DISS. ETH NO. 21418

SCALABLE MONITORING OF CONCURRENT SYSTEMS

A dissertation submitted to
ETH ZURICH
for the degree of
Doctor of Sciences

presented by
MATÚŠ HARVAN,
MSc in Computer Science, Jacobs University Bremen
born on November 29, 1983
citizen of Slovak Republic

accepted on the recommendation of
Prof. Dr. David Basin, examiner
Dr. Felix Klaedtke, co-examiner
Prof. Dr. Valtteri Niemi, co-examiner

2013

Abstract

It is a growing concern of companies and end users whether IT systems comply with security policies, which, for example, stipulate how sensitive data must and must not be used. In this thesis, we present a scalable solution for compliance checking of distributed and concurrent systems. Our solution is based on runtime monitoring, where policies are specified in an expressive temporal logic and system actions are logged.

While well-established methods for monitoring linearly-ordered system behavior exist, a major challenge is monitoring distributed and concurrent systems, where actions are locally observed in the different system parts. The observed actions can only be partially ordered while policy compliance may depend on the actions' actual order of appearance. In general, it is intractable to check compliance of partially ordered logs. To overcome the intractability, we identify fragments of our policy specification language for which compliance can be checked efficiently, namely, by monitoring a single representative log in which the observed actions are totally ordered.

Furthermore, to scale our approach to large logs, we parallelize the process of monitoring the logged actions. To this end, we provide a theoretical framework for slicing logs, that is, the reorganization of the logged actions into smaller log parts that can be analyzed independently of each other. Within this framework, we present orthogonal methods for generating such slices and provide means to combine these methods.

With two case studies we show that our fragments are capable of expressing non-trivial policies and that monitoring representative logs is feasible on real-world data. We utilize the MapReduce framework to implement our log slicing solution and demonstrate its feasibility and scalability by successfully monitoring large-scale logs in one of our case studies.

Zusammenfassung

Für Unternehmen sowie für Endbenutzer ist es immer wichtiger, dass in IT-Systemen diverse Sicherheitsregeln eingehalten werden. Diese Regeln legen zum Beispiel fest wie sensible Daten behandelt werden müssen. In dieser Dissertation stellen wir eine skalierbare Lösung vor, um verteilte und nebenläufige Systeme auf die Einhaltung solcher Regeln zu überprüfen. Diese Lösung stützt sich auf eine Überwachung der Systeme, bei der die Regeln in einer temporalen Logik spezifiziert werden und relevante Systemaktionen geloggt werden.

Methoden zur Überprüfung von Logs, in denen die beobachteten Systemaktionen linear geordnet sind, existieren bereits. Es ist allerdings unklar, wie man verteilte und nebenläufige Systeme überprüft, in denen die Systemaktionen in verschiedenen Teilen des Systems lokal beobachtet und geloggt werden. In solchen Fällen können die beobachteten Systemaktionen nur partiell geordnet werden. Die Einhaltung mancher Regeln hängt jedoch von der tatsächlichen Reihenfolge der Aktionen ab. Es ist im Allgemeinen nicht effizient lösbar, partiell geordnete Logs auf Regelverletzungen zu überprüfen, weil es exponentiell viele Linearizierungen geben kann. Daher identifizieren wir Fragmente unserer Regelspezifikationssprache, für die wir die Einhaltung der Regeln effizient überprüfen können. Für Regeln aus diesen Fragmenten reicht es nämlich aus, nur einen einzelnen, total-geordneten Log zu betrachten.

Um grosse Logs überprüfen zu können, parallelisieren wir unseren Überprüfungsansatz. Dafür entwickeln wir Verfahren für die Aufteilung von Logs in kleinere Teile, so dass diese unabhängig voneinander betrachtet und überprüft werden können. Diese Verfahren beinhalten verschiedene Aufteilungsmethoden und erlauben es, diese Methoden miteinander zu kombinieren.

Durch zwei Fallstudien belegen wir, dass nicht-triviale, in der Praxis vorkommende, Regeln in den beschriebenen Fragmenten ausgedrückt werden können und dass Logs aus realen Systemen überprüft werden können. Wir implementieren die Aufteilung von Logs mittels MapReduce und zeigen die praktische Einsetzbarkeit und die Skalierbarkeit unserer Lösung durch erfolgreiche Überprüfung von sehr grossen Logs in einer unserer Fallstudien.

