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 [13].

1.2 Our Approach

We extend MonPoly with a web based wrapper written in Python using Flask [11]. This wrapper provides a REST API [10] 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 [12]. 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, 17] 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 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.

The definition 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"), $\lozenge_I$ ("eventually"), $\square_I$ ("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 $i \geq 0$, $c^{\mathcal{D}_i} = c^{\mathcal{D}_{i+1}}$.

The indices in the sequences $\bar{\mathcal{D}}$ and $\bar{\tau}$ are called *time points*. It is important to high light the difference between *time stamps* and *time points*. Both are integers, but in our sequences $\bar{\mathcal{D}}$ and $\bar{\tau}$ time points are by their definition as indices strictly increasing and their is no "missing" time point between two others. On the other hand it is possible that succeeding time points have identical time stamps or that time stamps jump by more than one from time point to time point. 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 *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{array}{ll}
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models t \approx t' & \textit{iff } v(t) = v(t') \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \prec t \approx t' & \textit{iff } v(t) < v(t') \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \preceq t \approx t' & \textit{iff } v(t) \leq v(t') \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models r(t_1, \dots, t_{\iota(r)}) & \textit{iff } (v(t_1), \dots, v(t_{\iota(r)})) \in r^{\mathcal{D}_i} \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models (\neg\psi) & \textit{iff } (\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \psi \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models (\psi \vee \psi') & \textit{if } (\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \psi \textit{ or } (\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \psi' \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models (\exists x.\psi) & \textit{iff } (\bar{\mathcal{D}}, \bar{\tau}, v[x \mapsto d], i) \models \psi \textit{ for some } d \in |\bar{\mathcal{D}}| \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models (\bullet_I \psi) & \textit{iff } i > 0, \tau_i - \tau_{i+1} \in I, \textit{ and } (\bar{\mathcal{D}}, \bar{\tau}, v, i-1) \models \psi \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models (\bigcirc_I \psi) & \textit{iff } \tau_{i+1} - \tau_i \in I \textit{ and } (\bar{\mathcal{D}}, \bar{\tau}, v, i+1) \models \psi \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models (\psi \,\mathcal{S}_I\, \psi') & \textit{iff for some } j \leq i, \tau_i - \tau_j \in I, (\bar{\mathcal{D}}, \bar{\tau}, v, j) \models \psi' \textit{ and} \\
 & \quad (\bar{\mathcal{D}}, \bar{\tau}, v, k) \models \psi \textit{ for all } k \in \mathbb{N} \textit{ with } j < k \leq i \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models (\psi \,\mathcal{U}_I\, \psi') & \textit{iff for some } j \geq i, \tau_j - \tau_i \in I, (\bar{\mathcal{D}}, \bar{\tau}, v, j) \models \psi' \textit{ and} \\
 & \quad (\bar{\mathcal{D}}, \bar{\tau}, v, k) \models \psi \textit{ for all } k \in \mathbb{N} \textit{ with } i \leq k < j
\end{array}
$$

Anologous to the definition of MFOTL formulas 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 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 the definition of MFOTL formulas to include MFODL formulas.

Definition 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 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 $i, j \in \mathbb{N}$.*

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

With the help of this definition 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 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.*

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

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)
```

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.

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

For MonPoly we first get rid of the surrounding $\Box$, 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)))
)
```

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 [13].

QuestDB uses a column-based storage model [16]. It supports the PostgreSQL wire protocol [15] for querying and inserting data. It further provides a REST API and has a web console for both inserting and querying data. For best performance it supports the InfluxDB Line Protocol [14] with client libraries for most popular modern programming languages. QuestDB itself is written in Java, open source, and licensed under the Apache 2.0 license.

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.

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. To interact with the wrapper a user can use the provided REST API by sending web requests.

The wrapper runs MonPoly as a subprocess and handles all interactions with MonPoly itself. 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 stamp 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.

