Technical Report 791, ETH Zurich, Department of Computer Science
July 22, 2013

Checking System Compliance by Slicing and Monitoring Logs*

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

[1] Institute of Information Security, ETH Zurich
[2] Google Inc.
[3] MAIS, Computer Science, TU Darmstadt

Abstract. It is a growing concern of companies and end users whether the agents of an IT system, i.e., its processes and users, comply with security policies, which, e.g., stipulate how sensitive data must and must not be used by the agents. We present a scalable solution for compliance checking based on monitoring the agents' behavior, where policies are specified in an expressive temporal logic and the system actions are logged. In particular, our solution utilizes the MapReduce framework to parallelize the process of monitoring the logged actions. We also provide the theoretical underpinnings of our solution as a theoretical framework for slicing logs, i.e., the reorganization of the logged actions into parts that can be analyzed independently of each other. We present orthogonal methods for generating such slices and provide means to combine these methods. Finally, we report on a real-world case study, which demonstrates the feasibility and the scalability of our monitoring solution.

1 Introduction

Both public and private companies are increasingly required to check whether the agents of their IT systems, i.e., users and processes, comply with security policies, which, e.g., specify permissive and obligatory actions for agents and stipulate the usage of sensitive data. For example, US hospitals must follow the US Health Insurance Portability and Accountability Act (HIPAA) [1] and financial services must conform to the Sarbanes-Oxley Act (SOX) [2]. For end users, it is also a growing concern whether their private data collected by and shared within IT systems is used only in the intended way. A prominent approach to checking system compliance is based on monitoring the actions of the users and processes. Either the actions are checked online by a monitor, which reports violations when an action does not comply with a policy, or the observed actions are logged and a monitor checks the logs offline at a later time, such as during an audit.

Although various monitoring approaches have been developed for different policy specification languages, see, e.g., [10, 18, 19, 22, 30], they fall short for checking compliance of larger IT systems like cloud-based services and systems

* This work was supported by Google Inc.

that process machine-generated data. The monitors in these approaches do not cope with the enormous amount of logged system actions: these systems typically log terabytes or even petabytes of actions each day.

In this paper, we provide a scalable solution for offline monitoring, where the system components log their actions and monitors inspect the logged actions for detecting and reporting policy violations. An overview of our solution is as follows: For a given policy, we reorganize the logged actions into small parts, called *slices*. We show that each slice can be meaningfully monitored independently of the other slices. This allows us to parallelize and distribute the monitoring process over multiple computers.

In general, it is not obvious how to reorganize the logged actions into slices such that the slices can be analyzed independently. The slices must be *sound* and *complete* for the given policy and the logged data, that is, when monitoring these slices only valid violations are reported and every violation is reported by at least one of the monitors. We lay the foundations for obtaining slices with these properties. In particular, we provide a theoretical slicing framework, where logs are represented as temporal structures and policies are specified as formulas in metric first-order temporal logic (MFOTL) [9, 10]. Although we use temporal structures and MFOTL, the underlying principles of our slicing framework are general and apply also to other representations of logs and policy specification languages.

We present two basic orthogonal methods to generate sound a complete slices. With the first method, a slice is used to check system compliance during a specified period of time, e.g., a particular week. Based on the timestamps that record when an action is performed, the slicing method ensures that a slice contains those actions that are needed to check compliance of the system during the specified period of time. Note that not only the actions from the specified period of time are relevant since the policy might require actions that must have or must not have happened at previous or later time points. With the second method, a slice is used to check system compliance for specific entities, such as all users whose login names start with the letter "a." Again, note that actions of other users might also be relevant to determining whether a particular user is compliant. The slicing method uses the data parameters of the logged actions to ensure that a slice contains all those actions that are needed to check system compliance for the specific entities. In addition to these two basic slicing methods we describe filtering methods, which can also be understood as slicing methods. Filtering discards parts of a slice to speed up the monitoring process. Furthermore, we show that our slicing and filtering methods are compositional. This allows us to split a log into slices with respect to the different dimensions of time and data.

We implement our monitoring approach using the MapReduce framework [14]. MapReduce allows us to efficiently reorganize large logs into slices and monitor these slices in parallel. In general, MapReduce computations comprise a map phase, followed by a reduce phase. In the map phase, one applies multiple instances of a map function and in the reduce phase, one applies multiple instances of

a reduce function. Each of these instances accesses only a portion of the data set, which is exploited by the MapReduce framework to run these instances in parallel on multiple computers. In our monitoring solution, we identify in the map phase, for each slice, in parallel the relevant actions for the given policy. The identified actions for a slice are then collected by the MapReduce framework, which generates the slice and passes it to an instance of our reduce function. The slices are then monitored, again in parallel, in the reduce phase.

Finally, we evaluate our monitoring solution in a large real-world case study, where we check more than 35,000 computers for compliance. The policies considered concern the processes of updating system configurations and the access to sensitive resources. We successfully monitor the logs of these computers, which consist of several billion log entries from a two year period. This shows the scalability of our approach.

We summarize our contributions as follows. We lay the foundations for splitting logs into slices for monitoring. Moreover, we provide a scalable algorithmic realization for monitoring large logs in an offline setting. Both our foundations and our algorithmic realization account for compositionality. We experimentally evaluate our compliance checking approach in a real-world setting, thereby demonstrating its effectiveness and scalability.

The remainder of the text is structured as follows. In Section 2, we lay the foundations for generating sound and complete slices. In the Sections 3 and 4, we present the different slicing and filtering methods based on data parameters and timestamps. In Section 5, we describe our case study and the experimental evaluation of our approach. In Section 6, we discuss related work before we draw conclusions in Section 7. Additional proof details are given in the appendix.

2 Slice and Monitor

In this section, we introduce the theory underpinning our approach of splitting logs and monitoring them separately and in parallel. We begin with a motivating example.

2.1 Motivating Example

We assume that a system logs the actions together with the time when they are carried out. Actions are initiated by agents, such as users or processes. For example, when an SSH connection to the computer c with session identifier s at time τ is established, we assume that the system logs the action $ssh\_login(c, s)@\tau$. Since the system is concurrent and distributed, multiple actions can be logged at the same time at different places. We monitor these logs to check whether the agents' behavior complies to policies, where we assume that the policies are specified in terms of the actions. An example of such a policy is that SSH connections must be closed after at most 24 hours.

We illustrate the splitting of logs into slices on this policy. Assume that we log two kinds of actions: $ssh\_login(c, s)@\tau$ and $ssh\_logout(c, s)@\tau$, which have

the expected interpretation. We split the log data into slices, where each slice consists of the actions with respect to a specified set of computers. If the specified sets together cover all computers, it should intuitively be clear that it suffices to monitor each of the slices separately to detect policy violations. Note that for this example, we can alternatively "slice" the log data with respect to the session identifiers.

In the remainder of this section, we present the foundations of meaningfully splitting logs. We present these foundations for temporal structures, which we use for representing logs, and metric first-order temporal logic (MFOTL), which we use as a specification language for policies. We proceed by giving background on MFOTL.

2.2 Background on MFOTL

Syntax and Semantics. Let $\mathbb{I}$ be the set of nonempty intervals over $\mathbb{N}$. We write an interval $I \in \mathbb{I}$ as $[b, b') := \{a \in \mathbb{N} \mid b \leq a < b'\}$, where $b \in \mathbb{N}$, $b' \in \mathbb{N} \cup \{\infty\}$, and $b < b'$. A *signature* 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 $S = (C, R, \iota)$ be a signature and V a countably infinite set of variables, assuming $V \cap (C \cup R) = \emptyset$.

Formulas over the signature S are given by the grammar

$$\phi ::= t_1 \approx t_2 \mid t_1 \prec t_2 \mid r(t_1, \ldots, t_{\iota(r)}) \mid (\neg\phi) \mid (\phi \vee \phi) \mid (\exists x.\, \phi) \mid (\bullet_I\, \phi) \mid (\bigcirc_I\, \phi) \mid (\phi\, \mathsf{S}_I\, \phi) \mid (\phi\, \mathsf{U}_I\, \phi)\,,$$

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

To define MFOTL's semantics, we need the following notions. A *structure* $\mathcal{D}$ over the signature S 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 $c \in C$ and $r \in R$. A *temporal structure* over S is a pair $(\bar{\mathcal{D}}, \bar{\tau})$, where $\bar{\mathcal{D}} = (\mathcal{D}_0, \mathcal{D}_1, \ldots)$ is a sequence of structures over S and $\bar{\tau} = (\tau_0, \tau_1, \ldots)$ is a sequence of natural numbers, where the following conditions hold:

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

We call the indexes 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 $i \in \mathbb{N}$. Note that there can be successive time points with equal timestamps. Furthermore, note that 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 over time.

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 $v(c) = c^{\bar{\mathcal{D}}}$. We write $f[x \leftarrow\!\!\shortmid y]$ for altering a function $f : X \to Y$ pointwise at $x \in X$. In particular, for a valuation v, a variable x, and $d \in |\bar{\mathcal{D}}|$, $v[x \leftarrow\!\!\shortmid d]$ is the valuation mapping x to d and leaving other variables' valuation unchanged.

Satisfaction in MFOTL is defined by a binary relations $\models$ over a tuple consisting of a temporal structure, a valuation, and an index on the on side and an MFOTL formula on the other side. In the following, $(\bar{\mathcal{D}}, \bar{\tau})$ is a temporal structure over the signature S, with $\bar{\mathcal{D}} = (\mathcal{D}_0, \mathcal{D}_1, \dots)$, $\bar{\tau} = (\tau_0, \tau_1, \dots)$, v a valuation, $i \in \mathbb{N}$, and ϕ a formula over S.

$$
\begin{array}{lll}
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models t \approx t' & \text{iff} & v(t) = v(t') \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models t \prec t' & \text{iff} & v(t) < v(t') \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models r(t_1, \dots, t_{\iota(r)}) & \text{iff} & \big(v(t_1), \dots, v(t_{\iota(r)})\big) \in r^{\mathcal{D}_i} \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models (\neg\phi) & \text{iff} & (\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \phi \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models (\phi \vee \psi) & \text{iff} & (\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \phi \text{ or } (\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \psi \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models (\exists x.\, \phi) & \text{iff} & (\bar{\mathcal{D}}, \bar{\tau}, v[x \leftarrow\!\!\shortmid d], i) \models \phi, \text{for some } d \in |\bar{\mathcal{D}}| \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models (\bullet_I\, \phi) & \text{iff} & i > 0,\ \tau_i - \tau_{i-1} \in I, \text{ and } (\bar{\mathcal{D}}, \bar{\tau}, v, i-1) \models \phi \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models (\bigcirc_I\, \phi) & \text{iff} & \tau_{i+1} - \tau_i \in I \text{ and } (\bar{\mathcal{D}}, \bar{\tau}, v, i+1) \models \phi \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models (\phi\, \mathsf{S}_I\, \psi) & \text{iff} & \text{for some } j \le i,\ \tau_i - \tau_j \in I, (\bar{\mathcal{D}}, \bar{\tau}, v, j) \models \psi, \\
 & & \text{and } (\bar{\mathcal{D}}, \bar{\tau}, v, k) \models \phi, \text{for all } k \in [j+1, i+1) \\
(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models (\phi\, \mathsf{U}_I\, \psi) & \text{iff} & \text{for some } j \ge i,\ \tau_j - \tau_i \in I,\ (\bar{\mathcal{D}}, \bar{\tau}, v, j) \models \psi, \\
 & & \text{and } (\bar{\mathcal{D}}, \bar{\tau}, v, k) \models \phi,\ \text{for all } k \in [i, j)
\end{array}
$$

The temporal operators $\bullet_I$ ("previous"), $\bigcirc_I$ ("next"), S_I ("since"), and U_I ("until") allow us to express both quantitative and qualitative properties with respect to the ordering of elements in the relations of the $\mathcal{D}_i$s in the temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$. Note that they are labeled with intervals I and a formula of the form $(\bullet_I\, \phi)$, $(\bigcirc_I\, \phi)$, $(\phi\, \mathsf{S}_I\, \psi)$, or $(\phi\, \mathsf{U}_I\, \psi)$ is only satisfied in $(\bar{\mathcal{D}}, \bar{\tau})$ at the time point i, if it is satisfied within the bounds given by the interval I of the respective temporal operator, which are relative to the current timestamp τ_i.

When a formula is not satisfied, we say that it is *violated* for the given valuation and i.

Terminology and Notation. We omit parentheses where possible by using the standard conventions about the binding strengths of the logical connectives. For instance, Boolean operators bind stronger than temporal ones and unary operators bind stronger than binary ones. We use standard syntactic sugar such as $\blacklozenge_I\, \phi := \mathit{true}\, \mathsf{S}_I\, \phi$, $\Diamond_I\, \phi := \mathit{true}\, \mathsf{U}_I\, \phi$, $\blacksquare_I\, \phi := \neg \blacklozenge_I \neg\phi$, and $\Box_I\, \phi := \neg \Diamond_I \neg\phi$, where $\mathit{true} := \exists x.\, x \approx x$. Intuitively, the formula $\blacklozenge_I\, \phi$ states that ϕ holds at some time point in the past within the time window I and the formula $\blacksquare_I\, \phi$ states that ϕ holds at all time points in the past within the time window I. If

the interval I includes zero, then the current time point is also considered. The corresponding future operators are $\Diamond_I$ and $\Box_I$. We also use non-metric operators like $\Box\,\phi := \Box_{[0,\infty)}\,\phi$. A formula ϕ is *bounded* if the interval I of every temporal operator U_I occurring in ϕ is finite. We use standard terminology like *atomic formula* and *subformula*.

Monitoring Logs. We illustrate our use of MFOTL and our tool for monitoring a single stream of system actions [8]. Consider the policy stating that SSH connections must last no longer than 24 hours. This can be formalized in MFOTL as

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

where we assume that time units are in hours and the signature consists of the two binary predicate symbols *ssh_login* and *ssh_logout*. We also assume in the following that the actions for establishing SSH connections and ending SSH connections are logged in relations. Specifically, for each time point $i \in \mathbb{N}$, we have the relations $\mathit{SSH\_LOGIN}_i$ and $\mathit{SSH\_LOGOUT}_i$ such that (1) $(c,s) \in \mathit{SSH\_LOGIN}_i$ iff an SSH connection with session identifier s to the computer c is established and (2) $(c,s) \in \mathit{SSH\_LOGOUT}_i$ iff the SSH connection with session identifier s to the computer c is closed. Observe that at the same time point multiple SSH connections can be established and closed. Furthermore, every time point i has a timestamp $\tau_i \in \mathbb{N}$, recording the time when the actions in $\mathit{SSH\_LOGIN}_i$ and $\mathit{SSH\_LOGOUT}_i$ occurred.

The corresponding temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$ is as follows. The domain of $\bar{\mathcal{D}}$ contains all possible computer names and session identifiers. Without loss of generality, we assume that both computer names and session identifiers are represented as natural numbers, that is, $|\bar{\mathcal{D}}| = \mathbb{N}$. The ith structure in $\bar{\mathcal{D}}$ contains the relations $\mathit{SSH\_LOGIN}_i$ and $\mathit{SSH\_LOGOUT}_i$, interpreting the predicate symbols *ssh_login* and *ssh_logout* at time point i. The ith timestamp in $\bar{\tau}$ is simply τ_i.

To detect policy violations, we use the monitoring tool MONPOLY [8], which implements the monitoring algorithm in [10]. MONPOLY iteratively processes the temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$ representing the stream of logged actions. This can be done offline or online. At each time point i, it outputs the valuations satisfying the negation of the formula $\psi = \mathit{ssh\_login}(c,s) \rightarrow \Diamond_{[0,25)}\; \mathit{ssh\_logout}(c,s)$, which is $\neg\psi$ and equivalent to $\mathit{ssh\_login}(c,s) \wedge \neg \Diamond_{[0,25)}\; \mathit{ssh\_logout}(c,s)$. Note that we drop the outermost quantifiers since we are not only interested in whether the policy is violated but we also want additional information about the violations themselves, e.g., the session identifiers of the SSH connections that lasted longer than 24 hours.

In general, we assume that policies formalized in MFOTL are of the form $\Box\,\psi$, where ψ is bounded. Since ψ is bounded, the monitor only needs to process a finite prefix of $(\bar{\mathcal{D}}, \bar{\tau})$ 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 [10], a generalization of the standard notion of safe-range database queries [3]. We refer to [10] for a detailed description of the monitoring algorithm. The tool MONPOLYis publicly available at `https://projects.developer.nokia.com/MonPoly/`.

2.3 Slicing Framework

In the remainder of the text, we assume that all temporal structures are over the same signature (C, R, ι), have the same domain $\mathbb{D}$, and the constant symbols in C are interpreted equally. Recall that V is the set of variables. Furthermore, we assume without loss of generality that variables are quantified at most once in a formula and that quantified variables are disjoint from its free variables.

Slices. The theoretical core of our work is to split a temporal structure, which represents a stream of logged system actions, into smaller temporal structures. The smaller temporal structures, called slices, are formally defined in Definition 2.1.

Definition 2.1. *Let* $(\bar{\mathcal{D}}, \bar{\tau})$ *and* $(\bar{\mathcal{D}}', \bar{\tau}')$ *be temporal structures,* $\ell \in \mathbb{N} \cup \{\infty\}$ *a length, and* $s : [0, \ell) \to \mathbb{N}$ *a strictly increasing monotonic function.* $(\bar{\mathcal{D}}', \bar{\tau}')$ *is a* slice *of* $(\bar{\mathcal{D}}, \bar{\tau})$ *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* $r \in R$*.*

Intuitively, a slice consists only of *some* of the logged system actions, up to the time point ℓ. The function s determines the source of the structure $\mathcal{D}'_i$ and the timestamp τ'_i, for each $i \in [0, \ell)$. The case where ℓ is not ∞ corresponds to the situation where we consider only a finite subsequence of the stream of logged data. However, since temporal structures are by definition infinite sequences of structures and timestamps, we mark the end of the finite sequence by $\ell \in \mathbb{N}$. The suffix of $(\bar{\mathcal{D}}, \bar{\tau})$ after the position ℓ is irrelevant.

