

# Monitoring Security Policies with Metric First-order Temporal Logic

David Basin  
ETH Zurich, Switzerland  
basin@inf.ethz.ch

Felix Klaedtke  
ETH Zurich, Switzerland  
felixkl@inf.ethz.ch

Samuel Müller  
ETH Zurich, Switzerland  
smueller@inf.ethz.ch

## ABSTRACT

We show the practical feasibility of monitoring complex security properties using a runtime monitoring approach for metric first-order temporal logic. In particular, we show how a wide variety of security policies can be naturally formalized in this expressive logic, ranging from traditional policies like Chinese Wall and separation of duty to more specialized usage-control and compliance requirements. We also explain how these formalizations can be directly used for monitoring and experimentally evaluate the performance of the resulting monitors.

## Categories and Subject Descriptors

D.2.1 [Software Engineering]: Requirements/Specifications—*Languages*; D.2.5 [Software Engineering]: Testing and Debugging—*Monitors, Tracing*; D.4.6 [Operating Systems]: Security and Protection; J.1 [Computer Applications]: Administrative Data Processing—*Business, Law*

## General Terms

Security, Verification, Legal Aspects

## Keywords

Temporal Logic, Monitoring, Security Policies, Access Control, Separation of Duty, Compliance, Usage Control

## 1. INTRODUCTION

Security policies specify the allowed behavior of organizations and systems. These policies take many forms and are given at varying degrees of abstraction. When the policies are sufficiently formal, they provide a precise description of which behaviors are allowed and, conversely, forbidden. Moreover, their formalization provides a starting point for monitoring and even enforcing policy compliance.

What kinds of formalisms are adequate for these tasks? If we only intend to *formalize* policies, then any sufficiently

expressive logic will do. However, if we also wish to *monitor* whether systems comply to policies, then we are confronted with the standard tradeoff between expressiveness and complexity. The logic should be expressive enough to formalize a wide class of policies but checking policy compliance—satisfiability, in logical terms—should be decidable and efficient enough for practical use.

In this paper, we show that metric first-order temporal logic (MFOTL) is well suited for both of these tasks. MFOTL is an expressive first-order language with metric temporal operators. The first-order fragment is well suited for formalizing relations on system data, while the metric temporal operators can be used to specify properties depending on the times associated with past, present, and even future system events. Moreover, the monitoring approach for MFOTL that we recently presented in [9] can be used to monitor whether system behavior is policy conform. Namely, given a time-stamped sequence of first-order structures, representing system events and when they occur, the monitor determines whether the sequence satisfies, up to the current time, a policy expressed as an MFOTL formula and, if not, reports the violations.

Through a series of examples, we support our claim that MFOTL is well suited for both specifying and monitoring complex, realistic security policies. We formalize security policies from different areas including reporting and transaction requirements in banking, data retention requirements, and requirements based on Chinese Wall and separation-of-duty (SoD) policies. The examples illustrate both standard access-control requirements and more specialized usage-control and policy compliance requirements. An example from financial reporting is the requirement: *every transaction of a customer who has within the last 30 days been involved in a previous suspicious transaction, must be reported as suspicious within 2 days*. We will see how we can formalize such requirements in Section 3 and in Section 4 we describe our experimental results in monitoring such policies.

*Contributions.* We see our contributions as follows. First, our examples provide strong evidence that MFOTL is well suited for specifying a wide range of practically-relevant security policies. The class of policies covered constitute safety properties [5], where compliance can be checked by monitoring system traces. This class encompasses most traditional access-control policies as well as usage-control policies and policies arising in regulatory compliance. As we will see, such policies often combine events and state predicates, relations on data, and complex temporal relationships; all of

Permission to make digital or hard copies of all or part 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 bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SACMAT'10, June 9–11, 2010, Pittsburgh, Pennsylvania, USA.  
Copyright 2010 ACM 978-1-4503-0049-0/10/06 ...\$10.00.

these aspects can be naturally represented by MFOTL formulae interpreted over a point-based, metric semantics.

Second, our experimental results provide evidence of the practical feasibility of our approach for monitoring policy compliance. These are the first experimental results obtained using our monitoring algorithm from [9]. The results show that both the time and space requirements are manageable, as measured on synthetic, but realistic, data streams. They indicate that our monitoring algorithm can be used with modest computing and storage requirements for applications such as monitoring system logs to detect policy violations. Indeed, as our experiments show, events can be processed in milliseconds; the efficiency is such that our monitor could be used online to detect policy violations.

*Related work.* Monitors are a widely used mechanism for enforcing security policies or detecting policy violations [30, 33]. Several approaches have been developed that can be used to monitor such policies, e.g., [11, 22, 31]. However, these approaches mainly focus on access control and are of limited use in application areas such as compliance or business-activity monitoring [16, 18] since they cannot handle the temporal relations that occur in security policies from those areas [22] or at best can handle them in restricted ways [11, 31].

Temporal logics can naturally formalize many of these temporal relationships. For example, Lamport’s temporal logic of action TLA has been used in [38] and a distributed variant of the propositional linear-time temporal logic LTL has been used in [21] to formalize and categorize usage-control policies. Both logics lack some of MFOTL’s features. For example, their temporal operators are non-metric and thus cannot express quantitative timing constraints, which often occur in complex security policies. Furthermore, monitoring and enforcement issues are not addressed. The interval temporal logic ITL has also been used to formally reason about security policies and their enforcement mechanisms [23, 24, 34]. However, the monitors obtained from ITL specifications require that subjects and objects in a system are fixed in advance, i.e., they cannot be created and deleted during runtime. Furthermore, no experimental results for the obtained monitors are given. In [8], a first-order extension of LTL has been used to formalize and reason about policies. However, since the temporal operators are non-metric, the logic does not allow one to formalize timing constraints. Moreover, the reasoning in [8] about policies is limited, since it is carried out by a reduction to propositional LTL, assuming that variables range over a fixed and finite domain.

Summarizing the current state of the art of existing monitoring techniques for temporal logics, most fall short in at least one of the following points: they either only support properties expressed in propositional temporal logics and thus cannot cope with variables ranging over infinite domains [10, 20, 37], do not provide both universal and existential quantification [7, 15, 29, 32] or only in restricted ways [7, 19, 36], do not allow arbitrary quantifier alternation [7], do not provide quantitative temporal operators [20, 32], or cannot simultaneously handle both past and future temporal operators [14, 19, 29, 35]. Our monitoring algorithm for a fragment of MFOTL overcomes most of these limitations. However, this monitoring approach has not previously been evaluated experimentally.

*Organization.* The remainder of this paper is organized as follows. In Section 2, we introduce MFOTL and explain how we use it to formalize and monitor security policies. In Section 3, we formalize different security policies and in Section 4 we present our experimental results in monitoring these policies. Finally, we draw conclusions in Section 5.

## 2. MONITORING METRIC FIRST-ORDER TEMPORAL PROPERTIES

In this section, we define metric first-order temporal logic (MFOTL), with a semantics based on timed temporal structures. We then sketch our runtime monitoring approach for MFOTL, originally presented in [9]. Finally, we describe applications to formalizing and monitoring security policies.

### 2.1 Timed temporal structures

A (first-order) *signature*  $S$  is a triple  $(C, R, a)$ , where  $C$  is a set of constant symbols,  $R$  is a finite set of relation symbols, and  $a : R \rightarrow \mathbb{N}$  associates each relation symbol  $s \in R$  with an arity  $a(s) \geq 1$ . A (relational) *structure*  $D$  over the signature  $S = (C, R, a)$  consists of a domain  $|D| \neq \emptyset$  and *interpretations*  $c^D \in |D|$  and  $r^D \subseteq |D|^{a(r)}$ , for each  $c \in C$  and  $r \in R$ .

