A Real-Time, Flexible Logging and Monitoring Infrastructure for MonPoly

Jonas Degelo

Supervisor:
François Hublet
Professor:
Prof. Dr. David Basin

Bachelor's Thesis

Information Security Group
Department of Computer Science
ETH Zürich
February 2023

Contents

Chapter 1

Introduction

1.1 Motivation

Our digital world consists of many hardware and software systems. These systems are continuously performing a lot of actions. For a variety of reasons one might want to monitor those actions and make sure that they do not violate some predefined specification. One way to achieve this is to log relevant actions and analyze these logs. Such monitoring is part of the field of Runtime Verification (RV) [1]. The analysis of the logs can either be done *online* while the system is running or it can be done *offline* after the system has terminated.

Consider a social media site. It is bound by an increasing number of privacy laws and regulations. Let a hypothetical piece of regulation be that a user's location information may not be used to tailor advertisements to that user unless the user gave specific permission. Then the site could log every time instance when a user's location data is accessed and the purpose of the access. In words the predefined specification the site wants to check then could be: "If a user's location data is accessed and the purpose of the access is for tailoring advertisements, the user must have previously given permission for there location data to be used for advertising purposes".

MonPoly [7] is a tool for such runtime monitoring. It can perform both online and offline monitoring. For online monitoring it accepts new events via standard input. For offline monitoring it can read a timestamped log file that was generated during the runtime of a system. It uses Metric First-Order Temporal Logic (MFOTL) [5, 3, 9] as a formal specification language, which we will introduce in the background chapter.

MonPoly in its current state has some limitations that we want to improve. For one, online monitoring can not easily be done on a different machine from which the system is running on. This does not fit well with the way many modern systems operate. Modern systems are often very distributed. It is common that different functions of a system run on different machines. These can be physical or virtual machines and more and more applications are also containerized with technologies like Docker. Oftentimes multiple machines also perform the same kind of operation, e.g. caching servers. MonPoly in its current state does not fit well into this world of interconnected microservices.

Another issue that MonPoly faces currently is data portability. MonPoly can store its execution state to disk before stopping. It can also restore that state, but only on the same system as the way it stores the state is tied to the physical memory configuration of the system. We would like the ability to have MonPoly run on one system, then shut it down and restart it on a different system where it resumes with the same state that the monitor had before shutting down on the first system.

Further there is no built-in way to update the monitored policy. We aim to offer a first method for policy changes with minimal overhead in time and compute power.

We improve MonPoly in these aspects by building a wrapper that connects it with a database and also provides a new, more web friendly, interface to MonPoly. It has always been possible to use a database for logging purposes in addition to MonPoly. We integrate the logging and monitoring functionality into a single tool. This reduces potential redundancies and gives us more flexibility on the monitoring side.

MonPoly works with timestamped and tabular data. We make use of this fact for the choice of database. The temporal nature of the data leads us to time series databases [**empty citation**], which, as the name implies, are optimized for temporal data. By far the most common type of databases are relational databases. This is a great coincidence, because relational databases make extensive use of tables for storing data. We looked at a few different relational time series databases. In the end we opted for QuestDB [16].

1.2 Our Approach

We extend MonPoly with a web based wrapper written in Python using Flask [12]. This wrapper provides a REST API [11] to MonPoly. The wrapper accepts incoming events and manages both the monitoring and logging to the database. We aim for a consistent state between the database and the monitor. Consistent here means that an event is stored in the database if and only if it has been properly processed by the monitor.

The database connection allows us to stop MonPoly on one system and resume the monitoring on a different system, by querying the database.

To facilitate some functions of the wrapper we have added some extensions to MonPoly itself. We added flags to MonPoly to print the schema of a given signature in SQL as well as JSON format. One of our aims was fault tolerance and for this we added some options to keep that would keep the monitor running instead of exiting when encountering certain issues. For the policy change we need to potentially reload a lot of events into the monitor. We added an option to first read events from a file and then switch to standard input. Previously the monitor would either read a file and then stop or continuously read from standard input. Another addition are capabilities to get the relative interval of a MFOTL formula and also to get the relative intervals of predicates in a formula. We have begun work on a different method of doing a policy change by changing only parts of a formula while MonPoly keeps running and can keep the state of the formula parts that are unchanged.

1.3 Contributions

We package MonPoly as a web app and add a new API that offers greater flexibility in how MonPoly can be used. Through this wrapper we connect MonPoly to a database. The addition of a database gives us more options in terms of data portability.

We make use of relative intervals [6] to get a good over approximation of the data needed to continue monitoring a specific formula.

With the database we can offer a first version of a policy change by stopping the monitor and starting a new instance with the events within the relative interval of the new policy already loaded.

The code base of the wrapper is in a GitLab repository [13]. The repository contains over 1300 lines of Python code. Our extended MonPoly is a branch of the development version of MonPoly on BitBucket [8]. We have added over 500 lines of code to MonPoly.

Chapter 2

Background

2.1 Metric First-Order Temporal and Dynamic Logic

As mentioned in the introduction, Metric First-Order Temporal Logic (MFOTL) [5, 3, 9] is used as a policy specification language by MonPoly. Here we give a quick overview of MFOTL. MFOTL is well suited to express a variety of policies one might want to monitor. It combines First Order Logic (FOL) with metric temporal operators. The metric aspect of these operators is the interval I they are bound by. This interval denotes a time frame in which the formula needs to be satisfied.

Metric First-Order *Dynamic* Logic (MFODL) [2] is an even more expressive specification language than MFOTL. MFODL introduces the notion of regular expressions. It is more general than MFOTL and in theory the temporal operators from MFOTL could be replaced by the two more powerful operators introduced by MFODL. In practice, it is often useful to keep the basic temporal operators as we can apply specialized optimizations to them that cannot be done with regular expressions.

Basin et al. [4] extends MFOTL with aggregations. Aggregation operations like SUM are commonly seen in database contexts. When considering an example like a monthly spending limit for a credit card it becomes clear how aggregations can be useful in policy monitoring.

2.1.1 Syntax and Semantics

Let's recall the syntax and semantics of MFOTL [5, 3, 22] as well as MFODL [2]. Like Basin et al. [3] we let $\mathbb{I}$ be the set of nonempty intervals over $\mathbb{N}$. We use the common interval definition with round brackets for open intervals and square brackets for closed ones, e.g. $[a, b) := \{x \in \mathbb{N} \mid a \leq x < b\}$. A signature S, as defined by Basin et al. [3], is a tuple (C, R, ι). C is a finite set of constant symbols, R is a finite set of predicates, and $\iota : R \to C$ is a function that assigns each predicate $r \in R$ an arity $\iota(r)$. C and R are disjoint, i.e. $C \cap R = \emptyset$. Let $S = (C, R, \iota)$ be a signature and V a finite set of variables with $V \cap (C \cup R) = \emptyset$. Now we have enough information to restate the definition of an MFOTL formula (Definition 2.1 in Basin et al. [3]). Let $\circ$ be any ordering relation over $V \cup C$, i.e. $\approx$, $\prec$, $\preceq$. Basin et al. [3] only included the equality relation, the other comparisons were added in [2].

Definition 2.1 *The MFOTL formulas over S are inductively defined in the following way.*

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

(ii) For $r \in R$ and $t_1, t_2, \ldots, t_{\iota(r)} \in V \cup C$, $r(t_1, t_2, \ldots, t_{\iota(r)})$ is a formula.

(iii) For $x \in V$, if ϕ and ψ are formulas then $(\neg\phi), (\phi \vee \psi)$, and $(\exists x.\phi)$ are formulas.

(iv) For $I \in \mathbb{I}$, if ϕ and ψ are formulas then $(\bullet_I \phi), (\bigcirc_I \phi), (\phi \, \mathcal{S}_I \, \psi)$, and $(\phi \mathcal{U}_I \, \psi)$ are formulas.

Definition 2.1 omits common first order logical operators $\wedge$ ("and") and $\forall$ ("for all"). These can be constructed as syntactic sugar from the three FOL operators negation, or, and the exists quantifier. The metric temporal operators in MFOTL are $\mathcal{U}_I$ ("until"), $\mathcal{S}_I$ ("since"), $\bullet_I$

("previous"), and $\bigcirc_I$ ("next"). Similarly to FOL the four MFOTL operators can also be used to construct other convenient operators such as $\blacklozenge_I$ ("once"), $\Diamond_I$ ("eventually"), $\Box_I$ ("always"), and $\blacksquare_I$ ("historically"). See Basin et al. [3] for the concrete derivations of these additional operators.

The subscript I of the temporal operators specifies the time interval in the past or the future in which a formula must be satisfied. For the evaluation of a formula Basin et al. [3] introduces the notion of a structure $\mathcal{D}$ over a signature $S = (C, R, \iota)$. It consists of the domain $|\mathcal{D}| \neq \emptyset$ and interpretations $c^{\mathcal{D}} \in |\mathcal{D}|$ and $r^{\mathcal{D}} \in |\mathcal{D}|$, for each $c \in C$ and $r \in R$. Basin et al. [3] further defines a temporal structure as a pair $(\bar{\mathcal{D}}, \bar{\tau})$ whith $\bar{\mathcal{D}} = (\mathcal{D}_0, \mathcal{D}_1, \dots)$ is a sequence of structures over S and $\bar{\tau} = (\tau_0, \tau_1, \dots)$ is a sequence of non-negative integers (*time stamps*) with the following 3 properties.

1. The sequence is monotonically increasing, i.e. $\forall i.\tau_i \leq \tau_{i+1}$. Moreover, $\bar{\tau}$ makes progress for every $\tau \in \mathbb{N}$ there is some i such that $\tau < \tau_i$.
2. $\bar{\mathcal{D}}$ has constant domains, that is, for all $i \geq 0$, $|\mathcal{D}_i| = |\mathcal{D}_{i+1}|$.
3. Each constant symbol $c \in C$ has a rigid interpretation, that is, for all $i \geq 0$, $c^{\mathcal{D}_i} = c^{\mathcal{D}_{i+1}}$.

A temporal structure is sometimes called a trace and denoted σ. It is important to note the difference between *time stamps* and *time points*. Time points are the combination of a time stamp τ_i and a structure $\mathcal{D}_i$, where i is the index by which a time point is described. Since the domain of the structures $\mathcal{D}_i$ and the valuation of constants $c \in C$ do not change they are denoted as $|\bar{\mathcal{D}}|$ and $c^{\bar{\mathcal{D}}}$ respectively. Basin et al. defines a *valuation* as a mapping $v : V \to |\bar{\mathcal{D}}|$. And for a variable vector $\bar{x} = (x_1, \dots, x_n)$, and $\bar{d} = (d_1, \dots, d_n) \in |\bar{\mathcal{D}}$ it defines the notation $v[\bar{x} \mapsto \bar{d}]$ for the valuation that maps x_i to d_i for $1 \leq i \leq n$. Further it uses a notational hack to also apply a valuation v to constant symbols $c \in C$ where $v(c) = c^{\bar{\mathcal{D}}}$. For convenience, we restate Definition 2.2 from Basin et al. [3]. Since we changed the definition of a formula to include ordering relations $(\prec, \preceq)$ instead of only equality $(\approx)$, we append this definition to allow for that. These additions are also in Figure 1 of Basin et. al [2].

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

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

Analogous to Definition 2.2 we now give the definition of an MFODL formula. The definition is based on Figure 4 of Basin et al. [2], but in a notation closer to the one we already used for MFOTL formulas. For MFODL formulas we first have to define regular expressions, as MFODL formulas depend on them.

Definition 2.3 *The MFODL regular expressions are defined in the following way:*

(i) *If $k \in \mathbb{N}$ then $\star^k$ is a regular expression.*

(ii) *If ϕ is a formula then $(\phi?)$ is a regular expression.*

(iii) *If ρ is a regular expression then (ρ^*) is a regular expression.*

(iv) *If ρ and σ are regular expressions then $(\rho + \sigma)$ and $(\rho \cdot \sigma)$ are regular expressions.*

With this we extend Definition 2.1 to include MFODL formulas.

Definition 2.4 *The MFODL formulas over S are inductively defined in the following way.*

(i) *MFOTL formulas are also MFOTL formulas*

(ii) *For $I \in \mathbb{I}$, if ρ is a regular expression then $(\blacktriangleleft_I \rho)$ and $(\triangleright_I \rho)$ are formulas.*

For the evaluation of an MFODL formula we first need to define how regular expressions get evaluated. The definition is originally given through the `match` function in Figure 4 of Basin et al. [2]. We formulate the definition that is more in line with the mathematical notation we used this far.

Definition 2.5 *Let $(\bar{\mathcal{D}}, \bar{\tau})$ be a temporal structure over the signature $S = (C, R, \iota)$, with $\bar{\mathcal{D}} = (\mathcal{D}_0, \mathcal{D}_1, \dots)$ and $\bar{\tau} = (\tau_0, \tau_1, \dots)$, ϕ a formula over S, v a valuation, and $i, j \in \mathbb{N}$.*

$$
\begin{aligned}
&(\bar{\mathcal{D}}, \bar{\tau}, v, i, j) \models_r \star^k && \text{iff } i = j + k \\
&(\bar{\mathcal{D}}, \bar{\tau}, v, i, j) \models_r \phi? && \text{iff } (\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \phi \text{ and } i = j \\
&(\bar{\mathcal{D}}, \bar{\tau}, v, i, j) \models_r \rho + \sigma && \text{iff } (\bar{\mathcal{D}}, \bar{\tau}, v, i, j) \models_r \rho \text{ or } (\bar{\mathcal{D}}, \bar{\tau}, v, i, j) \models_r \sigma \\
&(\bar{\mathcal{D}}, \bar{\tau}, v, i, j) \models_r \rho \cdot \sigma && \text{iff for some } k \in \mathbb{N}, (\bar{\mathcal{D}}, \bar{\tau}, v, i, k) \models_r \rho \text{ and } (\bar{\mathcal{D}}, \bar{\tau}, v, k, j) \models_r \sigma \\
&(\bar{\mathcal{D}}, \bar{\tau}, v, i, j) \models_r \rho^* && \text{iff for some } l \in \mathbb{N}, o_1, \dots, o_l \in \mathbb{N}, (\bar{\mathcal{D}}, \bar{\tau}, v, i, o_1) \models_r \rho \text{ and } \dots \\
& && \quad \text{and } (\bar{\mathcal{D}}, \bar{\tau}, v, o_l, j) \models_r \rho
\end{aligned}
$$

With the help of Definition 2.5 we can now define the evaluation of MFODL formulas which is also given in Figure 4 of Basin et al. [2] through the `sat` function. The trace σ in the `sat` function is simply our temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$. The data list v is the valuation mapping. And i specifies the current time point in both notations.

Definition 2.6 *Let $(\bar{\mathcal{D}}, \bar{\tau})$ be a temporal structure over the signature $S = (C, R, \iota)$, with $\bar{\mathcal{D}} = (\mathcal{D}_0, \mathcal{D}_1, \dots)$ and $\bar{\tau} = (\tau_0, \tau_1, \dots)$, ϕ a formula over S, v a valuation, and $i \in \mathbb{N}$. We define the relation $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \phi$ inductively as follows.*

$$
\begin{aligned}
&(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \blacktriangleleft_I \rho \text{ iff for some } j \leq i, \tau_i - \tau_j \in I \text{ and } (\bar{\mathcal{D}}, \bar{\tau}, v, j, i) \models_r \rho \\
&(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \triangleright_I \rho \text{ iff for some } j \geq i, \tau_j - \tau_i \in I \text{ and } (\bar{\mathcal{D}}, \bar{\tau}, v, i, j) \models_r \rho
\end{aligned}
$$

2.2 MonPoly

MonPoly [7] is a policy monitoring tool written in OCaml that supports MFOTL with aggregations and in its newest iterations it also has support for MFODL. MonPoly can monitor a fragment of MFOTL/MFODL where all future operators must be bounded. One major exception to that rule is an (implicit) always operator around the desired policy.