In practice, we can only monitor finite prefixes of temporal structures. Hence, the original temporal structure and the slices are finite prefixes in an implementation. To ease the exposition, we require that temporal structures and thus also logs describe infinite streams of system actions.

Soundness and Completeness Requirements. To meaningfully monitor slices separately, we impose that at least one of the slices violates the given policy if and only if the original temporal structure violates the policy. We relax this requirement by associating each slice with a space that restricts the kind of violations. In the following, we define soundness and completeness requirements for the slices relative to such spaces. We call these spaces *restrictions*.

Definition 2.2. *A* restriction *is a pair* (D, T)*, where* $D : V \to 2^{\mathbb{D}}$ *and* $T \subseteq \mathbb{N}$ *is an interval.*

A valuation v is *permitted* by the restriction (D, T) if $v(x) \in D(x)$, for every variable $x \in V$. A timestamp τ is *permitted* by the restriction (D, T) if $\tau \in T$. The restriction (D, T) with $D(x) = \mathbb{D}$, for each $x \in V$, and $T = \mathbb{N}$ is called the *non-restrictive* restriction.

Definition 2.3. *Let* $(\bar{\mathcal{D}}, \bar{\tau})$ *and* $(\bar{\mathcal{D}}', \bar{\tau}')$ *be temporal structures,* (D, T) *a restriction, and* ϕ *a formula.*

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

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

Each slice of a temporal structure is associated with a restriction. The original temporal structure is associated with the non-restrictive restriction. If we split a temporal structure into slices, we must associate a restriction with each slice. These restrictions refine the restriction associated to the given temporal structure. In Definition 2.4 we give conditions that the refined restrictions must fulfill.

Definition 2.4. *A family of restrictions* $(D^k, T^k)_{k \in K}$ refines *the restriction* (D, T) *if*

(R1) $D(x) \supseteq \bigcup_{k \in K} D^k(x)$*, for every* $x \in V$*,*
(R2) $T \supseteq \bigcup_{k \in K} T^k$*, and*
(R3) for every valuation v *permitted by* (D, T) *and for every* $t \in T$*, there is some* $k \in K$ *such that* v *is permitted by* (D^k, T^k) *and* $t \in T^k$*.*

Intuitively, the conditions (R1) and (R2) require that the refined restrictions do not permit more than the original restriction. That is, the family of refined restrictions must not permit valuations and timestamps that are not permitted by the original restriction. The condition (R3) requires that the refined restrictions cover everything covered by the original restriction. That is, every combination of a valuation and a timestamp permitted by the original restriction must be permitted by at least one of the refined restrictions.

Slicers. We call a mechanism that generates the slices and the associated restrictions a *slicer*. In Definition 2.5 we give requirements that the slices and the associated restrictions produced by a slicer must fulfill. In Theorem 2.1 we show that these requirements suffice to ensure that monitoring the slices is equivalent to monitoring the original temporal structure with respect to the associated restrictions. In the Sections 3 and 4, we provide specific slicers that split a temporal structure by data and by timestamps and filter out parts of the temporal structure that are "irrelevant" with respect to the monitored formula. Algorithmic realizations of slicers are given in Section 5.

Definition 2.5. *A* slicer $\mathfrak{s}_\phi$ *for the formula* ϕ *is a function that takes as input a temporal structure* $(\bar{\mathcal{D}}, \bar{\tau})$ *and a restriction* (D, T)*. It returns a family of temporal structures* $(\bar{\mathcal{D}}^k, \bar{\tau}^k)_{k \in K}$ *and a family of restrictions* $(D^k, T^k)_{k \in K}$*, where the returned families satisfy the following criteria:*

(S1) $(D^k, T^k)_{k\in K}$ *refines* (D, T).
(S2) $(\bar{\mathcal{D}}^k, \bar{\tau}^k)$ *is* (D^k, T^k)-sound *for* $(\bar{\mathcal{D}}, \bar{\tau})$ *and* ϕ, *for each* $k \in K$.
(S3) $(\bar{\mathcal{D}}^k, \bar{\tau}^k)$ *is* (D^k, T^k)-complete *for* $(\bar{\mathcal{D}}, \bar{\tau})$ *and* ϕ, *for each* $k \in K$.

Theorem 2.1. *Let* $\mathfrak{s}_\phi$ *be a slicer for the formula* ϕ. *Let* $\mathfrak{s}_\phi$, *on input temporal structure* $(\bar{\mathcal{D}}, \bar{\tau})$ *and restriction* (D, T), *return the family of temporal structures* $(\bar{\mathcal{D}}^k, \bar{\tau}^k)_{k\in K}$ *and the family of restrictions* $(D^k, T^k)_{k\in K}$ *as output. The following conditions are equivalent:*

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

Proof. (2) follows from (1) because $(D^k, T^k)_{k\in K}$ refines (D, T) (condition (S1) in Definition 2.5) and because $(\bar{\mathcal{D}}^k, \bar{\tau}^k)$ is (D^k, T^k)-sound for $(\bar{\mathcal{D}}, \bar{\tau})$ and ϕ, for each $k \in K$ (condition (S2) in Definition 2.5).

To show that (2) implies (1), we show that the contrapositive. Let v be a valuation and $i \in \mathbb{N}$ such that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \phi$, v is permitted by (D, T) and $\tau_i \in T$. Because $(D^k, T^k)_{k\in K}$ refines (D, T) (condition (S1) in Definition 2.5), there is a $k \in K$ such that v is permitted by (D^k, T^k) and $\tau_i \in T^k$. Because $(\bar{\mathcal{D}}^k, \bar{\tau}^k)$ is (D^k, T^k)-complete for $(\bar{\mathcal{D}}, \bar{\tau})$ and ϕ (condition (S3) in Definition 2.5), $(\bar{\mathcal{D}}^k, \bar{\tau}^k, v, j) \not\models \phi$, for some $j \in \mathbb{N}$ with $\tau_j^k = \tau_i$. □

Note that Theorem 2.1 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, the valuation and timestamp for a violation must match between the original temporal structure and the slice.

Combination. For temporal structures representing very large logs, a single slicer may not suffice to obtain slices of manageable sizes. To overcome this problem, we combine slicers: the slices produced by one slicer can be further decomposed by another slicer to obtain smaller slices. We formalize the combination of slicers in Definition 2.6. Afterwards, in Theorem 2.2, we prove that the combination of slicers is again a slicer.

Definition 2.6. *Let* $\mathfrak{s}_\phi$ *and* $\mathfrak{s}'_\phi$ *be slicers for the formula* ϕ. *Given input temporal structure* $(\bar{\mathcal{D}}, \bar{\tau})$ *and restriction* (D, T), *the* combination $\mathfrak{s}'_\phi \circ_{\hat{k}} \mathfrak{s}_\phi$ *for the index* $\hat{k}$ *produces output as follows. Let* $(\bar{\mathcal{D}}^k, \bar{\tau}^k)_{k\in K}$ *and* $(D^k, T^k)_{k\in K}$ *be the family of temporal structures and the family of restrictions returned by* $\mathfrak{s}_\phi$ *for the input* $(\bar{\mathcal{D}}, \bar{\tau})$ *and* (D, T).

- *If* $\hat{k} \notin K$ *then* $\mathfrak{s}'_\phi \circ_{\hat{k}} \mathfrak{s}_\phi$ *returns* $(\bar{\mathcal{D}}^k, \bar{\tau}^k)_{k\in K}$ *and* $(D^k, T^k)_{k\in K}$.
- *If* $\hat{k} \in K$ *then* $\mathfrak{s}'_\phi \circ_{\hat{k}} \mathfrak{s}_\phi$ *returns* $(\bar{\mathcal{D}}^k, \bar{\tau}^k)_{k\in K''}$ *and* $(D^k, T^k)_{k\in K''}$, *where* $K'' := (K \setminus \{\hat{k}\}) \cup K'$ *and* $(\bar{\mathcal{D}}^k, \bar{\tau}^k)_{k\in K'}$ *and* $(D^k, T^k)_{k\in K'}$ *are the families returned by* $\mathfrak{s}'_\phi$ *for the input* $(\bar{\mathcal{D}}^{\hat{k}}, \bar{\tau}^{\hat{k}})$ *and* $(D^{\hat{k}}, T^{\hat{k}})$, *assuming* $K \cap K' = \emptyset$.

Intuitively, we first apply the slicer $\mathfrak{s}_\phi$. 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 $\mathfrak{s}'_\phi$ does nothing. Otherwise, we use $\mathfrak{s}'_\phi$ to make the $\hat{k}$th slice smaller. Note that by combing the slicer $\mathfrak{s}_\phi$ with different indices, we can slice all of $\mathfrak{s}_\phi$'s outputs further. Note too that an algorithmic realization of the function $\mathfrak{s}'_\phi \circ_{\hat{k}} \mathfrak{s}_\phi$ does not necessarily need to compute the output of $\mathfrak{s}_\phi$ first before applying $\mathfrak{s}'_\phi$.

Theorem 2.2. *The combination $\mathfrak{s}'_\phi \circ_{\hat{k}} \mathfrak{s}_\phi$ of the slicers $\mathfrak{s}_\phi$ and $\mathfrak{s}'_\phi$ for the formula ϕ is a slicer for the formula ϕ.*

Proof. We show that $\mathfrak{s}'_\phi \circ_{\hat{k}} \mathfrak{s}_\phi$ satisfies the conditions (S1) to (S3) in Definition 2.5. Regarding (S1), $\mathfrak{s}_\phi$ is a slicer and therefore the family $(D^k, T^k)_{k \in K}$ refines (D, T). If $\hat{k} \notin K$, then we are done. If $\hat{k} \in K$, then $\mathfrak{s}'_\phi$ is a slicer and therefore the family $(D^k, T^k)_{k \in K'}$ refines $(D^{\hat{k}}, T^{\hat{k}})$. From $K \cap K' = \emptyset$, it follows that $(D^k, T^k)_{k \in (K \setminus \{\hat{k}\}) \cup K'}$ refines (D, T).

Regarding (S2), $\mathfrak{s}_\phi$ is a slicer and therefore $(\bar{\mathcal{D}}^k, \bar{\tau}^k)$ is (D^k, T^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 $\mathfrak{s}'_\phi$ is a slicer and therefore $(\bar{\mathcal{D}}^k, \bar{\tau}^k)$ is (D^k, T^k)-sound for $(\bar{\mathcal{D}}^{\hat{k}}, \bar{\tau}^{\hat{k}})$ and ϕ, for every $k \in K'$. Because $(D^k, T^k)_{k \in K'}$ refines $(D^{\hat{k}}, T^{\hat{k}})$ and because $(\bar{\mathcal{D}}^{\hat{k}}, \bar{\tau}^{\hat{k}})$ is $(D^{\hat{k}}, T^{\hat{k}})$-sound for $(\bar{\mathcal{D}}, \bar{\tau})$ and ϕ, it follows that $(\bar{\mathcal{D}}^k, \bar{\tau}^k)$ is (D^k, T^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 $k \in (K \setminus \{\hat{k}\}) \cup K'$.

Finally, the proof for (S3) is analogous to the proof for (S2). □

3 Slicing Data

In this section, we present slicers that split the relations of a temporal structure. We call the resulting slices data slices. Formally, the temporal structure $(\bar{\mathcal{D}}', \bar{\tau}')$ is a *data slice* of the temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$ 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 2.1 is the identity function and $\ell = \infty$. In the following, we present concrete slicers, so-called *data slicers*, that return sound and complete data slices relative to a restriction. We also present a fragment of MFOTL for which the produced data slices are sound with respect to less restrictive restrictions.

3.1 Data Slicer

In a nutshell, a data slicer works as follows. A data slicer for a formula ϕ, a *slicing variable* x, which is a free variable in ϕ, and *slicing sets*, that is, sets of possible values for x, 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 the elements of the relations interpreting the predicate symbols that are irrelevant to determine 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 that a data slicer outputs.

Definition 3.1. *Let ϕ be a formula, $x \in V$ a slicing variable, $(\bar{\mathcal{D}}, \bar{\tau})$ a temporal structure, and $S \subseteq \mathbb{D}$ a slicing set. The (S, x, ϕ)-*slice *of $(\bar{\mathcal{D}}, \bar{\tau})$ is the data slice $(\bar{\mathcal{D}}', \bar{\tau}')$, where the relations are as follows. For all $r \in R$, all $i \in \mathbb{N}$, and all $a_1, \ldots, a_{\iota(r)} \in \mathbb{D}$, it holds that $(a_1, \ldots, a_{\iota(r)}) \in r^{\mathcal{D}'_i}$ iff for every j with $1 \leq j \leq \iota(r)$, there is an atomic subformula of ϕ of the form $r(t_1, \ldots, t_{\iota(r)})$ that fulfills at least one of the following conditions:*

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

Intuitively, the Conditions (a)–(c) ensure that a slice contains all elements from the relations interpreting predicate symbols that are needed to evaluate ϕ when x takes values from the slicing set. For this, we only need to consider atomic subformulas of ϕ that contain a predicate symbol. The elements themselves are tuples and every item of the tuple must satisfy at least one of the conditions. If a predicate symbol includes the slicing variable, then only values from the slicing set are relevant (Condition (a)). If it includes another variable, then all possible values are relevant (Condition (b)). Finally, if it includes a constant symbol, then the interpretation of the constant symbol is relevant (Condition (c)). In the special case of a predicate symbols with arity 0, tuples from the interpretation of these predicate symbols are always included in a data slice independently of the formula. Further optimizations are possible, namely, including only tuples from the interpretation of those predicate symbols that occur in the formula.

The following example illustrates Definition 3.1. It also demonstrates that the choice of the slicing variable can influence how lean the slices are and how much overhead, that is duplication of log data in slices, the slicing causes. Ideally, we want each logged action to appear in exactly one slice, but some logged actions may have to be duplicated in multiple slices. In the worst case, each slice contains the complete original temporal structure.

Example 3.1. Consider the formula $\phi = \mathit{snd}(\mathit{src}, \mathit{msg}) \rightarrow \Diamond_{[0,6)}\, \mathit{rcv}(\mathsf{0}, \mathit{msg})$, where *scr* and *msg* are variables and $\mathsf{0}$ is a constant symbol that is interpreted by the domain element 0. Intuitively, the formula ϕ formalizes that all sent messages are received by node 0 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 *snd* and *rcv* are

$$\mathit{snd}^{\mathcal{D}_0} = \{(1,1), (1,2), (3,3), (4,4)\} \text{ and } \mathit{rcv}^{\mathcal{D}_0} = \{(0,1), (0,2), (0,3), (0,4)\}\,.$$

We slice on the variable *src*. For the slicing sets $S = \{1, 2\}$, the (S, src, ϕ)-slice contains the structure $\mathcal{D}'_0$ with

$$\mathit{snd}^{\mathcal{D}'_0} = \{(1,1), (1,2)\} \text{ and } \mathit{rcv}^{\mathcal{D}'_0} = \{(0,1), (0,2), (0,3), (0,4)\}\,.$$

For the predicate symbol *snd*, 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 *src* in the formula. For the predicate symbol *rcv*, those tuples are included where the first parameter is 0 because it occurs as a constant symbol in the formula.

For the slicing set $S' = \{3, 4\}$, the (S', src, ϕ)-slice contains the structure $\mathcal{D}_0''$ with

$$snd^{\mathcal{D}_0''} = \{(3,3),(4,4)\} \text{ and } rcv^{\mathcal{D}_0''} = \{(0,1),(0,2),(0,3),(0,4)\}.$$

The tuples in the relation for the predicate symbol *rcv* are duplicated in all slices because the first element of the tuples, 0, occurs as a constant symbol in the formula. Condition (c) in Definition 3.1 is therefore always satisfied and the tuple is included.

Next, we slice on the variable *msg* instead of the variable *src*. The (S, msg, ϕ)-slice contains the structure $\mathcal{D}_0'$ with

$$snd^{\mathcal{D}_0'} = \{(1,1),(1,2)\} \text{ and } rcv^{\mathcal{D}_0'} = \{(0,1),(0,2)\}.$$

For both predicate symbols *snd* and *rcv*, only those tuples are included where the second parameter (variable *msg*) takes values from the slicing set S. This is because the second parameter occurs as the slicing variable *msg* in the formula. The (S', msg, ϕ)-slice contains the structure $\mathcal{D}_0''$ with

$$snd^{\mathcal{D}_0''} = \{(3,3),(4,4)\} \text{ and } rcv^{\mathcal{D}_0''} = \{(0,3),(0,4)\}.$$

The following lemma shows that an (S, x, ϕ)-slice is truth preserving for valuations of the slicing variable x within the slicing sets S. We use the lemma to establish the soundness and completeness of data slices, thereby showing in Theorem 3.1 that a data slicer is a slicer, and therefore Theorem 2.1 applies.

Lemma 3.1. *Let ϕ be a formula, $x \in V$ a variable not bound in ϕ, $(\bar{\mathcal{D}}, \bar{\tau})$ a temporal structure, $S \subseteq \mathbb{D}$ a slicing set, and $(\bar{\mathcal{D}}', \bar{\tau})$ the (S, x, ϕ)-slice of $(\bar{\mathcal{D}}, \bar{\tau})$. For all $i \in \mathbb{N}$ and valuations v with $v(x) \in S$ it holds that $(\bar{\mathcal{D}}', \bar{\tau}, v, i) \models \phi$ iff $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \phi$.*

Proof. 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 $r(t_1, \ldots, t_{\iota(r)})$.

Satisfaction of $t \prec t'$ and $t \approx t$ depends only on the valuation, so it trivially follows that $(\bar{\mathcal{D}}', \bar{\tau}, v, i) \models t \prec t'$ 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 $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models t \approx t'$. For the formula $r(t_1, \ldots, t_{\iota(r)})$ we show the two directions of the equivalence separately.

1. We first show that $(\bar{\mathcal{D}}', \bar{\tau}, v, i) \models r(t_1, \ldots, t_{\iota(r)})$ implies $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models r(t_1, \ldots, t_{\iota(r)})$. From $(\bar{\mathcal{D}}', \bar{\tau}, v, i) \models r(t_1, \ldots, t_{\iota(r)})$ it follows that $(v(t_1), \ldots, v(t_{\iota(r)})) \in r^{\mathcal{D}_i'}$. Since $(\bar{\mathcal{D}}', \bar{\tau})$ is a data slice of $(\bar{\mathcal{D}}, \bar{\tau})$ it follows that $(v(t_1), \ldots, v(t_{\iota(r)})) \in r^{\mathcal{D}_i}$ and hence $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models r(t_1, \ldots, t_{\iota(r)})$.

