Scalable Offline Monitoring*

David Basin[1], Germano Caronni[2], Sarah Ereth[3], Matúš Harvan[4], Felix Klaedtke[5], and Heiko Mantel[3]

[1] Institute of Information Security, ETH Zurich, Switzerland
[2] Google Inc., Switzerland
[3] Department of Computer Science, TU Darmstadt, Germany
[4] ABB Corporate Research, Switzerland
[5] NEC Europe Ltd., Heidelberg, Germany

Abstract. We propose an approach to monitoring IT systems offline, where system actions are logged in a distributed file system and subsequently checked for compliance against policies formulated in an expressive temporal logic. The novelty of our approach is that monitoring is parallelized so that it scales to large logs. Our technical contributions comprise a formal framework for slicing logs, an algorithmic realization based on MapReduce, and a high-performance implementation. We evaluate our approach analytically and experimentally, proving the soundness and completeness of our slicing techniques and demonstrating its practical feasibility and efficiency on real-world logs with 400 GB of relevant data.

1 Introduction

Data owners, such as individuals and companies, are increasingly concerned that their private data, collected and shared by IT systems, is used only for the purposes for which it was collected. Conversely, those parties responsible for collecting and managing such data must increasingly follow regulations on how it is processed. For example, US hospitals must follow the US Health Insurance Portability and Accountability Act (HIPAA) and financial services must conform to the Sarbanes-Oxley Act (SOX), and these laws even stipulate the use of mechanisms in IT system for monitoring system behavior. Although various monitoring approaches have been developed for different expressive policy specification languages, such as [9, 10, 13, 15, 18], they do not scale to checking compliance of large-scale IT systems like cloud-based services and systems that process machine-generated data. These systems typically log terabytes or even petabytes of system actions each day. Existing monitoring approaches fail to cope with such enormous quantities of logged data.

In this paper, we propose a scalable approach to offline monitoring, where system components log their actions and monitors inspect the logs to identify

* This work was partly done while Matúš Harvan was at ETH Zurich and Google Inc. and Felix Klaedtke was at ETH Zurich. The Center for Advanced Security Research Darmstadt (www.cased.de), the Zurich Information Security and Privacy Center (www.zisc.ethz.ch), and Google Inc. supported this work.

policy violations. Given a policy, our solution works by decomposing the logs into small parts, called slices, that can be independently analyzed. We can therefore parallelize and distribute the monitoring process over multiple computers.

One of the main challenges is to generate the slices without weakening the guarantees provided by monitoring. In particular, the slices must be *sound* and *complete* for the given policy and logged data. That means that only actual violations are reported and every violation is reported by at least one monitor. Furthermore, slicing should be effective, i.e., producing the slices should be fast and the slices should be small. We provide a framework for obtaining slices with these properties. In particular, our framework lays the foundations for slicing logs, where logs are represented as temporal structures and policies are given as formulas in metric first-order temporal logic (MFOTL) [8,9]. Although we use temporal structures for representing logs and MFOTL as a policy specification language, the underlying principles of our slicing framework are general and apply to other representations of logs and other logic-based policy languages.

Within our theoretical slicing framework, we define orthogonal methods to generate sound and complete slices. The first method constructs slices for checking system compliance for specific entities, such as all users whose login name starts with the letter "A." Note that it is not sufficient to consider just the actions of these users to check their compliance; other users' actions might also be relevant and must also be included in a slice to be sound. The second method checks system compliance during a specific time period, such as a particular week. In addition to these two basic methods for slicing with respect to data and time, we describe slicing by filtering, which discards parts of a slice to speed up monitoring. Finally, we show that slicing is compositional. We can therefore obtain new, more powerful slicing methods by composing existing methods.

We demonstrate how to employ the MapReduce framework [12] to parallelize and distribute the slicing and monitoring tasks. We propose algorithms, for both slicing and filtering. Moreover, we explain how to flexibly combine slicing and filtering. As required by MapReduce, we define map and reduce functions that constitute the backbone of the algorithmic realization of our slicing framework. The map function realizes slicing and the reduce function realizes monitoring. MapReduce runs in its map phase and in its reduce phase multiple instances of the respective function in parallel, where each instance is responsible for a part of the logged data. Splitting and parallelizing the workload this way enables monitoring to scale in the high-performance implementation of our approach.

We deploy and evaluate our monitoring solution in a real-world setting, where we check the compliance of more than 35,000 computers, producing approximately 1 TB of log data each day. The policies considered concern the updating of system configurations and access to sensitive resources. We successfully monitor the relevant actions logged by these computers. The log consist of several billion log entries from a two year period, requiring 0.4 TB of storage. The monitoring takes just a few hours, using only 1,000 machines in a MapReduce cluster.

Overall, we see our contributions as follows. First, we provide a framework for splitting logs into slices for monitoring. Second, we give a scalable algorithmic realization of our framework for monitoring large logs in an offline setting. Both our framework and our algorithmic realization support compositional slicing. Finally, with our case study, we show that the approach is effective and scales well. In particular, our deployment and the evaluation demonstrate the feasibility of checking compliance in large-scale IT systems.

We proceed as follows. In Section 2, we give background on MFOTL and monitoring. In Section 3, we describe our approach to slicing and monitoring, including its algorithmic realization in MapReduce. In Section 4, we experimentally evaluate our approach. We discuss related work in Section 5 before drawing conclusions in Section 6. Additional details, including proofs and pseudo code omitted in the main text, are given in the appendices.

2 Preliminaries

In this section, we explain how we use metric first-order temporal logic (MFOTL) to represent system requirements, and how we monitor a single stream of logged system actions. We start by introducing MFOTL.

Specification Language. We give just a brief overview of MFOTL; further details can be found in Appendix A. MFOTL is similar to propositional real-time logics like MTL [2]. However, as it is a first-order logic, MFOTL's syntax is defined with respect to a signature. Furthermore, instead of timed words, its models are temporal structures $(\bar{\mathcal{D}}, \bar{\tau})$, where $\bar{\mathcal{D}} = (\mathcal{D}_0, \mathcal{D}_1, \dots)$ is a sequence of structures and $\bar{\tau} = (\tau_0, \tau_1, \dots)$ is a sequence of natural numbers. As is usual, a structure $\mathcal{D}$ over a signature $\mathcal{S}$ (without function symbols) consists of a domain $|\mathcal{D}| \neq \emptyset$ and interpretations $c^{\mathcal{D}} \in |\mathcal{D}|$ and $r^{\mathcal{D}} \subseteq |\mathcal{D}|^{\iota(r)}$, for each constant symbol c and predicate symbol r of the signature $\mathcal{S}$, where $\iota(r)$ denotes r's arity.

The formulas over the signature $\mathcal{S}$ are given by the grammar

$$\varphi ::= t_1 \approx t_2 \,\big|\, t_1 \prec t_2 \,\big|\, r(t_1, \dots, t_{\iota(r)}) \,\big|\, \neg\varphi \,\big|\, \varphi \vee \varphi \,\big|\, \exists x.\, \varphi \,\big|\, \CIRCLE_I\, \varphi \,\big|\, \Circle_I\, \varphi \,\big|\, \varphi \,\mathsf{S}_I\, \varphi \,\big|\, \varphi \,\mathsf{U}_I\, \varphi\,,$$

where $t_1, t_2, \dots$ are variables or constant symbols of $\mathcal{S}$, r a predicate symbol of S, x a variable, and I an interval $[a, b) \subseteq \mathbb{N}$. The temporal operators $\CIRCLE_I$ ("previous"), $\Circle_I$ ("next"), S_I ("since"), and U_I ("until") require the satisfaction of a formula within a particular time interval in the past or future. An operator's subscript I specifies this time interval. MFOTL's satisfaction relation $\models$ is defined as expected for (i) a time point $i \in \mathbb{N}$, (ii) a valuation v interpreting the variables, and (iii) a temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$. We call the indices of the τ_is and $\mathcal{D}_i$s *time points* and the τ_is *timestamps*. In particular, τ_i is the timestamp at time point $i \in \mathbb{N}$.

We use standard terminology and syntactic sugar, see e.g., [3,14]. For instance, we use terms like *free variable* and *atomic formula*, and abbreviations such as $\blacklozenge_I\, \varphi := \mathit{true}\, \mathsf{S}_I\, \varphi$ ("once"), $\Diamond_I\, \varphi := \mathit{true}\, \mathsf{U}_I\, \varphi$ ("eventually"), $\blacksquare_I\, \varphi := \neg \blacklozenge_I \neg\varphi$ ("historically"), and $\Box_I\, \varphi := \neg \Diamond_I \neg\varphi$ ("always"), where $\mathit{true} := \exists x.\, x \approx x$. Intuitively, the formula $\blacklozenge_I\, \varphi$ states that φ holds at some time point in the past within the time window I and $\blacksquare_I\, \varphi$ states that φ holds at all time points in the past within the time window I. The corresponding future operators are $\Diamond_I$ and $\Box_I$. We also use non-metric operators like $\Box\,\varphi := \Box_{[0,\infty)}\,\varphi$. To omit parentheses, we use the standard conventions about the binding strength of logical connectives, e.g., Boolean operators bind stronger than temporal ones and unary operators bind stronger than binary ones.

Throughout the paper, we make the following assumptions when not stated otherwise. First, formulas and temporal structures are over the signature $\mathcal{S}$ consisting of the sets C and R of constant and predicate symbols, and the function ι assigns an arity to each predicate symbol. Second, the set of variables is V. Third, the structures' domain is $\mathbb{D}$ and constant symbols are interpreted identically in all structures. The set of all these temporal structures is $\mathbf{T}$. Finally, without loss of generality, variables are quantified at most once in a formula and quantified variables are disjoint from the formula's free variables.

Monitoring. We use MFOTL to check the policy compliance of a stream of system actions as follows [8]. Policies are given as MFOTL formulas of the form $\Box\,\psi$. For illustration, consider the policy stating that SSH connections must last no longer than 24 hours. This can be formalized in MFOTL as

$$\Box\,\forall c.\,\forall s.\,\mathit{ssh\_login}(c,s) \rightarrow \Diamond_{[0,25)}\,\mathit{ssh\_logout}(c,s)\,, \qquad (P0)$$

where we assume that time units are in hours and the signature consists of the two binary predicate symbols *ssh_login* and *ssh_logout*, where the first parameter specifies the computer on which the action is performed and the second the session identifier of the SSH connection. We also assume that the system actions are logged. In particular, the ith entry in the stream of logged actions consists of the performed actions and a timestamp τ_i that records the time when the actions occurred. For checking compliance with respect to the formula ($P0$), we assume that the logged actions are the logins and logouts, with the parameters specifying the computer's name and the session identifier.

The corresponding temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$ for such a stream of logged SSH login and logout actions is as follows. The domain of $\bar{\mathcal{D}}$ contains all possible computer names and session identifiers. The ith structure in $\bar{\mathcal{D}}$ contains the relations $\mathit{ssh\_login}^{\mathcal{D}_i}$ and $\mathit{ssh\_logout}^{\mathcal{D}_i}$, where (1) $(c,s) \in \mathit{ssh\_login}^{\mathcal{D}_i}$ iff there is a logged login action in the ith entry of the stream with the parameter values c and s, and (2) $(c,s) \in \mathit{ssh\_logout}^{\mathcal{D}_i}$ iff there is a logged logout action in the ith entry of the stream with the parameter values c and s. The ith timestamp in $\bar{\tau}$ is simply the timestamp τ_i of the ith log entry. This generalizes straightforwardly to an arbitrary stream of logged actions, where the kind of actions correspond to the predicate symbols specified by the temporal structure's signature and the actions' parameter values are elements from the temporal structure's domain.

In practice, we can only monitor finite prefixes of temporal structures to detect policy violations. However, to ease our exposition, we require that temporal structures, and thus also logs, describe infinite streams of system actions. We use the monitoring tool MONPOLY [7] to check whether a stream of system actions complies with a policy formalized in MFOTL. It implements the monitoring algorithm in [9]. MONPOLY iteratively processes the temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$ representing a stream of logged actions, either offline or online, and outputs the policy violations. Formally, for a formula $\Box\,\psi$, a *policy violation* is a pair (v, τ) of a valuation v and a timestamp τ such that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \neg\psi$, for some time point i with $\tau_i = \tau$. The formula ψ may contain free variables and the valuation v interprets these variables. As MONPOLY searches for all combinations of time-points and interpretations of the free variables for which a given stream of logged actions violates the policy, in practice we drop the outer universal quantifications in the policy's MFOTL formalization to obtain additional information about the violations. For instance, if we remove the universal quantification over s in the formula (*P0*), then the valuation v in each policy violation (v, τ) specifies a session identifier of an SSH connection that lasted 25 hours or more.

In general, we assume that the subformula ψ of $\Box\,\psi$ formalizing the given policy is *bounded*, i.e., the interval I of every temporal operator U_I occurring in ψ is finite. Since ψ is bounded, the monitor only needs to process a finite prefix of $(\bar{\mathcal{D}}, \bar{\tau}) \in \mathbf{T}$ when determining the valuations satisfying $\neg\psi$ at any given time point. To effectively determine all these valuations, we also assume here that predicate symbols have finite interpretations in $(\bar{\mathcal{D}}, \bar{\tau})$, that is, the relation $r^{\mathcal{D}_j}$ is finite, for every predicate symbol r and every $j \in \mathbb{N}$. Furthermore, we require that $\neg\psi$ can be rewritten to a formula that is temporal safe-range [9], a generalization of the standard notion of safe-range database queries [1]. In our SSH example, the rewritten formula of (*P0*) without the outermost temporal operator and quantifiers is $\mathit{ssh\_login}(c, s) \wedge \neg \Diamond_{[0,25)}\, \mathit{ssh\_logout}(c, s)$. We refer to [9] for a detailed description of the monitoring algorithm.

3 Log Slicing

In Section 3.1, we present the logical foundation of our slicing framework. A slicer splits the temporal structure to be monitored into *slices*. We introduce the notions of soundness and completeness for individual slices relative to sets of possible violations, called *restrictions*. We show that soundness and completeness of each individual slice in a set are sufficient to find all violations of a given policy, provided that the restrictions are chosen appropriately. We also show that slicing is compositional. In Section 3.2, we present concrete instances of slicers and in Section 3.3, we present an algorithmic realization of our slicing framework.

3.1 Slicing Foundations

Slices. Slicing entails splitting a temporal structure, which represents a stream of logged actions, into multiple temporal structures. Each such temporal structure contains only a subset of the logged actions. Formally, a slice is defined as follows.

Definition 1. *Let $s : [0, \ell) \to \mathbb{N}$ be a strictly increasing function, with $\ell \in \mathbb{N} \cup \{\infty\}$. The temporal structure $(\bar{\mathcal{D}}', \bar{\tau}') \in \mathbf{T}$ is a* slice *of $(\bar{\mathcal{D}}, \bar{\tau}) \in \mathbf{T}$ (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 $i \in [0, \ell)$ and all $r \in R$.*

Recall that the logged system actions at a time point $i \in \mathbb{N}$ are represented as the elements in $\mathcal{D}_i$'s relations $r^{\mathcal{D}_i}$, with $r \in R$. The function s determines which time points of the temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$ are in the slice $(\bar{\mathcal{D}}', \bar{\tau}')$. For the time points present in the slice, some actions may be ignored since $r^{\mathcal{D}'_i} \subseteq r^{\mathcal{D}_{s(i)}}$, for $i \in [0, \ell)$. Note that the domain of the function s may be finite or infinite. If its domain is infinite, i.e. when $\ell = \infty$, we require that each action in the slice is an action of the original stream of actions, i.e. $r^{\mathcal{D}'_i} \subseteq r^{\mathcal{D}_{s(i)}}$, for each $i \in \mathbb{N}$. If s's domain is finite, i.e. when $\ell \in \mathbb{N}$, we relax this requirement by not imposing any restrictions on the structures $\mathcal{D}'_i$ and the timestamps τ'_i with $i \geq \ell$. In this case, the suffix of the slice starting at time point ℓ is ignored when monitoring the slice.

To meaningfully monitor slices independently, we require that slices are *sound* and *complete*. Intuitively, this means that at least one of the monitored slices violates the given policy if and only if the original temporal structure violates the policy. We define these requirements in Definition 2 below, relative to a set $\mathcal{R} \subseteq ((V \to \mathbb{D}) \times \mathbb{N})$, called a *restriction*. We use $\mathbf{R}$ to denote the set of all such restrictions and say that a violation (v, t) is *permitted* by $\mathcal{R} \in \mathbf{R}$ if $(v, t) \in \mathcal{R}$.

