Monitoring of Temporal First-order Properties with Aggregations

David Basin, Felix Klaedtke, Srdjan Marinovic, and Eugen Zălinescu

Institute of Information Security, ETH Zurich, Switzerland

Abstract. Compliance policies often stipulate conditions on aggregated data. Current policy monitoring approaches are limited in the kind of aggregations that they can handle. We rectify this as follows. First, we extend metric first-order temporal logic with aggregation operators. This extension is inspired by the aggregation operators common in database query languages like SQL. Second, we provide a monitoring algorithm for this enriched policy specification language. Finally, we experimentally evaluate our monitor's performance.

1 Introduction

Motivation. Compliance policies represent normative regulations, which specify permissive and obligatory actions for agents. Both public and private companies are increasingly required to monitor whether agents using their IT systems, i.e., users and their processes, comply with such policies. For example, US hospitals must follow the US Health Insurance Portability and Accountability Act (HIPAA) and financial services must conform to the Sarbanes-Oxley Act (SOX). First-order temporal logics are not only well-suited for formalizing such regulations, they also admit efficient monitoring. When used online, these monitors observe the agents' actions as they are made and promptly report violations. Alternatively, the actions are logged and the monitor checks them later, such as during an audit. See, for example, [6, 18].

Current logic-based monitoring approaches are limited in their support for expressing and monitoring aggregation conditions. Such conditions are often needed for compliance policies, such as the following simple example from fraud prevention: A user must not withdraw more than $10,000 within a 30 day period from his credit card account. To formalize this policy, we need an operator to express the aggregation of the withdrawal amounts over the specified time window, grouped by the users. In this paper, we address the problem of expressing and monitoring first-order temporal properties built from such aggregation operators.

Solution. First, we extend metric first-order temporal logic (MFOTL) with aggregation operators and with functions. This follows Hella et al.'s [19] extension of first-order logic with aggregations. We also ensure that the semantics of aggregations and grouping operations in our language mimics that of SQL. As illustration, a formalization in our language of the above fraud-detection policy is

$$\Box\, \forall u.\, \forall s.\, [\mathsf{SUM}_a\, a.\, \blacklozenge_{[0,31)}\, \mathit{withdraw}(u, a)](s; u) \to s \preceq 10000\,. \qquad \text{(P0)}$$

The SUM operator, at the current time point, groups all withdrawals, in the past 30 days, for a user u and then sums up their amounts a. The aggregation formula defines a binary relation where the first coordinate is the SUM's result s and the second coordinate is the user u for whom the result is calculated. If the user's sum is greater than 10000, then the policy is violated at the current time point. Finally, the formula states that the aggregation condition must hold for each user and every time point.

A corresponding SQL query for determining the violations with respect to the above policy at a specific time is

$$\mathsf{SELECT\ SUM}(a)\ \mathsf{AS}\ s, u\ \mathsf{FROM}\ W\ \mathsf{GROUP\ BY}\ u\ \mathsf{HAVING\ SUM}(a) > 10000\,,$$

where W is the dynamically created view consisting of the withdrawals of all users within the 30 day time window relative to the given time. Note that the subscript a of the formula's aggregation operator in (P0) corresponds to the a in the SQL query and the third appearance of a in (P0) is implicit in the query, as it is fixed by the view's definition. The second a in (P0) is redundant and emphasizes that the variable a is quantified, i.e., it does not correspond to a coordinate in the resulting relation.

Not all formulas in our language are monitorable. Unrestricted use of logic operators may require infinite relations to be built and manipulated. The second part of our solution, therefore, is a monitorable fragment of our language. It can express all our examples, which represent typical policy patterns, and it allows the liberal use of aggregations and functions. We extend our monitoring algorithm for MFOTL [7] to this fragment. In more detail, the algorithm processes log files sequentially and handles aggregation formulas by translating them into extended relational algebra. Functions are handled similarly to Prolog, where variables are instantiated before functions are evaluated.

We have implemented and evaluated our monitoring solution. For the evaluation, we use fraud-detection policy examples and synthetically generated log files. We first compare the performance of our prototype implementation with the performance of the relational database management system PostgreSQL [22]. Our language is better suited for expressing the policy examples and our prototype's performance is superior to PostgreSQL's performance. This is not surprising since the temporal reasoning must be explicitly encoded in SQL queries and PostgreSQL does not process logged data in the time sequential manner. We also compare our prototype implementation with the stream-processing tool STREAM [2]. Its performance is better than our tool's performance because, in contrast to our tool, STREAM is limited to a restricted temporal pattern for which it is optimized. Although we have not explored performance optimizations for our tool, it is, nevertheless, already efficient enough for practical use.

Contributions. Although aggregations have appeared previously in monitoring, to our knowledge, our language is the first to add expressive SQL-like aggregation operators to a first-order temporal setting. This enables us to express complex compliance policies with aggregations. Our prototype implementation of the presented monitoring algorithm is therefore the first tool to handle such policies, and it does so with acceptable performance.

Related Work. Our MFOTL extension is inspired by the aggregation operators in database query languages like SQL and by Hella et al.'s extension of first-order logic with aggregation operators [19]. Hella et al.'s work is theoretically motivated: they investigate the expressiveness of such an extension in a non-temporal setting. A minor difference between their aggregation operators and ours is that their operators yield terms rather than formulas as in our extension.

Monitoring algorithms for different variants of first-order temporal logics have been proposed by Hallé and Villemaire [18], Bauer at al. [9], and Basin et al. [7]. Except for the counting quantifier [9], none of them support aggregations. Bianculli et al. [10] present a policy language based on a first-order temporal logic with a restricted set of aggregation operators that can only be applied to atomic formulas. For monitoring, they require a fixed finite domain and provide a translation to a propositional temporal logic. Such a translation is not possible in our setting since variables range over an infinite domain. In the context of database triggers and integrity constraints, Sistla and Wolfson [23] describe an integration of aggregation operators into their monitoring algorithm for a first-order temporal logic. Their aggregation operators are different from those presented here in that they involve two formulas that select the time points to be considered for aggregation and they use a database query to select the values to be aggregated from the selected time points.

Other monitoring approaches that support different kinds of aggregations are LarvaStat [13], LOLA [15], EAGLE [4], and an approach based on algebraic alternating automata [16]. These approaches allow one to aggregate over the events in system traces, where events are either propositions or parametrized propositions. They do not support grouping, which is needed to obtain statistics per group of events, e.g., the events generated by the same agent. Moreover, quantification over data elements and correlating data elements is more restrictive in these approaches than in a first-order setting.

Most data stream management systems like STREAM [2] and Gigascope [14] handle SQL-like aggregation operators. For example, in STREAM's query language CQL [3] one selects events in a specified time range, relative to the current position in the stream, into a table on which one performs aggregations. The temporal expressiveness of such languages is weaker than our language, in particular, linear-time temporal operators are not supported.

Organization. In Section 2, we extend MFOTL with aggregation operators. In Section 3, we present our monitoring algorithm, which we evaluate in Section 4. In Section 5, we draw conclusions. Additional details are given in the appendix.

2 MFOTL with Aggregation Operators

2.1 Preliminaries

We use standard notation for sets and set operations. We also use set notation with sequences. For instance, for a set A and a sequence $\bar{s} = (s_1, \ldots, s_n)$, we write $A \cup \bar{s}$ for the union $A \cup \{s_i \mid 1 \leq i \leq n\}$ and we denote the length of $\bar{s}$

by $|\bar{s}|$. 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 *multi-set* M with domain D is a function $M : D \to \mathbb{N} \cup \{\infty\}$. This definition extends the standard one to multi-sets where elements can have an infinite multiplicity. A multi-set is *finite* if $M(a) \in \mathbb{N}$ for any $a \in D$ and the set $\{a \in D \mid M(a) > 0\}$ is finite. We use the brackets $\{\!|$ and $|\!\}$ to specify multi-sets. For instance, $\{\!|2 \cdot \lfloor n/2 \rfloor \mid n \in \mathbb{N}|\!\}$ denotes the multi-set $M : \mathbb{N} \to \mathbb{N} \cup \{\infty\}$ with $M(n) = 2$ if n is even and $M(n) = 0$ otherwise.

An *aggregation operator* is a function from multi-sets to $\mathbb{Q} \cup \{\bot\}$ such that finite multi-sets are mapped to elements of $\mathbb{Q}$ and infinite multi-sets are mapped to $\bot$. Common examples are $\mathsf{CNT}(M) := \sum_{a \in D} M(a)$, $\mathsf{SUM}(M) := \sum_{a \in D} M(a) \cdot a$, $\mathsf{MIN}(M) := \min\{a \in D \mid M(a) > 0\}$, $\mathsf{MAX}(M) := \max\{a \in D \mid M(a) > 0\}$, and $\mathsf{AVG}(M) := \mathsf{SUM}(M)/\mathsf{CNT}(M)$ if $\mathsf{CNT}(M) \neq 0$ and $\mathsf{AVG}(M) := 0$ otherwise, where $M : D \to \mathbb{N} \cup \{\infty\}$ is a finite multi-set. We assume that the given aggregation operators are only applied over the multisets with the domain $\mathbb{Q}$.

