{\beta}^{\hat{\mathcal{D}}_{i+k}}, \text{ for all } k \text{ with } j \leq k \leq \ell_i\big\}$$

and $U_r := \emptyset$ if $i = 0$, and

$$U_r := \big\{(\bar{a}, j \ominus 1) \bigm| (\bar{a}, j) \in r_\alpha^{\hat{\mathcal{D}}_{i-1}} \text{ and } (\bar{a}, \ell_{i-1}) \in N_r\big\},$$

for $i > 0$, where $x \ominus y := \max\{0, x - y\}$ is subtraction on the non-negative integers.

Analogously to $r_\alpha^{\hat{\mathcal{D}}_i}$, we define the relation $s_\alpha^{\hat{\mathcal{D}}_i}$ 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 $i + \ell_{i-1}, \ldots, i + \ell_i$. U_s contains the updated data from the time points $i, \ldots, i + \ell_{i-1} - 1$. E_s contains the data from the time points $i, \ldots, i + \ell_{i-1} - 1$ that can be extended to the new time points $i + \ell_{i-1}, \ldots, i + \ell_i$. Formally, we define

$$\begin{aligned} N_s := \big\{(\bar{a}, j, j', t) \bigm| \ell_{i-1} \leq j \leq j' \leq \ell_i,\ \bar{a} \in \hat{\gamma}^{\hat{\mathcal{D}}_{i+j'}},\ t = \tau_{i+j'} - \tau_i, t \geq b, \text{ and} \\ \bar{a} \in \hat{\beta}^{\hat{\mathcal{D}}_{i+k}}, \text{ for all } k \text{ with } j \leq k < j'\big\} \end{aligned}$$

and $U_s := E_s := \emptyset$ if $i = 0$. For $i > 0$, we define

$$U_s := \big\{(\bar{a}, j \ominus 1, j' \ominus 1, t) \bigm| (a, j, j', t') \in s_\alpha^{\hat{\mathcal{D}}_{i-1}}, t = t' - (\tau_i - \tau_{i-1}), \text{ and } t \geq b\big\}$$

and, with the help of $r_\alpha^{\hat{\mathcal{D}}_{i-1}}$ and N_s, we define

$$E_s := \big\{(\bar{a}, j \ominus 1, j', t) \bigm| (\bar{a}, j) \in r_\alpha^{\hat{\mathcal{D}}_{i-1}} \text{ and } (\bar{a}, \ell_{i-1}, j', t) \in N_s\big\}.$$

Finally, with the relation $s_\alpha^{\hat{\mathcal{D}}_i}$ at hand, we define

$$p_\alpha^{\hat{\mathcal{D}}_i} := \big\{\bar{a} \bigm| (\bar{a}, 0, j', t) \in s_\alpha^{\hat{\mathcal{D}}_i}, \text{ for some } j', t \geq 0\big\}.$$

LEMMA 3.8. *Let $\alpha = \beta \mathsf{U}_{[b,b')} \gamma$ with $b' \in \mathbb{N}$. Under the assumption that the relations $p_\phi^{\hat{\mathcal{D}}_k}$ are regular and $p_\phi^{\hat{\mathcal{D}}_k} = \phi^{(\bar{\mathcal{D}},\bar{\tau},k)}$, for all $k \leq i + \ell_i$ and $\phi \in tsub(\beta) \cup tsub(\gamma)$, the following properties hold:*

(i) The relation $r_\alpha^{\hat{\mathcal{D}}_i}$ is regular and for all $\bar{a} \in \mathbb{N}$ and $j \in \mathbb{N}$,

$$(\bar{a}, j) \in r_\alpha^{\hat{\mathcal{D}}_i} \qquad \textit{iff} \qquad \bar{a} \in \beta^{(\bar{\mathcal{D}},\bar{\tau},i+k)}, \textit{ for all } k \textit{ with } j \leq k \leq \ell_i\,.$$

(ii) The relation $s_\alpha^{\hat{\mathcal{D}}_i}$ is regular and for all $\bar{a} \in \mathbb{N}^n$ and $j, j' \in \mathbb{N}$,

$$(\bar{a}, j, j', t) \in s_\alpha^{\hat{\mathcal{D}}_i} \qquad \textit{iff} \qquad \begin{array}{l} j \leq j',\ t = \tau_{i+j'} - \tau_i \in [b, b'),\ \bar{a} \in \gamma^{(\bar{\mathcal{D}},\bar{\tau},i+j')}, \textit{ and} \\ \bar{a} \in \beta^{(\bar{\mathcal{D}},\bar{\tau},i+k)}, \textit{ for all } k \textit{ with } j \leq k < j'\,. \end{array}$$

(iii) The relation $p_\alpha^{\hat{\mathcal{D}}_i}$ is regular and $p_\alpha^{\hat{\mathcal{D}}_i} = \alpha^{(\bar{\mathcal{D}},\bar{\tau},i)}$.

PROOF. Property (iii) follows immediately from (ii) and the definition of $p_\alpha^{\hat{\mathcal{D}}_i}$. 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 $r_\alpha^{\hat{\mathcal{D}}_0} = N_r$. For each j with $0 \leq j \leq \ell_0$, the set of the first components $\bar{a}$ of the tuples $(\bar{a}, j)$ 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 $r_\alpha^{\hat{\mathcal{D}}_i}$ 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. □

3.5. Example

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

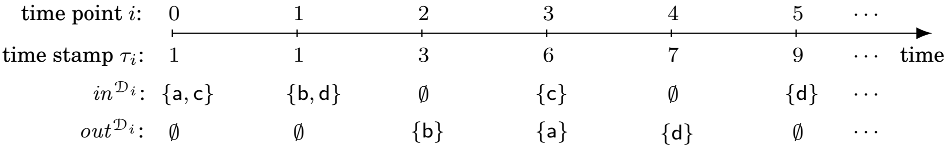
$$\Box\, \forall x.\, in(x) \to \Diamond_{[0,6)}\, out(x)$$

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

Since $\alpha := \Diamond_{[0,6)}\, out(x)$ is the only temporal subformula of Φ, the extended signature $\hat{S}$ contains, in addition to the unary predicates in and out, the unary predicate p_α, the binary predicate r_α, and the ternary predicate s_α. Recall that $\Diamond_{[0,6)}\, out(x)$ is syntactic sugar for $true\ \mathsf{U}_{[0,6)}\, out(x)$. The transformed formula $\hat{\Phi}$ is $\neg in(x) \vee p_\alpha(x)$.

We illustrate the incremental constructions of the auxiliary relations for the temporal formula α by considering the temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$ in Figure 1, where a, b, c, and d are pairwise distinct elements in $|\bar{\mathcal{D}}| = \mathbb{N}$. Since the incremental construction for the temporal operator $\mathsf{U}_{[0,6)}$ 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 $\tau_3 - \tau_0 < 6$ and $\tau_4 - \tau_0 = 6$. The relation $r_\alpha^{\hat{\mathcal{D}}_0}$ is $\mathbb{N} \times \{0, 1, 2, 3\}$ and the relation $s_\alpha^{\hat{\mathcal{D}}_0}$ consists of the pairs (a, j, j', t) with $j \leq j' \leq \ell_0$, $t = \tau_{j'} - \tau_0$, and $a \in out^{\mathcal{D}_{j'}}$, that is, $s_\alpha^{\hat{\mathcal{D}}_0} = \{(\mathsf{b}, 0, 2, 2), (\mathsf{b}, 1, 2, 2), (\mathsf{b}, 2, 2, 2), (\mathsf{a}, 0, 3, 5),$



time point i:	0	1	2	3	4	5	$\cdots$
time stamp τ_i:	1	1	3	6	7	9	$\cdots$ time
$in^{\mathcal{D}_i}$:	$\{\mathsf{a},\mathsf{c}\}$	$\{\mathsf{b},\mathsf{d}\}$	$\emptyset$	$\{\mathsf{c}\}$	$\emptyset$	$\{\mathsf{d}\}$	$\cdots$
$out^{\mathcal{D}_i}$:	$\emptyset$	$\emptyset$	$\{\mathsf{b}\}$	$\{\mathsf{a}\}$	$\{\mathsf{d}\}$	$\emptyset$	$\cdots$

Fig. 1. A temporal structure.

$(\mathsf{a},1,3,5),(\mathsf{a},2,3,5),(\mathsf{a},3,3,5)\}$. The relation $p_\alpha^{\hat{\mathcal{D}}_0}$ is $\{\mathsf{a},\mathsf{b}\}$. When evaluating $\hat{\Phi}$ at time point 0, we obtain $\hat{\Phi}^{\hat{\mathcal{D}}_0} = (\mathbb{N}\setminus in^{\mathcal{D}_0}) \cup p_\alpha^{\hat{\mathcal{D}}_0} = \mathbb{N}\setminus\{\mathsf{c}\}$. The violating elements at time point 0 are therefore $(\neg\hat{\Phi})^{\hat{\mathcal{D}}_0} = \{\mathsf{c}\}$.

At time point 1, the lookahead ℓ_1 is 2. Since $\ell_1 = \ell_0 - 1$, we need not consider any new time points. We obtain $r_\alpha^{\hat{\mathcal{D}}_1}$ and $s_\alpha^{\hat{\mathcal{D}}_1}$ from $r_\alpha^{\hat{\mathcal{D}}_0}$ and $s_\alpha^{\hat{\mathcal{D}}_0}$, respectively, by updating the relative indices and ages in the tuples contained in $r_\alpha^{\hat{\mathcal{D}}_0}$ and $s_\alpha^{\hat{\mathcal{D}}_0}$, yielding $r_\alpha^{\hat{\mathcal{D}}_1} = \mathbb{N}\times\{0,1,2\}$, $s_\alpha^{\hat{\mathcal{D}}_1} = \{(\mathsf{b},0,1,2),(\mathsf{b},1,1,2),(\mathsf{a},0,2,5),(\mathsf{a},1,2,5),(\mathsf{a},2,2,5)\}$, and $p_\alpha^{\hat{\mathcal{D}}_1} = \{\mathsf{a},\mathsf{b}\}$. The set of violating elements at time point 1 is $(\neg\hat{\Phi})^{\hat{\mathcal{D}}_1} = \mathbb{N}\setminus((\mathbb{N}\setminus in^{\mathcal{D}_1}) \cup p_\alpha^{\hat{\mathcal{D}}_1}) = \{\mathsf{d}\}$.

For the time point $i = 2$, we must also account for the new time point 4, since $\ell_2 = 2$. We obtain the relation $r_\alpha^{\hat{\mathcal{D}}_2} = \mathbb{N}\times\{0,1,2\}$ and the relation $s_\alpha^{\hat{\mathcal{D}}_2} = U_s \cup N_s \cup E_s$, with $U_s = \{(\mathsf{b},0,0,0),(\mathsf{a},0,1,3),(\mathsf{a},1,1,3)\}$ by updating the indices and ages of the tuples in $s_\alpha^{\hat{\mathcal{D}}_1}$, and $N_s = \{(\mathsf{d},2,2,4)\}$ and $E_s = \{(\mathsf{d},0,2,4),(\mathsf{d},1,2,4)\}$ by taking the additional structure at time point 4 into account. Furthermore, we get $p_\alpha^{\hat{\mathcal{D}}_2} = \{\mathsf{a},\mathsf{b},\mathsf{d}\}$. The set of violating elements at time point 2 is $(\neg\hat{\Phi})^{\hat{\mathcal{D}}_2} = \mathbb{N}\setminus((\mathbb{N}\setminus in^{\mathcal{D}_2}) \cup p_\alpha^{\hat{\mathcal{D}}_2}) = \emptyset$.

Obviously, the incremental construction for the bounded future operator $\Diamond_I$ can be optimized. In particular, the auxiliary predicate r_α 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 $(\bar{a},j,j',t)$ of the relations for the auxiliary predicate s_α can be ignored.