Definition 2. *Let φ be a formula and $\mathcal{R} \in \mathbf{R}$.*

(i) $(\bar{\mathcal{D}}', \bar{\tau}') \in \mathbf{T}$ *is* $\mathcal{R}$-sound *for* $(\bar{\mathcal{D}}, \bar{\tau}) \in \mathbf{T}$ *and φ if for all pairs (v, t) permitted by $\mathcal{R}$, it holds that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \varphi$, for all $i \in \mathbb{N}$ with $\tau_i = t$, implies $(\bar{\mathcal{D}}', \bar{\tau}', v, j) \models \varphi$, for all $j \in \mathbb{N}$ with $\tau'_j = t$.*

(ii) $(\bar{\mathcal{D}}', \bar{\tau}') \in \mathbf{T}$ *is* $\mathcal{R}$-complete *for* $(\bar{\mathcal{D}}, \bar{\tau}) \in \mathbf{T}$ *and φ if for all pairs (v, t) permitted by $\mathcal{R}$, it holds that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \varphi$, for some $i \in \mathbb{N}$ with $\tau_i = t$, implies $(\bar{\mathcal{D}}', \bar{\tau}', v, j) \not\models \varphi$, for some $j \in \mathbb{N}$ with $\tau'_j = t$.*

We equip each slice with a restriction. The original temporal structure is equipped with the *non-restrictive* restriction $\mathcal{R}_0 := ((V \to \mathbb{D}) \times \mathbb{N})$, which permits any pair (v, t).

Slicers. We call a mechanism that splits a temporal structure into slices a *slicer*. Additionally, a slicer equips the resulting slices with restrictions. In Definition 3, we give requirements that the slices and their restrictions must fulfill. In Theorem 4, we show that these requirements suffice to ensure that monitoring the slices is equivalent to monitoring the original temporal structure.

Definition 3. *A* slicer $\mathfrak{s}_\varphi$ *for the formula φ is a function that maps $(\bar{\mathcal{D}}, \bar{\tau}) \in \mathbf{T}$ and $\mathcal{R} \in \mathbf{R}$ to a family of temporal structures $(\bar{\mathcal{D}}^k, \bar{\tau}^k)_{k \in K}$ and a family of restrictions $(\mathcal{R}^k)_{k \in K}$ that satisfy the following conditions.*

(S1) *$(\mathcal{R}^k)_{k \in K}$ refines $\mathcal{R}$, i.e., $\bigcup_{k \in K} \mathcal{R}^k = \mathcal{R}$.*
(S2) *$(\bar{\mathcal{D}}^k, \bar{\tau}^k)$ is $\mathcal{R}^k$-sound for $(\bar{\mathcal{D}}, \bar{\tau})$ and φ, for all $k \in K$.*
(S3) *$(\bar{\mathcal{D}}^k, \bar{\tau}^k)$ is $\mathcal{R}^k$-complete for $(\bar{\mathcal{D}}, \bar{\tau})$ and φ, for all $k \in K$.*

Theorem 4. *Let $\mathfrak{s}_\varphi$ be a slicer for the formula φ. Assume that $\mathfrak{s}_\varphi$ maps $(\bar{\mathcal{D}}, \bar{\tau}) \in \mathbf{T}$ and $\mathcal{R} \in \mathbf{R}$ to the family of temporal structures $(\bar{\mathcal{D}}^k, \bar{\tau}^k)_{k \in K}$ and the family of restrictions $(\mathcal{R}^k)_{k \in K}$. The following conditions are equivalent.*

(1) $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \varphi$*, for all valuations* v *and* $i \in \mathbb{N}$ *with* $(v, \tau_i) \in \mathcal{R}$*.*
(2) $(\bar{\mathcal{D}}^k, \bar{\tau}^k, v, i) \models \varphi$*, for all* $k \in K$*, valuations* v*, and* $i \in \mathbb{N}$ *with* $(v, \tau_i) \in \mathcal{R}^k$*.*

Composition. We define next an operation for composing slicers. Theorem 6 shows that the composition of slicers is again a slicer. Hence we can restrict ourselves to a few basic slicers, which we provide in Section 3.2 and their algorithmic realization in Section 3.3. By composition, we obtain more powerful slicers, which may be needed to obtain slices of manageable size from very large logs.

Definition 5. *Let* $\mathfrak{s}_\varphi$ *and* $\mathfrak{s}'_\varphi$ *be slicers for the formula* φ*. The* combination $\mathfrak{s}'_\varphi \circ_{\hat{k}} \mathfrak{s}_\varphi$ *for the index* $\hat{k}$ *is the function that maps* $(\bar{\mathcal{D}}, \bar{\tau}) \in \mathbf{T}$ *and* $\mathcal{R} \in \mathbf{R}$ *to the following families of temporal structures and restrictions, assuming that* $\mathfrak{s}_\varphi$ *maps* $(\bar{\mathcal{D}}, \bar{\tau})$ *and* $\mathcal{R}$ *to* $(\bar{\mathcal{D}}^k, \bar{\tau}^k)_{k \in K}$ *and* $(\mathcal{R}^k)_{k \in K}$

- *If* $\hat{k} \notin K$ *then* $\mathfrak{s}'_\varphi \circ_{\hat{k}} \mathfrak{s}_\varphi$ *returns* $(\bar{\mathcal{D}}^k, \bar{\tau}^k)_{k \in K}$ *and* $(\mathcal{R}^k)_{k \in K}$*.*
- *If* $\hat{k} \in K$ *then* $\mathfrak{s}'_\varphi \circ_{\hat{k}} \mathfrak{s}_\varphi$ *returns* $(\bar{\mathcal{D}}^k, \bar{\tau}^k)_{k \in K''}$ *and* $(\mathcal{R}^k)_{k \in K''}$*, where* $K'' := (K \setminus \{\hat{k}\}) \cup K'$ *and* $(\bar{\mathcal{D}}^k, \bar{\tau}^k)_{k \in K'}$ *and* $(\mathcal{R}^k)_{k \in K'}$ *are the families returned by* $\mathfrak{s}'_\varphi$ *for the input* $(\bar{\mathcal{D}}^{\hat{k}}, \bar{\tau}^{\hat{k}})$ *and* $\mathcal{R}^{\hat{k}}$*, assuming* $K \cap K' = \emptyset$*.*

Intuitively, we first apply the slicer $\mathfrak{s}_\varphi$. The index $\hat{k}$ specifies which of the obtained slices should be sliced further. If there is no $\hat{k}$th slice, the second slicer $\mathfrak{s}'_\varphi$ does nothing. Otherwise, we use $\mathfrak{s}'_\varphi$ to make the $\hat{k}$th slice smaller. Note that by combing the slicer $\mathfrak{s}_\varphi$ with different indices, we can slice all of $\mathfrak{s}_\varphi$'s outputs further. Note too that an algorithmic realization of the function $\mathfrak{s}'_\varphi \circ_{\hat{k}} \mathfrak{s}_\varphi$ need not necessarily compute the output of $\mathfrak{s}_\varphi$ before applying $\mathfrak{s}'_\varphi$.

Theorem 6. *The combination* $\mathfrak{s}'_\varphi \circ_{\hat{k}} \mathfrak{s}_\varphi$ *of the slicers* $\mathfrak{s}_\varphi$ *and* $\mathfrak{s}'_\varphi$ *for the formula* φ *is a slicer for the formula* φ*.*

3.2 Basic Slicers

We now introduce three basic slicers. We focus on just one of them here. We provide full details of the other two in the Appendices B.4 and B.5.

Slicing Data. Data slicers split the relations of a temporal structure. We call the resulting slices data slices. Formally, $(\bar{\mathcal{D}}', \bar{\tau}') \in \mathbf{T}$ is a *data slice* of $(\bar{\mathcal{D}}, \bar{\tau}) \in \mathbf{T}$ if $(\bar{\mathcal{D}}', \bar{\tau}')$ is a slice of $(\bar{\mathcal{D}}, \bar{\tau})$, where the function $s : [0, \ell) \to \mathbb{N}$ in Definition 1 is the identity function and $\ell = \infty$. In the following, we introduce data slicers that return sound and complete slices relative to a restriction.

In a nutshell, a data slicer takes as input a formula φ, a *slicing variable* x, which is a free variable in φ, and *slicing sets*, which are sets of possible values for x. It constructs one slice for each slicing set. The slicing sets can be chosen freely, and can overlap, as long as their union covers all possible values for x. Intuitively, each slice excludes those elements of the relations interpreting the predicate symbols that are irrelevant to determining φ's truth value when x takes values from the slicing set. For values outside of the slicing set, the formula may evaluate to a different truth value on the slice than on the original temporal structure.

We begin by defining the slices output by our data slicer.

Definition 7. *Let φ be a formula, $x \in V$, $(\bar{\mathcal{D}}, \bar{\tau}) \in \mathbf{T}$, and $S \subseteq \mathbb{D}$ a slicing set. The (φ, x, S)-*slice *of $(\bar{\mathcal{D}}, \bar{\tau})$ is the data slice $(\bar{\mathcal{D}}', \bar{\tau}')$, where the relations are as follows. For all $r \in R$, $i \in \mathbb{N}$, and $\bar{a} \in \mathbb{D}^{\iota(r)}$, it holds that $\bar{a} \in r^{\mathcal{D}'_i}$ iff $\bar{a} \in r^{\mathcal{D}_i}$ and there is an atomic subformula of φ of the form $r(\bar{t})$ such that for every j with $1 \leq j \leq \iota(r)$, at least one of the following conditions is satisfied.*

(D1) t_j is the variable x and $a_j \in S$.
(D2) t_j is a variable y different from x.
(D3) t_j is a constant symbol c with $c^{\bar{\mathcal{D}}} = a_j$.

Intuitively, the conditions (D1) to (D3) ensure that a slice contains the tuples from the relations interpreting the predicate symbols that are sufficient to evaluate φ when x takes values from the slicing set. For this, it suffices to consider only atomic subformulas of φ with a predicate symbol. Every item of a tuple from the symbol's interpretation must satisfy at least one of the conditions. If the subformula includes the slicing variable, then only values from the slicing set are relevant (D1). If it includes another variable, then all possible values are relevant (D2). Finally, if it includes a constant symbol, then the interpretation of the constant symbol is relevant (D3).

The following example illustrates Definition 7. It also demonstrates that the choice of the slicing variable can influence how lean the slices are and how much overhead the slicing causes in terms of duplicated log data. Ideally, each logged action appears in at most one slice. However, this is not generally the case and a logged action can appear in multiple slices. In the worst case, each slice ends up being the original temporal structure.

Example 8. Let φ be the formula $\mathit{ssh\_login}(c, s) \rightarrow \Diamond_{[0,6)}\, \mathit{notify}(\mathsf{reg\_server}, s)$, where c and s are variables and $\mathsf{reg\_server}$ is a constant symbol, which is interpreted by the domain element $0 \in \mathbb{D}$, with $\mathbb{D} = \mathbb{N}$. The formula φ expresses that a notification of the session identifier of an SSH login must be sent to the registration server within 5 time units. Assume that at time point 0 the relations of $\mathcal{D}_0$ of the original temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$ for the predicate symbols *ssh_login* and *notify* are $\mathit{ssh\_login}^{\mathcal{D}_0} = \{(1,1), (1,2), (3,3), (4,4)\}$ and $\mathit{notify}^{\mathcal{D}_0} = \{(0,1), (0,2), (0,3), (0,4)\}$.

We slice on the variable c. For the slicing set $S = \{1, 2\}$, the (φ, c, S)-slice contains the structure $\mathcal{D}'_0$ with $\mathit{ssh\_login}^{\mathcal{D}'_0} = \{(1,1), (1,2)\}$ and $\mathit{notify}^{\mathcal{D}'_0} = \{(0,1), (0,2), (0,3), (0,4)\}$. For the predicate symbol *ssh_login*, only those tuples are included where the first parameter takes values from the slicing set. This is because the first parameter occurs as the slicing variable c in the formula. For the predicate symbol *notify*, those tuples are included where the first parameter is 0 because the constant symbol $\mathsf{0}$ occurs in the formula.

For the slicing set $S' = \{3, 4\}$, the (φ, c, S')-slice contains the structure $\mathcal{D}''_0$ with $\mathit{ssh\_login}^{\mathcal{D}''_0} = \{(3,3),(4,4)\}$ and $\mathit{notify}^{\mathcal{D}''_0} = \{(0,1),(0,2),(0,3),(0,4)\}$. The tuples in the relation for the predicate symbol *notify* are duplicated in all slices because the first element of the tuples, 0, occurs as a constant symbol in the formula. The condition (D3) in Definition 7 is therefore always satisfied and the tuple is included.

Next, we slice on the variable s instead of c. For the slicing set S, the (φ, s, S)-slice contains the structure $\mathcal{D}'_0$ with $\mathit{ssh\_login}^{\mathcal{D}'_0} = \{(1,1),(1,2)\}$ and $\mathit{notify}^{\mathcal{D}'_0} = \{(0,1),(0,2)\}$. For both of the predicate symbols *ssh_login* and *notify*, only those tuples are included where the second parameter takes values from the slicing set S. This is because the second parameter occurs as the slicing variable s in the formula. For the slicing set S, the (φ, s, S')-slice contains the structure $\mathcal{D}''_0$ with $\mathit{ssh\_login}^{\mathcal{D}''_0} = \{(3,3),(4,4)\}$ and $\mathit{notify}^{\mathcal{D}''_0} = \{(0,3),(0,4)\}$.

According to Definition 9 and Theorem 10 below, a data slicer is a slicer that splits a temporal structure into a family of (φ, x, S)-slices. Furthermore, it refines the given restriction with respect to the given slicing sets.

Definition 9. *Let φ be a formula, $x \in V$ a variable, and $(S^k)_{k\in K}$ a family of slicing sets. The* data slicer $\mathfrak{d}_{\varphi,x,(S^k)_{k\in K}}$ *is the function that maps a temporal structure $(\bar{\mathcal{D}}, \bar{\tau}) \in \mathbf{T}$ and a restriction $\mathcal{R} \in \mathbf{R}$ to the family of temporal structures $(\bar{\mathcal{D}}^k, \bar{\tau}^k)_{k\in K}$ and the family of restrictions $(\mathcal{R}^k)_{k\in K}$, where $(\bar{\mathcal{D}}^k, \bar{\tau}^k)$ is the (φ, x, S'^k)-slice of $(\bar{\mathcal{D}}, \bar{\tau})$, with $S'^k := S^k \cap \{v(x) \mid (v,t) \in \mathcal{R},$ for some $t \in \mathbb{N}\}$, and $\mathcal{R}^k = \{(v,t) \in \mathcal{R} \mid v(x) \in S^k\}$, for each $k \in K$.*

Theorem 10. *A data slicer $\mathfrak{d}_{\varphi,x,(S^k)_{k\in K}}$ is a slicer for the formula φ if the slicing variable x is not bound in φ and $\bigcup_{k\in K} S^k = \mathbb{D}$.*

Slicing Time. Another possibility is to slice a temporal structure along its temporal dimension. A time slice contains all the logged actions over a sufficiently large time interval to determine the policy violations over a given time period. We obtain this time interval from the formula's temporal operators and their intervals. The time slices might overlap, i.e., the logged actions at the beginning of a slice are also be contained at the end of another time slice. We refer to Appendix B.4 for the details of how we produce the time slices, and the soundness and completeness guarantees when monitoring these slices independently. Instead, we illustrate time slicing by the following example.

Example 11. Recall the formula (*P0*) from Section 2. We can split a log into time slices that are equivalent to the original log over 1-day periods. However, to evaluate the formula over a 1-day period, each time slice must also include the log entries of the next 24 hours. This is because the formula's temporal operator $\Diamond_{[0,25)}$ refers to SSH logout events up to 24 hours into the future from a time point. Hence each time point would be monitored twice: once when checking compliance for a specific day and also in the slice for checking compliance of the previous day. If we split the log into time slices that are equivalent to the original log over 1-week periods then 6/7 of the time points are monitored once and 1/7 are monitored twice. This longer period produces less monitoring overhead. However, less parallelization is possible.

Filtering. Removing time points in which all the structures' relations are empty from a temporal structure can significantly speed up monitoring. Empty relations can, for example, originate from the application of a data slicer. Filtering empty time points is sound and complete for the formula (*P0*) from Section 2. However, in general, this is not the case. For instance, for the formula $\Box\, \forall x.\, p(x) \rightarrow \blacklozenge_{[0,1)} \neg q(x)$ the filtering of empty time points prior to monitoring is not sound. We refer to Appendix B.5 for details, including the identification of a fragment for which it is safe to filter empty time points.

3.3 Parallel Implementation

Our slicing framework establishes the theoretical foundations for splitting logs into parts that can be monitored independently in a sound and complete fashion. We now explain how we exploit this in a concrete technical framework for parallelizing computations, the MapReduce framework [12]. Using MapReduce, we monitor a log corresponding to a temporal structure in three phases: map, shuffle, and reduce.