2.2 Syntax

A *signature* $\mathcal{S}$ is a tuple $(\mathsf{F}, \mathsf{R}, \iota)$, where F is a finite set of function symbols, R is a finite set of predicate symbols disjoint from F, and the function $\iota : \mathsf{F} \cup \mathsf{R} \to \mathbb{N}$ assigns to each symbol $s \in \mathsf{F} \cup \mathsf{R}$ an arity $\iota(s)$. In the following, let $\mathcal{S} = (\mathsf{F}, \mathsf{R}, \iota)$ be a signature and V a countably infinite set of variables, where $\mathsf{V} \cap (\mathsf{F} \cup \mathsf{R}) = \emptyset$.

Function symbols of arity 0 are called *constants*. Let $\mathsf{C} \subseteq \mathsf{F}$ be the set of constants of $\mathcal{S}$. *Terms* over $\mathcal{S}$ are defined inductively: Constants and variables are terms, and $f(t_1, \ldots, t_n)$ is a term if $t_1, \ldots, t_n$ are terms and f is a function symbol of arity $n > 0$. We denote by $fv(t)$ the set of the variables that occur in the term t. We denote by T the set of all terms over $\mathcal{S}$, and by $\mathsf{T}_\emptyset$ the set of ground terms. A *substitution* θ is a function from variables to terms. We use the same symbol θ to denote its homomorphic extension to terms.

Given a finite set Ω of aggregation operators, the MFOTL_Ω *formulas* over the signature $\mathcal{S}$ are given by the grammar

$$\varphi ::= r(t_1, \ldots, t_{\iota(r)}) \mid (\neg\varphi) \mid (\varphi \vee \varphi) \mid (\exists x.\, \varphi) \mid (\bullet_I\, \varphi) \mid (\varphi\, \mathsf{S}_I\, \psi) \mid [\omega_t\, \bar{z}.\, \varphi](y; \bar{g})\,,$$

where r, t and the t_is, I, and ω range over the elements in R, T, $\mathbb{I}$, and Ω, respectively, x and y range over elements in V, and $\bar{z}$ and $\bar{g}$ range over sequences of elements in V. Note that we overload notation: ω denotes both an aggregation operator and its corresponding symbol. This grammar extends MFOTL's [20] in two ways. First, it introduces aggregation operators. Second, terms may also be built from function symbols and not just from variables and constants. For ease of exposition, we do not consider future-time temporal operators.

We call $[\omega_t\, \bar{z}.\, \psi](y; \bar{g})$ an *aggregation formula*. It is inspired by the homonymous relational algebra operator. Intuitively, by viewing variables as (relation) attributes, $\bar{g}$ are the attributes on which grouping is performed, t is the term on which the aggregation operator ω is applied, and y is the attribute that stores the result. The variables in $\bar{z}$ are ψ's attributes that do not appear in the described relation. We define the semantics in Section 2.3, where we also provide examples.

The set of *free variables* of a formula φ, denoted $fv(\varphi)$, is defined as expected for the standard logic connectives. For an aggregation formula, it is defined as $fv\big([\omega_t\, \bar{z}.\, \varphi](y; \bar{g})\big) := \{y\} \cup \bar{g}$. A variable is *bound* if it is not free. We denote by $\bar{fv}(\varphi)$ the sequence of free variables of a formula φ that is obtained by ordering the free variables of φ by their occurrence when reading the formula from left to right. A formula is *well-formed* if for each of its subformulas $[\omega_t\, \bar{z}.\, \psi](y; \bar{g})$, it holds that (a) $y \notin \bar{g}$, (b) $fv(t) \subseteq fv(\psi)$, (c) the elements of $\bar{z}$ and $\bar{g}$ are pairwise distinct, and (d) $\bar{z} = \bar{fv}(\psi) \setminus \bar{g}$. Note that, given condition (d), the use of one of the sequences $\bar{z}$ and $\bar{g}$ is redundant. However, we use this syntax to make explicit the free and bound variables in aggregation formulas. Throughout the paper, we consider only well-formed formulas.

To omit parenthesis, we assume that Boolean connectives bind stronger than temporal connectives, and unary connectives bind stronger than binary ones, except for the quantifiers, which bind weaker than Boolean ones. As syntactic sugar, we use standard Boolean connectives such as $\varphi \wedge \psi := \neg(\neg\varphi \vee \neg\psi)$, the universal quantifier $\forall x.\, \varphi := \neg\exists x.\, \neg\varphi$, and the temporal operators $\blacklozenge_I\, \varphi := (p \vee \neg p)\, \mathsf{S}_I\, \varphi$, $\blacksquare_I\, \varphi := \neg \blacklozenge_I \neg\varphi$, where $I \in \mathbb{I}$ and p is some predicate symbol of arity 0, assuming without loss of generality that R contains such a symbol. Non-metric variants of the temporal operators are easily defined, e.g., $\blacklozenge\, \varphi := \blacklozenge_{[0,\infty)}\, \varphi$.

2.3 Semantics

We distinguish between predicate symbols whose corresponding relations are *rigid* over time and those that are *flexible*, i.e., their interpretations can change over time. We denote by R_r and R_f the sets of rigid and flexible predicate symbols, where $\mathsf{R} = \mathsf{R}_r \cup \mathsf{R}_f$ with $\mathsf{R}_r \cap \mathsf{R}_f = \emptyset$. We assume R_r contains the binary predicate symbols $\approx$ and $\prec$, which have their expected interpretation, namely, equality and ordering.

A *structure* $\mathcal{D}$ over the signature $\mathcal{S}$ consists of a domain $\mathbb{D} \neq \emptyset$ and interpretations $f^{\mathcal{D}} \in \mathbb{D}^{\iota(f)} \to \mathbb{D}$ and $r^{\mathcal{D}} \subseteq \mathbb{D}^{\iota(r)}$, for each $f \in \mathsf{F}$ and $r \in \mathsf{R}$. A *temporal structure* over the signature $\mathcal{S}$ 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 $\mathcal{S}$ and $\bar{\tau} = (\tau_0, \tau_1, \dots)$ is a sequence of non-negative integers, 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. All structures $\mathcal{D}_i$, with $i \geq 0$, have the same domain, denoted $\mathbb{D}$.
3. Function symbols and rigid predicate symbols have rigid interpretations, that is, $f^{\mathcal{D}_i} = f^{\mathcal{D}_{i+1}}$ and $p^{\mathcal{D}_i} = p^{\mathcal{D}_{i+1}}$, for all $f \in \mathsf{F}$, $p \in \mathsf{R}_r$, and $i \geq 0$. We also write $f^{\mathcal{D}}$ and $p^{\mathcal{D}}$ for $f^{\mathcal{D}_i}$ and $p^{\mathcal{D}_i}$, respectively.

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

A *valuation* is a mapping $v : \mathsf{V} \to \mathbb{D}$. For a valuation v, the variable sequence $\bar{x} = (x_1, \dots, x_n)$, and $\bar{d} = (d_1, \dots, d_n) \in \mathbb{D}^n$, 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 also applying a valuation v to terms. That is, given a structure $\mathcal{D}$, we extend v homomorphically to terms.

For the remainder of the paper, we fix a countable domain $\mathbb{D}$ with $\mathbb{Q} \cup \{\bot\} \subseteq \mathbb{D}$. We only consider a single-sorted logic. One could alternatively have sorts for the different types of elements like data elements and the aggregations. Furthermore, note that function symbols are always interpreted by total functions. Partial functions like division over scalar domains can be extended to total functions, e.g., by mapping elements outside the function's domain to $\bot$. Since the treatment of partial functions is not essential to our work, we treat $\bot$ as any other element of $\mathbb{D}$. Alternative treatments are, e.g., based on multi-valued logics [21].

Definition 1. *Let $(\bar{\mathcal{D}}, \bar{\tau})$ be a temporal structure over the signature $\mathcal{S}$, with $\bar{\mathcal{D}} = (\mathcal{D}_0, \mathcal{D}_1, \dots)$ and $\bar{\tau} = (\tau_0, \tau_1, \dots)$, φ a formula over $\mathcal{S}$, v a valuation, and $i \in \mathbb{N}$. We define the relation $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \varphi$ inductively as follows:*

$$
\begin{array}{lll}
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models p(t_1, \dots, t_{\iota(r)}) & \textit{iff} & \big(v(t_1), \dots, v(t_{\iota(r)})\big) \in p^{\mathcal{D}_i} \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \neg\psi & \textit{iff} & (\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \psi \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \psi \vee \psi' & \textit{iff} & (\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \psi \text{ or } (\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \psi' \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \exists x.\, \psi & \textit{iff} & (\bar{\mathcal{D}}, \bar{\tau}, v[x \mapsto d], i) \models \psi, \text{ for some } d \in \mathbb{D} \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \bullet_I \psi & \textit{iff} & i > 0,\ \tau_i - \tau_{i-1} \in I, \text{ and } (\bar{\mathcal{D}}, \bar{\tau}, v, i-1) \models \psi \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \psi\, \mathsf{S}_I\, \psi' & \textit{iff} & \text{for some } j \le i,\ \tau_i - \tau_j \in I,\ (\bar{\mathcal{D}}, \bar{\tau}, v, j) \models \psi', \\
 & & \text{and } (\bar{\mathcal{D}}, \bar{\tau}, v, k) \models \psi, \text{ for all } k \text{ with } j < k \le i \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models [\omega_t\, \bar{z}.\, \psi](y; \bar{g}) & \textit{iff} & v(y) = \omega(M),
\end{array}
$$

where $M : \mathbb{D} \to \mathbb{N} \cup \{\infty\}$ is the multi-set

$$\Big\{\!\!\Big| v[\bar{z} \mapsto \bar{d}](t) \;\Big|\; (\bar{\mathcal{D}}, \bar{\tau}, v[\bar{z} \mapsto \bar{d}], i) \models \psi, \text{ for some } \bar{d} \in \mathbb{D}^{|\bar{z}|} \Big\}\!\!\Big|.$$

Note that the semantics for the aggregation formula is independent of the order of the variables in the sequence $\bar{z}$.