3.6. Monitoring Algorithm

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

M_Φ uses two counters ℓ and i. The counter ℓ is the index of the current element $(\mathcal{D}_\ell, \tau_\ell)$ in the input sequence $(\mathcal{D}_0,\tau_0),(\mathcal{D}_1,\tau_1),\dots$, which is processed sequentially. Initially, ℓ 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 ℓ's point of view) for which we evaluate $\hat{\Phi}$ over the extended structure $\hat{\mathcal{D}}_i$. The evaluation is delayed until $\hat{\mathcal{D}}_i$ is complete, that is, all the auxiliary relations are built (lines 8–11). Furthermore, M_Φ uses the list[1] Q to ensure that the auxiliary relations of $\hat{\mathcal{D}}_0, \hat{\mathcal{D}}_1, \dots$ 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 α of the structure $\hat{\mathcal{D}}_j$. M_Φ initializes Q in line 3. The function *waitfor* identifies the subformulas

[1] We abuse notation by using set notation for lists. Moreover, we assume that Q is ordered so that (α, j, S) occurs before (α', j', S'), whenever α is a proper subformula of α', or $\alpha = \alpha'$ and $j < j'$.

```
1  $\ell \leftarrow 0$                              % current index in input sequence $(\mathcal{D}_0,\tau_0),(\mathcal{D}_1,\tau_1),\ldots$
2  $i \leftarrow 0$                              % index of next query evaluation in input sequence $(\mathcal{D}_0,\tau_0),(\mathcal{D}_1,\tau_1),\ldots$
3  $Q \leftarrow \{(\alpha, 0, waitfor(\alpha)) \mid \alpha$ is a temporal subformula of $\Phi\}$
4  loop
5      Carry over constants and relations of $\mathcal{D}_\ell$ to $\hat{\mathcal{D}}_\ell$.
6      forall $(\alpha, j, \emptyset) \in Q$ do                        % respect ordering of subformulas
7          Build auxiliary relation $p_\alpha^{\hat{\mathcal{D}}_j}$ and, depending on $\alpha$'s main connective,
               build also the auxiliary relations $r_\alpha^{\hat{\mathcal{D}}_j}$ and $s_\alpha^{\hat{\mathcal{D}}_j}$.
8      while $\hat{\mathcal{D}}_i$ is complete do                                  % evaluate query
9          Output $(\neg\hat{\Phi})^{\hat{\mathcal{D}}_i}$ and $\tau_i$.
10         If $i > 0$, discard $\hat{\mathcal{D}}_{i-1}$.
11         $i \leftarrow i + 1$
12     $Q \leftarrow \{(\alpha, \ell+1, waitfor(\alpha)) \mid \alpha$ is a temporal subformula of $\Phi\} \cup$
           $\{(\alpha, j, \bigcup_{\alpha' \in update(S,\tau_{\ell+1}-\tau_\ell)} waitfor(\alpha')) \mid (\alpha, j, S) \in Q$ and $S \neq \emptyset\}$
13     $\ell \leftarrow \ell + 1$                                % process next element in input sequence
```

Fig. 2. The monitoring algorithm M_Φ.

that delay the formula evaluation:

$$waitfor(\alpha) := \begin{cases} waitfor(\beta) & \text{if } \alpha = \neg\beta,\ \alpha = \exists x.\,\beta, \text{ or } \alpha = \bullet_I\,\beta,\\ waitfor(\beta) \cup waitfor(\gamma) & \text{if } \alpha = \beta \vee \gamma \text{ or } \alpha = \beta\ \mathsf{S}_I\ \gamma,\\ \{\alpha\} & \text{if } \alpha = \bigcirc_I\,\beta \text{ or } \alpha = \beta\ \mathsf{U}_I\ \gamma,\\ \emptyset & \text{otherwise.} \end{cases}$$

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

$$\{\beta \mid \bigcirc_I\,\beta \in U\} \cup \{\beta\ \mathsf{U}_{[\max\{0,b-t\},b'-t)}\ \gamma \mid \beta\ \mathsf{U}_{[b,b')}\ \gamma \in U, \text{ with } b'-t > 0\} \cup$$
$$\{\beta \mid \beta\ \mathsf{U}_{[b,b')}\ \gamma \in U \text{ or } \gamma\ \mathsf{U}_{[b,b')}\ \beta \in U, \text{ with } b'-t \leq 0\}\,.$$

In line 7, we build the auxiliary relations for which enough time has elapsed, that is, the relations for α in $\hat{\mathcal{D}}_j$ 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 $\hat{\mathcal{D}}_i$ have been built, then M_Φ outputs the valuations violating Φ at time point i together with the time stamp τ_i. Furthermore, after each output, the extended structure $\hat{\mathcal{D}}_{i-1}$ is discarded (if $i > 0$) and i is incremented.

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

THEOREM 3.9. *The monitoring algorithm M_Φ has the following properties:*

(i) Whenever M_Φ executes line 9, then the output set is effectively computable, regular, and equal to $(\neg\Phi)^{(\bar{\mathcal{D}},\bar{\tau},i)}$.
(ii) For each $n \in \mathbb{N}$, M_Φ 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_Φ 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 ℓ with $k \in \mathbb{N}$: i_k and ℓ_k denote the counters' values when entering the $(k+1)$st loop iteration. Note that $\ell_k = k$.

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

(1) For all $k \in \mathbb{N}$, we have $(\alpha, k, \mathit{waitfor}(\alpha)) \in Q_k$. This follows directly 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 $\mathit{waitfor}$ and update, and because the sequence of time stamps $\tau_0, \tau_1, \dots$ 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 ℓ have a second component that is at least i_k. Furthermore, we only increment the counter i (line 11) after all relations of $\hat{\mathcal{D}}_{i_k}$ have been built. Also note that, after building the corresponding relations for a tuple in the list Q in line 7, we remove the tuple from Q when updating it in line 12.

From (1) and (2), it follows that for every temporal subformula α of Φ and $j \in \mathbb{N}$, we eventually execute line 7, where we build the auxiliary relations for α of the extended structure $\hat{\mathcal{D}}_j$. 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 α of Φ and all $k, j \in \mathbb{N}$, if $(\alpha, j, \emptyset) \in Q_k$ then line 7 of the monitoring algorithm M_Φ can be executed. That is, the relations involved in the respective incremental construction (depending on α's main connective and given in Section 3.4) of the auxiliary relations for the temporal subformula α have been built earlier and have not yet been discarded. From the respective lemma in Section 3.4, it follows that $p_\alpha^{\hat{\mathcal{D}}_j} = p_\alpha^{(\hat{\mathcal{D}}, \bar{\tau}, j)}$ and $p_\alpha^{\hat{\mathcal{D}}_j}$ 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_Φ are from the extended structure $\hat{\mathcal{D}}_{i_k - 1}$ 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 show that the relations are not built too late. We make a case split on α's main temporal connective, assuming $(\alpha, j, \emptyset) \in Q_k$, for some $j, k \in \mathbb{N}$.

Case $\alpha = \bullet_I\, \beta$. For $j = 0$, there is nothing to prove since $p_\alpha^{\hat{\mathcal{D}}_0} = \emptyset$. For $j > 0$, the construction from Section 3.4.1 of the relation $p_\alpha^{\mathcal{D}_j}$ uses at most the relations $r^{\hat{\mathcal{D}}_{j-1}}$ with $r \in R$ and the auxiliary relations $p_\delta^{\hat{\mathcal{D}}_{j-1}}$ with $\delta \in \mathit{tsub}(\beta)$. The relations for the predicates in R have been carried over to the extended structure $\hat{\mathcal{D}}_{j-1}$ by the execution of line 5 of the monitoring algorithm M_Φ in a previous loop iteration in which the counter ℓ had the value $j-1$.

Assume $\delta \in \mathit{tsub}(\beta)$. 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_Φ puts the tuple $(\delta, j-1, \mathit{waitfor}(\delta))$ into the list Q in the jth loop iteration and the tuple $(\alpha, j, \mathit{waitfor}(\alpha))$ in the $(j+1)$st loop iteration. In each subsequent loop iteration, M_Φ updates the third component of each of these tuples until it becomes the empty set (line 12). By the functions $\mathit{waitfor}$ and update, we have that the third component S of the tuple (α, 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_Φ builds the relation $p_\delta^{\hat{\mathcal{D}}_{j-1}}$ before $p_\alpha^{\hat{\mathcal{D}}_j}$.

Case $\alpha = \beta \mathsf{S}_I \gamma$. The construction from Section 3.4.3 of $p_\alpha^{\hat{\mathcal{D}}_j}$ is only based on the auxiliary relation $r_\alpha^{\hat{\mathcal{D}}_j}$. The given construction of $r_\alpha^{\hat{\mathcal{D}}_j}$ in turn uses at most the relations $r^{\hat{\mathcal{D}}_j}$ with $r \in R$, the auxiliary relations $p_\delta^{\hat{\mathcal{D}}_j}$ with $\delta \in \mathit{tsub}(\beta) \cup \mathit{tsub}(\gamma)$, and the auxiliary relation $r_\alpha^{\hat{\mathcal{D}}_{j-1}}$ if $j > 0$. By an argument similar to the one given above for the temporal operator $\bullet_I$, it follows that the monitoring algorithm M_Φ builds all these relations before building $r_\alpha^{\hat{\mathcal{D}}_j}$.

Case $\alpha = \bigcirc_I \beta$. The construction from Section 3.4.2 of $p_\alpha^{\hat{\mathcal{D}}_j}$ uses at most the relations $r^{\hat{\mathcal{D}}_{j+1}}$ with $r \in R$ and the auxiliary relations $p_\delta^{\hat{\mathcal{D}}_{j+1}}$ with $\delta \in \mathit{tsub}(\beta)$. 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 $(\alpha, j, \mathit{waitfor}(\beta)) \in Q_{j+1}$. It follows that $k \geq j+1$. Thus, the monitoring algorithm M_Φ carries over the relations for the predicates in R to the extended structure $\hat{\mathcal{D}}_{j+1}$ before building the auxiliary relation $p_\alpha^{\hat{\mathcal{D}}_j}$. For $\delta \in \mathit{tsub}(\beta)$, we have that $(\delta, j+1, \mathit{waitfor}(\delta)) \in Q_{j+1}$ with $\mathit{waitfor}(\delta) \subseteq \mathit{waitfor}(\beta)$. Analogously, as in the case for the temporal operator $\bullet_I$, we conclude that M_Φ builds $p_\delta^{\hat{\mathcal{D}}_{j+1}}$ before $p_\alpha^{\hat{\mathcal{D}}_j}$.

Case $\alpha = \beta \mathsf{U}_I \gamma$ *with* $I = [b, b')$. The monitoring algorithm M_Φ postpones the constructions of the auxiliary relations $r_\alpha^{\hat{\mathcal{D}}_j}$, $s_\alpha^{\hat{\mathcal{D}}_j}$, and $p_\alpha^{\hat{\mathcal{D}}_j}$ 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 $\mathit{waitfor}$ 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 $(\alpha, j, \{\beta \mathsf{U}_{I'} \gamma\}) \in Q_{j+k''}$.

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 $r_\alpha^{\hat{\mathcal{D}}_j}$, $s_\alpha^{\hat{\mathcal{D}}_j}$, and $p_\alpha^{\hat{\mathcal{D}}_j}$ have been carried over by the monitoring algorithm M_Φ to the extended structures. Assume $\delta \in \mathit{tsub}(\beta) \cup \mathit{tsub}(\gamma)$. The monitoring algorithm M_Φ postpones the construction of $r_\alpha^{\hat{\mathcal{D}}_j}$, $s_\alpha^{\hat{\mathcal{D}}_j}$, and $p_\alpha^{\hat{\mathcal{D}}_j}$ further until the auxiliary relations $p_\delta^{\hat{\mathcal{D}}_{j+k''}}$, 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 $(\delta, j+k'', \mathit{waitfor}(\delta)) \in Q_{j+k''}$ and $(\alpha, j, \mathit{waitfor}(\beta) \cup \mathit{waitfor}(\gamma)) \in Q_{j+k'}$ and $\mathit{waitfor}(\delta) \subseteq \mathit{waitfor}(\beta) \cup \mathit{waitfor}(\gamma)$. □

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_Φ. This alternative yields a more efficient implementation, as demonstrated by the experimental evaluation in Section 6.3. Furthermore, some of the restrictions in Section 3.1 can even be weakened. However, when representing relations as finite tables, 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_Φ 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 $\Box\, \forall x.\, \mathit{in}(x) \rightarrow \Diamond_{[0,6)}\, \mathit{out}(x)$ from Example 2.4. The relations for the predicates in and 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. This is because the relation interpreting $\mathit{in}(x) \rightarrow \Diamond_{[0,6)}\, \mathit{out}(x)$ is infinite. If we push the negation all the way down to the predicates we obtain $\Phi := \mathit{in}(x) \wedge \Box_{[0,6)}\, \neg \mathit{out}(x)$. Unfortunately, we cannot also use Φ for monitoring, since this time the relation interpreting $\neg \mathit{out}(x)$ is infinite. However, we can monitor the formula $\mathit{in}(x) \wedge \neg \Diamond_{[0,6)}\, \mathit{out}(x)$. The auxiliary relations for the subformula $\Diamond_{[0,6)}\, \mathit{out}(x)$ are always finite and, furthermore, although $\neg \Diamond_{[0,6)}\, \mathit{out}(x)$ describes an infinite set, its conjunction with $\mathit{in}(x)$ guarantees the finiteness of the result. In particular, if I and O are the finite sets of elements that satisfy $\mathit{in}(x)$ and $\Diamond_{[0,6)}\, \mathit{out}(x)$ at a time point $i \in \mathbb{N}$, respectively, then $I \setminus O$ is the set of elements that satisfy $\mathit{in}(x) \wedge \neg \Diamond_{[0,6)}\, \mathit{out}(x)$ at time point i.

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

As a second example, consider

$$\Box\, \forall x.\, \forall y.\, \mathit{in}(x,y) \rightarrow \Diamond_{[0,6)}\, \mathit{out}(x) \wedge (\neg \mathit{out}(y) \vee x \approx y)\,,$$

where in is a binary predicate. The formula states that the first component x of in must eventually be output (within the given time bound) and the second component y must not simultaneously be output if y is different from x. Observe that neither $\Diamond_{[0,6)}\, \mathit{out}(x) \wedge (\neg \mathit{out}(y) \vee x \approx y)$ nor its negation is guaranteed to be fulfilled by only finitely many elements. However, by rewriting, we obtain the formula $\mathit{in}(x,y) \wedge \Box_{[0,6)}\, \big(\neg \mathit{out}(x) \vee \mathit{out}(y) \wedge \neg x \approx y\big) \wedge \blacklozenge_{[0,6)}\, \mathit{in}(x,y)$, which is monitorable.

4.2. Monitorable Fragment

Throughout this section, we fix a signature $S = (C, R, \iota)$, assuming that C is nonempty and true abbreviates a formula $c \approx c$, for some $c \in C$. This technical assumption becomes clear in the following subsections, when we introduce the class of monitorable formulas.