In the *map phase*, the log is fragmented by MapReduce. For each log fragment, we create a stream of log entries in a pointwise fashion. To this end, we implement a collection of slicing functions realizing the slicers and the composition of slicers within MapReduce. Each slicing function takes a single log entry $(\mathcal{D}, \tau)$ as an argument and returns (a) the structure $\mathcal{D}$ unmodified, (b) a structure $\mathcal{D}'$ that results from $\mathcal{D}$ by deleting actions (i.e., $r^{\mathcal{D}'} \subseteq r^{\mathcal{D}}$ must hold for each $r \in R$), or (c) the special symbol $\perp$ indicating that the log entry shall be deleted. We also associate a key with each log entry.

The *shuffle phase* reorganizes log entries into chunks, i.e., streams of key-value pairs with matching keys and each value is a single log entry from the map phase. Chunks can be viewed as slices in the sense of Definition 1. However, it is important that the associated keys are chosen in the map phase in such a way that the shuffle puts all log entries of one slice into the same chunk and that log entries of different slices are put into different chunks.

In the *reduce phase*, we individually monitor each chunk produced during the shuffle phase against the given policy and afterwards we combine the monitoring results thereby yielding the set of all violations. Due to the one-to-one correspondence between chunks and slices, Theorem 4 is applicable; hence no violations are lost by monitoring the constructed chunks in this phase.

In each of the three phases, computations are parallelized by MapReduce. In particular, the map and reduce phases comprise the parallel execution of multiple instances of a map function and a reduce function, respectively. In Appendix C, we provide the details as well as pseudo code for the map, reduce, and slicing functions. Note that the shuffle phase is built into MapReduce.

4 The Google Case Study

Scenario. We consider a setting with over 35,000 computers accessing sensitive resources. These computers are used both within Google, connected directly to the corporate network, and outside of Google, accessing Google's network from remote unsecured networks.

Google uses access-control mechanisms to minimize the risk of unauthorized access to sensitive resources. In particular, computers must obtain time-limited authentication tokens using a tool, which we call AUTH. Furthermore, the Secure Shell protocol (SSH) is used to remotely login to servers. Additionally, to minimize the risk of security exploits, computers must regularly update their configuration and apply security patches according to a centrally managed configuration. To do this, every computer regularly starts an update tool, which we call UPD, connects to a central server to download the latest centrally managed configuration, and attempts to reconfigure and update itself. To prevent over-loading the configuration server, if the computer has recently updated its configuration then the update tool does not attempt to connect to the server.

Policies. The policies we consider specify restrictions on the authorization process, SSH sessions, and the update process. All computers are intended to comply with these policies. However, due to misconfiguration, server outages, hardware failures, and the like, this is not always the case. The policies are as follows.

(*P1*) Entering credentials with the tool AUTH must take at least 1 second. The motivation is that authentication with the tool AUTH should not be automated. That is, the authentication credentials must be entered manually and not by a script when executing the tool.

(*P2*) The tool AUTH may only be used if the computer has been updated to the latest centrally-managed configuration within the last 3 days.

(*P3*) Long-running SSH sessions present a security risk. Therefore, they must not last longer than 24 hours.

(*P4*) Each computer must be updated at least once every 3 days unless it is turned off or not connected to the corporate network.

(*P5*) If a computer connects to the central configuration server and downloads the new configuration, then it should successfully reconfigure itself within the next 30 minutes.

(*P6*) If the tool UPD aborts the update process, claiming that the computer was recently successfully updated, then this update must have occurred within the last 24 hours.

Table 1 presents our formalization of these policies, where we use the predicate symbols given in Table 2. We explain here the less obvious aspects of our formalization. The variable c represents a computer, s represents an SSH session, and t represents the time taken by a user to enter authentication credentials. In (*P3*), we assume that if a computer is disconnected from the corporate network,

Tab. 1: Policy formalization.

policy	MFOTL formula
$(P1)$	$\square\, \forall c.\, \forall t.\, \mathit{auth}(c,t) \rightarrow \mathsf{1000} \prec t$
$(P2)$	$\square\, \forall c.\, \forall t.\, \mathit{auth}(c,t) \rightarrow \blacklozenge_{[0,3d]} \lozenge_{[0,0]}\, \mathit{upd\_success}(c)$
$(P3)$	$\square\, \forall c.\, \forall s.\, \mathit{ssh\_login}(c,s) \wedge \big(\lozenge_{[1min,20min]}\, \mathit{net}(c) \wedge \square_{[0,1d]} \blacksquare_{[0,0]}\, \mathit{net}(c) \rightarrow \lozenge_{[1min,20min]}\, \mathit{net}(c)\big) \rightarrow \lozenge_{[0,1d)} \blacklozenge_{[0,0]}\, \mathit{ssh\_logout}(c,s)$
$(P4)$	$\square\, \forall c.\, \mathit{net}(c) \wedge \big(\lozenge_{[10min,20min]}\, \mathit{net}(c)\big) \wedge \big(\blacklozenge_{[1d,2d]}\, \mathit{alive}(c)\big) \wedge \neg\big(\blacklozenge_{[0,3d]} \lozenge_{[0,0]}\, \mathit{upd\_success}(c)\big) \rightarrow \lozenge_{[0,20min)} \blacklozenge_{[0,0]}\, \mathit{upd\_connect}(c)$
$(P5)$	$\square\, \forall c.\, \mathit{upd\_connect}(c) \wedge \big(\lozenge_{[5min,20min]}\, \mathit{alive}(c)\big) \rightarrow \lozenge_{[0,30min)} \blacklozenge_{[0,0]}\, \mathit{upd\_success}(c) \vee \mathit{upd\_skip}(c)$
$(P6)$	$\square\, \forall c.\, \mathit{upd\_skip}(c) \rightarrow \blacklozenge_{[0,1d]} \lozenge_{[0,0]}\, \mathit{upd\_success}(c)$

Tab. 2: Predicate symbols and their interpretation.

predicate symbol	description
$\mathit{alive}(c)$	The computer c is running. This event is generated at least once every 20 minutes when c is running but at most twice every 5 minutes.
$\mathit{net}(c)$	The computer c is connected to the corporate network. This event is generated at least once every 20 minutes when c is connected to the corporate network but at most once every 5 minutes.
$\mathit{auth}(c,t)$	The tool AUTH is invoked to obtain an authentication token on the computer c. The second argument t indicates the time in milliseconds it took the user to enter the authentication credentials.
$\mathit{upd\_start}(c)$	The tool UPD started on the computer c.
$\mathit{upd\_connect}(c)$	The tool UPD on the computer c connected to the central server and downloaded the latest configuration.
$\mathit{upd\_success}(c)$	The tool UPD updated the configuration and applied patches on the computer c.
$\mathit{upd\_skip}(c)$	The tool UPD on the computer c terminated because it believes that the computer was recently updated.
$\mathit{ssh\_login}(c,s)$	An SSH session with identifier s to the computer c was opened. We use the session identifier s to match the login event with the corresponding logout event.
$\mathit{ssh\_logout}(c,s)$	An SSH session with identifier s to the computer c was closed.

then the SSH session is closed. In $(P4)$, because of the subformula $\blacklozenge_{[1d,2d]}\, \mathit{alive}(c)$, we only consider computers that have recently been used. In particular, the subformula suppresses false positives stemming from newly installed computers, which do not generate *alive* events prior to their installation. Similarly, we only require an update of a computer if it is connected to the network for a given amount of time. In $(P5)$, since a computer can be turned off after downloading the latest configuration but before modifying its local configuration, we only require a successful update if the computer is still running 5 to 20 minutes after downloading the new configuration.

Logs. The computers log entries describing their local system actions and upload their logs to a log cluster. Approximately 1 TB of log data is uploaded each day. We restricted ourselves to log data that spans approximately two years. We then processed the uploaded data to obtain a temporal structure consisting of the events relevant for the policies considered. Since events occur concurrently, we collapsed the temporal structure [8], that is, the structures at time points with

Tab. 3: Log statistics.

event	count	
alive	16 B	(15,912,852,267)
net	8 B	(7,807,707,082)
auth	8 M	(7,926,789)
upd_start	65 M	(65,458,956)
upd_connect	46 M	(45,869,101)
upd_success	32 M	(31,618,594)
upd_skip	6 M	(5,960,195)
ssh_login	1 B	(1,114,022,780)
ssh_logout	1 B	(1,047,892,209)

Tab. 4: Monitor performance.

policy	runtime (overall)	runtime (per slice)			memory (per slice)	
		median	max	cumulative	median	max
	[hh:mm]	[sec]	[hh:mm]	[days]	[MB]	[MB]
(*P1*)	2:04	169	0:46	21.4	6.1	6.1
(*P2*)	2:10	170	0:51	21.4	6.1	10.3
(*P3*)	11:56	170	10:40	22.7	7.1	510.2
(*P4*)	2:32	169	1:06	21.3	9.2	13.1
(*P5*)	2:28	168	1:01	21.3	6.1	6.1
(*P6*)	2:13	168	0:48	21.1	6.1	7.1

equal timestamps are merged into a single structure. By doing this, we make the assumption that equally timestamped events happen simultaneously. The size of the collapsed temporal structure is approximately 600 MB per day on average and 0.4 TB for the two years, in a protocol buffers [16] format. It contains approximately 77.2 million time points and 26 billion events, i.e., tuples in the relations interpreting the predicate symbols. Table 3 presents a breakdown of the numbers of the events in the temporal structure by predicate symbols.

Slicing and Monitoring. For each policy, we used 1,000 computers for slicing and monitoring. Here we used Google's MapReduce framework [12] and the MONPOLY tool [7]. We split the collapsed temporal structure into 10,000 slices so that each computer processed 10 slices on average. The decision to use 10 times more slices than computers makes the individual map and reduce computations small. This has the advantage that if the monitoring of a slice fails and must be restarted, then less computation is wasted. Furthermore, for slicing and monitoring, we used the formulas in Table 1 without universally quantifying over the variables c, t, and s. The resulting formulas fall into the fragment that the MONPOLY tool handles and our slicing techniques from Section 3 are applicable, i.e., they are sound and complete.

We employed data slicing with respect to the variable c, which occurs in all the atomic subformulas with a predicate symbol, and filtering of empty time points. We did not slice by time. Our implementation generates the primary keys of the key-value pairs emitted by a mapper from c's interpretation in an event. Concretely, we apply the MurmurHash [25] function to this value and take the remainder after dividing it by 10,000 (the number of slices). The values of the key-value pairs emitted by the implemented mappers are log entries consisting of a single event and a timestamp. Slices are generated with respect to the conjunction of all policies. Figure 1 depicts the distribution of the size of the slices. Note that generating the slices for each policy individually would result in smaller slices and thus simplify the monitoring process. Note too that although we use the same set of slices for all policies, each policy was checked separately and the slices were generated during this check.

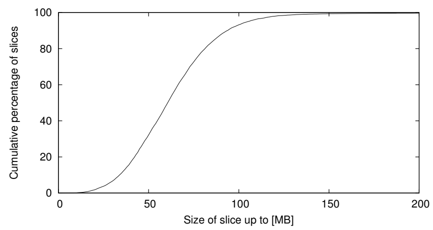

Fig. 1: Distribution of the size of the log slices.

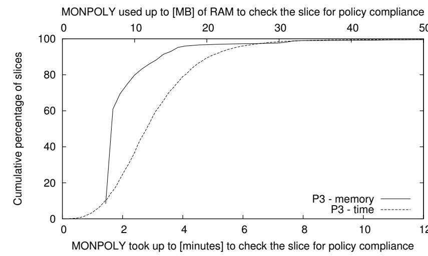

Fig. 2: Distribution of memory (upper x-axis) and time (lower x-axis) used to monitor individual slices for (*P3*).

Evaluation. Figure 1 shows the distribution of the sizes of the slices in the format used as input for MONPOLY. On the y-axis is the percentage of slices whose size is less than or equal to the value on the x-axis. The median size of a slice is 61 MB and 99% of the slices have a size of at most 135 MB. There are three slices with sizes over 1 GB and the largest slice is 1.8 GB. Recall that we used the same slicing method for all policies. The sum of the sizes of all slices (0.6 TB) is larger than the size of the collapsed temporal structure (0.4 TB). Since we slice by the computer (variable c), the slices do not overlap. However, some overhead results from timestamps and predicate symbol names being replicated in multiple slices. Moreover, we consider the sizes of the slices in the more verbose text-based MONPOLY format rather than the protocol buffers format.

Table 4 shows the performance of our monitoring solution. The second column shows for each policy the time for the entire MapReduce job, including both slicing and monitoring, that is, the time from starting the MapReduce job until the monitor finished on the last slice and its output was collected by the corresponding reducer. Except for (*P3*), the slicing and monitoring took up to $2\frac{1}{2}$ hours. Slicing and monitoring (*P3*) took almost 12 hours. Table 4 also gives details about the monitoring of the individual slices. The overhead of the MapReduce framework and time necessary for slicing is small; most resources are spent on monitoring the slices. The cumulative running times roughly amount to the time necessary to monitor all slices sequentially on a single computer.

We first discuss the time taken to monitor the individual slices and then the memory used. For (*P3*), Figure 2 shows on the y-axis the percentage of slices for which the monitoring time is within the limit on the lower x-axis. We do not give the curves for the other policies as they are similar to (*P3*). The similarities indicate that for most slices the monitoring time does not vary much across the considered policies. 99% of the slices are monitored within 8.2 minutes each and do not need more than 35 MB of memory.

(*P3*) required substantially more time to monitor than the other formulas due to the nesting of temporal operators. This additional overhead is particularly pronounced on large slices and results in waiting for a few large slices that take substantially longer to monitor than the rest. There are several options to deal with such slices. We can stop the monitor after a timeout and ignore the slices and any policy violations involving them. Note that the monitoring of the other slices and the validity of violations found on them would be unaffected. Alternatively, we can split large slices into smaller ones, either prior to monitoring or after a timeout when monitoring a large slice. For (*P3*), we can slice further by the variable c and also by s. We can also slice by time.

Due to the sensitive nature of the logged data, we do not report here on the policy violations. However, we remark that monitoring a large population of computers and aggregating the violations found can be used to identify systematic policy violations and policy violations due to system misconfiguration. An example of the former is not letting a computer update after the weekend before using it to access sensitive resources on a Monday; cf. (*P2*). An example of the latter is that the monitoring helped determine when the update process was not operating as expected for certain types of computers during a specific time period. This information can be useful for identifying seemingly unrelated changes in the configuration of other components in the IT infrastructure.

Given the amount of logged data and the modest computational power (1,000 computers in a MapReduce cluster), the monitoring times are in general low, and reasonable even for (*P3*). The presented monitoring solution allows us to cope with even larger logs and to speed-up the monitoring process by deploying additional slicing mechanisms provided by our general framework and by using additional computers in a MapReduce cluster.

5 Related Work

This work builds upon and extends the work by Basin et al. [7–9], where a single monitor is used to check system compliance with respect to policies expressed in metric first-order temporal logic. By parallelizing and distributing the monitoring process, we overcome a central limitation of this prior work and enable it to scale to logging scenarios that are substantially larger than those previously considered [8], namely, approximately 100 times larger in terms of the number of events and 50 times larger in the data volume.

For different logic-based specification languages, various monitoring algorithms exist, e.g., [5, 6, 10, 11, 13, 15, 17–19, 23, 24]. These algorithms have been developed with different applications in mind, such as intrusion detection [23], program verification [5], and checking temporal integrity constraints for databases [11]. In principle, these algorithms can also be used to check compliance of IT systems, where a single centralized monitor observes the system online or checks the system logs offline. However, none of these algorithms, including the one of Basin et al. [9], would scale to IT system of realistic size due to the lack of parallelization.

Similar to our work, Barre et al. [4] monitor parts of a log in parallel and independently of other log parts with a MapReduce framework. While we split the log into multiple slices and evaluate the entire formula on these slices in parallel, they evaluate the given formula in multiple iterations of MapReduce. All subformulas of the same depth are evaluated in the same MapReduce job and the results are used to evaluate subformulas of a lower depth during another MapReduce job. The evaluation of a subformula is performed in both the map and the reduce phase. While the evaluation in the map phase is parallelized for different time points of the log, the results of the map phase for a subformula for the whole log are collected and processed by a single reducer. The reducer therefore becomes a bottleneck and their approach's scalability remains unclear. Furthermore, in their experiments they used a log with fewer than five million entries and performed monitoring on a single computer with respect to formulas of a propositional temporal logic, which is limited in its ability to express realistic policies.

Roşu and Chen [22] present a generic monitoring algorithm for parametric specifications. They group logged events into slices by their parameter instances, one slice for each parameter value in case of a single parameter and one slice for each combination of values when the specification has multiple parameters. The slices are then processed by a monitoring algorithm unaware of parameters. In contrast to our work, they do not provide a solution for parallelizing the monitoring process; they provide an algorithmic solution to generate the slices online. We note that the extension of the temporal logic LTL with parameterized propositions, as considered by Roşu and Chen, is less expressive than a first-order extension like MFOTL, used in our work. Roşu and Chen also report on experiments with logs containing up to 155 million entries, all monitored on a single computer. This is orders of magnitude smaller than the log in our case study.