Acknowledgments

This dissertation would not have been possible without the great support that I received from many colleagues, friends, and family members.

First, I thank David Basin for giving me the opportunity of pursuing my dissertation in this great research environment, for his beneficial scientific advice, comments, feedback, and his generous support. I am also deeply indebted to Felix Klaedtke who has guided me with lots of good advice around all the obstacles that I encountered during my PhD. His rigorous and diligent reviews of numerous drafts of my work helped me in making my ideas more precise. During the early stage of my PhD, I also received a lot of guidance from Alexander Pretschner. I am grateful for the guidance I received from Germano Caronni during our collaboration. I also want to express my appreciation to Valtteri Niemi for his flexibility and his willingness to serve as my co-examiner.

Second, I would like to thank all present and past members of the Institute of Information Security at ETH Zurich, who all contributed to a very pleasant working environment. Especially I thank my office mates Christian Dax, Lukas Brügger, and Hubert Ritzdorf for their good company and Michèle Feltz, Simon Meier, Benedikt Schmidt, Petar Tsankov, and Eugen Zălinescu for fruitful discussions.

Third, I would like to thank my parents, Milica and Jozef, and my brother Peter for their unconditional support and their sustained confidence in my abilities.

Contents

1 Introduction

Both public and private companies are increasingly required to check whether the agents of their IT systems, that is, users and processes, comply with security policies. These policies, for example, 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) [HIP96] and financial services must conform to the Sarbanes-Oxley Act (SOX) [SOX02]. 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.

Problem Statement. Although various monitoring approaches have been developed for different policy specification languages, see, for example, [RGL01, DJLS08, HV12, GJD11, BKMP08], they fall short for checking compliance of concurrently logged actions and do not scale to large logs. In the following, we explain the reasons for these short-comings in more detail.

The underlying semantic model of these languages is that the observed system actions are totally ordered. However, a total ordering is often not available. Even simple IT systems are composed of multiple interacting subsystems, which typically are distributed and act concurrently. Hence system actions can only be observed locally and independently in each subsystem. Although we might have a total ordering on the actions observed in each individual subsystem, it is unclear how to combine them with actions observed in other subsystems. And policy compliance may depend on how all observed actions are totally ordered.

Synchronization of all subsystems for each observed system action leads to a total ordering, but this is usually prohibitively expensive. Not requiring it leads to a partial order on the observed actions. Determining whether at least one or whether all possible extensions of such a partial order into a total order violate a policy is in general an intractable problem. Intuitively, this is because a partial ordering on a finite set has, in the worst case, exponentially many different extensions to a total ordering.

Another shortcoming stems from the fact that the approaches described in [RGL01, DJLS08, HV12, GJD11, BKMP08] also fall short for checking compliance of larger IT systems like cloud-based services and systems that process machine-generated data. Due to a lack of parallelization, the monitors in these approaches do not cope with the enormous amount of logged system actions: these systems can easily log terabytes of actions each day.

Solution. In this dissertation, we present a solution for monitoring distributed and concurrent systems that scales to large logs. In particular, we identify policies for which compliance can

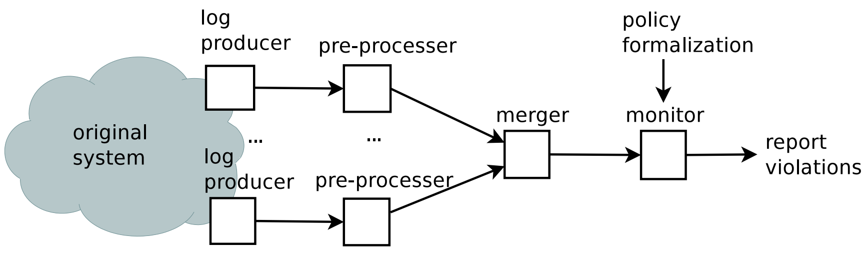

Figure 1.1: System Extensions

be checked efficiently by inspecting a single representative sequence in which the observed system actions are totally ordered. In a case study with Nokia, we deploy and evaluate this solution in a real-world concurrent and distributed IT system. Furthermore, to scale the compliance checking process to large logs, we provide techniques to parallelize the monitoring of logs and we evaluate their scalability in a large-scale case study with Google.