For a temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$, a time point $i \in \mathbb{N}$, a formula φ, a valuation v, and a sequence $\bar{z}$ of variables with $\bar{z} \subseteq fv(\varphi)$, we define the set

$$[\![\varphi]\!]_{\bar{z},v}^{(\bar{\mathcal{D}},\bar{\tau},i)} := \{\bar{d} \in \mathbb{D}^{|\bar{z}|} \mid (\bar{\mathcal{D}}, \bar{\tau}, v[\bar{z} \mapsto \bar{d}], i) \models \varphi\}\,.$$

We drop the superscript when it is clear from the context. We drop the subscript when $\bar{z} = \bar{fv}(\varphi)$. Note that in this case the valuation v is irrelevant and $[\![\varphi]\!]^{(\bar{\mathcal{D}},\bar{\tau},i)}$ denotes the set of satisfying elements of φ at time point i in $(\bar{\mathcal{D}}, \bar{\tau})$.

With this notation, we illustrate the semantics for aggregation formulas in the case where we aggregate over a variable. We use the same notation as in Definition 1. In particular, consider a formula $\varphi = [\omega_x\, \bar{z}.\, \psi](y; \bar{g})$, with $x \in \mathsf{V}$, and a valuation v. Note that v (and thus also $v[\bar{z} \mapsto \bar{d}]$) fixes the values of the variables in $\bar{g}$ because these are free in φ. The multi-set M is as follows. If $x \notin \bar{g}$, then $M(a) = |\{\bar{d} \in [\![\varphi]\!]_{\bar{z},v} \mid d_j = a\}|$, for any $a \in \mathbb{D}$, where j is the index of x in $\bar{z}$. If $x \in \bar{g}$, then $M(v(x)) = |[\![\varphi]\!]_{\bar{z},v}|$ and $M(a) = 0$, for any $a \in \mathbb{D} \setminus \{v(x)\}$.

Example 2. Let $(\bar{\mathcal{D}}, \bar{\tau})$ be a temporal structure over a signature with a ternary predicate symbol p with $p^{\mathcal{D}_0} = \{(1, b, a), (2, b, a), (1, c, a), (4, c, b)\}$. Moreover, let φ be the formula $[\mathsf{SUM}_x\, x, y.\, p(x, y, g)](s; g)$ and $\bar{z} = (x, y)$. At time point 0,

x	y	g
1	b	a
2	b	a
1	c	a
4	c	b

x	y	g
1	b	
2	b	a
1	c	
4	c	b

Fig. 1. Relation $p^{\mathcal{D}_0}$ from Example 2. The two boxes represent the multi-set M for the two valuations v_1 and v_2, respectively.

for a valuation v_1 with $v_1(g) = a$, we have $[\![p(x, y, g)]\!]_{\bar{z},v_1} = \{(1, b), (2, b), (1, c)\}$ and $M = \{\!|1, 2, 1|\!\}$. For a valuation v_2 with $v_2(g) = b$, we have $[\![p(x, y, g)]\!]_{\bar{z},v_2} = \{(4, c)\}$ and $M = \{\!|4|\!\}$. Finally, for a valuation v_3 with $v_3(g) \notin \{a, b\}$, we have that $[\![p(x, y, g)]\!]_{\bar{z},v_3}$ and M are empty. So the formula φ is only satisfied under a valuation v with $v(s) = 4$ and either $v(g) = a$ or $v(g) = b$. Indeed, we have $[\![\varphi]\!] = \{(4, a), (4, b)\}$. The tables in Figure 1 illustrate this example. We obtain $[\![[\mathsf{SUM}_x\, y, g.\, p(x, y, g)](s; x)]\!] = \{(2, 1), (2, 2), (4, 4)\}$, if we group on the variable x instead of g and $[\![[\mathsf{SUM}_x\, x, y, g.\, p(x, y, g)](s)]\!] = \{(8)\}$, if we do not group.

Example 3. Consider the formula $\varphi = [\mathsf{SUM}_a\, a.\, \psi](s; u)$, where ψ is the formula $\blacklozenge_{[0,31)}\, \mathit{withdraw}(u, a)$. Let $(\bar{\mathcal{D}}, \bar{\tau})$ be a temporal structure with the relations $\mathit{withdraw}^{\mathcal{D}_0} = \{(\text{Alice}, 9), (\text{Alice}, 3)\}$ and $\mathit{withdraw}^{\mathcal{D}_1} = \{(\text{Alice}, 3)\}$, and the timestamps $\tau_0 = 5$ and $\tau_1 = 8$. We have that $[\![\psi]\!]^{(\bar{\mathcal{D}},\bar{\tau},0)} = [\![\psi]\!]^{(\bar{\mathcal{D}},\bar{\tau},1)} = \{(\text{Alice}, 9), (\text{Alice}, 3)\}$ and therefore $[\![\varphi]\!]^{(\bar{\mathcal{D}},\bar{\tau},0)} = [\![\varphi]\!]^{(\bar{\mathcal{D}},\bar{\tau},1)} = \{(12, \text{Alice})\}$. Our semantics ignores the fact that the tuple $(\text{Alice}, 3)$ occurs at both time points 0 and 1. Note that the withdraw events do not have unique identifiers in this example.

To account for multiple occurrences of an event, we can attach to each event additional information to make it unique. For example, assume we have a predicate symbol ts at hand that records the timestamp at each time point, i.e., $ts^{\mathcal{D}_i} = \{\tau_i\}$, for $i \in \mathbb{N}$. For the formula $\varphi' = [\mathsf{SUM}_a\, a.\, \psi'](s; u)$ with $\psi' = \blacklozenge_{[0,31)}\, \mathit{withdraw}(u, a) \wedge ts(t)$, we have that $[\![\varphi']\!]^{(\bar{\mathcal{D}},\bar{\tau},0)} = \{(12, \text{Alice})\}$ and $[\![\varphi']\!]^{(\bar{\mathcal{D}},\bar{\tau},1)} = \{(15, \text{Alice})\}$ because $[\![\psi']\!]^{(\bar{\mathcal{D}},\bar{\tau},0)} = \{(\text{Alice}, 9, 5), (\text{Alice}, 3, 5)\}$ while $[\![\psi']\!]^{(\bar{\mathcal{D}},\bar{\tau},1)} = \{(\text{Alice}, 9, 5), (\text{Alice}, 3, 5), (\text{Alice}, 3, 8)\}$. To further distinguish between withdraw events at time points with equal timestamps, we would need additional information about the occurrence of an event, e.g., information obtained from a predicate symbol tpts that is interpreted as $\mathit{tpts}^{\mathcal{D}_i} = \{(i, \tau_i)\}$, for $i \in \mathbb{N}$.

The multiplicity issue illustrated by Example 3 also appears in databases. SQL is based on a multi-set semantics and one uses the DISTINCT keyword to switch to a set-based semantics. However, it is problematic to define a multi-set semantics for first-order logic, i.e., one that attaches a multiplicity to a tuple $\bar{d} \in \mathbb{D}^{|fv(\varphi)|}$ for how often it satisfies the formula φ instead of a Boolean value. For instance, there are several ways to define a multi-set semantics for disjunction: the multiplicity of $\bar{d}$ for $\psi \vee \psi'$ can be either the maximum or the sum of the multiplicities of $\bar{d}$ for ψ and ψ'. Depending on the choice, standard logical laws become invalid, namely, distributivity of existential quantification or conjunction over disjunction. Defining a multi-set semantics for negation is even more problematic.