6 Conclusion

We presented a scalable solution for checking compliance of IT systems, where behavior is monitored offline and checked against policies. To achieve scalability, we parallelize monitoring, supported by a framework for slicing logs and an algorithmic realization within the MapReduce framework.

MapReduce is particularly well suited for implementing parallel monitoring. It allows us to efficiently reorganize huge logs into slices. It also allocates and distributes the computations for monitoring the slices, accounting for the available computational resources, the location of the logged data, failures, etc. Finally, additional computers can easily be added to speedup the monitoring process when splitting the log into more slices, thereby increasing the degree of parallelization.

Our slicing framework allows logs to be sliced in multiple dimensions by composing different slicing methods. As future work, we will evaluate different possibilities of obtaining a larger number of smaller slices that are equally expensive to monitor. We also plan to adapt our approach to check system compliance *online*. In this regard, there are extensions and alternatives to the MapReduce framework for online data processing, such as S4 [21] and STORM [20], which can potentially be used to obtain a scalable online monitoring solution.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases: The Logical Level.* Addison Wesley, 1994.
2. R. Alur and T. A. Henzinger. Logics and models of real time: A survey. In *Proceedings of the 1991 REX Workshop on Real Time: Theory in Practice*, volume 600 of *Lect. Notes Comput. Sci.*, pages 74–106. Springer, 1992.
3. C. Baier and J.-P. Katoen. *Principles of Model Checking.* The MIT Press, 2008.
4. B. Barre, M. Klein, M. Soucy-Boivin, P.-A. Ollivier, and S. Hallé. MapReduce for parallel trace validation of LTL properties. In *Proceedings of the 3rd International Conference on Runtime Verification (RV)*, volume 7687 of *Lect. Notes Comput. Sci.*, pages 184–198. Springer, 2013.
5. H. Barringer, A. Goldberg, K. Havelund, and K. Sen. Rule-based runtime verification. In *Proceedings of the 5th International Conference on Verification, Model Checking and Abstract Interpretation*, volume 2937 of *Lect. Notes Comput. Sci.*, pages 44–57. Springer, 2004.
6. H. Barringer, A. Groce, K. Havelund, and M. Smith. Formal analysis of log files. *J. Aero. Comput. Inform. Comm.*, 7:365–390, 2010.
7. D. Basin, M. Harvan, F. Klaedtke, and E. Zălinescu. MONPOLY: Monitoring usage-control policies. In *Proceedings of the 2nd International Conference on Runtime Verification (RV)*, volume 7186 of *Lect. Notes Comput. Sci.*, pages 360–364. Springer, 2012.
8. D. Basin, M. Harvan, F. Klaedtke, and E. Zălinescu. Monitoring data usage in distributed systems. *IEEE Trans. Software Eng.*, 39(10):1403–1426, 2013.
9. D. Basin, F. Klaedtke, S. Müller, and B. Pfitzmann. Runtime monitoring of metric first-order temporal properties. In *Proceedings of the 28th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 2 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 49–60. Schloss Dagstuhl - Leibniz Center for Informatics, 2008.
10. A. Bauer, R. Goré, and A. Tiu. A first-order policy language for history-based transaction monitoring. In *Proceedings of the 6th International Colloquium on Theoretical Aspects of Computing (ICTAC)*, volume 5684 of *Lect. Notes Comput. Sci.*, pages 96–111. Springer, 2009.
11. J. Chomicki. Efficient checking of temporal integrity constraints using bounded history encoding. *ACM Trans. Database Syst.*, 20(2):149–186, 1995.
12. J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI)*, pages 137–150. USENIX Association, 2004.
13. N. Dinesh, A. K. Joshi, I. Lee, and O. Sokolsky. Checking traces for regulatory conformance. In *Proceedings of the 8th International Workshop on Runtime Verification (RV)*, volume 5289 of *Lect. Notes Comput. Sci.*, pages 86–103, 2008.
14. H. Enderton. *A Mathematical Introduction to Logic.* Academic Press, 2nd edition, 2001.
15. D. Garg, L. Jia, and A. Datta. Policy auditing over incomplete logs: theory, implementation and applications. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS)*, pages 151–162. ACM Press, 2011.
16. Google. Protocol Buffers: Googles Data Interchange Format, 2013. `http://code.google.com/p/protobuf/`.
17. A. Groce, K. Havelund, and M. Smith. From scripts to specification: The evaluation of a flight testing effort. In *Proceedings of the 32nd ACM/IEEE International*

Conference on Software Engineering (ICSE), volume 2, pages 129–138. ACM Press, 2010.

18. S. Hallé and R. Villemaire. Runtime enforcement of web service message contracts with data. *IEEE Trans. Serv. Comput.*, 5(2):192–206, 2012.
19. F. M. Maggi, M. Montali, M. Westergaard, and W. M. P. van der Aalst. Monitoring business constraints with linear temporal logic: An approach based on colored automata. In *Proceedings of the 9th International Conference on Business Process Management (BPM)*, volume 6896 of *Lect. Notes Comput. Sci.*, pages 132–147. Springer, 2011.
20. N. Marz. STORM: Distributed and fault-tolerant realtime computation. `http://storm-project.net`.
21. L. Neumeyer, B. Robbins, A. Nair, and A. Kesari. S4: Distributed stream computing. In *Proceedings of the 11th International Conference on Data Mining Workshops (ICDMW)*, pages 170–177. IEEE Computer Society, 2010.
22. G. Roşu and F. Chen. Semantics and algorithms for parametric monitoring. *Log. Method. Comput. Sci.*, 8(1):1–47, 2012.
23. M. Roger and J. Goubault-Larrecq. Log auditing through model-checking. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop (CSFW)*, pages 220–234. IEEE Computer Society, 2001.
24. A. P. Sistla and O. Wolfson. Temporal triggers in active databases. *IEEE Trans. Knowl. Data Eng.*, 7(3):471–486, 1995.
25. Wikipedia. MurmurHash — Wikipedia, the free encyclopedia, 2013. `https://en.wikipedia.org/wiki/MurmurHash`.

A Additional Details: MFOTL

A *signature* $\mathcal{S}$ is a tuple (C, R, ι), where C is a finite set of constant symbols, R is a finite set of predicate symbols disjoint from C, and the function $\iota : R \to \mathbb{N}$ associates each predicate symbol $r \in R$ with an arity $\iota(r) \in \mathbb{N}$. In the following, let $\mathcal{S} = (C, R, \iota)$ be a signature and V a countably infinite set of variables, assuming $V \cap (C \cup R) = \emptyset$. Moreover, let $\mathbb{I}$ be the set of nonempty intervals over $\mathbb{N}$. We write $[b, b')$ for the interval of natural numbers from $b \in \mathbb{N}$ to $b' \in \mathbb{N} \cup \{\infty\}$, i.e., $[b, b') = \{a \in \mathbb{N} \mid b \le a < b'\}$.

A *temporal structure* over $\mathcal{S}$ is a pair $(\bar{\mathcal{D}}, \bar{\tau})$, where $\bar{\mathcal{D}} = (\mathcal{D}_0, \mathcal{D}_1, \dots)$ is a sequence of structures over $\mathcal{S}$ and $\bar{\tau} = (\tau_0, \tau_1, \dots)$ is a sequence of natural numbers, where the following conditions hold.

1. The sequence $\bar{\tau}$ is monotonically increasing (i.e., $\tau_i \le \tau_{i+1}$, for all $i \ge 0$) and makes progress (i.e., for every $i \ge 0$, there is some $j > i$ such that $\tau_j > \tau_i$).
2. $\bar{\mathcal{D}}$ has constant domains, i.e., $|\mathcal{D}_i| = |\mathcal{D}_{i+1}|$, for all $i \ge 0$. Its elements are strictly linearly ordered by the relation $<$.
3. Each constant symbol $c \in C$ has a rigid interpretation, i.e., $c^{\mathcal{D}_i} = c^{\mathcal{D}_{i+1}}$, for all $i \ge 0$.

Note that there can be successive time points with equal timestamps. Furthermore, the relations $r^{\mathcal{D}_0}, r^{\mathcal{D}_1}, \dots$ in a temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$ corresponding to a predicate symbol $r \in R$ may change over time. In contrast, the interpretation of the constant symbols $c \in C$ and the domain of the $\mathcal{D}_i$s do not change. We denote them by $c^{\bar{\mathcal{D}}}$ and $|\bar{\mathcal{D}}|$, respectively.

A *valuation* is a mapping $v : V \to |\bar{\mathcal{D}}|$. We abuse notation by applying a valuation v also to constant symbols $c \in C$, with $v(c) = c^{\bar{\mathcal{D}}}$, and vectors over $V \cup C$. Vectors are written in the usual way. For example, we write $r(\bar{t})$ instead of $r(t_1, \dots, t_{\iota(r)})$, assuming that $\bar{t}$ has the dimension $\iota(r)$. We write $f[x \mapsto y]$ for altering a function $f : X \to Y$ pointwise at $x \in X$. In particular, for a valuation v, a variable x, and $d \in |\bar{\mathcal{D}}|$, $v[x \mapsto d]$ is the valuation mapping x to d and leaving other variables' valuation unchanged.

MFOTL's satisification relation $\models$ is inductively defined over the formula structure. For a $(\bar{\mathcal{D}}, \bar{\tau})$ temporal structure, with $\bar{\mathcal{D}} = (\mathcal{D}_0, \mathcal{D}_1, \dots)$ and $\bar{\tau} = (\tau_0, \tau_1, \dots)$, v a valuation, and $i \in \mathbb{N}$, we define:

$$
\begin{array}{lll}
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models t \approx t' & \text{iff} & v(t) = v(t') \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models t \prec t' & \text{iff} & v(t) < v(t') \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models r(\bar{t}) & \text{iff} & v(\bar{t}) \in r^{\mathcal{D}_i} \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \neg\varphi & \text{iff} & (\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \varphi \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \varphi \lor \psi & \text{iff} & (\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \varphi \text{ or } (\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \psi \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \exists x.\, \varphi & \text{iff} & (\bar{\mathcal{D}}, \bar{\tau}, v[x \mapsto d], i) \models \varphi, \text{for some } d \in |\bar{\mathcal{D}}| \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \bullet_I\, \varphi & \text{iff} & i > 0,\ \tau_i - \tau_{i-1} \in I, \text{ and } (\bar{\mathcal{D}}, \bar{\tau}, v, i-1) \models \varphi \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \bigcirc_I\, \varphi & \text{iff} & \tau_{i+1} - \tau_i \in I \text{ and } (\bar{\mathcal{D}}, \bar{\tau}, v, i+1) \models \varphi \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \varphi\, \mathsf{S}_I\, \psi & \text{iff} & \text{for some } j \le i,\ \tau_i - \tau_j \in I, (\bar{\mathcal{D}}, \bar{\tau}, v, j) \models \psi, \\
 & & \text{and } (\bar{\mathcal{D}}, \bar{\tau}, v, k) \models \varphi, \text{for all } k \in [j+1, i+1) \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \varphi\, \mathsf{U}_I\, \psi & \text{iff} & \text{for some } j \ge i,\ \tau_j - \tau_i \in I,\ (\bar{\mathcal{D}}, \bar{\tau}, v, j) \models \psi, \\
 & & \text{and } (\bar{\mathcal{D}}, \bar{\tau}, v, k) \models \varphi,\ \text{for all } k \in [i, j)
\end{array}
$$

For instance, the formula $\bigcirc_I \varphi$ is satisfied in a temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$ under valuation v at time point i if the elapsed time to the next time stamp in $\bar{\tau}$ is within the time interval I (i.e., $\tau_{i+1} - \tau_i \in I$) and φ is satisfied at time point $i+1$ in $(\bar{\mathcal{D}}, \bar{\tau})$ under v. Note that the time interval I is used relative to the current time stamp τ_i in the semantics of all four temporal operators.

B Additional Details: Slicing Framework

B.1 Slicers

We provide the proof details for Theorem 4.

(1) implies (2) because $(\mathcal{R}^k)_{k\in K}$ refines $\mathcal{R}$ and because $(\bar{\mathcal{D}}^k, \bar{\tau}^k)$ is $\mathcal{R}^k$-sound for $(\bar{\mathcal{D}}, \bar{\tau})$ and φ, for each $k \in K$.

To show that (2) implies (1), we prove the contrapositive. Let v be a valuation and $i \in \mathbb{N}$ such that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \varphi$ and (v, τ_i) is permitted by $\mathcal{R}$. Because $(\mathcal{R}^k)_{k\in K}$ refines $\mathcal{R}$, there is a $k \in K$ such that (v, τ_i) is permitted by $\mathcal{R}^k$. Because $(\bar{\mathcal{D}}^k, \bar{\tau}^k)$ is $\mathcal{R}^k$-complete for $(\bar{\mathcal{D}}, \bar{\tau})$ and φ, we have that $(\bar{\mathcal{D}}^k, \bar{\tau}^k, v, j) \not\models \varphi$, for some $j \in \mathbb{N}$ with $\tau^k_j = \tau_i$.

Note that Theorem 4 does not require that if the original temporal structure is violated then a slice is violated for the same valuation and timestamp as the original temporal structure. The theorem's proof establishes a stronger result. Namely, that the valuation and timestamp for a violation match between the original temporal structure and the slice.

B.2 Composition

We provide the proof details for Theorem 6.

We show that $\mathfrak{s}'_\varphi \circ_{\hat{k}} \mathfrak{s}_\varphi$ satisfies the conditions (S1) to (S3) in Definition 3. Regarding (S1), $\mathfrak{s}_\varphi$ is a slicer and therefore the family $(\mathcal{R}^k)_{k\in K}$ refines $\mathcal{R}$. If $\hat{k} \notin K$, then we are done. If $\hat{k} \in K$, then $\mathfrak{s}'_\varphi$ is a slicer and therefore the family $(\mathcal{R}^k)_{k\in K'}$ refines $\mathcal{R}^{\hat{k}}$. From $K \cap K' = \emptyset$, it follows that $(\mathcal{R}^k)_{k\in (K\setminus\{\hat{k}\})\cup K'}$ refines $\mathcal{R}$.

Regarding (S2), $\mathfrak{s}_\varphi$ is a slicer and therefore $(\bar{\mathcal{D}}^k, \bar{\tau}^k)$ is $\mathcal{R}^k$-sound for $(\bar{\mathcal{D}}, \bar{\tau})$ and φ, for every $k \in K$. If $\hat{k} \notin K$, then we are done. If $\hat{k} \in K$, then $\mathfrak{s}'_\varphi$ is a slicer and therefore $(\bar{\mathcal{D}}^k, \bar{\tau}^k)$ is $\mathcal{R}^k$-sound for $(\bar{\mathcal{D}}^{\hat{k}}, \bar{\tau}^{\hat{k}})$ and φ, for every $k \in K'$. Because $(\mathcal{R}^k)_{k\in K'}$ refines $\mathcal{R}^{\hat{k}}$ and because $(\bar{\mathcal{D}}^{\hat{k}}, \bar{\tau}^{\hat{k}})$ is $\mathcal{R}^{\hat{k}}$-sound for $(\bar{\mathcal{D}}, \bar{\tau})$ and φ, it follows that $(\bar{\mathcal{D}}^k, \bar{\tau}^k)$ is $\mathcal{R}^k$-sound for $(\bar{\mathcal{D}}, \bar{\tau})$ and φ, for every $k \in K'$. From $K \cap K' = \emptyset$, it follows that $(\bar{\mathcal{D}}^k, \bar{\tau}^k)$ is sound for $(\bar{\mathcal{D}}, \bar{\tau})$ and φ, for every $k \in (K \setminus \{\hat{k}\}) \cup K'$.

The condition (S3) is proved analogously to (S2).

B.3 Slicing Data

The following lemma states that a (φ, x, S)-slice is truth preserving for valuations of the slicing variable x within the slicing set S. We use the lemma to establish soundness and completeness in Theorem 10, showing that a data slicer as defined in Definition 9 is a slicer, and therefore Theorem 4 applies.

Lemma 12. *Let φ be a formula, $x \in V$ a variable not bound in φ, $S \subseteq \mathbb{D}$ a slicing set, and $(\bar{\mathcal{D}}', \bar{\tau}) \in \mathbf{T}$ the (φ, x, S)-slice of $(\bar{\mathcal{D}}, \bar{\tau}) \in \mathbf{T}$. For all $i \in \mathbb{N}$ and valuations v with $v(x) \in S$, $(\bar{\mathcal{D}}', \bar{\tau}, v, i) \models \varphi$ iff $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \varphi$.*

We provide the proof details for Lemma 12. We proceed by induction over the structure of the formula φ. The base case consists of the atomic formulas $t \prec t'$, $t \approx t'$, and $r(\bar{t})$. Satisfaction of $t \prec t'$ and $t \approx t$ depends only on the valuation, therefore $(\bar{\mathcal{D}}', \bar{\tau}, v, i) \models t \prec t'$ iff $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models t \prec t'$ and $(\bar{\mathcal{D}}', \bar{\tau}, v, i) \models t \approx t'$ iff $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models t \approx t'$. We show the two directions of the equivalence separately for an atomic subformula $r(\bar{t})$ of φ.