A timed temporal structure is a sequence of relational structures over the same signature, where each relational structure is associated with a time stamp. More formally:

**DEFINITION 2.1.** *A timed temporal structure over the signature  $S = (C, R, a)$  is a pair  $(D, \tau)$ , where  $D = (D_0, D_1, \dots)$  and  $\tau = (\tau_0, \tau_1, \dots)$  are infinite sequences of structures  $D_i$  over  $S$  and time stamps  $\tau_i \in \mathbb{N}$  with the following properties:*

- (i)  *$D$  has constant domains, i.e.,  $|D_i| = |D_{i+1}|$ , for all  $i \geq 0$ . We denote the domain by  $|D|$  and require that  $|D|$  is linearly ordered by the relation  $<$ .*
- (ii) *Each constant symbol  $c \in C$  has a rigid interpretation, i.e.,  $c^{D_i} = c^{D_{i+1}}$ , for all  $i \geq 0$ . We denote the interpretation of  $c$  by  $c^D$ .*
- (iii) *The sequence of time stamps  $\tau$  is monotonically increasing and progressing, i.e.,  $\tau_i \leq \tau_{i+1}$ , for all  $i \geq 0$  and for every  $i \geq 0$ , there is some  $j > i$  such that  $\tau_j > \tau_i$ .*

Because the structures in the sequence  $D$  have a possibly infinite domain  $|D|$ , timed temporal structures can represent system executions with an unbounded number of individuals. Many other common system execution models, such as (timed)  $\omega$ -words or database histories, are special cases of this execution model [28].

We remark that a timed temporal structure associates with each time point  $i \in \mathbb{N}$  a structure  $D_i$  and a time stamp  $\tau_i$ . While the sequence  $D = (D_0, D_1, \dots)$  provides a qualitative ordering on the individual structures, the sequence of time stamps  $\tau = (\tau_0, \tau_1, \dots)$  associates each structure  $D_i$  with quantitative time information, where adjacent time points  $i$  and  $i+1$  can have equal time stamps, i.e.,  $\tau_i = \tau_{i+1}$ . The metric temporal operators in MFOTL, which we define in Section 2.2, take both orderings into account.

Finally, we remark that as an alternative to our *point-based* time semantics, one might choose an *interval-based* semantics, where each structure  $D_i$  in  $D$  is associated with an interval from an appropriate time domain. While the

point-based semantics is better suited for modeling system events, the interval-based semantics lends itself better to representing system states. As we will show in Section 3, many interval-based properties can be naturally expressed and monitored using a purely point-based semantics.

## 2.2 Metric first-order temporal logic

We now introduce metric first-order temporal logic [14], MFOTL for short, which extends propositional metric temporal logic [6, 25] with predicates and quantification over an infinite domain of individuals.

*Syntax.* Let  $\mathbb{I}$  be the set of nonempty intervals over  $\mathbb{N}$ . We often write an interval in  $\mathbb{I}$  as  $[c, d]$ , where  $c \in \mathbb{N}$ ,  $d \in \mathbb{N} \cup \{\infty\}$ , and  $c < d$ , i.e.,  $[c, d] := \{a \in \mathbb{N} \mid c \leq a < d\}$ . For the rest of this paper,  $\mathbb{V}$  denotes a countably infinite set of variables, where we assume that  $\mathbb{V} \cap (\mathbb{C} \cup \mathbb{R}) = \emptyset$ , for every signature  $S = (\mathbb{C}, \mathbb{R}, a)$ .

**DEFINITION 2.2.** *The set of (MFOTL) formulae over a signature  $S = (\mathbb{C}, \mathbb{R}, a)$  is given by the grammar*

$$\begin{aligned} \phi ::= & t_1 \approx t_2 \mid t_1 < t_2 \mid r(t_1, \dots, t_{a(r)}) \mid \\ & (\neg\phi) \mid (\phi \wedge \phi) \mid (\exists x. \phi) \mid \\ & (\bullet_I \phi) \mid (\circ_I \phi) \mid (\phi \mathcal{S}_I \phi) \mid (\phi \mathcal{U}_I \phi) \end{aligned}$$

where  $r$  ranges over the elements in  $\mathbb{R}$ ,  $t_1, t_2, \dots$  range over  $\mathbb{V} \cup \mathbb{C}$ ,  $x$  ranges over  $\mathbb{V}$ , and  $I$  ranges over  $\mathbb{I}$ .

A formula  $\phi$  is *temporal* if its outermost connective is a temporal operator, i.e.,  $\phi$  is of the form  $(\bullet_I \psi)$ ,  $(\circ_I \psi)$ ,  $(\psi \mathcal{S}_I \psi')$ , or  $(\psi \mathcal{U}_I \psi')$ , where  $\psi$  and  $\psi'$  are formulae and  $I \in \mathbb{I}$ . A formula  $\phi$  is *bounded* if the interval  $I$  is finite, for every operator  $\mathcal{U}_I$  that occurs in  $\phi$ .

We use standard conventions to omit parentheses, e.g.,  $\neg$  binds stronger than  $\wedge$ , which in turn binds stronger than  $\exists$ . Moreover, temporal operators bind weaker than Boolean connectives and quantifiers. Furthermore, we use standard syntactic sugar like *true* for  $\exists x. x \approx x$ ,  $\theta_1 \vee \theta_2$  for  $\neg(\neg\theta_1 \wedge \neg\theta_2)$ , and  $\forall x. \theta$  for  $\neg\exists x. \neg\theta$ . The classical unary temporal operators are defined as follows  $\blacklozenge_I \theta := \text{true } \mathcal{S}_I \theta$ ,  $\blacksquare_I \theta := \neg \blacklozenge_I \neg\theta$ ,  $\blacklozenge_I \theta := \text{true } \mathcal{U}_I \theta$ , and  $\blacklozenge_I \theta := \neg \blacklozenge_I \neg\theta$ , where  $I \in \mathbb{I}$ . The non-metric variants of the temporal operators are easily defined, e.g.,  $\theta \mathcal{S} \theta' := \theta \mathcal{S}_{[0, \infty)} \theta'$  and  $\square \theta := \square_{[0, \infty)} \theta$ .

*Semantics.* The semantics of MFOTL is defined with respect to timed temporal structures. A *valuation* is a mapping  $v : \mathbb{V} \rightarrow |D|$ . We abuse notation by also applying a valuation  $v$  to constant symbols  $c \in \mathbb{C}$ , with  $v(c) = c^D$ . For a valuation  $v$ , a variable  $x \in \mathbb{V}$ , and  $d \in |D|$ ,  $v[x/d]$  is the valuation that maps  $x$  to  $d$  and the valuation of the other variables is unaltered.

**DEFINITION 2.3.** *Let  $(D, \tau)$  be a timed temporal structure over the signature  $S$ , with  $D = (D_0, D_1, \dots)$  and  $\tau = (\tau_0, \tau_1, \dots)$ ,  $\phi$  a formula over the signature  $S$ ,  $v$  a valuation,*

and  $i \in \mathbb{N}$ . We define the relation  $(D, \tau, v, i) \models \phi$  as follows:

$$\begin{aligned} (D, \tau, v, i) \models t \approx t' & \text{ iff } v(t) = v(t') \\ (D, \tau, v, i) \models t < t' & \text{ iff } v(t) < v(t') \\ (D, \tau, v, i) \models r(t_1, \dots, t_{a(r)}) & \text{ iff } (v(t_1), \dots, v(t_{a(r)})) \in r^{D_i} \\ (D, \tau, v, i) \models (\neg\phi_1) & \text{ iff } (D, \tau, v, i) \not\models \phi_1 \\ (D, \tau, v, i) \models \phi_1 \wedge \phi_2 & \text{ iff } (D, \tau, v, i) \models \phi_1 \text{ and } \\ & (D, \tau, v, i) \models \phi_2 \\ (D, \tau, v, i) \models \exists x. \phi_1 & \text{ iff } (D, \tau, v[x/d], i) \models \phi_1, \\ & \text{ for some } d \in |D| \\ (D, \tau, v, i) \models \bullet_I \phi_1 & \text{ iff } i > 0, \tau_i - \tau_{i-1} \in I, \text{ and } \\ & (D, \tau, v, i-1) \models \phi_1 \\ (D, \tau, v, i) \models \circ_I \phi_1 & \text{ iff } \tau_{i+1} - \tau_i \in I \text{ and } \\ & (D, \tau, v, i+1) \models \phi_1 \\ (D, \tau, v, i) \models \phi_1 \mathcal{S}_I \phi_2 & \text{ iff for some } j \leq i, \tau_i - \tau_j \in I, \\ & (D, \tau, v, j) \models \phi_2, \text{ and } \\ & (D, \tau, v, k) \models \phi_1, \\ & \text{ for all } k \in [j+1, i+1) \\ (D, \tau, v, i) \models \phi_1 \mathcal{U}_I \phi_2 & \text{ iff for some } j \geq i, \tau_j - \tau_i \in I, \\ & (D, \tau, v, j) \models \phi_2, \text{ and } \\ & (D, \tau, v, k) \models \phi_1, \\ & \text{ for all } k \in [i, j) \end{aligned}$$

**EXAMPLE 2.4.** To illustrate our use of MFOTL for formalizing security policies, consider a security policy about publishing business reports within a company. For ease of exposition, we consider here a simplistic policy where reports must be approved before they are published. In Section 3.1, we give a more realistic version of this policy.