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 $(\bar{\mathcal{D}}, \bar{\tau})$ be a temporal structure with $\bar{\mathcal{D}} = (\mathcal{D}_0, \mathcal{D}_1, \dots)$. We call $(\bar{\mathcal{D}}, \bar{\tau})$ a *temporal database* if (1) the domain $|\bar{\mathcal{D}}|$ is countably infinite, (2) for each $r \in F$ and $i \in \mathbb{N}$, the relation $r^{\mathcal{D}_i}$ is finite, and (3) for each $r \in R \setminus F$ and $i \in \mathbb{N}$, the relation $r^{\mathcal{D}_i}$ is a decidable set and $r^{\mathcal{D}_i} = r^{\mathcal{D}_{i+1}}$. We also assume in the following that $\mathbb{N} \subseteq |\bar{\mathcal{D}}|$ and that there is a binary predicate $\prec$ in $R \setminus F$, which is interpreted as the standard ordering $<$ on $\mathbb{N}$.

Note that we do not fix the domain of a temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$ to $\mathbb{N}$ as done in Section 3.1. The assumption that the time stamps in $\bar{\tau}$ are non-negative integers can also be relaxed, for instance, by assuming that they are non-negative rationals. However, as noted in Section 3.1, a dense time domain is unrealistic for monitoring since the time stamps originate from physical clocks with limited precision. We therefore assume as in Section 3 that the time stamps in $\bar{\tau}$ are non-negative integers.

Furthermore, 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. The finiteness assumption of the flexible predicates allows us to provide the corresponding relations at each time point to the monitoring algorithm by enumerating the relations' elements. Since the relations of a rigid predicate are decidable sets that do not change over time, we assume that the monitoring algorithm has a membership checking procedure at hand. Common examples for the relations of rigid predicates are the graphs of arithmetic operations like addition and subtraction and a relation ordering the domain elements.

Since the monitoring algorithm must compute and store the auxiliary relations, as illustrated in Section 4.1, we impose next additional syntactic restrictions on the monitored formula. These restrictions guarantee the finiteness of the auxiliary relations and allow us to construct them inductively over the formula structure.

4.2.1. Domain Independence. In database theory, the finiteness of the output of queries can be guaranteed 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 $(\bar{\mathcal{D}}, \bar{\tau})$ be a temporal database, with $\bar{\mathcal{D}} = (\mathcal{D}_0, \mathcal{D}_1, \dots)$ and $\bar{\tau} = (\tau_0, \tau_1, \dots)$. We say that $\ell \in \mathbb{N} \cup \{\infty\}$ is a *lookahead* at time point $i \in \mathbb{N}$ for the formula ϕ and $(\bar{\mathcal{D}}, \bar{\tau})$ if $\phi^{(\bar{\mathcal{D}}, \bar{\tau}, i)} = \phi^{(\bar{\mathcal{D}}', \bar{\tau}', i)}$, for all temporal databases $(\bar{\mathcal{D}}', \bar{\tau}')$, with $\mathcal{D}'_k = \mathcal{D}_k$ and $\tau'_k = \tau_k$, for all $k < \ell$. When ϕ is bounded then there is always a lookahead $\ell \in \mathbb{N}$ at i for ϕ and $(\bar{\mathcal{D}}, \bar{\tau})$, since bounded formulas refer only to finitely many time points in the future. For $D \subseteq |\bar{\mathcal{D}}|$, $\models_D$ denotes the relation $\models$ defined in Definition 2.2, except that quantification is relativized to the set D. The *active domain* of $(\bar{\mathcal{D}}, \bar{\tau})$ and $\ell \in \mathbb{N} \cup \{\infty\}$ is

$$\begin{aligned} adom(\bar{\mathcal{D}}, \ell) := {} & \{c^{\bar{\mathcal{D}}} \mid c \in C\} \cup \\ & \bigcup_{r \in F} \bigcup_{0 \leq k < \ell} \big\{d_i \in |\bar{\mathcal{D}}| \bigm| \text{for some } (d_1, \dots, d_{\iota(r)}) \in r^{\mathcal{D}_k} \text{ and } 1 \leq i \leq \iota(r)\big\} . \end{aligned}$$

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

$$\big\{\bar{d} \in D^n \bigm| (\bar{\mathcal{D}}, \bar{\tau}, v[\bar{x} \mapsto \bar{d}], i) \models_D \phi\big\} = \big\{\bar{d} \in D'^n \bigm| (\bar{\mathcal{D}}, \bar{\tau}, v[\bar{x} \mapsto \bar{d}], i) \models_{D'} \phi\big\} ,$$

whenever $adom(\bar{\mathcal{D}}, \ell) \subseteq D, D'$, where $\ell \in \mathbb{N} \cup \{\infty\}$ is a lookahead at i for ϕ and $(\bar{\mathcal{D}}, \bar{\tau})$.

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 restrictions 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 evaluated 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.

$$\frac{}{\alpha : \emptyset}\ \alpha \text{ is an atomic formula} \qquad \frac{\alpha : L}{\alpha : L \cup \{B \to h\}}\ \begin{array}{l}\alpha \text{ is an atomic formula and}\\ B \to h \text{ is admissible for } \alpha\end{array} \qquad \frac{\phi : L}{\phi : L[L^*]}$$

$$\frac{\phi : L}{\neg\phi : \emptyset} \qquad \frac{\phi : L \quad \psi : L'}{\phi \wedge \psi : L \cup L'} \qquad \frac{\phi : L \quad \psi : L'}{\phi \vee \psi : \{B \cup B' \to h \mid B \to h \in L \text{ and } B' \to h \in L'\}}$$

$$\frac{\phi : L}{\exists x.\, \phi : \{B \to h \in L \mid x \notin B \text{ and } x \neq h\}}\ x \in L^* \qquad \frac{\phi : L}{\newcommand{\prevop}{\bullet}\prevop_I\, \phi : L} \qquad \frac{\phi : L}{\bigcirc_I\, \phi : L} \qquad \frac{\phi : L \quad \psi : L'}{\phi\, \mathsf{S}_I\, \psi : L'} \qquad \frac{\phi : L \quad \psi : L'}{\phi\, \mathsf{U}_I\, \psi : L'}$$

Fig. 3. Labeling rules.

$$\dfrac{\dfrac{\mathit{in}(x) : \emptyset}{\mathit{in}(x) : \{\emptyset \to x\}} \quad \dfrac{\dfrac{c \approx c : \emptyset \quad \dfrac{\mathit{out}(x) : \emptyset}{\mathit{out}(x) : \{\emptyset \to x\}}}{c \approx c\, \mathsf{U}_{[0,6)}\, \mathit{out}(x) : \{\emptyset \to x\}}}{\neg\big(c \approx c\, \mathsf{U}_{[0,6)}\, \mathit{out}(x)\big) : \emptyset}}{\mathit{in}(x) \wedge \neg\big(c \approx c\, \mathsf{U}_{[0,6)}\, \mathit{out}(x)\big) : \{\emptyset \to x\}}$$

Fig. 4. Example derivation.

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 connective $\wedge$ as a primitive. We label the subformulas of a formula ϕ, starting with the atomic formulas and propagate these labels to the root of ϕ's syntax tree. A labeling is a set of *restriction facts*, each of the form $B \to h$, with $B \subseteq V$ and $h \in V$. Intuitively, the meaning of $B \to h$ is that if the ranges of the variables in B are restricted, then the range of the variable h is restricted, that is, there are only finitely many possible instantiations of h. Formally, a restriction fact $\{y_1, \dots, y_n\} \to x$ is *admissible* for the formula ϕ if $x, y_1, \dots, y_n \in \mathit{free}(\phi)$ and for every structure $\mathcal{D}$, all finite sets $D_1, \dots, D_n \subseteq |\mathcal{D}|$, and every valuation v with $v(y_1) \in D_1, \dots, v(y_n) \in D_n$, there are only finitely many $d \in |\mathcal{D}|$ with $(\mathcal{D}, v[x \mapsto d]) \models \phi$. We say that a variable x is *range restricted in* ϕ if $\emptyset \to x$ is an admissible restriction fact for ϕ.

The labeling rules are given in Figure 3, which we briefly explain in the following. Atomic formulas are labeled by the empty set. Admissible restriction facts can always be added to a labeling of an atomic formula. We assume that we can determine whether a restriction fact is admissible for an atomic formula. For example, restriction facts of the form $\emptyset \to h$ are admissible for an atomic formula $r(t_1, \dots, t_n)$ if $r \in F$ and $h = t_i$, for some $i \in \mathbb{N}$ with $1 \leq i \leq n$. The restriction fact $\emptyset \to x$ is admissible for $x \approx c$ when c is a constant symbol in C and $\{y\} \to x$ 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 ϕ can be simplified to $L[L^*]$, where $L^* := \{h \mid \emptyset \to h \in L\}$ and $L[X] := \{B \setminus X \to h \mid B \to h \in L\}$, where $X \subseteq V$. For instance, if $L = \{\emptyset \to x, \{x\} \to y\}$, then $L[L^*] = \{\emptyset \to x, \emptyset \to y\}$.

The labeling rule for the Boolean connective $\neg$ removes all restriction facts from the labeling set. For the Boolean connectives $\wedge$ and $\vee$, we combine the labels of the subformulas. The labeling rule for the existential quantifier requires that the quantified variable x is range restricted in the subformula ϕ. The rule propagates all labels in L, except those containing the quantified variable x. The labeling rules for the temporal operators $\bullet_I$ and $\bigcirc_I$ propagate the labels from the operator's subformula. The labeling rules for S_I and U_I only propagate the second subformula's label, as the first subformula just restricts the satisfying valuations of the second subformula.

The derivation in Figure 4 shows that the range of the variable x is restricted in the formula $\mathit{in}(x) \wedge \neg \Diamond_{[0,6)}\, \mathit{out}(x)$ from Section 4.1. Recall that $\Diamond_I\, \phi$ abbreviates $\mathit{true}\ \mathsf{U}_I\, \phi$.

Here true is syntactic sugar for $c \approx c$, for some $c \in C$. We cannot use the formula $\exists x.\, x \approx x$ as abbreviation for true since x is not range restricted in $x \approx x$.

A formula ϕ is *X-range-restricted*, with $X \subseteq \mathit{free}(\phi)$, if there is a derivation tree for $\phi : L$, for some labeling L with $X \subseteq L^*$. If $X = \mathit{free}(\phi)$, we just say that ϕ is range-restricted. Note that the ranges of the quantified variables are restricted in $\emptyset$-range-restricted formulas. Furthermore, the free variables of a range-restricted formula have only finitely many satisfying instantiations.

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

PROOF. To determine whether ϕ is X-range-restricted, we label the leaves of ϕ's syntax tree and propagate these to the root. It is sufficient to consider a maximal labeling L for each atomic subformula α, that is, if $B \to h$ is admissible for atomic formula α then there is a $B' \to h \in L$ with $B' \subseteq B$. Furthermore, we only propagate a labeling L if it is simplified, that is, $L = L[L^*]$. These observations lead to a simple deterministic procedure for deciding whether ϕ is X-range-restricted.

When ϕ is bounded, the finiteness of $\phi^{(\bar{\mathcal{D}},\bar{\tau},i)}$ follows from the invariant that the restriction facts in a derivation tree for a labeling of ϕ are admissible. This invariant is straightforward to show by structural induction.

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

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. Extensions for handling more expressive first-order fragments are, for example, presented by Van Gelder and Topor [1991], which also distinguish between predicates for finite and infinite relations. 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. An inductive evaluation of range-restricted first-order formulas that also include rigid predicates is straightforward and follows the one described in [Abiteboul et al. 1995]. For illustration, consider the formula $p(y) \wedge \exists x.\, q(x,y) \vee x \prec y$, which is range-restricted under the assumption that p and q are flexible predicates. To evaluate this formula, we rewrite it to $p(y) \wedge \exists x.\, q(x,y) \vee p(y) \wedge x \prec y$ to restrict the range of y in the subformula $p(y) \wedge x \prec y$, and hence also in $q(x,y) \vee p(y) \wedge x \prec y$. 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 α 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 $(\bar{\mathcal{D}}, \bar{\tau})$ be a temporal database, with $\bar{\mathcal{D}} = (\mathcal{D}_0, \mathcal{D}_1, \dots)$ and $\bar{\tau} = (\tau_0, \tau_1, \dots)$.

For the incremental construction from Section 3.4.1 for a formula $\alpha = \bullet_I \beta$, we require that $\hat{\beta}$ is range-restricted. Let $\bar{x}$ be the free variables of β and let $I = [b, b')$ with $b \in \mathbb{N}$ and $b' \in \mathbb{N}$. (We omit the case where $b' = \infty$ as it is an obvious adaption.) The construction of the auxiliary relation $p_\alpha^{\hat{\mathfrak{D}}_i}$ is obvious for the time point $i = 0$. For $i > 0$, the range-restricted first-order formula

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

describes the tuples in $p_\alpha^{\hat{\mathfrak{D}}_i}$ 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 $p_\alpha^{\hat{\mathfrak{D}}_i}$ 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 $\alpha = \bigcirc_I \beta$ is done analogously, where we also assume that $\hat{\beta}$ is range-restricted.