Let's return to the social media example from the introduction and look at how we would go about monitoring that policy with MonPoly. We recall our description in words: "If a user's location data is accessed and the purpose of the access is for tailoring advertisements, the user must have previously given permission for their location data to be used for advertising purposes." In MonPoly a policy is tied to a signature. A signature can be compared with a database schema and describes the arity and types of possible events. So let's consider a possible signature for our example:

```
loc_accessed(user_id: int, purpose: string)
perm_granted(user_id: int)
perm_revoked(user_id: int)
```

Figure 2.1: Example MonPoly Signature

This is a basic signature with 3 predicates. The first one means that a users location data has been used for a specified purpose. The last two events get triggered when a user either grants or revokes permission for their location data to be used for advertising purposes. Let's now define the policy in a formal manner.

$$\square(\texttt{loc\_accessed(i, "advertising")} \implies (\blacklozenge_{[0,\infty)}\texttt{perm\_granted(i)} \\ \wedge\neg(\texttt{perm\_revoked(i)}\ \mathcal{S}_{[0,\infty)}\ \texttt{perm\_granted(i)})))$$

For MonPoly we first get rid of the surrounding $\square$, because MonPoly implicitly adds an always-operator around any policy. The remaining formula in MonPoly syntax is the following:

```
loc_accessed(i, "advertising")
IMPLIES
(
    (ONCE[0,*) perm_granted(i))
    AND
    (NOT (perm_revoked(i) SINCE[0,*) perm_granted(i)))
)
```

Figure 2.2: Example MonPoly Policy

While MonPoly cannot actually monitor this formula directly, it can monitor the negation of this formula. For this on can use the `-negate` flag when running MonPoly.

2.3 Time Series Databases

Time series databases are a class of databases optimized for timestamped data. For example, they optimize for data retrieval within a certain time range. With the advance of internet of things devices with built-in sensors time series databases are experiencing explosive growth. And as we have established they happen to fit well with our monitoring goals. There are many different options of time series databases available. We were looking for something with good performance, good support for tables of data, and good usability. We have opted for QuestDB [16].

2.3.1 QuestDB

QuestDB uses a column-based storage model [20]. It supports the PostgreSQL wire protocol [18] for querying and inserting data. This is the same protocol used by the popular relational database PostgreSQL [15]. The support for SQL as a query language is of great use, because SQL is the most widely used and taught database query language. PostgreSQL is in itself a very popular dialect of SQL and thus has a lot of tooling available which can also be used with QuestDB. It further provides a REST API [19] and has a web console for both inserting and querying data. For best performance it supports the InfluxDB Line Protocol [17, 14] with client libraries for most popular modern programming languages. InfluxDB is itself a time series database and the line protocol is a write-protocol developed by and for InfluxDB optimized for time series data. The QuestDB client libraries for different programming languages are implementations of the InfluxDB line protocol by QuestDB. QuestDB recommends using the Line Protocol for the best write performance, compared to the other methods of writing data. The Line Protocol cannot be used to query data. For that the PostgreSQL Wire Protocol is the recommended method. QuestDB is written in Java, open source, and licensed under the Apache 2.0 license.

2.3.2 Time Series Databases for Runtime Monitoring

In this section we explore how a time series database can be utilized in the context of runtime monitoring. We will contrast policies and database queries, as well as signatures and database schemas. An important and possibly counterintuitive thing is that MFOTL is not any more expressive than FOL. In fact, they have the same expression strength, i.e. any MFOTL formula can be expressed as a FOL formula and vice versa. Any FOL formula is by definition also a MFOTL formula. The other way around any trace in MFOTL can be simulated in FOL by extending predicates with two attributes for the time point and time stamp. Take for example the formula $\phi = \bullet_{[a,b)} P()$ where P is a predicate with arity 0. ϕ is equivalent to the FOL formula $\phi' = P(t_s, t_p) \wedge \exists t'_s . P(t'_s, t_p - 1) \wedge a \leq t_s - t'_s < b$.

It is relatively straight forward to create a database schema from a signature like the one in Figure 2.2. We create one table per predicate. The predicate name is used as the table name. The columns and their types are given by the attributes. MonPoly allows for optional attribute names, we disregard those, if they are present and rely on the ordering of attributes. We need two additional columns, one for the time point, i.e. an integer index, and one for the time stamp when the event occurred. One additional table keeping track of all time points and their time stamps is necessary, because a time point could occur without any event attached to it, if we only had the events stored, we would lose any record of such time points. And any time point can influence the evaluation of a formula. This time stamp and time point table technically makes it redundant to store the time stamps with every predicate, since the mapping from time point indices to time stamps is injective. But QuestDB requires a time stamp column, and it is also beneficial for performance as QuestDB is optimized for queries on time ranges.

The database schema as SQL create statements for our example policy from Figure 2.2 can be seen in Figure 2.3.2. The `timestamp()` function is a special extension to SQL for QuestDB. It specifies the column that is used as the dedicated time stamp in a table. If it is not specified an additional time stamp column would get added. By default, no partitioning would be used.

```
CREATE TABLE perm_revoked(x1 INT,
                          time_stamp TIMESTAMP,
                          time_point INT)
                          timestamp(time_stamp);
CREATE TABLE perm_granted(x1 INT,
                          time_stamp TIMESTAMP,
                          time_point INT)
                          timestamp(time_stamp);
CREATE TABLE loc_accessed(x1 INT, x2 STRING,
                          time_stamp TIMESTAMP,
                          time_point INT)
                          timestamp(time_stamp);
CREATE TABLE ts( time_stamp TIMESTAMP,
                 time_point INT)
                 timestamp(time_stamp);
```

Figure 2.3: SQL Schema for Example Policy

Chapter 3

Architecture

In this section we introduce the general architecture of our wrapper for MonPoly. A more in depth look at the specific technical implementation will be provided in the implementation section. In general, we have three components for this project. On one hand we have the MonPoly with a few extensions. On the other hand is QuestDB. Our wrapper acts as the glue between the two. In addition the wrapper also provides a new interface to MonPoly in the form of a REST API. Figure 3.1 shows the structure of the MonPoly wrapper.

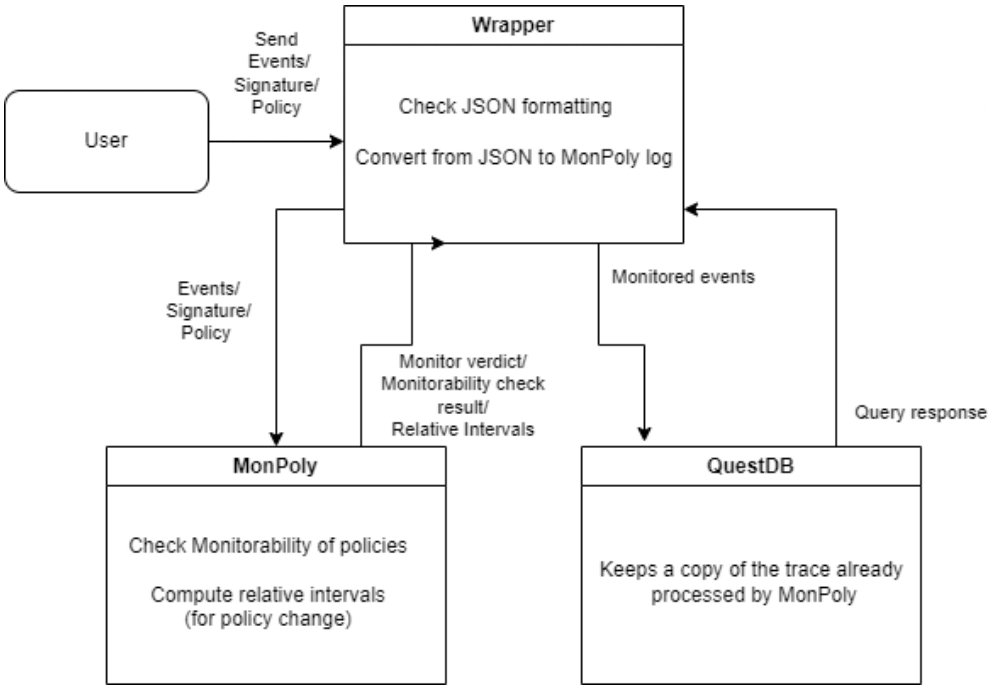

Figure 3.1: Illustration of the project structure

3.1 Wrapper

The wrapper can be run directly on a system with MonPoly installed. The alternative and more portable way to run it is with a Docker container. For Docker, the MonPoly Docker image is used as a base image and then the wrapper code and the required Python libraries are installed on top. In the end the wrapper is simply run like any Flask app.

A QuestDB instance must be running separately. This can be on the same system locally or somewhere on a remote server. The connection details for QuestDB, like ports, username, database password, and hostname must be configured in the wrapper for it to be fully operational.

On a high level the user first provides the connection information for the database. Next a policy and a signature must be sent to the wrapper. If the signature and policy are monitorable, the user can issue a command to start the monitor and from this point onward the wrapper will accept and process incoming events.

At any point the user can issue a policy change by sending the appropriate command and the new policy. The wrapper will then perform the policy change and report once it is complete.

3.2 The REST API

In this section we provide a list of all the REST API endpoints, describe their purpose, and give some basic usage examples.

- `/` This endpoint is designed to be viewed in the browser, and it displays basic information about the state of the wrapper. It shows the current policy and signature, if they have been set. Perhaps most importantly it displays the output from MonPoly.
- `/set-policy` This takes any policy or formula file for MonPoly. At this point no checks on the policy file are performed. A monitorability check is only done once both, a policy and a signature, have been set, and the user attempts to start the monitor with `/start-monitor`. For convenience a separate endpoint that does not start the monitor, but performs the monitorability check could be provided. Through the optional `negate` form key, the user can specify whether the negated or non-negated policy should be monitored. For this any value provided along with the `negate` key gets ignored. Se the following snippet for how `/set-policy` could be used with the command line tool curl [10].

  ```
  $ curl -X POST -F policy=@<path/to/policy> [-F negate=]
                  <wrapper-hostname>[:port]/set-policy
  > {"message":"policy set to  <contents of policy file>"}
  ```

- `/set-signature` This works very similarly to `/set-policy`, except it has no optional parameters like `negate` and as its name describes, it sets the signature and not the policy.

  ```
  $ curl -X POST -F signature=@<path/to/signature> \
                  <wrapper-hostname>[:port]/set-signature

  {"message":"signature set to <contents of signature file>"}
  ```

- `/start-monitor` With this the user can start the monitoring process. If either a policy or signature file has not been set yet, this gets reported to the user and the command exits. First a monitorability check with the provided policy and signature files is done. Any potential issues get reported back. When everything is fine, a MonPoly subprocess gets launched and the wrapper is then ready to accept and process incoming events. It is worth noting that until the monitor gets started the first time, the policy as well as the signature can be changed freely. Passing the `existing-db` argument along with this command restores the state of the monitor according to potential database entries. This can be used, when moving the wrapper from one machine to another. With this command the monitor can also be restarted on the same machine if it was stopped properly, e.g. using the `/stop-monitor` command, and the monitor state was saved to disk. Below we illustrate the basic usage for the command along with a sample response for when the command succeeds.

  ```
  $ curl -X GET [-F existing-db=] <wrapper-hostname>[:port]/start-monitor
  {
      "pid": <process id of MonPoly>,
      "args": [ <command line arguments passed to MonPoly> ]
  }
  ```

- `/stop-monitor` This command stops the MonPoly subprocess in a controlled manner. It waits for any processing to be done and then attempts to stop MonPoly with its `save_and_exit` command, which stores the memory state of MonPoly to disk.
- `/change-policy` The policy change endpoint can only be used when MonPoly is already running. Its usage is analogous to `set-policy`, as can be seen in the following.

  ```
  $ curl -X POST -F policy=@<path/to/policy> [-F negate=]
                  <wrapper-hostname>[:port]/change-policy
  > {"success":"changed policy from  <previous policy> to <new policy>"}
  ```

  The inner workings are of course different from those of `set-policy`. We describe these further in the chapters on algorithms and the implementation. From a user perspective it is important to know that this command only has an effect on the monitor if the newly provided policy is monitorable against the existing signature. If not, the wrapper simply reports any potential issues back to the user and continues monitoring the old policy. An important note is that there is no way to change the signature once MonPoly has been started.
- `/get-policy` and `/get-signature`. These command retrieve the monitored policy or signature respectively. If no policy or signature has been set yet, the returned value instead of the policy or signature is a string informing the user of that. These commands can be used like so:

  ```
  $ curl -X GET localhost:5000/get-policy
  > {"policy":"<current policy"}

  $ curl -X GET localhost:5000/get-signature
  > {"signature":"<current signature"}
  ```

- `/reset-everything` This is a destructive command. In a real world context this should be used exceedingly rarely. It stops any running sub-processes, deletes or resets any configuration files, and clears the database.

  ```
  $ curl -X GET <wrapper-hostname>[:port]/reset-everything
  > {
      "config":"deleted <path/to/config/file>",
      "query":<SQL query used to drop tables>,
      "stopped":"stopped monpoly"
    }
  ```

- `/log-events` This is perhaps the most important endpoint. It is certainly the one that will be used the most as it is the one handling incoming events. This command takes as an argument a JSON file with a list of one or more, ordered, time points. These time points can optionally have a time stamp associated with them. For time points with no specified time stamp the wrapper assigns all them the same time stamp corresponding to the current system time. The list does not technically have to be sorted, but any out of order time points will be skipped. Time points might also be skipped if they contain predicates with unknown names, a wrong number of attributes, or the wrong types of attributes. Time points that are skipped are reported back to the user along with the reason as to why they were skipped. We aimed for a high fault tolerance in this regard. If some time points in a log file get skipped, they have no effect on the processing of time points that were in order and correctly formatted.

  ```
  $ curl -X POST -F events=@<path/to/JSON/log/file> \
                      <wrapper-hostname>[:port]/log-events
  > {"skipped-timepoints":<dictionary of any skipped time points>}
  ```

```
@10
loc_accessed(2, "advertising") (3, "navigation")

@20
perm_granted(2)
loc_accessed(4, "fitness-tracking")
perm_revoked(3)

@35

@40
perm_revoked(2)
```

Figure 3.2: Example Log in MonPoly format

Here we quickly describe our JSON format and how it corresponds to a MonPoly log with the help of a short example in Figure 3.2. Consider the following potential log for our ongoing example with the signature from Figure 2.2. The same log in our JSON format can be seen in Figure 3.2. Note that in JSON we have opted for the more human-readable version of a time stamp in Unix format, whereas MonPoly internally always works with integers. Though MonPoly does provide a flag with which time stamps are displayed to the user in Unix format, for example when a policy violation occurs at a time point.