We assume that the events for publishing and approving reports are logged in relations. 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  $(D, \tau)$  with  $D = (D_0, D_1, \dots)$  and  $\tau = (\tau_0, \tau_1, \dots)$  of a sequence of logged publishing and approval events is as follows. The only relational symbols in  $D$ 's signature are *publish* and *approve*, both of arity 1. The domain of  $D$  consists of all reports. The  $i$ th structure in  $D$  is time-stamped with  $\tau_i$  and contains the relations  $PUBLISH_i$  and  $APPROVE_i$ .

We express the policy by the MFOTL formula

$$\square \forall f. \text{publish}(f) \rightarrow \blacklozenge \text{approve}(f). \quad (\text{P1})$$

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

$$\square \forall f. \text{publish}(f) \rightarrow \blacklozenge_{[0, 11)} \text{approve}(f). \quad (\text{P2})$$

Note that in this last formula we speak of *time units* when measuring the time difference  $\tau_j - \tau_i$  between the time stamps  $\tau_i$  and  $\tau_j$  of two time points  $i$  and  $j$ , with  $i \leq j$ . The interpretation of a time unit within a system depends on the granularity in which time is tracked. For instance, if the system only time-stamps each time point with the current date, i.e., year, month, and day, then the smallest possible time unit is a day. If the 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 will be clear from the context.

## 2.3 Monitoring algorithm

We now sketch our monitoring algorithm from [9] for a fragment of MFOTL. In the following, let  $(D, \tau)$  be a timed temporal structure over the signature  $S = (C, R, a)$ , representing an infinite execution of some system. We assume here that at each time point the given relations are finite, i.e.,  $r^{D_i}$  is finite, for each  $r \in R$  and  $i \geq 0$ . Furthermore, let  $\phi$  be an MFOTL formula over the signature  $S$  that we want to monitor. We assume that  $\phi$  is of the form  $\Box \psi$ , where  $\psi$  is bounded. From these syntactic restrictions it follows that  $\phi$  describes a safety property [5] and hence violations can be detected in finite time. Note that these assumptions on  $(D, \tau)$  and  $\phi$  are natural in our application domain, namely, monitoring security policies. First, a system generates at each time point  $i$  only finitely many events, which are given as the elements in the relations  $r^{D_i}$ . Second, a policy should be fulfilled at every time point. Third, at each time point, policies usually only relate events from a bounded time window.

Since we want to detect violations, the monitor works with the negated formula, i.e.,  $\Diamond \neg \psi$ , and outputs for each time point the satisfying assignments of  $\neg \psi$ . Note that to identify violations,  $\Diamond \neg \psi$  usually contains free variables. For instance, when monitoring the policy from Example 2.4, formalized by the formula (P1), we monitor its negation, which can be simplified to  $\Diamond \text{publish}(f) \wedge \blacksquare \neg \text{approve}(f)$ . We leave the variable  $f$  free since we wish to learn which reports have been published and not previously approved.

In a nutshell, the monitor works as follows. It sequentially processes the timed temporal structure  $(D, \tau)$  and determines for each time point those elements in  $(D, \tau)$  that violate  $\phi$ . More precisely, the monitor iterates over the structures  $D_i$  and their associated time stamps  $\tau_i$ , where  $i$  is initially 0 and is incremented with each iteration. At each iteration, the monitor incrementally maintains a collection of finite relations for previous time points. Roughly speaking, these relations for each time point  $j \leq i$  store the elements that satisfy the temporal subformulae of  $\neg \psi$  at the time point  $j$ . If the temporal subformula of  $\neg \psi$  refers to future time points, the monitor might need to postpone the construction of such an auxiliary relation to a later iteration, until the processed prefix of  $(D, \tau)$  is long enough to evaluate the subformula at time point  $j$ . However, since  $\neg \psi$  is bounded, we never must postpone such a construction indefinitely. Moreover, the monitor discards auxiliary relations whenever they become irrelevant for detecting further violations. In fact, we prove in [9] that under the additional restriction that the stuttering of equal time stamps is bounded, then in each iteration  $i$  the monitor's space consumption is only polynomial in the cardinality of the so-called *active domain* of the processed prefix, i.e.,  $\text{adom}(D, i) := \bigcup_{j \leq i} \text{adom}(D_j)$ , where  $\text{adom}(D_j) := \{c^D \in |D| \mid c \in C\} \cup \bigcup_{r \in R} \{d_k \in |D| \mid (d_1, \dots, d_{a(r)}) \in r^{D_j} \text{ and } 1 \leq k \leq a(r)\}$ .

Observe that in the above description of the monitor, we implicitly require that the auxiliary relations are finite. In order to guarantee this requirement, we assume that  $\neg \psi$  is *temporal-subformula domain independent* [14], which generalizes the well-known notion of domain-independent queries from classical database theory, see, e.g., [4]. Unfortunately, even for database queries, it is undecidable whether they are domain independent. However, along the lines taken in database theory, we can identify syntactic fragments of

MFOTL that only contain temporal-subformula-domain-independent formulae as described in [9].

## 2.4 Specification methodology

Although more complex, the security policies in Section 3 are formalized analogously to Example 2.4. In the following, 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 RBAC such a well-formedness assumption is that users can only be assigned to existing roles.
3. Specify the security policy as formulae  $\phi_1, \dots, \phi_n$  in the MFOTL fragment for which we can use the monitoring algorithm described in Section 2.3.

The monitors for the formulae  $\phi_1, \dots, \phi_n$  can be used offline to read log files and report policy violations. When the monitors are built into a policy decision point, they can be used online to enforce policies in many cases. We return to this point in Section 3.6.

## 3. FORMALIZATION OF SECURITY POLICIES

In this section, we show how MFOTL can be used to formalize a wide variety of security policies including compliance and history-based access-control policies, which are important for many enterprises and which govern the access and the usage of sensitive data. Later, in Section 4, we use our formalizations for monitoring and auditing such policies.

### 3.1 Approval requirements

Recall from Example 2.4 the policy that whenever a business report is published, its publication must have been previously approved. We formalized this by the MFOTL formula  $\Box \forall f. \text{publish}(f) \rightarrow \blacklozenge \text{approve}(f)$ . This formalization is somewhat simplistic. In realistic settings, we would also require, for example, that the person who publishes the report must be an accountant and the person who approves the publication must be the accountant's manager. Moreover, the approval must happen within a given time window, such as at most 10 days before the publication.

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

*Signature.* The signature consists of the unary relation symbols  $acc_S$  and  $acc_F$ , and the binary relation symbols  $mgr_S$ ,  $mgr_F$ ,  $publish$ , and  $approve$ . Intuitively speaking,  $mgr_S(m, a)$  marks the time when  $m$  becomes  $a$ 's manager and  $mgr_F(m, a)$  marks the corresponding finishing time. Analogously,  $acc_S(a)$  and  $acc_F(a)$  mark the starting and finishing times when  $a$  is an accountant. With these markers, we can simulate state predicates in MFOTL, e.g., the formula  $\underline{acc}(a) := \neg acc_F(a) \mathcal{S} acc_S(a)$  holds at the time points where  $a$  is an accountant. It states that a starting event for  $a$  being an accountant has previously occurred and the corresponding finishing event has not occurred since then. Analogously, we use the formula  $\underline{mgr}(m, a) := \neg mgr_F(m, a) \mathcal{S} mgr_S(m, a)$  for the state predicate that  $m$  is  $a$ 's manager.