2. Next, we show that $(\bar{\mathcal{D}}', \bar{\tau}, v, i) \not\models r(t_1, \ldots, t_{\iota(r)})$ implies $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models r(t_1, \ldots, t_{\iota(r)})$. From $(\bar{\mathcal{D}}', \bar{\tau}, v, i) \not\models r(t_1, \ldots, t_{\iota(r)})$ it follows that $(v(t_1), \ldots, v(t_{\iota(r)})) \notin r^{\mathcal{D}'_i}$. We first show that one of the Conditions (a)–(c) in Definition 3.1 is satisfied for the tuple $(v(t_1), \ldots, v(t_{\iota(r)}))$. For any j with $1 \leq j \leq \iota(r)$, we make a case split based on whether the term t_j in $r(t_1, \ldots, t_{\iota(r)})$ is the slicing variable x, another variable $y \neq x$, or a constant symbol c.
   (a) If t_j is the slicing variable x then from $v(x) \in S$ we know that $v(t_j) \in S$. Therefore, Condition (a) is satisfied.
   (b) If t_j is a variable $y \neq x$ then Condition (b) is satisfied.
   (c) If t_j is the constant symbol c then $v(t_j) = c^{\bar{\mathcal{D}}}$ and hence Condition (c) is satisfied.

   It follows that $(v(t_1), \ldots, v(t_{\iota(r)})) \notin r^{\mathcal{D}_i}$ and hence $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models r(t_1, \ldots, t_{\iota(r)})$.

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

According to Definition 3.2 and Theorem 3.1 below, a data slicer is a slicer that splits a temporal structure into a family of (S, x, ϕ)-slices. Furthermore, it refines the given restriction with respect to the given slicing sets.

Definition 3.2. *A* data slicer $\mathfrak{d}_{\phi,x,(S^k)_{k\in K}}$ *for the formula* ϕ*, slicing variable* $x \in V$*, and family of slicing sets* $(S^k)_{k\in K}$ *is a function that takes as input a temporal structure* $(\bar{\mathcal{D}}, \bar{\tau})$ *and a restriction* (D, T)*. It returns a family of temporal structures* $(\bar{\mathcal{D}}^k, \bar{\tau}^k)_{k\in K}$ *and a family of restrictions* $(D^k, T^k)_{k\in K}$*, where* $(\bar{\mathcal{D}}^k, \bar{\tau}^k)$ *is the* $(S^k \cap D(x), x, \phi)$*-slice of* $(\bar{\mathcal{D}}, \bar{\tau})$ *and* $(D^k, T^k) = (D[x \leftarrow S^k \cap D(x)], T)$*, for each* $k \in K$*.*

Theorem 3.1. *A data slicer* $\mathfrak{d}_{\phi,x,(S^k)_{k\in K}}$ *is a slicer for the formula* ϕ *if the slicing variable* x *is not bound in* ϕ *and* $\bigcup_{k\in K} S^k = \mathbb{D}$*.*

Proof. We show that $\mathfrak{d}_{\phi,x,(S^k)_{k\in K}}$ satisfies the criteria (S1)–(S3) in Definition 2.5.

For (S1), we show that the family $(D^k, T^k)_{k\in K}$ fulfills the conditions (R1)–(R3) in Definition 2.4: (R1) follows from $\bigcup_{k\in K} D^k(x) = \bigcup_{k\in K}(S^k \cap D(x)) \subseteq \bigcup_{k\in K} D(x) = D(x)$. (R2) follows from $T^k = T$, for each $k \in K$. (R3) follows from the assumption $\bigcup_{k\in K} S^k = \mathbb{D}$ and the equalities $D^k = D[x \leftarrow S^k \cap D(x)]$ and $T^k = T$, for each $k \in K$.

(S2) and (S3) follow directly from Lemma 3.1. □

3.2 Data Filter

We can speed up the monitoring of temporal structures by filtering them before monitoring. Filtering removes those parts of the temporal structure that are not

needed to evaluate the monitored formula. In the following, we present a slicer, which we call a *data filter*, that discards logged actions.

A data filter is a special case of a data slicer, where the slicing variable does not occur in ϕ and where we consider only the single slicing set of all possible values. The data filter produces a single slice, the filtered temporal structure, and a single restriction, which is identical to the restriction of the original temporal structure. Intuitively, the obtained slice excludes all tuples from the relations interpreting those predicate symbols that do not occur in ϕ. Furthermore, if a predicate symbol $r \in R$ occurs in ϕ only with arguments that are constant symbols, then tuples with interpretations of those constant symbols are excluded that do not occur in ϕ as arguments of r.

Definition 3.3. *A* data filter $\mathfrak{f}_\phi$ *for the formula* ϕ *is a data slicer* $\mathfrak{d}_{\phi,x,(S^k)_{k\in K}}$*, where the slicing variable* $x \in V$ *does not occur in* ϕ*,* $S^0 = \mathbb{D}$*, and* $K = \{0\}$*.*

It is easy to see that the slice that a data filter outputs is independent of the choice of the slicing variable and therefore the slice is unique. From Lemma 3.1 we obtain that the filtered and the original temporal structures are equivalent in the sense that the filtered temporal structure satisfies the formula ϕ exactly when the original temporal structure satisfies the formula with respect to the restriction (D, T). If D does not restrict the range of the slicing variable, we obtain full equivalence in the sense that, irrespective of any restrictions, the filtered temporal structure satisfies ϕ exactly when the original temporal structure satisfies ϕ. Note that the data filter is a slicer by Theorem 3.1.

The filtering feature of the data filter is built-in into the data slicer. Therefore, applying the data filter to a data slice would have no effect. However, such a filtering feature may be missing in other slicers, such as the time slicer described in Section 4.1, so it makes sense to data filter slices in general.

3.3 A Non-Restrictive Fragment

In the following, we describe a fragment where no restrictions are needed. By Lemma 3.1 the slices from Definition 3.1 are sound and complete for valuations where the slicing variable takes values from the slicing set. That is, policy violations where the slicing variable takes values outside of the slicing set are "spurious" violations. For formulas in the fragment, no spurious violations exist. This allows us to use any MFOTL monitoring algorithm without modifying it to suppress the spurious violations.

To describe the fragment, we first introduce the notion of *valid* slicing sets in Definition 3.4 and *variable overlap* in Definition 3.5. Intuitively, a slicing set is valid if it includes the interpretations of the constant symbols from the signature. Note that we can always assume the smallest such set that contains only the constant symbols that occur in the formula ϕ. Distinct variables overlap in a formula if they are used as the same argument of a predicate symbol.

Definition 3.4. *The set* S *is a* valid *slicing set for the temporal structure* $(\bar{\mathcal{D}}, \bar{\tau})$ *if* $c^{\bar{\mathcal{D}}} \in S$*, for all* $c \in C$*.*

Example 3.2. Consider the formula $\phi = \Box(p(x) \rightarrow q(x)) \wedge p(c)$, where c is a constant symbol. Suppose we slice for the variable x, where we choose an invalid slicing set S that does not contain c's interpretation. In the slice, due to Condition (c) in Definition 3.1, c's interpretation is included in p's interpretation at a time point whenever it is contained in p's interpretation in the original temporal structure. However, it is not in q's interpretation. Therefore, spurious violations might be reported when monitoring the (S, x, ϕ)-slice.

Definition 3.5. *Two distinct* variables x *and* y overlap *in the formula* ϕ *if for some predicate symbol* $r \in R$, ϕ *contains atomic subformulas* $r(s_1, \ldots, s_{\iota(r)})$ *and* $r(t_1, \ldots, t_{\iota(r)})$ *where* $s_j = x$ *and* $t_j = y$, *for some* j *with* $1 \leq j \leq \iota(r)$,

Example 3.3. If the slicing variable overlaps with another variable for the predicate symbol $r \in R$, then all tuples from r's interpretation are included in a slice, independently of the contents of the slicing set. This leads to spurious violations if there is no such variable overlap for other predicate symbols. For example, consider the formula $\Box(p(x) \rightarrow q(x)) \vee \blacksquare_{[0,3)} \neg p(y)$ and slicing for the variable x. Only some tuples from the relations for q are included in a slice but all tuples of the relations of p are included. Therefore, spurious violations might be reported when monitoring the slice.

Next, we define the sets DT, DF, and DE. Membership of a formula in these sets reflects whether the monitored formula is satisfied on the slice for a slicing variable interpretation that lies outside of the slicing set. In a nutshell, for all slicing variable interpretations outside of the slicing set, a formula in the set DF is never satisfied, a formula in the set DT is always satisfied, and a formula in the set DE is satisfied whenever it is satisfied on the original temporal structure. The sets are parametrized by the slicing variable. For example, for the slicing variable x, the sets are DT_x, DF_x, and DE_x.

Definition 3.6. *Let* ϕ *be a formula and* $x \in V$ *a variable that does not overlap with another variable in* ϕ.

1. $\phi \in \mathrm{DT}_x$ *iff for all formulas* ζ *where* x *does not overlap with another variable in* ζ *and* ϕ *is a subformula of* ζ, *all temporal structures* $(\bar{\mathcal{D}}, \bar{\tau})$, *all slicing sets* $S \subseteq \mathbb{D}$ *that are valid for* $(\bar{\mathcal{D}}, \bar{\tau})$ *and* ϕ, *all valuations* v *with* $v(x) \notin S$, *and all* $i \in \mathbb{N}$, *it holds that* $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models \phi$, *where* $(\bar{\mathcal{P}}, \bar{\tau})$ *is the* (S, x, ζ)*-slice of* $(\bar{\mathcal{D}}, \bar{\tau})$,
2. $\phi \in \mathrm{DF}_x$ *iff for all formulas* ζ *where* x *does not overlap with another variable in* ζ *and* ϕ *is a subformula of* ζ, *all temporal structures* $(\bar{\mathcal{D}}, \bar{\tau})$, *all slicing sets* $S \subseteq \mathbb{D}$ *that are valid for* $(\bar{\mathcal{D}}, \bar{\tau})$ *and* ϕ, *all valuations* v *with* $v(x) \notin S$, *and all* $i \in \mathbb{N}$, *it holds that* $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \not\models \phi$, *where* $(\bar{\mathcal{P}}, \bar{\tau})$ *is the* (S, x, ζ)*-slice of* $(\bar{\mathcal{D}}, \bar{\tau})$,
3. $\phi \in \mathrm{DE}_x$ *iff for all formulas* ζ *where* x *does not overlap with another variable in* ζ *and* ϕ *is a subformula of* ζ, *all temporal structures* $(\bar{\mathcal{D}}, \bar{\tau})$, *all slicing sets* $S \subseteq \mathbb{D}$ *that are valid for* $(\bar{\mathcal{D}}, \bar{\tau})$ *and* ϕ, *all valuations* v *with* $v(x) \notin S$, *and all* $i \in \mathbb{N}$, *it holds that* $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models \phi$ *iff* $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \phi$, *where* $(\bar{\mathcal{P}}, \bar{\tau})$ *is the* (S, x, ζ)*-slice of* $(\bar{\mathcal{D}}, \bar{\tau})$.

$$\frac{}{r(t_1,\ldots,t_{\iota(r)}) : \mathsf{DF}_x}\ x \in \{t_1,\ldots,t_{\iota(r)}\} \qquad \frac{}{r(t_1,\ldots,t_{\iota(r)}) : \mathsf{DE}_x}\ x \notin \{t_1,\ldots,t_{\iota(r)}\}$$

$$\frac{}{t \approx t' : \mathsf{DE}_x} \qquad \frac{}{t \prec t' : \mathsf{DE}_x} \qquad \frac{}{\mathit{true} : \mathsf{DT}_x} \qquad \frac{}{\mathit{true} : \mathsf{DE}_x}$$

$$\frac{\psi : \mathsf{DF}_x}{\neg\psi : \mathsf{DT}_x} \qquad \frac{\psi : \mathsf{DT}_x}{\neg\psi : \mathsf{DF}_x} \qquad \frac{\psi : \mathsf{DE}_x}{\neg\psi : \mathsf{DE}_x}$$

$$\frac{\psi : \mathsf{DT}_x}{\psi \vee \chi : \mathsf{DT}_x} \qquad \frac{\chi : \mathsf{DT}_x}{\psi \vee \chi : \mathsf{DT}_x} \qquad \frac{\psi : \mathsf{DF}_x \quad \chi : \mathsf{DF}_x}{\psi \vee \chi : \mathsf{DF}_x} \qquad \frac{\psi : \mathsf{DE}_x \quad \chi : \mathsf{DE}_x}{\psi \vee \chi : \mathsf{DE}_x}$$

$$\frac{\psi : \mathsf{DT}_x}{\exists y.\, \psi : \mathsf{DT}_x}\ x \neq y \qquad \frac{\psi : \mathsf{DF}_x}{\exists y.\, \psi : \mathsf{DF}_x}\ x \neq y \qquad \frac{\psi : \mathsf{DE}_x}{\exists y.\, \psi : \mathsf{DE}_x}\ x \neq y$$

$$\frac{\psi : \mathsf{DF}_x}{\bullet_I\, \psi : \mathsf{DF}_x} \qquad \frac{\psi : \mathsf{DE}_x}{\bullet_I\, \psi : \mathsf{DE}_x} \qquad \frac{\psi : \mathsf{DF}_x}{\circ_I\, \psi : \mathsf{DF}_x} \qquad \frac{\psi : \mathsf{DE}_x}{\circ_I\, \psi : \mathsf{DE}_x}$$

$$\frac{\chi : \mathsf{DT}_x}{\psi\ \mathsf{S}_I\ \chi : \mathsf{DT}_x} \qquad \frac{\chi : \mathsf{DF}_x}{\psi\ \mathsf{S}_I\ \chi : \mathsf{DF}_x} \qquad \frac{\psi : \mathsf{DE}_x \quad \chi : \mathsf{DE}_x}{\psi\ \mathsf{S}_I\ \chi : \mathsf{DE}_x}$$

$$\frac{\chi : \mathsf{DT}_x}{\psi\ \mathsf{U}_I\ \chi : \mathsf{DT}_x} \qquad \frac{\chi : \mathsf{DF}_x}{\psi\ \mathsf{U}_I\ \chi : \mathsf{DF}_x} \qquad \frac{\psi : \mathsf{DE}_x \quad \chi : \mathsf{DE}_x}{\psi\ \mathsf{U}_I\ \chi : \mathsf{DE}_x}$$

Figure 1. Labeling Rules (Slicing by Data)

Membership in the sets DT_x, DF_x, and DE_x is in general undecidable. We delay the proof of this statement to the end of this section because the proof uses Lemmas and Theorems established in the rest of this section. Note that these Lemmas and Theorems do not rely on the statement about undecidability or its proof.

Given undecidability, we approximate membership with syntactic fragments. The fragments are defined in terms of a labeling algorithm that assigns the labels DT_x, DF_x, and DE_x to a formula. The fragments are sound but incomplete in the sense that if a formula is assigned to a label (DT_x, DF_x, or DE_x) then the formula is in the corresponding set (DT_x, DF_x, or DE_x, respectively). However, not every formula in one of the sets is assigned to the corresponding label. The algorithm labels atomic subformulas of a formula and propagates the labels bottom-up to the formula's root using the labeling rules in Figure 1. Note that any syntactic sugar must be unfolded before applying the rules.

Remark 3.1. From the labeling rules for *true* and the operators S and U we see that the formulas $\blacklozenge_I\, \psi$, $\diamondsuit_I\, \psi$, $\blacksquare_I\, \psi$, and $\square_I\, \psi$ are labeled exactly as the formula ψ, that is, DT_x if $\psi : \mathsf{DT}_x$, DF_x if $\psi : \mathsf{DF}_x$, and DE_x if $\psi : \mathsf{DE}_x$.

Remark 3.2. We cannot propagate the label DT_x over the operators $\bullet_I$ and $\circ_I$ because we do not know whether $\tau_i - \tau_{i-1} \in I$ and $\tau_{i+1} - \tau_i \in I$, respectively. For $\bullet_I$, we also do not know whether $i > 0$.

Example 3.4. Consider the formula $\square\, \mathit{snd}(\mathit{src}, \mathit{msg}) \rightarrow \diamondsuit_{[0,6)}\, \mathit{rcv}(\mathsf{0}, \mathit{msg})$. After unfolding the syntactic sugar for $\rightarrow$ we obtain the formula $\square \neg \mathit{snd}(\mathit{src}, \mathit{msg}) \vee \diamondsuit_{[0,6)}\, \mathit{rcv}(\mathsf{0}, \mathit{msg})$. We explain the labeling for the variables *msg*. The atomic

subformulas $snd(src, msg)$ and $rcv(0, msg)$ are labeled DF_{msg}. The subformula $\neg snd(src, msg)$ is labeled DT_{msg}. The label DT_{msg} propagates through the operator $\vee$ and through the temporal operator $\Box$, so $\Box \neg snd(src, msg) \vee \Diamond_{[0,6)} rcv(0, msg)$ is labeled DT_{msg}. The labeling for the variable *src* is analogous and $\Box \neg snd(src, msg) \vee \Diamond_{[0,6)} rcv(0, msg)$ is labeled DT_{src}.

Example 3.5. Consider the formula $\Box\, ssh\_login(c, s) \rightarrow \Diamond_{[0,25)}\, ssh\_logout(c, s)$. After unfolding the syntactic sugar for $\rightarrow$ we obtain the formula $\Box \neg ssh\_login(c, s) \vee \Diamond_{[0,25)}\, ssh\_logout(c, s)$. We explain the labeling for the variable c. The atomic subformulas $ssh\_login(c, s)$ and $ssh\_logout(c, s)$ are labeled DF_c. $\neg ssh\_login(c, s)$ is labeled DT_c. The label DT_c propagates through the operator $\vee$ and then through the temporal operator $\Box$, so $\neg ssh\_login(c, s) \vee \Diamond_{[0,25)}\, ssh\_logout(c, s)$. and $\Box \neg ssh\_login(c, s) \vee \Diamond_{[0,25)}\, ssh\_logout(c, s)$ are labeled DT_c. The labeling for the variable s is analogous and $\Box \neg ssh\_login(c, s) \vee \Diamond_{[0,25)}\, ssh\_logout(c, s)$ is labeled DT_s.