- `/get-events` This endpoint is a direct interface to the database and queries all time points in a specified time range. The time range can be specified with optional `start` and `end` parameters. These parameters should be formatted just like the time stamps in our JSON log files. The time points get returned in the same JSON format as described above.

  ```
  $ curl -X GET [-F start=<start-date>] [-F end=<end-date>] \
                 <wrapper-hostname>[:port]/get-events
  > [<list of all time points with their events>]
  ```

- `/get-most-recent` This command simply returns the latest time point that is stored in the database. If the database is still empty, the returned value is simply `null`.

  ```
  $ curl -X GET <wrapper-hostname>[:port]/get-most-recent
  > {"response": <time stamp of newest time point, or null>}
  ```

The wrapper also provides various "getter"- and "setter"- methods for the configuration details of QuestDB. They do not change the configuration of QuestDB itself, but simply tell the wrapper how to reach QuestDB. These should be fairly self-explanatory. We will quickly list them here for completeness. For more information on what each of these values does, the QuestDB documentation [16, 18, 17] can be consulted. The QuestDB related endpoints are:
`/db-set-user`,
`/db-set-password`,
`/db-set-host`,
`/db-set-pgsql-port`,
`/db-set-influxdb-port`,
`/db-set-database`,
`/db-get-user`,
`/db-get-password`,
`/db-get-host`,
`/db-get-pgsql-port`,
`/db-get-influxdb-port`, and
`/db-get-database`.

```
[
    {
        "timestamp": "1970-01-01 00:00:10",
        "predicates": [
            {
                "name": "loc_accessed",
                "occurrences": [
                    [2, "advertising"], [3, "navigation"]
                ]
            }
        ]
    },
    {
        "timestamp": "1970-01-01 00:00:20",
        "predicates": [
            {
                "name": "loc_accessed",
                "occurrences": [
                    [4, "fitness-tracking"]
                ]
            },
            {
                "name": "perm_granted",
                "occurrences": [
                    [2]
                ]
            },
            {
                "name": "perm_revoked",
                "occurrences": [
                    [3]
                ]
            }
        ]
    },
    {
        "timestamp": "1970-01-01 00:00:35",
        "predicates": []
    },
    {
        "timestamp": "1970-01-01 00:00:35",
        "predicates": [
            {
                "name": "perm_revoked",
                "occurrences": [
                    [2]
                ]
            }
        ]
    }
]
```

Figure 3.3: Example Log in JSON format

Chapter 4

Algorithms

4.1 Policy Change

This section gives a high level view of our policy change method. The individual parts of the policy change will be explained in the following sections of this chapter. We have a running instance of MonPoly monitoring some policy. The user asks the wrapper to monitor a new policy. The wrapper checks the monitorability of the new policy against the existing policy. If it is not monitorable the wrapper keeps the current instance of MonPoly running and reports the issue with the new policy to the user. Otherwise the wrapper uses MonPoly to get the extended relative intervals of the new policy. Then these extended relative intervals get converted to SQL queries and the wrapper runs these queries on QuestDB. The response from QuestDB gets converted into a MonPoly log file. Next the wrapper stops the current iteration of MonPoly and starts a new one that first reads the created log file. At this point the policy change is done, and the wrapper can continue with its normal operation.

4.2 Relative Intervals

First we append the definition of relative intervals from Basin et al. [6] to include all operators currently supported by MonPoly. Namely we add definitions for the MFODL operators. Intervals are defined over $\mathbb{Z}$ and can either be open or closed. The operators $\oplus$ and $\Cup$ are defined the same way as in Basin et al. [6]. Let I and J be some arbitrary intervals then $I \oplus J := \{i+j \mid i \in I \text{ and } j \in J\}$ and $I \Cup J$ is the smallest interval containing all values in both I and J.

Definition 4.1 *The relative interval of the formula ϕ, $\mathrm{RI}(\phi) \subseteq \mathbb{Z}$ is defined recursively over the formula structure:*

$$\mathrm{RI}(\phi) = \begin{cases} \{0\} & \text{if } \phi \text{ is an atomic formula,} \\ \mathrm{RI}(\psi) & \text{if } \phi \text{ is of the form } \neg\psi,\ \exists x.\psi, \\ & \text{or } \forall x.\psi, \\ \mathrm{RI}(\psi) \Cup \mathrm{RI}(\chi) & \text{if } \phi \text{ is of the form } \psi \vee \chi, or \psi \wedge \chi, \\ (-b, 0] \Cup ((-b, -a] \oplus \mathrm{RI}(\psi)) & \text{if } \phi \text{ is of the form } \bullet_{[a,b)}\psi, \\ [0, b) \Cup ([a, b) \oplus \mathrm{RI}(\psi)) & \text{if } \phi \text{ is of the form } \bigcirc_{[a,b)}, \\ (-b, 0] \Cup ((-b, 0] \oplus \mathrm{RI}(\psi)) \Cup ((-b, -a] \oplus \mathrm{RI}(\chi)) & \text{if } \phi \text{ is of the form } \psi\, \mathcal{S}_{[a,b)} \chi, \\ [0, b) \Cup ([0, b) \oplus \mathrm{RI}(\psi)) \Cup ([a, b) \oplus \mathrm{RI}(\chi)) & \text{if } \phi \text{ is of the form } \psi\, \mathcal{U}_{[a,b)}\, \chi, \\ [0, b) \Cup ([0, b) \oplus \mathrm{RI}_{reg}(\rho)) & \text{if } \phi \text{ is of the form } \triangleright_{[a,b)}\, \rho, \text{ and} \\ (-b, 0] \Cup ((-b, 0] \oplus \mathrm{RI}_{reg}(\rho)) & \text{if } \phi \text{ is of the form } \blacktriangleleft_{[a,b)}\, \rho. \end{cases}$$

We recursively define the relative interval of regular expressions as seen in Basin et al. [2] in the following way.

Definition 4.2 *The relative interval of the regular expression ρ, $\mathrm{RI}_{reg}(\rho) \subseteq \mathbb{Z}$ is defined recursively over the structure of the regular expression:*

$$\mathrm{RI}_{reg}(\rho) = \begin{cases} \{0\} & \text{if } \rho \text{ is of the form } \star^k, \\ \mathrm{RI}(\phi) & \text{if } \rho \text{ is of the form } \phi?, \\ \mathrm{RI}_{reg}(\sigma) \Cup \mathrm{RI}_{reg}(\tau) & \text{if } \rho \text{ is of the form } \sigma + \tau \text{ or } \sigma \cdot \tau, \text{ and} \\ \mathrm{RI}_{reg}(\sigma) & \text{if } \rho \text{ is of the form } \sigma^*. \end{cases}$$

4.2.1 Correctness

We want to show that it is sufficient to extract all time points with a time stamp in the relative interval $\mathrm{RI}(\phi)$ for a formula ϕ from a trace to evaluate ϕ. Lemmas A.2 and A.4 in Basin et al. [6] provides exactly this. We will outline how at the end of this section.

For convenience, we restate Lemmas A.2 and A.4 as well as the definitions they require, and some auxiliary lemmas here. First we recall Definition A.2 from Basin et al. [6].

Basin et al. [6] defines a *slice* of a temporal structure in Definition 3.1. We restate this definition here. (In the definition R is part of a signature $S = (C, R, \iota)$.)

Definition 4.3 *Let $s : [0, l) \to \mathbb{N}$ be a strictly increasing function, with $l \in \mathbb{N} \cup \{\infty\}$. The temporal structure $(\bar{\mathcal{D}}', \bar{\tau}')$ is a slice of $(\bar{\mathcal{D}}, \bar{\tau})$ (with respect to the function s) if $\tau_{i'} = \tau_{s(i)}$ and $r^{\mathcal{D}'_i} \subseteq r^{\mathcal{D}_{s(i)}}$, for all $i \in [0, l)$ and all $r \in R$.*

Definition 4.4 *Let $T \subseteq \mathbb{Z}$ be an interval and $(\bar{\mathcal{D}}, \bar{\tau})$ a trace. The T-slice of $(\bar{\mathcal{D}}, \bar{\tau})$ is the time slice $(\bar{\mathcal{D}}', \bar{\tau}')$ of $(\bar{\mathcal{D}}, \bar{\tau})$, where $s : [0, l) \to \mathbb{N}$ is the function $s(i') = i' + c, l = |\{i \in \mathbb{N} \mid \tau_i \in T\}|$ and $c = \min\{i \in \mathbb{N} \mid \tau_i \in T\}$. We also require that $\tau'_l \notin T$ and $\mathcal{D}'_{i'} = \mathcal{D}_{s(i')}$, for all $i' \in [0, l)$.*

We also need Definition A.4 from Basin et al. [6].

Definition 4.5 *Let $I \subseteq \mathbb{Z}$ be an interval and $c, i \in \mathbb{N}$. The temporal structures $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are (I, c, i)-overlapping if the following conditions hold:*

1. *$j \geq c, \mathcal{D}_j = \mathcal{D}'_{j-c}$, and $\tau_j = \tau'_{j-c}$, for all $j \in \mathbb{N}$ with $\tau_j - \tau_i \in I$.*
2. *$\mathcal{D}_{j'+c} = \mathcal{D}'_j$, and $\tau_{j'+c} = \tau'_{j'}$, for all $j' \in \mathbb{N}$ with $\tau'_{j'} - \tau_i \in I$.*

Lemma A.1 in Basin et al. [6] is useful, we restate it here.

Lemma 4.1 *For every formula ϕ, $0 \in \mathrm{RI}(\phi)$.*

Proof The proof that this holds is a structural induction over the formula structure for both MFOTL and MFODL, except that the MFODL part requires a similar lemma for regular expressions which is mutually recursive with this one.

Lemma 4.2 *For every regular expression ρ, $0 \in \mathrm{RI}_{reg}(\rho)$.*

Proof For a complete proof, this proof would need to go over both, Lemma 4.1 and Lemma 4.2. The first case is a regular expression $\rho = \star^k$ for $k \in \mathbb{N}$. Trivially $0 \in \mathrm{RI}_{\mathrm{reg}}(\rho) = \mathrm{RI}_{\mathrm{reg}}(\star^k) = 0$. Next $\rho = \phi?$ for a formula ϕ. $\mathrm{RI}_{\mathrm{reg}}(\rho) = \mathrm{RI}_{\mathrm{reg}}(\phi?) = \mathrm{RI}(\phi)$. By Lemma 4.1 $0 \in \mathrm{RI}(\phi)$ and thus $0 \in \mathrm{RI}_{\mathrm{reg}}(\rho)$. Now we consider $\rho = \sigma \Cup \tau$ for two regular expressions σ and τ. By the induction hypothesis (IH) $0 \in \mathrm{RI}_{\mathrm{reg}}(\sigma)$ and $0 \in \mathrm{RI}_{\mathrm{reg}}(\tau)$. Thus $0 \in \mathrm{RI}_{\mathrm{reg}}(\sigma) \Cup \mathrm{RI}_{\mathrm{reg}}(\tau) = \mathrm{RI}_{\mathrm{reg}}(\rho)$. Finally $\rho = \sigma*$ for a regular expression σ. By IH $0 \in \mathrm{RI}_{\mathrm{reg}}(\sigma) = \mathrm{RI}_{\mathrm{reg}}(\rho)$. Therefore Lemma 4.2 holds for all regular expressions under the condition that Lemma 4.1 also holds.

Lemma A.2 in Basin et al. [6] goes as follows:

Lemma 4.3 *Let $T \subseteq \mathbb{N}$ and $I \subseteq \mathbb{Z}$ be intervals, $(\bar{\mathcal{D}}, \bar{\tau})$ a temporal structure, and $(\bar{\mathcal{D}}', \bar{\tau}')$ a $(T \oplus I)$-slice of $(\bar{\mathcal{D}}, \bar{\tau})$. The temporal structures $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are (I, c, i)- overlapping for all $i \in \mathbb{N}$ with $\tau_i \in T$, where c is the value in Definition 4.4 used by the function s with respect to $(\bar{\mathcal{D}}, \bar{\tau})$ and its time slice $(\bar{\mathcal{D}}', \bar{\tau}')$.*

And finally Lemma A.4 itself.

Lemma 4.4 *Let ϕ be a formula and $(\bar{\mathcal{D}}, \bar{\tau}), (\bar{\mathcal{D}}', \bar{\tau}')$ temporal structures. If $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are $(\mathrm{RI}(\phi), c, i)$-overlapping, for some c and i, then for all valuations v, it holds that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \phi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i) \models \phi$.*

A proof by structural induction on the structure of MFOTL formulas for Lemma A.4 [6] / Lemma 4.4 is provided in Basin et al. [6]. We would like to extend the lemma to MFODL formulas as well. For this an analogous lemma that works for regular expressions is needed.

Lemma 4.5 *Let ρ be a regular expression and $(\bar{\mathcal{D}}, \bar{\tau}), (\bar{\mathcal{D}}', \bar{\tau}')$ temporal structures. If $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are $(\mathrm{RI}_{reg}(\rho), c, i)$-overlapping, for some c and i, then for all valuations v and $j \in \mathbb{N}$, it holds that $(\bar{\mathcal{D}}, \bar{\tau}, v, i, j) \models_r \rho$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i-c, j-c) \models_r \rho$.*

Proof Lemmas 4.4 and 4.5 are mutually recursive and must be proofed in combination by structural induction on both the structure of MFODL formulas (Definition 2.3) and the structure of regular expressions (Definition 2.4). For time reasons we forgo this proof and focus on our extension of the idea behind relative intervals in the next section.

We close this section by outlining how Lemmas 4.4, 4.5, and 4.3. For our purposes of a policy change we are essentially taking a $\{\tau_k\} \oplus \mathrm{RI}(\phi)$- slice at some "current" time point with time stamp τ_k, $(\bar{\mathcal{D}}', \bar{\tau}')$, of some temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$. From Lemma 4.3 it follows that the slice and $(\bar{\mathcal{D}}, \bar{\tau})$ are $(\mathrm{RI}(\phi), c, k)$ -overlapping for the time point k. And thus by the Lemmas 4.4 and 4.5 it follows that for all valuations v, $(\bar{\mathcal{D}}, \bar{\tau}, v, k) \models \phi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, k-c) \models \phi$. Hence the two traces assign the formula ϕ the same truth value at the "current" time τ_k.

4.3 Relative Interval Extension

This idea of relative intervals can already filter an existing trace down to a much smaller one by removing events that are unnecessary for the evaluation of a given policy. We expand on this by creating and using a data structure that allows us to select an even smaller sub trace with the same effect of not changing the truth value of the policy.

First we move from one relative interval for an entire policy to one relative interval per predicate occurring in a policy. We break this down further. Every predicate comes with a number of attributes as defined in the signature. Some attributes are potentially constant.

Looking back at our policy from Figure 2.2, `"advertising"` is one such constant attribute in the predicate `loc_accessed`. This means any occurrence of the predicate `loc_accessed` where the second attribute is not `"advertising"`, has no influence on our policy and is therefore not needed in a potential sub trace. We check every predicate in our policy for constant attributes. Then we take the set of different arrangements of constant and variable attributes per predicate. We define a structure that captures constant and variable attributes of a predicate.

Definition 4.6 *Let $S = (C, R, \iota)$ be a signature and $r \in R$ a predicate with arity $\iota(r)$. A mask for the predicate r is a tuple $m = (m_1, \ldots, m_{\iota(r)})$ with $m_1, \ldots, m_{\iota(r)} \in C \cup \{v\}$ and $v \notin V \cup C$.*

v is a placeholder value denoting attributes in the mask that have a non-constant value. Each mask has its own relative interval. For our example the masks with their corresponding relative intervals can be seen in Figure 4.3.

```
loc_accessed(*,"advertising") -> [0,0]
perm_granted(*) -> (*,0]
perm_revoked(*) -> (*,0]
```

Figure 4.1: Extended Relative Intervals of Example Policy

A `*` (asterisk) in the attributes denotes a variable value, v in Definition 4.6. In larger formulas there can be multiple different masks per predicate. Let $\mathcal{M}(r)$ be the set of possible masks for a predicate r.

Definition 4.7 *Let $m = (m_1, \ldots, m_{\iota(r)})$ and $n = (n_1, \ldots, n_{\iota(r)})$ be two masks for a predicate $r \in R$, where R is part of the signature $S = (C, R, \iota)$. $m \sqsubseteq n$ denotes that m is no more precise than n, i.e. for all $i \in \mathbb{N}$ with $0 \leq i \leq \iota(r)$, $m_i = n_i$ or $m_i = v$.*

We use a map data structure to store the predicates, masks, and their respective relative intervals.