For a formula $\alpha = \beta \mathsf{S}_I \gamma$, we require that $\mathit{free}(\beta) \subseteq \mathit{free}(\gamma)$, $\hat{\beta}$ 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 $r_\alpha^{\hat{\mathfrak{D}}_i}$ 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 γ and $I = [b, b')$ with $b \in \mathbb{N}$ and $b' \in \mathbb{N}$. (Again, we omit the case where $b' = \infty$.) The tuples in the relation $r_\alpha^{\hat{\mathfrak{D}}_i}$ are described by the range-restricted first-order formula

$$\big(\hat{\gamma}(\bar{x}) \wedge y \approx 0\big) \vee \big(\exists y'.\, \hat{\beta}(\bar{x}) \wedge r_\alpha(\bar{x}, y') \wedge y \approx y' + \tau_i - \tau_{i-1} \wedge y \prec b'\big)\,,$$

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 $\hat{\beta}(\bar{x}) \wedge r_\alpha(\bar{x}, y')$ is range-restricted, since r_α is a flexible predicate and, by assumption, $\hat{\beta}$ is $\emptyset$-range-restricted and $\mathit{free}(\beta) \subseteq \mathit{free}(\gamma)$. The range-restricted first-order formula $\exists y.\, r_\alpha(\bar{x}, y) \wedge \neg y \prec b$ describes the tuples in the auxiliary relation $p_\alpha^{\hat{\mathfrak{D}}_i}$ where the predicate r_α is interpreted by the relation $r_\alpha^{\hat{\mathfrak{D}}_i}$.

For the incremental construction from Section 3.4.4 for a formula $\alpha = \beta \mathsf{U}_I \gamma$, we require that $\mathit{free}(\beta) \subseteq \mathit{free}(\gamma)$ and that $\hat{\beta}$ and $\hat{\gamma}$ are range-restricted. Similar to the above cases, although more involved, we can describe the auxiliary relations $r_\alpha^{\hat{\mathfrak{D}}_i}$, $s_\alpha^{\hat{\mathfrak{D}}_i}$, and $p_\alpha^{\hat{\mathfrak{D}}_i}$ 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_α, which depend on β and are not restricted by γ. However, for the important case of $\Diamond_I \gamma$, which is syntactic sugar for *true* $\mathsf{U}_I \gamma$, 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_α. See also Section 5.3.

4.2.4. Formula Rewriting. 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 Ψ from the formula $\Box\,\Phi$, where Ψ 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 ψ, $\neg(\psi \vee \psi')$ to $\neg\psi \wedge \neg\psi'$, and $\neg(\psi \wedge \psi')$ to $\neg\psi \vee \neg\psi'$. If we have not succeeded yet, we try to rewrite the formula further by applying the rewrite rules in Figure 5.

$$\begin{array}{lcll}
\alpha \land \bullet_I\, \beta & \longmapsto & \alpha \land \bullet_I (\bigcirc_I\, \alpha) \land \beta & \\
\alpha \land \bigcirc_I\, \beta & \longmapsto & \alpha \land \bigcirc_I (\bullet_I\, \alpha) \land \beta & \\
\alpha \land (\beta\, \mathsf{S}_I\, \gamma) & \longmapsto & \alpha \land (\beta\, \mathsf{S}_I\, (\Diamond_I\, \alpha) \land \gamma) & \text{if } I \text{ is finite} \\
\alpha \land (\beta\, \mathsf{U}_I\, \gamma) & \longmapsto & \alpha \land (\beta\, \mathsf{U}_I\, (\blacklozenge_I\, \alpha) \land \gamma) & \\
\beta\, \mathsf{S}_I\, \gamma \land \alpha & \longmapsto & (\blacklozenge_{[0,b')}\, \alpha) \land \beta\, \mathsf{S}_I\, \gamma \land \alpha & \text{if } I = [b, b') \\
\beta\, \mathsf{U}_I\, \gamma \land \alpha & \longmapsto & (\Diamond_{[0,b')}\, \alpha) \land \beta\, \mathsf{U}_I\, \beta \land \alpha & \text{if } I = [b, b') \\
\alpha \land (\beta \lor \gamma) & \longmapsto & (\alpha \land \beta) \lor (\alpha \land \gamma) & \\
\alpha \land \neg\beta & \longmapsto & \alpha \land \neg(\alpha \land \beta) & \\
\alpha \land \exists x.\, \beta & \longmapsto & \exists x.\, \alpha \land \beta &
\end{array}$$

Fig. 5. Rewrite rules as a heuristic to obtain monitorable formulas.

$$\frac{\phi : L \quad \psi : L'}{\phi\, \mathsf{T}_I\, \psi : L'}\ 0 \in I \qquad \frac{\phi : L \quad \psi : L'}{\phi\, \mathsf{R}_I\, \psi : L'}\ 0 \in I$$

Fig. 6. Additional labeling rules.

These rules aim to push a subformula α inwards, where it is assumed that α restricts the range of variables that are not restricted by β and γ.

Furthermore, we can try to push negation over temporal operators to obtain monitorable formulas. For example, given a subformula $\neg \bigcirc \alpha$, where $\hat{\alpha}$ is not range-restricted, we can first rewrite it to $\bigcirc \neg\alpha$ and then push the negation into α. 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. Recall that $\beta\, \mathsf{T}_I\, \gamma$ and $\beta\, \mathsf{R}_I\, \gamma$ are logically equivalent to $\neg(\neg\beta\, \mathsf{S}_I\, \neg\gamma)$ and $\neg(\neg\beta\, \mathsf{U}_I\, \neg\gamma)$, respectively. 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 $\phi\, \mathsf{T}_I\, \psi$ and $\phi\, \mathsf{R}_I\, \psi$ 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, $\phi\, \mathsf{T}_I\, \psi$ and $\phi\, \mathsf{R}_I\, \psi$ 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 $\bar{\tau} = (\tau_0, \tau_1, \dots)$ of a temporal database $(\bar{\mathcal{D}}, \bar{\tau})$ 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 AND TIME REQUIREMENTS

In this section, we analyze the resource requirements of the monitoring algorithm M_Φ and present optimizations.

5.1. Memory Usage

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

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

When representing relations by automata, the sizes of these automata are not predictable and we are not aware of 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 measure for the representation size in this case is the cardinality of a relation.

We make the assumption that temporal subformulas α of Ψ are domain independent. It is easy to see that every temporal subformula α of Ψ is range-restricted, since Ψ satisfies the restrictions of Section 4.2.3. Thus, from Lemma 4.1, the stated assumption is fulfilled when Ψ contains no rigid predicates. When Ψ contains rigid predicates, the assumption is not always fulfilled. For instance, $\alpha := (x \prec c) \mathsf{S}\, r(x)$, with $r \in F$, is domain independent, while $\beta := \blacklozenge(x \prec c)$ is not, where c is some constant with $c^{\bar{\mathcal{D}}} \in \mathbb{N}$. Furthermore, note that the cardinality of $p_\beta^{\mathcal{D}_0}$ is $c^{\bar{\mathcal{D}}}$ and is independent of the cardinality of the active domain at time point 0.

For a domain independent subformula α of Ψ, we have that every domain element that occurs in $p_\alpha^{\hat{\mathcal{D}}_j}$ is also in $adom(\bar{\mathcal{D}}, \ell)$, where $j \in \mathbb{N}$ with $i \leq j \leq \ell$. It follows that the cardinality of an auxiliary relation for the predicates p_α is polynomially bounded by the cardinality of the active domain at the time point when M_Φ constructs it, where the degree of the polynomial is the number of the free variables in α.

If α is of the form $\beta \,\mathsf{U}_I\, \gamma$, then the tuples in $r_\alpha^{\hat{\mathcal{D}}_j}$ and $s_\alpha^{\hat{\mathcal{D}}_j}$ are of the form $(\bar{a}, j')$ and $(\bar{a}, j', j'', t)$, respectively, where the elements in $\bar{a}$ also occur in the active domain, $j', j'' \in \{0, \dots, \ell - i\}$ and $t \in \{0, \dots, s\}$. We obtain upper bounds on the cardinality of $r_\alpha^{\hat{\mathcal{D}}_j}$ and $s_\alpha^{\hat{\mathcal{D}}_j}$, which are larger by a factor of $(m \cdot s + 1)$ and $(m \cdot s + 1)^2 \cdot (s + 1)$, respectively, than the polynomial bounds for the auxiliary relations for the predicate p_α.

If α is of the form $\beta \,\mathsf{S}_I\, \gamma$, with $I = [b, b')$, then the tuples in $r_\alpha^{\hat{\mathcal{D}}_j}$ are of the form $(\bar{a}, t)$, where t is from the set $\mathbb{N}$ if $b' = \infty$ and from the set $\{0, \dots, b'\}$ if $b' \in \mathbb{N}$. When $b' \in \mathbb{N}$, the cardinality of $r_\alpha^{\hat{\mathcal{D}}_j}$ is at most $|p_\alpha^{\hat{\mathcal{D}}_j}| \cdot (b' + 1)$. To obtain a polynomial upper bound for the case where $b' = \infty$, we must optimize the incremental construction of the auxiliary relations for $r_{\beta \mathsf{S}_{[b,\infty)} \gamma}$ (see Section 5.3) 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. Time Complexity

We complement the upper bounds on the space requirements for M_Φ with upper bounds on the run time of M_Φ during one iteration. As in the previous section, and for the same reasons, we focus on the case where relations are finite and the temporal subformulas α of the given formula Ψ are domain independent. We also use the same notation, namely i, ℓ, m, s, and k are as in Section 5.1. In addition, we denote by n the maximum number of free variables among all of Ψ's subformulas.

We assume that the lines 7 and 9 of M_Φ in Figure 2 are implemented using extended relational algebra operations, namely, set union, set difference, Cartesian product, selection, projection, and natural join [Abiteboul et al. 1995]. More precisely, the formula $\hat{\Psi}$ and the formulas that define the auxiliary relations (see Section 4.2.3) are translated into extended relational algebra expressions before monitoring, and these expressions are evaluated during monitoring, for each time point i. Extended relational algebra expressions cater for the arithmetic and comparisons used in those formulas. The expression for $\hat{\Psi}$ has size $O(k)$, while the expressions for the formulas that define the auxiliary relations have constant size.

An extended relational algebra operation runs in time polynomial in the arity and cardinality of its input relations. This holds even for naive implementations and data structures, such as when implementing relations as lists of tuples. The cardinality of the output relation is at most the product of the cardinality of the input relations.

The relation symbols in the obtained expressions refer to the relations $p^{\mathcal{D}_j}$ with $p \in F$ and $i \leq j \leq \ell$ and the auxiliary relations stored at time point ℓ. The arity of these relations is in $O(n)$. As explained in the previous section, their cardinality is polynomially bounded by the cardinality of the active domain at ℓ and by the parameters m and s. The degree of the polynomial is linear in n. It follows that the evaluation of these expressions at time point ℓ takes polynomial time in cardinality of the active domain at ℓ and in the parameters m and s, with the degree of the polynomial being linear in n and k. Note that the update of Q at line 12 is polynomial in k.

5.3. Optimizations

In the following, we optimize the memory usage of our monitoring algorithm M_Φ.

Discarding Relations. Some of the relations from the extended structures $\hat{\mathcal{D}}_0, \hat{\mathcal{D}}_1 \ldots$ can be discarded earlier, that is, before executing line 10 of M_Φ (Figure 2 on page 16) 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_Φ in subsequent loop iterations. For instance, if α in line 7 is of the form $\beta \, \mathsf{S}_I \, \gamma$, we can discard the auxiliary relations $p_\delta^{\hat{\mathcal{D}}_j}$ with $\delta \in \mathit{tsub}(\beta) \cup \mathit{tsub}(\gamma)$ directly after executing line 7. Moreover, if $j > 0$ we can also discard the auxiliary relation $r_\alpha^{\hat{\mathcal{D}}_{j-1}}$. We cannot discard $r_\alpha^{\hat{\mathcal{D}}_j}$ since the incremental construction in Section 3.4.3 uses $r_\alpha^{\hat{\mathcal{D}}_j}$ to build the relation $r_\alpha^{\hat{\mathcal{D}}_{j+1}}$. Finally, instead of checking in line 8 whether $\hat{\mathcal{D}}_i$ is complete, we check if $i \leq \ell$ and whether each relation has been built for which the corresponding predicate occurs in $\hat{\Phi}$.

Improving Incremental Constructions. To minimize the size of the auxiliary relations, we can optimize our incremental constructions by removing redundant data tuples from these relations. For instance, we can optimize the incremental construction for a formula $\alpha = \beta \, \mathsf{S}_I \, \gamma$ as follows. If $(\bar{a}, t), (\bar{a}, t') \in r_\alpha^{\hat{\mathcal{D}}_i}$ with $t, t' \in I$ and $t > t'$, then we can remove $(\bar{a}, t)$ from $r_\alpha^{\hat{\mathcal{D}}_i}$. Since $t, t' \in I$, both tuples satisfy the condition of our construction so that $\bar{a}$ is put into the relation $p_\alpha^{\hat{\mathcal{D}}_i}$. Moreover, if the updated version of $(\bar{a}, t)$ is in $r_\alpha^{\hat{\mathcal{D}}_{i+1}}$, then the updated version of $(\bar{a}, t')$ is also in $r_\alpha^{\hat{\mathcal{D}}_{i+1}}$, and $t + \tau_{i+1} - \tau_i > t' + \tau_{i+1} - \tau_i$. Again, both updated tuples satisfy the condition such that $\bar{a}$ is put into the relation $p_\alpha^{\hat{\mathcal{D}}_{i+1}}$. Similar optimizations apply to formulas $\alpha = \beta \, \mathsf{U}_I \, \gamma$. There the auxiliary relations for the predicate s_α may contain redundant elements. Namely, if $(\bar{a}, j_1, j_1', t_1), (\bar{a}, j_2, j_2', t_2) \in s_\alpha^{\hat{\mathcal{D}}_i}$ with $[j_1, j_1') \subsetneq [j_2, j_2')$ then we can remove $(\bar{a}, j_1, j_1', t_1)$ from $s_\alpha^{\hat{\mathcal{D}}_i}$.