Monitoring Setup. To explain our approach in more detail, we describe our monitoring setup. Given a system, we extend it to observe system actions. We log the actions locally in the different subsystems and annotate each action with a timestamp. We assume that the clocks used for timestamping are synchronized [TvS02] and of limited precision (timestamps come from a non-dense set). Hence even with clock synchronization, the timestamps lead only to a partial ordering since actions can be logged in different subsystems with equal timestamps. We pre-process the local logs, merge them, and monitor this merged stream of logged actions. These system extensions are depicted in Figure 1.1.

To express policies, we use a metric first-order temporal logic (MFOTL). In general, temporal logics [Pnu77] are well suited to formalize system properties and to algorithmically reason about system behavior. In particular, the standard temporal operators allow us to naturally express temporal aspects of data usage policies, such as whenever a user requests the deletion of his data then the data must eventually be deleted. Metric temporal logics [AH91] associate timing constraints with temporal operators. We can thereby straightforwardly express requirements that commonly occur in data usage policies, for example that data deletion must happen within 30 days. A first-order logic allows us to formulate dependencies between the finite but unbounded number of agents and data elements in IT systems.

Basin et al. [BKMP08] presented a monitoring algorithm for an expressive fragment of MFOTL for a totally ordered sequence of timestamped actions and in [BHKZ12] we described an implementation of this algorithm. Basin et al. [BKM10] also showed that many security policies can naturally be expressed in an MFOTL fragment, which can be effectively monitored.

Monitoring Data Usage Policies. For monitoring security policies in general, we consider systems that consist of multiple subsystems. For monitoring data usage policies in particular, we consider a more restricted system model. The types of entities in this system are *data*, *(data) stores*, *agents*, and *actions*. Data is stored in distributed data stores such as databases and repositories and created, read, modified, combined, propagated, and deleted by actions initiated by agents. Agents are humans, applications, and processes, including database triggers. The agents do not necessarily comply with policies.

We assume that agents always access data directly from a store and never indirectly from another agent. Whenever an agent wants to use some data, it accesses the appropriate store, uses the data, and discards it afterwards. For subsequent usage, it must access the store again. Before discarding the data, the agent may write it, possibly after processing it in some way, into the same or a different store. In this way, data can propagate between stores. A consequence of this restriction on the interaction between system entities is that the use of data is always observable at the data stores and can be logged there locally.

Monitoring Concurrently Logged Actions. An overview of our solution to monitor concurrently logged actions is as follows: We identify two fragments of MFOTL that describe policies insensitive to the ordering of actions labeled with equal timestamps. For policies expressed in these fragments, it suffices to monitor a single stream of logged actions. For the first fragment, an arbitrary interleaving can be monitored. For the second fragment, it suffices to monitor the collapse of an interleaving, where actions logged with equal timestamps are assumed to have happened simultaneously. Both an interleaving and a collapse can be easily obtained by merging the logs produced by the subsystems.

The first fragment subsumes the second one in terms of expressiveness. However, system monitoring with respect to formulas in the second fragment is more efficient. Both fragments are defined by labeling a formula's atomic formulas and using rules to propagate the labels to the formula's root. The labels describe semantic properties about the insensitivity of the labeled subformula to the ordering of actions with equal timestamps. Furthermore, we provide means to approximate policies to fall within these fragments.

We evaluate our approach in a real-world case study, Nokia's Data-collection Campaign [AN10, KBD$^+$10, LGPA$^+$12]. In this campaign, sensitive data is collected by mobile phones and propagated between databases. The underlying IT system is an instance of our system model for monitoring data usage policies. For the evaluation, we extended it to support logging and monitoring as indicated in Figure 1.1.

Monitoring Large Logs. An overview of our solution to scale the monitoring process to large logs 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 MFOTL.

We present two basic orthogonal methods to generate sound and complete slices. With the first method, a slice is used to check system compliance during a specified period of time, for example, 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 refer to 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 irrelevant 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 [DG04]. 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 with Google, 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.