Definition 4.8 *Let $S = (C, R, \iota)$ be a signature and let $\mathcal{M} : R \to \{(C \cup \{v\})^*\}$ be the function $\mathcal{M}(r) = M$ that gives the set of all possible masks M for a predicate r. A **masked predicate map** is a set $\{(k, i)\}$ where $k = (r, m)$ with $r \in R$ and $m \in \mathcal{M}(r)$ and $i \subseteq \mathbb{Z}$ is an interval over $\mathbb{Z}$.*

On this data structure we define the operators $\ddot{\Cup}$, $\dot{\Cup}$ and $\dot{\oplus}$.

Definition 4.9 *Let M and N be two masked predicate maps and T a positive interval, then*

$$\begin{aligned}
M \ddot{\Cup} N =& \{p(l) \to (I \Cup J) \mid p(l) \to I \in m \text{ and } p(l) \to J \in n\} \\
&\cup \{p(l) \to I \mid (p(l) \to I \in m \text{ and } p(l) \in \text{keys}(M) \setminus \text{keys}(N))\} \\
&\cup \{p(l) \to I \mid (p(l) \to I \in n \text{ and } p(l) \in \text{keys}(N) \setminus \text{keys}(M))\} \\
T \dot{\Cup} M =& \{p(l) \to (T \Cup I) \mid p(l) \to I \in M\} \\
T \dot{\oplus} M =& \{p(l) \to (T \oplus I) \mid p(l) \to I \in M\}
\end{aligned}$$

The notation $p(l) \to I$ denotes an element in a masked predicate map. It is equivalent to the notation $((p, l), I)$, but it better shows how we are working with a map. The keys operator gives all the first elements, the predicate name and mask tuples, in a masked predicate map. With the help of the operators $\ddot{\Cup}$, $\dot{\Cup}$ and $\dot{\oplus}$ we now give a recursive definition for our extension of relative intervals.

We define a relation on masked predicate maps that is analogous to the subset-equals relation for intervals and illustrates that one map is contained in another.

Definition 4.10 *Let M and N be two masked predicate maps.*

$$N \ddot{\subseteq} M$$

if for all $((p, l) \to I) \in N$ there exists $((p', l') \to I') \in M$ with

$$(p = p') \wedge (l' \sqsubseteq l) \wedge (I \subseteq I')$$

Lemma 4.6 *Let $I_1, I_2 \subseteq \mathbb{Z}$ be intervals, and M a masked predicate map. Then $(I_1 \oplus I_2) \dot{\oplus} M = I_1 \dot{\oplus} (I_2 \dot{\oplus} M)$.*

Proof This follows directly from the definitions of the two operators. We omit a detailed proof.

Lemma 4.7 *Let M be a masked predicate map. Then $\{0\} \dot{\oplus} M = M$.*

Proof This is a direct consequence of the definition of $\dot{\oplus}$. Adding zero to every interval in the map obviously does not change the intervals and thus it also does not change the map itself.

Lemma 4.8 *Let M and N be masked predicate maps, then*

$$M \ddot{\subseteq} M \ddot{\Cup} N$$

and

$$N \ddot{\subseteq} M \ddot{\Cup} N.$$

Proof Since the definition of $M \ddot{\Cup} N$ is symmetric, it suffices to show on of the two relations in the lemma. From Definition 4.9 it follows that any key consisting of a predicate and a mask is also in $M \ddot{\Cup} N$. Since the mask is there unaltered, it satisfies the condition of being no more precise than the one in M. The condition for $(I \subseteq I')$ is apparent from the definition as well, no interval is discarded and all intervals are either enlarged by the special union of intervals or kept the same.

Definition 4.11 *The extended relative interval of the formula φ, $\mathrm{ERI}(\varphi)$ is defined recursively over the formula structure:*

$$\mathrm{ERI}(\varphi) = \begin{cases} \{\} & \text{if } \varphi \text{ is an atomic formula and not a predicate,} \\ \{p(m) \to [0,0]\} & \text{if } \varphi \text{ is a predicate with name } p \text{ and mask } m, \\ \mathrm{ERI}(\psi) & \text{if } \varphi \text{ is of the form } \neg\psi, \exists x.\psi, \text{ or } \forall x.\psi, \\ \mathrm{ERI}(\psi) \,\ddot{\Cup}\, \mathrm{ERI}(\chi) & \text{if } \varphi \text{ is of the form } \psi \vee \chi, \text{ or } \psi \wedge \chi, \\ (-b,0] \,\dot{\Cup}\, ((-b,-a] \,\dot{\oplus}\, \mathrm{ERI}(\psi)) & \text{if } \varphi \text{ is of the form } \bullet_{[a,b)}\psi, \\ [0,b) \,\dot{\Cup}\, ([a,b) \,\dot{\oplus}\, \mathrm{ERI}(\psi)) & \text{if } \varphi \text{ is of the form } \bigcirc_{[a,b)}\psi, \\ (-b,0] \,\dot{\Cup}\, ((-b,0] \,\dot{\oplus}\, \mathrm{ERI}(\psi)) \,\ddot{\Cup}\, ((-b,-a] \,\dot{\oplus}\, \mathrm{ERI}(\chi)) & \text{if } \varphi \text{ is of the form } \psi \,\mathcal{S}_{[a,b)}\, \chi, \\ [0,b) \,\dot{\Cup}\, ([0,b) \,\dot{\oplus}\, \mathrm{ERI}(\psi)) \,\ddot{\Cup}\, ([a,b) \,\dot{\oplus}\, \mathrm{ERI}(\chi)) & \text{if } \varphi \text{ is of the form } \psi \,\mathcal{U}_{[a,b)}\, \chi, \\ [0,b) \,\dot{\Cup}\, ([0,b) \,\dot{\oplus}\, \mathrm{ERI}_{reg}(\psi)) & \text{if } \varphi \text{ is of the form } \rhd_{[a,b)}\, \psi, \text{ and} \\ (-b,0] \,\dot{\Cup}\, ((-b,0] \,\dot{\oplus}\, \mathrm{ERI}_{reg}(\psi)) & \text{if } \varphi \text{ is of the form } \blacktriangleleft_{[a,b)}\, \psi. \end{cases}$$

And for regular expressions we define

Definition 4.12 *The extended relative interval of the regular expression ρ, $\mathrm{ERI}_{reg}(\rho)$ is defined recursively over the structure of the regular expression:*

$$\mathrm{ERI}_{reg}(\rho) = \begin{cases} \{\} & \text{if } \rho \text{ is of the form } \star^k, \\ \mathrm{ERI}(\varphi) & \text{if } \rho \text{ is of the form } \varphi?, \\ \mathrm{ERI}_{reg}(\sigma) \,\ddot{\Cup}\, \mathrm{ERI}_{reg}(\tau) & \text{if } \rho \text{ is of the form } \sigma + \tau \text{ or } \sigma \cdot \tau, \text{ and} \\ \mathrm{ERI}_{reg}(\sigma) & \text{if } \rho \text{ is of the form } \sigma^*. \end{cases}$$

4.3.1 Correctness

Analogously to regular intervals we want to use the extended relative intervals to extract (slice) a sub trace from a larger trace for a given formula that has the property that the sub trace is sufficient to evaluate the formula. We first need a few definitions and lemmas before we can proof this property.

Definition 4.13 *Let $\mathcal{D}$ be a structure over the signature $S = (C, R, \iota)$, $r^{\mathcal{D}} \in |\mathcal{D}|$ the set of interpretations for the predicate $r \in R$, and m a mask for r. The arity of r, $\iota(r)$ is the same as the length of the mask m, $|m|$. The mask $m = (m_1, \ldots, m_{\iota(r)})$ matches the interpretation $k = (k_1, \ldots, k_{\iota(r)}) \in r^{\mathcal{D}}$ if for all $i \in [1, \iota(r)]$ either $m_i = v$ or $m_i = k_i$, where v is again the placeholder value for variable values. We write $m \rightsquigarrow k$ for m matches k.*

Lemma 4.9 *Let $m = (m_1, \ldots, m_{\iota(r)})$ and $n = (n_1, \ldots, n_{\iota(r)})$ be two masks for a predicate $r \in R$ with $m \sqsubseteq n$, where R is part of the signature $S = (C, R, \iota)$. Then for every interpretation x, $n \rightsquigarrow x$ implies that also $m \rightsquigarrow x$.*

Proof Note the order of appearance of m and n in the implication is different from that in the relation. From 4.7 it follows that for all $i \in \mathbb{N}$ with $0 \leq i \leq \iota(r)$, either $m_i = n_i$ or $m_i = v$. If $n \rightsquigarrow k$ then for all $i \in \mathbb{N}$ with $0 \leq i \leq \iota(r)$ either $n_i = v$ or $n_i = k_i$. We now show that for all i with $i \in \mathbb{N}$ and $0 \leq i \leq \iota(r)$ it also holds that either $m_i = k_i$ or $m_i = v$ by making a case distinction on the two possible values for each n_i. We make a case distinction on these two options

- $n_i = v$. In this case $m_i = v$ which satisfies one of the two options of $m_i = k_i$ or $m_i = v$.
- $n_i = k_i$. In this case $m = n_i$ or $m = v$ are both possible, but for both options either $m = n_i = k_i$ or $m = v$.

We can thus conclude that for all $i \in \mathbb{N}$ with $0 \leq i \leq \iota(r)$, $m_i = k_i$ or $m_i = v$ and thus $m \rightsquigarrow k$.

Definition 4.14 *Let M be a masked predicate map, $\tau_i, tau_j \in \mathbb{N}$ two time stamps, and $\mathcal{D}$ a structure over the signature $S = (C, R, \iota)$. Let $\mathbb{D}$ be the set of all structures over the signature S. $\text{filter}(M, \mathcal{D}, \tau_i, \tau_j) : \mathbb{D} \to \mathbb{D}$ is a function filtering the structure $\mathcal{D}$ on the condition whether τ_i is in the relative intervals of the masked predicates in M shifted by τ_j. Constant interpretations remain unchanged by the filtering. Concretely for all $r \in R$,*

$$r^{\text{filter}(M,\mathcal{D},\tau_i,\tau_j)} = \{v(\bar{t}) \in r^{\mathcal{D}} \mid \exists m, I.((r, m), I) \in M. \wedge m \rightsquigarrow v(\bar{t}) \wedge \tau_i - \tau_j \in I\}.$$

Intuitively τ_j can be seen as the reference time stamp and we check whether a time stamp τ_i lies inside the relative intervals from the perspective of τ_j. The following lemma states that filtering a structure twice has the same effect as filtering once.

Lemma 4.10 *Let $\mathcal{D}$ be a structure, let M be a masked predicate map and let $\tau_i, \tau_J \in \mathbb{N}$ be two time stamps. Then*

$$\text{filter}(M, \text{filter}(M, \mathcal{D}, \tau_i, \tau_j), \tau_i, \tau_j) = \text{filter}(M, \mathcal{D}, \tau_i, \tau_j).$$

Proof From Definition 4.14 we have that the constant interpretations remain unchanged by the filter operation. It remains to show that the left-hand side (LHS) and the right-hand side (RHS) in Lemma 4.10 have the same interpretations for predicates. By Definition 4.14 we have for all $r \in R$,

$$r^{\text{filter}(M,\mathcal{D},\tau_i,\tau_j)} = \{v(\bar{t}) \in r^{\mathcal{D}} \mid \exists m, I.((r, m), I) \in M \wedge m \rightsquigarrow v(\bar{t}) \wedge \tau_i - \tau_j \in I\}.$$

And also for all $r \in R$,

$$\begin{aligned} &r^{\text{filter}(M,\text{filter}(M,\mathcal{D},\tau_i,\tau_j),\tau_i,\tau_j)} \\ &= \{v(\bar{t}) \in r^{\text{filter}(M,\mathcal{D},\tau_i,\tau_j)} \mid \exists m, I.((r, m), I) \in M. \wedge m \rightsquigarrow v(\bar{t}) \wedge \tau_i - \tau_j \in I\}. \end{aligned}$$

From this it is apparent that the second set of interpretations, the one filtered twice is a subset of the first one. Further by them having the same conditional part, if an interpretation $v(\bar{t})$ is in $r^{\mathcal{D}}$ it must also be in $r^{\text{filter}(M,\mathcal{D},\tau_i,\tau_j)}$, because it must have already satisfied the same condition to be in $r^{\mathcal{D}}$.

As seen in the previous section Basin et al. [6] a T-slice in Definition A.2 for slicing a temporal structure in a temporal manner. We define an analogous notion of an MT_τ- slice which is a more fine-grained way of slicing that makes use of masked predicate maps.

Definition 4.15 *Let $T \subseteq \mathbb{Z}$ be an interval, and M a masked predicate map where for all $(m, J) \in M$, $J \subseteq T$. And let $(\bar{\mathcal{D}}, \bar{\tau})$ a temporal structure. The MT_τ- slice of $(\bar{\mathcal{D}}, \bar{\tau})$ is the time slice $(\bar{\mathcal{D}}', \bar{\tau}')$ of $(\bar{\mathcal{D}}, \bar{\tau})$, where $s : [0, l) \to \mathbb{N}$ is the function $s(i') = i' + c, l = |\{i \in \mathbb{N} \mid \tau_i \in T\}|$, and $c = \min\{i \in \mathbb{N} \mid \tau_i \in T\}$. We also require that $\tau'_l \notin T$ and*

$$\mathcal{D}'_{i'} = \text{filter}(M, \mathcal{D}_{s(i')}, \tau_{s(i')}, \tau),$$

for all $i' \in [0, l)$.

Definition 4.16 *Let $I \subseteq \mathbb{Z}$ be an interval, and M be a masked predicate map where for all $(m, J) \in M$, $J \subseteq I$. Let $c, i \in \mathbb{N}$. The temporal structures $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are (M, I, c, i)-overlapping if the following conditions hold:*

1. *$j \geq c$, $\tau_j = \tau'_{j-c}$, and*

$$\text{filter}(M, \mathcal{D}_j, \tau_j, \tau_i) = \text{filter}(M, \mathcal{D}'_{j-c}, \tau'_{j-c}, \tau_i)$$

   for all $j \in \mathbb{N}$ with $\tau_j - \tau_i \in I$,

2. *$\tau_{j'+c} = \tau'_{j'}$, and*

$$\text{filter}(M, \mathcal{D}_{j'+c}, \tau_{j'+c}, \tau_i) = \text{filter}(M, \mathcal{D}'_{j'}, \tau'_{j'}, \tau_i)$$

   for all $j' \in \mathbb{N}$ with $\tau'_{j'} - \tau_i \in I$,

The next lemma establishes that a trace and its MT_{τ_i}- slice are (M, T, c, i)- overlapping. It is analogous to Lemma A.2 in Basin et al. [6].