$$\frac{p \in \mathsf{R}_f \quad x_1, \ldots, x_{\iota(p)} \in \mathsf{V} \text{ are pairwise distinct}}{p(x_1, \ldots, x_{\iota(p)}) \in \mathcal{F}}\ \mathsf{FLX}$$

$$\frac{\varphi \in \mathcal{F} \quad p \in \mathsf{R}_r \quad \bigcup_{i=1}^{\iota(p)} fv(t_i) \subseteq fv(\varphi)}{\varphi \wedge p(t_1, \ldots, t_{\iota(p)}) \in \mathcal{F}}\ \mathsf{RIG}_{\wedge} \qquad \frac{\varphi \in \mathcal{F} \quad p \in \mathsf{R}_r \quad \bigcup_{i=1}^{\iota(p)} fv(t_i) \subseteq fv(\varphi)}{\varphi \wedge \neg p(t_1, \ldots, t_{\iota(p)}) \in \mathcal{F}}\ \mathsf{RIG}_{\wedge\neg}$$

$$\frac{\varphi \in \mathcal{F} \quad p \in \mathsf{R}_r \quad \bigcup_{i=1, i\neq j}^{\iota(p)} fv(t_i) \subseteq fv(\varphi) \quad t_j \in \mathsf{V} \quad j \in H_p}{\varphi \wedge p(t_1, \ldots, t_{\iota(p)}) \in \mathcal{F}}\ \mathsf{RIG}'_{\wedge}$$

$$\frac{\varphi, \psi \in \mathcal{F}}{\varphi \wedge \psi \in \mathcal{F}}\ \mathsf{GEN}_{\wedge} \qquad \frac{\varphi, \psi \in \mathcal{F} \quad fv(\psi) \subseteq fv(\varphi)}{\varphi \wedge \neg\psi \in \mathcal{F}}\ \mathsf{GEN}_{\wedge\neg} \qquad \frac{\varphi, \psi \in \mathcal{F} \quad fv(\psi) = fv(\varphi)}{\varphi \vee \psi \in \mathcal{F}}\ \mathsf{GEN}_{\vee}$$

$$\frac{\varphi \in \mathcal{F}}{\exists x.\, \varphi \in \mathcal{F}}\ \mathsf{GEN}_{\exists} \qquad \frac{\varphi \in \mathcal{F}}{\bullet_I\, \varphi \in \mathcal{F}}\ \mathsf{GEN}_{\bullet} \qquad \frac{\varphi \in \mathcal{F}}{[\omega_t\, \bar{z}.\, \varphi](y; \bar{g}) \in \mathcal{F}}\ \mathsf{GEN}_{\omega}$$

$$\frac{\varphi, \psi \in \mathcal{F} \quad fv(\varphi) \subseteq fv(\psi)}{\varphi\, \mathsf{S}_I\, \psi \in \mathcal{F}}\ \mathsf{GEN}_{\mathsf{S}} \qquad \frac{\varphi, \psi \in \mathcal{F} \quad fv(\varphi) \subseteq fv(\psi)}{\neg\varphi\, \mathsf{S}_I\, \psi \in \mathcal{F}}\ \mathsf{GEN}_{\neg\mathsf{S}}$$

Fig. 2. The derivation rules defining the fragment $\mathcal{F}$ of monitorable formulas.

3 Monitoring Algorithm

We assume that policies are of the form $\Box\, \forall \bar{x}.\, \varphi$, where φ is an MFOTL$_\Omega$ formula and $\bar{x}$ is the sequence of free variables of φ. The policy requires that $\forall \bar{x}.\, \varphi$ holds at every time point in temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$. In the following, we assume that $(\bar{\mathcal{D}}, \bar{\tau})$ is a *temporal database*, i.e., (1) the domain $\mathbb{D}$ is countably infinite, (2) the relation $p^{\mathcal{D}_i}$ is finite, for each $p \in \mathsf{R}_f$ and $i \in \mathbb{N}$, (3) $p^{\mathcal{D}}$ is a recursive relation, for each $p \in \mathsf{R}_r$, and (4) $f^{\mathcal{D}}$ is computable, for each $f \in \mathsf{F}$. We also assume that the aggregation operators in Ω are computable functions on finite multi-sets.

The inputs of our monitoring algorithm are a formula ψ, which is logically equivalent to $\neg\varphi$, and a temporal database $(\bar{\mathcal{D}}, \bar{\tau})$, which is processed iteratively. The algorithm outputs, again iteratively, the relation $[\![\psi]\!]^{(\bar{\mathcal{D}}, \bar{\tau}, i)}$, for each $i \geq 0$. As ψ and $\neg\varphi$ are equivalent, the tuples in $[\![\psi]\!]^{(\bar{\mathcal{D}}, \bar{\tau}, i)}$ are the policy violations at time point i. Note that we drop the outermost quantifier as we are interested not only in whether the policy is violated. An instantiation of the free variables $\bar{x}$ that satisfies ψ provides additional information about the violations.

3.1 Monitorable Fragment

Not all formulas are effectively monitorable. Consider, for example, the policy $\Box\, \forall x.\, \forall y.\, p(x) \rightarrow q(x, y)$ with the formula $\psi = p(x) \wedge \neg q(x, y)$ that we use for monitoring. There are infinitely many violations for time points i with $p^{\mathcal{D}_i} \neq \emptyset$, namely, any tuple $(a, b) \in \mathbb{D}^2 \setminus q^{\mathcal{D}_i}$ with $a \in p^{\mathcal{D}_i}$. In such a case, $[\![\psi]\!]^{(\bar{\mathcal{D}}, \bar{\tau}, i)}$ is infinite and its elements cannot be enumerated in finite time. We define a fragment of MFOTL$_\Omega$ that guarantees finiteness. Furthermore, the set of violations at each time point can be effectively computed bottom-up over the formula structure. In the following, we treat the Boolean connective $\wedge$ as a primitive.

Definition 4. *The set $\mathcal{F}$ of* monitorable formulas *with respect to $(H_p)_{p \in \mathsf{R}_r}$ is defined by the rules given in Figure 2, where $H_p \subseteq \{1, \ldots, \iota(p)\}$, for each $p \in \mathsf{R}_r$.*

Let ℓ be a label of a rule from Figure 2. We say that a formula $\varphi \in \mathcal{F}$ is of *kind* ℓ if there is a derivation tree for φ having as root a rule labeled by ℓ.

Before describing some of the rules, we first explain the meaning of the set H_p, for $p \in \mathsf{R}_r$ with arity k. The set H_p contains the indexes j for which we can determine the values of the variable x_j that satisfy $p(x_1, \ldots, x_k)$, given that the values of the variables x_i with $i \neq j$ are fixed. Formally, given a temporal database $(\bar{\mathcal{D}}, \bar{\tau})$ and a rigid predicate symbol p of arity $k > 0$, we say that an index j, with $1 \leq j \leq k$, is *effective* for p if for any $\bar{a} \in \mathbb{D}^{k-1}$, the set $\{d \in \mathbb{D} \mid (a_1, \ldots, a_{j-1}, d, a_j, \ldots, a_{k-1}) \in p^{\mathcal{D}}\}$ is finite. For instance, for the rigid predicate $\approx$, the set of effective indexes is $H_{\approx} = \{1, 2\}$. Similarly, for the rigid predicate $\prec_{\mathbb{N}}$, defined as $a \prec_{\mathbb{N}} b$ iff $a, b \in \mathbb{N}$ and $a < b$, we have $H_{\prec_{\mathbb{N}}} := \{1\}$.

We describe the intuition behind the first four rules in Figure 2. The meaning of the other rules should then be obvious. The first rule (FLX) requires that in an atomic formula $p(\bar{t})$ with $p \in \mathsf{R}_f$, the terms t_i are pairwise distinct variables. This formula is monitorable since we assume that p's interpretation is always a finite relation. For the rules ($\mathsf{RIG}_{\wedge}$) and ($\mathsf{RIG}_{\wedge\neg}$), consider formulas of the form $\varphi \wedge p(\bar{t})$ and $\varphi \wedge \neg p(\bar{t})$ with $p \in \mathsf{R}_r$ and $\bigcup_{i=1}^{\iota(p)} fv(t_i) \subseteq fv(\varphi)$. In both cases, the second conjunct restricts on the tuples satisfying φ. A simple example is the formula $p(x, y) \wedge x + 1 \approx y$. If φ is monitorable, such a formula is also monitorable as its evaluation can be performed by filtering out the tuples in $[\![\varphi]\!]$ that do not satisfy the second conjunct. The rule ($\mathsf{RIG}'_{\wedge}$) treats the case where one of the terms t_i is a variable that does not appear in φ. We require here that the index j is effective, so that the values of this variable are determined by the values of the other variables, which themselves are given by the tuples in $[\![\varphi]\!]$. An example is the formula $p(x, y) \wedge z \approx x + y$. The required conditions on t_j are necessary. If j is not effective, then we cannot guarantee finiteness. Consider, e.g., the formula $q(x) \wedge x \not\approx y$. If t_j is neither a variable nor a constant, then we must solve equations to determine the value of the variable that does not occur in φ. Consider, e.g., the formula $q(x) \wedge x \approx y \cdot y$.

The rule (FLX) may seem very restrictive. However, one can often rewrite a formula of the form $p(t_1, \ldots, t_n)$ with $p \in \mathsf{R}_f$ into an equivalent formula in $\mathcal{F}$. For instance, $p(x + 1, x)$ can be rewritten to $\exists y.\, p(y, x) \wedge x + 1 \approx y$. Alternatively, one can add additional rules that handle such cases directly.

We now show that φ's membership in $\mathcal{F}$ guarantees the finiteness of $[\![\varphi]\!]$.

Lemma 5. *Let $(\bar{\mathcal{D}}, \bar{\tau})$ be a temporal database, $i \in \mathbb{N}$ a time point, φ a formula, and H_p the set of effective indexes for p, for each $p \in \mathsf{R}_r$. If φ is a monitorable formula with respect to $(H_p)_{p \in \mathsf{R}_r}$, then $[\![\varphi]\!]^{(\bar{\mathcal{D}}, \bar{\tau}, i)}$ is finite.*

There are formulas like $(x \approx y)\, \mathsf{S}\, p(x, y)$ that describe finite relations but are not in $\mathcal{F}$. However, the policies considered in this paper all fall into the monitorable fragment. They follow the common pattern $\Box \forall \bar{x}, \bar{y}.\, \varphi(\bar{x}, \bar{y}) \wedge c(\bar{x}, \bar{y}) \rightarrow \psi(\bar{y}) \wedge c'(\bar{y})$, where c and c' represent restrictions, i.e., formulas of the form $r(\bar{t})$ and $\neg r(\bar{t})$ with $r \in \mathsf{R}_r$. The formula to be monitored, i.e., $\varphi(\bar{x}, \bar{y}) \wedge c(\bar{x}, \bar{y}) \wedge \neg(\psi(\bar{y}) \wedge c'(\bar{y}))$ is in $\mathcal{F}$ if φ and ψ are in $\mathcal{F}$, and c, c' satisfy the conditions of the (RIG) rules.