($\Rightarrow$) From $(\bar{\mathcal{D}}', \bar{\tau}, v, i) \models r(\bar{t})$ it follows that $v(\bar{t}) \in r^{\mathcal{D}'_i}$. Since $(\bar{\mathcal{D}}', \bar{\tau})$ is a data slice of $(\bar{\mathcal{D}}, \bar{\tau})$ it follows that $v(\bar{t}) \in r^{\mathcal{D}_i}$ and hence $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models r(\bar{t})$.

($\Leftarrow$) From $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models r(\bar{t})$ it follows that $v(\bar{t}) \in r^{\mathcal{D}_i}$. Let j with $1 \leq j \leq \iota(r)$. We make a case split based on whether the term t_j in $r(\bar{t})$ is the slicing variable x, another variable $y \neq x$, or a constant symbol c.

1. If t_j is the slicing variable x then from $v(x) \in S$ we know that $v(t_j) \in S$. Therefore, (D1) is satisfied.
2. If t_j is a variable $y \neq x$ then (D2) is satisfied.
3. If t_j is the constant symbol c then $v(t_j) = c^{\bar{\mathcal{D}}}$ and hence (D3) is satisfied.

It follows that $v(\bar{t}) \in r^{\mathcal{D}'_i}$ and hence $(\bar{\mathcal{D}}', \bar{\tau}, v, i) \models r(\bar{t})$.

The step case follows straightforwardly from the base case and because the slice and the original temporal structure have the same sequence of timestamps $\bar{\tau}$. In particular, any difference when evaluating a formula stems only from a difference in the evaluation of its atomic subformulas.

We provide the proof details for Theorem 10. We show that $\mathfrak{d}_{\varphi, x, (S^k)_{k \in K}}$ satisfies the conditions (S1) to (S3) in Definition 3. For (S1), note that the family $(R^k)_{k \in K}$ refines the given restriction, which follows directly from $\mathcal{R}^k = \{(v, t) \in \mathcal{R} \mid v(x) \in S^k\}$ and $\bigcup_{k \in K} S^k = \mathbb{D}$. (S2) and (S3) follow directly from Lemma 12.

B.4 Slicing Time

We present slicers that split temporal structures in their temporal dimension. A temporal structure $(\bar{\mathcal{D}}', \bar{\tau}') \in \mathbf{T}$ is a *time slice* of $(\bar{\mathcal{D}}, \bar{\tau}) \in \mathbf{T}$ if $(\bar{\mathcal{D}}', \bar{\tau}')$ is a slice of $(\bar{\mathcal{D}}, \bar{\tau})$, where $\ell \in \mathbb{N} \cup \{\infty\}$ and the function $s : [0, \ell) \to \mathbb{N}$ are according to Definition 1 such that $r^{\mathcal{D}'_i} = r^{\mathcal{D}_{s(i)}}$, for all $r \in R$ and $i \in [0, \ell)$.

For a formula φ, we determine a time range that suffices to evaluate a formula on a single time point of a temporal structure. The time range depends on the temporal operators and their intervals. The temporal structure is then split by a time slicer into slices according to this time range. Each slice can be monitored independently. We proceed as follows. In Definition 13, we define the sufficient time range for a formula. We formalize the slicing of a log by time in Definition 14 and in Definition 15, we formalize the time slicer.

We extend our notation for intervals over $\mathbb{N}$ to intervals over $\mathbb{Z}$. For example, for $b, b' \in \mathbb{Z}$, $[b, b']$ denotes the set $\{a \in \mathbb{Z} \mid b \leq a \leq b'\}$. Moreover, we use the following operations, where I and J are intervals over $\mathbb{Z}$.

– $I \Cup J$ is the smallest interval containing I and J.
– $I \oplus J := \{i + j \mid i \in I \text{ and } j \in J\}$.

Definition 13. *The* relative interval *of the formula* φ, $\mathrm{RI}(\varphi) \subseteq \mathbb{Z}$, *is defined recursively over the formula structure:*

– $\{0\}$, *if* φ *is an atomic formula.*
– $\mathrm{RI}(\psi)$, *if* φ *is of the form* $\neg\psi$ *or* $\exists x.\, \psi$.
– $\mathrm{RI}(\psi) \Cup \mathrm{RI}(\chi)$, *if* φ *is of the form* $\psi \vee \chi$.
– $(-b, 0] \Cup \big((-b, -a] \oplus \mathrm{RI}(\psi)\big)$, *if* φ *is of the form* $\bullet_{[a,b)}\, \psi$.
– $[0, b) \Cup \big([a, b) \oplus \mathrm{RI}(\psi)\big)$, *if* φ *is of the form* $\bigcirc_{[a,b)}\, \psi$.
– $(-b, 0] \Cup \big((-b, 0] \oplus \mathrm{RI}(\psi)\big) \Cup \big((-b, -a] \oplus \mathrm{RI}(\chi)\big)$, *if* φ *is of the form* $\psi \,\mathsf{S}_{[a,b)}\, \chi$.
– $[0, b) \Cup \big([0, b) \oplus \mathrm{RI}(\psi)\big) \Cup \big([a, b) \oplus \mathrm{RI}(\chi)\big)$, *if* φ *is of the form* $\psi \,\mathsf{U}_{[a,b)}\, \chi$.

We give intuition for Definition 13. The relative interval of φ specifies a time range, which contains relative timestamps. These relative timestamps describe time points that are sufficient to evaluate φ on the current time point. Relative timestamps that refer to the future are positive integers and relative timestamps that refer to the past are negative integers. In the following, we give some intuition about the different cases of RI's definition.

The evaluation of an atomic formula φ only depends on the current time point. Therefore, it suffices to consider time points with equal timestamps, and hence $\mathrm{RI}(\varphi) = \{0\}$. To evaluate a formula of the form $\neg\psi$, $\exists x.\, \psi$, or $\psi \vee \chi$ it is sufficient to consider the time points needed to evaluate its subformulas. Hence, we choose the smallest interval subsuming the relative intervals of the subformulas.

The evaluation of $\bigcirc_I\, \psi$ depends only on the time points of whose timestamps that fall in the interval needed for ψ's evaluation, shifted by the interval I. Moreover, the timestamp of the next time point must be the same in the time slice as in the original log. This is ensured by considering the interval from 0 to the furthest value from 0 in I. Considering only an interval I with $0 \notin I$ would allow for additional time points to be inserted in the time slice between the current time point and the original next time point. The evaluation of $\psi \,\mathsf{U}_I\, \chi$, with $I = [a, b)$, depends on having the same timestamps for the time points in the time slice as in the original log between the current time point and the one furthest away, but with its timestamp still falling within I. This is ensured by $[0, b)$. The subformula ψ is evaluated on time points between the current time point and the furthest time point with a timestamp that falls into I, so we need to consider the relative interval of this subformula shifted by $[0, b)$. The subformula χ is evaluated only on time points whose relative timestamps fall within I, so we need to consider the relative interval of this subformula shifted by $[a, b)$. Formulas of the form $\bullet_I\, \psi$ and $\psi \,\mathsf{S}_I\, \chi$ are treated similarly to formulas with the corresponding future operators. However, the their relative intervals are mirrored over 0, since these temporal operators refer to the past.

Definition 14. *Let* $T \subseteq \mathbb{Z}$ *be an interval and* $(\bar{\mathcal{D}}, \bar{\tau}) \in \mathbf{T}$. *The* T-slice *of* $(\bar{\mathcal{D}}, \bar{\tau})$ *is the time slice* $(\bar{\mathcal{D}}', \bar{\tau}')$ *of* $(\bar{\mathcal{D}}, \bar{\tau})$ *with* $\ell = |\{i \in \mathbb{N} \mid \tau_i \in T\}|$, $s(i') = i' + c$, *where* $c = \min\{i \in \mathbb{N} \mid \tau_i \in T\}$, $\mathcal{D}'_{i'} = \mathcal{D}_{s(i')}$, *for all* $i' \in [0, \ell)$, *and* $\tau'_\ell \notin T$.

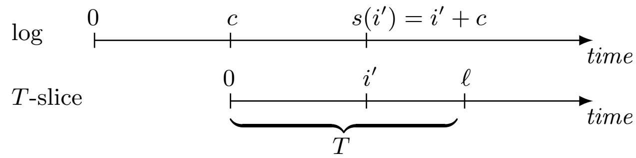

Fig. 3: Illustration of a T-slice.

Figure 3 illustrates Definition 14, where the original log refers to the temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$ and a T-slice of the original log to $(\bar{\mathcal{D}}', \bar{\tau}')$. Intuitively, the first time point in a T-slice is the first time point in $(\bar{\mathcal{D}}, \bar{\tau})$ with the timestamp in T. There are ℓ time points in $(\bar{\mathcal{D}}, \bar{\tau})$ whose timestamps fall into T. Those time points are identical in the T-slice. To ensure the soundness and completeness of time slices, the ℓth time point in the T-slice must have a timestamp that lies outside of T, just like the corresponding time point in $(\bar{\mathcal{D}}, \bar{\tau})$.

For a time range that extends beyond the evaluated time point, the slices must partially overlap. Since the monitor must inspect those overlapping parts once for each slice, we try to minimize the overlap. This leads to a trade-off between how many slices we create (and hence how many monitors can run in parallel) and how much overhead there is due to monitoring overlapping time points. This overhead is illustrated in Example 11.

Definition 15. *The* time slicer $\mathfrak{t}_{\varphi,(I^k)_{k\in K}}$ *for the formula* φ *and the family of intervals* $(I^k)_{k\in K}$ *is the function mapping* $(\bar{\mathcal{D}}, \bar{\tau}) \in \mathbf{T}$ *and* $\mathcal{R} \in \mathbf{R}$ *to the family of temporal structures* $(\bar{\mathcal{D}}^k, \bar{\tau}^k)_{k\in K}$ *and the family of restrictions* $(\mathcal{R}^k)_{k\in K}$*, where* $(\bar{\mathcal{D}}^k, \bar{\tau}^k)$ *is the* T^k*-slice of* $(\bar{\mathcal{D}}, \bar{\tau})$*, with* T^k *the smallest interval containing* $\big(I^k \cap \{t \in \mathbb{N} \mid (v,t) \in \mathcal{R},\ \text{for some valuation } v\}\big) \oplus \mathrm{RI}(\varphi)$*, and* $\mathcal{R}^k = \{(v,t) \mid (v,t) \in \mathcal{R} \text{ with } t \in I^k\}$*, for each* $k \in K$*.*

The following theorem shows that a time slicer is a slicer.

Theorem 16. *The time slicer* $\mathfrak{t}_{\varphi,(I^k)_{k\in K}}$ *is a slicer for the formula* φ *if* $\bigcup_{k\in K} I^k = \mathbb{N}$*.*

The remainder of this section is dedicated to proving Theorem 16. We first introduce additional machinery, which we use for the theorem's proof.

Definition 17. *Let* $I \subseteq \mathbb{Z}$ *be an interval and* $c, i \in \mathbb{N}$*. The temporal structures* $(\bar{\mathcal{D}}, \bar{\tau}) \in \mathbf{T}$ *and* $(\bar{\mathcal{D}}', \bar{\tau}') \in \mathbf{T}$ *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* $\tau'_{j'} - \tau_i \in I$*.*

Intuitively, two temporal structures are (I, c, i)-overlapping if their time points (timestamps and structures) are "the same" on an interval of timestamps. This is the case for time slices. The value c here corresponds to the c in Definition 14.

It specifies by how many time points are the two temporal structures “shifted” relatively to each other. The interval I specifies the timestamps for which time points must be “the same”. These are those timestamps whose difference to the timestamp τ_i lies within I.

The following lemmas below (Lemma 18 and Lemma 19) establish that (a) time slices overlap and (b) if temporal structures overlap for an interval I, then they also overlap for other time points in I and for subintervals of I.

Lemma 18. *Let $T \subseteq \mathbb{N}$ and $I \subseteq \mathbb{Z}$ be intervals, $(\bar{\mathcal{D}}, \bar{\tau}) \in \mathbf{T}$, and $(\bar{\mathcal{D}}', \bar{\tau}') \in \mathbf{T}$ 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 $c \in \mathbb{N}$ is the value in Definition 14 used by the function s with respect to $(\bar{\mathcal{D}}, \bar{\tau})$ and its time slice $(\bar{\mathcal{D}}', \bar{\tau}')$.*

Proof. We first show that Condition 1 in Definition 17 is satisfied. For all $i \in \mathbb{N}$ with $\tau_i \in T$ and all $j \in \mathbb{N}$ with $\tau_j - \tau_i \in I$, it holds that $\tau_j \in T \oplus I$. From $c = \min\{k \in \mathbb{N} | \tau_k \in T \oplus I\}$ in Definition 14 it follows that $j \geq c$. Let $j' := j - c$. It also follows from $\tau_j \in T \oplus I$ that $j' \in [0, \ell)$. Therefore, $\mathcal{D}_j = \mathcal{D}_{s(j')} = \mathcal{D}'_{j'} = \mathcal{D}'_{j-c}$ and $\tau_j = \tau_{s(j')} = \tau'_{j'} = \tau'_{j-c}$.

Next, we show that Condition 2 is satisfied. For all $i \in \mathbb{N}$ with $\tau_i \in T$ and all $j' \in \mathbb{N}$ with $\tau'_{j'} - \tau_i \in I$, it holds that $\tau'_{j'} \in T \oplus I$. Since $\tau'_\ell \notin T \oplus I$, it follows that $j' \in [0, \ell)$. Therefore, $\mathcal{D}_{j'+c} = \mathcal{D}_{s(j')} = \mathcal{D}'_{j'}$ and $\tau_{j'+c} = \tau_{s(j')} = \tau'_{j'}$.

Lemma 19. *Let $(\bar{\mathcal{D}}, \bar{\tau}) \in \mathbf{T}$ and $(\bar{\mathcal{D}}', \bar{\tau}') \in \mathbf{T}$ be temporal structures that are (I, c, i)-overlapping, for some $I \subseteq \mathbb{Z}$, $c \in \mathbb{N}$, and $i \in \mathbb{Z}$. Then $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are (K, c, k)-overlapping, for each $k \in \mathbb{N}$ with $\tau_k - \tau_i \in I$ and $K \subseteq \{\tau_i - \tau_k\} \oplus I$.*

Proof. For all $j \in \mathbb{N}$ with $\tau_j - \tau_k \in K$, it follows from $\tau_j - \tau_k \in K$ that $\tau_j - \tau_k + \tau_k - \tau_i \in \{\tau_k - \tau_i\} \oplus K$ and hence $\tau_j - \tau_i \in \{\tau_k - \tau_i\} \oplus K$. From the assumption $K \subseteq \{\tau_i - \tau_k\} \oplus I$, it follows that $\{\tau_k - \tau_i\} \oplus K \subseteq \{\tau_k - \tau_i\} \oplus \{\tau_i - \tau_k\} \oplus I = I$ and hence $\tau_j - \tau_i \in I$. Since $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are (I, c, i)-overlapping, Condition 1 in Definition 17 holds for them to be (K, c, k)-overlapping.

Similarly, for all $j' \in \mathbb{N}$ with $\tau'_{j'} - \tau_k \in K$, it follows that $\tau'_{j'} - \tau_i \in I$ and hence Condition 2 in Definition 17 holds.

The next lemma (Lemma 20) establishes that 0 is included in the relative interval of every formula. This guarantees that the statements in Lemma 21 is always well-defined.

Lemma 20. *For every formula φ, it holds that $0 \in \mathrm{RI}(\varphi)$.*

Proof. We proceed by structural induction on the form of the formula φ. We have the following cases:

- $t \prec t'$, $t \approx t'$, and $r(\bar{t})$, where $t, t', t_1, \ldots, t_{\iota(r)} \in V \cup C$. It follows trivially from Definition 13 that $0 \in \mathrm{RI}(\varphi)$.
- $\neg\psi$ and $\exists x.\, \psi$. It follows from the inductive hypothesis that $0 \in \mathrm{RI}(\psi)$ and hence $0 \in \mathrm{RI}(\varphi)$.
- $\psi \vee \chi$. It follows from the inductive hypothesis that $0 \in \mathrm{RI}(\psi)$ and $0 \in \mathrm{RI}(\chi)$. Therefore, $0 \in \mathrm{RI}(\psi) \Cup \mathrm{RI}(\chi)$ and hence $0 \in \mathrm{RI}(\varphi)$.