Lemma 4.11 *Let $T \subseteq \mathbb{N}$ and $I \subseteq \mathbb{Z}$ be intervals, $(\bar{\mathcal{D}}, \bar{\tau})$ a temporal structure, and $(\bar{\mathcal{D}}', \bar{\tau}')$ a $(T \dot{\oplus} M, T \oplus I)_{\tau_i}$- slice of $(\bar{\mathcal{D}}, \bar{\tau})$. The temporal structures $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are (M, I, c, i)- overlapping, for all $i \in \mathbb{N}$ with $\tau_i \in T$, where $c \in \mathbb{N}$ is the value in Definition 4.16 used by the function s with respect to $(\bar{\mathcal{D}}, \bar{\tau})$ and its slice $(\bar{\mathcal{D}}', \bar{\tau}')$.*

Proof This proof follows the same structure as the proof to Lemma A.2 in Basin et al. [6] and the first few steps are the same. The precondition in Definition 4.16, that every interval in M is contained in the interval T, is satisfied, as it is also a precondition for Lemma 4.11. First we show condition 1 in Definition 4.16. For all $j \in \mathbb{N}$ with $\tau_j - \tau_i \in T$, it follows from $\tau_i \in I$ that $\tau_j \in T \oplus I$. From $c = \min\{k \in \mathbb{N} \mid \tau_k \in T\}$ we get $j \geq c$. Even more precisely for all $j \in \mathbb{N}$ with $\tau_j \in T \oplus I$, we get that $j \in [c, l + c)$. From Definition 4.15 it follows that

$$\begin{aligned}\mathcal{D}'_{j-c} &= \text{filter}(T \dot{\oplus} M, \mathcal{D}_{s(j-c)}, \tau_{s(j-c)}, \tau_i) \\ &= \text{filter}(T \dot{\oplus} M, \mathcal{D}_{j-c+c}, \tau_{j-c+c}, \tau_i) \\ &= \text{filter}(T \dot{\oplus} M, \mathcal{D}_j, \tau_j, \tau_i),\end{aligned}$$

for all $j \in \mathbb{N}$ with $\tau_j - \tau_i \in T \oplus I$. With the help of Lemma 4.7 and Definition 4.14 (shifting all intervals in a map, shifts all time stamps that fall into these intervals the same way) we can convince ourselves that the above fact implies

$$\begin{aligned}\mathcal{D}'_{j-c} &= \text{filter}((-T \oplus T) \dot{\oplus} M, \mathcal{D}_{s(j-c)}, \tau_{s(j-c)}, \tau_i) \\ &= \text{filter}(\{0\} \dot{\oplus} M, \mathcal{D}_{j-c+c}, \tau_{j-c+c}, \tau_i) \\ &= \text{filter}(M, \mathcal{D}_j, \tau_j, \tau_i),\end{aligned}$$

for all $j \in \mathbb{N}$ with $\tau_j - \tau_i \in (-T \oplus T) \oplus I = I$. Thus we get

$$\mathcal{D}'_{j-c} = \text{filter}(M, \mathcal{D}_j, \tau_j, \tau_i),$$

for all $i \in \mathbb{N}$ with $\tau_i \in T$, and $j \in \mathbb{N}$ with $\tau_j - \tau_i \in I$. From Lemma 4.10 it follows that

$$\begin{aligned}\mathcal{D}'_{j-c} &= \text{filter}(M, \mathcal{D}_j, \tau_j, \tau_i) \\ &= \text{filter}(M, \text{filter}(M, \mathcal{D}_j, \tau_j, \tau_i), \tau_j, \tau_i) \\ &= \text{filter}(M, \mathcal{D}'_{j-c}, \tau_j, \tau_i) \\ &= \text{filter}(M, \mathcal{D}'_{j-c}, \tau'_{j-c}, \tau_i)\end{aligned}$$

for all $j \in \mathbb{N}$. And with that we have shown everything for Condition 1 in Definition 4.16.

For Condition 2 we can deduce that for all $i \in \mathbb{N}$ with $\tau_i \in T$ and for all $j' \in \mathbb{N}$ with $\tau'_{j'} - \tau_i \in I$, $\tau'_{j'} \in T \oplus I$. And because $\tau'_l \notin T \oplus I$, $j' \in [0, l)$. Hence $\tau_{j'+c} = \tau_{s(j')} = \tau'_{j'}$. From Definition 4.15 it follows that

$$\begin{aligned}\mathcal{D}'_{j'} &= \text{filter}(T \dot{\oplus} M, \mathcal{D}_{s(j')}, \tau_{s(j')}, \tau_i) \\ &= \text{filter}(T \dot{\oplus} M, \mathcal{D}_{j'+c}, \tau_{j'+c}, \tau_i)\end{aligned}$$

for all $j' \in \mathbb{N}$ with $\tau'_j - \tau_i \in T \oplus I$. Again with the help of Lemma 4.7 and Definition 4.14 we can convince ourselves that the above fact implies

$$\begin{aligned}\mathcal{D}'_{j'} &= \text{filter}((-T \oplus T) \dot{\oplus} M, \mathcal{D}_{s(j')}, \tau_{s(j')}, \tau_i) \\ &= \text{filter}(\{0\} \dot{\oplus} M, \mathcal{D}_{j'+c}, \tau_{j'+c}, \tau_i) \\ &= \text{filter}(M, \mathcal{D}_{j'+c}, \tau_{j'+c}, \tau_i),\end{aligned}$$

for all $j' \in \mathbb{N}$ with $\tau'_j - \tau_i \in (-T \oplus T) \oplus I = I$. Thus we get

$$\mathcal{D}'_{j'} = \text{filter}(M, \mathcal{D}_{j'+c}, \tau_{j+c}, \tau_i),$$

And like with condition 1, from Lemma 4.10 it follows that

$$\begin{aligned}
\mathcal{D}'_{j'} &= \text{filter}(M, \mathcal{D}_{j'+c}, \tau_{j'+c}, \tau_i) \\
&= \text{filter}(M, \mathcal{D}_{j'+c}, \tau_{j'+c}, \tau_i) \\
&= \text{filter}(M, \text{filter}(M, \mathcal{D}_{j'+c}, \tau_{j'+c}, \tau_i), \tau_{j'+c}, \tau_i) \\
&= \text{filter}(M, \mathcal{D}'_{j'}, \tau_{j'+c}, \tau_i) \\
&= \text{filter}(M, \mathcal{D}'_{j'}, \tau'_{j'}, \tau_i),
\end{aligned}$$

for all $j' \in \mathbb{N}$. And with that we have shown everything for Condition 2 in Definition 4.16.

Analogous to Lemma A.3 in Basin et al. [6] the following Lemma states that any overlapping traces also overlap on parts of the overlapping section.

Lemma 4.12 *Let $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ be two temporal structures that are (M, I, c, i)-overlapping, for some masked predicate map M, $I \subseteq \mathbb{Z}$, $c \in \mathbb{N}$, and $i \in \mathbb{Z}$. Then $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are (N, K, c, k)-overlapping, for each $k \in \mathbb{N}$ with $\tau_k - \tau_i \in I$, $K \subseteq \{\tau_i - \tau_k\} \oplus I$, and $N \ddot{\subseteq} \{\tau_i - \tau_k\} \dot{\oplus} M$.*

Proof This proof follows a similar structure as the proof of Lemma A.3 in Basin et al. [6]. We first show that Condition 1 in Definition 4.16 is satisfied, i.e. for all $j \in \mathbb{N}$ with $\tau_j - \tau_k \in K$, the following three things hold: $j \geq c$, $\tau_j = \tau'_{j-c}$, and $\text{filter}(N, \mathcal{D}_j, \tau_j, \tau_k) = \text{filter}(N, \mathcal{D}'_{j-c}, \tau'_{j-c}, \tau_k)$.

For all $j \in \mathbb{N}$ with $\tau_j - \tau_k \in K$, it follows that $\tau_j - \tau_k + \tau_k - \tau_i \in \{\tau_k - \tau_i\} \oplus K$, it then follows that $\tau_j - \tau_i \in \{\tau_k - \tau_i\} \oplus K$. From $K \subseteq \{\tau_i - \tau_k\} \oplus I$, we get $\{\tau_k - \tau_i\} \oplus K \subseteq \{\tau_k - \tau_i\} \oplus \{\tau_i - \tau_k\} \oplus I = I$, or in short $\tau_j - \tau_i \in I$. And as we know that $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are (M, I, c, i)-overlapping, it follows from Condition 1 in Definition 4.16 that for all $j \in \mathbb{N}$ with $\tau_j - \tau_k \in K$, $j \geq c$, $\tau_j = \tau'_{j-c}$, and $\text{filter}(M, \mathcal{D}_j, \tau_j, \tau_i) = \text{filter}(M, \mathcal{D}'_{j-c}, \tau'_{j-c}, \tau_i)$.

For any interpretation $x \in r^{\text{filter}(N, \mathcal{D}_j, \tau_j, \tau_k)}$, it follows that there exists a mask m for r that matches x and has a relative interval T_{rm} with $\tau_j - \tau_k \in T_{rm}$. Since $N \ddot{\subseteq} \{\tau_i - \tau_k\} \dot{\oplus} M$, there must exist a mask m' for r with $m' \sqsubseteq m$ and a relative interval $T_{rm'}$, for which $\tau_j - \tau_k \in \{\tau_i - \tau_k\} \oplus T_{rm'}$. It follows that $\tau_k - \tau_i + \tau_j - \tau_k \in \{\tau_k - \tau_i\} \oplus \{\tau_i - \tau_k\} \oplus T_{rm'}$. And hence $\tau_j - \tau_i \in T_{rm'}$. From

$$\text{filter}(M, \mathcal{D}_j, \tau_j, \tau_i) = \text{filter}(M, \mathcal{D}'_{j-c}, \tau'_{j-c}, \tau_i),$$

for all $j \in \mathbb{N}$ with $\tau_j - \tau_i \in I$ and $T_{rm'} \subseteq I$, it follows that $\tau'_{j-c} - \tau_i \in T_{rm'}$. Moreover, we observe that for any δ, we have

$$\text{filter}(M, \mathcal{D}_j, \tau_j, \tau_i) = \text{filter}(\{\delta\} \dot{\oplus} M, \mathcal{D}'_{j'}, \tau'_{j'}, \tau_i - \delta).$$

We can convince ourselves that this holds by looking at Definitions 4.9 and 4.14. First $\dot{\oplus}$ shifts all intervals in a map. Then filter checks for any interpretation of a predicate if it has a matching mask in the masked predicate map and if a time stamp is in the relative interval of that mask. It is now easy to see that, if the original time stamp was in the original interval, the shifted time stamp is also in the shifted interval. And hence, by choosing $\delta = \tau_i - \tau_k$, we get

$$\text{filter}(\{\tau_i - \tau_k\} \dot{\oplus} M, \mathcal{D}_j, \tau_j, \tau_k) = \text{filter}(\{\tau_i - \tau_k\} \dot{\oplus} M, \mathcal{D}'_{j'}, \tau'_{j'}, \tau_k)$$

for all $j \in \mathbb{N}$ with $\tau_j - \tau_k \in K$. Since $N \ddot{\subseteq} \{\tau_i - \tau_k\} \dot{\oplus} M$ with $\tau_i - \tau_k \in I$, we get

$$\text{filter}(N, \mathcal{D}_j, \tau_j, \tau_k) = \text{filter}(N, \mathcal{D}'_{j'}, \tau'_{j'}, \tau_k).$$

With this Condition 1 of Definition 4.16 holds for $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ to be (N, K, c, k)- overlapping.

To show that Condition 2 of Definition 4.16 also holds, we can start by looking at all $j' \in \mathbb{N}$ with $\tau'_{j'} - \tau_k \in K$. From our work for the first part it follows that $\tau'_{j'} - \tau_i \in I$. Hence it follows from Condition 2 of Definition 4.16 and $(\bar{\mathcal{D}}, \bar{\tau})$, the observation we used to show Condition 1, and $(\bar{\mathcal{D}}', \bar{\tau}')$ are (M, I, c, k)- overlapping, that they are also (N, K, c, k)- overlapping.

With this we get to the key Lemma that we rely on to make our optimization when selecting and querying a sub-trace from the database. It is analogous to Lemma A.4 in Basin et al. [6].

Lemma 4.13 *Let ϕ be a formula and $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ temporal structures. If $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are $(\mathrm{ERI}(\phi), \mathrm{RI}(\phi), c, i)$-overlapping, for some c and i, then for all valuations v, it holds that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \phi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i - c) \models \phi$.*

This Lemma applies to MFOTL formulas. It can be extended to MFODL as well, but it needs a mutually recursive extensions to include regular expressions. We focus on the core MFOTL formulas.

Proof This proof is in large parts analogous to the one to Lemma A.4 in Basin et al. [6], and we follow the same structure. First we note that for the same reason as in Lemma A.4 in Basin et al. [6], the lemma's statement is well-defined, as the definition of $\mathrm{RI}(\phi)$ did not change and still includes 0. We proof the lemma by structural induction on the formula ϕ. In this proof $S = (C, R, \iota)$ is a signature, with constants C, predicates R, and the arity function ι. Similarly V is the set of variables. We have the following cases.

- $t \approx t'$, where $t, t' \in V \cup C$. The satisfaction of $t \approx t'$ depends only on the valuation v. Therefore it follows that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models t \approx t'$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i - c) \models t \approx t'$.

- $t \prec t'$ and $t \preceq$ are analogous to the first one and their detailed proofs are omitted.

- $r(\bar{t})$, where $t_1, \ldots, t_{\iota(r)} \in V \cup C$. Since $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are $(\mathrm{ERI}(\phi), \mathrm{RI}(\phi), c, i)$-overlapping it follows from Condition 1 in Definition 4.16 that

  $$\mathrm{filter}(\mathrm{ERI}(\phi), \mathcal{D}_j, \tau_j, \tau_i) = \mathrm{filter}(\mathrm{ERI}(\phi), \mathcal{D}'_{j-c}, \tau'_{j-c}, \tau_i),$$

  $\tau_j = \tau'_{j-c}$, and $j \geq c$ for all $j \in \mathbb{N}$ with $\tau_j - \tau_i \in \mathrm{RI}(\phi)$. By Definition 4.14, there exists a valuation $v(\bar{t})$ in $r^{\mathcal{D}_j}$ iff there exist m, J, with $((r, m), J) \in \mathrm{ERI}(\phi)$ with $\tau_j - \tau_i \in J$, and $m \rightsquigarrow v(\bar{t})$. And from the equality of the two filters given to us by Condition 1 of Definition 4.16 this is the case iff there exist m', J', with $((r, m'), J') \in \mathrm{ERI}(\phi)$ with $\tau'_{j-c} - \tau_i \in J'$, and $m' \rightsquigarrow v(\bar{t})$ for all valuations v. And further this last part holds iff $v(\bar{t}) \in r^{\mathcal{D}'_{j-c}}$. Thus, by Definition, 2.2 $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models r(\bar{t})$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i - c) \models r(\bar{t})$.

- $\neg\psi$. $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are $(\mathrm{ERI}(\phi), \mathrm{RI}(\phi), c, i)$-overlapping. By definition $\mathrm{ERI}(\neg\psi) = \mathrm{ERI}(\psi)$ and $\mathrm{RI}(\neg\psi) = \mathrm{RI}(\psi)$. By the inductive hypothesis $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \psi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i - c) \models \psi$ for all valuations v. Therefore $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \neg\psi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i - c) \models \neg\psi$ for all valuations v.

- $\psi \vee \chi$. $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are $(\mathrm{ERI}(\psi) \mathbin{\ddot{\cup}} \mathrm{ERI}(\chi), \mathrm{RI}(\psi) \cup \mathrm{RI}(\chi), c, i)$-overlapping. From $\mathrm{ERI}(\psi) \ddot{\subseteq} \mathrm{ERI}(\psi) \mathbin{\ddot{\cup}} \mathrm{ERI}(\chi)$, $\mathrm{ERI}(\chi) \ddot{\subseteq} \mathrm{ERI}(\psi) \mathbin{\ddot{\cup}} \mathrm{ERI}(\chi)$, $\mathrm{RI}(\psi) \subseteq \mathrm{RI}(\psi) \cup \mathrm{RI}(\chi)$, and $\mathrm{RI}(\chi) \subseteq \mathrm{RI}(\psi) \cup \mathrm{RI}(\chi)$, it follows from Lemma 4.12 that $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are $(\mathrm{ERI}(\psi), \mathrm{RI}(\psi), c, i)$- and $(\mathrm{ERI}(\chi), \mathrm{RI}(\chi), c, i)$-overlapping. By the inductive hypothesis, $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \psi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i - c) \models \psi$ and $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \chi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i - c) \models \chi$. Therefore $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \psi \vee \chi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i - c) \models \psi \vee \chi$.