*Formalization.* Before we formalize the refined approval policy, we formally state the assumptions about the start and finish events in a timed temporal structure  $(D, \tau)$ . These assumptions reflect the system requirement that those 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  $(D, \tau)$  satisfies the formula

$$\Box \forall a. \neg (acc_S(a) \wedge acc_F(a)). \quad (A1)$$

In other words, we require that  $a$  cannot become an accountant and, at the same time point,  $a$  stops being an accountant. 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  $(D, \tau)$  satisfies the formulae

$$\Box \forall a. acc_F(a) \rightarrow \bullet (\neg acc_F(a) \mathcal{S} acc_S(a)) \quad (A2)$$

and

$$\Box \forall a. acc_S(a) \rightarrow \neg \bullet (\neg acc_F(a) \mathcal{S} acc_S(a)). \quad (A3)$$

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

$$\Box \forall a. \forall f. publish(a, f) \rightarrow \underline{acc}(a) \wedge \diamond_{[0,11]} \exists m. \underline{mgr}(m, a) \wedge approve(m, f). \quad (P3)$$

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 (P3) 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 3.1.** 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 will just introduce the predicate  $\underline{P}$  of arity  $n \geq 1$  and implicitly assume that the signature contains the corresponding  $n$ -ary relation symbols  $P_S$  and  $P_F$ . Moreover, we require that a given timed temporal structure satisfies the corresponding assumptions (A1)–

(A3) for  $\underline{P}$ . Finally, we use  $\underline{P}(x_1, \dots, x_n)$  as an abbreviation of the formula  $\neg P_F(x_1, \dots, x_n) \mathcal{S} P_S(x_1, \dots, x_n)$ .

A syntactically-defined state predicate like being an accountant ( $\underline{acc}(a) = \neg acc_F(a) \mathcal{S} acc_S(a)$ ) does not always capture the intuitive meaning of the corresponding state predicate. For example, consider the formula  $\diamond_{[3,4]} \underline{acc}(a)$ , which not only requires that  $a$  was previously, say at time point  $j$  an accountant, 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  $\diamond_{[3,4]} \underline{acc}(a)$  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 metric temporal operators are relativized by event predicates like in the subformula  $\diamond_{[0,11]} \exists m. \underline{mgr}(m, a) \wedge approve(m, f)$  of (P3). In this example, the existence of a time point in the past, required by the temporal operators  $\diamond_{[0,11]}$ , is constrained by the occurrence of an approval event.

## 3.2 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 [1] and the USA Patriot Act [3].

*Signature.* We use the signature  $(C, R, a)$ , with  $C := \{th\}$ ,  $R := \{trans, auth, report\}$ , and  $a(trans) := 3$ ,  $a(auth) := 2$ , and  $a(report) := 1$ . 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 first formalize the requirement that executed transactions  $t$  of any customers  $c$  must be reported within at most 5 days if the transferred money  $a$  exceeds a given threshold  $th$ :

$$\Box \forall c. \forall t. \forall a. trans(c, t, a) \wedge th < a \rightarrow \diamond_{[0,6]} report(t). \quad (P4)$$

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

$$\Box \forall c. \forall t. \forall a. trans(c, t, a) \wedge th < a \rightarrow \diamond_{[2,21]} \exists e. auth(e, t). \quad (P5)$$

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:

$$\Box \forall t. \forall c. \forall a. trans(t, c, a) \wedge (\diamond_{[0,31]} \exists t'. \exists a'. trans(t', c, a') \wedge \diamond_{[0,6]} report(t')) \rightarrow \diamond_{[0,3]} report(t). \quad (P6)$$

## 3.3 Data retention requirements

In privacy-sensitive areas such as healthcare, the handling of patient data is usually subject to strong restrictions. We

now show how to formalize typical data retention requirements inspired by privacy regulations like the Health Insurance Portability and Accountability Act (HIPAA) [2].

*Signature.* We use the unary relation symbols *hospitalize* and *release*, the binary relation symbols *update* and *delete*, and the ternary relation symbol *copy*. Their intuitive meaning is as follows: *hospitalize*(*p*) holds at those time points when the patient *p* is hospitalized, *release*(*p*) holds when the patient *p* is released from the hospital, *delete*(*d*, *p*) holds when the patient *p*'s health record is deleted from the database *d*, and *copy*(*d*, *d'*, *p*) holds when patient *p*'s health record is copied from the database *d* to the database *d'*. Furthermore, we use the constant symbols *db* and *archive* to refer to the hospital's central database and the archive database, respectively.

*Formalization.* A typical data retention requirement, which a hospital might put into place to comply to HIPAA, is that the patients' health records are only stored for a limited time in the hospital's central database. However, the hospital must archive the health records for auditing and liability purposes.

The following formula states that a patient's health record must be deleted from the hospital database within at most 14 days after the patient has been released from the hospital, unless the patient is readmitted to the hospital within this 14 day time window:

$$\Box \forall p. \text{release}(p) \rightarrow \Diamond_{[0,15]} \text{delete}(db, p) \vee \text{hospitalize}(p). \quad (\text{P7})$$

Furthermore, we require that a health record is archived at most 7 days before it is deleted from the central database:

$$\Box \forall p. \text{delete}(db, p) \rightarrow \blacklozenge_{[0,8]} \text{copy}(db, \text{archive}, p). \quad (\text{P8})$$

Finally, archived data must be stored for at least 8 years:

$$\Box \forall p. \text{copy}(db, \text{archive}, p) \rightarrow \Box_{[0,9]} \neg \text{delete}(\text{archive}, p). \quad (\text{P9})$$

### 3.4 Chinese Wall

The Chinese Wall policy (also known as the Brewer and Nash model) [13] forbids a *subject* *s* to access an *object* *o* when *s* has previously accessed another object in a different *dataset* than *o* and both datasets are in the same *conflict-of-interest class*.

*Signature.* We assume here that every access is logged and stored in a relation and that the events that manipulate the datasets and the conflict-of-interest classes are also logged. Specifically, we consider timed temporal structures over the signature with the binary relation symbols *access* and binary relation symbols for *dataset* and *conflict*. Their intuitive meaning is as follows: *access*(*s*, *o*) holds at those time points when the system allows subject *s* to access object *o*, *dataset*(*o*, *d*) holds when object *o* is in the dataset *d*, and *conflict*(*d*, *d'*) holds when the datasets *d* and *d'* conflict. Recall that underlined predicates like *dataset* are derived from the corresponding start and finish events and that *dataset*(*d*, *d'*) abbreviates  $\neg \text{dataset}_F(d, d') \wedge \text{dataset}_S(d, d')$ .

*Formalization.* We first formalize the assumption that an object cannot be in two datasets at the same time point:

$$\Box \forall o. \forall d. \forall d'. \text{dataset}(o, d) \wedge \text{dataset}(o, d') \rightarrow d \approx d'. \quad (\text{A4})$$

Two further assumptions are that the conflict-of-interest relation is irreflexive and symmetric at every time point; these are easily expressed in MFOTL and we omit their formalization.

We formalize the Chinese Wall policy as

$$\Box \forall s. \forall o. \forall d. \forall d'. \text{access}(s, o) \wedge \text{dataset}(o, d) \wedge (\exists o'. (\blacklozenge \text{access}(s, o') \wedge \text{dataset}(o', d'))) \rightarrow \neg \text{conflict}(d, d'). \quad (\text{P10})$$

A variant of the policy, where we only forbid mutual access to conflicting objects by the same subject for a restricted amount of time, say 3 years, can be formalized as

$$\Box \forall s. \forall o. \forall d. \forall d'. \text{access}(s, o) \wedge \text{dataset}(o, d) \wedge (\exists o'. (\blacklozenge_{[0,4]} \text{access}(s, o') \wedge \text{dataset}(o', d'))) \rightarrow \neg \text{conflict}(d, d'). \quad (\text{P11})$$

REMARK 3.2. Our formalizations allow both datasets and conflict-of-interest classes to change over time. For example, the formula (A4) does not require that an object belongs to different datasets at different time points. This is in contrast to the description in [13], where the datasets and conflict-of-interest classes are assumed to be rigid. However, changes to these relations are realistic since companies might trade objects. In particular, note that the formulae (P10) and (P11) require that the conflict of interest between *d* and *d'* should not exist at the time point where the subject *s* accesses the object *o* and that the object *o'*, which was previously accessed, is in the dataset *d'*. Other interpretations of the Chinese Wall policy are possible here, e.g., the datasets *d* and *d'* should not conflict at all time points between the time points where *s* has accessed the objects *o'* and *o*. Our MFOTL formalization could be adapted for this stricter policy. Furthermore, note that the additional timing constraint in formula (P11) formalizes a more realistic policy than formula (P10). Contracts and nondisclosure agreements are usually only valid for a given time frame.