– $\CIRCLE_I\,\psi$, $\Circle_I\,\psi$, $\psi\ \mathsf{S}_I\ \chi$, and $\psi\ \mathsf{U}_I\ \chi$. It follows trivially from Definition 13 that $0 \in \mathrm{RI}(\varphi)$.

Lemma 21. *Let φ be a formula and $(\bar{\mathcal{D}}, \bar{\tau}), (\bar{\mathcal{D}}', \bar{\tau}') \in \mathbf{T}$. If $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are $(\mathrm{RI}(\varphi), c, i)$-overlapping, for some c and i, then for all valuations v, it holds that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \varphi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i-c) \models \varphi$.*

Proof. We prove Lemma 21 by structural induction on the form of the formula φ. Note that for all cases, $(\bar{\mathcal{D}}', \bar{\tau}', v, i-c) \models \varphi$ is defined: it follows from Lemma 20 that $0 \in \mathrm{RI}(\varphi)$ and from Condition 1 in Definition 17 that $i \geq c$ and hence $i - c \in \mathbb{N}$. We have the following cases:

– $t \approx t'$, where $t, t' \in V \cup C$. Since the satisfaction of the formula $t \approx t'$ depends only on the valuation v, it follows that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models t \approx t'$ iff $v(t) = v(t')$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i-c) \models t \approx t'$, for all valuations v.
– $t \prec t'$, where $t, t' \in V \cup C$. This case is similar to the previous one.
– $r(\bar{t})$, where $t_1, \dots, t_{\iota(r)} \in V \cup C$. Since $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are $(\mathrm{RI}(r(\bar{t}), c, i)$-overlapping and $0 \in \mathrm{RI}(r(\bar{t}))$, it also follows from Condition 1 in Definition 17 that $\mathcal{D}_i = \mathcal{D}'_{i-c}$ and hence $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models r(\bar{t})$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i-c) \models r(\bar{t})$, for all valuations v.
– $\neg\psi$. $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are $(\mathrm{RI}(\neg\psi), c, i)$-overlapping and $\mathrm{RI}(\neg\psi) = \mathrm{RI}(\psi)$, so by the inductive hypothesis we have that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \psi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i-c) \models \psi$, for all valuations v. It follows that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \neg\psi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i-c) \models \neg\psi$.
– $\psi \vee \chi$. $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are $(\mathrm{RI}(\psi) \Cup \mathrm{RI}(\chi), c, i)$-overlapping. From $\mathrm{RI}(\psi) \subseteq \mathrm{RI}(\psi) \Cup \mathrm{RI}(\chi)$, $\mathrm{RI}(\chi) \subseteq \mathrm{RI}(\psi) \Cup \mathrm{RI}(\chi)$, and Lemma 19 it follows that $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are $(\mathrm{RI}(\psi), c, i)$-overlapping and $(\mathrm{RI}(\psi), c, i)$-overlapping. Then by the inductive hypothesis we know that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \psi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i-c) \models \psi$ and $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \chi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i-c) \models \chi$, for all valuations v. It follows that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \psi \vee \chi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i-c) \models \psi \vee \chi$.
– $\exists x.\, \psi$. From $\mathrm{RI}(\exists x.\, \psi) = \mathrm{RI}(\psi)$ it follows that $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are $(\mathrm{RI}(\psi), c, i)$-overlapping. Then by the inductive hypothesis we know that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \psi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i-c) \models \psi$, for all valuations v. Hence, for all $d \in \mathbb{D}$ we have that $(\bar{\mathcal{D}}, \bar{\tau}, v[x \mapsto d], i) \models \psi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v[x \mapsto d], i-c) \models \psi$. It follows 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.
– $\CIRCLE_{[a,b)}\,\psi$. $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are $(\mathrm{RI}(\CIRCLE_{[a,b)}\,\psi), c, i)$-overlapping, where $\mathrm{RI}(\CIRCLE_{[a,b)}\,\psi) = (-b, 0] \Cup \big((-b, -a] \oplus \mathrm{RI}(\psi)\big)$.
From $0 \in \mathrm{RI}(\CIRCLE_{[a,b)}\,\psi)$ and from Condition 1 in Definition 17 it follows that $i - c \in \mathbb{N}$ and hence $\tau_i = \tau'_{i-c}$.
We make a case split on the value of i. If $i = 0$, then trivially $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \CIRCLE_{[a,b)}\,\psi$, for all valuations v. From Definition 17 it follows that $c = 0$ and hence $i - c = 0$. Trivially, $(\bar{\mathcal{D}}', \bar{\tau}', v, i-c) \not\models \CIRCLE_{[a,b)}\,\psi$, for all valuations v. Next, we consider the case that $i > 0$ and make a case split on whether $\tau_i - \tau_{i-1}$ is included in the interval $[a, b)$.
  1. If $\tau_i - \tau_{i-1} \in [a, b)$, then $\tau_{i-1} - \tau_i \in \mathrm{RI}(\CIRCLE_{[a,b)}\,\psi)$ and from Condition 1 in Definition 17 it follows that $i - 1 \geq c$, $\tau_{i-1} = \tau'_{i-c-1}$, and hence $\tau'_{i-c} - \tau'_{i-c-1} \in [a, b)$. From $\tau_i - \tau_{i-1} \in [a, b)$ it also follows that $\mathrm{RI}(\psi) \subseteq \{\tau_i - \tau_{i-1}\} \oplus \{\tau_{i-1} - \tau_i\} \oplus \mathrm{RI}(\psi) \subseteq \{\tau_i - \tau_{i-1}\} \oplus (-b, -a] \oplus \mathrm{RI}(\psi) \subseteq$

$\{\tau_i - \tau_{i-1}\} \oplus \mathrm{RI}(\newmoon_{[a,b)}\,\psi)$ and hence by Lemma 19 $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are $(\mathrm{RI}(\psi), c, i-1)$-overlapping. By the inductive hypothesis we have that $(\bar{\mathcal{D}}, \bar{\tau}, v, i-1) \models \psi$ 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 $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \newmoon_{[a,b)}\,\psi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i-c) \models \newmoon_{[a,b)}\,\psi$, for all valuations v.

2. If $\tau_i - \tau_{i-1} \notin [a,b)$ then trivially $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \newmoon_{[a,b)}\,\psi$, for all valuations v. From Definition 17 we know that $i \geq c$. We make a case split on whether $i = c$ or $i > c$.
   (a) If $i = c$ then $i - c \not> 0$ and hence $(\bar{\mathcal{D}}', \bar{\tau}', v, i-c) \not\models \newmoon_{[a,b)}\,\psi$, for all valuations v.
   (b) Consider the case $i > c$.
   To achieve a contradiction, suppose that $\tau'_{i-c} - \tau'_{i-c-1} \in [a,b)$. From Condition 2 in Definition 17 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 trivially follows that $(\bar{\mathcal{D}}', \bar{\tau}', v, i-c) \not\models \newmoon_{[a,b)}\,\psi$, for all valuations v.

– $\bigcirc_{[a,b)}\,\psi$. This case is similar to the previous one. In fact, it is simpler because we do not have to consider $i = 0$ and $i - c = 0$ as a special case.
$(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are $(\mathrm{RI}(\bigcirc_{[a,b)}\,\psi), c, i)$-overlapping, where $\mathrm{RI}(\bigcirc_{[a,b)}\,\psi) = [0,b) \Cup \big([a,b) \oplus \mathrm{RI}(\psi)\big)$.
From $0 \in \mathrm{RI}(\bigcirc_{[a,b)}\,\psi)$ and from Condition 1 in Definition 17 it follows that $i - c \in \mathbb{N}$ and hence $\tau_i = \tau'_{i-c}$. We make a case split on whether $\tau_{i+1} - \tau_i$ is included in the interval $[a,b)$.

1. If $\tau_{i+1} - \tau_i \in [a,b)$ then $\tau_{i+1} - \tau_i \in \mathrm{RI}(\bigcirc_{[a,b)}\,\psi)$ and from Condition 1 in Definition 17 it follows that $\tau_{i+1} = \tau'_{i-c+1}$ and hence $\tau'_{i-c+1} - \tau'_{i-c} \in [a,b)$. It also follows from $\tau_{i+1} - \tau_i \in [a,b)$ that $\mathrm{RI}(\psi) \subseteq \{\tau_i - \tau_{i+1}\} \oplus \{\tau_{i+1} - \tau_i\} \oplus \mathrm{RI}(\psi) \subseteq \{\tau_i - \tau_{i+1}\} \oplus [a,b) \oplus \mathrm{RI}(\psi) \subseteq \{\tau_i - \tau_{i+1}\} \oplus \mathrm{RI}(\bigcirc_{[a,b)}\,\psi)$ and hence by Lemma 19 $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are $(\mathrm{RI}(\psi), c, i+1)$-overlapping. By the inductive hypothesis we have that $(\bar{\mathcal{D}}, \bar{\tau}, v, i+1) \models \psi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i-c+1) \models \psi$, for all valuations v. From $\tau_{i+1} - \tau_i \in [a,b)$ iff $\tau'_{i-c+1} - \tau'_{i-c} \in [a,b)$ it follows that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \bigcirc_{[a,b)}\,\psi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i-c) \models \bigcirc_{[a,b)}\,\psi$. for all valuations v.
2. If $\tau_{i+1} - \tau_i \notin [a,b)$ then trivially $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \bigcirc_{[a,b)}\,\psi$, for all valuations v. To achieve a contradiction, suppose that $\tau'_{i-c+1} - \tau'_{i-c} \in [a,b)$. From Condition 2 in Definition 17 it follows that $\tau_{i+1} = \tau'_{i-c+1}$ and hence $\tau_{i+1} - \tau_i = \tau'_{i-c+1} - \tau'_{i-c} \in [a,b)$. This contradicts $\tau_{i+1} - \tau_i \notin [a,b)$, so it must be the case that $\tau'_{i-c+1} - \tau'_{i-c} \notin [a,b)$. It trivially follows that $(\bar{\mathcal{D}}', \bar{\tau}', v, i-c) \not\models \bigcirc_{[a,b)}\,\psi$, for all valuations v.

– $\psi\ \mathsf{S}_{[a,b)}\ \chi$. $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are $(\mathrm{RI}(\psi\ \mathsf{S}_{[a,b)}\ \chi), c, i)$-overlapping, where $\mathrm{RI}(\psi\ \mathsf{S}_{[a,b)}\ \chi) = (-b,0] \Cup \big((-b,0] \oplus \mathrm{RI}(\psi)\big) \Cup \big((-b,-a] \oplus \mathrm{RI}(\chi)\big)$.
Note that $0 \in \mathrm{RI}(\psi\ \mathsf{S}_{[a,b)}\ \chi)$, so from Condition 1 in Definition 17 it follows that $i - c \in \mathbb{N}$ and hence $\tau_i = \tau'_{i-c}$. We show the following two claims, which we use later:

1. For all $j \in \mathbb{N}$ with $j \leq i$ and $\tau_i - \tau_j \in [a,b)$, it holds that $\mathrm{RI}(\chi) \subseteq \{\tau_i - \tau_j\} \oplus \{\tau_j - \tau_i\} \oplus \mathrm{RI}(\chi) \subseteq \{\tau_i - \tau_j\} \oplus (-b,-a] \oplus \mathrm{RI}(\chi) \subseteq \{\tau_i - \tau_j\} \oplus$

RI($\psi\,\mathsf{S}_{[a,b)}\,\chi$) and $j \geq c$. By Lemma 19, $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are $(\mathrm{RI}(\chi), c, j)$-overlapping. It follows from the inductive hypothesis that $(\bar{\mathcal{D}}, \bar{\tau}, v, j) \models \chi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, j-c) \models \chi$, for all valuations v.

2. For all $k \in \mathbb{N}$ with $k \leq i$ and $\tau_i - \tau_k \in [0, b)$, it holds that $\mathrm{RI}(\psi) \subseteq \{\tau_i - \tau_k\} \oplus \{\tau_k - \tau_i\} \oplus \mathrm{RI}(\psi) \subseteq \{\tau_i - \tau_k\} \oplus (-b, 0] \oplus \mathrm{RI}(\psi) \subseteq \{\tau_i - \tau_k\} \oplus \mathrm{RI}(\psi\,\mathsf{S}_{[a,b)}\,\chi)$ and $k \geq c$. By Lemma 19 $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are $(\mathrm{RI}(\psi), c, k)$-overlapping. It follows from the inductive hypothesis that $(\bar{\mathcal{D}}, \bar{\tau}, v, k) \models \psi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, k-c) \models \psi$, for all valuations v.

We show that for all valuations v, 1. $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \psi\,\mathsf{S}_{[a,b)}\,\chi$ implies $(\bar{\mathcal{D}}', \bar{\tau}', v, i-c) \models \psi\,\mathsf{S}_{[a,b)}\,\chi$ and 2. $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \psi\,\mathsf{S}_{[a,b)}\,\chi$ implies $(\bar{\mathcal{D}}', \bar{\tau}', v, i-c) \not\models \psi\,\mathsf{S}_{[a,b)}\,\chi$:

1. If $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \psi\,\mathsf{S}_{[a,b)}\,\chi$ 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 $\tau_j - \tau_i \in \mathrm{RI}(\psi\,\mathsf{S}_{[a,b)}\,\chi)$ and from Condition 1 in Definition 17 we see that $j \geq c$ and $\tau_j = \tau'_{j-c}$. From claim 1 above 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 $\tau'_{k'} - \tau_i \in \mathrm{RI}(\psi\,\mathsf{S}_{[a,b)}\,\chi)$. From Condition 2 in Definition 17 we see that $\tau_{k'+c} = \tau'_{k'}$. From claim 2 above and from $(\bar{\mathcal{D}}, \bar{\tau}, v, k'+c) \models \psi$ it follows that $(\bar{\mathcal{D}}', \bar{\tau}', v, k') \models \psi$. Therefore, $(\bar{\mathcal{D}}', \bar{\tau}', v, i-c) \models \psi\,\mathsf{S}_{[a,b)}\,\chi$.
2. If $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \psi\,\mathsf{S}_{[a,b)}\,\chi$ then there are two possibilities:
   (a) For all $j \leq i$ with $\tau_i - \tau_j \in [a, b)$ it holds that $(\bar{\mathcal{D}}, \bar{\tau}, v, j) \not\models \chi$.
   Then for all $j' \leq i - c$ with $\tau'_{i-c} - \tau'_{j'} = \tau_i - \tau'_{j'} \in [a, b)$, it holds that $\tau'_{j'} - \tau_i \in \mathrm{RI}(\psi\,\mathsf{S}_{[a,b)}\,\chi)$. From Condition 2 in Definition 17 it follows that $\tau'_{j'} = \tau_{j'+c}$. That is, there are no additional time points with a timestamp within the interval $[a, b)$ in $(\bar{\mathcal{D}}', \bar{\tau}')$ that would not be present in $(\bar{\mathcal{D}}, \bar{\tau})$. Since $\tau_i - \tau_{j'+c} \in [a, b)$, it follows from claim 1 above and from $(\bar{\mathcal{D}}, \bar{\tau}, v, j'+c) \not\models \chi$ that $(\bar{\mathcal{D}}', \bar{\tau}', v, j') \not\models \chi$. Therefore, $(\bar{\mathcal{D}}', \bar{\tau}', v, i-c) \not\models \psi\,\mathsf{S}_{[a,b)}\,\chi$.
   (b) 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 $k \in [j+1, i+1)$ and $(\bar{\mathcal{D}}, \bar{\tau}, v, k) \not\models \psi$.
   Then 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 is a j with $j = j' + c$. We show that $\tau'_{j'} = \tau_j$ and $j \leq i$. From $\tau'_{i-c} - \tau'_{j'} \in [a, b)$ and from $\tau'_{i-c} = \tau_i$ it follows that $\tau'_{j'} - \tau_i \in (-b, -a]$ and hence $\tau'_{j'} - \tau_i \in \mathrm{RI}(\psi\,\mathsf{S}_{[a,b)}\,\chi)$. From Condition 2 in Definition 17 it follows that $\tau'_{j'} = \tau_{j'+c} = \tau_j$. From $j = j' + c$ and $j' \leq i - c$ it follows that $j \leq i$.
   Since $\tau'_{j'} = \tau_j$ and $j \leq i$, we can use claim 1 above for j. From claim 1 and from $(\bar{\mathcal{D}}', \bar{\tau}', v, j-c) \models \chi$ it follows that $(\bar{\mathcal{D}}, \bar{\tau}, v, j) \models \chi$. As a consequence, there is a $k \in \mathbb{N}$ with $k \in [j+1, i+1)$ and $(\bar{\mathcal{D}}, \bar{\tau}, v, k) \not\models \psi$. If follows from $k \in [j+1, i+1)$ that $k \leq i$. Furthermore, from $\tau'_{i-c} - \tau'_{j'} \in [a, b)$ it follows that $\tau_i - \tau_j \in [a, b)$ and hence $\tau_i - \tau_k \in [0, b)$. Therefore, we can use claim 2 above for k. From claim 2 and from $(\bar{\mathcal{D}}, \bar{\tau}, v, k) \not\models \psi$ it follows that $(\bar{\mathcal{D}}', \bar{\tau}', v, k-c) \not\models \psi$. From $k \in [j+1, i+1)$ it follows that $k - c \in [j'+1, i-c+1)$ and hence $(\bar{\mathcal{D}}', \bar{\tau}', v, i-c) \not\models \psi\,\mathsf{S}_{[a,b)}\,\chi$.