Contributions. We summarize our contributions as follows. We provide a solution for efficiently monitoring partially ordered logs, which is a central problem in monitoring concurrent distributed systems. Furthermore, we lay the foundations for splitting logs into slices for monitoring and provide a scalable algorithmic realization for monitoring large logs in an offline setting. Both our foundations and our algorithmic realization account for compositionality of slicing methods. Moreover, we demonstrate the feasibility, effectiveness, and scalability of our solutions on two real-world applications. In particular, the two identified MFOTL fragments are sufficiently expressive to capture real-world policies, the monitor can efficiently check such policies on real-world logs, and our slicing approach scales the monitoring process to terabytes of logged actions.

Although we focus here on MFOTL as the policy specification language and represent logs as temporal structures, the underlying principles of our slicing framework and monitoring a single representative trace to check compliance of an IT system are general and apply to other policy specification languages and representations of logs.

Organization. The remainder of this thesis is structured as follows. In Chapter 2, we provide background on MFOTL and the monitor. In Chapter 3, we present the theory underpinning the monitoring of concurrently logged actions. Namely, we prove that monitoring a partially ordered set of actions is in general intractable, define fragments of formulas for which this problem can be solved efficiently, compare these fragments, and explain how a policy can be approximated by one that can be monitored efficiently. In Chapter 4, we report on the Nokia case study. In Chapter 5, we present the theory that allows us to split logs into slices and monitor the slices in parallel. In particular, we lay the foundations for generating sound and complete slices and we present slicing and filtering methods based on data parameters and timestamps. In Chapter 6, we describe the Google case study and the experimental evaluation of monitoring log slices. In Chapter 7, we discuss related work and in Chapter 8, we draw conclusions.

The work described in Chapters 3 and 4 was done in collaboration with David Basin, Felix Klaedtke, and Eugen Zălinescu. Portions of the research used the MDC Database made available by Idiap Research Institute, Switzerland and owned by Nokia. Part of the work was supported by the Nokia Research Center, Switzerland. I thank Imad Aad, Debmalya Biswas, Olivier Bornet, Olivier Dousse, Juha Laurila, and Valtteri Niemi for valuable input and support in performing the Nokia case study.

The work described in Chapters 5 and 6 was done in collaboration with David Basin, Germano Caronni, Sarah Ereth, Felix Klaedtke, and Heiko Mantel. Part of the work was supported by Google Inc. and the Zurich Information Security and Privacy Center (`www.zisc.ethz.ch`). We 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 Google case study.

2 Preliminaries

In this chapter, we review metric first-order temporal logic (MFOTL), its use in monitoring, and the MFOTL monitoring algorithm.

2.1 Metric First-order Temporal Logic

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

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

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 a 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 $r^{\mathcal{D}_0}, r^{\mathcal{D}_1}, \ldots$ 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 $f[x \leftarrow y]$ for altering a function $f : X \to Y$

pointwise at $x \in X$. In particular, for a valuation v, a variable x, and $d \in |\bar{\mathcal{D}}|$, $v[x \hookleftarrow d]$ is the valuation mapping x to d and leaving other variables' valuation unchanged.

Satisfaction in MFOTL is defined by a binary relation $\models$ over a tuple consisting of a temporal structure, a valuation, and an index on the on side and a 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.

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

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 time point.

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 $\blacklozenge_I\, \phi := \mathit{true}\ \mathsf{S}_I\ \phi$, $\lozenge_I\, \phi := \mathit{true}\ \mathsf{U}_I\ \phi$, $\blacksquare_I\, \phi := \neg \blacklozenge_I \neg\phi$, and $\square_I\, \phi := \neg \lozenge_I \neg\phi$, 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 $\lozenge_I$ and $\square_I$. We also use non-metric operators like $\square\, \phi := \square_{[0,\infty)}\, \phi$. 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*.