### 3.5 Separation of duty

Last but not least, we formalize in MFOTL 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) [17]. 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 by means of mutually exclusive roles.

*Signature.* We first describe the signature for formalizing RBAC. It contains unary relation symbols for the state predicates  $\underline{U}$ ,  $\underline{R}$ ,  $\underline{A}$ ,  $\underline{O}$ , and  $\underline{S}$ , the binary relation symbols for the state predicates  $\underline{UA}$ ,  $\underline{user}$ , and  $\underline{roles}$ , and the ternary relation symbols for the state predicate  $\underline{PA}$ . The unary predicates represent the sets of users  $\mathbf{U}$ , roles  $\mathbf{R}$ , actions  $\mathbf{A}$ , objects  $\mathbf{O}$ , and sessions  $\mathbf{S}$  in the RBAC system at a given time point. The predicates  $\underline{UA}$  and  $\underline{PA}$  represent the user-assignment relation  $\mathbf{UA} \subseteq \mathbf{U} \times \mathbf{R}$  and the permission-assignment relation  $\mathbf{PA} \subseteq \mathbf{R} \times \mathbf{A} \times \mathbf{O}$  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.

In order to formalize different SoD policies, our signature also contains binary relation symbols for  $\underline{X}$  and the ternary relation symbol  $exec$ . 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  $a$  in session  $s$ .

*Formalization.* Before we formalize different SoD constraints, we state our assumptions, which reflect system requirements that, intuitively speaking, ensure the desired RBAC semantics of the predicates  $\underline{U}$ ,  $\underline{R}$ ,  $\underline{A}$ , etc. The formula (A5) requires that, at every time point, the predicate  $\underline{UA}$  is correctly typed, i.e., it always only relates currently existing users with currently existing roles:

$$\Box \forall u. \forall r. \underline{UA}(u, r) \rightarrow \underline{U}(u) \wedge \underline{R}(r). \quad (\text{A5})$$

The formulae that ensure that the other predicates are correctly typed at each time point are similar and we omit them. Formulae (A6)–(A9) 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:

$$\Box \forall s. S_S(s) \rightarrow \exists u. \underline{U}(u) \wedge \underline{user}(s, u), \quad (\text{A6})$$

$$\Box \forall s. \forall u. \forall u'. \underline{user}(s, u) \wedge \underline{user}(s, u') \rightarrow u \approx u', \quad (\text{A7})$$

$$\Box \forall s. \forall u. \forall u'. \underline{user}(s, u) \wedge (\bigcirc \underline{user}(s, u')) \rightarrow u \approx u', \quad (\text{A8})$$

and

$$\Box \forall s. \forall u. \forall u'. \neg(\underline{user}_F(s, u) \wedge \underline{user}_S(s, u')). \quad (\text{A9})$$

The formula (A10) ensures that only those roles may be activated in a session that are presently assigned to the user associated with the session:

$$\Box \forall s. \forall r. \underline{roles}_S(s, r) \rightarrow \exists u. \underline{user}(s, u) \wedge \underline{UA}(u, r). \quad (\text{A10})$$

The formula (A11) expresses that actions can only be carried out on objects when the necessary credentials are available:

$$\Box \forall s. \forall a. \forall o. exec(s, a, o) \rightarrow \exists r. \underline{roles}(s, r) \wedge \underline{PA}(r, a, o). \quad (\text{A11})$$

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

$$\Box \forall r. \forall r'. \underline{X}(r, r') \rightarrow \neg \exists u. \underline{UA}(u, r) \wedge \underline{UA}(u, r'). \quad (\text{P12})$$

*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

$$\Box \forall r. \forall r'. \underline{X}(r, r') \rightarrow \neg \exists s. \underline{roles}(s, r) \wedge (\neg S_F(s) \mathcal{S} \underline{roles}(s, r')). \quad (\text{P13})$$

Recall that a session is always associated with the same user and that the user remains constant over the session's lifetime. The formula (P14) formalizes *object-based SoD*, which states that a user may be a member of any two exclusive roles and may also activate them both at the same time (i.e., in the same session), but he must not act on the same

object through both:

$$\begin{aligned} \Box \forall r. \forall r'. \underline{X}(r, r') \rightarrow \\ \neg \exists s. \exists o. (\exists a. exec(s, a, o) \wedge \\ \underline{roles}(s, r) \wedge \underline{PA}(r, a, o)) \wedge \\ (\neg S_F(s) \mathcal{S} \exists a'. exec(s, a', o) \wedge \\ \underline{roles}(s, r') \wedge \underline{PA}(r', a', o)). \end{aligned} \quad (\text{P14})$$

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

### 3.6 Discussion

The examples illustrate how MFOTL can be used to directly formalize different kinds of security policies. To begin with, many security policies have an operational character: an event like accessing an object is only authorized when a condition is satisfied in the present or past, i.e., a provision in the sense of [12], or the event triggers an obligation for the future. Despite the different application domains, almost all the policies considered in Sections 3.1–3.3 are of this form. This is reflected in MFOTL by their common syntactic form,  $\Box \psi \rightarrow \psi'$ , where the antecedent  $\psi$  refers just to the present and the succedent  $\psi'$  either refers just to the past and present or to the present and future.

Not all security policies are formulated in an operational way, however. Often policies specify general constraints on which behaviors are (un)acceptable. The policies in Sections 3.4 and 3.5 provide examples of this. Another example is (P6), which expresses constraints on the past, present, and future. Such policies can be directly formulated in MFOTL where we can freely combine state and event predicates with temporal operators.

We remark that in MFOTL a policy might have syntactically distinct but semantically equivalent formalizations. In fact, in any rich language there are many syntactically equivalent ways of expressing the same semantic property. For example, the formula (P9) is logically equivalent to

$$\Box \forall p. delete(archive, p) \rightarrow \blacksquare_{[0,9]} \neg copy(db, archive, p). \quad (\text{P9}')$$

Moreover, and less obvious, the formula (P7) is logically equivalent to the conjunction

$$\Phi_{\geq 15} \wedge \bigwedge_{1 \leq k < 15} \Phi_{=k}, \quad (\text{P7}')$$

where  $\Phi_{\geq 15}$  is the formula

$$\begin{aligned} \Box \forall p. (\bullet_{[15, \infty)} true) \rightarrow \\ \bullet \neg \left( \neg (delete(db, p) \vee hospitalize(p)) \mathcal{S} release(p) \right) \end{aligned}$$

and for  $k \in \{1, \dots, 14\}$ ,  $\Phi_{=k}$  is the formula

$$\begin{aligned} \Box \forall p. (\bullet_{[k, k+1)} true) \rightarrow \\ \bullet \neg \left( \neg (delete(db, p) \vee hospitalize(p)) \mathcal{S}_{[15-k, \infty)} \right. \\ \left. release(p) \right). \end{aligned}$$

Note that both (P9') and the conjuncts of (P7') also have the syntactic form  $\Box \psi \rightarrow \psi'$ . However, each succedent in the conjuncts of (P7') is triggered not by the occurrence of some particular event, but by the passage of time. Namely, if more than 14 time units have passed between the adjacent time points  $i$  and  $i+1$ , i.e.,  $\tau_{i+1} - \tau_i \geq 15$ , then the succedent in the formula  $\Phi_{\geq 15}$  is checked and if  $\tau_{i+1} - \tau_i = k$ , with  $k \in \{1, \dots, 14\}$ , we check the succedent in the formula  $\Phi_{=k}$ . No

succedent needs to be checked if no time has elapsed between two adjacent time points  $i$  and  $i + 1$ , i.e.,  $\tau_{i+1} - \tau_i = 0$ .