Finiteness can also be guaranteed by semantic notions like domain independence or syntactic notions like range restriction, see, e.g., [1] and also [7, 12] for

a generalization of these notions to a temporal setting. If we restrict ourselves to MFOTL without future operators, the range restricted fragment in [7] is more general than the fragment $\mathcal{F}$. This is because, in contrast to the rules in Figure 2, range restrictions are not local conditions, that is, conditions that only relate formulas with their direct subformulas. However, the evaluation procedures in [1, 7, 12] also work in a bottom-up recursive manner. So one still must rewrite the formulas to evaluate them bottom-up. No rewriting is needed for formulas in $\mathcal{F}$. Furthermore, the fragment ensures that aggregation operators are always applied to *finite* multi-sets. Thus, for any $\varphi \in \mathcal{F}$, the element $\bot \in \mathbb{D}$ never appears in a tuple of $[\![\varphi]\!]$, provided that $p^{\mathcal{D}_i} \subseteq D^{\iota(p)}$ and $f^{\mathcal{D}}(\bar{a}) \in D$, for every $p \in \mathsf{R}$, $f \in \mathsf{F}$, $i \in \mathbb{N}$, and $\bar{a} \in D^{\iota(f)}$, where $D = \mathbb{D} \setminus \{\bot\}$.

3.2 Extended Relational Algebra Operators

Our monitoring algorithm is based on a translation of MFOTL$_\Omega$ formulas in $\mathcal{F}$ to extended relational algebra expressions. The translation uses equalities, which we present in Section 3.3, that extend the standard ones [1] expressing the relationship between first-order logic (without function symbols) and relational algebra to function symbols, temporal operators, and group-by operators. In this section, we introduce the extended relational algebra operators.

We start by defining constraints. We assume a given infinite set of variables $Z = \{z_1, z_2, \dots\} \subseteq \mathsf{V}$, ordered by their indices. A *constraint* is a formula $r(t_1, \dots, t_n)$ or its negation, where r is a rigid predicate symbol of arity n and the t_is are constraint terms, i.e., terms with variables in Z. We assume that for each domain element $d \in \mathbb{D}$, there is a corresponding constant, denoted also by d. A tuple $(a_1, \dots, a_k)$ satisfies the constraint $r(t_1, \dots, t_n)$ iff $\bigcup_{i=1}^{n} fv(t_i) \subseteq \{z_1, \dots, z_k\}$ and $(v(t_1), \dots, v(t_n)) \in r^{\mathcal{D}}$, where v is a valuation with $v(z_i) = a_i$, for all $i \in \{1, \dots, k\}$. Satisfaction for a constraint $\neg r(t_1, \dots, t_n)$ is defined similarly.

In the following, let C be a set of constraints, $A \subseteq \mathbb{D}^m$, and $B \subseteq \mathbb{D}^n$. The *selection* of A with respect to C is the m-ary relation

$$\sigma_C(A) := \{\bar{a} \in A \mid \bar{a} \text{ satisfies all constraints in } C\}.$$

The integer i is a *column* in A if $1 \le i \le m$. Let $\bar{s} = (s_1, s_2, \dots, s_k)$ be a sequence of $k \ge 0$ columns in A. The *projection* of A on $\bar{s}$ is the k-ary relation

$$\pi_{\bar{s}}(A) := \left\{(a_{s_1}, a_{s_2}, \dots, a_{s_k}) \in \mathbb{D}^k \,\middle|\, (a_1, a_2, \dots, a_m) \in A\right\}.$$

Let $\bar{s}$ be a sequence of columns in $A \times B$. The *join* and the *antijoin* of A and B with respect to $\bar{s}$ and C is defined as

$$A \bowtie_{\bar{s},C} B := (\pi_{\bar{s}} \circ \sigma_C)(A \times B) \qquad \text{and} \qquad A \rhd_{\bar{s},C} B := A \setminus (A \bowtie_{\bar{s},C} B).$$

Let ω be an operator in Ω, G a set of $k \ge 0$ columns in A, and t a constraint term. The *ω-aggregate* of A on t with grouping by G is the $(k+1)$-ary relation

$$\omega_t^G(A) := \left\{(b, \bar{a}) \,\middle|\, \bar{a} = (a_{g_1}, a_{g_2}, \dots, a_{g_k}) \in \pi_{\bar{g}}(A) \text{ and } b = \omega(M_{\bar{a}})\right\}.$$

Here $\bar{g} = (g_1, g_2, \dots, g_k)$ is the maximal subsequence of $(1, 2, \dots, m)$ such that $g_i \in G$, for $1 \le i \le k$, and $M_{\bar{a}} : \mathbb{D}^{m-k} \to \mathbb{N}$ is the finite multi-set

$$M_{\bar{a}} := \left\{\!\!\left\{(\pi_{\bar{h}} \circ \sigma_{\{d \approx t\} \cup D})(A) \,\middle|\, d \in \mathbb{D}\right\}\!\!\right\},$$

where $\bar{h}$ is the maximal subsequence of $(1, 2, \dots, m)$ with no element in G and $D := \{a_i \approx z_{g_i} \mid 1 \le i \le k\}$.

3.3 Translation to Extended Relational Algebra

Let $(\bar{\mathcal{D}}, \bar{\tau})$ be a temporal database, $i \in \mathbb{N}$, and $\varphi \in \mathcal{F}$. We express $[\![\varphi]\!]^{(\bar{\mathcal{D}},\bar{\tau},i)}$ in terms of the generalized relational algebra operators defined in Section 3.2.

Kind (FLX). This case is straightforward: for a predicate symbol $p \in \mathsf{R}_f$ of arity n and pairwise distinct variables $x_1, \ldots, x_n \in \mathsf{V}$,

$$[\![p(x_1, \ldots, x_n)]\!]^{(\bar{\mathcal{D}},\bar{\tau},i)} = p^{\mathcal{D}_i}\,.$$

Kind ($\mathsf{RIG}_\wedge$). Let $\psi \wedge p(t_1, \ldots, t_n)$ be a formula of kind ($\mathsf{RIG}_\wedge$). Then

$$[\![\psi \wedge p(t_1, \ldots, t_n)]\!]^{(\bar{\mathcal{D}},\bar{\tau},i)} = \sigma_{\{p(\theta(t_1),\ldots,\theta(t_n))\}}\big([\![\psi]\!]^{(\bar{\mathcal{D}},\bar{\tau},i)}\big)\,,$$

where the substitution $\theta : fv(\psi) \to \{z_1, \ldots, z_{|fv(\psi)|}\}$ is given by $\theta(x) = z_j$ with j the index of x in $\bar{fv}(\psi)$. For instance, if $\varphi \in \mathcal{F}$ is the formula $\psi(x, y) \wedge (x - y) \ mod\ 2 \approx 0$ then $[\![\varphi]\!]^{(\bar{\mathcal{D}},\bar{\tau},i)} = \sigma_{\{(z_1 - z_2)\ mod\ 2 \approx 0\}}[\![\psi]\!]^{(\bar{\mathcal{D}},\bar{\tau},i)}$.

Kind (GEN_S). Let $\psi\, \mathsf{S}_I\, \psi'$ be a formula of kind (GEN_S) with $\bar{fv}(\psi) = (y_1, \ldots, y_n)$ and $\bar{fv}(\psi') = (y'_1, \ldots, y'_\ell)$. Then

$$[\![\psi\, \mathsf{S}_I\, \psi']\!]^{(\bar{\mathcal{D}},\bar{\tau},i)} = \bigcup_{j \in \{i' \mid i' \le i,\ \tau_i - \tau_{i'} \in I\}} \Big([\![\psi']\!]^{(\bar{\mathcal{D}},\bar{\tau},j)} \bowtie_{\bar{s},C} \big(\bigcap_{k \in \{j+1,\ldots,i\}} [\![\psi]\!]^{(\bar{\mathcal{D}},\bar{\tau},k)}\big)\Big)\,,$$

where (a) $\bar{s} = (1, \ldots, n, n + i_1, \ldots, n + i_\ell)$ with i_j such that $(i_1, \ldots, i_\ell)$ is the maximal subsequence of $(1, \ldots, \ell)$ with $y'_{i_j} \notin fv(\psi)$ and (b) $C = \{z_j \approx z_{n+h} \mid y_j = y'_h,\ 1 \le j \le n, \text{ and } 1 \le h \le \ell\}$. For instance, for $\bar{fv}(\psi) = (x, y, z)$ and $\bar{fv}(\psi') = (z, z', x)$, we have $\bar{s} = (1, 2, 3, 5)$ and $C = \{z_1 \approx z_6, z_3 \approx z_4\}$.