2.2 Monitoring

We now illustrate how we use of MFOTL and the monitoring algorithm for compliance checking [BKMP08]. Consider the simple policy stating that reports must have been approved within at most 10 time units before they are published:

$$\Box\,\forall x.\, \mathit{publish}(x) \rightarrow \blacklozenge_{[0,11)}\, \mathit{approve}(x)\,.$$

We assume that the actions for publishing and approving reports are logged in relations. Specifically, for each time point $i \in \mathbb{N}$, we have the unary relations $\mathit{PUBLISH}_i$ and $\mathit{APPROVE}_i$ such that (1) $f \in \mathit{PUBLISH}_i$ iff the report f is published at time point i and (2) $f \in \mathit{APPROVE}_i$ iff the report f is approved at time point i. Observe that there can be multiple approvals at the same time point for different reports. Furthermore, every time point i has a timestamp $\tau_i \in \mathbb{N}$.

Given a sequence of logged publishing and approval actions, the corresponding temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$ with $\bar{\mathcal{D}} = (\mathcal{D}_0, \mathcal{D}_1, \dots)$ and $\bar{\tau} = (\tau_0, \tau_1, \dots)$ is as follows. The only predicate symbols in $\bar{\mathcal{D}}$'s signature are *publish* and *approve*, both of arity 1. We assume that every report is uniquely identified by a natural number. The domain of $\bar{\mathcal{D}}$ contains all these numbers, that is, $|\bar{\mathcal{D}}| \supseteq \mathbb{N}$. The ith structure in $\bar{\mathcal{D}}$ contains the relations $\mathit{PUBLISH}_i$ and $\mathit{APPROVE}_i$. The ith timestamp is simply τ_i, the time when these actions occurred.

To detect policy violations, our monitoring algorithm 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 $\psi = \mathit{publish}(x) \rightarrow \blacklozenge_{[0,11)}\, \mathit{approve}(x)$, which is $\neg\psi$ and equivalent to $\mathit{publish}(x) \wedge \blacksquare_{[0,11)}\, \neg \mathit{approve}(x)$. Note that we drop the outermost quantifier since we are not only interested in whether the policy is violated but we also want to provide additional information about the reported violations, namely, the reports that were published and not approved within the specified time window.

In a nutshell, the monitoring algorithm works as follows. It iterates over the structures $\mathcal{D}_i$ and their associated timestamps τ_i, where i is initially 0 and is incremented with each iteration. At each iteration, the algorithm incrementally maintains a collection of finite auxiliary relations for previous time points. Roughly speaking, for each time point $j \leq i$, these relations store the elements that satisfy the temporal subformulas of $\neg\psi$ at the time point j. If the temporal subformula of $\neg\psi$ refers to future time points, the algorithm might need to postpone the construction of such an auxiliary relation to a later iteration, until the processed prefix of $(\bar{\mathcal{D}}, \bar{\tau})$ is long enough to evaluate the subformula at time point j. The algorithm discards auxiliary relations whenever they become irrelevant for detecting further violations. The monitoring algorithm has been implemented in the tool MONPOLY [BHKZ12].

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 take into account a finite prefix of $(\bar{\mathcal{D}}, \bar{\tau})$ when determining the satisfying valuations of $\neg\psi$ 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 [BKMP08], a generalization of the standard notion of safe-range database queries [AHV94]. We refer to [BKMP08] for a detailed description of the monitoring algorithm. Additional algorithmic details are also presented below, for the sake of completeness.

Figure 2.1 presents the monitoring algorithm M_ψ. In the following, we briefly explain the pseudo code.

M_ψ first rewrites the negation of ψ. Heuristics are used that try to rewrite $\neg\psi$ into a formula ϕ that satisfies certain syntactic criteria that guarantee that it is temporal safe-range.

```
1  Rewrite $\neg\psi$; proceed only if rewritten formula $\phi$ is in the monitorable fragment of MFOTL.
2  $\ell \leftarrow 0$
3  $i \leftarrow 0$
4  $Q \leftarrow \{(\alpha, 0, \mathit{wait}(\alpha)) \mid \alpha$ is a temporal subformula of $\phi\}$
5  loop
6      forall $(\alpha, j, \emptyset) \in Q$ do
7          Build auxiliary relation for $\alpha$ and time point $j$ using the auxiliary relations for the
           temporal subformulas of $\alpha$ at time point $j-1$, if $j > 0$.
8      while all auxiliary relations for time point $i$ are built do
9          Evaluate $\phi$ at time point $i$ by using the auxiliary relations for time point $i$; output
           violations together with timestamp $\tau_i$.
10         If $i > 0$, discard all relations for time point $i-1$.
11         $i \leftarrow i+1$
12     $Q \leftarrow \{(\alpha, \ell+1, \mathit{wait}(\alpha)) \mid \alpha$ is a temporal subformula of $\phi\} \cup$
           $\{(\alpha, j, \bigcup_{\alpha' \in \mathit{update}(S, \tau_{\ell+1}-\tau_\ell)} \mathit{wait}(\alpha')) \mid (\alpha, j, S) \in Q$ and $S \neq \emptyset\}$
13     $\ell \leftarrow \ell+1$
```