Another optimization is to tune the incremental constructions for certain kinds of formulas. For instance, if $\alpha = \Diamond_I \, \gamma$, which is syntactic sugar for $\mathit{true} \, \mathsf{U}_I \, \gamma$, then we do not need the auxiliary relations for r_α at all and instead of storing tuples of the form

$(\bar{a}, j, j', t)$ in the relations for s_α, it suffices to store only $(\bar{a}, j', t)$. Furthermore, some of the tuples can be removed. Namely, we can remove the tuple $(\bar{a}, j', t)$ if there is another tuple $(\bar{a}, j'', t')$ in the relation, with $j'' > j'$. This can be seen by an argument similar to the one we gave when optimizing the relations that handle the temporal operator S_I.

Simplifying Formulas. The rewriting techniques given for past-only first-order temporal 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 decrease their arity. For example, by rewriting the formula $\exists x.\, \blacklozenge_I \beta$ to $\blacklozenge_I \exists x.\, \beta$ we reduce the arity of the predicates $p_{\blacklozenge_I \exists x.\, \beta}$ and $r_{\blacklozenge_I \exists x.\, \beta}$ by one. Under certain conditions, formulas containing nested metric operators can also be simplified. For example, if $0 \in I \cap J$ then $\blacklozenge_I \lozenge_J \beta$ can be rewritten to $(\blacklozenge_I \beta) \vee \lozenge_J \beta$, where for the two disjuncts we share the relations for the auxiliary predicates occurring in $\hat{\beta}$.

6. CASE STUDIES

In this section, we demonstrate that MFOTL is well suited for formalizing a wide variety of security policies including compliance policies and history-based access-control policies. We also evaluate the performance of two prototype implementations of our monitoring algorithm for the settings described in Sections 3 and 4. Our evaluation demonstrates that monitoring IT systems with respect to such policies is feasible in practice, especially when using the implementation based on finite relations.

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 implementing 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 $\phi_1, \dots, \phi_n$ in the MFOTL fragment for which we can use the monitoring algorithm described in Sections 3 and 4.

We can then use the monitoring algorithm either online to monitor events as they occur or offline to read log files for detecting and reporting policy violations.

We illustrate these steps in the remainder of this subsection for three different policies. Afterwards we report on the monitors’ performance.

6.1.1. Approval Requirements. Recall from Example 2.3 the policy that whenever a business report is published, its publication must have been previously approved. The formalization $\Box \forall f.\, \mathit{publish}(f) \rightarrow \blacklozenge\, \mathit{approve}(f)$ from Example 2.3 is somewhat simplistic. In practice, 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 that have a duration, like system states, 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, $mgr_{\mathrm{s}}(m,a)$ marks the time when m starts being a's manager and $mgr_{\mathrm{f}}(m,a)$ marks the corresponding finishing time. Analogously, $acc_{\mathrm{s}}(a)$ and $acc_{\mathrm{f}}(a)$ 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 $\underline{acc}(a) := \neg acc_{\mathrm{f}}(a) \mathsf{S} acc_{\mathrm{s}}(a)$ 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 $\underline{mgr}(m,a) := \neg mgr_{\mathrm{f}}(m,a) \mathsf{S} mgr_{\mathrm{s}}(m,a)$ 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 $(\bar{\mathcal{D}},\bar{\tau})$. 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 $(\bar{\mathcal{D}},\bar{\tau})$ satisfies the formula

$$\Box\,\forall a.\, \neg\big(acc_{\mathrm{s}}(a) \wedge acc_{\mathrm{f}}(a)\big)\,. \tag{A1}$$

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 $(\bar{\mathcal{D}},\bar{\tau})$ satisfies the formulas

$$\Box\,\forall a.\, acc_{\mathrm{f}}(a) \rightarrow \newmoon \big(\neg acc_{\mathrm{f}}(a) \mathsf{S}\, acc_{\mathrm{s}}(a)\big) \tag{A2}$$

and

$$\Box\,\forall a.\, acc_{\mathrm{s}}(a) \rightarrow \neg \newmoon \big(\neg acc_{\mathrm{f}}(a) \mathsf{S}\, acc_{\mathrm{s}}(a)\big)\,. \tag{A3}$$

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

$$\Box\,\forall a.\,\forall f.\, publish(a,f) \rightarrow \underline{acc}(a) \wedge \blacklozenge_{[0,11)}\, \exists m.\, \underline{mgr}(m,a) \wedge approve(m,f)\,. \tag{P1}$$

Note that the state predicates $\underline{acc}$ and $\underline{mgr}$ 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 $\underline{acc}$ and $\underline{mgr}$ 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 $\underline{p}$ of arity $n \geq 1$ and implicitly assume that the signature contains the corresponding n-ary predicates p_{s} and p_{f}. Moreover, we require that a given temporal structure satisfies the assumptions (A1) to (A3) for $\underline{p}$. Finally, we use $\underline{p}(x_1,\dots,x_n)$ as an abbreviation of the formula $\neg p_{\mathrm{f}}(x_1,\dots,x_n) \mathsf{S}\, p_{\mathrm{s}}(x_1,\dots,x_n)$.

Note that under the assumptions (A1) to (A3), the semantics of a syntactically-defined state predicate like being an accountant ($\underline{acc}(a) = \neg acc_{\mathrm{f}}(a) \mathsf{S}\, acc_{\mathrm{s}}(a)$) does not necessarily capture the intuitive meaning of the corresponding state predicate when $\underline{acc}(a)$ occurs in the scope of a temporal operator with metric constraints. For example, consider the formula $\blacklozenge_{[3,4)}\, \underline{acc}(a)$. Recall that $\blacklozenge_{[3,4)}\, \underline{acc}(a)$ 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 $\blacklozenge_{[3,4)}\, \underline{acc}(a)$ 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 $\underline{mgr}(m, a)$ in the formula (P1) with the event predicate $\mathit{approve}(m, f)$.

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

Formalization. We assume that the constant 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 th:

$$\Box\, \forall c.\, \forall t.\, \forall a.\, \mathit{trans}(c, t, a) \wedge \mathit{th} \prec a \rightarrow \Diamond_{[0,6)}\, \mathit{report}(t)\,. \qquad \textbf{(P2)}$$

Moreover, transactions that exceed the threshold must be authorized by some employee e before they are executed. We formalize this requirement as the formula

$$\Box\, \forall c.\, \forall t.\, \forall a.\, \mathit{trans}(c, t, a) \wedge \mathit{th} \prec a \rightarrow \blacklozenge_{[2,21)}\, \exists e.\, \mathit{auth}(e, t)\,. \qquad \textbf{(P3)}$$

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:

$$\Box\, \forall c.\, \forall t.\, \forall a.\, \mathit{trans}(c, t, a) \wedge \big(\blacklozenge_{[0,31)}\, \exists t'.\, \exists a'.\, t \not\approx t' \wedge \mathit{trans}(c, t', a') \wedge \Diamond_{[0,6)}\, \mathit{report}(t') \big) \rightarrow \\ \Diamond_{[0,3)}\, \mathit{report}(t)\,. \qquad \textbf{(P4)}$$

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 $\underline{U}, \underline{R}, \underline{A}, \underline{O}, \underline{S}$, binary predicates for the state predicates $\underline{UA}, \underline{user}, \underline{roles}$, and a ternary predicates for the state predicates $\underline{PA}$. 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 $\underline{UA}$ and $\underline{PA}$ represent the user-assignment relation $\mathsf{UA} \subseteq \mathsf{U} \times \mathsf{R}$ and the permission-assignment relation $\mathsf{PA} \subseteq \mathsf{R} \times \mathsf{A} \times \mathsf{O}$ at a given time point. Furthermore, the predicate $\underline{user}$ indicates a user's sessions at a time point and $\underline{roles}$ 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 $\underline{X}$ and the ternary predicate $exec$. The intuitive meaning of these predicates is that $\underline{X}(r, r')$ 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 $\underline{U}$, $\underline{R}$, $\underline{A}$, and so on. The formula (A4) requires that, at every time point, the predicate $\underline{UA}$ is correctly typed, namely, it always only relates currently existing users with currently existing roles:

$$\Box\, \forall u.\, \forall r.\, \underline{UA}(u, r) \rightarrow \underline{U}(u) \wedge \underline{R}(r)\,. \tag{A4}$$

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 $\underline{user}$ represents a function from sessions to users that is constant over a session's lifetime:

$$\Box\, \forall s.\, S_{\mathrm{s}}(s) \rightarrow \exists u.\, \underline{U}(u) \wedge \underline{user}(s, u)\,, \tag{A5}$$

$$\Box\, \forall s.\, \forall u.\, \forall u'.\, \underline{user}(s, u) \wedge \underline{user}(s, u') \rightarrow u \approx u'\,, \tag{A6}$$

$$\Box\, \forall s.\, \forall u.\, \forall u'.\, \underline{user}(s, u) \wedge \big(\bigcirc\, \underline{user}(s, u')\big) \rightarrow u \approx u'\,, \tag{A7}$$

and

$$\Box\, \forall s.\, \forall u.\, \forall u'.\, \neg\big(user_{\mathrm{f}}(s, u) \wedge user_{\mathrm{s}}(s, u')\big)\,. \tag{A8}$$

Recall that $\underline{user}(s, u)$ abbreviates $\neg user_{\mathrm{f}}(s, u) \,\mathsf{S}\, user_{\mathrm{s}}(s, u)$, where the predicates $user_{\mathrm{s}}$ and $user_{\mathrm{f}}$ mark the start events and the finish events for the relationship between subjects and users. The other predicates like S_{s} have similar interpretations. See Remark 6.1. The formula (A9) ensures that the only roles that may be activated in a session are those that are presently assigned to the user associated with the session:

$$\Box\, \forall s.\, \forall r.\, roles_{\mathrm{s}}(s, r) \rightarrow \exists u.\, \underline{user}(s, u) \wedge \underline{UA}(u, r)\,. \tag{A9}$$

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

$$\Box\, \forall s.\, \forall a.\, \forall o.\, exec(s, a, o) \rightarrow \exists r.\, \underline{roles}(s, r) \wedge \underline{PA}(r, a, o)\,. \tag{A10}$$

Finally, we assume that $\underline{X}$ 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

$$\Box\, \forall r.\, \forall r'.\, \underline{X}(r, r') \rightarrow \neg\exists u.\, \underline{UA}(u, r) \wedge \underline{UA}(u, r')\,. \tag{P5}$$

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

$$\Box\, \forall r.\, \forall r'.\, \underline{X}(r,r') \to \neg\exists s.\, \underline{\mathit{roles}}(s,r) \land \big(\neg S_{\mathrm{f}}(s)\ \mathsf{S}\ \underline{\mathit{roles}}(s,r')\big)\,. \tag{P6}$$

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:

$$\begin{aligned} &\Box\, \forall r.\, \forall r'.\, \underline{X}(r,r') \to \\ &\qquad \neg\exists s.\, \exists o.\, \big(\exists a.\, \mathit{exec}(s,a,o) \land \underline{\mathit{roles}}(s,r) \land \underline{\mathit{PA}}(r,a,o)\big) \land \\ &\qquad\qquad \big(\neg S_{\mathrm{f}}(s)\ \mathsf{S}\ \exists a'.\, \mathit{exec}(s,a',o) \land \underline{\mathit{roles}}(s,r') \land \underline{\mathit{PA}}(r',a',o)\big)\,. \end{aligned} \tag{P7}$$

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

6.2. Monitor Implementations

We implemented two prototype tools, MonPoly-Reg and MonPoly-Fin, which respectively implement our monitoring algorithm for the regular-relation setting (Section 3) and the finite-relation setting (Section 4). Both tools are written in the OCaml programming language and their source code is publicly available at [MONPOLY 2013].

For MonPoly-Fin, the monitored formula must satisfy the requirements given in Section 4.2.3. In particular, each formula must be range restricted or the heuristics described Section 4.2.4 must succeed in rewriting the given formula to a range restricted one. MonPoly-Reg does not need these restrictions. It only requires that the given formula is bounded. Both tools use the same input format for representing the temporal structure, which is incrementally processed by the tools. In particular, every relation of a structure at a time point must be finite and given by an enumeration of its elements.

Since both tools extensively manipulate relations, the data structure used to represent them has a huge impact on the tools' performance. MonPoly-Fin uses the data type for finite sets from OCaml's standard library, which is implemented using balanced binary trees. MonPoly-Reg represents regular relations by deterministic finite automata (DFAs), which we always minimize. For this we use the automata library from the MONA tool [Henriksen et al. 1995; Klarlund et al. 2002], which is implemented in C and provides a BDD-based data structure for DFAs along with basic automaton constructions like the product construction. Domain elements and time stamps are natural numbers, which we encode as bit strings, with the least significant bit first. Since padding 0s at the end of such a string does not alter the element it represents, we do not need the special letter # used in convolution to encode tuples of domain elements; see Section 3.1.