Theorem 3.2. *For all formulas ϕ and all variables $x \in V$, if the derivation rules shown in Figure 1 assign the label DT_x, DF_x, or DE_x to ϕ then ϕ is in the set DT_x, DF_x, or DE_x, respectively.*

Theorem 3.2 establishes the correctness of our labeling rules. The proof, which uses induction on the size of the derivation tree, is in Appendix A.1. Here, we explain just the most representative rules. In the explanations, we consider formula satisfaction only for valuations of the slicing variable with values outside of the slicing set. For ease of exposition, we do not explicitly state this in every sentence.

The first line in Figure 1 shows rules for predicate symbols. If the predicate symbol contains as a parameter the slicing variable x, then it will not be satisfied on a data slice for values outside of the slicing set, it is therefore in DF_x. If the predicate symbol does not contain as a parameter the slicing variable x, then it will evaluate on the slice exactly as it would evaluate on the original log and hence it is in DE_x.

The next line shows rules for the other atomic formulas. The formulas $t \approx t'$ and $t \prec t'$ are in DE_x because their evaluation depends only on the valuation, so they evaluate in the same way on a data slice as they evaluate on the original log. The formula *true* is syntactic sugar for $\exists y.\, y \approx y$. This formula is always satisfied, so it is in DT_x. It also evaluates in the same way on a data slice as it evaluates on the original log: it is satisfied. Therefore, it is also in DE_x.

The third line shows rules for negation. Memberships in the sets DT_x and DF_x are swapped: if a formula is satisfied on the slice then its negation will not be satisfied on the slice and vice versa for the set DF_x. If a formula is in DE_X then it is satisfied on the slice whenever it is satisfied on the original log. The negation of this formula will also be satisfied on the slice whenever it is satisfied on the original log, so the negation is also in DE_x.

The next two lines show rules for disjunction. If one of the disjunction operands is always satisfied (in DT_x) then the disjunction is also always satisfied and is in DT_x. If neither of the operands is ever satisfied (in DF_x) then so is the disjunction.

If both operands of the disjunction are satisfied on the slice whenever they are satisfied on the original log (in DE_x) then so is the disjunction and it is in DE_x.

Finally, we explain the rules for the operator S in the second to last line in Figure 1. Consider the formula $\phi \,\mathsf{S}_I\, \psi$. If ψ is always satisfied (in DT_x) then independently of ϕ the formula $\phi \,\mathsf{S}_I\, \psi$ is also satisfied and is in DT_x. If ψ is never satisfied (in DF_x) then independently of ϕ the formula $\phi \,\mathsf{S}_I\, \psi$ cannot be satisfied. In this case it is in DF_x. If both ϕ and ψ are satisfied on the slice whenever they are satisfied on the original log (in DE_x) then so is $\phi \,\mathsf{S}_I\, \psi$ and it is in DE_x.

We next show that no spurious violations exist in a data slice for a formula from the sets DE and DT. We use this result subsequently in Theorem 3.3 to show that in this case the data slicer does not need to tighten the restrictions.

Lemma 3.2. *Let ζ be a formula, $x \in V$ a variable that does not overlap with another variable in ζ, $(\bar{\mathcal{D}}, \bar{\tau})$ a temporal structure, $S \subseteq \mathbb{D}$ a slicing set that is valid for $(\bar{\mathcal{D}}, \bar{\tau})$, and $(\bar{\mathcal{P}}, \bar{\tau})$ the (S, x, ζ)-slice of $(\bar{\mathcal{D}}, \bar{\tau})$. If the formula ζ is in DE_x or DT_x, then for all $i \in \mathbb{N}$ and all valuations v with $v(x) \notin S$, it holds that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \zeta$ implies $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models \zeta$.*

Proof. If ζ is in DT_x, then it follows from $v(x) \notin S$ that $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models \zeta$. Because the consequence of the implication is satisfied, it follows trivially that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \zeta$ implies $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models \zeta$.

If ζ is in DE_x, then it follows from $v(x) \notin S$ that $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models \zeta$ iff $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \zeta$. □

Theorem 3.3. *Let ζ be a formula, $x \in V$ a variable that does not overlap with another variable in ζ, $(\bar{\mathcal{D}}, \bar{\tau})$ a temporal structure, $S \subseteq \mathbb{D}$ a slicing set that is valid for $(\bar{\mathcal{D}}, \bar{\tau})$, and $(\bar{\mathcal{P}}, \bar{\tau})$ the (S, x, ζ)-slice of $(\bar{\mathcal{D}}, \bar{\tau})$. If the formula ζ is in DE_x or DT_x, then $(\bar{\mathcal{P}}, \bar{\tau})$ is (D, T)-sound for $(\bar{\mathcal{D}}, \bar{\tau})$ and ζ, where (D, T) is a non-restrictive restriction.*

Proof. The theorem follows directly from Lemma 3.2. □

Finally, we show that membership in the sets DT_x, DF_x, or DE_x is undecidable. We first establish Lemma 3.3 that we use in the undecidability proof.

Lemma 3.3. *If a formula ϕ does not contain the variable $x \in V$ then ϕ is in the set DE_x.*

Proof. We proceed by induction on the structure of the formula ϕ. The base case consists of the atomic formulas $t \prec t'$, $t \approx t$, and $r(t_1, \ldots, t_{\iota(r)})$. Since ϕ does not contain the variable x, x is not in $\{t_1, \ldots, t_{\iota(r)}\}$ and hence all atomic formulas can be labeled with DE_x.

The step case follows straightforwardly from the fact that all atomic subformulas are labeled with DE_x and from the fact that, according to the labeling rules in Figure 1, all operators propagate the label DE_x when all their operands are labeled with DE_x. Therefore, ϕ can be labeled with DE_x. It follows from Theorem 3.2 that ϕ is in the set DE_x. □

Theorem 3.4 establishes undecidability of set membership. The theorem follows from the undecidability of the tautology problem for FOTL formulas, that is, MFOTL formulas that do not have any metric constraints, established in [9] in Lemma B.4.

Theorem 3.4. *Given a formula ϕ and a variable x, it is undecidable whether ϕ is in the set* DT_x, DF_x, *or* DE_x.

Proof. First, we show that membership in the set DT_x is undecidable for an MFOTL formula ϕ and a variable $x \in V$. Without loss of generality, we restrict ourselves to FOTL formulas. Given a FOTL formula ϕ, we pick a variable $x \in V$ that does not occur in ϕ and proceed by reducing the problem of deciding whether ϕ is a tautology to deciding whether ϕ is in the set DT_x. To this end, we show that ϕ is a tautology iff $\phi \vee \bullet \mathit{true}$ is in the set DT_x.

We first show the direction from left to right. $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \bullet \mathit{true}$ and hence $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \phi \vee \bullet \mathit{true}$, for every $i \in \mathbb{N}$ with $i > 0$, temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$, and valuation v. For $i = 0$, it follows from ϕ being a tautology that $(\bar{\mathcal{D}}, \bar{\tau}, v, 0) \models \phi$ and hence $(\bar{\mathcal{D}}, \bar{\tau}, v, 0) \models \phi \vee \bullet \mathit{true}$, for every temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$ and valuation v. Because these temporal structures include the (S, x, ζ)-slices of any temporal structure, for all slicing sets S and all formulas ζ, and the valuations include all valuations with $v(x) \notin S$, ϕ is in the set DT_x.

We show the direction from right to left. That is, we show that if ϕ is not a tautology then $\phi \vee \bullet \mathit{true}$ is not in the set DT_x. From ϕ not being a tautology it follows that there is a temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$ and a valuation v such that $(\bar{\mathcal{D}}, \bar{\tau}, v, 0) \not\models \phi$. From Lemma 3.3 it follows that ϕ is in the set DE_x. Therefore $(\bar{\mathcal{D}}', \bar{\tau}', v, 0) \not\models \phi$, where $(\bar{\mathcal{D}}', \bar{\tau}')$ is the (S, x, ζ)-slice of $(\bar{\mathcal{D}}, \bar{\tau})$, for some slicing set S with $S \subsetneq \mathbb{D}$ and formula ζ of which ϕ is a subformula. Since the variable x does not occur in ϕ, it follows that $(\bar{\mathcal{D}}', \bar{\tau}', v[x \leftarrow\!\shortmid\, d], 0) \not\models \phi$, for every $d \in \mathbb{D} \setminus S$. There is at least one such value d because $S \subsetneq \mathbb{D}$. From $(\bar{\mathcal{D}}', \bar{\tau}', v[x \leftarrow\!\shortmid\, d], 0) \not\models \bullet \mathit{true}$ it follows that $(\bar{\mathcal{D}}', \bar{\tau}', v[x \leftarrow\!\shortmid\, d], 0) \not\models \phi \vee \bullet \mathit{true}$. Therefore $\phi \vee \bullet \mathit{true}$ is not in the set DT_x.

To show that membership in the set DF_x is undecidable for an MFOTL formula ϕ and a variable $x \in V$, we reduce the problem of deciding whether ϕ is a tautology to deciding whether $\neg(\phi \vee \bullet \mathit{true})$ is in the set DF_x, for a variable $x \in V$ that does not occur in ϕ. The proof is analogous to the case DT_x.

Finally, we show that membership in the set DE_x is undecidable for an MFOTL formula ϕ and a variable $x \in V$. Without loss of generality, we restrict ourselves to FOTL formulas. Given a FOTL formula ϕ, we pick a variable $x \in V$ and a predicate symbol of arity 1, $p \in R$, that are not used in ϕ. We proceed by reducing the problem of deciding whether ϕ is a tautology to deciding whether $\phi \vee p(x) \vee \bullet \mathit{true}$ is in the set DE_x.

We first show the direction from left to right. $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \bullet \mathit{true}$ and hence $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \phi \vee p(x) \vee \bullet \mathit{true}$, for every $i \in \mathbb{N}$ with $i > 0$, temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$, and valuation v. For $i = 0$, it follows from ϕ being a tautology that $(\bar{\mathcal{D}}, \bar{\tau}, v, 0) \models \phi$, for every temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$ and valuation v. As a consequence, $(\bar{\mathcal{D}}, \bar{\tau}, v, 0) \models \phi \vee p(x) \vee \bullet \mathit{true}$. Hence, $\phi \vee p(x) \vee \bullet \mathit{true}$ is in the set DE_x.

We show the direction from right to left. That is, we show that if ϕ is not a tautology then $\phi \vee p(x) \vee \bullet\, \mathit{true}$ is not in the set DE_x. From ϕ not being a tautology it follows that there is a temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$ and a valuation v such that $(\bar{\mathcal{D}}, \bar{\tau}, v, 0) \not\models \phi$. Let the temporal structure $(\bar{\mathcal{D}}', \bar{\tau}')$ be like $(\bar{\mathcal{D}}, \bar{\tau})$ except that $p^{\mathcal{D}_0} = \{d\}$, for some $d \in \mathbb{D}$. It follows that $(\bar{\mathcal{D}}', \bar{\tau}', v[x \leftarrow d], 0) \models p(x)$ and hence $(\bar{\mathcal{D}}', \bar{\tau}', v[x \leftarrow d], 0) \models \phi \vee p(x) \vee \bullet\, \mathit{true}$. Let S be a slicing set with $d \notin S$. Then $(\bar{\mathcal{D}}'', \bar{\tau}'', v[x \leftarrow d], 0) \not\models p(x)$, where $(\bar{\mathcal{D}}'', \bar{\tau}'')$ is the $(S, x, \phi \vee p(x) \vee \bullet\, \mathit{true})$-slice of $(\bar{\mathcal{D}}', \bar{\tau}')$. From Lemma 3.3 it follows that ϕ is in the set DE_x. Therefore, it follows from $(\bar{\mathcal{D}}, \bar{\tau}, v, 0) \not\models \phi$ that $(\bar{\mathcal{D}}', \bar{\tau}', v, 0) \not\models \phi$ and $(\bar{\mathcal{D}}'', \bar{\tau}'', v, 0) \not\models \phi$. From x not being used in ϕ it follows that $(\bar{\mathcal{D}}'', \bar{\tau}'', v[x \leftarrow d], 0) \not\models \phi$. Finally, from $(\bar{\mathcal{D}}'', \bar{\tau}'', v[x \leftarrow d], 0) \not\models \bullet\, \mathit{true}$ it follows that $(\bar{\mathcal{D}}'', \bar{\tau}'', v[x \leftarrow d], 0) \not\models \phi \vee p(x) \vee \bullet\, \mathit{true}$. Since $(\bar{\mathcal{D}}', \bar{\tau}', v[x \leftarrow d], 0) \models \phi \vee p(x) \vee \bullet\, \mathit{true}$, but $(\bar{\mathcal{D}}'', \bar{\tau}'', v[x \leftarrow d], 0) \not\models \phi \vee p(x) \vee \bullet\, \mathit{true}$, the formula $\phi \vee p(x) \vee \bullet\, \mathit{true}$ is not in the set DE_x. □

4 Slicing Time

In this section, we present slicers that slice temporal structures in their temporal dimension. We call them time slicers and they produce time slices. Formally, the temporal structure $(\bar{\mathcal{D}}', \bar{\tau}')$ is a *time slice* of the temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$ 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 2.1 such that $r^{\mathcal{D}'_i} = r^{\mathcal{D}_{s(i)}}$, for all $r \in R$ and $i \in [0, \ell)$.

4.1 Time Slicer

In the following, we slice logs by time. For a formula ϕ, we determine a time range of a log that is sufficient to evaluate a formula on a single time point of the log. The time range depends on the temporal operators and their attached intervals. The log is then split by a time slicer into slices according to this time range. Each slice can be monitored separately and in parallel.

We proceed as follows. In Definition 4.1 we show how we determine the relevant time range for a formula. We formalize the slicing of a log by time in Definition 4.2 and in Definition 4.3 we formalize the time slicer.

We extend the notation for intervals over $\mathbb{N}$ to denote 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 $\mathbb{Z}$:

- $I \Cup J := K$, where K is the smallest interval with $I \subseteq K$ and $J \subseteq K$.
- $I \oplus J := \{i + j \mid i \in I \text{ and } j \in J\}$.

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

- $[0, 0]$*, 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 $\bullet_{[a,b)}\, \psi$.*

- $[0, b) \uplus \big([a, b) \oplus \mathrm{RI}(\psi)\big)$, *if* ϕ *is of the form* $\bigcirc_{[a,b)} \psi$.
- $(-b, 0] \uplus \big((-b, 0] \oplus \mathrm{RI}(\psi)\big) \uplus \big((-b, -a] \oplus \mathrm{RI}(\chi)\big)$, *if* ϕ *is of the form* $\psi \,\mathsf{S}_{[a,b)}\, \chi$.
- $[0, b) \uplus \big([0, b) \oplus \mathrm{RI}(\psi)\big) \uplus \big([a, b) \oplus \mathrm{RI}(\chi)\big)$, *if* ϕ *is of the form* $\psi \,\mathsf{U}_{[a,b)}\, \chi$.

We give intuition for Definition 4.1. The relative interval of ϕ specifies a range of timestamps. To evaluate ϕ on a particular time point, it is sufficient to consider all time point whose time stamp falls within this range. The range is relative to the timestamp of the evaluated time point. Timestamps in the future are indicated with a positive value and timestamps in the past are indicated with a negative value.

The intuition about the relative intervals for a formula is as follows. Atomic formulas only depend on the current time point. Therefore, it is sufficient to consider time points with equal timestamps. Formulas of the form $\neg\psi$, $\exists x.\, \psi$, and $\psi \vee \chi$ depend only on the time points required for their subformulas. Analogously, we only need to consider time stamps in the intervals of the subformulas. Formulas of the form $\bigcirc_I \psi$ depend on the time points whose timestamps fall into the interval needed by the subformula ψ, shifted by the interval I. Furthermore, 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 formulas of the form $\psi \,\mathsf{U}_I\, \chi$, 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 into 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 timestamps fall within the range of I, so we need to consider the relative interval of this subformula shifted by $[a, b)$.

Formulas of the form $\bullet_I \psi$ and $\psi \,\mathsf{S}_I\, \chi$, which include past operators, are treated similarly to formulas with the corresponding future operators. However, the relative intervals are mirrored over 0, with negative values indicating that past time points are relevant to evaluate the formula.

Definition 4.2. *Let* $(\bar{\mathcal{D}}, \bar{\tau})$ *be a temporal structures and* $T \subseteq \mathbb{Z}$ *an interval. A* T-slice *of* $(\bar{\mathcal{D}}, \bar{\tau})$ *is a 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* $\tau'_\ell \notin T$.

Figure 2 illustrates Definition 4.2, 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

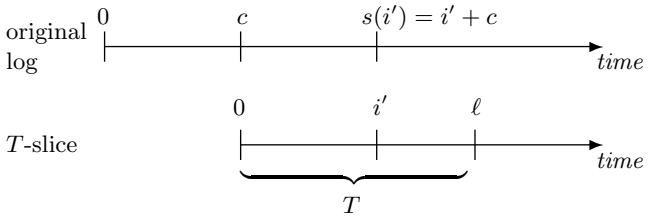

Figure 2. Illustration of a T-slice.

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 $(\bar{\mathcal{D}}, \bar{\tau})$.

Definition 4.3. *A* time slicer $\mathfrak{t}_{\phi,(I^k)_{k\in K}}$ *for the formula* ϕ *and family of intervals* $(I^k)_{k\in K}$ *is a function that takes as input a temporal structure* $(\bar{\mathcal{D}}, \bar{\tau})$ *and a restriction* (D, T)*. It returns a family of temporal structures* $(\bar{\mathcal{D}}^k, \bar{\tau}^k)_{k\in K}$ *and a family of restrictions* $(D^k, T^k)_{k\in K}$*, where* $(\bar{\mathcal{D}}^k, \bar{\tau}^k)$ *is a* $\big((I^k \cap T) \oplus \mathrm{RI}(\phi)\big)$*-slice of* $(\bar{\mathcal{D}}, \bar{\tau})$ *and* $(D^k, T^k) = (D, I^k \cap T)$*, for each* $k \in K$*.*

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

Theorem 4.1. *The time slicer* $\mathfrak{t}_{\phi,(I^k)_{k\in K}}$ *is a slicer for the formula* ϕ *if* $\bigcup_{k\in K} I^k = \mathbb{N}$.