Kind (GEN_ω). Let $[\omega_t\, \bar{z}'.\, \psi](y; \bar{g})$ be a formula of kind (GEN_ω). It holds that

$$[\![[\omega_t\, \bar{z}'.\, \psi](y; \bar{g})]\!]^{(\bar{\mathcal{D}},\bar{\tau},i)} = \omega^G_{\theta(t)}\big([\![\psi]\!]^{(\bar{\mathcal{D}},\bar{\tau},i)}\big)\,,$$

where $\bar{fv}(\psi) = (y_1, \ldots, y_n)$, for some $n \ge 0$, $G = \{i \mid y_i \in \bar{g}\}$, and $\theta : fv(\psi) \to \{z_1, \ldots, z_n\}$ is given by $\theta(x) = z_j$ with j being the index of x in $\bar{fv}(\psi)$. For instance, for $[\mathsf{SUM}_{x+y}\, x, y.\, p(x, y, z)](s; z)$, we have $G = \{3\}$ and $\theta(t) = z_1 + z_2$.

Other kinds. The case for ($\mathsf{RIG}_{\wedge\neg}$) is similar to the one for ($\mathsf{RIG}_\wedge$). The cases for ($\mathsf{GEN}_\wedge$), ($\mathsf{GEN}_{\wedge\neg}$), and ($\mathsf{GEN}_{\neg\mathsf{S}}$) are similar to the one for (GEN_S). The cases for ($\mathsf{GEN}_{\wedge\neg}$) and ($\mathsf{GEN}_{\neg\mathsf{S}}$) use the antijoin instead of the join. The cases for ($\mathsf{GEN}_\vee$), ($\mathsf{GEN}_\exists$), ($\mathsf{GEN}_\bullet$) are obvious. Additional details are in the appendix of the full version of the paper available at the authors' web pages.

3.4 Algorithmic Realization

Our monitoring algorithm for MFOTL$_\Omega$ is inspired by those in [7, 8, 11]. We only sketch it here. Further details are given in the appendix.

For a formula $\psi \in \mathcal{F}$, the algorithm iteratively processes the temporal database $(\bar{\mathcal{D}}, \bar{\tau})$. At each time point i, it calls the procedure eval to compute $[\![\psi]\!]^{(\bar{\mathcal{D}},\bar{\tau},i)}$. The input of eval at time point i is the formula ψ, the time point i with its timestamp τ_i, and the interpretations of the flexible predicate

$$\Box \forall u.\, \forall s.\, [\mathsf{SUM}_a\, a, i.\, \blacklozenge_{[0,31)}\, \psi(u,a,i)](s;u) \to s \preceq 10000 \tag{P1}$$

$$\Box \forall u.\, \forall s.\, [\mathsf{SUM}_a\, a, i.\, \blacklozenge_{[0,31)}\, \psi(u,a,i)](s;u) \wedge (\neg \mathit{limit\_off}(u)\ \mathsf{S}\ \mathit{limit\_on}(u)) \to s \preceq 10000 \tag{P2}$$

$$\Box \forall u.\, \forall s.\, \forall m.\, [\mathsf{AVG}_a\, a, i.\, \blacklozenge_{[0,91)}\, \psi(u,a,i)](s;u) \wedge [\mathsf{MAX}_a\, a.\, \blacklozenge_{[0,8)}\, \mathit{withdraw}(u,a)](m;u) \to m \preceq 2 \cdot s \tag{P3}$$

$$\Box \forall s.\, [\mathsf{AVG}_u\, u, c.\, [\mathsf{CNT}_i\, a, i.\, \blacklozenge_{[0,31)}\, \psi(u,a,i)](c;u)](s) \to s \preceq 150 \tag{P4}$$

$$\Box \forall u.\, \forall c.\, [\mathsf{CNT}_j\ v, p, j.\, [\mathsf{AVG}_a\, a, i.\, \blacklozenge_{[0,31)}\, \psi(u,a,i)](v;u) \wedge \blacklozenge_{[0,31)}\, \psi(u,p,j) \wedge 2 \cdot v \prec p](c;u) \to c \preceq 5 \tag{P5}$$

Fig. 3. Policy formalizations, where $\psi(u,a,i)$ abbreviates $\mathit{withdraw}(u,a) \wedge \mathit{ts}(i)$.

symbols, i.e., $r^{\mathcal{D}_i}$, for each $r \in \mathsf{R}_f$. Note that $\bar{\mathcal{D}}$'s domain and the interpretations of the rigid predicate symbols and the function symbols, including the constants, do not change over time. We assume that they are fixed in advance.

The computation of $[\![\psi]\!]^{(\bar{\mathcal{D}},\bar{\tau},i)}$ is by recursion over ψ's formula structure and is based on the equalities in Section 3.3. Note that extended relational algebra operators have standard, efficient implementations [17], which can be used to evaluate the expressions on the right-hand side of the equalities from Section 3.3.

To accelerate the computation of $[\![\psi]\!]^{(\bar{\mathcal{D}},\bar{\tau},i)}$, the monitoring algorithm maintains state for each temporal subformula, storing previously computed intermediate results. The monitor's state is initialized by the procedure init and updated in each iteration by the procedure eval. For subformulas of the form $\bullet_I\, \psi'$, we store at time point $i > 0$, the tuples that satisfy ψ' at time-point $i-1$, i.e., the relation $[\![\psi']\!]^{(\bar{\mathcal{D}},\bar{\tau},i-1)}$. For formulas of the form $\psi_1\ \mathsf{S}_{[a,b)}\ \psi_2$, we store at time point i, the list of relations $[\![\psi_2]\!]^{(\bar{\mathcal{D}},\bar{\tau},j)} \bowtie_{\bar{s},C} \big(\bigcap_{j<k\le i} [\![\psi_1]\!]^{(\bar{\mathcal{D}},\bar{\tau},k)}\big)$ with $j \le i$ such that $\tau_i - \tau_j < b$, where $\bar{s}$ and C are defined as in Section 3.3. By storing these relations, the subformulas ψ', ψ_1, and ψ_2 need not be evaluated again at time points $j < i$ during the evaluation of ψ at time point i. Further optimizations are possible. For instance, one can store and reuse some of the intermediate relations used for computing the relation $[\![\psi_1\ \mathsf{S}_{[a,b)}\ \psi_2]\!]^{(\bar{\mathcal{D}},\bar{\tau},i)}$ from the relations stored in the previously mentioned list. Also, when $a = 0$ and $b = \infty$, it is sufficient to store the resulting relation from the previous time point, as $[\![\psi_1\ \mathsf{S}\ \psi_2]\!]^{(\bar{\mathcal{D}},\bar{\tau},i)} = [\![\psi_2]\!]^{(\bar{\mathcal{D}},\bar{\tau},i)} \cup \big([\![\psi_1\ \mathsf{S}\ \psi_2]\!]^{(\bar{\mathcal{D}},\bar{\tau},i-1)} \bowtie [\![\psi_1]\!]^{(\bar{\mathcal{D}},\bar{\tau},i)}\big)$.

Theorem 6. *Let $(\bar{\mathcal{D}},\bar{\tau})$ be a temporal database, $i \in \mathbb{N}$, and $\psi \in \mathcal{F}$. The procedure* $\mathsf{eval}(\psi,\ i,\ \tau_i,\ \Gamma_i)$ *returns the relation $[\![\psi]\!]^{(\bar{\mathcal{D}},\bar{\tau},i)}$, whenever* $\mathsf{init}(\psi)$, $\mathsf{eval}(\psi,\ 0,\ \tau_0,\ \Gamma_0)$, …, $\mathsf{eval}(\psi,\ i-1,\ \tau_{i-1},\ \Gamma_{i-1})$ *were called previously in this order, where $\Gamma_j = (p^{\mathcal{D}_j})_{p\in\mathsf{R}_f}$ is the family of interpretations of flexible predicates at j, for every time point $j \in \mathbb{N}$.*

4 Experimental Evaluation

We compare our prototype implementation, which extends our monitoring tool MonPoly [5] for MFOTL, with the relational database PostgreSQL [22] and

Tab. 1. Running times (STREAM / MonPoly extension / PostgreSQL) in seconds.

policy \ time span	400	800	1200	1600	2000
(P1)	8 / 9 / 76	9 / 19 / 279	11 / 29 / 610	12 / 39 / 1065	14 / 48 / 1650
(P2)	21 / 10 / 247	23 / 20 / 1646	24 / 30 / 5233	26 / 40 / 11989	28 / 50 / 23260
(P3)	† / 22 / 168	† / 44 / 604	† / 66 / 1230	† / 88 / 2251	† / 110 / 3458
(P4)	12 / 9 / 75	15 / 19 / 280	15 / 29 / 612	17 / 38 / 1068	19 / 48 / 1650
(P5)	24 / 76 / 83	33 / 157 / 337	41 / 234 / 745	49 / 313 / 1351	59 / 395 / 2099

the stream-processing tool STREAM [2]. For our evaluation, we consider the following five policies. Figure 3 contains their MFOTL$_\Omega$ formalizations.

(P1) The sum of withdrawals of each user over the last 30 days does not exceed the limit of $10,000.

(P2) Similar to (P1), except that the withdrawals must not exceed $10,000 only when the flag for checking the limit is set.