3.2 The REST API

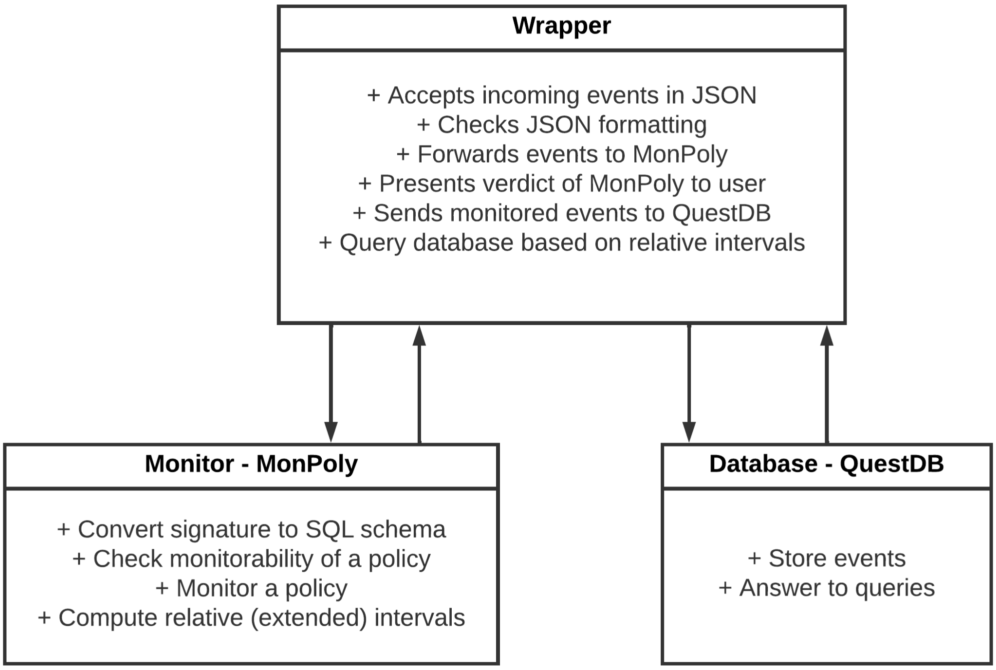

Figure 3.1: Illustration of the project structure

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

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

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

Definition 8 *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

Basin et al. [6] provides an intuition for why the definition of RI makes sense. We aim provide a more formal proof of correctness.

Definition 9 *We call filtering of a trace* $\sigma = ((\tau_i, D_i))_i$ *according to an interval* I *at time point* j *the trace*

$$\mathrm{filter}(\sigma, I, j) = \sigma' = ((\tau_k, D_k), ..., (\tau_j, D_j))$$

where k *is the largest index for which* $\tau_j - \tau_k \in I \cap (-\infty, 0]$.

The intuition for why we intersect the interval I with all negative numbers, is that this limits the range of possible time points to ones that are in the past from the perspective of τ_j. This is sensible, because the set of future events should be empty anyways and we can thus disregard it directly. *comment: this restriction isn't strictly necessary, it might just over complicate things.*

Theorem 1 *For all* σ, v, i*:*

$$(\sigma, v, i) \models \phi \Leftrightarrow (\mathrm{filter}(\sigma, \mathrm{RI}(\phi), i), v, i) \models \phi$$

We prove this theorem by induction over the formula structure of MFOTL/MFODL as defined in the background chapter. Our induction hypothesis is that the theorem holds for any subformulas. We then show that the theorem holds for any formula ϕ. Only formulas of the structures $(\CIRCLE_I \psi), (\Circle_I \psi), (\psi \, \mathcal{S}_I \, \psi'), (\psi \, \mathcal{U}_I \, \psi'), (\blacktriangleleft_I \, \rho)$, and $(\triangleright_I \, \rho)$ depend on the trace σ. Therefore the theorem holds trivially if ϕ has any other structure (first order logic and relations).

We start with $\phi = (\CIRCLE_{[a,b)} \psi)$. There are two cases to consider. Either the left hand side (LHS) is true, i.e. $(\sigma, v, i \models \phi)$ or it is not true, i.e. $(\sigma, v, i \not\models \phi)$. We first consider the LHS to be true, that is ϕ evaluates to true at the time point i with the valuation mapping v and the trace σ. The relative interval of $\phi = (\CIRCLE_{[a,b)} \psi)$ is defined as $\mathrm{RI}(\phi) = (-b, 0] \Cup ((-b, -a] \oplus \mathrm{RI}(\psi))$. By our induction hypothesis $(\sigma, v, i) \models \psi \Leftrightarrow (\mathrm{filter}(\sigma, \mathrm{RI}(\psi), i), v, i) \models \psi$ holds.