The proof of Theorem 4.1 has the same structure as the proof of Theorem 3.1. The main lemma, which establishes the soundness and completeness of the slices, is proven by induction over the formula structure, as the corresponding Lemma 3.1. In contrast to Lemma 3.1, the arguments for the atomic subformulas are trivial and the arguments for the non-atomic subformulas are complex. However, the proof needs additional machinery. Therefore, along with proof of Theorem 4.1, it is in Appendix B.

For $\mathrm{RI}(\phi)$ that extends beyond 0, the slices must partially overlap. Since the monitor has to inspect those overlapping parts more than once (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.

So how would we split a large log into time slices? Any set of time slices that satisfies the condition in Theorem 4.1 will do, but, as illustrated by Example 4.1, the choice can influence the overhead of monitoring the slices.

Example 4.1. Consider the formula $\Box p \to \blacklozenge_{[0,15)}\, q$ and assume a log, where the timestamps are given as days. We have that $\mathrm{RI}(p \to \blacklozenge_{[0,15)}\, q) = (-15, 0]$. To evaluate the formula over the given log we can split the log into time slices that are equivalent with the original log over 1-week periods. In addition to the 1-week equivalent period, each time slice includes the 14 days prior to the 1-week equivalent period. Each time point would be monitored three times. Namely, once when monitoring the 1-week equivalent period and in each of the two slices when monitoring the next two week periods. If we split the log into time slices that are

equivalent with the original log over 4-weeks periods then half of the time points are monitored once and half are monitored twice. This longer period produces less overhead for monitoring. However, less parallelization of the monitoring process is possible.

4.2 Filtering Time Points

After removing tuples from relations in a temporal structure via a data slicer, there may be many time points with just empty relations. Removing these empty time points can noticeably speed up the monitoring. The *empty-time-point filter* removes such time points (Definition 4.6). However, the filtered temporal structure is not sound and complete for some formulas. We identify a fragment for which it can be safely used (Theorem 4.3).

Definition 4.4. *Let* $(\bar{\mathcal{D}}, \bar{\tau})$ *be a temporal structure. The time point* $i \in \mathbb{N}$ *is* empty *if* $r^{\mathcal{D}_i} = \emptyset$*, for every predicate symbol* r*, and* non-empty *otherwise.*

Definition 4.5. *The temporal structure* $(\bar{\mathcal{D}}', \bar{\tau}')$ *is the* empty-time-point-filtered slice *of the temporal structure* $(\bar{\mathcal{D}}, \bar{\tau})$ *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 non-empty 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 4.5 is uniquely determined in both cases. We make a case distinction in the above definition because if there are only finitely many non-empty 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 monitor always only a finite prefix of a temporal structure from which we remove the empty time points. We assume here that the suffix of the temporal structure contains infinitely many non-empty time points.

Definition 4.6. *The* empty-time-point filter $\mathfrak{f}'_\phi$ *for the formula* ϕ *is a function that takes as input a temporal structure* $(\bar{\mathcal{D}}, \bar{\tau})$ *and a restriction* (D, T)*. It returns a family that contains only the temporal structure* $(\bar{\mathcal{D}}', \bar{\tau}')$ *and a family that contains only the restriction* (D, T)*, 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 4.3. To define the fragment, we use the sets FT, FF, and FE. 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 the addition or removal of empty time points in the temporal structure.

$$\frac{}{r(t_1,\ldots,t_{\iota(r)}) : \mathsf{FF}} \qquad \frac{}{\mathit{true} : \mathsf{FT}} \qquad \frac{\phi : \mathsf{FF}}{\neg\phi : \mathsf{FT}} \qquad \frac{\phi : \mathsf{FT}}{\neg\phi : \mathsf{FF}}$$

$$\frac{\phi : \mathsf{FT}}{\phi \vee \psi : \mathsf{FT}} \qquad \frac{\psi : \mathsf{FT}}{\phi \vee \psi : \mathsf{FT}} \qquad \frac{\phi : \mathsf{FF} \quad \psi : \mathsf{FF}}{\phi \vee \psi : \mathsf{FF}}$$

$$\frac{\phi : \mathsf{FT}}{\exists y.\, \phi : \mathsf{FT}} \qquad \frac{\phi : \mathsf{FF}}{\exists y.\, \phi : \mathsf{FF}}$$

Figure 3. Labeling Rules 1 (Empty-time-point Filter)

Definition 4.7. *The sets* FT, FF, *and* FE *are defined such that for every formula* ϕ *the following holds:*

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

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

holds, for all temporal structures $(\bar{\mathcal{D}}, \bar{\tau})$ *and* $(\bar{\mathcal{D}}', \bar{\tau}')$, *all valuations* v, *and all non-empty 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. The fragments are defined in terms of a labeling algorithm that assigns the labels FT, FF, and FE to formulas. The fragments are sound but incomplete 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, not every formula in one of the sets is assigned to the corresponding label. The algorithm labels the atomic subformulas of a formula and propagates the labels bottom-up to the formula's root. It first propagates the labels FF, FT according to the labeling rules are shown in Figure 3. Afterwards, it assigns the label FE according to the rules in Figure 4. Note that syntactic sugar must be unfolded before applying the rules. We use the expression $\phi : \ell$ as shorthand for the formula ϕ being labeled with the label ℓ. We show the soundness of our labeling rules in Theorem 4.2.

Theorem 4.2. *For all formulas* ϕ, *if the derivation rules shown in Figures 3 and 4 assign the label* FT, FF, *or* FE *to* ϕ *then* ϕ *is in the set* FT, FF, *or* FE, *respectively.*

Theorem 4.3. *The empty-time-point filter* $\mathfrak{f}'_\phi$ *is a slicer for the formula* ϕ *if the formula* ϕ *is in both* FE *and* FT.

$$\frac{}{r(t_1,\ldots,t_{\iota(r)}) : \mathsf{FE}} \qquad \frac{}{t \approx t' : \mathsf{FE}} \qquad \frac{}{t \prec t' : \mathsf{FE}}$$

$$\frac{\phi : \mathsf{FE}}{\neg\phi : \mathsf{FE}} \qquad \frac{\phi : \mathsf{FE}}{\exists x.\, \phi : \mathsf{FE}} \qquad \frac{\phi : \mathsf{FE} \quad \psi : \mathsf{FE}}{\phi \vee \psi : \mathsf{FE}}$$

$$\frac{\phi : \mathsf{FE} \quad \phi : \mathsf{FT} \quad \psi : \mathsf{FE} \quad \psi : \mathsf{FF}}{\phi\, \mathsf{S}_I\, \psi : \mathsf{FE}}$$

$$\frac{\phi : \mathsf{FE} \quad \phi : \mathsf{FT} \quad \psi : \mathsf{FE} \quad \psi : \mathsf{FF}}{\phi\, \mathsf{U}_I\, \psi : \mathsf{FE}}$$

$$\frac{\phi : \mathsf{FE} \quad \phi : \mathsf{FF}}{\blacklozenge_I \Diamond_J \phi : \mathsf{FE}}\ 0 \in I \cap J \qquad \frac{\phi : \mathsf{FE} \quad \phi : \mathsf{FF}}{\Diamond_I \blacklozenge_J \phi : \mathsf{FE}}\ 0 \in I \cap J$$

$$\frac{\phi : \mathsf{FE} \quad \phi : \mathsf{FT}}{\blacksquare_I \Box_J \phi : \mathsf{FE}}\ 0 \in I \cap J \qquad \frac{\phi : \mathsf{FE} \quad \phi : \mathsf{FT}}{\Box_I \blacksquare_J \phi : \mathsf{FE}}\ 0 \in I \cap J$$

Figure 4. Labeling Rules 2 (Empty-time-point Filter)

The proofs for Theorem 4.2 and for Theorem 4.3 are similar to the proofs of Theorem 3.2 and Theorem 3.1 and are in Appendix B.2 and Appendix B.3, respectively.

It follows from Theorem 4.3 that the empty-time-point filter is a slicer for all formulas that can be labeled with FE and FT.

The empty-time-point filter is implemented in the monitoring tool MONPOLY. The tool checks whether the monitored formula can be labeled with FE and FT and if so, then it applies the filter to the input temporal structure by default unless disabled by command line flags.

5 The Google Case Study

In this section, we describe our deployment of compliance monitoring in a case study with Google. We first explain the scenario, the monitored policies, and the logging and monitoring setup. Afterwards, we present our experimental results.

5.1 Setting

Scenario. In our case study, we consider a setting of over 35,000 computers that are used both within Google while connected directly to the corporate network and outside of Google, accessing Google's network from remote unsecured networks. These computers are used to access other computers and sensitive resources.

To minimize the risk of unauthorized access to sensitive resources, access control mechanisms are used. In particular, computers must obtain authentication tokens via a tool, which we refer to as AUTH. The validity of the token is limited in time. Furthermore, the Secure Shell protocol (SSH) is used to remotely login into other computers. 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 achieve this, every computer regularly starts an update tool, which we refer to as UPD, connects to the central server to download the latest centrally managed configuration, and attempts to reconfigure and update itself. To prevent over-burdening the configuration server, if the computer has successfully updated its configuration recently then the update tool UPD aborts and does not attempt a connection to the server.

Policies. We give below a set of policies specifying restrictions on the authorization process, SSH sessions, and the update process. Afterwards, we formalize them in MFOTL. The computers in our case study are intended to comply with these policies. However, due to misconfiguration, server outages, hardware failures, etc. this is not always the case. The policies are as follows:

***P1*:** Entering the credentials with the tool AUTH must take at least 1 second. The motivation is that authentication with the tool AUTH must 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.

***P5*:** Each computer must be updated at least once every 3 days unless it is turned off or not connected to the corporate network.

***P6*:** 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.

***P7*:** If the tool UPD aborts the update process claiming that the computer was successfully updated recently, then there must have been a successful update within the last 24 hours.

Formalization. We formalize the above policies in MFOTL. The signature contains the predicate symbols *alive*, *net*, *upd_start*, *upd_connect*, *upd_success*, *upd_skip*, *auth*, *ssh_login*, and *ssh_logout*. Their interpretations at a time point in a temporal structure are as follows:

- $alive(c : string)$: The computer c is running. This event is generated at least once every 20 minutes when the computer c is running. For busy computers, we limit this to at most 2 events every 5 minutes.
- $net(c : string)$: The computer c is connected to the corporate network. This event is generated at least once every 20 minutes when the computer c is connected to the corporate network. For busy computers, we rate limit this event to at most 1 every 5 minutes.
- $auth(c : string, t : integer)$: 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.
- $upd\_start(c : string)$: The tool UPD started on the computer c.

Table 1. Policy formalizations in MFOTL

Policy	MFOTL formalization
P1	$\Box \forall c.\, \forall t.\, auth(c,t) \rightarrow 1000 \prec t$
P2	$\Box \forall c.\, \forall t.\, auth(c,t) \rightarrow \blacklozenge_{[0,3d]} \Diamond_{[0,0]}\, upd\_success(c)$
P3	$\Box \forall c.\, \forall s.\, ssh\_login(c,s) \wedge \big(\Diamond_{[1min,20min]}\, net(c) \wedge \Box_{[0,1d]} \blacksquare_{[0,0]}\, net(c) \rightarrow \Diamond_{[1min,20min]}\, net(c)\big) \rightarrow \Diamond_{[0,1d)} \blacklozenge_{[0,0]}\, ssh\_logout(c,s)$
P5	$\Box \forall c.\, net(c) \wedge \big(\Diamond_{[10min,20min]}\, net(c)\big) \wedge \big(\blacklozenge_{[1d,2d]}\, alive(c)\big) \wedge \neg\big(\blacklozenge_{[0,3d]} \Diamond_{[0,0]}\, upd\_success(c)\big) \rightarrow \Diamond_{[0,20min)} \blacklozenge_{[0,0]}\, upd\_connect(c)$
P6	$\Box \forall c.\, upd\_connect(c) \wedge \big(\Diamond_{[5min,20min]}\, alive(c)\big) \rightarrow \Diamond_{[0,30min)} \blacklozenge_{[0,0]}\, upd\_success(c) \vee upd\_skip(c)$
P7	$\Box \forall c.\, upd\_skip(c) \rightarrow \blacklozenge_{[0,3d]} \Diamond_{[0,0]}\, upd\_success(c)$

- $upd\_connect(c : string)$: The tool UPD on the computer c connected to the central server and downloaded the latest configuration.
- $upd\_success(c : string)$: The tool UPD successfully updated the local configuration and applied security patches on the computer c.
- $upd\_skip(c : string)$: The tool UPD on the computer c terminated because it believes that the computer was successfully updated recently.
- $ssh\_login(c : string, s : string)$: 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.
- $ssh\_logout(c : string, s : string)$: An SSH session with identifier s to the computer c was closed.

Our formalization of the policies is shown in Table 1. We explain the less obvious aspects of the formalization. We use the variable c to represent a computer, the variable s to represent an SSH session, and the variable t to represent the time it takes a user the enter authentication credentials. In the policy *P3*, we assume that if a computer is disconnected from the corporate network, then the SSH session is closed. In the policy *P5*, because of the subformula $\blacklozenge_{[1d,2d]}\, alive(c)$ we only consider computers that have recently been used. This is an approximation to not consider newly installed computers. Similarly, we only require an update of a computer if it is connected to the network for a certain amount of time. In the policy *P6*, since computers 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 in 5 to 20 minutes after downloading the new configuration from the central server.

The actions performed by the different computers are logged with a timestamp recording when the actions happened. If actions are logged by different computers with the same timestamp, then we do not know the relative ordering of these actions. However, we do not care about the actual ordering of such actions and express this by including the operators $\Diamond_{[0,0]}$, $\blacklozenge_{[0,0]}$, and $\blacksquare_{[0,0]}$ in the formalization of the policies. This makes the formulas collapse-sufficient [9], that

is, all possible orderings of actions logged with an equal timestamp either all satisfy or all violate a policy. We monitor the policies on a collapsed temporal structure, that is, on a temporal structure where structures with an equal timestamp are merged into a single structure. This allows us to omit the operators $\Diamond_{[0,0]}$, $\blacklozenge_{[0,0]}$, and $\blacksquare_{[0,0]}$ from the formulas that we actually monitor.

The monitored formulas representing the policies P1 to P7 are temporal-domain independent and fall within the fragment of MFOTL that the tool MONPOLY handles. After removing the leading $\Box$ operator and making the variable c a free variable, all formulas can be labeled with FE, FT, and DT_c (see Sections 4.2 and 3.3 for details about the labels). Therefore, filtering empty time points and slicing on the variable c do not introduce any spurious violations that would need to be removed from the monitor output in a post-processing step.

Note that we use the constant symbol $\mathsf{1000}$. Since its corresponding value, namely 1,000 ms, does not represent a computer identifier, the slicing carrier sets need not contain this value.

Logs. The relevant computers log locally and upload their logs to a log cluster. These logs consist of entries describing the system events that occurred. Every day, approximately 1 TB of log data is uploaded. Due to their sizes, the logs are stored in a distributed file system spread across a large number of physical computers. In our case study, we restricted ourselves to log data that spans approximately two years. Furthermore, the logs also contain entries that are irrelevant for our policies.

We processed the logs to obtain a temporal structure that consists of the events relevant for our policies. We used regular expression matching to find log entries for extracting the corresponding interpretations of the predicate symbols at each time point in the temporal structure. For example, to determine the elements c in the relations for the predicate symbol *alive*, we considered every logged entry of the computer c and extracted at most two of them every five minutes for the computer c.

To account for the fact that the events are carried out in a concurrent setting, we collapsed the extracted temporal structure [9]. The collapsed temporal structure contains approximately 77.2 million time points and 26 billion log events, i.e., tuples in the relations interpreting the predicate symbols. A breakdown of the numbers of logged events in the collapsed temporal structure by predicate symbols is presented in Table 2. The collapsed temporal structure encoded in a protocol buffer format [20] amounts to approximately 600 MB per day on average and to 0.4 TB for the two years. Protocol buffer formats are widely used within Google and well-supported by the infrastructure that we used.

5.2 Monitoring Setup

We first motivate why a MapReduce framework is suitable to implement our monitoring setup. Afterwards, we describe and evaluate our implementation.

We have not considered other parallelization frameworks beyond MapReduce because MapReduce's performance was sufficient for monitoring the logs in this

Table 2. Log statistics by log event

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

case study. A comparison of the performance of various frameworks for the parallelization of computations is beyond the scope of this article.

Why MapReduce? The logs are too large to be reasonably stored and processed on a single computer. They are stored in a distributed file system where their content is spread across multiple physical computers.

Although we could write a script to split the log into slices and start monitoring processes for the different slices on different computers, existing MapReduce frameworks like Hadoop or the one we used in our case study at Google [14] offer several advantages over such a manual approach. A MapReduce framework automatically allocates the monitoring tasks to different machines, restarts failed tasks, and speculatively starts tasks on multiple computers in order to minimize the time until all tasks complete. It also attempts to minimize the fetching of data over the network by allocating tasks to computers on which the required data is already present.

However, the main advantage of MapReduce frameworks is efficient shuffling, that is, transferring the output of the mappers to the correct reducers and sorting the data for each reducer. We use the shuffling to rearrange how the temporal structure is split when it is stored across multiple computers. Given the size of the log data, we assume that it is initially stored in a distributed file system and spread across multiple computers. The way it is distributed might not correspond to the way that we want to slice the extracted temporal structure. Even if the initial distribution corresponds to one slicing method, different policies may require different slicing methods. We use the sorting in the shuffling phase to ensure that the slices are sorted by timestamps. This is a prerequisite for the correctness of the MFOTL monitoring algorithm.