- $\exists x.\psi$. From $\mathrm{ERI}(\exists x.\psi) = \mathrm{ERI}(\psi)$ and $\mathrm{RI}(\exists x.\psi) = \mathrm{RI}(\psi)$ it follows that $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are $(\mathrm{ERI}(\psi), \mathrm{RI}(\psi), c, i)$-overlapping. By the inductive hypothesis we have $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \psi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i - c) \models \psi$ for all valuations v. Thus for all d, $(\bar{\mathcal{D}}, \bar{\tau}, v[x \mapsto d], i) \models \psi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v[x \mapsto d], i - c) \models \psi$. By Definition 2.2 it follows directly that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \exists x.\psi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i - c) \models \exists x.\psi$ for all valuations v.

- $\bullet_{[a,b)}\psi$. $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are $(\mathrm{ERI}(\bullet_{[a,b)}\psi), \mathrm{RI}(\bullet_{[a,b)}\psi), c, i)$- overlapping, where

  $$\mathrm{ERI}(\bullet_{[a,b)}\psi) = (-b, 0] \mathbin{\ddot{\cup}} ((-b, -a] \mathbin{\dot{\oplus}} \mathrm{ERI}(\psi))$$

  and

  $$\mathrm{RI}(\bullet_{[a,b)}\psi) = (-b, 0] \cup ((-b, -a] \oplus \mathrm{RI}(\psi))$$

  From $0 \in \mathrm{RI}(\bullet_{[a,b)}\psi)$ and Condition 1 in Definition 4.16, it follows that $\tau_i = \tau'_{i-c}$.

  We make a case split on the value of i. If $i = 0$, then $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \bullet_{[a,b)}\psi$, for all valuations v. From $c \in \mathbb{N}, i - c \in \mathbb{N}$, and $i = 0$, it follows that $i - c = 0$. Since $i - c = 0 = i$ it is trivial that $(\bar{\mathcal{D}}', \bar{\tau}', v, i - c) \not\models \bullet_{[a,b)}\psi$ for all valuations v.

  Next we consider $i > 0$ and make another case split on whether $\tau_i - \tau_{i-1} \in [a, b)$.

– $\tau_i - \tau_{i-1} \in [a,b)$. Then $\tau_{i-1} - \tau_i \in \mathrm{RI}(\bullet_{[a,b)}\psi)$ and from Condition 1 in Definition 4.16 it follows that $i-1 \geq c$, $\tau_{i-1} = \tau'_{i-c-1}$, and hence $\tau'_{i-c} - \tau'_{i-c-1} \in [a,b)$. It further follows that

$$\begin{aligned}\mathrm{ERI}(\psi) &\ddot{\subseteq} \{\tau_i - \tau_{i-1}\} \dot{\oplus} \{\tau_{i-1} - \tau_i\} \dot{\oplus} \mathrm{ERI}(\psi)\\ &\ddot{\subseteq} \{\tau_i - \tau_{i-1}\} \dot{\oplus} (-b,-a] \dot{\oplus} \mathrm{ERI}(\psi)\\ &\ddot{\subseteq} \{\tau_i - \tau_{i-1}\} \dot{\oplus} \mathrm{ERI}(\bullet_{[a,b)}\psi)\end{aligned}$$

and

$$\begin{aligned}\mathrm{RI}(\psi) &\subseteq \{\tau_i - \tau_{i-1}\} \oplus \{\tau_{i-1} - \tau_i\} \oplus \mathrm{RI}(\psi)\\ &\subseteq \{\tau_i - \tau_{i-1}\} \oplus (-b,-a] \oplus \mathrm{RI}(\psi)\\ &\subseteq \{\tau_i - \tau_{i-1}\} \oplus \mathrm{RI}(\bullet_{[a,b)}\psi).\end{aligned}$$

Thus by Lemma 4.12 $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are $(\mathrm{ERI}(\psi), \mathrm{RI}(\psi), c, i-1)$- overlapping. By the induction hypothesis $(\bar{\mathcal{D}}, \bar{\tau}, v, i-1) \models \psi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i-c-1) \models \psi$ for all valuations v. Because $\tau_i = \tau'_{i-c}$ and $\tau_{i-1} = \tau'_{i-c-1}$, it follows that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \bullet_{[a,b)}\psi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i-c) \models \bullet_{[a,b)}\psi$ for all valuations v.

– $\tau_i - \tau_{i-1} \notin [a,b)$. Then $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \bullet_{[a,b)}\psi$, for all valuations v. From Condition 1 of Definition 4.16 we still have $i \geq c$. We make a case distinction on whether $i = c$ or $i > c$.

  * $i = c$. In this case $i - c = 0$. Therefore $(\bar{\mathcal{D}}, \bar{\tau}, v, i-c) \not\models \bullet_{[a,b)}\psi$, for all valuations v.
  * $i > c$. We proof this case by contradiction. Assume that $\tau'_{i-c} - \tau'_{i-c-1} = \tau'_{i-c} - \tau'_{i-c-1} \in [a,b)$. From Condition 2 in Definition 4.16 it follows that $\tau_{i-1} = \tau'_{i-c-1}$ and hence $\tau_i - \tau_{i-1} = \tau'_{i-c} - \tau'_{i-c-1} \in [a,b)$. This contradicts $\tau_i - \tau_{i-1} \notin [a,b)$, so it must be the case that $\tau'_{i-c} - \tau'_{i-c-1} \notin [a,b)$. It follows that $(\bar{\mathcal{D}}', \bar{\tau}', v, i-c) \not\models \bullet_{[a,b)}\psi$, for all valuations v.

- $\bigcirc_{[a,b)}$. The "next" case is in large parts similar to the one for "previous". Like with Lemma A.4 [6] it is simpler than "previous", because no special distinctions have to be made for $i = 0$ and $i - c = 0$.

- $\psi \,\mathcal{S}_{[a,b)}\, \chi$. Like the other cases this one is also very similar to the same case in the proof to Lemma A.4 [6] and we follow the same structure. $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are $(\mathrm{ERI}(\psi \,\mathcal{S}_{[a,b)}\, \chi), \mathrm{RI}(\psi \,\mathcal{S}_{[a,b)}\, \chi), v, i)$-overlapping, where

$$\mathrm{ERI}(\psi \,\mathcal{S}_{[a,b)}\, \chi) = (-b,0] \,\dot{\Cup}\, ((-b,0] \dot{\oplus} \mathrm{ERI}(\psi)) \,\dot{\Cup}\, ((-b,-a] \dot{\oplus} \mathrm{ERI}(\chi))$$

and

$$\mathrm{RI}(\psi \,\mathcal{S}_{[a,b)}\, \chi) = (-b,0] \cup ((-b,0] \oplus \mathrm{RI}(\psi)) \cup ((-b,-a] \oplus \mathrm{RI}(\chi)).$$

From $0 \in \mathrm{RI}(\psi \,\mathcal{S}_{[a,b)}\, \chi)$ and Condition 1 in Definition 4.16 it follows that $\tau_i = \tau'_i - c$.

First we establish the two following claims, by applying the inductive hypothesis to the two sub-formulas ψ and χ.

(i) (χ) For all $j \in \mathbb{N}$ with $j \leq i$ and $\tau_i - \tau_j \in [a,b)$, it holds that

$$\begin{aligned}\mathrm{RI}(\chi) &\subseteq \{\tau_i - \tau_j\} \oplus \{\tau_j - \tau_i\} \oplus \mathrm{RI}(\chi)\\ &\subseteq \{\tau_i - \tau_j\} \oplus (-b,-a] \oplus \mathrm{RI}(\chi)\\ &\subseteq \{\tau_i - \tau_j\} \oplus \mathrm{RI}(\psi \,\mathcal{S}_{[a,b)}\, \chi)\end{aligned}$$

as well as

$$\begin{aligned}\mathrm{ERI}(\chi) &\ddot{\subseteq} \{\tau_i - \tau_j\} \dot{\oplus} \{\tau_j - \tau_i\} \dot{\oplus} \mathrm{ERI}(\chi)\\ &\ddot{\subseteq} \{\tau_i - \tau_j\} \dot{\oplus} (-b,-a] \dot{\oplus} \mathrm{ERI}(\chi)\\ &\ddot{\subseteq} \{\tau_i - \tau_j\} \dot{\oplus} \mathrm{ERI}(\psi \,\mathcal{S}_{[a,b)}\, \chi)\end{aligned}$$

and $j \geq c$. We can convince ourselves that the second step in this equation holds, by considering how the $\dot{\oplus}$ operator shifts and or resizes the relative intervals inside a masked predicate map. By Lemma 4.12 $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are $(\mathrm{ERI}(\chi), \mathrm{RI}(\chi), c, j)$-overlapping. It then follows from the inductive hypothesis that

$$(\bar{\mathcal{D}}, \bar{\tau}, v, j) \models \chi \text{ iff } (\bar{\mathcal{D}}', \bar{\tau}', v, j - c) \models \chi,$$

for all valuations v.

(ii) (ψ) For all $k \in \mathbb{N}$ with $k \leq i$ and $\tau_i - \tau_k \in [0, b)$, it holds that

$$\begin{aligned} \mathrm{RI}(\psi) &\subseteq \{\tau_i - \tau_k\} \oplus \{\tau_k - \tau_i\} \oplus \mathrm{RI}(\psi) \\ &\subseteq \{\tau_i - \tau_k\} \oplus (-b, 0] \oplus \mathrm{RI}(\psi) \\ &\subseteq \{\tau_i - \tau_k\} \oplus \mathrm{RI}(\psi \,\mathcal{S}_{[a,b)}\, \chi) \end{aligned}$$

as well as

$$\begin{aligned} \mathrm{ERI}(\psi) &\ddot{\subseteq} \{\tau_i - \tau_k\} \dot{\oplus} \{\tau_k - \tau_i\} \dot{\oplus} \mathrm{ERI}(\psi) \\ &\ddot{\subseteq} \{\tau_i - \tau_k\} \dot{\oplus} (-b, 0] \dot{\oplus} \mathrm{ERI}(\psi) \\ &\ddot{\subseteq} \{\tau_i - \tau_k\} \dot{\oplus} \mathrm{ERI}(\psi \,\mathcal{S}_{[a,b)}\, \chi) \end{aligned}$$

and $k \geq c$. By Lemma 4.12 $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are $(\mathrm{ERI}(\psi), \mathrm{RI}(\psi), c, k)$- overlapping. It follows from the inductive hypothesis that

$$(\bar{\mathcal{D}}, \bar{\tau}, v, k) \models \psi \text{ iff } (\bar{\mathcal{D}}', \bar{\tau}', v, k - c) \models \psi,$$

for all valuations v.

We show each direction of the claimed equivalence, $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \psi \,\mathcal{S}_{[a,b)}\, \chi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i-c) \models \psi \,\mathcal{S}_{[a,b)}\, \chi$, for all valuations v, separately and start with the direction from left to right. If $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \psi \,\mathcal{S}_{[a,b)}\, \chi$ then there is some $j \leq i$ with $\tau_i - \tau_j \in [a, b)$ such that $(\bar{\mathcal{D}}, \bar{\tau}, v, j) \models \chi$ and $(\bar{\mathcal{D}}, \bar{\tau}, v, k) \models \psi$, for all $k \in [j + 1, i + 1)$.

From $\tau_i - \tau_j \in [a, b)$ it follows that $\tau_j - \tau_i \in \mathrm{RI}(\psi \,\mathcal{S}_{[a,b)}\, \chi)$ and from Condition 1 in Definition 4.16 it follows that $j \geq c$ and $\tau_j = \tau'_{j-c}$. From Claim (i) and from $(\bar{\mathcal{D}}, \bar{\tau}, v, j) \models \chi$ it follows that $(\bar{\mathcal{D}}', \bar{\tau}', v, j - c) \models \chi$.

For all $k' \in [j + 1 - c, i + 1 - c)$, it holds that $\tau'_{k'} - \tau'_{i-c} = \tau'_{k'} - \tau_i \in (-b, 0]$ and hence $\tau'_{k'} - \tau_i \in \mathrm{RI}(\psi \,\mathcal{S}_{[a,b)}\, \chi)$. It follows from Condition 2 in Definition 4.16 that $\tau_{k'+c} = \tau'_{k'}$. With $(\bar{\mathcal{D}}, \bar{\tau}, v, k' + c) \models \psi$ and Claim (ii) we get $(\bar{\mathcal{D}}', \bar{\tau}', v, k') \models \psi$. And with that $(\bar{\mathcal{D}}', \bar{\tau}', v, i - c) \models \psi \,\mathcal{S}_{[a,b)}\, \chi$. This concludes the proof in the first direction.

Next we show the other direction of the equivalence, i.e. if $(\bar{\mathcal{D}}', \bar{\tau}', v, i - c) \models \psi \,\mathcal{S}_{[a,b)}\, \chi$ then $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \psi \,\mathcal{S}_{[a,b)}\, \chi$. This is equivalent to $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \psi \,\mathcal{S}_{[a,b)}\, \chi$ then $(\bar{\mathcal{D}}', \bar{\tau}', v, i - c) \not\models \psi \,\mathcal{S}_{[a,b)}\, \chi$. Like in the proof to Lemma A.4 [6], we show the second implication. From Definition 2.2 it follows that there are two possibilities for $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \psi \,\mathcal{S}_{[a,b)}\, \chi$. The first is that for no $j \leq i$, $\tau_i - \tau_j \in [a, b)$ and $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \chi$. Or written differently, for all $j \leq i$ with $\tau_i - \tau_j \in [a, b)$, $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \chi$. From the Definition 4.1 it follows that for all $j' \leq i - c$ with $\tau'_{i-c} - \tau'_{j'} = \tau_i - \tau'_{j'} \in [a, b)$, $\tau'_{j'} - \tau_i \in \mathrm{RI}(\psi \,\mathcal{S}_{[a,b)}\, \chi)$. This is a useful property, because it means we can conclude from Condition 2 of Definition 4.16 that $\tau_{j'+c} = \tau'_{j'}$ for all $j' \leq i - c$. And as in the proof to Lemma A.4 [6] this means that there cannot be a time stamp within the interval $[a, b)$ in $(\bar{\mathcal{D}}', \bar{\tau}')$ that is not present in $(\bar{\mathcal{D}}, \bar{\tau})$. We are currently looking at the case where for all $j \leq i$, $(\bar{\mathcal{D}}, \bar{\tau}, v, j) \not\models \chi$. Thus also $(\bar{\mathcal{D}}, \bar{\tau}, v, j' + c) \not\models \chi$ for all $j' \leq i - c$. Combining this with Claim (i) from above we get that $(\bar{\mathcal{D}}', \bar{\tau}', v, j') \not\models \chi$. And by Definition 2.2 it follows that $(\bar{\mathcal{D}}', \bar{\tau}', v, i - c) \not\models \psi \,\mathcal{S}_{[a,b)}\, \chi$.