6.3. Monitor Performance

In the following, we report on an experimental evaluation of our two tools. We used versions 1.0 of MonPoly-Reg and 1.1.2 of MonPoly-Fin, and a standard desktop computer with an Intel Core i5 2.67 GHz CPU and 8 GBytes of RAM. Furthermore, for our experiments, we used the formulas (P1) to (P4), formalizing the security policies described in Sections 6.1.1 and 6.1.2. We evaluated these formulas on synthetically generated log files, which satisfy the assumptions that "well-formed" systems should satisfy, as stated in the respective sections. We dropped the formulas' universal quantifiers so that the tools report policy violations as pairs of an accountant a and a report f, in the case of (P1), and as triples of a customer c, a transaction t and an amount a, in the other

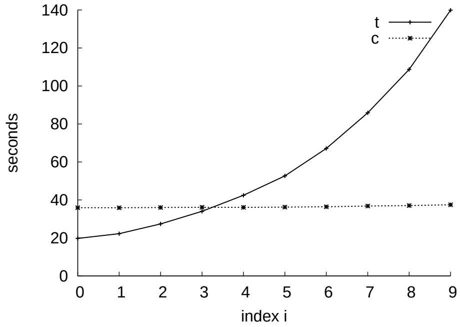


Fig. 7. Run time for MonPoly-Reg for (P3) and increasing upper bounds on values for the parameters t and c.

cases. The scripts used in the evaluation, for instance, to generate the data, are publicly available at the tools' web page [MONPOLY 2013].

We assess the tools' performance by carrying out experiments to answer the following questions. (1) What are the tools' run time and memory consumption with respect to different event rates, which is the average number of events per second? (2) What is the maximum event rate that the tools can handle online? (3) How do the tools' performance compare with an off-the-shelf database management system (DBMS)?

6.3.1. Preliminaries. Before describing the experiments, we make several remarks. First, when generating log files, we restrict ourselves for simplicity to relational structures with singleton relations. Thus there is exactly one event per time point in a generated log file. By default, a generated log file spans over 300 seconds. For generation, we also fix the event rate, that is, the average number of events per second. For each time stamp, the number of events is randomly chosen within $\pm 10\%$ of the fixed event rate. Moreover, the generated log files are such that the number of violations with respect to a policy depends on the event rate. For instance, for policy (P1) the number of violations 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 such that the event and violation rates are respected.

Second, except for the formula (P4), MonPoly-Fin's rewriter automatically obtains monitorable formulas. For (P4), the implemented heuristics fail and we had to manually rewrite the formula to guide MonPoly-Fin's rewriter to obtain a monitorable formula.

Finally, note that MonPoly-Reg's run times depend on the magnitude of the data values occurring in the processed log file. Indeed, the sizes of the DFAs built during the runs have a huge impact on the tool's running times, and the DFA's sizes in turn depend on the sizes of the constants. For instance, the minimal DFA for the formula $x \approx c$, with x a variable and $c \in \mathbb{N}$ a constant, has size $O(\log c)$. MonPoly-Fin's performance has no such dependencies since it uses machine integers with a fixed bit length.

Figure 7 provides a concrete illustration of the impact of the sizes of the data values on MonPoly-Reg's run time. In this experiment, we generated 10 logs files that differ in the upper bound of one of the parameters t or c in the formula (P3). For each index i on the x-axis, the parameter t (for the solid line) and c (for the dashed line) is at most $100 \cdot 2^i$. The upper bounds for the parameters a, e, and t or c are 2500, 100, and 1000, respectively. Note that either only t's or only c's upper bound varies with the index i. The parameter th is always 2000. The time span of the logs is fixed to 60 seconds and the

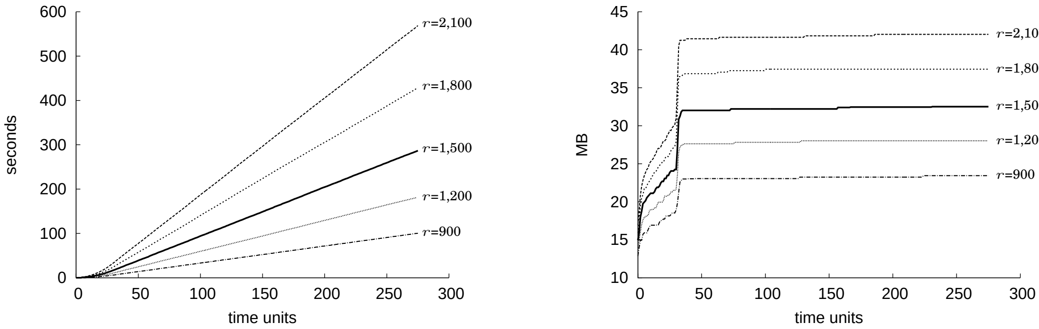


Fig. 8. Average run time and maximal memory usage for MonPoly-Fin, for (P4) and event rates r.

event rate is 100 events per second on average. Our experiment shows that increasing c's upper bound has almost no effect on MonPoly-Reg's run time (the dashed line slowly increases from 36 to 37 seconds). In contrast, the impact of increasing t's upper bound is significant.

The observed behavior can be explained as follows. First, the upper bound for either c or t has a minor impact on the average size of the minimal DFA for the relation $trans^{\mathcal{D}_j}$. The average number of states is between 14 and 18, where the average is taken over all time points j, for each upper bound. Second, the sizes of the minimal DFAs for the subformula $\alpha := \blacklozenge_{[2,21)}\, \exists e.\ auth(e,t)$ grow significantly when increasing the upper bound on t. Concretely, the average size of the minimal DFAs for α grows from 49 to 427 states. Instead, when increasing the upper bound on c, the average size of the minimal DFAs for α is always 159, as c does not occur in α. We note that the average cardinality of the relation $\alpha^{\mathcal{D}_j}$ is always around 30 tuples, when varying the upper bounds for either c or t.

In the following experiments, the upper bounds for parameters m in policy (P1) and a in policies (P2) to (P4) are 10 and respectively 2500, while the upper bounds on all other parameters depend on the event rate and are at most 50 times larger than the event rate.

6.3.2. Resource Consumption with Respect to the Event Rate. Figure 8 shows MonPoly-Fin's resource consumption for the formula (P4) for the event rates 900, 1,200, 1,500, 1,800, and 2,100 events per second on average. 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. The run time on individual log files deviates from the average by at most 15%. We conducted similar experiments for all the other formulas with both tools. The graphs are similar and are thus omitted.

We observe in the graph of the run times (left-hand side of Figure 8) that the time needed to process the logs grows linearly with the time span of the log files, where the slope depends on 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).

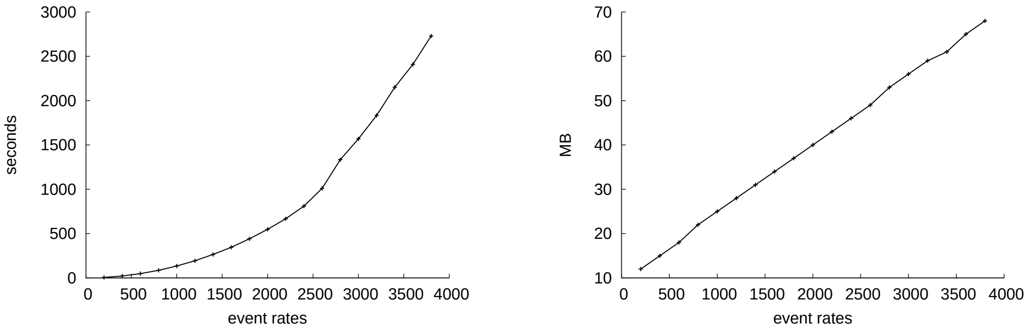


Fig. 9. Average run time and memory usage for MonPoly-Fin for (P4) for increasing event rates.

We further observe in the graph of memory usage (right-hand side of Figure 8) that after a start-up phase the memory consumption stabilizes. In particular, memory consumption does not grow as the total number of events increases. In the case of MonPoly-Reg, the memory consumption increases slightly over time, up to 5 MBytes during a run. This is because, in our implementation, the index and age fields in the auxiliary relations are absolute and not relative to the current time point. This results in larger constants at later time points and thus larger DFAs, as discussed in Section 6.3.1.

Figure 9 shows how MonPoly-Fin'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 the size of one of the intermediate relations (that is, the satisfying valuations of the subformula $\blacklozenge_{[0,11)} \exists m.\, \underline{mgr}(m,a) \wedge \mathit{approve}(m,f))$ being quadratic in the event rate. Furthermore, for the 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. 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 indices.

We conducted the same experiments with MonPoly-Reg. The graphs and the observations are similar to those for Figure 9. However, the slopes of the graphs are smaller, especially for memory usage. This is because the event rates used in the experiments for MonPoly-Reg, are smaller compared to the ones used in the MonPoly-Fin experiments, namely, 20, 100, 1000, and 10 times smaller for the policies (P1), (P2), (P3), and (P4), respectively. The reason for using smaller event rates becomes clear with the following experiments.

6.3.3. Maximal Event Rate for Online Monitoring. As we have seen in the previous subsection, the performance of both tools 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

Table I. Maximal event rate for online monitoring and corresponding space consumption.

formula	MonPoly-Reg		MonPoly-Fin	
	event rate	space	event rate	space
(P1)	61	19	1,038	389
(P2)	95	24	14,272	70
(P3)	140	18	156,761	99
(P4)	46	31	1,506	32

Note: Event rate is in events per second and space in MBytes.

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 the tools' can be used online.

We determine this maximal event rate for online monitoring 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. Table I lists the obtained event rates for each tool and each of the four formulas, together with the maximal space consumption during a run at these event rates.

These numbers also show which policies are hard to monitor with each tool. As expected, (P1) and (P4) are harder to monitor than (P2) and (P3), because (P1) and (P4) are larger and contain more temporal operators. The formula (P2) is easier to monitor than (P3) because the constructions of the auxiliary relations for past temporal operators are simpler than those for future operators. Furthermore, for MonPoly-Fin, this difference is accentuated by the previously mentioned optimization. Monitoring the formula (P1) is faster than monitoring (P4) for MonPoly-Reg, and conversely for MonPoly-Fin. Due to the different structures of both the formulas and of the generated log files, it is difficult to pinpoint the precise reasons for this behavior. One explanation is that, due to optimizations in MonPoly-Fin, the presence of future temporal operators in (P4) has a smaller impact for MonPoly-Fin than for MonPoly-Reg. What has a larger impact for MonPoly-Fin is the fact that an intermediary relation for (P1) has quadratic size in the event rate, while all intermediary relations for (P4) are at most linear in the event rate.

In addition to the experiments reported on here, we also used MonPoly-Fin in a real-world case study. In [Basin et al. 2013], we analyzed a log file containing more than 218 million events, representing roughly one year's worth of logged data. In this case study, the average event rate was thus 6, with a peak of $3{,}964$ events per second. Furthermore, there were 14 formulas and the formulas' largest time window was 30 days. Only two of the formulas needed to be manually rewritten for monitoring. For each formula, the log file was processed in less than an hour. While we used MonPoly-Fin offline in this case study to report policy violations after the fact, it could also have been used online, since the lowest throughput was approximately $60{,}771$ events per second, which is significantly larger than the average event rate.

6.3.4. Comparison with a DBMS. As a final experiment, we compare both tools 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-Reg, MonPoly-Fin, 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

Table II. Comparison of run times of PostgreSQL, MonPoly-Reg, and MonPoly-Fin.

formula	tool \ time span	300	600	1,200	2,400	4,800	9,600	19,200	38,400
(P1)	PostgreSQL	4.3	16	67	266	1,065	4,234	16,974	†
	MonPoly-Reg	7.3	16	35	74	158	338	703	1,493
	MonPoly-Fin	0.02	0.05	0.1	0.2	0.5	0.9	1.9	3.8
(P2)	PostgreSQL	0.3	0.8	2.0	22	76	289	199	†
	MonPoly-Reg	†							
	MonPoly-Fin	2.6	5.4	11	21	42	87	167	307
(P3)	PostgreSQL	0.3	0.8	2.1	22	75	290	197	†
	MonPoly-Reg	18,275	†						
	MonPoly-Fin	1.6	3.1	6.2	12	25	51	100	201
(P4)	PostgreSQL	0.7	2.3	840	3,246	12,563	†		
	MonPoly-Reg	1,456	3,087	6,465	13,326	†			
	MonPoly-Fin	1.7	3.3	6.7	13	27	55	110	221

Notes: Run times are in second. The symbol † means that the run did not finish within 6 hours (i.e. 21,600 seconds).

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 $S = (C, R, \iota)$ into new signatures S' by increasing the arity of each predicate in R by 2, adding a new predicate 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 ϕ over S into first-order formulas $\bar{\phi}$ over S'. Given a temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$, we build a structure $\mathcal{M}$ with $\mathit{tpts}^{\mathcal{M}} := \{(i, \tau_i) \mid i \in \mathbb{N}\}$ and $r^{\mathcal{M}} := \{(i, \tau_i, \bar{a}) \mid i \in \mathbb{N} \text{ and } \bar{a} \in r^{\mathcal{D}_i}\}$, 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 $\overline{\blacklozenge_{[b,b')}\, \phi} := \exists i'.\, \exists t'.\, \mathit{tpts}(i', t') \wedge i' \leq i \wedge b \leq t - t' \wedge t - t' < b' \wedge \bar{\phi}$, 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 $\bigcup_{i \in \mathbb{N}} \phi^{(\bar{\mathcal{D}}, \bar{\tau}, i)} = (\exists i.\, \exists t.\, \mathit{tpts}(i, t) \wedge \bar{\phi})^{\mathcal{M}}$. 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 $\mathcal{M}_{\mathit{fin}}$ where the relations for the flexible predicates are finite.

We generate log files with the following event rates: 10 events per second on average 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-Reg and MonPoly-Fin 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-Fin's run time doubles at each iteration. This behavior corresponds to the one illustrated in Figure 8. We observe a similar behavior for MonPoly-Reg, with a multiplication factor slightly larger than 2, due to the use of absolute indices and time stamps, as previously explained. For PostgreSQL, the run-time growth rate is not constant, because PostgreSQL generally changes the query execution plan from one iteration to the next. In all cases, the run time explodes after some iterations.