We use the mappers in MapReduce to split the temporal structure into slices and we use the reducers to check the slices for compliance with the policies. This allows us to reap the benefits of the shuffling phase between map and reduce.

Realization. To monitor the temporal structure, obtained from logged data as described in Section 5.1, for checking compliance with the policies, we used the MONPOLY tool [8] together with Google's MapReduce framework [14]. For each

policy, we used 1,000 computers for slicing and monitoring. We split the temporal structure into 10,000 slices so that each computer had to process 10 slices on average. The decision of using a magnitude more slices than computers follows the recommendation of making the individual map and reduce computations small. In particular, if the monitoring of a slice fails and has to be restarted, less computational power has been wasted.

We implemented only data slicing by the variable c. With Definition 3.1 we obtain a method for constructing the slices in a straight-forward way. Namely, for each time point in the original temporal structure, iterate through the tuples in the relations interpreting predicate symbols and copy only those tuples that satisfy conditions specified in Definition 3.1. The other slicing methods can be implemented in a similar way.

Instead of listing the members of the slicing sets explicitly, we provide a function mapping every interpretation of the variable c to a slicing set. Each slicing set is identified by a natural number between 0 and 9,999. The function applies a variant of the MurmurHash [32] hash function to the variable interpretation, which is a computer identifier, and takes the remainder after division by the desired number of slices. This function leads to a relatively even distribution of the size of slices, as shown in Figure 5. Since the slices obtained in this way were sufficient for our purposes, we did not implement other slicing methods, such as slicing by time.

We explain our use of the MapReduce framework in more detail. The mappers split the relations of the temporal structure into data slices by the variable c and output each element of a relation as a separate structure along with two keys. The primary key indicates the log slice and the secondary key contains the timestamp.

During the shuffling phase, the output of the mappers is passed to the reducers. Because we use the primary key to identify a slice, for each slice, a reducer receives all structures of the slice. The structures are sorted by the timestamp because we use the timestamp as the secondary key.

The reducer collapses the received temporal structure. That is, it merges all structures with the same timestamp into a single structure. Since the structures are already sorted by the timestamp, this can be done in time linear in the slice's length. For each slice, a reducer starts a MONPOLY instance in a child process. It converts the collapsed temporal structure into the MONPOLY format on the fly and pipes it into the MONPOLY process in one thread. In another thread, it reads the output of the MONPOLY process and returns it as the output of the reducer.

Note that due to the way we implemented slicing, empty time points are filtered out from the slices. To further improve efficiency, the MONPOLY tool additionally filters out "irrelevant" tuples with a data filter and any subsequent empty time points with the empty-time-point filter. As explained in Section 5.1, the filtering and slicing does not introduce any spurious violations that must be removed from the monitor output in a post-processing step.

Table 3. Monitor performance

Policy	Runtime [hh:mm]
P1	2:04
P2	2:10
P3	11:56
P5	2:32
P6	2:28
P7	2:13

The output of the reducers in our monitoring setup with MapReduce contains the actual policy violations. Those can be fed into various auditing and monitoring dashboards. They can also be further processed to generate alerts based on statistical anomaly detection or predefined limits, such as on the number of detected violations per time unit.

Evaluation. We evaluate our monitoring approach to show that it is feasible to check compliance of large IT systems and that our monitoring approach scales to large logs. In particular, each of the policies in Table 1 could be monitored within 12 hours on logs as large as 0.4 TB. In the following, we provide details about the distribution of the size of the slices and how the monitoring tool MONPOLY performed.

Figure 5 shows the distribution of the size of log slices in the MONPOLY format used as input for the monitoring tool MONPOLY. On the y-axis is the percentage of slices whose size is smaller or equal to the value on the x-axis. From the figure, we see that the log volume is rather evenly distributed among the slices. The median size of a slice is 61 MB and 90% of the slices have a size of at most 94 MB (93% of at most 100 MB and 99% of at most 135 MB). There are three slices with sizes over 1GB and the largest slice is 1.8 GB. The total log volume, that is the sum of the sizes of all slices, is 626.6 GB. Note that we used the same slicing method for all policies.

Note that the sum of the size 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. Furthermore, we consider the size of the collapsed temporal structure in a protocol buffer format and the size of the slices in the more verbose text-based MONPOLY format.

In Table 3 we show the time it took to monitor each policy using our monitoring setup. [4] For most policies, the monitoring took up to two and a half hours. Monitoring the policy *P3* took almost 12 hours.

We first discuss the time it took to monitor the different slices and then how much memory the monitoring tool used. Table 4 presents details about the

[4] This is the time for the whole MapReduce job. That is, from starting the MapReduce job until the monitor on the last slice has finished and its output has been collected by the corresponding reducer.

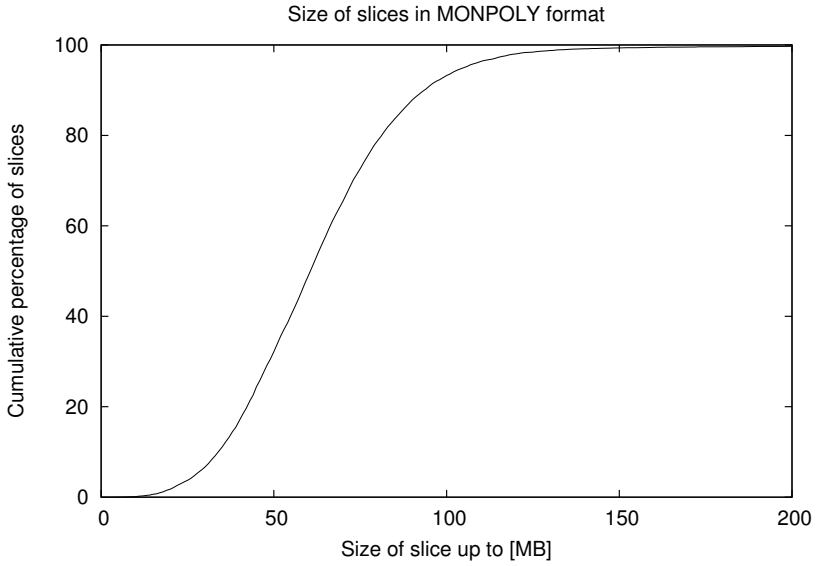

Figure 5. Distribution of the size of log slices

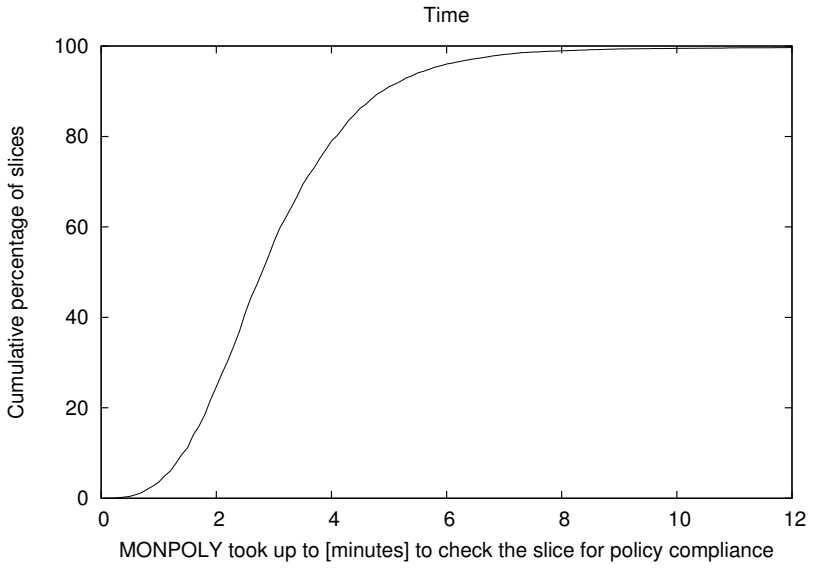

Figure 6. Distribution of time to monitor individual slices for policy *P3*

Table 4. Monitor performance per slice

Policy	Runtime			Memory used	
	median [sec]	max [hh:mm]	sum [days]	median [MB]	max [MB]
P1	169	0:46	21.4	6.1	6.1
P2	170	0:51	21.4	6.1	10.3
P3	170	10:40	22.7	7.1	510.2
P5	169	1:06	21.3	9.2	13.1
P6	168	1:01	21.3	6.1	6.1
P7	168	0:48	21.1	6.1	7.1

monitoring of the individual slices. For the policy *P3*, Figure 6 shows on the y-axis the percentage of slices for which the monitoring time is within the limit on the x-axis. Curves for the other policies are not shown as they are almost identical to *P3*. The almost identical curves indicate that for most slices the monitoring time does not differ across policies. Independently of the monitored policy, the median time to monitor a slice is around 3 minutes, 90% of the slices can be monitored within 5 minutes each (99% within 8.2 minutes), and the sum of the time to monitor each slice is between 21 and 23 days. However, there is a difference in monitoring the few large slices. For each policy except for *P3*, the maximum time to monitor a slice is between 46 minutes and 66 minutes. For policy *P3*, 30 slices took longer than one hour to monitor each with the largest 1.8 GB slice taking almost 11 hours. An additional burden of policy *P3* is the nesting of multiple temporal operators, in particular three of them. This burden exhibits itself especially on the large slices.

Independently of the monitored policies, the median amount of memory needed by the monitoring tool is between 6 MB and 10 MB and 90% of the slices do not require more than 14 MB (99% are within 35 MB). For all policies except for *P3*, the monitor never needed more than 13 MB of memory. The few large slices present outliers for the policy *P3*, where memory usage grew up to 510 MB. As Figure 7 demonstrates these outliers represent a very small proportion of the slices.

From the analysis of the time it took to monitor the individual slices, we see that we end up waiting for a few "stragglers", that is, slices that take significantly longer to monitor than other slices. There are several options to deal with these slices. We can stop the monitor after a timeout and ignore the slice and any policy violations on the slice. Note that the monitoring of the other slices and the validity of violations found on them would not be affected. Alternatively, we can split the large slice into smaller slices, either in advance before we start monitoring or after a timeout when monitoring the large slice. For policy *P3*, we can slice further by the variable c. As the formalization of the policy *P3* can be labeled DT_s we can also slice by the variable s. Finally, we can slice by time.

Given the sheer size of the logs, the time spent monitoring them is reasonable. After implementing a solution to deal with the stragglers, even faster monitoring

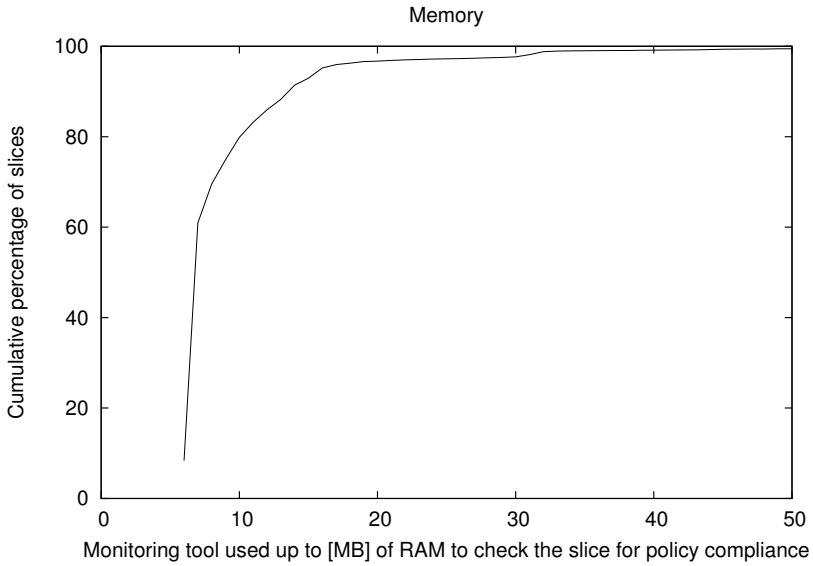

Figure 7. Distribution of memory used to monitor individual slices for policy *P3*

times can be achieved by using more computers for monitoring. In our setup, every computer monitored ten slices on average. We could increase the number of computers tenfold so that each computer would monitor only a single slice. For even larger logs and more computers available to do the monitoring, we could further increase the number of possible slices by doing both slicing by data on the variable c and slicing by time. The MapReduce framework makes it trivial to add more computers.

Due to the sensitive nature of the logs, we do not report on the detected violations of the monitored policies. However, we remark that monitoring a large population of computers and aggregating the violations found by the monitor can be used to identify systematic policy violations as well as policy violations due to reconfiguring the system setup. An example of the former is not letting a computer update after the weekend before using it to access sensitive resources on Monday (policy *P2*). As an example of the latter, the monitoring turned out to be helpful in determining when the update process was not operating as expected for certain type of computers during a specific time period. This information can be helpful for identifying seemingly unrelated changes in the configuration of other components in the IT infrastructure.

6 Related Work

Temporal logics have been widely used to formalize and analyze security and privacy policies. For example, Zhang et al. [33] formalize the UCON_{ABC} model [27]

for usage control in the temporal logic of actions [25]. The Obligation Specification Language, presented by Hilty et al. [23], includes temporal operators. Barth et al. [7] present a framework for specifying privacy policies in a first-order temporal logic and DeYoung et al. [17] show how parts of the HIPAA and GLBA policies can be formalized in this framework. The focus of these works is primarily on formalizing policies whereas we focus on monitoring compliance with policies, in particular on handling large amounts of log data.

Based on formally specified policies, various algorithms have been presented for monitoring system behavior with a single monitor [6, 8, 10, 18, 21, 22]. However, these approaches do not scale to large logs due to a lack of parallelization.

Similar to our approach, Barre et al. [5] monitor parts of a log in parallel and independently of the other log parts with a MapReduce framework. While we split the log into multiple slices and evaluate the whole 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 in a single reducer. Therefore, the reducer becomes a bottleneck and the scalability of their approach remains unclear. Furthermore, their case study with a log consisting of less than five million log tuples, monitored on a single computer, is rather small and they evaluated their approach only for a propositional temporal logic, which is limited in expressing realistic policies.

Roşu and Chen [29] present a general extension for different property specification languages and associated monitoring algorithms where they add parameters to logged events. This allows them to monitor parts of a log in parallel and independently of the other log parts. For monitoring, the log with events containing parameters is split into slices, with one slice for each parameter value in case of a single parameter, and one slice for each combination of values for different parameters in case multiple parameters are used. The slices are processed by the original monitoring algorithm unaware of parameters. In contrast to our work, they do not use a MapReduce framework. Neither do they explain how their monitoring approach can be implemented to run in parallel. We note that a parametric extension of a propositional temporal logic is less expressive than a first-order extension, such as MFOTL used in our work. Roşu and Chen also describe a case study with up to 155 million log events, all monitored on a single computer. This is orders of magnitude smaller than the log monitored in the Google case study.

Since most IT systems are distributed, events are initiated and carried out locally, that is, by their system components. As a consequence, logs are generated distributively. We collect the logged events and redistribute them to the monitors such that each monitor obtains the necessary events. In contrast, the monitoring approaches presented by Sen et al. [31], Bauer and Falcone [11], and Zhou et al. [34]

directly monitor the system components and their monitors communicate their observations. These approaches work in an online setting and the communication is needed because not every monitor necessarily observes all log events that it needs to evaluate the policy. In contrast, our approach is restricted to an offline setting and we use the MapReduce framework to provide each monitor with the necessary log events. As a result of the communication in these approaches, one slow monitor can slow down all other monitors. Furthermore, it remains unclear how the communicating monitors cope with a monitor that crashes. In our approach, a crashed monitor is automatically restarted by the MapReduce framework. In the following, we discuss further differences between those distributed monitoring approaches and our approach.

To specify properties, Sen et al.'s [31] distributed monitoring approach uses a propositional past linear-time distributed temporal logic with epistemic operators [28] that reflect the local knowledge of a process. The semantics of temporal operators in this logic is defined with respect to a partial ordering, the causal ordering [24] commonly used in distributed systems. Their logic therefore does not allow one to express temporal constraints on events that are not causally related. Policies are defined with respect to the local view point of a single process and checked with respect to these view points, using the last known states of other processes. Thus two processes can reach different verdicts as to whether a property is satisfied or violated. This is in contrast to our approach where the semantics of the temporal operators is defined with respect to a total ordering based on the time stamps of the logged events.

Bauer and Falcone [11] present a distributed monitoring algorithm for propositional future linear-time temporal logic where monitors are distributed throughout the system and exchange partially evaluated formulas between each other. They assume that observations of the system are done simultaneously in lock-step. Each monitor evaluates the subformulas for which it can observe the relevant system actions and the monitoring progresses in lock-step, requiring synchronization between the monitors, so one slow monitor slows down all other monitors. This is in contrast to our approach, where different slices can be monitored independently and at different speeds.

Zhou et al. [34] present a distributed monitoring framework aimed at monitoring network protocols. Instead of using a temporal logic to specify properties, they rely on a Datalog-like language with additional support for temporal constraints. The monitors are executed together with the network protocols.

Instead of formalizing policies in a temporal logic and using dedicated monitoring algorithms for checking system compliance, one can use SQL-like languages to express policies as database queries and evaluate the queries with a database managements system (DBMS). Techniques used to parallelize the evaluation of such queries in parallel DBMSs, see [15, 26], are related our work. In the terminology of the DBMS community [26], the slicing of logs can be seen as vertical fragmentation, that is, partitioning the data in a database table by rows of the table. Monitoring log slices in parallel for compliance with a single formula corresponds to intra-query parallelism in DBMSs, that is, evaluating a query in

parallel on multiple computers. Here, two orthogonal techniques are used in parallel DBMSs: intra-operator and inter-operator parallelism. With intra-operator parallelism, the same operator is evaluated in parallel on subsets of the data. With inter-operator parallelism, different operators are evaluated in parallel. The monitoring of slices corresponds to intra-operator parallelism. Inter-operator parallelism in our setting would correspond to evaluating different subformulas of the monitored formula in parallel, similar to the approach of Barre et al. [5], but we evaluate the whole formula in parallel on different log slices. In contrast to DBMSs, the results of evaluating a formula on different slices can easily be combined by concatenating them, as long as the restrictions corresponding to the slices do not overlap. DBMSs use more complex algorithms [16] to merge the results of evaluating the join operator on subsets of the data.