From 1. and 2. it follows that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \psi\,\mathsf{S}_{[a,b)}\,\chi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i-c) \models \psi\,\mathsf{S}_{[a,b)}\,\chi$, for all valuations v.

– $\psi \mathsf{U}_{[a,b)} \chi$. This case is analogous to the previous one.

We prove Theorem 16 by showing that a time slicer $\mathfrak{t}_{\varphi,(I^k)_{k\in K}}$ satisfies the conditions (S1) to (S3) from Definition 3 if $\bigcup_{k\in K} I^k = \mathbb{N}$. For (S1), we need to show that $\mathcal{R} = \bigcup_{k\in K} \mathcal{R}^k$. This follows from the definition of $\mathcal{R}^k$ and the assumption that $\bigcup_{k\in K} I^k = \mathbb{N}$. (S2) and (S3) follow from the Lemmas 18 and 21.

B.5 Filtering Empty Time Points

In the following, we introduce a filter that removes empty time points. A time point $i \in \mathbb{N}$ is *empty* in the temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$ if $r^{\mathcal{D}_i} = \emptyset$, for every predicate symbol r, and *nonempty* otherwise. Removing empty time points can substantially speed up monitoring. In particular, after removing tuples from relations in a temporal structure via a data slicer, the slices might contain many empty time points. However, removing empty time points from a temporal structure is not guaranteed to be sound and complete for a formula. We identify a fragment for which it is safe of removing such time points.

Definition 22. *The temporal structure* $(\bar{\mathcal{D}}', \bar{\tau}') \in \mathbf{T}$ *is the* empty-time-point-filtered slice *of* $(\bar{\mathcal{D}}, \bar{\tau}) \in \mathbf{T}$ *if* $(\bar{\mathcal{D}}', \bar{\tau}')$ *is a time slice of* $(\bar{\mathcal{D}}, \bar{\tau})$*, where* $\ell = \infty$ *and* $s : [0, \ell) \to \mathbb{N}$ *satisfies the following conditions.*

- *If* $(\bar{\mathcal{D}}, \bar{\tau})$ *contains finitely many nonempty time points then* s *is the identity function.*
- *Otherwise,* s *is the monotonically increasing injective function such that* $i \notin \{s(i') \in \mathbb{N} \mid i' \in \mathbb{N}\}$ *iff* i *is an empty time point in* $(\bar{\mathcal{D}}, \bar{\tau})$*, for every* $i \in \mathbb{N}$.

Note that the function s in Definition 22 is uniquely determined in both cases. We make a case distinction in the definition because if there are only finitely many nonempty time points, then removing all the empty time points would result in a finite "temporal structure," but temporal structures are by definition infinite sequences. In practice, we always monitor only a finite prefix of a temporal structure from which we remove the empty time points. We assume here that there are infinitely many nonempty time points in the temporal structure's suffix.

Definition 23. *The* function $\mathfrak{f}'_\varphi$ *for the formula* φ *maps* $(\bar{\mathcal{D}}, \bar{\tau}) \in \mathbf{T}$ *and* $\mathcal{R} \in \mathbf{R}$ *to a family that contains only the temporal structure* $(\bar{\mathcal{D}}', \bar{\tau}')$ *and a family that contains only the restriction* $\mathcal{R}$*, where* $(\bar{\mathcal{D}}', \bar{\tau}')$ *is the empty-time-point-filtered slice of* $(\bar{\mathcal{D}}, \bar{\tau})$.

Next, we present a fragment of formulas for which the empty-time-point-filtered slice is sound and complete with respect to the original temporal structure. See Theorem 26. To define the fragment, we use the sets FT, FF, and FE (Definition 24). Membership of a formula in these sets reflects whether the formula is satisfied at an empty time point. In a nutshell, at an empty time point, a formula in the set FF is not satisfied, a formula in the set FT is satisfied, and the satisfaction of a formula in the set FE is not affected by adding or removing empty time points in the temporal structure.

$$\frac{}{r(\bar{t}) : \mathsf{FF}} \quad \frac{}{\mathit{true} : \mathsf{FT}} \quad \frac{\varphi : \mathsf{FF}}{\neg\varphi : \mathsf{FT}} \quad \frac{\varphi : \mathsf{FT}}{\neg\varphi : \mathsf{FF}}$$

$$\frac{\varphi : \mathsf{FT}}{\varphi \lor \psi : \mathsf{FT}} \quad \frac{\psi : \mathsf{FT}}{\varphi \lor \psi : \mathsf{FT}} \quad \frac{\varphi : \mathsf{FF} \quad \psi : \mathsf{FF}}{\varphi \lor \psi : \mathsf{FF}} \quad \frac{\varphi : \mathsf{FT}}{\exists y.\, \varphi : \mathsf{FT}} \quad \frac{\varphi : \mathsf{FF}}{\exists y.\, \varphi : \mathsf{FF}}$$

$$\frac{}{r(\bar{t}) : \mathsf{FE}} \quad \frac{}{t \approx t' : \mathsf{FE}} \quad \frac{}{t \prec t' : \mathsf{FE}} \quad \frac{\varphi : \mathsf{FE}}{\neg\varphi : \mathsf{FE}} \quad \frac{\varphi : \mathsf{FE}}{\exists x.\, \varphi : \mathsf{FE}} \quad \frac{\varphi : \mathsf{FE} \quad \psi : \mathsf{FE}}{\varphi \lor \psi : \mathsf{FE}}$$

$$\frac{\varphi : \mathsf{FE} \quad \varphi : \mathsf{FT} \quad \psi : \mathsf{FE} \quad \psi : \mathsf{FF}}{\varphi \,\mathsf{S}_I\, \psi : \mathsf{FE}} \qquad \frac{\varphi : \mathsf{FE} \quad \varphi : \mathsf{FT} \quad \psi : \mathsf{FE} \quad \psi : \mathsf{FF}}{\varphi \,\mathsf{U}_I\, \psi : \mathsf{FE}}$$

$$\frac{\varphi : \mathsf{FE} \quad \varphi : \mathsf{FF}}{\blacklozenge_I \lozenge_J \varphi : \mathsf{FE}}\ 0 \in I \cap J \qquad \frac{\varphi : \mathsf{FE} \quad \varphi : \mathsf{FF}}{\lozenge_I \blacklozenge_J \varphi : \mathsf{FE}}\ 0 \in I \cap J$$

$$\frac{\varphi : \mathsf{FE} \quad \varphi : \mathsf{FT}}{\blacksquare_I \square_J \varphi : \mathsf{FE}}\ 0 \in I \cap J \qquad \frac{\varphi : \mathsf{FE} \quad \varphi : \mathsf{FT}}{\square_I \blacksquare_J \varphi : \mathsf{FE}}\ 0 \in I \cap J$$

Fig. 4: Labeling rules (empty-time-point filter).

Definition 24. *The sets* FT*,* FF*, and* FE *of formulas are defined as follows.*

- $\varphi \in$ FT *iff* $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \varphi$*, for all* $(\bar{\mathcal{D}}, \bar{\tau}) \in \mathbf{T}$*, all valuations* v*, and all empty time points* i *of* $(\bar{\mathcal{D}}, \bar{\tau})$*.*
- $\varphi \in$ FF *iff* $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \varphi$*, for all* $(\bar{\mathcal{D}}, \bar{\tau}) \in \mathbf{T}$*, all valuations* v*, and all empty time points* i *of* $(\bar{\mathcal{D}}, \bar{\tau})$*.*
- $\varphi \in$ FE *iff the equivalence*

$$(\bar{\mathcal{D}}', \bar{\tau}', v, i') \models \varphi \quad \textit{iff} \quad (\bar{\mathcal{D}}, \bar{\tau}, v, s(i')) \models \varphi$$

holds, for all $(\bar{\mathcal{D}}, \bar{\tau}), (\bar{\mathcal{D}}', \bar{\tau}') \in \mathbf{T}$*, all valuations* v*, and all nonempty time points* i' *of* $(\bar{\mathcal{D}}', \bar{\tau}')$*, where* $(\bar{\mathcal{D}}', \bar{\tau}')$ *is the empty-time-point-filtered slice of* $(\bar{\mathcal{D}}, \bar{\tau})$ *and* s *is the function used in the filtering of* $(\bar{\mathcal{D}}, \bar{\tau})$*.*

We approximate membership in the sets FT, FF, and FE with syntactic fragments, since these sets are undecidable, which follows from the undecidability of the satisfiability problem of MFOTL. The fragments are defined in terms of a labeling algorithm that assigns the labels FT, FF, and FE to formulas. The fragments are sound in the sense that if a formula is assigned to a label (FT, FF, FE) then the formula is in the corresponding set (FT, FF, FE, respectively). However, the fragments are incomplete, i.e., not every formula in one of the sets is assigned by the algorithm to the corresponding label. The algorithm labels the atomic subformulas of a formula and propagates the labels bottom-up to the formula's root. The labeling rules are shown in Figure 4, where we use the expression $\varphi : \ell$ to denote that the formula φ is labeled with ℓ. Note that a formula can have multiple labels. We show the soundness of the labeling rules in Theorem 25.

Theorem 25. *For all formulas* φ*, if the derivation rules shown in Figure 4 assign the label* FT*,* FF*, or* FE *to* φ *then* φ *is in the set* FT*,* FF*, or* FE*, respectively.*

Theorem 26. *The empty-time-point filter $\mathfrak{f}'_\varphi$ is a slicer for the formula φ, if the formula φ is in both* FE *and* FT*.*

It follows that the empty-time-point filter is a slicer for all formulas that can be labeled with FE and FT.

In the remainder of this subsection, we provide the proof details for the assertions made about the empty-time-point filter in the Theorems 25 and 26. We first provide the details for Theorem 25. We begin with the labels FT and FF. We proceed by induction on the size of the derivation tree assigning label ℓ to the formula φ. We make a case distinction based on the rules applied to label the formula, that is, the rule at the tree's root. However, for clarity, we generally group cases by the formula's form. For readability, and without loss of generality, we fix the temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$, a time point $i \in \mathbb{N}$, and a valuation v.

A formula $r(\bar{t})$ is labeled FF. If i is an empty time point in $(\bar{\mathcal{D}}, \bar{\tau})$ then clearly $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models r(\bar{t})$, for any predicate symbol $r \in R$ and any terms $\bar{t}$. The formula *true* is labeled FT. Trivially, $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \mathit{true}$. The other rules propagate the assigned label of the subformulas through the non-temporal connectives according to their semantics. The rules' correctness is straightforward.

Next, we prove Theorem 25 for the label FE. Again, we proceed by induction on the size of the derivation tree assigning label FE to formula φ. We make a case distinction based on the rules applied to label the formula, that is, the rule at the tree's root. However, for clarity, we generally group cases by the formula's form.

For every valuation v and $i' \in \mathbb{N}$, the evaluation of the formulas $r(\bar{t})$, $t \approx t'$, and $t \prec t'$ only depends on the current time point and hence they are in FE. The other rules not involving temporal operators depend only on the value of their subformulas at the current time point. If the subformulas are labeled with FE, then by the induction hypothesis the subformulas are in FE, so the formula is also in FE.

For readability, and without loss of generality, we already fix the temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$ and its empty-time-point-filtered slice $(\bar{\mathcal{D}}', \bar{\tau}')$. The proof is trivial for the case where s is the identity function. In the rest of the proof, we assume that $(\bar{\mathcal{D}}, \bar{\tau})$ has infinitely many nonempty time points and hence s is not the identity function.

For the remaining rules we show separately that, for every valuation v and $i' \in \mathbb{N}$,

1. $(\bar{\mathcal{D}}', \bar{\tau}', v, i') \models \varphi$ implies $(\bar{\mathcal{D}}, \bar{\tau}, v, s(i')) \models \varphi$, and
2. $(\bar{\mathcal{D}}, \bar{\tau}, v, s(i')) \models \varphi$ implies $(\bar{\mathcal{D}}', \bar{\tau}', v, i') \models \varphi$

– $\varphi \, \mathsf{S}_I \, \psi$:
  1. $(\bar{\mathcal{D}}', \bar{\tau}', v, i') \models \varphi \, \mathsf{S}_I \, \psi$ implies $(\bar{\mathcal{D}}, \bar{\tau}, v, s(i')) \models \varphi \, \mathsf{S}_I \, \psi$
     From $(\bar{\mathcal{D}}', \bar{\tau}', v, i') \models \varphi \, \mathsf{S}_I \, \psi$ we know that there is a $j' \leq i'$ such that $\tau'_{i'} - \tau'_{j'} \in I$ and $(\bar{\mathcal{D}}', \bar{\tau}', v, j') \models \psi$ and, for every k' with $j' < k' \leq i'$, we have that $(\bar{\mathcal{D}}', \bar{\tau}', v, k') \models \varphi$.
     Since ψ is labeled FE, it follows from the induction hypothesis that ψ is in FE and hence $(\bar{\mathcal{D}}, \bar{\tau}, v, s(j')) \models \psi$. For each k with $s(j') < k \leq s(i')$

either k is an empty or a nonempty time point in $(\bar{\mathcal{D}}, \bar{\tau})$. If it is an empty time point then from φ being labeled FT and hence in FT we know that $(\bar{\mathcal{D}}, \bar{\tau}, v, k) \models \varphi$. If it is a nonempty time point then we know that there is a time point k' in $(\bar{\mathcal{D}}', \bar{\tau}')$ with $j' < k' \leq i'$ and $k = s(k')$. From φ being labeled FE and hence in FE we know that $(\bar{\mathcal{D}}, \bar{\tau}, v, k) \models \varphi$. In both cases $(\bar{\mathcal{D}}, \bar{\tau}, v, k) \models \varphi$ and therefore $(\bar{\mathcal{D}}, \bar{\tau}, v, s(i')) \models \varphi \,\mathsf{S}_I\, \psi$.

2. $(\bar{\mathcal{D}}, \bar{\tau}, v, s(i')) \models \varphi \,\mathsf{S}_I\, \psi$ implies $(\bar{\mathcal{D}}', \bar{\tau}', v, i') \models \varphi \,\mathsf{S}_I\, \psi$
From $(\bar{\mathcal{D}}, \bar{\tau}, v, s(i')) \models \varphi \,\mathsf{S}_I\, \psi$ it follows that there is a $j \leq s(i')$ with $\tau_{s(i')} - \tau_j \in I$ and $(\bar{\mathcal{D}}, \bar{\tau}, v, j) \models \psi$, and that, for every k with $j < k \leq s(i')$, we have that $(\bar{\mathcal{D}}, \bar{\tau}, v, k) \models \varphi$.
Since $(\bar{\mathcal{D}}, \bar{\tau}, v, j) \models \psi$ and ψ is labeled FF, so that ψ is in FF, we know that j cannot be an empty time point in $(\bar{\mathcal{D}}, \bar{\tau})$. Therefore, there is a j' such that $j = s(j')$. We have that $j' \leq i'$ because s is monotonically increasing. From ψ being labeled FE it follows that ψ is in FE and hence $(\bar{\mathcal{D}}, \bar{\tau}, v, j) \models \psi$ implies $(\bar{\mathcal{D}}', \bar{\tau}', v, j') \models \psi$.
Furthermore, for every k' with $j' < k' \leq i'$ there is a corresponding time point k in $(\bar{\mathcal{D}}, \bar{\tau})$ such that $k = s(k')$. As s is a monotonously increasing function we have that $s(j') < k \leq s(i')$. From $(\bar{\mathcal{D}}, \bar{\tau}, v, s(i')) \models \varphi \,\mathsf{S}_I\, \psi$ it follows that $(\bar{\mathcal{D}}, \bar{\tau}, v, k) \models \varphi$. From φ being labeled FE it follows that φ is in FE and hence $(\bar{\mathcal{D}}', \bar{\tau}', v, k') \models \varphi$. Therefore, $(\bar{\mathcal{D}}, \bar{\tau}, v, s(i')) \models \varphi \,\mathsf{S}_I\, \psi$.