(P3) The maximal withdrawal of each user over the last seven days must be at most be twice as large as the average of the user's withdrawals over the last 90 days.

(P4) The average of the number of withdrawals of all users over the last 30 days should be less than a given threshold of 150.

(P5) For each user, the number of peaks over the last 30 days does not exceed a threshold of 5, where a peak is a value at least twice the average over some time window.

Note that in the formalization of the policy (P2), the event $\mathit{limit\_on}(u)$ sets the limit flag for the user u, while $\mathit{limit\_off}(u)$ unsets it.

We use synthetically generated logs[1] with different time spans (in days). The logs contain withdraw events from 500 users, except for (P5), for which we consider only 100 users. Each user makes on average five withdrawals per day. Table 1 shows the running times of the three tools on a standard desktop computer with 8 GB of RAM and an Intel Core i5 CPU with 2.67 GHz. The SQL queries for PostgreSQL and the CQL queries for STREAM were manually obtained from the corresponding MFOTL$_\Omega$ formulas. For the considered policies and logs, the semantic differences between the languages are not substantial. In particular, the tools output the same violations. PostgreSQL's running times only account for the query evaluation, performed once per log file, and not for populating the database. For MAX aggregations, STREAM aborts with a runtime error, and we mark this with the symbol †.

Note that the formulas in Figure 3 vary in their complexity: e.g., they contain different numbers of aggregations and temporal operators, with time windows of different sizes. STREAM and our tool scale linearly on these examples with respect to the time spans of the logs. This is not the case for PostgreSQL. Overall, our tool's performance is between STREAM's and PostgreSQL's on these examples.

[1] Our prototype, the formulas, and the input data are available as an archive at `https://projects.developer.nokia.com/MonPoly/files/rv13-experiments.tgz`.

We first focus on the performance of our tool. (P2) is only slightly slower to monitor than (P1) because the relations for the additional subformula are not large: they contain around 50 tuples, as the limit flag is toggled for each user, on average, every 10 days. (P3) takes longer to monitor for two reasons. First, it contains a significantly larger time window. Second, the join of two relations is computed, which is also the case for (P5). For (P3), the two input relations and the output relation each have size n, where n is the number of users. For (P5), the size of the input relations is approximately $31mn$, where m is the average number of withdrawals per day of a user, while the output relation is approximately of size 31^2m^2n. This explains why (P5) takes longer to monitor than (P3). Since aggregating over a relation does not increase its size, the nesting of aggregation operators has only a minor impact on the running times, compare (P1) and (P4).

PostgreSQL performs worst in these experiments. This is not surprising as PostgreSQL is not designed for this application domain. In particular, PostgreSQL has no support for temporal reasoning and we must treat time as just another data value. In more detail, we load log files into database tables that have two additional attributes to represent the time point and the timestamp of an event occurrence, and we adapt the standard embedding of temporal logic into first-order logic to represent MFOTL_{Ω} formulas as SQL queries. Treating time as data has the following disadvantages. First, it is not suited for online processing of events: query evaluation does not scale, because the query must be reevaluated on the entire database each time new events are added. Second, even for offline processing (as done in our experiments), the query evaluation procedure does not take advantage of the temporal ordering of events. This deficiency is most evident when evaluating the SQL query for (P2).

In contrast to PostgreSQL, STREAM is designed for online event processing. However, temporal reasoning in STREAM is limited. In particular, CQL's only temporal construct collects all event tuples within a specified time range relative to the current time. It roughly corresponds[2] to the $\blacklozenge_I$ operator in MFOTL_{Ω}, where I is of the form $[0, t)$ with $t \in \mathbb{N} \cup \{\infty\}$. We cannot select only tuples from a time window that is strictly in the past. It is therefore not clear how to handle temporal properties of the form $\blacklozenge_I \varphi$ with $0 \notin I$. It is also not clear how to handle nested temporal operators as this also requires handling time windows that do not contain the current time point. Finally, it is also not obvious how to check that certain event patterns happen at every time point in a given time window. Consider, e.g., the policy stating that a user may not make large withdrawals if he is continuously in an over-withdrawn state during the last seven days. In MFOTL_{Ω}, the policy is naturally expressed as

$$\square \forall u. \left(\blacksquare_{[0,8)} (\neg \mathit{out\text{-}debt}(u) \mathsf{S} \mathit{in\text{-}debt}(u)) \right) \rightarrow \neg \exists a. \mathit{withdraw}(u, a) \wedge a \succ 1000 \,.$$

Note that the subformula $\neg \mathit{out\text{-}debt}(u) \mathsf{S} \mathit{in\text{-}debt}(u)$ can be encoded in CQL by requiring for each user u that at the current time the total number of $\mathit{out\text{-}debt}(u)$

[2] CQL's time model differs from that of MFOTL_{Ω}. In CQL, there is no notion of time point and query evaluation is performed for each timestamp $\tau \in \mathbb{N}$. Furthermore, CQL has a multi-set semantics.

events is smaller than the total number of *in-debt*(u) events. We have used such an encoding for (P2). We remark that the addition to (P1) of the since subformula in (P2) has a larger impact on STREAM's performance than on our tool.

While MFOTL$_\Omega$ has a richer tool set than CQL to express temporal patterns, STREAM's performance is consistently better than our tool's. Nevertheless, the differences are not as large as one might expect for a prototype implementation. Our prototype has not yet been systematically optimized. We expect substantial performance improvements by carefully adapting data structures and query evaluation techniques used in databases and stream processing.

5 Conclusion

Existing logic-based policy monitoring approaches offer little support for aggregations. To rectify this shortcoming we extended metric first-order temporal logic with expressive SQL-like aggregation operators and presented a monitoring algorithm for this language. Our experimental results for a prototype implementation of the algorithm are promising. The prototype's performance is in the reach of optimized stream-processing tools, despite its richer input language and its lack of systematic optimization. As future work, we will investigate performance optimizations for our monitor. In general, it remains to be seen how logic-based monitoring approaches can benefit from the techniques used in stream processing.

Acknowledgements. This work was partially supported by the Zurich Information Security and Privacy Center. It represents the views of the authors.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.
2. A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, R. Motwani, I. Nishizawa, U. Srivastava, D. Thomas, R. Varma, and J. Widom. STREAM: The Stanford stream data manager. *IEEE Data Eng. Bull.*, 26(1):19–26, 2003.
3. A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15(2):121–144, 2006.
4. 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)*, vol. 2937 of *LNCS*, pp. 44–57, 2004.
5. D. Basin, M. Harvan, F. Klaedtke, and E. Zălinescu. MONPOLY: Monitoring usage-control policies. In *Proceedings of the 2nd International Conference on Runtime Verification (RV'11)*, vol. 7186 of *LNCS*, pp. 360–364, 2012.
6. D. Basin, M. Harvan, F. Klaedtke, and E. Zălinescu. Monitoring data usage in distributed systems. *IEEE Trans. Software Eng.*, to appear.
7. D. Basin, F. Klaedtke, S. Müller, and B. Pfitzmann. Runtime monitoring of metric first-order temporal properties. In *Proceedings of the 28th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'08)*, vol. 2 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 49–60, 2008.

8. D. Basin, F. Klaedtke, and E. Zălinescu. Algorithms for monitoring real-time properties. In *Proceedings of the 2nd International Conference on Runtime Verification (RV'11)*, vol. 7186 of *LNCS*, pp. 260–275, 2012.
9. A. Bauer, R. Goré, and A. Tiu. A first-order policy language for history-based transaction monitoring. In *Proceedings of the 6th International Colloquium on Theoretical Aspects of Computing (ICTAC'09)*, vol. 5684 of *LNCS*, pp. 96–111, 2009.
10. D. Bianculli, C. Ghezzi, and P. S. Pietro. The tale of SOLOIST: A specification language for service compositions interactions. In *Proceedings of the 9th International Symposium on Formal Aspects of Component Software (FACS'12)*, vol. 7684 of *LNCS*, pp. 55–72, 2013.
11. J. Chomicki. Efficient checking of temporal integrity constraints using bounded history encoding. *ACM Trans. Database Syst.*, 20(2):149–186, 1995.
12. J. Chomicki, D. Toman, and M. H. Böhlen. Querying ATSQL databases with temporal logic. *ACM Trans. Database Syst.*, 26(2):145–178, 2001.
13. C. Colombo, A. Gauci, and G. J. Pace. LarvaStat: Monitoring of statistical properties. In *Proceedings of the 1st International Conference on Runtime Verification (RV'10)*, vol. 6418 of *LNCS*, pp. 480–484, 2010.
14. C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk. Gigascope: A stream database for network applications. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pp. 647–651, 2003.
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)*, pp. 166–174, 2005.
16. B. Finkbeiner, S. Sankaranarayanan, and H. Sipma. Collecting statistics over runtime executions. *Form. Method. Syst. Des.*, 27(3):253–274, 2005.
17. H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database systems: The complete book*. Pearson Education, 2009.
18. S. Hallé and R. Villemaire. Runtime enforcement of web service message contracts with data. *IEEE Trans. Serv. Comput.*, 5(2):192–206, 2012.
19. L. Hella, L. Libkin, J. Nurmonen, and L. Wong. Logics with aggregate operators. *J. ACM*, 48(4):880–907, 2001.
20. R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Syst.*, 2(4):255–299, 1990.
21. O. Owe. Partial logics reconsidered: A conservative approach. *Form. Asp. Comput.*, 5(3):208–223, 1993.
22. PostgreSQL Global Development Group. PostgreSQL, Version 9.1.4, 2012. `http://www.postgresql.org/`.
23. A. P. Sistla and O. Wolfson. Temporal conditions and integrity constraints in active database systems. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pp. 269–280, 1995.