Our focus in this paper is on system monitoring to determine policy compliance. All the given policy formalizations can be directly used for monitoring. We benefit here from the fact that our MFOTL fragment of [9] is one of the most expressive fragments of a temporal logic available for which we can effectively obtain monitors. However, the formalization of a policy has an impact on the efficiency of the obtained monitors. In this respect, (P7) is superior over (P7') because (P7) is smaller in length than (P7') and, more importantly, it has fewer temporal operators. We investigate the efficiency of the obtained monitors in detail in the following section.

We now consider to what extent monitoring can be used for enforcement. With the exceptions of (P6), (P7), and (P7'), all given policies can be enforced by suppressing events (i.e., denying access). Enforceability by execution monitoring and suppressing events is actually independent of the formalization and can be characterized semantically [33]. Concretely, both (P9) and (P9') can be enforced by disallowing delete events whenever there was a corresponding copy event within the given time window. Note that, for each time point, the monitoring algorithm of [9] checks for policy violations only when sufficient information is available, otherwise the check is delayed. Hence it can only be used as an enforcement mechanism when this check always coincides with the event to be suppressed. This is the case for formulae like (P9') but not the equivalent form (P9). The monitor obtained from (P9) detects a violation at a time point  $q$  when processing the first time point  $i$  where the time difference is larger than 8 years, i.e.,  $\tau_i - \tau_q > 8$ . So here syntax matters!

In contrast to (P9), (P7) and thus also (P7') cannot be enforced by suppressing events, since we cannot stop the progression of time. Edit automata [27] might be an appropriate enforcement mechanism here since they can not only suppress events but also insert events. However, this would require first solving the problem of how events should best be inserted, i.e., how such a mechanism should choose between different corrective actions.

## 4. EXPERIMENTAL EVALUATION

We now report on our experiments evaluating the feasibility of monitoring security policies like those from Section 3 using our monitoring algorithm from [9].

All our experiments were carried out on a 1.4 GHz dual core computer with 3 GBytes of RAM.

### 4.1 Setup

We implemented a Java prototype of our monitoring algorithm and evaluated its performance. In particular, we investigated its memory consumption and its incremental updating speed in processing events.

*Methodology.* Recall that the monitoring algorithm iteratively processes timed temporal structures. Since these structures consist of infinite sequences of time stamps and relational structures, our evaluation focuses on the algorithm's performance in the long run. To properly assess its long-run performance, we conducted a steady-state analysis [26], which is a standard evaluation method for estimating the

behavior of non-terminating processes in the limit.<sup>1</sup> For the test cases in which we observed large variances, we did not carry out a steady-state analysis, since the steady-state estimates resulting from such test cases can greatly differ from the real performance. For those test cases, namely, the formulae (P10), (P12), (P13), and (P14), we determined instead the average performance over a finite sample set of finite prefixes of timed temporal structures. The reason that these formulae do not satisfy the statistical requirements for a meaningful steady-state analysis is because these formulae describe an unbounded time window.

Since the actual input processed by a monitor will vary considerably between different organizations, we evaluated the monitoring algorithm on finite prefixes of synthetically-generated timed temporal structures. This allowed for studying the algorithm's behavior under different parameter settings. These are (1) the monitored formula, (2) the sample space, i.e., the different data domains, and (3) the event frequency, i.e., the average number of time points in a given time interval.

*Inputs.* For our experiments, we used the formulae from the previous sections that formalize security policies. For each formula, we synthesized finite prefixes of timed temporal structures over the formula's signature by drawing the time stamps and the elements of the relations from predefined sample spaces using a discrete uniform distribution. We restricted ourselves to relational structures with singleton relations. For example, for the formula (P2), each synthesized relational structure consists of relations for the unary predicates *publish* and *approve*, each consisting of an element drawn from the sample space  $\Omega_{25000}$ , i.e., the numbers between 1 and 25000. The sample space of the time stamps  $\Omega_\tau$  was chosen so that the different lengths of the generated timed temporal structures simulate scenarios with the (approximate) event frequencies 110, 220, ..., 550. For example, for the formula (P2), the event frequency 110, and for monitoring a prefix of length 15000 of a timed temporal structure, we drew time stamps so that they are uniformly distributed over a time period of 1350 time units. Note that (P2) describes a time window of 10 time units and  $15000 \cdot 10 / 110 \approx 1350$ .

Finally, the generated prefixes were modified where needed so that the well-formedness assumptions from Section 3 for the scenario were fulfilled. For example, we removed finish events from relational structures without corresponding start events. In some cases, we also made minor optimizations. For instance, for formula (P4), instead of choosing a concrete value for the constant *th* and drawing values for *a*, we drew a Boolean value indicating whether a relational structure satisfies the atomic formula  $th \prec a$ .

*Measurements.* Since the monitoring algorithm iteratively processes timed temporal structures, we measured the time needed to process a single relational structure in the synthesized finite prefixes of timed temporal structures. To evaluate the algorithm's memory usage, we recorded the cardinal-

<sup>1</sup>In contrast to determining the average performance over a finite sample set, a steady-state analysis has the advantage that the results are not distorted by the warm-up phase. Namely, in this initial phase, the monitoring algorithm usually stores and updates fewer events than in the long run. Furthermore, computing the average performance over a finite sample set just provides information about the observed behaviors. In contrast, a steady-state analysis provides a point estimate of the behavior in the long run.

ities of the auxiliary relations after processing a relational structure. This has the advantage of measuring the algorithm’s space consumption in an abstract way. In particular, the actual memory consumption is dominated by these sizes. Moreover, these sizes are general in the sense that they are representation and implementation independent, i.e., they do not depend on the kind of data elements and the data structures used to store the auxiliary relations.

We also recorded for each time point, the cardinality of the *relevant active domain*, i.e., the set of all data elements in the timed temporal structure that appear in the formula’s time window at the time point. Although these cardinalities are only a rough complexity measure for the processed input prefix, they help us judge the algorithm’s performance better than more simplistic measures like the cardinality of the active domain of the processed prefix or the length of the prefix. In particular, the cardinalities of the relevant active domains relate the incremental update time and the cardinalities of the auxiliary relations to the input prefix of a timed temporal structure with respect to the formula to be monitored. The elements that do not occur in the relevant active domain for a time point are irrelevant for detecting policy violations at that time point.

## 4.2 Results

We summarize the results of our steady-state analysis in Table 1. For each formula, we present a point estimate of the steady-state mean space consumption, where the actual average space consumption lies in the specified interval with a probability of 95%. Table 2 and Figure 1 summarize the experimental results of the average performance analysis for the formulae (P10), (P12), (P13), and (P14). The average values were determined from 12 input prefixes, where we used the same input prefixes for the formulae (P12), (P13), and (P14). Note that the event frequency does not have an impact on the performance of the runtime algorithm for the formulae (P10), (P12), (P13), and (P14), since the temporal operators that occur in these formulae are non-metric.