- $\varphi \,\mathsf{U}_I\, \psi$: This case is similar to $\varphi \,\mathsf{S}_I\, \psi$.
- $\Diamond_I \blacklozenge_J \varphi$ and $\blacklozenge_I \Diamond_J \varphi$ with $0 \in I \cap J$:
Note that this formula can be rewritten to $\Diamond_I \varphi \vee \blacklozenge_J \varphi$, which can be labeled with the rules proven above.
- $\Box_I \blacksquare_J \varphi$ and $\blacksquare_J \Box_I \varphi$ with $0 \in I \cap J$:
Note that this formula can be rewritten to $\blacksquare_J \varphi \wedge \Box_I \varphi$, which can be labeled with the rules proven above.

Finally, we prove Theorem 26. We first state a lemma, which we use for the theorem's proof.

Lemma 27. *Let φ be a formula in the intersection of* FE *and* FT*, $(\bar{\mathcal{D}}, \bar{\tau}) \in \mathbf{T}$, and $(\bar{\mathcal{D}}', \bar{\tau}') \in \mathbf{T}$ the empty-time-point-filtered slice of $(\bar{\mathcal{D}}, \bar{\tau})$. $(\bar{\mathcal{D}}', \bar{\tau}')$ is $\mathcal{R}_0$-sound and $\mathcal{R}_0$-complete for $(\bar{\mathcal{D}}, \bar{\tau}, v, 0)$ and φ.*

Proof. We first show soundness. That is, for all valuations v and timestamps $t \in \mathbb{N}$, it holds that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \varphi$, for all $i \in \mathbb{N}$ with $\tau_i = t$, implies $(\bar{\mathcal{D}}', \bar{\tau}', v, i') \models \varphi$, for all $i' \in \mathbb{N}$ with $\tau'_{i'} = t$. We first show that $(\bar{\mathcal{D}}, \bar{\tau}, v, 0) \models \Box \varphi$ implies $(\bar{\mathcal{D}}', \bar{\tau}', v, 0) \models \Box \varphi$. From $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \varphi$ it follows that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \varphi$, for all i.

As s is a function we know that for all time points i' in $(\bar{\mathcal{D}}', \bar{\tau}')$ there is a time point i in $(\bar{\mathcal{D}}, \bar{\tau})$ such that $i = s(i')$ and $\tau_i = \tau'_{i'}$. From φ being in FE and from $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \varphi$ it follows that $(\bar{\mathcal{D}}', \bar{\tau}', v, i') \models \varphi$.

We continue by showing completeness. That is, for all valuations v and timestamps $t \in \mathbb{N}$, it holds that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \varphi$, for some $i \in \mathbb{N}$ with $\tau_i = t$, implies $(\bar{\mathcal{D}}', \bar{\tau}', v, i') \not\models \varphi$, for some $i' \in \mathbb{N}$ with $\tau'_{i'} = t$.

Each time point i in $(\bar{\mathcal{D}}, \bar{\tau})$ is either empty or nonempty. If it is empty, then from φ being in FT we know that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \varphi$. If it is nonempty then there

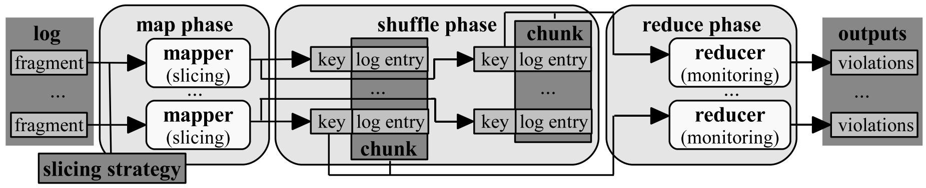

Fig. 5: Scalable offline monitoring with MapReduce.

exists a time point i' in $(\bar{\mathcal{D}}', \bar{\tau}')$ such that $i = s(i')$ and $\tau_i = \tau'_{i'}$. From φ being in FE and from $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \varphi$ it follows that $(\bar{\mathcal{D}}', \bar{\tau}', v, i') \not\models \varphi$.

We prove Theorem 26 by showing that the empty-time-point filter satisfies the conditions (S1) to (S3) from Definition 3. (S1) follows trivially because the filter does not modify the restriction $\mathcal{R}$. (S2) and (S3) follow directly from Lemma 27.

C Additional Details: Parallelization

In the map phase, instances of our map function are executed in parallel. In the shuffle phase, the reorganization of log entries into chunks is parallelized. In the reduce phase, instances of our reduce function are executed in parallel. Figure 5 visualizes our parallelization with MapReduce. In the following, we provide the details of our algorithmic realization.

C.1 Algorithms for Map and Reduce

Map Phase. MapReduce requires an implementation of a *map function* taking two arguments, a key and a value. This function is called by the so-called *mappers* of MapReduce. Algorithm 1 realizes our map function that takes as arguments an identifier for the log fragment (*key*) and the log fragment itself (*value*). The *slicing strategy* object *slicingStrategy* used in Algorithm 1 is generated for each call of the map function by copying the slicing strategy that we provide to MapReduce. A slicing strategy object stores the formula to be monitored (accessible by the selector *getFormula()*), the initial restriction (accessible by the selector *getRestriction()*), and a set of configurations (accessible by the selector *getConfigs()*). A *configuration* is a pair (fs, fr) where $fs : (\mathbf{D} \times \mathbb{N} \times \mathbf{R}) \to (\mathbf{D} \cup \{\bot\})$ is a *slicing function* and $fr : \mathbf{R} \to \mathbf{R}$ is a *restriction modifier*. We use $\mathbf{C}$ to denote the set of all configurations and present concrete algorithms for configurations in Section C.2. $\mathbf{D}$ denotes the set of all the structures in the temporal structures in $\mathbf{T}$.

The map function iterates over all log entries in the given log fragment. For each log entry $(\mathcal{D}, \tau)$ and each slicing function fs, a sliced log entry $(\mathcal{D}', \tau)$ is computed. Note that the inner loop only iterates over slicing functions that are *applicable* in the sense that $fs(\mathcal{D}, \tau, \mathcal{R}) \neq \bot$ holds. Our algorithm leaves it underspecified how the applicable slicing functions fs are determined, thus,

Algorithm 1: *map(key, value)*

$\mathcal{C} \leftarrow$ *slicingStrategy.getConfigs( )*
$\mathcal{R} \leftarrow$ *slicingStrategy.getRestriction( )*
foreach $(\mathcal{D}, \tau)$ **in** *value* **do**
  foreach $(fs, fr) \in \{(fs', fr') \in \mathcal{C} \mid fs'(\mathcal{D}, \tau, \mathcal{R}) \neq \bot\}$ **do**
    emitIntermediate($((fs, fr), \tau)$, $(fs(\mathcal{D}, \tau, \mathcal{R}), \tau)$)

leaving a design space for different implementations of map. Possible solutions range from simply iterating over all slicing functions in the slicing strategy, which can be inefficient, to the use of data structures that support an efficient look up of applicable slicing functions.

Our map function emits key-value pairs where each value is a log entry $(\mathcal{D}', \tau)$ and each key consists of a primary key and a secondary key. Both parts of a key are used in the shuffle phase of MapReduce: The primary key determines the chunk into which a key-value pair is placed, while the secondary key is used for sorting key-value pairs within a chunk. Our map function attaches to each sliced log entry $(\mathcal{D}', \tau)$ the configuration (fs, fr) used for computing $\mathcal{D}'$ as primary key and the timestamp τ as secondary key. When implementing the map function, one might prefer keys that require less space. For instance, instead of using a configuration as primary key, one could use identifiers, that allow one to retrieve the configuration using efficient data structures, as primary keys.

Shuffle Phase. In the shuffle phase, MapReduce transforms the key-value pairs resulting from the map phase into pairs consisting of a key and a chunk of log entries. Such a pair is constructed for each primary key k generated in the map phase. All log entries from key-value pairs with primary key k are combined to one chunk of log entries. As mentioned before, the log entries within a chunk are sorted based on their secondary keys. Since shuffle is built-in into MapReduce, we do not provide an algorithm here.

Reduce Phase. MapReduce requires an implementation of a *reduce function* taking two arguments, a key and a list of values. This function is called by the so-called *reducers* of MapReduce. Algorithm 2 realizes our reduce function that takes *key* $\in (\mathbf{C} \times \mathbb{N})$, identifying the chunk to be monitored, and *values* $\in (\mathbf{D} \times \mathbb{N})^*$, the chunk itself, as arguments. The formula specifying the policy to be checked by the monitor and the restriction are both retrieved from a slicing strategy object that is generated for each call of reduce.

Our reduce function collapses all log entries with identical timestamps. The method *collapse* realizes this functionality by iterating over the list *values* while merging two adjacent log entries $(\mathcal{D}_1, \tau)$ and $(\mathcal{D}_2, \tau)$ with identical timestamp τ to the log entry $(\mathcal{D}, \tau)$ where $r^{\mathcal{D}} = r^{\mathcal{D}_1} \cup r^{\mathcal{D}_2}$ holds for each $r \in R$. The sorting of log entries in the shuffle phase ensures that *collapse* only needs to inspect adjacent elements in the list *values*. After collapsing, each timestamp occurs at

Algorithm 2: *reduce(key, values)*

$\varphi \leftarrow$ *slicingStrategy.getFormula()*
$\mathcal{R} \leftarrow$ *slicingStrategy.getRestriction()*
$((\mathit{fs}, \mathit{fr}), \tau) \leftarrow$ *key*
emit(mon(φ*, collapse(values))* $\cap$ *fr(*$\mathcal{R}$*))*

most once in the resulting list of log entries. The reduce function forwards the collapsed list to the monitoring algorithm *mon* together with the formula φ to be monitored. In the set of violations returned by the monitor, the reduce function removes all violations that are not permitted by *fr(*$\mathcal{R}$*)* and emits the resulting set of violations. The set of all violations of φ equals the union of the sets emitted by the individual reducers. In our implementation described in Section 4, we use the monitoring tool MONPOLY as monitoring algorithm *mon*.

C.2 Algorithms for Slicing Functions and Restriction Modifiers

Algorithm 3 describes the pointwise slicing function $\mathit{fs}_{d,\varphi,x,S}$ and the restriction modifier $\mathit{fr}_{d,\varphi,x,S}$ for data slicing. For brevity, we omit the subscripts for the formula φ, the slicing variable x, and the slicing set S when they are clear from the context or unimportant. The body of fs_d iterates over all atomic subformulas $r(t_1, \ldots, t_{\iota(r)})$ of φ, and each tuple $(a_1, \ldots, a_{\iota(r)})$ in $r^{\mathcal{D}}$ where $\mathcal{D}$ is the structure supplied to fs_d. The while loop and the subsequent conditional update iteratively construct the structure $\mathcal{D}'$ starting from the initial structure $\mathcal{D}_\emptyset$ with empty relations (i.e., $r^{\mathcal{D}_\emptyset} = \emptyset$ for each $r \in R$). Note that the condition on $a_1, \ldots, a_{\iota(r)}$ checked by the entire while loop corresponds to the condition in our definition of (φ, x, S)-slices (see Definition 7). The restriction modifier $\mathit{fr}_{d,\varphi,x,S}$ removes all violations that do not fit S from a given restriction $\mathcal{R}$.

Algorithm 4 describes the pointwise slicing function $\mathit{fs}_{t,\varphi,I}$ and the restriction modifier $\mathit{fr}_{t,\varphi,I}$ for time slicing. The body of $\mathit{fs}_{t,\varphi,I}$ first determines whether the timestamp τ is within the time interval $I \oplus \mathrm{RI}(\varphi)$, where $\mathrm{RI}(\varphi)$ is the relative interval of φ (see Definition 13) and $I \oplus \mathrm{RI}(\varphi)$ is the interval containing each timestamp that results from the addition of some timestamp in I with some timestamp in $\mathrm{RI}(\varphi)$. Note that this check corresponds to the condition on timestamps in our definition of a T-slice with $T = I \oplus \mathrm{RI}(\varphi)$ (see Definition 14). If τ is within the computed interval then $\mathit{fs}_{t,\varphi,I}$ returns $\mathcal{D}$ unmodified and, otherwise, returns $\perp$ to indicate that the log entry shall be deleted. The restriction modifier $\mathit{fr}_{t,\varphi,I}$ removes all violations with timestamps outside I from a given restriction $\mathcal{R}$.

Algorithm 5 describes the pointwise slicing function fs_f and the restriction modifier fr_f for filtering empty time points. The function fs_f returns $\mathcal{D}$ if there is at least one $r \in R$ for which $r^{\mathcal{D}}$ is nonempty and, otherwise, returns $\perp$ to indicate that the time point shall be deleted. Note that the check $\{r \in R \mid r^{\mathcal{D}} \neq \emptyset\} \neq \emptyset$ in the body of fs_f corresponds to the condition for nonempty time points introduced in Section 3.2. For efficiency, one should avoid an explicit construction of the

Algorithm 3: *DataSlicing*

method $fs_{d,\varphi,x,S}(\mathcal{D}, \tau, \mathcal{R})$ **is**
  $\mathcal{D}' \leftarrow \mathcal{D}_\emptyset$
  $\Psi \leftarrow$ *atomicSubformulas*(φ)
  foreach $r(t_1, \dots, t_{\iota(r)}) \in \Psi$ **do**
    foreach $(a_1, \dots, a_{\iota(r)}) \in r^{\mathcal{D}}$ **do**
      $i \leftarrow 1$; $b \leftarrow$ *false*
      while $(i \leq \iota(r)) \wedge \neg b$ **do**
        $b \leftarrow ((t_i = x \wedge a_i \notin S) \vee (t_i \in C \wedge a_i \neq t_i^{\mathcal{D}}))$
        $i \leftarrow i + 1$
      if $\neg b$ **then** $r^{\mathcal{D}'} \leftarrow r^{\mathcal{D}'} \cup \{(a_1, \dots, a_{\iota(r)})\}$
  return $\mathcal{D}'$
method $fr_{d,\varphi,x,S}(\mathcal{R})$ **is**
  return $\{(v,t) \in \mathcal{R} \mid v(x) \in S\}$

Algorithm 4: *TimeSlicing*

method $fs_{t,\varphi,I}(\mathcal{D}, \tau, \mathcal{R})$ **is**
  if $\tau \in I \oplus RI(\varphi)$ **then**
    return $\mathcal{D}$
  else
    return $\perp$
method $fr_{t,\varphi,I}(\mathcal{R})$ **is**
  return $\{(v,t) \in \mathcal{R} \mid t \in I\}$

set $\{r \in R \mid r^{\mathcal{D}} \neq \emptyset\}$ when implementing this check. The restriction modifier fr_f returns $\mathcal{R}$ without modifications.

Algorithm 6 describes the pointwise slicing function and the restriction modifier resulting from the composition of two configurations (fs, fr) and (fs', fr'). In the body of $fs_{c,(fs,fr),(fs',fr')}$, the slicing function of the first configuration is applied before the slicing function of the second configuration. If fs returns $\perp$ then fs' is not applied, but $\perp$ is returned directly. If fs returns a structure $\mathcal{D}'$ then $fs'(\mathcal{D}', \tau, fr(\mathcal{R}))$ is returned, which either is a structure or $\perp$. The restriction modifier $fr_{c,(fs,fr),(fs',fr')}$ first applies fr and then fr' to the given restriction. Overall, Algorithm 6 realizes a pointwise composition of slicing similar to the composition of slicers in Section 3.

To ease the presentation, we have taken the liberty to describe data slicing, temporal slicing, filtering, and composed slicing by pairs of functions of the form (fs, fr) that constitute specific configurations. In an object-oriented language, one would instead define an interface Config that declares two methods *slice* and *restrict* and a class that implements Config for each of data slicing, temporal slicing, filtering, and composed slicing. The implementation of the methods *slice* and *restrict*, for instance, in the class for data slicing would then realize the

Algorithm 5: *EmptyTimePointFiltering*

method $fs_f(\mathcal{D}, \tau, \mathcal{R})$ **is**
  if $\{r \in R \mid r^{\mathcal{D}} \neq \emptyset\} \neq \emptyset$ **then return** $\mathcal{D}$
  else return $\perp$
method $fr_f(\mathcal{R})$ **is**
  return $\mathcal{R}$

Algorithm 6: *ComposedSlicing*

method $fs_{c,(fs,fr),(fs',fr')}(\mathcal{D}, \tau, \mathcal{R})$ **is**
  $\mathcal{D}' \leftarrow fs(\mathcal{D}, \tau, \mathcal{R})$
  if $\mathcal{D}' \neq \perp$ **then return** $fs'(\mathcal{D}', \tau, fr(\mathcal{R}))$
  else return $\perp$
method $fr_{c,(fs,fr),(fs',fr')}(\mathcal{R})$ **is**
  return $fr'(fr(\mathcal{R}))$

functions $fs_{d,\varphi,x,S}$ and $fr_{d,\varphi,x,S}$ respectively. The formula φ, the slicing variable x, and the slicing set S would not be passed to *slice* and *restrict*, but rather as arguments to the constructor when creating objects.