A Appendix

The pseudo-code of the procedures init and eval is given in Figure 4. Our pseudo-code is written in a functional-programming style with pattern matching. The symbol $\langle\rangle$ denotes the empty sequence, $+\!\!+$ sequence concatenation, and $h :: L$ the sequence with head h and tail L.

```
proc init(φ)
  for each ψ ∈ sf(φ) with ψ = ψ S_I ψ' do
    L_ψ ← ⟨⟩
  for each ψ ∈ sf(φ) with ψ = ●_I ψ' do
    A_ψ ← ∅
    τ_ψ ← 0

proc eval(φ, i, τ, Γ)
  case φ = p(x_1, ..., x_n)
    return Γ_p

  case φ = ψ ∧ p(t_1, ..., t_n) & kind_rig(φ)
  case φ = ψ ∧ ¬p(t_1, ..., t_n) & kind_rig(φ)
    A ← eval(ψ, i, τ, Γ)
    C ← get_info_rig(φ)
    return σ_C(A)

  case φ = ψ ∧ p(t_1, ..., t_n) & kind_rig'(φ)
    A ← eval(ψ, i, τ, Γ)
    k ← get_info_rig'(φ)
    R ← ∅
    for each ā ∈ A
      R ← R ∪ reval(p, k, ā)
    return R

  case φ = ψ ∧ ¬ψ'
  case φ = ψ ∧ ψ'
    A ← eval(ψ, i, τ, Γ)
    A' ← eval(ψ', i, τ, Γ)
    C, s̄ ← get_info_and(φ)
    if φ = ψ ∧ ψ' then
      return A ⋈_{C,s̄} B
    else
      return A ▷_{C,s̄} B

  case φ = ψ ∨ ψ'
    A ← eval(ψ, i, τ, Γ)
    A' ← eval(ψ', i, τ, Γ)
    return A ∪ A'

  case φ = ∃x̄. ψ
    A ← eval(ψ, i, τ, Γ)
    s̄ ← get_info_exists(φ)
    return π_s̄(A)

  case φ = [ω_t z̄. ψ](y; ḡ)
    A ← eval(ψ, i, τ, Γ)
    H, t' ← get_info_agg(φ)
    return ω^H_{t'}(A)

  case φ = ●_I ψ
    A' ← A_φ
    A_φ ← eval(ψ, i, τ, Γ)
    τ' ← τ_φ
    τ_φ ← τ
    if i > 0 and (τ − τ') ∈ I then
      return A'
    else
      return ∅

  case φ = ¬ψ S_I ψ'
  case φ = ψ S_I ψ'
    A ← eval(ψ, i, τ, Γ)
    A' ← eval(ψ', i, τ, Γ)
    return eval_since(φ, τ, A, A')
```

Fig. 4. The init and eval procedures.

```
proc eval_since(φ, τ, A, A')
  b ← interval_right_margin(φ)
  drop_old(L_φ, b, τ)
  C, s̄ ← get_info_and(φ)
  case φ = ¬ψ S_I ψ' then
    f ← λB.B ▷_{s̄,C} A
  case φ = ψ S_I ψ' then
    f ← λB.B ⋈_{s̄,C} A
  g ← λ(κ, B).(κ, f(B))
  L_φ ← map(g, L_φ)
  L_φ ← L_φ +++ ⟨(τ, A')⟩
  return fold_left(aux_since, ∅, L_φ)

proc drop_old(L, b, τ)
  case L = ⟨⟩
    return ⟨⟩
  case L = (κ, B) :: L'
    if τ − κ ≥ b then
      return drop_old(L', b, τ)
    else return L

proc aux_since(R, (κ, B))
  if (τ − κ) ∈ I then return R ∪ B
  else return R
```

Fig. 5. The eval_since procedure.

We describe the eval procedure in the following in more detail. The cases correspond to the rules defining the set of monitorable formulas. The pseudo-code for the cases corresponding to non-temporal connectives follows closely the equalities given in Section 3.3 and also given in the appendix of the full version of the paper. The predicates kind_rig and kind_rig' check whether the input formula φ is indeed of the intended kind. The get_info_$*$ procedures return the parameters used by the corresponding relational algebra operators. For instance, get_info_rig returns the singleton set consisting of the constraint corresponding to the restrictions $p(\bar{t})$ or $\neg p(\bar{t})$. Similarly, get_info_rig' returns the effective index corresponding to the only variable that appears only in the right conjunct of φ. The procedure

$\mathsf{reval}(p, k, \bar{a})$ returns the set $\{d \in \mathbb{D} \mid (a_1, \ldots, a_{k-1}, d, a_k, \ldots, a_{n-1}) \in p^{\mathcal{D}}\}$, for any $\bar{a} \in \mathbb{D}^{n-1}$, where n is the arity of the rigid predicate symbol p.

The case for the formulas of the form $\bullet_I \psi$ is straightforward. We recursively evaluate the subformula ψ, we update the state, and we return the relation resulting from the evaluation of ψ at the previous time point, provided that the temporal constraint is satisfied. Otherwise we return the empty relation.

The case for the formulas φ of the form $\psi \,\mathsf{S}_I\, \psi'$ or $\neg\psi \,\mathsf{S}_I\, \psi'$ is more involved. It is mainly handled by the sub-procedure $\mathsf{eval\_since}$, given in Figure 5. The notation $\lambda x.f(x)$ denotes a function f. For the clarity of the presentation, we assume that $\varphi = \psi \,\mathsf{S}_I\, \psi'$, the other case being similar. The evaluation of φ reflects the logical equivalence $\psi \,\mathsf{S}_I\, \psi' \equiv \bigvee_{d \in I} \psi \,\mathsf{S}_{[d,d]}\, \psi'$. Note that we abuse notation here, as the right-hand side is not necessarily a formula, because I may be infinite. The function $\mathsf{interval\_right\_margin}(\varphi)$ returns b, where $I = [a, b)$ for some $a \in \mathbb{N}$ and $b \in \mathbb{N} \cup \{\infty\}$.

The state at time point i, that is, after the procedure $\mathsf{eval}(\varphi, i, \tau_i, \Gamma_i)$ has been executed, consists of the list L_φ of tuples (τ_j, R_j^i) ordered with j ascending, where j is such that $j \leq i$ and $\tau_i - \tau_j < b$ and where

$$R_j^i := [\![\psi']\!]^{(\bar{\mathcal{D}}, \bar{\tau}, j)} \bowtie_{\bar{s}, C} \big(\bigcap_{j < k \leq i} [\![\psi]\!]^{(\bar{\mathcal{D}}, \bar{\tau}, k)}\big),$$

with $\bar{s}$ and C defined as in Section 3.3. We have

$$[\![\varphi]\!]^{(\bar{\mathcal{D}}, \bar{\tau}, i)} = \bigcup_{j \leq i, \tau_i - \tau_j \in I} R_j^i.$$

The computation of this union is performed in the last line of the $\mathsf{eval\_since}$ procedure. Note that, in general, not all the relations R_j^i in the list L_φ are needed for the evaluation of φ at time point i. However, the relations R_j^i with j such that $\tau_i - \tau_j \notin I$, that is $\tau_i - \tau_j < a$, are stored for the evaluation of φ at future time points $i' > i$.

We now explain how the state is updated at time point i from the state at time point $i - 1$. We first drop from the list L_φ the tuples that are not longer relevant. More precisely, we drop the tuples that have as first component a timestamp τ_j for which the distance to the current timestamp τ_i is too large with respect to the right margin of I. This is done by the procedure $\mathsf{drop\_old}$. Next, the state is updated according to the logical equivalence $\alpha \,\mathsf{S}\, \beta \equiv (\alpha \wedge \bullet(\alpha \,\mathsf{S}\, \beta)) \vee \beta$. This is done in two steps. First, we update each element of L_φ so that the tuples in the stored relations also satisfy ψ at the current time point i. This step corresponds to the conjunction in the above equivalence and it is performed by the map function. The update is based on the equality $R_j^i = R_j^{i-1} \bowtie_{\bar{s}, C} [\![\psi]\!]^{(\bar{\mathcal{D}}, \bar{\tau}, i)}$. Note that the join distributes over the intersection. The second step, which corresponds to the disjunction in the above equivalence, consists of appending the tuple (τ_i, R_i^i) to L_φ. Note that $R_i^i = [\![\psi']\!]^{(\bar{\mathcal{D}}, \bar{\tau}, i)}$.

Finally, we note that the proof of Theorem 6 follows the above presentation of the algorithm, and is done by induction using the lexicographic ordering on tuples $(i, |\varphi|)$, where $i \in \mathbb{N}$ and $|\varphi|$ denotes φ's size, defined as expected. Furthermore, the proof of Lemma 5 is straightforward. It follows by induction on the formula structure and from the equalities given in Section 3.3, as each relational algebra operator produces a finite relation when applied to finite relations.