Figure 2.1: The monitoring algorithm M_ψ

If these criteria are not satisfied, then M_ψ stops. For monitoring, M_ψ uses two counters ℓ and i. The counter ℓ is the index of the current element $(\mathcal{D}_\ell, \tau_\ell)$ in the input sequence $(\mathcal{D}_0, \tau_0), (\mathcal{D}_1, \tau_1), \ldots$, which is processed sequentially. Initially, ℓ is 0 and it is incremented with each loop iteration (lines 5–13). The counter i is the index of the next time point i (possibly in the past, from ℓ's point of view) that is checked for violations, that is, the next time point for which the monitor outputs the assignments satisfying ϕ.

The evaluation is delayed until all auxiliary relations for the temporal subformulas of ϕ are built (lines 8–11), that is, subformulas of ϕ where the outermost connective is one of the temporal operators $\bullet_I$, $\bigcirc_I$, S_I, or U_I. Furthermore, M_ψ uses the list[1] Q to ensure that the auxiliary relations for the time points are built at the right time: if $(\alpha, j, \emptyset)$ is an element of Q at the beginning of a loop iteration, enough time has elapsed to build the auxiliary relations for the temporal subformula α and time point j. Without loss of generality, we assume that each temporal subformula α occurs only once in ϕ. M_ψ initializes Q in line 4. The function *wait* identifies the subformulas that delay the formula evaluation:

$$\mathit{wait}(\alpha) := \begin{cases} \mathit{wait}(\beta) & \text{if } \alpha = \neg\beta, \alpha = \exists x.\beta, \text{ or } \alpha = \bullet_I \beta, \\ \mathit{wait}(\beta) \cup \mathit{wait}(\gamma) & \text{if } \alpha = \beta \vee \gamma \text{ or } \alpha = \beta \,\mathsf{S}_I\, \gamma, \\ \{\alpha\} & \text{if } \alpha = \bigcirc_I \beta \text{ or } \alpha = \beta \,\mathsf{U}_I\, \gamma, \\ \emptyset & \text{otherwise.} \end{cases}$$

The list Q is updated in line 12 before we increment ℓ in line 13 and start a new loop iteration. The update adds a new tuple $(\alpha, \ell+1, \mathit{wait}(\alpha))$ to Q, for each temporal subformula α of ϕ, and it removes tuples of the form $(\alpha, j, \emptyset)$ from Q. Moreover, for tuples (α, j, S) with $S \neq \emptyset$, the set S is updated using the functions *wait* and *update*, accounting for the elapsed time to

[1]We abuse notation by using set notation for lists. Moreover, we assume that Q is ordered so that (α, j, S) occurs before (α', j', S'), whenever α is a proper subformula of α', or $\alpha = \alpha'$ and $j < j'$.

the next time point, that is, $\tau_{\ell+1} - \tau_\ell$. For a set of formulas U and $t \in \mathbb{N}$, $\mathit{update}(U, t)$ is the set

$$\begin{aligned}&\{\beta \mid \bigcirc_I \beta \in U\} \cup \\ &\{\beta \,\mathsf{U}_{[\max\{0,b-t\},b'-t)}\, \gamma \mid \beta \,\mathsf{U}_{[b,b')}\, \gamma \in U, \text{ with } b' - t > 0\} \cup \\ &\{\beta \mid \beta \,\mathsf{U}_{[b,b')}\, \gamma \in U \text{ or } \gamma \,\mathsf{U}_{[b,b')}\, \beta \in U, \text{ with } b' - t \le 0\}\,.\end{aligned}$$

In line 7, we build the auxiliary relations for which enough time has elapsed, that is, the relations for α at time point j with $(\alpha, j, \emptyset) \in Q$. To efficiently build these relations, we use incremental constructions that reuse relations from the previous time point. In lines 8–11, if all the relations for time point i have been built, then M_ψ outputs the valuations violating ϕ at time point i together with the timestamp τ_i. Furthermore, after each output, the relations at time point $i - 1$ are discarded (if $i > 0$) and i is incremented.