The results of the steady-state analysis (Table 1) predict low space consumption and running times of the monitoring algorithm in the long-run for our experimental setting. Furthermore, the monitoring algorithm scales well with respect to the event frequency. Observe that the growth rates for *space*, *radom*, *omax*, and *ssmpt* with respect to the event frequency is approximately linear. The results of the average space consumption (Figure 1) for the formulae (P10), (P12), (P13), and (P14) and their average incremental processing times (Table 2) also show that the monitoring algorithm performed well. The average space consumption remained low with respect to the sizes of the active domains. Moreover, for the formulae (P12) and (P14), the average space consumption stabilized after a short initial phase. Not surprisingly, the average work load (space consumptions and incremental processing times) for (P10) are higher than the corresponding work load for (P11), since for the formula (P10), the monitor must take all past events into account. For the formula (P11), only the events in the specified time window are relevant for detecting policy violations. Note that in contrast to the SoD policies given by (P12), (P13), and (P14), the average space consumption for the Chinese Wall policy (P10) is higher with respect to the average sizes of the active domain. One reason for this is that for SoD, the use

**Table 1: Experimental results of steady-state analysis: point estimates ( $\alpha = 95\%$ ) of the steady-state mean space consumption of the monitoring algorithm (*space*), steady-state mean cardinalities of relevant active domains (*radom*), observed maxima (*omax*), and steady-state mean incremental processing times (*ssmpt*, in milliseconds).**

formula	aspect	event frequency				
		110	220	330	440	550
(P2)	<i>space</i>	119±1.6	235±2.7	350±3.7	464±4.6	579±5.5
	<i>radom</i>	276	525	765	1,006	1,238
	<i>omax</i>	165	288	406	529	658
	<i>ssmpt</i>	1.4	2.8	4.2	5.5	6.9
	sample space	$\Omega_{25000}$				
(P3)	<i>space</i>	672±70.5	1,267±135.2	1,857±200.3	2,442±265.4	3,024±331.2
	<i>radom</i>	281	477	661	818	950
	<i>omax</i>	1,208	2,155	3,006	3,988	4,884
	<i>ssmpt</i>	14.1	21.8	26.0	37.7	39.4
	sample space	$\Omega_{20 \times 20 \times 2000}$				
(P4)	<i>space</i>	353±4.4	700±8.7	1,044±12.0	1,386±15.2	1,725±20.7
	<i>radom</i>	404	762	1,098	1,422	1,726
	<i>omax</i>	2,135	3,959	5,172	7,377	8,714
	<i>ssmpt</i>	7.0	13.1	17.9	21.0	29.6
	sample space	$\Omega_{1000 \times 25000 \times 2}$				
(P5)	<i>space</i>	119±1.3	235±2.6	350±3.9	465±5.0	579±5.6
	<i>radom</i>	492	893	1,252	1,583	1,893
	<i>omax</i>	158	282	412	545	659
	<i>ssmpt</i>	1.7	2.8	3.7	4.8	10.4
	sample space	$\Omega_{1000 \times 25000 \times 2 \times 200}$				
(P6)	<i>space</i>	140±2.8	405±9.0	801±19.1	1,334±32.2	1,994±47.8
	<i>radom</i>	404	762	1,098	1,422	1,726
	<i>omax</i>	723	1,270	2,242	3,302	4,360
	<i>ssmpt</i>	2.2	3.5	4.7	6.0	7.6
	sample space	$\Omega_{1000 \times 25000 \times 2}$				
(P7)	<i>space</i>	469±4.5	928±10.0	1,386±14.2	1,840±18.4	2,292±22.9
	<i>radom</i>	400	751	1,081	1,398	1,705
	<i>omax</i>	3,150	6,004	7,801	12,432	17,547
	<i>ssmpt</i>	8.9	16.0	24.5	32.5	40.6
	sample space	$\Omega_{10000 \times 2}$				
(P8)	<i>space</i>	30±0.7	60±1.4	89±2.0	118±2.5	147±2.9
	<i>radom</i>	276	518	753	970	1,184
	<i>omax</i>	52	87	122	148	181
	<i>ssmpt</i>	1.1	1.7	1.9	2.2	2.5
	sample space	$\Omega_{2 \times 10000 \times 2}$				
(P9)	<i>space</i>	148±2.2	292±4.5	436±6.8	579±8.4	721±2.9
	<i>radom</i>	276	518	753	970	1,184
	<i>omax</i>	1,126	2,173	2,667	4,607	5,764
	<i>ssmpt</i>	4.8	9.6	14.4	18.0	21.8
	sample space	$\Omega_{2 \times 10000 \times 2}$				
(P9')	<i>space</i>	30±0.7	60±1.4	89±2.0	118±2.5	147±2.9
	<i>radom</i>	276	518	753	970	1,184
	<i>omax</i>	52	87	122	148	181
	<i>ssmpt</i>	0.8	1.2	1.4	1.8	2.2
	sample space	$\Omega_{2 \times 10000 \times 2}$				
(P11)	<i>space</i>	521±57.4	632±57.5	747±60.8	852±57.9	961±58.1
	<i>radom</i>	261	399	517	627	695
	<i>omax</i>	867	988	1,140	1,222	1,335
	<i>ssmpt</i>	37.6	48.5	61.2	73.3	79.3
	sample space	$\Omega_{100 \times 3000 \times 50}$				

of sessions limits the amount of past data that is stored for detecting policy violations.

Our experimental results also shed light on the relative efficiency of our monitor for different classes of formulae. When comparing the results for the different formulae, we see that the monitoring algorithm performed better for formulae that only refer to the past at each time point, i.e., past operators were handled more efficiently than future operators. Observe that the maximal observed space consumption *omax* is close to the estimated mean steady-state space consumption *space* for formulae that only refer to the past at each time point. For formulae that contain future operators like (P9), these values differ up to a factor of 10. The reason for this is that the monitoring algorithm delays the policy check at a time point when it depends on future events. To

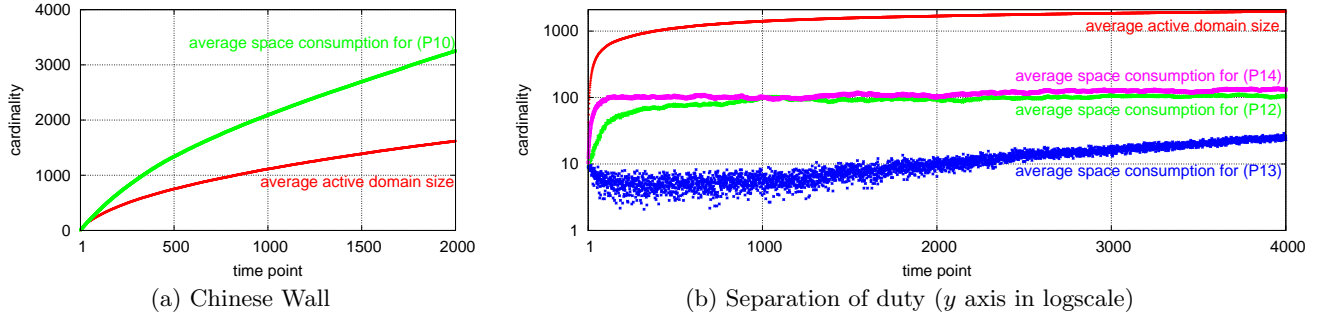


Figure 1: Average space consumption for the formulae (P10), (P12), (P13), and (P14).

Table 2: Observed maxima (*omax*), average incremental processing times (*aipt*, in milliseconds), and sample spaces of the average performance analysis.

formula	<i>omax</i>	<i>aipt</i>	sample space
(P10)	3,267	615.7	$\Omega_{100 \times 3000 \times 50}$
(P12)	169	8.4	$\Omega_{50 \times 50 \times 50 \times 500 \times 500}$
(P13)	56	3.6	
(P14)	248	13.5	

perform this check later, information at the current time point must be stored in the auxiliary relations.

Another observation is that the estimated space consumption in the long run for the formulae that contain state predicates is higher than for formulae without state predicates. This is not surprising since the elements of a state predicate at a time point are stored in an auxiliary relation. Recall from Remark 3.1 that we simulate state predicates by start and finish events. Moreover, since these predicates need to be updated at each iteration, the estimated incremental processing times are also higher but still manageable.

Summing up, our monitoring algorithm can be used to monitor complex policies with modest overhead. Therefore this approach is well suited for auditing and policy enforcement.

## 5. CONCLUSION

Through a series of examples we have shown that the logic MFOTL can be used both to formalize and monitor a wide variety of realistic security policies. For the application domains considered, the formalizations were natural and the runtime monitors had good performance.