We have restricted ourselves to compliance checking in an offline setting. One of the reasons for this restriction is that the MapReduce framework is inherently offline: the Reduce phase can start only after all mappers have finished processing all of their inputs. Condie et al. [12] overcome this limitation and present an extension of MapReduce where the reducers process the output of mappers while the mappers are still running. It needs to be further investigated whether this extension of MapReduce allows for a scalable deployment of our monitoring approach in an online setting. Note that only data slicing can be used in an online setting, time slicing is inherently restricted to an offline setting.

In the context of an online setting, a disadvantage of DBMSs is their inherent restriction to an offline setting: they must first import the complete data set before they can evaluate any queries on it. This restriction of DBMSs is overcome by complex event processing systems. These systems continuously evaluate queries expressed in SQL-like languages [4] on rapidly evolving data streams in an online setting. We refer to [13] for a survey on complex event processing. Since the specification languages employed by these systems are not based on temporal logics, a direct comparison is difficult and it remains to be seen if and how we can benefit from work in this domain.

7 Conclusion

We presented a solution for compliance checking of IT systems, where the behavior of the system agents is monitored offline and checked against security policies. Our case study shows the scalability of our solution. The scalability is rooted in parallelizing the monitoring process, for which we provide a theoretical framework and an algorithmic realization within the MapReduce framework.

The MapReduce framework is particularly well suited for implementing the monitoring process: First, it allows us to efficiently reorganize a huge log, which contains the actions carried out by the system agents, into slices. Second, it allocates and distributes the computations for monitoring the slices, accounting for the available computational resources, the location of the logged data, failures, and so on. Third, 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 theoretical framework allows one to slice in multiple dimensions by composing different slicing methods. It remains as future work to exploit and evaluate the possibilities to obtain a larger number of smaller slices that are equally expensive to monitor. It also remains to be seen how to utilize and extend the framework for obtaining a scalable solution for checking system compliance online.

Acknowledgements. The authors would like to thank Thomas Dübendorfer, Eric Grosse, Sebastiaan Indesteege, Loren Kohnfelder, Susanne Landers, Jan Monsch, Christoph Raupacher, Mario Strasser, and Harald Wagener for helpful discussions and support in performing the case study.

References

1. The Health Insurance Portability and Accountability Act of 1996 (HIPAA), 1996. Public Law 104-191.
2. Sarbanes-Oxley Act of 2002 (SOX), 2002. Public Law 107-204.
3. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases: The Logical Level*. Addison Wesley, 1994.
4. A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: semantic foundations and query execution. *VLDB Journal*, 15(2):121–142, 2006.
5. 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'12)*, volume 7687 of *Lect. Notes Comput. Sci.*, pages 184–198. Springer, 2013.
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. A. Barth, A. Datta, J. C. Mitchell, and H. Nissenbaum. Privacy and contextual integrity: Framework and applications. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, SP '06, pages 184–198, Washington, DC, USA, 2006. IEEE Computer Society.
8. D. Basin, M. Harvan, F. Klaedtke, and E. Zălinescu. MONPOLY: Monitoring usage-control policies. In *Proceedings of the 2nd International Conference on Runtime Verification (RV'11)*, volume 7186 of *Lect. Notes Comput. Sci.*, pages 360–364. Springer, 2012.
9. D. Basin, M. Harvan, F. Klaedtke, and E. Zălinescu. Monitoring data usage in distributed systems. *IEEE Trans. Software Eng.*, to appear.
10. 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 *Leibiz International Proceedings in Informatics (LIPIcs)*, pages 49–60. Schloss Dagstuhl - Leibniz Center for Informatics, 2008.
11. A. Bauer and Y. Falcone. Decentralised LTL monitoring. In *Proceedings of the 18th International Symposium on Formal Methods (FM)*, volume 7436 of *Lect. Notes Comput. Sci.*, pages 85–100. Springer, 2012.

12. T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears. MapReduce online. In *Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 313–328. USENIX Association, 2010.
13. G. Cugola and A. Margara. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.*, 44(3), 2012.
14. J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *6th Symposium on Operating System Design and Implementation (OSDI 2004)*, pages 137–150, 2004.
15. D. DeWitt and J. Gray. Parallel database systems: the future of high performance database systems. *Commun. ACM*, 35(6):85–98, June 1992.
16. D. J. DeWitt and R. H. Gerber. Multiprocessor hash-based join algorithms. In *Proceedings of the 11th international conference on Very Large Data Bases - Volume 11*, VLDB '85, pages 151–164. VLDB Endowment, 1985.
17. H. DeYoung, D. Garg, L. Jia, D. Kaynar, and A. Datta. Experiences in the logical specification of the hipaa and glba privacy laws. In *Proceedings of the 9th annual ACM workshop on Privacy in the electronic society*, WPES '10, pages 73–82, New York, NY, USA, 2010. ACM.
18. 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.
19. 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, New York, NY, USA, 2011. ACM Press.
20. Google. Protocol Buffers: Googles Data Interchange Format, 2013. [Online; accessed 6-June-2013].
21. 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.
22. S. Hallé and R. Villemaire. Runtime enforcement of web service message contracts with data. *IEEE Trans. Serv. Comput.*, 5(2):192–206, 2012.
23. M. Hilty, A. Pretschner, D. A. Basin, C. Schaefer, and T. Walter. A policy language for distributed usage control. In *ESORICS*, pages 531–546, 2007.
24. L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.
25. L. Lamport. The temporal logic of actions. *ACM Trans. Program. Lang. Syst.*, 16(3):872–923, May 1994.
26. M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*. Springer, 3rd edition, 2011.
27. J. Park and R. Sandhu. The uconabc usage control model. *ACM Trans. Inf. Syst. Secur.*, 7(1):128–174, Feb. 2004.
28. R. Ramanujam. Local knowledge assertions in a changing world. In *Proceedings of the Sixth Conference on Theoretical Aspects of Rationality and Knowledge (TARK)*, pages 1–14. Morgan Kaufmann, 1996.
29. G. Roşu and F. Chen. Semantics and algorithms for parametric monitoring. *Log. Method. Comput. Sci.*, 8(1):1–47, 2012.
30. 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.

31. K. Sen, A. Vardhan, G. Agha, and G. Roşu. Efficient decentralized monitoring of safety in distributed systems. In *Proceedings of the 26th International Conference on Software Engineering (ICSE)*, pages 418–427. IEEE Computer Society, 2004.
32. Wikipedia. MurmurHash — Wikipedia, the free encyclopedia, 2013. [Online; accessed 14-April-2013].
33. X. Zhang, F. Parisi-Presicce, R. Sandhu, and J. Park. Formal model and policy specification of usage control. *ACM Trans. Inf. Syst. Secur.*, 8(4):351–387, Nov. 2005.
34. W. Zhou, O. Sokolsky, B. T. Loo, and I. Lee. *DMaC*: Distributed monitoring and checking. In *Proceedings of the 9th International Workshop on Runtime Verification (RV)*, volume 5779 of *Lecture Notes in Computer Science*, pages 184–201. Springer, 2009.

A Additional Proof Details for Slicing Data

In this appendix, we provide additional proof details for the assertions made in Section 3.

A.1 Theorem 3.2

We prove Theorem 3.2 by induction on the size of the derivation tree assigning label ℓ 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 readability, and without loss of generality, we fix the slicing variable x and the temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$. Consider formula ϕ of the form:

- $r(t_1, \ldots, t_{\iota(r)})$ where $x \in \{t_1, \ldots, t_{\iota(r)}\}$. For every formula ζ where x does not overlap with another variable inζ and ϕ is a subformula of ζ, every slicing set $S \subseteq \mathbb{D}$ that is valid for $(\bar{\mathcal{D}}, \bar{\tau})$ and ϕ, every $i \in \mathbb{N}$, and every valuations v with $v(x) \notin S$, let $(\bar{\mathcal{P}}, \bar{\tau})$ be the (S, x, ζ)-slice of $(\bar{\mathcal{D}}, \bar{\tau})$. For the tuple $(v(t_1), \ldots, v(t_{\iota(r)}))$, none of the Conditions (a)–(c) in Definition 3.1 are satisfied. In particular, Condition (a) is not satisfied because $v(x) \notin S$. Condition (b) is not satisfied because the variable x does not overlap with any other variable in ζ. Finally, Condition (c) is not satisfied because $v(x) \notin S$ and S is a valid slicing set for $(\bar{\mathcal{D}}, \bar{\tau})$. It follows that $(v(t_1), \ldots, v(t_{\iota(r)}) \notin r^{\mathcal{P}_i}$, $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \not\models r(t_1, \ldots, t_{\iota(r)})$, and hence $r(t_1, \ldots, t_{\iota(r)})$ is in DF_x.
- $r(t_1, \ldots, t_{\iota(r)})$ where $x \notin \{t_1, \ldots, t_{\iota(r)}\}$. For every formula ζ where x does not overlap with another variable in ζ and ϕ is a subformula of ζ, every slicing set $S \subseteq \mathbb{D}$ that is valid for $(\bar{\mathcal{D}}, \bar{\tau})$ and ϕ, every $i \in \mathbb{N}$, and every valuations v with $v(x) \notin S$, let $(\bar{\mathcal{P}}, \bar{\tau})$ be the (S, x, ζ)-slice of $(\bar{\mathcal{D}}, \bar{\tau})$. We show that $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models r(t_1, \ldots, t_{\iota(r)})$ iff $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models r(t_1, \ldots, t_{\iota(r)})$, so that $r(t_1, \ldots, t_{\iota(r)})$ is in DE_x.
  We first show the implication from right to left. Suppose that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models r(t_1, \ldots, t_{\iota(r)})$, i.e., $(v(t_1), \ldots, v(t_{\iota(r)}) \in r^{\mathcal{D}_i}$. Because the variable x is not among the terms $t_1, \ldots, t_{\iota(r)}$, those terms consist only of constants and of variables other than x. It follows that at least one of the Conditions (a)–(c) in Definition 3.1 is satisfied for every such tuple $(v(t_1), \ldots, v(t_{\iota(r)}))$. Hence, $(v(t_1), \ldots, v(t_{\iota(r)}) \in r^{\mathcal{P}_i}$ and thus $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models r(t_1, \ldots, t_{\iota(r)})$.
  We show the implication from left to right by contradiction. Suppose that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models r(t_1, \ldots, t_{\iota(r)})$, i.e., $(v(t_1), \ldots, v(t_{\iota(r)}) \notin r^{\mathcal{D}_i}$. It follows that $(v(t_1), \ldots, v(t_{\iota(r)}) \notin r^{\mathcal{P}_i}$ and $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \not\models r(t_1, \ldots, t_{\iota(r)})$.
- $t \prec t'$. For every formula ζ where x does not overlap with another variable in ζ and ϕ is a subformula of ζ, every slicing set $S \subseteq \mathbb{D}$ that is valid for $(\bar{\mathcal{D}}, \bar{\tau})$ and ϕ, every $i \in \mathbb{N}$, and every valuations v with $v(x) \notin S$, let $(\bar{\mathcal{P}}, \bar{\tau})$ be the (S, x, ζ)-slice of $(\bar{\mathcal{D}}, \bar{\tau})$. Satisfaction of $t \prec t'$ depends only on the valuation, so it trivially follows that $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models t \prec t'$ iff $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models t \prec t'$. Therefore, $t \prec t'$ is in DE_x.
- $t \approx t'$. This case is analogous to the previous one.

– *true*. The subformula *true* is syntactic sugar for $\exists y.\, y \approx y$. Note that we can always take a fresh variable y without affecting the meaning of the formula ϕ. Therefore, the assumption made at the beginning of the section that variables are quantified at most once holds.

  For every formula ζ where x does not overlap with another variable in ζ and ϕ is a subformula of ζ, every slicing set $S \subseteq \mathbb{D}$ that is valid for $(\bar{\mathcal{D}}, \bar{\tau})$ and ϕ, every $i \in \mathbb{N}$, and every valuations v with $v(x) \notin S$, let $(\bar{\mathcal{P}}, \bar{\tau})$ be the (S, x, ζ)-slice of $(\bar{\mathcal{D}}, \bar{\tau})$. Trivially, $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models \mathit{true}$ and hence *true* is in DT_x. It also trivially holds that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \mathit{true}$ and hence $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models \mathit{true}$ iff $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \mathit{true}$. Therefore, *true* is also in DE_x.

– $\neg\psi$. For every formula ζ where x does not overlap with another variable in ζ and ϕ is a subformula of ζ, every slicing set $S \subseteq \mathbb{D}$ that is valid for $(\bar{\mathcal{D}}, \bar{\tau})$ and ϕ, every $i \in \mathbb{N}$, and every valuations v with $v(x) \notin S$, let $(\bar{\mathcal{P}}, \bar{\tau})$ be the (S, x, ζ)-slice of $(\bar{\mathcal{D}}, \bar{\tau})$.

  We first show how the label DF_x is propagated. Suppose that ψ is labeled DF_x. From the induction hypothesis it follows that ψ is in DF_x so that $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \not\models \psi$ and hence $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models \neg\psi$. Therefore, $\neg\psi$ is in DT_x.

  Next, we show how the label DT_x is propagated. Suppose that ψ is labeled DT_x. From the induction hypothesis it follows that ψ is in DT_x so that $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models \psi$ and hence $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \not\models \neg\psi$. Therefore, $\neg\psi$ is in DF_x.

  Finally, we show how the label DE_x is propagated. Suppose that ψ is labeled DE_x. From the induction hypothesis it follows that ψ is in DE_x so that $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models \psi$ iff $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \psi$. Therefore $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models \neg\psi$ iff $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \neg\psi$ and $\neg\psi$ is in DE_x.

– $\psi \vee \chi$. For every formula ζ where x does not overlap with another variable in ζ and ϕ is a subformula of ζ, every slicing set $S \subseteq \mathbb{D}$ that is valid for $(\bar{\mathcal{D}}, \bar{\tau})$ and ϕ, every $i \in \mathbb{N}$, and every valuations v with $v(x) \notin S$, let $(\bar{\mathcal{P}}, \bar{\tau})$ be the (S, x, ζ)-slice of $(\bar{\mathcal{D}}, \bar{\tau})$.

  Suppose that ψ is labeled DT_x. It follows from the induction hypothesis that ψ is in DT_x so that $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models \psi$ and hence $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models \psi \vee \chi$. Therefore, $\psi \vee \chi$ is in DT_x.

  Suppose that χ is labeled DT_x. It follows from the induction hypothesis that χ is in DT_x so that $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models \chi$ and hence $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models \psi \vee \chi$. Therefore, $\psi \vee \chi$ is in DT_x.

  Suppose that ψ and χ are labeled DF_x. It follows from the induction hypothesis that ψ and χ are in DF_x so that $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \not\models \psi$ and $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \not\models \chi$. Therefore, $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \not\models \psi \vee \chi$ and $\psi \vee \chi$ is in DF_x.

  Suppose that ψ and χ are labeled DE_x. It follows from the induction hypothesis that ψ and χ are in DE_x, so that $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models \psi$ iff $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \psi$ and that $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models \chi$ iff $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \chi$. Therefore, $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models \psi \vee \chi$ iff $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \psi \vee \chi$ and $\psi \vee \chi$ is in DE_x.

– $\exists y.\, \psi$ where $x \neq y$. For every formula ζ where x does not overlap with another variable in ζ and ϕ is a subformula of ζ, every slicing set $S \subseteq \mathbb{D}$ that is valid for $(\bar{\mathcal{D}}, \bar{\tau})$ and ϕ, every $i \in \mathbb{N}$, and every valuations v with $v(x) \notin S$, let $(\bar{\mathcal{P}}, \bar{\tau})$ be the (S, x, ζ)-slice of $(\bar{\mathcal{D}}, \bar{\tau})$.

Suppose that ψ is labeled DT_x. It follows from the induction hypothesis that ψ is in DT_x and hence $(\bar{\mathcal{P}}, \bar{\tau}, v', i) \models \psi$ for all valuations v' with $v'(x) \notin S$. Therefore, $(\bar{\mathcal{P}}, \bar{\tau}, v[y \hookleftarrow e], i) \models \psi$, for all $e \in \mathbb{D}$. Because $\mathbb{D}$ is non-empty, it follows that $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models \exists y.\, \psi$ and $\exists y.\, \psi$ is in DT_x.

Suppose that ψ is labeled DF_x. It follows from the induction hypothesis that ψ is in DF_x and hence $(\bar{\mathcal{P}}, \bar{\tau}, v', i) \not\models \psi$ for all valuations v'. Therefore, there is no $e \in \mathbb{D}$ with $(\bar{\mathcal{P}}, \bar{\tau}, v[y \hookleftarrow e], i) \models \psi$. It follows that $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \not\models \exists y.\, \psi$ and $\exists y.\, \psi$ is in DF_x.

Finally, suppose that ψ is labeled DE_x. It follows from the induction hypothesis that ψ is in DE_x and hence $(\bar{\mathcal{P}}, \bar{\tau}, v', i) \models \psi$ iff $(\bar{\mathcal{D}}, \bar{\tau}, v', i) \models \psi$ for all valuations v'. Therefore, $(\bar{\mathcal{P}}, \bar{\tau}, v[y \hookleftarrow e], i) \models \psi$ for some $e \in \mathbb{D}$ iff $(\bar{\mathcal{D}}, \bar{\tau}, v[y \hookleftarrow e], i) \models \psi$ for some $e \in \mathbb{D}$. It follows that $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models \exists y.\, \psi$ iff $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \exists y.\, \psi$ and $\exists y.\, \psi$ is in DE_x.

- $\bullet_I\, \psi$ For every formula ζ where x does not overlap with another variable in ζ and ϕ is a subformula of ζ, every slicing set $S \subseteq \mathbb{D}$ that is valid for $(\bar{\mathcal{D}}, \bar{\tau})$ and ϕ, every $i \in \mathbb{N}$, and every valuations v with $v(x) \notin S$, let $(\bar{\mathcal{P}}, \bar{\tau})$ be the (S, x, ζ)-slice of $(\bar{\mathcal{D}}, \bar{\tau})$.