The second possibility for $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \psi \,\mathcal{S}_{[a,b)}\, \chi$ is, that for all $j \leq i$ with $\tau_i - \tau_j \in [a, b)$ and $(\bar{\mathcal{D}}, \bar{\tau}, v, j) \models \chi$ there is some $k \in \mathbb{N}$ with $j < k \leq i$, $(\bar{\mathcal{D}}, \bar{\tau}, v, k) \not\models \psi$. It follows from this

that for every $j' \in \mathbb{N}$ with $j' \leq i - c$, $\tau'_{i-c} - \tau'_{j'} \in [a, b)$, and $(\bar{\mathcal{D}}', \bar{\tau}', v, j') \models \chi$, there exists a $j \in \mathbb{N}$ with $j = j' + c$. It follows from $\tau'_{i-c} - \tau'_{j'} \in [a, b)$ and $\tau'_{i-c} = \tau_i$, that $\tau'_{j'} - \tau_i \in (-b, -a]$ and further with the help of Definition 4.1 it follows that $\tau'_{j'} - \tau_i \in \mathrm{RI}(\psi \, \mathcal{S}_{[a,b)} \, \chi)$. From this we recall Condition 2 from Definition 4.16 and see that $\tau_j = \tau'_{j-c} = \tau' j'$. Since $j' \leq i - c$ and $j = j' + c$ it follows that $j \leq i$. By having shown that $\tau'_{j'} = \tau_j$ and $j \leq i$, we can now make use of Claim (i) from above again and combine it with $(\bar{\mathcal{D}}, \bar{\tau}, v, j) \models \chi$. It follows that $(\bar{\mathcal{D}}', \bar{\tau}', v, j' + c) \models \chi$. From $\tau'_{j'} = \tau_{j+c} = \tau_j$ it follows that $\tau_k = \tau'_{k-c}$.

We can use Claim (i) for j, because $\tau'_{j'} = \tau_j$ and $j \leq i$. Combining Claim (i) with $(\bar{\mathcal{D}}', \bar{\tau}', v, j - c) \models \chi$ yields that $(\bar{\mathcal{D}}, \bar{\tau}, v, j) \models \chi$. And for this we have already established that there must exist a $k \in \mathbb{N}$ with $j < k \leq i$ and $(\bar{\mathcal{D}}, \bar{\tau}, v, k) \not\models \psi$. It follows from $\tau'_{i-c} - \tau'_{j'} \in [a, b)$ that $\tau_i - \tau_j \in [a, b)$. And with $j < k \leq i$ it follows that $\tau_i - \tau_k \in [0, b)$. This means Claim (ii) from above is applicable to k and with it, it follows from $(\bar{\mathcal{D}}, \bar{\tau}, v, k) \not\models \psi$ that $(\bar{\mathcal{D}}', \bar{\tau}', v, k - c) \not\models \psi$. And thus $(\bar{\mathcal{D}}', \bar{\tau}', v, k - c) \not\models \psi \, \mathcal{S}_{[a,b)} \, \chi$.

- $\psi \, \mathcal{U}_{[a,b)} \, \chi$. The "Until" case is analogous to the "Since" case.

For the MFODL cases, just like for "normal" regular intervals an additional lemma is needed, the proof of needs to be done over both of them, as they are mutually recursive. We state this additional lemma here, but forgo the proof for time reasons.

Lemma 4.14 *Let ρ be a regular expression and $(\bar{\mathcal{D}}, \bar{\tau}), (\bar{\mathcal{D}}', \bar{\tau}')$ temporal structures. If $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are $(\mathrm{ERI}_{reg}(\rho), \mathrm{RI}_{reg}(\rho), c, i)$-overlapping, for some c and i, then for all valuations v and $j \in \mathbb{N}$, it holds that $(\bar{\mathcal{D}}, \bar{\tau}, v, i, j) \models_r \rho$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i - c, j - c) \models_r \rho$.*

We finish this section the same way we finished the section about "normal" regular intervals, by showing how the extended relative intervals can be used to extract a sub-trace that assigns a formula ϕ the same truth value at the "current" time point as the original trace. Let k be the current time point with time stamp τ_k, and let $(\bar{\mathcal{D}}, \bar{\tau})$ be a trace, and $(\bar{\mathcal{D}}', \bar{\tau}')$ the $(\{\tau_k\} \dot{\oplus} M, \{\tau_k\} \oplus I)$- slice of $(\bar{\mathcal{D}}, \bar{\tau})$. Then it follows from Lemma 4.11 that $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are (M, I, c, k)- overlapping. And from Lemma 4.13 it follows that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \phi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i - c) \models \phi$. Thus at the "current" time point k with time stamp $\tau_k = \tau'_{k-c}$ the two traces assign the formula ϕ the same truth value.

4.4 Conversion to SQL-Query

Now let's consider how we can convert the extended relative intervals as we have them in Figure 4.3 to a SQL query. On a high level we have one `SELECT` query for each table corresponding to a predicate. The different masks and their relative intervals are appended as conjunctions with a single `WHERE` clause. Let's assume we have a predicate p with $n \in \mathbb{N}$ mask- relative interval tuples, $((m_1, r_1), \ldots (m_n, r_n))$. Variable values in a mask are disregarded. If the constant values in a mask $m_i = (m_{i1}, \ldots, m_{i\iota(p)})$ have the indices $j_1, \ldots, j_l$, with $l \leq \iota(p)$ and the relative interval is $[a, b)$, then the corresponding SQL clause is `WHERE (x<`j_1`> =` m_{ij_1} `AND ... AND x<`j_l`> AND ts >=` a `AND ts <` b`)`. Depending on whether an interval bound is open or closed, the corresponding comparison operators are used. If an interval is unbounded, then that bound is omitted and all events are retrieved. When multiple masks and intervals for a predicate exist, multiple of the described conditions are concatenated with `AND`. This approach adds potentially overlapping conditions. A good query processor will hopefully recognize those and make optimizations when appropriate. We run every query for each predicate on the database. One additional query is necessary for the `ts` table that gets all time points in the relative interval covering the whole formula. For our example policy in Figure 2.2 with the extended relative intervals as seen in Figure 4.3 and the schema in Figure 2.3.2 we get the following SQL query.

```
FROM loc_accessed SELECT * WHERE (x2 = "advertising" AND
                                  time_stamp >= τ_i − 0 AND
                                  time_stamp <= τ_i − 0)
FROM perm_granted SELECT * WHERE (time_stamp <= τ_i − 0)
FROM perm_revoked SELECT * WHERE (time_stamp <= τ_i − 0)
FROM ts           SELECT * WHERE (time_stamp <= τ_i − 0)
```

Figure 4.2: Extended Relative Intervals of Example Policy

The time stamp is converted to a Unix style time stamp string that can be understood by QuestDB [21].

Chapter 5

Implementation and Evaluation

5.1 Wrapping MonPoly

We opted to use Python as a programming language for our wrapper. It is a powerful high level language that is widely used and has a large ecosystem of tools available. While we tried to keep them to a minimum, some changes directly to MonPoly itself were necessary for the wrapper to work.

The wrapper code can be divided into two core parts. The first part is responsible for providing the REST API endpoints and handling user input. One might call this part the front end of the wrapper. For this we use Flask [12], a lightweight web framework for Python. It offers a convenient way of specifying the endpoints and handling web requests.

The second part handles the interactions with MonPoly and QuestDB. One might call this the backend of the wrapper.

The distinction of these parts is reflected in the way we split the code into different files. There are just three of them, in the "src" directory of the repository [13]. The first one is "app.py" and it directly corresponds to the first part. It is conventional to have a file called "app.py". The "app.py" is to a Flask project, what a "main.py" file would be to regular python projects. Both of these are however merely conventions. Inside "app.py" there is one method for each REST API endpoint as described in the Architecture chapter.

The remaining two files, "monitor.py" and "db_helper.py" correspond to the second part from above. In "monitor.py" we define a "monitor" class, which is the heart of the wrapper. During runtime there is always one single "monitor"- object, that keeps track of everything from the current signature, the current policy, how to reach QuestDB, to a potential MonPoly subprocess. "db_helper.py" is a small class, defining a "db_helper" class that keeps track of the QuestDB configuration details, like ports, hostname, or the username. The "monitor"- class has a "db_helper"- object as a member.

The wrapper runs MonPoly as a subprocess and handles all interactions with MonPoly itself. This subprocess is a member of the "monitor"- object and any interaction with the subprocess goes through this class. Incoming events are first parsed and checked on some major formatting errors. When the formatting is deemed acceptable the events get forwarded to MonPoly on a per time point basis. If MonPoly reports an issue with a certain time stamp, either it is out of order or one event at that timestamp does not comply with the given signature, this time stamp gets ignored by MonPoly, and in turn the wrapper discards it as well. If no issue is detected with a timestamp all events in at that timestamp get forwarded to the database. Figure 5.2 illustrates the control flow of this process.

In our implementation, one "monitor" object monitors one policy at any given time. This policy can be changed, but never are there more than two policies being monitored. An obvious extension, with many perceivable applications, would be to extend the wrapper to monitor multiple policies at the same time. Multiple policies might be based on the same data, i.e. signature and it would thus be highly practical to monitor them in the same wrapper. This would remove large redundancies if the identical data doesn't need to be stored multiple times in multiple databases.

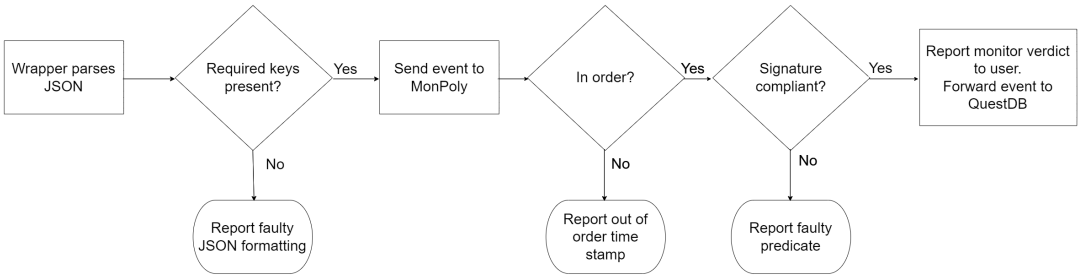

Figure 5.1: Control flow for an incoming

5.2 Policy Change

The naive approach to a policy change is simply reloading all events that have ever been seen by the monitor and in turn are in the database. But we can potentially do much better than this. An initial improvement is to only load all events in the relative interval of the new policy around the most recent time point. We apply our idea of extended relative intervals to optimize the performance of this first version a policy change. This version works by stopping the current MonPoly process and starting a new one.

5.3 Additions to MonPoly

We added two flags to MonPoly that, for a signature file, produce the SQL schema, as previously described, or a representation of the signature in JSON respectively. Those flags are `-sql` and `-sig_to_json`. Their usage is shown in the following example.

```
$ monpoly -sql <path/to/signature>
$ monpoly -sig_to_json <path/to/signature>
```

The flag `-sql_drop` works analogous to the `-sql` flag and produces the SQL statements for "dropping", i.e. deleting the tables in the schema.

```
$ monpoly -sql_drop <path/to/signature>
```

The wrapper needs some kind of feedback when a time point has been processed, in order to return potential feedback to the user. We added the flag `-ack_sep` that can be added when starting to monitor a formula with MonPoly. A separator is a semicolon. With this flag, every time the parser reaches a semicolon, an acknowledgment gets sent to stdout. The wrapper can thus be sure that a time point has been fully processed, and no more waiting is required.

As the wrapper does not perform any check on the names, types, and arity of predicates, we want to avoid that MonPoly exits if the wrapper sends a predicate that is "faulty" in any way. For this purpose we have added a `-tolerate_faulty_predicates` flag. This flag makes it so that any time points that contain a predicate with wrong arity and/or types of attributes, get skipped.

Next we made some additions to make MonPoly work better for our policy change method. By default, MonPoly either runs continuously and reads input from standard input or it processes a log file and terminates once it fully processed all events in the file. We retrieve a potentially large amount of time points from the database during a policy change. It is useful if these events could be sent to MonPoly as a file. But as we want to continue monitoring after processing these past time points, MonPoly cannot terminate at this point. For this purpose we added the flag `-switch_to_stdin_after_log`. With this flag MonPoly processes a log file and once done with that it awaits new input from standard in. We are not necessarily interested in all the monitor verdicts of the new policy on all these past time points. Therefore we added a flag (`-suppress_stdout`) that suppresses any potential output while processing the time points in the log file. The output resumes normally afterwards.

The next three flags that we added concern the relative intervals and the extended relative intervals of formulas. They are `-get_relative_interval`, `-relative_interval_per_predicate`,

and `-relative_interval_per_predicate_json`. The wrapper uses these commands to know which time points and events must be queried from the QuestDB upon a policy change.

5.4 Performance Analysis

For the performance evaluation of the wrapper there are essentially two angles to take.

RQ1 The first one is, how large is the overhead for monitoring a policy when using the wrapper compared to using MonPoly directly?

RQ2 The second point of interest is, how much can our optimizations, using extended relative intervals, improve a policy change compared to the naive approach?

5.4.1 Wrapper vs. MonPoly

We expect the wrapper to add a certain time overhead to the monitoring as it performs the same action as MonPoly plus some more things. We are interested in seeing how much more time these other things add to the total time. Of course, we aim for it to be as small as possible.

To get an idea of this overhead, we measure the time that the wrapper and MonPoly alone take to process equivalent traces with the same policy and signature for both. We ran 20 experiments with the same policy. For every experiment we generated increasingly long traces. In Figure 5.4.1 we present the average total time that the wrapper and MonPoly respectively took to process the events. We can see that the wrapper takes increasingly more time, but the growth is linear. This does not come unexpected. A constant minimal cost will always come from running the trace through a server. As an added measurement we sent every time point separately to MonPoly and measured the time for this. The idea behind this is that in our implementation this is how we forward time points to MonPoly, and we wanted to get an idea how much of the overhead this might explain. Figure 5.4.1 shows the same information, but averaged by the number of time points/ length of the trace. One interesting observation from this graph is the different behavior of MonPoly when processing an entire log file and when receiving individual time points continuously from standard input. For very few time points it appears that the wrapper is slightly faster than MonPoly per time point. This probably comes down to sample size.

5.4.2 Policy Change Performance

In this section we give an idea of the performance of the naive policy change where all events are in the database are processed compared to the performance with our optimization that uses only the events in the extended relative intervals (ERI).

We have compared a policy change using our optimizations to the naive approach by using two random policies and generating random traces of various lengths. We then compare how the two methods perform for an increasing number of time points. We have run the same randomized experiment 20 times. The setup is that we monitor a random policy and measure the time it takes to change the policy after traces with 4^i, for $i \in [1, 9]$, time points have been stored in the database previously. Figure 5.4.2 shows the average time it takes to change the policy for both versions of a policy change in respect to the trace length. Figure 5.4.2 shows the same information, but the time has been averaged by time point. We see that there is large variation for small traces and for a bit the naive change is actually faster, but for large traces our optimization consistently beats the baseline. From the averaged graph we can't make out much of an improvement. We do see that both get faster per time point, when there are more total time points. This can potentially be attributed to amortization costs that have to be paid for a small number as well as for a large number of time points. Of course the expected performance improvement depends heavily on the formula, if it has large extended relative intervals it is barely different from the naive case, and also on the size of the trace stored in the database.