Note that when this explosion occurs, we observe that temporary files are created on disk, indicating that intermediary data no longer fits into main memory. For the formulas (P2) and (P3), PostgreSQL is faster up to a point, while for the other two formulas, MonPoly-Fin is faster. MonPoly-Reg is substantially slower than the other tools for all formulas except (P1), for which it quickly outperforms PostgreSQL. Furthermore, indexing does not significantly influence PostgreSQL's run times. Note that a comparison of the run times between the different formulas for the same tool is not sensible since the event rates of the generated log files for the formulas differ.

In summary, MonPoly-Reg is outperformed by PostgreSQL. In contrast, MonPoly-Fin performs reasonably well even in an offline setting, where it may outperform PostgreSQL, especially 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 PostgreSQL's performance. This experiment also demonstrates that MonPoly-Reg's generality comes at a cost in performance. The observed performance difference between MonPoly-Reg and MonPoly-Fin is consistent with our observations when measuring the maximal throughput of the two tools.

7. RELATED WORK

Temporal logics are widely applicable in computing since they allow one to formally and naturally express system properties and to reason about them algorithmically. For instance, the propositional temporal logics LTL, CTL, and PSL are extensively used in system verification, in particular, in model checking [Pnueli 1977; Clarke and Emerson 1982; Vardi 2009]. In the following, we focus on related monitoring algorithms that handle temporal logic specifications. We group these 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 by instrumenting the program's source or binary code. In some cases, the monitors themselves are directly instrumented into the code. Many of the developed monitoring algorithms for program verification use a propositional temporal logic for specifying properties. For example, monitoring algorithms exist for LTL and variants [Giannakopoulou and Havelund 2001; Finkbeiner and Sipma 2004] and for propositional real-time logics [Thati and Roşu 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 based on the current event, resulting in a formula that states the obligations that must be satisfied by the remainder of the execution [Havelund and Roşu 2004; Roşu 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 monitoring algorithms overcome this limitation by handling specification languages with propositions that have parameters. Examples include EAGLE [Barringer et al. 2004], LOLA [D'Angelo et al. 2005], J-LO [Stolz and Bodden 2006], RuleR [Barringer et al. 2010b], LogScope [Barringer et al. 2010a], and TraceContract [Barringer and Havelund 2011]. The semantic models underlying these monitoring algorithms are different from ours. For instance, EAGLE's models are sequences of states, where a state is a mapping from parameters to data values, while MFOTL models are temporal structures. EAGLE's models can be seen as temporal structures over a signature with a single predicate, say $state$, whose interpretation at every time point is a singleton.

Another difference between parameterized approaches and MFOTL is how parameters and variables are bound and instantiated. EAGLE, for example, does not have an explicit notion of quantification. However its variable binding has the flavor of binding under freeze quantification, which binds a variable to the corresponding data value at the current time point. The freeze quantifier was introduced by Alur and Henzinger [1994] for time variables, which are variables that relate the time stamps of different time points. Freeze quantification corresponds to a restricted form of standard first-order quantification, see also [Henzinger 1990]. In particular, it restricts variable instantiations to state or event parameters, rather than permitting quantification over the entire domain. For the restricted semantic models described above, the three forms of quantification coincide only when an MFOTL formula implicitly restricts the scope of an universal or existential quantifier to the current state, since then there is exactly one possible variable instantiation. For instance, replacing both quantifiers in the MFOTL formula $\Box \forall x.\, p(x) \rightarrow \Box_{[1,5)} \exists y.\, q(x,y)$ by freeze quantifiers would closely mimic the formula's MFOTL semantics. Note, however, that the placement of the quantifiers does matter for the formula's meaning. For instance, by moving the existential quantifier outside the second $\Box$ operator, the scope of y's existential quantification is no longer locally restricted to a single time point and freeze quantification would be too weak then. The formulas (P5), (P6), and (P7) from Section 6.1 are further examples where the quantification is not locally restricted to a time point and freeze quantification is insufficient. In these examples, the roles r and r', referenced at each time point, might occur as data values only at previous time points.

Local variables from the temporal specification languages PSL [IEEE Std 1850-2010] and SVA [IEEE Std 1800-2009] are also related to the parameterized monitoring semantics. In fact, they can be used to mimic freeze quantification. A data value that occurs at the current position in the trace can be assigned to a local variable, which can be read at other positions in the trace. However, local variables are different from logical variables. In particular, we can apply functions to them like increment and decrement, which modify the local variables' stored value. This means that local variables have the flavor of variables in imperative programming. Although monitoring approaches for PSL and SVA exist, for example, that of Pnueli and Zaks [2006], we are not aware of any monitoring approach for PSL or SVA that supports local variables. Note that since the type of a local variable in PSL and SVA is always a finite set, local variables do not increase the expressivity of these specification languages. However, they can be useful for specifying properties succinctly.

Another approach to monitoring parametric specifications is that of JavaMOP [Meredith et al. 2012], which was further extended by Roşu and Chen [2012]. This approach separates parameter binding from property checking, and this leads to a monitoring framework that can handle various specification languages like regular expressions and temporal logics. The framework slices the input trace at run-time by removing parameters and non-relevant events, and monitors each slice with respect to the non-parameterized version of the specification. Note that, as with the other parameterized monitoring approaches, no distinction can be made between universal and existential quantification of variables. Furthermore, in contrast to other approaches, the scope of parameters is the entire formula and cannot be restricted to subformulas. Finally, in contrast to our approach, no verdict is given for the initial parameterized trace and instead a verdict is given for each slice. Work of [Barringer et al. 2012] generalizes the parametric trace slicing approach by using so-called quantified event automata. There, parameters can be explicitly quantified and the quantification ranges over the values that appear in the trace. However, automata for complex policies can be large and thus difficult to specify, understand, and maintain. It is unclear if there is an equivalent declarative specification language.

In summary, while MFOTL's semantic model is more general than parameterized event or state sequences, its semantics and expressivity is in general incomparable to parameterized specification languages. This incompatibility is rooted in the differences between MFOTL's standard first-order quantification and the specific way in which parameters are instantiated in a particular parameterized specification language. Nevertheless, in terms of expressing system properties, MFOTL seems better suited for security and compliance policies, such as the ones in Section 6, especially due to its more general semantic model and its support for universal and existential quantification. In contrast, when verifying program behavior during runtime, the model restrictions of the parameterized monitoring approaches are often met, and these approaches have proved to be effective there.

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 [2013] 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. [2013] 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]. [Dinesh et al. 2008] uses a formula-rewriting approach, similar to EAGLE, for checking conformance of traces to 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. In this work, 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 parameterized 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, parameterized as previously explained.

In contrast to the above, the monitoring algorithm of 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 $\Box\,\forall x.\, p(x) \rightarrow \exists y.\ \Box_{[1,5)}\, q(x,y)$. 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 (P5), (P6), and (P7) 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 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 is independent of the number of nested quantifiers. A final difference is that the monitoring algorithm of 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 restrictions similar to those of 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 for processing data 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. 2013a].

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 algorithm handles a richer specification language than Chomicki's. For example, it 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, variable quantification 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 and relations between them, 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 can be seen as processing an input data stream, given as a temporal structure, and producing an output data stream, which is 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 whether we can leverage work in these domains to increase the scope and efficiency of our monitoring algorithm, in particular for the finite-relation setting.

8. CONCLUSION

Runtime monitoring has evolved over the past several decades 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. We show that the algorithm has wide applicability and that, for the specialization to finite relations, time and space requirements are moderate in practice. Overall, our results show that MFOTL is an effective language for specifying and monitoring a wide variety of practically relevant system properties.

We emphasize that our approach is not a panacea: there is no one silver bullet that covers all applications and handles all system properties equally well. Recently, Basin et al. [2013b] extended MFOTL with features commonly found in stream processing languages namely, aggregation operators like the maximum, sum, and average over a specified time window. Returning to our transaction-processing example from Section 6.1.2, these extensions allow one to formalize and monitor requirements like “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.” We can envision further extensions here, for example, support for specifications with arbitrary user-defined recursive functions. Additionally, one could liberalize some of our semantic assumptions, for example, by weakening the assumption that the time stamps associated with events are exact to that they are merely approximate, for instance, within some interval.

Another area for future work concerns distributed and highly scalable monitoring. Many IT systems are composed of distributed, concurrently executing subsystems and monitoring their compliance to policies is a major challenge. One fundamental problem is to soundly and effectively distribute monitoring for a global system property. Since the 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 problem 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, parallelized monitoring appears necessary. While progress on both problems has been made, see, for example, [Bauer and Falcone 2012] and [Basin et al. 2014] for decentralized and parallel monitoring respectively, many challenges remain both in the design of robust theoretical solutions and their application in practice. For instance, Bauer and Falcone [2012] assume a lock-step semantics of the system components, and Basin et al. [2014] use the MapReduce framework [Dean and Ghemawat 2008], which is ill-suited for monitoring system behavior online.

REFERENCES

107th Congress. 2001. Uniting and Strengthening America by Providing Appropriate Tools Required to Intercept and Obstruct Terrorism Act of 2001 (USA PATRIOT ACT). (2001). Public Law 107-56.

Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Bowen Alpern and Fred B. Schneider. 1985. Defining Liveness. *Inform. Process. Lett.* 21, 4 (1985), 181–185.

Rajeev Alur and Thomas A. Henzinger. 1992. Logics and Models of Real Time: A Survey. In *Proceedings of the 1991 REX Workshop on Real Time: Theory in Practice (Lect. Notes Comput. Sci.)*, Vol. 600. Springer, Heidelberg, Germany, 74–106.

Rajeev Alur and Thomas A. Henzinger. 1994. A Really Temporal Logic. *J. ACM* 41, 1 (1994), 181–204.

Arvind Arasu, Shivnath Babu, and Jennifer Widom. 2006. The CQL continuous query language: semantic foundations and query execution. *VLDB Journal* 15, 2 (2006), 121–142.

Franz Baader, Andreas Bauer, and Marcel Lippmann. 2009. Runtime Verification Using a Temporal Description Logic. In *Proceedings of the 7th International Symposium on Frontiers of Combining Systems (Lect. Notes Comput. Sci.)*, Vol. 5749. Springer, Heidelberg, Germany, 149–164.

Luciano Baresi, Domenico Bianculli, Sam Guinea, and Paola Spoletini. 2009. Keep It Small, Keep It Real: Efficient Run-Time Verification of Web Service Compositions. In *Formal Techniques for Distributed Systems, Proceedings of the Joint 11th IFIP WG 6.1 International Conference FMOODS and 29th IFIP WG 6.1 International Conference FORTE (Lect. Notes Comput. Sci.)*, Vol. 5522. Springer, Heidelberg, Germany, 26–40.

Howard Barringer, Yliès Falcone, Klaus Havelund, Giles Reger, and David E. Rydeheard. 2012. Quantified Event Automata: Towards Expressive and Efficient Runtime Monitors. In *Proceedings of the 18th International Symposium on Formal Methods (Lect. Notes Comput. Sci.)*, Vol. 7436. Springer, Heidelberg, Germany, 68–84.

Howard Barringer, Allen Goldberg, Klaus Havelund, and Koushik Sen. 2004. Rule-Based Runtime Verification. In *Proceedings of the 5th International Conference on Verification, Model Checking and Abstract Interpretation (Lect. Notes Comput. Sci.)*, Vol. 2937. Springer, Heidelberg, Germany, 44–57.

Howard Barringer, Alex Groce, Klaus Havelund, and Margaret H. Smith. 2010a. Formal Analysis of Log Files. *Journal of Aerospace Computing, Information, and Communication* 7, 11 (2010), 365–390.

Howard Barringer and Klaus Havelund. 2011. TraceContract: A Scala DSL for Trace Analysis. In *Proceedings of the 18th International Symposium on Formal Methods (Lect. Notes Comput. Sci.)*, Vol. 6664. Springer, Heidelberg, Germany, 57–72.

Howard Barringer, David E. Rydeheard, and Klaus Havelund. 2010b. Rule Systems for Run-Time Monitoring: From Eagle to RuleR. *J. Logic Comput.* 20, 3 (2010), 675–706.

Adam Barth, Anupam Datta, John C. Mitchell, and Helen Nissenbaum. 2006. Privacy and Contextual Integrity: Framework and Applications. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*. IEEE Computer Society, Los Alamitos, CA, USA, 184–198.

David Basin, Germano Caronni, Sarah Ereth, Matúš Harvan, Felix Klaedtke, and Heiko Mantel. 2014. Scalable Offline Monitoring. In *Proceedings of the 5th International Conference on Runtime Verification (Lect. Notes Comput. Sci.)*, Vol. 8734. Springer, Heidelberg, Germany, 31–47.

David Basin, Matúš Harvan, Felix Klaedtke, and Eugen Zălinescu. 2012. MONPOLY: Monitoring Usage-control Policies. In *Proceedings of the 2nd International Conference on Runtime Verification (Lect. Notes Comput. Sci.)*, Vol. 7186. Springer, Heidelberg, Germany, 360–364.

David Basin, Matúš Harvan, Felix Klaedtke, and Eugen Zălinescu. 2013. Monitoring Data Usage in Distributed Systems. *IEEE Trans. Software Eng.* 39, 10 (2013), 1403–1426.