  Suppose that ψ is labeled DF_x. For $i \in \mathbb{N}$ with $i > 0$ and $\tau_i - \tau_{i-1} \in I$ it follows from the induction hypothesis that ψ is in DF_x, so that $(\bar{\mathcal{P}}, \bar{\tau}, v, i-1) \not\models \psi$ and thus $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \not\models \bullet_I\, \psi$. For $i > 0$ with $\tau_i - \tau_{i-1} \notin I$ and for $i = 0$ it follows from the definition of the operator $\bullet_I$ that $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \not\models \bullet_I\, \psi$. Therefore, $\bullet_I\, \psi$ is in DF_x.

  Suppose that ψ is labeled DE_x. For $i \in \mathbb{N}$ with $i > 0$ and $\tau_i - \tau_{i-1} \in I$ it follows from the induction hypothesis that ψ is in DE_x, so that $(\bar{\mathcal{P}}, \bar{\tau}, v, i-1) \models \psi$ iff $(\bar{\mathcal{D}}, \bar{\tau}, v, i-1) \models \psi$ and hence $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models \bullet_I\, \psi$ iff $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \bullet_I\, \psi$. For $i > 0$ with $\tau_i - \tau_{i-1} \notin I$ and for $i = 0$ it follows from the definition of the operator $\bullet_I$ that $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \not\models \bullet_I\, \psi$ and $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \bullet_I\, \psi$. It follows trivially that $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \not\models \bullet_I\, \psi$ iff $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \bullet_I\, \psi$. Therefore, $\bullet_I\, \psi$ is in DE_x.
- $\circ_I\, \psi$. This case is analogous to the previous one.
- $\psi\, \mathsf{S}_I\, \chi$. For every formula ζ where x does not overlap with another variable in ζ and ϕ is a subformula of ζ, every slicing set $S \subseteq \mathbb{D}$ that is valid for $(\bar{\mathcal{D}}, \bar{\tau})$ and ϕ, every $i \in \mathbb{N}$, and every valuations v with $v(x) \notin S$, let $(\bar{\mathcal{P}}, \bar{\tau})$ be the (S, x, ζ)-slice of $(\bar{\mathcal{D}}, \bar{\tau})$.

  Suppose that χ is labeled DT_x. It follows from the induction hypothesis that χ is in DT_x, so that $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models \chi$ and thus $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models \psi\, \mathsf{S}_I\, \chi$. Therefore, $\psi\, \mathsf{S}_I\, \chi$ is in DT_x.

  Suppose that χ is labeled DF_x. It follows from the induction hypothesis that χ is in DF_x, so that $(\bar{\mathcal{P}}, \bar{\tau}, v, j) \not\models \chi$ for all $j \in \mathbb{N}$ and thus $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \not\models \psi\, \mathsf{S}_I\, \chi$. Therefore, $\psi\, \mathsf{S}_I\, \chi$ is in DF_x.

  Suppose that ψ and χ are labeled DE_x. We show that $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models \psi\, \mathsf{S}_I\, \chi$ iff $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \psi\, \mathsf{S}_I\, \chi$. $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models \psi\, \mathsf{S}_I\, \chi$ iff for some $j \leq i$, $\tau_i - \tau_j \in I$, $(\bar{\mathcal{P}}, \bar{\tau}, v, j) \models \chi$, and $(\bar{\mathcal{P}}, \bar{\tau}, v, k) \models \psi$ for all k with $j < k \leq i$. It follows from the induction hypothesis and from ψ and χ being labeled DE_x that ψ and χ are in DE_x. Therefore, $(\bar{\mathcal{P}}, \bar{\tau}, v, j) \models \chi$ iff $(\bar{\mathcal{D}}, \bar{\tau}, v, j) \models \chi$ and thus

$(\bar{\mathcal{P}}, \bar{\tau}, v, k) \models \psi$ iff $(\bar{\mathcal{D}}, \bar{\tau}, v, k) \models \psi$. It follows that $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models \psi \,\mathsf{S}_I\, \chi$ iff $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \psi \,\mathsf{S}_I\, \chi$. Therefore, the formula $\psi \,\mathsf{S}_I\, \chi$ is in DE_x.
– $\psi \,\mathsf{U}_I\, \chi$. This case is analogous to the previous one. □

B Additional Proof Details for Slicing Time

In this appendix, we provide additional proof details for the assertions made in Section 4.

B.1 Theorem 4.1

We first state a definition about overlapping temporal structures and several lemmas that we use in the proof of Theorem 4.1.

Definition B.1. *Let $I \subseteq \mathbb{Z}$ be an interval, $c \in \mathbb{N}$, $i \in \mathbb{N}$, and $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ be temporal structures. $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are* (I, c, i)-overlapping *if the following conditions hold:*

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

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 4.2. 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.

Lemma B.1 establishes that time slices overlap and Lemma B.2 shows that if temporal structures overlap for an interval I, then they also overlap for other time points in I and for sub-intervals of I.

Lemma B.1. *Let $T \subseteq \mathbb{N}$ and $I \subseteq \mathbb{Z}$ be intervals, $(\bar{\mathcal{D}}, \bar{\tau})$ a temporal structure, and $(\bar{\mathcal{D}}', \bar{\tau}')$ a $(T \oplus I)$-slice of $(\bar{\mathcal{D}}, \bar{\tau})$, where $c \in \mathbb{N}$ is the value used by the s-function in Definition 4.2. $(\bar{\mathcal{D}}', \bar{\tau}')$ and $(\bar{\mathcal{D}}, \bar{\tau})$ are (I, c, i)-overlapping, for all $i \in \mathbb{N}$ with $\tau_i \in T$.*

Proof. We first show that Condition 1 in Definition B.1 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 4.2 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 B.2. *Let $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ 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 B.1 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 B.1 holds. □

Lemma B.3 establishes that 0 is included in the relative interval of every formula. This guarantees that the satisfaction relation in Lemma B.4 is defined.

Lemma B.3. *For every formula ϕ, it holds that $0 \in \mathrm{RI}(\phi)$.*

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 $r(t_1, \ldots, t_{\iota(r)})$, where t, t', and $t_1, \ldots, t_{\iota(r)}$ are variables or constants. It follows trivially from Definition 4.1 that $0 \in \mathrm{RI}(\phi)$.
- $\neg\psi$ and $\exists x.\, \psi$. It follows from the inductive hypothesis that $0 \in \mathrm{RI}(\psi)$ and hence $0 \in \mathrm{RI}(\phi)$.
- $\psi \vee \chi$. It follows from the inductive hypothesis that $0 \in \mathrm{RI}(\psi)$ and $0 \in \mathrm{RI}(\chi)$. Therefore, $0 \in \mathrm{RI}(\psi) \uplus \mathrm{RI}(\chi)$ and hence $0 \in \mathrm{RI}(\phi)$.
- $\bullet_I \psi$, $\bigcirc_I \psi$, $\psi \,\mathsf{S}_I\, \chi$, and $\psi \,\mathsf{U}_I\, \chi$. It follows trivially from Definition 4.1 that $0 \in \mathrm{RI}(\phi)$. □

Lemma B.4. *Let ϕ be a formula and $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ temporal structures. If $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are $(\mathrm{RI}(\phi), c, i)$-overlapping, for some c and i, then for all valuations v it holds that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \phi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i - c) \models \phi$.*

Proof. We prove Lemma B.4 by structural induction on the form of the formula ϕ. Note that for all cases, $(\bar{\mathcal{D}}', \bar{\tau}', v, i - c) \models \phi$ is defined: it follows from Lemma B.3 that $0 \in \mathrm{RI}(\phi)$ and from Condition 1 in Definition B.1 that $i \geq c$ and hence $i - c \in \mathbb{N}$. We have the following cases:

- $t \approx t'$, where t and t' are variables or constants. 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 and t' are variables or constants. This case is similar to the previous one.
- $r(t_1, \ldots, t_{\iota(r)})$, where $t_1, \ldots, t_{\iota(r)}$ are variables or constants. Since $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are $(RI(r(t_1, \ldots, t_{\iota(r)}), c, i)$-overlapping and $0 \in RI(r(t_1, \ldots, t_{\iota(r)}))$, it also follows from Condition 1 in Definition B.1 that $\mathcal{D}_i = \mathcal{D}'_{i-c}$ and hence $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models r(t_1, \ldots, t_{\iota(r)})$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i - c) \models r(t_1, \ldots, t_{\iota(r)})$, for all valuations v.

- $\neg\psi$. $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are $(RI(\neg\psi), c, i)$-overlapping and $RI(\neg\psi) = RI(\psi)$, 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 $(\mathrm{RI}(\psi) \uplus \mathrm{RI}(\chi), c, i)$-overlapping. From $RI(\psi) \subseteq \mathrm{RI}(\psi) \uplus \mathrm{RI}(\chi)$, $RI(\chi) \subseteq \mathrm{RI}(\psi) \uplus \mathrm{RI}(\chi)$, and Lemma B.2 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 $(\bar{\mathcal{D}}, \bar{\tau}, v[x \leftarrow\!\!\shortmid d], i) \models \psi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v[x \leftarrow\!\!\shortmid d], i - c) \models \psi$. 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.
- $\bullet_{[a,b)}\, \psi$. $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are $(\mathrm{RI}(\bullet_{[a,b)}\, \psi), c, i)$-overlapping, where $\mathrm{RI}(\bullet_{[a,b)}\, \psi) = (-b, 0] \uplus \big((-b, -a] \oplus \mathrm{RI}(\psi)\big)$.
  From $0 \in \mathrm{RI}(\bullet_{[a,b)}\, \psi)$ and from Condition 1 in Definition B.1 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 $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \bullet_{[a,b)}\, \psi$, for all valuations v. From Definition B.1 it follows that $c = 0$ and hence $i - c = 0$. Trivially, $(\bar{\mathcal{D}}', \bar{\tau}', v, i - c) \not\models \bullet_{[a,b)}\, \psi$, 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 $\tau_{i-1} - \tau_i \in \mathrm{RI}(\bullet_{[a,b)}\, \psi)$ and from Condition 1 in Definition B.1 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 $\mathrm{RI}(\psi) \subseteq \{\tau_i - \tau_{i-1}\} \oplus \{\tau_{i-1} - \tau_i\} \oplus \mathrm{RI}(\psi) \subseteq \{\tau_i - \tau_{i-1}\} \oplus (-b, -a] \oplus \mathrm{RI}(\psi) \subseteq \{\tau_i - \tau_{i-1}\} \oplus \mathrm{RI}(\bullet_{[a,b)}\, \psi)$ and hence by Lemma B.2 $(\bar{\mathcal{D}}, \bar{\tau})$ 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 $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \bullet_{[a,b)}\, \psi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i - c) \models \bullet_{[a,b)}\, \psi$, for all valuations v.
  2. If $\tau_i - \tau_{i-1} \notin [a, b)$ then trivially $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \bullet_{[a,b)}\, \psi$, for all valuations v. From Definition B.1 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 $(\bar{\mathcal{D}}', \bar{\tau}', v, i - c) \not\models \bullet_{[a,b)}\, \psi$, 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 B.1 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 $(\bar{\mathcal{D}}', \bar{\tau}', v, i - c) \not\models \bullet_{[a,b)}\, \psi$, 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 $\mathrm{RI}(\bigcirc_{[a,b)} \psi) = [0, b) \uplus \big([a, b) \oplus \mathrm{RI}(\psi)\big)$.
From $0 \in \mathrm{RI}(\bigcirc_{[a,b)} \psi)$ and from Condition 1 in Definition B.1 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 B.1 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 B.2 $(\bar{\mathcal{D}}, \bar{\tau})$ 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 B.1 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 $\mathrm{RI}(\psi \; \mathsf{S}_{[a,b)} \; \chi) = (-b, 0] \uplus \big((-b, 0] \oplus \mathrm{RI}(\psi)\big) \uplus \big((-b, -a] \oplus \mathrm{RI}(\chi)\big)$.
Note that $0 \in \mathrm{RI}(\psi \; \mathsf{S}_{[a,b)} \; \chi)$, so from Condition 1 in Definition B.1 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 $\mathrm{RI}(\chi) \subseteq \{\tau_i - \tau_j\} \oplus \{\tau_j - \tau_i\} \oplus \mathrm{RI}(\chi) \subseteq \{\tau_i - \tau_j\} \oplus (-b, -a] \oplus \mathrm{RI}(\chi) \subseteq \{\tau_i - \tau_j\} \oplus \mathrm{RI}(\psi \; \mathsf{S}_{[a,b)} \; \chi)$ and $j \geq c$. By Lemma B.2, $(\bar{\mathcal{D}}, \bar{\tau})$ 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 B.2 $(\bar{\mathcal{D}}, \bar{\tau})$ 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 B.1 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 B.1 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 B.1 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 B.1 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 4.1 by showing that a time slicer $\mathfrak{t}_{\phi,(I^k)_{k\in K}}$ satisfies the criteria (S1)-(S3) in Definition 2.5 if $\bigcup_{k\in K} I^k = \mathbb{N}$ and therefore is a slicer for the formula ϕ.

For (S1), we show that the family $(D^k, T^k)_{k\in K}$ fulfills the conditions (R1)–(R3) in Definition 2.4: (R1) follows from $D^k = D$, for each $k \in K$. (R2) follows from $\bigcup_{k\in K} T^k = \bigcup_{k\in K} (I^k \cap T) \subseteq \bigcup_{k\in K} T = T$. (R3) follows from the assumption $\bigcup_{k\in K} I^k = \mathbb{N}$ and the equalities $D^k = D$ and $T^k = I^k \cap T$, for each $k \in K$.

(S2) and (S3) follow from Lemma B.1 and from Lemma B.4. □

B.2 Theorem 4.2

We first prove Theorem 4.2 for the labels FT and FF. We proceed by induction on the size of the derivation tree assigning label ℓ 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 readability, and without loss of generality, we already fix the temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$, a time point $i \in \mathbb{N}$, and a valuation v.

A formula $r(t_1, \ldots, t_{r(l)})$ is labeled FF. If i is an empty time point in $\mathcal{D}$ then clearly $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models r(t_1, \ldots, t_{r(l)})$.

The formula *true* is labeled FT. Trivially, $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \mathit{true}$.

The other rules propagate the assigned label through the non-temporal operators. The correctness of these rules can be seen by plugging the true and false values into the semantic definitions of these operators.

Next, we prove Theorem 4.2 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 $r(t_1, \ldots, t_{r(l)})$, $t \approx t'$, 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 non-empty 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. $(\bar{\mathcal{D}}', \bar{\tau}', v, i') \models \phi$ implies $(\bar{\mathcal{D}}, \bar{\tau}, v, s(i')) \models \phi$, and
2. $(\bar{\mathcal{D}}, \bar{\tau}, v, s(i')) \models \phi$ implies $(\bar{\mathcal{D}}', \bar{\tau}', v, i') \models \phi$

– $\phi \,\mathsf{S}_I\, \psi$:
  1. $(\bar{\mathcal{D}}', \bar{\tau}', v, i') \models \phi \,\mathsf{S}_I\, \psi$ implies $(\bar{\mathcal{D}}, \bar{\tau}, v, s(i')) \models \phi \,\mathsf{S}_I\, \psi$
     From $(\bar{\mathcal{D}}', \bar{\tau}', v, i') \models \phi \,\mathsf{S}_I\, \psi$ 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 $(\bar{\mathcal{D}}', \bar{\tau}', v, k') \models \phi$.
     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 non-empty 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 $(\bar{\mathcal{D}}, \bar{\tau}, v, k) \models \phi$. If it is a non-empty 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 $(\bar{\mathcal{D}}, \bar{\tau}, v, k) \models \phi$. In both cases $(\bar{\mathcal{D}}, \bar{\tau}, v, k) \models \phi$ and therefore $(\bar{\mathcal{D}}, \bar{\tau}, v, s(i')) \models \phi \,\mathsf{S}_I\, \psi$.

2. $(\bar{\mathcal{D}}, \bar{\tau}, v, s(i')) \models \phi \,\mathsf{S}_I\, \psi$ implies $(\bar{\mathcal{D}}', \bar{\tau}', v, i') \models \phi \,\mathsf{S}_I\, \psi$
   From $(\bar{\mathcal{D}}, \bar{\tau}, v, s(i')) \models \phi \,\mathsf{S}_I\, \psi$ 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 $(\bar{\mathcal{D}}, \bar{\tau}, v, k) \models \phi$.
   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 $(\bar{\mathcal{D}}, \bar{\tau}, v, s(i')) \models \phi \,\mathsf{S}_I\, \psi$ it follows that $(\bar{\mathcal{D}}, \bar{\tau}, v, k) \models \phi$. From ϕ being labeled FE it follows that ϕ is in FE and hence $(\bar{\mathcal{D}}', \bar{\tau}', v, k') \models \phi$. Therefore, $(\bar{\mathcal{D}}, \bar{\tau}, v, s(i')) \models \phi \,\mathsf{S}_I\, \psi$.

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

B.3 Theorem 4.3

We first state a lemma that we use in the proof of Theorem 4.3.

Lemma B.5. *Let ϕ be a formula in the intersection of* FE *and* FT*, $(\bar{\mathcal{D}}, \bar{\tau})$ a temporal structure, and $(\bar{\mathcal{D}}', \bar{\tau}')$ the empty-time-point-filtered slice of $(\bar{\mathcal{D}}, \bar{\tau})$). $(\bar{\mathcal{D}}', \bar{\tau}')$ is (D, T)-sound and -complete for $(\bar{\mathcal{D}}, \bar{\tau}, v, 0)$ and ϕ, where (D, T) is a non-restrictive restriction.*

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

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 $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \phi$ it follows that $(\bar{\mathcal{D}}', \bar{\tau}', v, i') \models \phi$.

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

Each time point i in $(\bar{\mathcal{D}}, \bar{\tau})$ is either empty or non-empty. If it is empty, then from ϕ being in FT we know that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \phi$. If it is non-empty then there 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 $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \phi$ it follows that $(\bar{\mathcal{D}}', \bar{\tau}', v, i') \not\models \phi$. □

We prove Theorem 4.3 by showing that the empty-time-point filter satisfies the criteria (S1)-(S3) in Definition 2.5 and therefore is a slicer. (S1) follows trivially because the filter does not modify the restriction (D, T). (S2) and (S3) follow directly from Lemma B.5. □