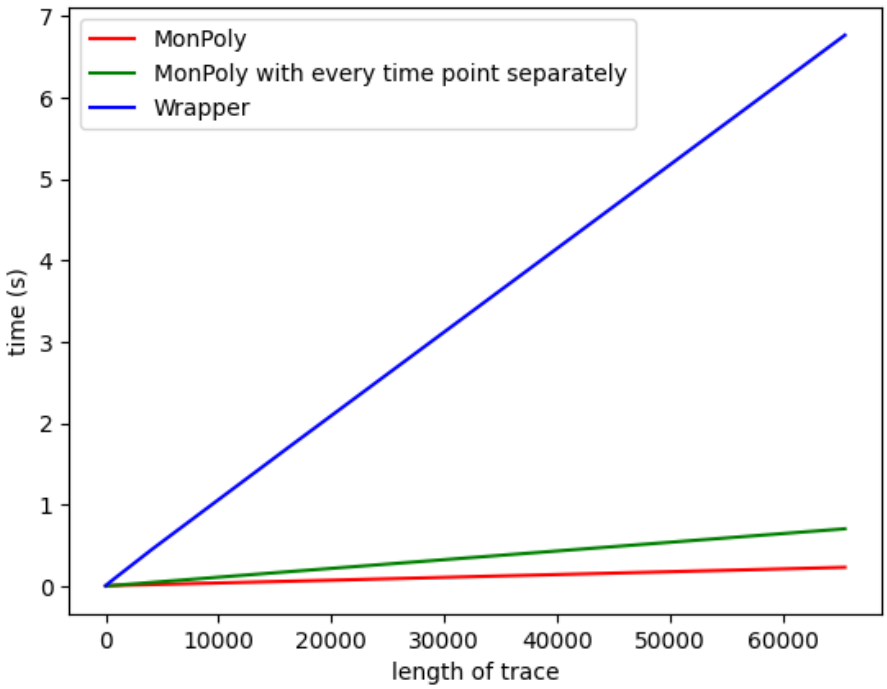

Figure 5.2: Time it takes to process variably long traces

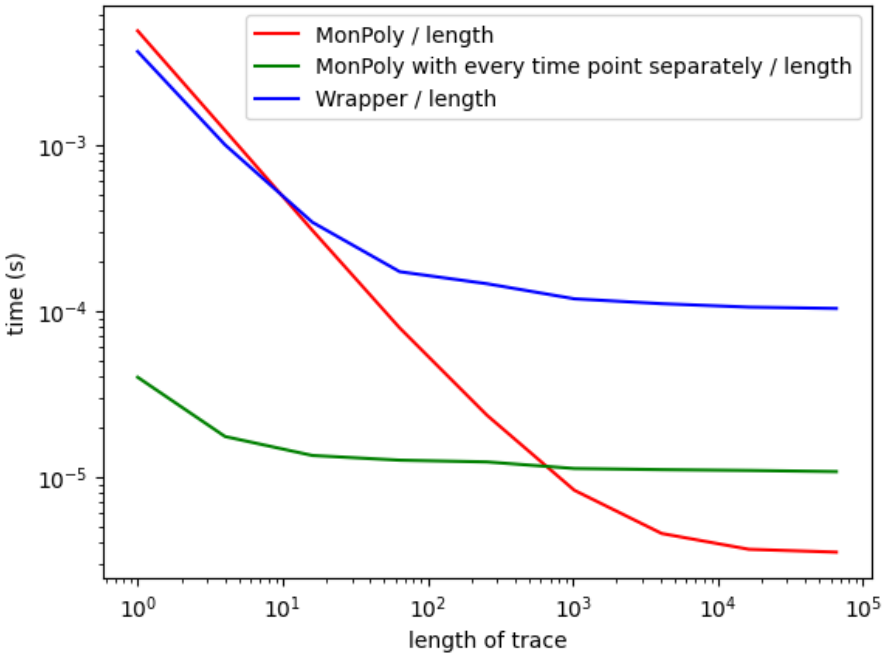

Figure 5.3: Time it takes to process a trace by the number of time points in the trace

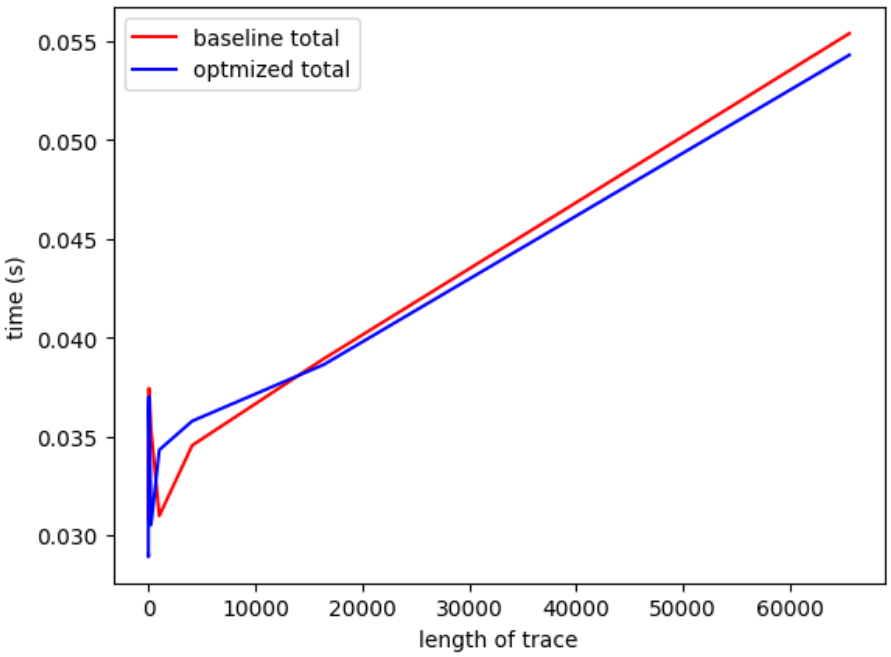

Figure 5.4: Total time a policy change takes for various numbers of time points in a trace

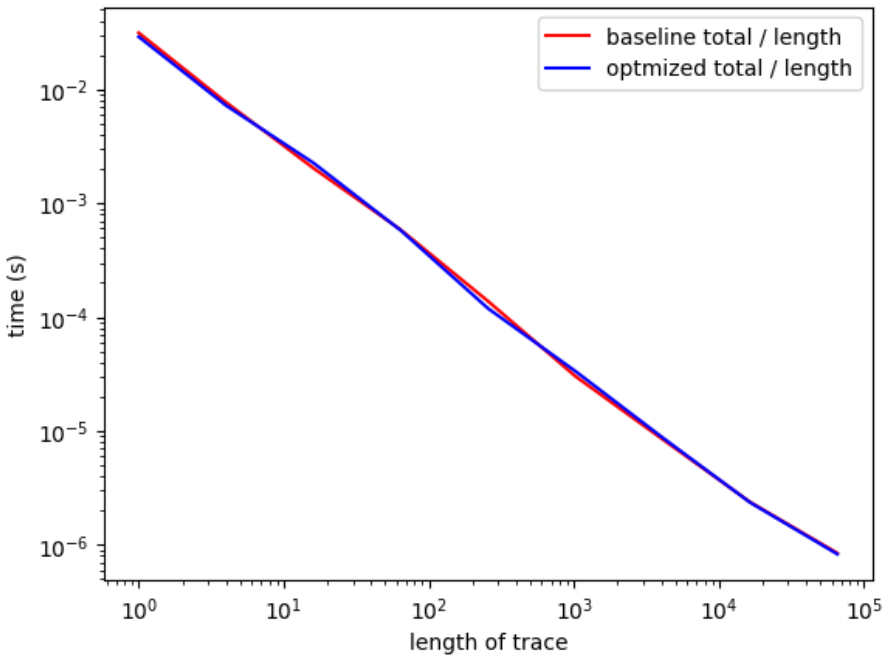

Figure 5.5: Time it takes to change the policy in respect to the total number of events in the trace

5.5 Partial Policy Change

We have started looking into a different method for a policy change that could be done directly through MonPoly and would not rely on the wrapper and the presence of a database. The general idea is that in many use cases it is common that a policy is made up of many individual parts and one might want to change only a small part of a formula. Consider for example a privacy policy on a website. Let's say a user has the ability to specify where their data may and may not be used. Maybe they had selected very restrictive options, but now want to allow their data to be used in one specific case. MonPoly has now continuously been monitoring the restrictive policy and also has the state for all the sub-formulas in the larger policy. In terms of efficiency it would be useful, if the state would only need to be recomputed for the one part that got changed and if the rest of the state could be kept.

One case might be where policies are all in conjunctive normal form (CNF) and single conjuncts might get added or removed. To remove a conjunct from the formula it needs some sort of identifier. In a regular structure like a CNF formula this could be done by automatic indexing. But it still poses some questions, like what happens if the monitor rewrites the formula to an equivalent one, but changes the order. Then the user doesn't have an inherent idea of a conjuncts' location anymore. Our solution to this problem was thus to let the user provide identifiers in the form of strings for certain sub-formulas.

We have begun implementing this change in MonPoly. Our idea makes use of the existing commands feature that lets a user control certain aspects of MonPoly during its runtime. We envision commands of the form "remove sub-formula x", "add formula y as a conjunct to sub-formula x". The second example would replace place the sub formula "x and y" in place of "x". An analogous thing for "or" with an "add disjunct" command is also perceivable. Or also a command like "negate sub-formula x".

With all this, MonPoly would parse the new formula part, add it to the existing formula, check if the new formula is monitorable, and if so it updates the policy. Similarly if a part gets removed a check on monitorability of the remaining formula needs to be done. Before adding a sub-formula, the existing trace has to be checked against that sub-formula and this state must get combined with the pre-existing state of MonPoly.

We have started implementing a feature that allows for named formulas inside MonPoly. The names should have no effect on the monitoring and solely act as the identifiers of formula parts. For this we extended `formula` in MonPoly by an additional type, `NamedFormula of (string * formula)`. And similarly we added the type `ENamedExtf of string * extformula * int` to `extformula`.

To specify a named (sub)formula in a MonPoly policy, we have extended the formula parser and lexer to accept formulas with the following syntax,

```
NAME[f1, name1] AND NAME[f2 OR f3, name2].
```

Chapter 6

Conclusion

To recap, we have set out to build a wrapper around MonPoly that handles persistent data storage, offers a web friendly interface to MonPoly, and provides a policy change method. MonPoly is a runtime monitoring tool that uses MFOTL (and in its newest version also MFODL) as a policy specification language. MFOTL and MFODL are powerful, as in they can express a large amount of properties.

The amount of computer systems in the world is only growing and with it the need for security guarantees is also increasing. While MonPoly does not directly offer such guarantees, it is a useful tool in detecting policy violations.

Considering that a runtime monitor and a database storing system activities often rely on the same data, the idea to combine storing events and monitoring them is a natural one. Some regulation, like GDPR, also requires logging of certain information. This combination of technologies also offers new possibilities, that were not easily achievable before. One such thing is a policy change. When past data can simply be read from the database, it becomes easy to restore the same state when starting to monitor a new policy.

There are many points of future interest around the wrapper specifically and more generally about policy change. As far as the wrapper is concerned, we have already briefly mentioned that one natural expansion would be to monitor multiple policies at the same time. The wrapper would be well suited to abstract away the intricacies and provide a simple interface to the user.

An obvious area for future work is our partial policy change, the core idea of which is outlined at the end of chapter 3. Such an approach may be sufficient to achieve ideal performance. Data protection, where a user can add or remove certain permissions, can be one such use case.

Bibliography

[1] Ezio Bartocci et al. *Lectures on Runtime Verification*. Ed. by Ezio Bartocci and Yliès Falcone. Vol. 10457. Springer International Publishing, 2018. ISBN: 978-3-319-75631-8. DOI: 10.1007/978-3-319-75632-5. URL: http://link.springer.com/10.1007/978-3-319-75632-5.

[2] David Basin et al. "A Formally Verified, Optimized Monitor for Metric First-Order Dynamic Logic". In: *Automated Reasoning: 10th International Joint Conference, IJCAR 2020, Paris, France, July 1–4, 2020, Proceedings, Part I 10*. Springer. 2020, pp. 432–453.

[3] David Basin et al. "Monitoring Metric First-Order Temporal Properties". In: *Journal of the ACM (JACM)* 62 (2 2015), pp. 1–45. ISSN: 0004-5411.

[4] David Basin et al. "Monitoring of Temporal First-Order Properties with Aggregations". In: *Formal methods in system design* 46 (2015), pp. 262–285.

[5] David Basin et al. "Runtime Monitoring of Metric First-Order Temporal Properties". In: Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2008.

[6] David Basin et al. "Scalable Offline Monitoring of Temporal Specifications". In: *Formal Methods in System Design* 49 (1 2016), pp. 75–108. ISSN: 1572-8102.

[7] David A Basin, Felix Klaedtke, and Eugen Zalinescu. "The MonPoly Monitoring Tool." In: *RV-CuBES* 3 (2017), pp. 19–28.

[8] *BitBucket MonPoly Branch*. Accessed: 2023-01-04. URL: https://bitbucket.org/jshs/monpoly/commits/branch/BA_logging_backend.

[9] Jan Chomicki. "Efficient Checking of Temporal Integrity Constraints Using Bounded History Encoding". In: *ACM Transactions on Database Systems (TODS)* 20 (2 1995), pp. 149–186. ISSN: 0362-5915.

[10] *curl*. Accessed: 2023-02-12. URL: https://curl.se/.

[11] Roy Thomas Fielding. *Architectural Styles and the Design of Network-Based Software Architectures*. University of California, Irvine, 2000. ISBN: 0599871180.

[12] *Flask*. Accessed: 2023-01-30. URL: https://flask.palletsprojects.com/.

[13] *GitLab MonPoly Wrapper*. Accessed: 2023-01-04. URL: https://gitlab.inf.ethz.ch/fhublet/monpoly-server.

[14] *InfluxDB Line Protocol*. Accessed: 2023-02-07. URL: https://docs.influxdata.com/influxdb/v1.8/write_protocols/line_protocol_tutorial/.

[15] *PostgreSQL*. Accessed: 2023-02-07. URL: https://www.postgresql.org/.

[16] *QuestDB*. Accessed: 2023-01-24. URL: https://questdb.io/.

[17] *QuestDB - InfluxDB Line Protocol*. Accessed: 2023-01-24. URL: https://questdb.io/docs/reference/api/ilp/overview/.

[18] *QuestDB - Postgres Wire Protocol*. Accessed: 2023-01-24. URL: https://questdb.io/docs/reference/api/postgres/.

[19] *QuestDB - REST API*. Accessed: 2023-02-07. URL: https://questdb.io/docs/reference/api/rest/.

[20] *QuestDB - Storage Model*. Accessed: 2023-01-24. URL: https://questdb.io/docs/concept/storage-model/.

[21] *QuestDB - Time Stamps*. Accessed: 2023-02-13. URL: https://questdb.io/docs/reference/function/date-time/#date-and-timestamp-format.

[22] Joshua Schneider et al. "A Formally Verified Monitor for Metric First-Order Temporal Logic". In: Springer, 2019, pp. 310–328. ISBN: 3030320782.

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Eigenständigkeitserklärung

Die unterzeichnete Eigenständigkeitserklärung ist Bestandteil jeder während des Studiums verfassten Semester-, Bachelor- und Master-Arbeit oder anderen Abschlussarbeit (auch der jeweils elektronischen Version).

Die Dozentinnen und Dozenten können auch für andere bei ihnen verfasste schriftliche Arbeiten eine Eigenständigkeitserklärung verlangen.

Ich bestätige, die vorliegende Arbeit selbständig und in eigenen Worten verfasst zu haben. Davon ausgenommen sind sprachliche und inhaltliche Korrekturvorschläge durch die Betreuer und Betreuerinnen der Arbeit.

Titel der Arbeit (in Druckschrift):

A Real-Time, Flexible Logging and Monitoring Infrastructure for MonPoly

Verfasst von (in Druckschrift):
Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich.

Name(n):	**Vorname(n):**
Degelo	Jonas

Ich bestätige mit meiner Unterschrift:
- Ich habe keine im Merkblatt „Zitier-Knigge" beschriebene Form des Plagiats begangen.
- Ich habe alle Methoden, Daten und Arbeitsabläufe wahrheitsgetreu dokumentiert.
- Ich habe keine Daten manipuliert.
- Ich habe alle Personen erwähnt, welche die Arbeit wesentlich unterstützt haben.

Ich nehme zur Kenntnis, dass die Arbeit mit elektronischen Hilfsmitteln auf Plagiate überprüft werden kann.

Ort, Datum	**Unterschrift(en)**
Zürich, 13.02.2023	

Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich. Durch die Unterschriften bürgen sie gemeinsam für den gesamten Inhalt dieser schriftlichen Arbeit.