David Basin, Felix Klaedtke, Srdjan Marinovic, and Eugen Zălinescu. 2013a. Monitoring Compliance Policies over Incomplete and Disagreeing Logs. In *Proceedings of the 3rd International Conference on Runtime Verification (Lect. Notes Comput. Sci.)*, Vol. 7687. Springer, Heidelberg, Germany, 151–167.

David Basin, Felix Klaedtke, Srdjan Marinovic, and Eugen Zălinescu. 2013b. Monitoring of Temporal First-order Properties with Aggregations. In *4th International Conference on Runtime Verification (RV'13) (Lect. Notes Comput. Sci.)*, Vol. 8174. Springer, Heidelberg, Germany, 40–58.

David Basin, Felix Klaedtke, and Samuel Müller. 2010a. Monitoring Security Policies with Metric First-order Temporal Logic. In *Proceedings of the 15th ACM Symposium on Access Control Models and Technologies*. ACM Press, New York, NY, USA, 23–33.

David Basin, Felix Klaedtke, and Samuel Müller. 2010b. Policy Monitoring in First-Order Temporal Logic. In *Proceedings of the 22nd International Conference on Computer Aided Verification (Lect. Notes Comput. Sci.)*, Vol. 6174. Springer, Heidelberg, Germany, 1–18.

David Basin, Felix Klaedtke, Samuel Müller, and Birgit Pfitzmann. 2008. Runtime Monitoring of Metric First-order Temporal Properties. In *Proceedings of the 28th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (Leibniz International Proceedings in Informatics (LIPIcs))*, Vol. 2. Schloss Dagstuhl - Leibniz Center for Informatics, 49–60.

David Basin, Felix Klaedtke, and Eugen Zălinescu. 2012. Algorithms for Monitoring Real-time Properties. In *Proceedings of the 2nd International Conference on Runtime Verification (Lect. Notes Comput. Sci.)*, Vol. 7186. Springer, Heidelberg, Germany, 260–275.

Andreas Bauer and Yliès Falcone. 2012. Decentralised LTL Monitoring. In *Proceedings of the 18th International Symposium on Formal Methods (Lect. Notes Comput. Sci.)*, Vol. 7436. Springer, Heidelberg, Germany, 85–100.

Andreas Bauer, Rajeev Goré, and Alwen Tiu. 2009. A First-Order Policy Language for History-Based Transaction Monitoring. In *Proceedings of the 6th International Colloquium on Theoretical Aspects of Computing (Lect. Notes Comput. Sci.)*, Vol. 5684. Springer, Heidelberg, Germany, 96–111.

Andreas Bauer, Martin Leucker, and Christian Schallhart. 2011. Runtime Verification for LTL and TLTL. *ACM Trans. Softw. Eng. Methodol.* 20, 4 (2011).

Achim Blumensath and Erich Grädel. 2004. Finite Presentations of Infinite Structures: Automata and Interpretations. *Theory Comput. Syst.* 37, 6 (2004), 641–674.

Jan Chomicki. 1995. Efficient Checking of Temporal Integrity Constraints Using Bounded History Encoding. *ACM Trans. Database Syst.* 20, 2 (1995), 149–186.

Jan Chomicki and Damian Niwiński. 1995. On the Feasibility of Checking Temporal Integrity Constraints. *J. Comput. Syst. Sci.* 51, 3 (1995), 523–535.

Jan Chomicki and David Toman. 1995. Implementing Temporal Integrity Constraints Using an Active DBMS. *IEEE Trans. Knowl. Data Eng.* 7, 4 (1995), 566–582.

Jan Chomicki, David Toman, and Michael H. Böhlen. 2001. Querying ATSQL Databases with Temporal Logic. *ACM Trans. Database Syst.* 26, 2 (2001), 145–178.

Edmund M. Clarke and E. Allen Emerson. 1982. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *Proceedinggs of the 1981 Workshop on Logics of Programs (Lect. Notes Comput. Sci.)*, Vol. 131. Springer, Heidelberg, Germany, 52–71.

Gianpaolo Cugola and Alessandro Margara. 2012. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.* 44, 3 (2012).

Ben D'Angelo, Sriram Sankaranarayanan, César Sánchez, Will Robinson, Bernd Finkbeiner, Henny B. Sipma, Sandeep Mehrotra, and Zohar Manna. 2005. LOLA: Runtime Monitoring of Synchronous Systems. In *Proceedings of the 12th International Symposium on Temporal Representation and Reasoning*. IEEE Computer Society, Los Alamitos, CA, USA, 166–174.

Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.

Department of the Treasury. 1970. Bank Secrecy Act of 1970 (BSA). (1970). 31 USC 5311-5332 and 31 CFR 103.

Robert A. Di Paola. 1969. The Recursive Unsolvability of the Decision Problem for the Class of Definite Formulas. *J. ACM* 16, 2 (1969), 324–327.

Nikhil Dinesh, Aravind Joshi, Insup Lee, and Oleg Sokolsky. 2008. Checking Traces for Regulatory Conformance. In *Proceedings of the 8th Workshop on Runtime Verification (Lect. Notes Comput. Sci.)*, Vol. 5289. Springer, Heidelberg, Germany, 86–103.

Daniel J. Dougherty, Kathi Fisler, and Shriram Krishnamurthi. 2007. Obligations and Their Interaction with Programs. In *Proceedings of the 12th European Symposium on Research in Computer Security (Lect. Notes Comput. Sci.)*, Vol. 4734. Springer, Heidelberg, Germany, 375–289.

Herbert B. Enderton. 1972. *A Mathematical Introduction to Logic*. Academic Press, San Diego, CA, USA.

David F. Ferraiolo, Ravi S. Sandhu, Serban I. Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. 2001. Proposed NIST standard for role-based access control. *ACM Trans. Inform. Syst. Secur.* 4, 3 (2001), 224–274.

Bernd Finkbeiner and Henny Sipma. 2004. Checking Finite Traces Using Alternating Automata. *Form. Method. Syst. Des.* 24, 2 (2004), 101–127.

Deepak Garg, Limin Jia, and Anupam Datta. 2011. Policy auditing over incomplete logs: theory, implementation and applications. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*. ACM Press, New York, NY, USA, 151–162.

Dimitra Giannakopoulou and Klaus Havelund. 2001. Automata-Based Verification of Temporal Properties on Running Programs. In *Proceedings of the 16th IEEE International Conference on Automated Software Engineering*. IEEE Computer Society, Los Alamitos, CA, USA, 412–416.

Christopher Giblin, Alice Y. Liu, Samuel Müller, Birgit Pfitzmann, and Xin Zhou. 2005. Regulations Expressed As Logical Models (REALM). In *Proceedings of the 18th Annual Conference on Legal Knowledge and Information Systems (Frontiers Artificial Intelligence Appl.)*, Vol. 134. IOS Press, Amsterdam, The Netherlands, 37–48.

Sylvain Hallé and Roger Villemaire. 2012. Runtime Enforcement of Web Service Message Contracts with Data. *IEEE Trans. Serv. Comput.* 5, 2 (2012), 192–206.

Klaus Havelund and Grigore Roşu. 2004. Efficient monitoring of safety properties. *Int. J. Softw. Tools Technol. Trans.* 6, 2 (2004), 158–173.

Klaus Havelund and Willem Visser. 2002. Program model checking as a new trend. *Int. J. Softw. Tools Technol. Trans.* 4, 1 (2002), 8–20.

Jesper G. Henriksen, Jakob L. Jensen, Michael E. Jørgensen, Nils Klarlund, Robert Paige, Theis Rauhe, and Anders Sandholm. 1995. Mona: Monadic Second-Order Logic in Practice. In *Proceedings of the 1st International Workshop on Tools and Algorithms for Contruction and Analysis of Systems (Lect. Notes Comput. Sci.)*, Vol. 1019. Springer, Heidelberg, Germany, 89–110.

Thomas A. Henzinger. 1990. Half-order modal logic: how to prove real-time properties. In *Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing*. ACM Press, New York, NY, USA, 281–296.

Thomas A. Henzinger. 1992. Sooner is safer than later. *Inform. Process. Lett.* 43, 3 (1992), 135–141.

Manuel Hilty, David Basin, and Alexander Pretschner. 2005. On Obligations. In *Proceedings of the 10th European Symposium on Research in Computer Security (Lect. Notes Comput. Sci.)*, Vol. 3679. Springer, Heidelberg, Germany, 98–117.

IEEE Std 1800-2009 2009. Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language. (December 2009). http://ieeexlore.ieee.org/xpls/abs_all.jsp?arnumber=5354441&tag=1.

IEEE Std 1850-2010 2012. Standard for Property Specification Language (PSL). (June 2012). http://ieeexlore.ieee.org/xpl/articleDetails.jsp?arnumber=6228486.

Yonit Kesten, Oded Maler, Monica Marcus, Amir Pnueli, and Elad Shahar. 2001. Symbolic model checking with rich assertional languages. *Theoret. Comput. Sci.* 256, 1–2 (2001), 93–112.

Bakhadyr Khoussainov and Anil Nerode. 1995. Automatic Presentations of Structures. In *Proceedings of the International Workshop on Logical and Computational Complexity (Lect. Notes Comput. Sci.)*, Vol. 960. Springer, Heidelberg, Germany, 367–392.

Nils Klarlund, Anders Møller, and Michael I. Schwartzbach. 2002. MONA Implementation Secrets. *Int. J. Found. Comput. Sci.* 13, 4 (2002), 571–586.

Ron Koymans. 1990. Specifying Real-Time Properties with Metric Temporal Logic. *Real-Time Syst.* 2, 4 (1990), 255–299.

Orna Lichtenstein, Amir Pnueli, and Lenore D. Zuck. 1985. The Glory of the Past. In *Proceedings of the Conference on Logic of Programs (Lect. Notes Comput. Sci.)*, Vol. 193. Springer, Heidelberg, Germany, 196–218.

Udo Walter Lipeck and Gunter Saake. 1987. Monitoring dynamic integrity constraints based on temporal logic. *Inf. Sys.* 12, 3 (1987), 255–269.

Fabrizio Maria Maggi, Marco Montali, Michael Westergaard, and Wil M. P. van der Aalst. 2011. Monitoring Business Constraints with Linear Temporal Logic: An Approach Based on Colored Automata. In *Proceedings of the 9th International Conference on Business Process Management (Lect. Notes Comput. Sci.)*, Vol. 6896. Springer, Heidelberg, Germany, 132–147.

Oded Maler and Dejan Nickovic. 2013. Monitoring properties of analog and mixed-signal circuits. *Int. J. Softw. Tools Technol. Trans.* 15 (2013), 247–268. Issue 3.

Patrick O'Neil Meredith, Dongyun Jin, Dennis Griffith, Feng Chen, and Grigore Roşu. 2012. An overview of the MOP runtime verification framework. *Int. J. Softw. Tools Technol. Trans.* 14, 3 (2012), 249–289.

MONPOLY 2013. (2013). MonPoly source code and examples, available at http://sourceforge.net/projects/monpoly.

Amir Pnueli. 1977. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society, Los Alamitos, CA, USA, 46–57.

Amir Pnueli and Aleksandr Zaks. 2006. PSL Model Checking and Run-Time Verification Via Testers. In *Proceedings of the 14th International Symposium on Formal Methods (Lect. Notes Comput. Sci.)*, Vol. 4085. Springer, Heidelberg, Germany, 573–586.

PostgreSQL Global Development Group. 2012. PostgreSQL, Version 9.1.4. (2012). http://www.postgresql.org/.

Thomas Reinbacher, Matthias Fuegger, and Jörg Brauer. 2013. Real-Time Runtime Verification on chip. In *Proceedings of the 3rd International Conference on Runtime Verification (Lect. Notes Comput. Sci.)*, Vol. 7687. Springer, Heidelberg, Germany, 110–125.

Muriel Roger and Jean Goubault-Larrecq. 2001. Log Auditing through Model-Checking. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop*. IEEE Computer Society, Los Alamitos, CA, USA, 220–234.

Grigore Roşu and Feng Chen. 2012. Semantics and Algorithms for Parametric Monitoring. *Log. Method. Comput. Sci.* 8, 1 (2012).

Grigore Roşu and Klaus Havelund. 2005. Rewriting-Based Techniques for Runtime Verification. *Automat. Softw. Eng.* 12, 2 (2005), 151–197.

A. Prasad Sistla and Ouri Wolfson. 1995. Temporal Triggers in Active Databases. *IEEE Trans. Knowl. Data Eng.* 7, 3 (1995), 471–486.

Volker Stolz and Eric Bodden. 2006. Temporal Assertions using AspectJ. *Elec. Notes Theo. Comput. Sci.* 144, 4 (2006), 109–124.

Prasanna Thati and Grigore Roşu. 2005. Monitoring Algorithms for Metric Temporal Logic Specifications. In *Proceedings of the 4th Workshop on Runtime Verification (Elec. Notes Theo. Comput. Sci.)*, Vol. 113. Elsevier Science Inc., Amsterdam, The Netherlands, 145–162.

Allen Van Gelder and Rodney W. Topor. 1991. Safety and translation of relational calculus. *ACM Trans. Database Syst.* 16, 2 (1991), 235–278.

Moshe Y. Vardi. 2009. From Philosophical to Industrial Logics. In *Proceedings of the 3rd Indian Conference on Logic and its Applications (Lect. Notes Comput. Sci.)*, Vol. 5378. Springer, Heidelberg, Germany, 89–115.

Xinwen Zhang, Francesco Parisi-Presicce, Ravi Sandhu, and Jaehong Park. 2005. Formal Model and Policy Specification of Usage Control. *ACM Trans. Inform. Syst. Secur.* 8, 4 (2005), 351–387.

Received .; revised .; accepted .