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 example from earlier, `"advertising"` is one such constant attribute in the predicate `loc_accessed`.

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

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 call one such arrangement a mask. Each mask has its own relative interval. For our example the masks with their corresponding relative intervals are the following.

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

A `*` in the attributes denotes a variable value. In larger formulas there can be multiple different masks per predicate.

We use a doubly nested map data structure to store the predicates with there masks and relative intervals and call such a structure the extended relative intervals of a formula. On the first level the keys are predicate names and values are maps from masks to intervals. On this data structure we define the operators $\ddot{\Cup}$, $\dot{\Cup}$ and $\dot{\oplus}$.

Definition 10 *Let m and n be two extended relative intervals and i a positive interval, then*

$$\begin{aligned}
m \,\ddot{\Cup}\, n =& \{p(l) \to (i \Cup j) \mid p(l) \to i \in m \text{ and } p(l) \to j \in n\}\\
& \cup \{p(l) \to i \mid (p(l) \to i \in m \text{ and } p(l) \in \mathrm{keys}(m) \setminus \mathrm{keys}(n))\}\\
& \cup \{p(l) \to i \mid (p(l) \to i \in n \text{ and } p(l) \in \mathrm{keys}(n) \setminus \mathrm{keys}(m))\}\\
i \,\dot{\Cup}\, m =& \{p(l) \to (i \Cup j) \mid p(l) \to j \in m\}\\
i \,\dot{\oplus}\, m =& \{p(l) \to (i \oplus j) \mid p(l) \to j \in m\}
\end{aligned}$$

The notation $p(l) \to i$ denotes an element in our doubly nested map structure. p is a first level key, i.e. a predicate name, l is a second level key, i.e. a mask and i denotes the interval the key combination $p(l)$ is pointing to. The keys operator gives all combinations of outer keys (predicate names) and inner keys (masks) in an extended relative intervals structure. With the help of the operators $\ddot{\Cup}$, $\dot{\Cup}$ and $\dot{\oplus}$ we now give a recursive definition for our extended relative intervals.

Definition 11 *The extended relative interval of the formula φ, $\mathrm{ERI}(\varphi)$ is defined recursively over the formula structure:*

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

And for regular expressions we define

Definition 12 *The extended relative interval of the regular expression ρ,* $\mathrm{ERI}_{reg}(\rho)$ *is defined recursively over the structure of the regular expression:*

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

Definition 13 *We call fine-grained filtering of a trace $\sigma = ((\tau_i, D_i))_i$ according to the <name of your datastructure> X the trace*

$$\mathrm{filter}'(\sigma, X, j) = ...$$

Theorem 2 *For all σ, v, i:*

$$\sigma, v, i \models \varphi \Leftrightarrow \mathrm{filter}'(\sigma, \mathrm{ERI}(\phi), i), v, i \models \varphi$$

4.3.1 Correctness

Analogous to the definition of RI we now want to proof that φ evaluated on the sub trace σ' extracted with $\mathrm{ERI}(\varphi)$ leads to the same truth value as with the full trace σ.

To get a correct sub trace σ' we cannot simply extract the sub trace per predicate and mask. There are possibly time points and time stamps in the original trace that have no predicate attached to them. Therefore we additionally need to extract all such time stamps that fall into the regular relative interval of the formula. Empty time points or time stamps can influence the truth value of a formula and can therefore not be omitted.

The first case of atomic formulas that are not a predicate is trivial. They do not depend on any events that may or may not be present in a log and can be evaluated as is. Next a simple predicate only depends on the current time point with time stamp 0. For the unary and binary first order logic formulas (negation, quantification, and, or) the extended relative interval is the special union of relative intervals of the sub formula(s). Special union referring to the operator that keeps the intervals as intervals.

4.4 Conversion to SQL-Query

Chapter 5

Implementation and Evaluation

Chapter 6

Conclusion

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

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

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

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

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

[15] *QuestDB - Postgres Wire Protocol.* Accessed: 2023-01-24. URL: https://questdb.io/docs/develop/insert-data/#postgresql-wire-protocol.

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

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