We emphasize though that our approach is not a panacea: there is no one silver bullet that covers all applications equally well. Policies outside the scope of MFOTL include those whose formalization is not domain independent or those requiring a more expressive logic. An example of the latter, which involves the aggregation operator for summation, is *a report must be filed within 3 days when all transactions of a trader over the last week sum up to more than \$50 million*. Similarly, our experiments indicate that the algorithm of [9] does not handle all policies equally well since a policy’s syntactic form plays a role in monitoring efficiency. For example, past-time formulae are usually handled more efficiently than future-time formulae. Furthermore, as indicated in Section 3.6, the syntactic form plays a role in

how soon violations are reported. In general, for monitoring those properties formalizable in MFOTL, there may be more efficient, specialized algorithms than ours given in [9]. Still, despite these limitations, MFOTL appears to sit in the sweet-spot between expressivity and complexity: it is a large hammer, applicable to many problems, and has acceptable runtime performance.

We have indicated that, in some cases, our monitors can be used for policy enforcement. As future work we would like to explore how this can be done best and compare the performance against competing approaches. We would also like to carry out concrete case studies in the application domains presented in this paper.

## Acknowledgments

This work was partially supported by the Nokia Research Center, Switzerland, and by Mirasense AG, Switzerland.

## 6. REFERENCES

- [1] Bank Secrecy Act of 1970, 1970. 31 USC 5311-5332 and 31 CFR 103.
- [2] The Health Insurance Portability and Accountability Act of 1996 (HIPAA), 1996. Public Law 104-191.
- [3] USA Patriot Act of 2001, 2001. Public Law 107-56, HR 3162 RDS.
- [4] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases: The Logical Level*. Addison Wesley, 1994.
- [5] B. Alpern and F. B. Schneider. Defining liveness. *Inform. Process. Lett.*, 21(4):181–185, 1985.
- [6] R. Alur and T. A. Henzinger. Logics and models of real time: A survey. In *Proceedings of the REX Workshop on Real Time: Theory in Practice*, volume 600 of *Lect. Notes Comput. Sci.*, pages 74–106, 1992.
- [7] H. Barringer, A. Goldberg, K. Havelund, and K. Sen. Rule-based runtime verification. In *Proceedings of the 5th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI ’04)*, volume 2937 of *Lect. Notes Comput. Sci.*, pages 44–57, 2004.
- [8] A. Barth, A. Datta, J. C. Mitchell, and H. Nissenbaum. Privacy and contextual integrity: Framework and applications. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, pages 184–198, 2006.
- [9] D. Basin, F. Klaedtke, S. Müller, and B. Pfitzmann. Runtime monitoring of metric first-order temporal

- properties. In *Proceedings of the 28th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS '08)*, Dagstuhl Seminar Proceedings, pages 49–60, 2008.
- [10] A. Bauer, M. Leucker, and C. Schallhart. Monitoring of real-time properties. In *Proceedings of the 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS '06)*, volume 4337 of *Lect. Notes Comput. Sci.*, pages 260–272, 2006.
- [11] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati. An access control model supporting periodicity and temporal reasoning. *ACM Trans. Database Syst.*, 23(3):231–285, 1998.
- [12] C. Bettini, S. Jajodia, X. S. Wang, and D. Wijesekera. Provisions and obligations in policy rule management. *J. Netw. Syst. Manage.*, 11(3):351–372, 2003.
- [13] D. F. Brewer and M. J. Nash. The Chinese wall security policy. In *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, pages 206–214, 1989.
- [14] J. Chomicki. Efficient checking of temporal integrity constraints using bounded history encoding. *ACM Trans. Database Syst.*, 20(2):149–186, 1995.
- [15] B. D’Angelo, S. Sankaranarayanan, C. Sánchez, W. Robinson, B. Finkbeiner, H. B. Sipma, S. Mehrotra, and Z. Manna. LOLA: Runtime monitoring of synchronous systems. In *Proceedings of the 12th International Symposium on Temporal Representation and Reasoning (TIME '05)*, pages 166–174, 2005.
- [16] N. Dinesh, A. Joshi, I. Lee, and O. Sokolsky. Checking traces for regulatory conformance. In *Proceedings of the 8th Workshop on Runtime Verification (RV '08)*, volume 5289 of *Lect. Notes Comput. Sci.*, pages 86–103, 2008.
- [17] D. F. Ferraiolo, R. S. Sandhu, S. I. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Trans. Inform. Syst. Secur.*, 4(3):224–274, 2001.
- [18] C. Giblin, A. Y. Liu, S. Müller, B. Pfizmann, and X. Zhou. Regulations expressed as logical models (REALM). In *Proceedings of the 18th Annual Conference on Legal Knowledge and Information Systems (JURIX '05)*, volume 134 of *Frontiers Artificial Intelligence Appl.*, pages 37–48, 2005.
- [19] S. Hallé and R. Villemaire. Browser-based enforcement of interface contracts in web applications with BeepBeep. In *Proceedings of the 21st International Conference on Computer Aided Verification (CAV '09)*, volume 5643 of *Lect. Notes Comput. Sci.*, pages 648–653, 2009.
- [20] K. Havelund and G. Roşu. Efficient monitoring of safety properties. *Int. J. Softw. Tools Technol. Trans.*, 6(2):158–173, 2004.
- [21] M. Hilty, D. Basin, and A. Pretschner. On obligations. In *Proceedings of the 10th European Symposium on Research in Computer Security (ESORICS '05)*, volume 3679 of *Lect. Notes Comput. Sci.*, pages 98–117, 2005.
- [22] S. Jajodia, P. Samarati, and V. S. Subrahmanian. A logical language for expressing authorizations. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 31–42, 1997.
- [23] H. Janicke, A. Cau, F. Siewe, and H. Zedan. Deriving enforcement mechanisms from policies. In *Proceedings of the 8th IEEE International Workshop for Policies for Distributed Systems and Networks (POLICY '07)*, pages 161–172, 2007.
- [24] H. Janicke, A. Cau, and H. Zedan. A note on the formalisation of UCON. In *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies (SACMAT '07)*, pages 163–168, 2007.
- [25] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.
- [26] A. M. Law. *Simulation, Modeling & Analysis*. McGraw-Hill, 4th edition, 2007.
- [27] J. Ligatti, L. Bauer, and D. Walker. Edit automata: enforcement mechanisms for run-time security policies. *Int. J. Inf. Secur.*, 4(1-2):2–16, 2005.
- [28] S. Müller. *Theory and Applications of Runtime Monitoring Metric First-order Temporal Logic*. PhD thesis, Swiss Federal Institute of Technology (ETH) Zurich, 2009.
- [29] D. Nickovic and O. Maler. AMT: A property-based monitoring tool for analog systems. In *Proceedings of the 5th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS '07)*, volume 4763 of *Lect. Notes Comput. Sci.*, pages 304–319, 2007.
- [30] A. Pretschner, M. Hilty, and D. Basin. Distributed usage control. *Commun. ACM*, 49:39–44, 2006.
- [31] C. Ribeiro, A. Zuquete, P. Ferreira, and P. Guedes. SPL: An access control language for security policies with complex constraints. In *Proceedings of the Network and Distributed System Security Symposium (NDSS '01)*, 2001.
- [32] M. Roger and J. Goubault-Larrecq. Log auditing through model-checking. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop (CSFW '01)*, pages 220–234, 2001.
- [33] F. B. Schneider. Enforceable security policies. *ACM Trans. Inform. Syst. Secur.*, 3(1):30–50, 2000.
- [34] F. Siewe, A. Cau, and H. Zedan. A compositional framework for access control policies enforcement. In *Proceedings of the 2003 ACM Workshop on Formal Methods in Security Engineering (FMSE '03)*, pages 32–42, 2003.
- [35] A. P. Sistla and O. Wolfson. Temporal triggers in active databases. *IEEE Trans. Knowl. Data Eng.*, 7(3):471–486, 1995.
- [36] O. Sokolsky, U. Sampaun, I. Lee, and J. Kim. Run-time checking of dynamic properties. *Elec. Notes Theo. Comput. Sci.*, 144(4):91–108, 2006.
- [37] P. Thati and G. Roşu. Monitoring algorithms for metric temporal logic specifications. *Elec. Notes Theo. Comput. Sci.*, 113:145–162, 2005.
- [38] X. Zhang, F. Parisi-Presicce, R. Sandhu, and J. Park. Formal model and policy specification of usage control. *ACM Trans. Inform. Syst. Secur.*, 8(4):351–387, 2005.