3 Monitoring Concurrently Logged Actions

In this chapter, we provide the theory for monitoring concurrently logged actions. In particular, in Section 3.1, we prove the intractability of monitoring when multiple log files are produced in a concurrent setting. In Section 3.2, we motivate two solutions where only a single representative log is monitored. In Sections 3.3 and 3.4, we show when monitoring a single such log is sufficient. In Section 3.5, we compare the two solutions.

We first illustrate the problem with monitoring concurrently logged actions which give rise to partially ordered logs. Note that the monitoring algorithm assumes a total ordering on the logged actions. However, a total ordering is not necessarily available in a distributed and concurrent system. Moreover, policy compliance may depend on such a total ordering. For example, consider the policy for publishing and approving reports and a system in which the publish and approval actions are performed and logged by different system parts. If two such corresponding actions are equally timestamped and when assuming an interleaving semantics of the system parts, then two orderings of the actions are possible: (i) the report is first approved and then published and (ii) the report is published before being approved. For the ordering (i) the policy is satisfied, while for (ii) it is violated in case there is no other approval within the specified time window.

3.1 Intractability

In this section, we prove the intractability of monitoring when multiple log files are produced in a concurrent setting.

In the remainder of this thesis, unless otherwise noted, we assume that all temporal structures have the same signature (C, R, ι), the same domain $\mathbb{D}$, and that constant symbols in C are interpreted equally. Note that any two temporal structures whose common constant symbols are equally interpreted can easily be extended so that their extensions fulfill this requirement.

Log Interleavings. Intuitively, an interleaving of logs preserves the ordering of the logged actions with respect to their timestamps, but allows for any possible ordering of actions with equal timestamps that are recorded by different log producers. To define an interleaving, for a function $f : X \to Y$, let $\operatorname{img}(f)$ denote the set $\{y \in Y \mid f(x) = y, \text{ for some } x \in X\}$.

Definition 3.1.1. *Let $(\bar{\mathcal{D}}^1, \bar{\tau}^1)$, $(\bar{\mathcal{D}}^2, \bar{\tau}^2)$, and $(\bar{\mathcal{D}}, \bar{\tau})$ be temporal structures. $(\bar{\mathcal{D}}, \bar{\tau})$ is an* interleaving *of $(\bar{\mathcal{D}}^1, \bar{\tau}^1)$ and $(\bar{\mathcal{D}}^2, \bar{\tau}^2)$ if there are strictly monotonic functions $f_1, f_2 : \mathbb{N} \to \mathbb{N}$ with*

(1) $\operatorname{img}(f_1) \cup \operatorname{img}(f_2) = \mathbb{N}$,

(2) $\operatorname{img}(f_1) \cap \operatorname{img}(f_2) = \emptyset$, *and*

(3) $\tau^k_i = \tau_{f_k(i)}$ *and* $r^{\mathcal{D}^k_i} = r^{\mathcal{D}_{f_k(i)}}$, *for all* $k \in \{1, 2\}$, $i \in \mathbb{N}$, $r \in R$.

We denote by $(\bar{\mathcal{D}}^1, \bar{\tau}^1) \between (\bar{\mathcal{D}}^2, \bar{\tau}^2)$ *the* set of interleavings *of the temporal structures* $(\bar{\mathcal{D}}^1, \bar{\tau}^1)$ *and* $(\bar{\mathcal{D}}^2, \bar{\tau}^2)$*.*

Since there are usually multiple interleavings of two temporal structures, we formulate policy violations with respect to a set of temporal structures.

Definition 3.1.2. *Let* $\mathbf{T}$ *be a set of temporal structures.*

(1) $\mathbf{T}$ weakly violates *the formula* ϕ *at time point* $i \in \mathbb{N}$ *if for some* $(\bar{\mathcal{D}}, \bar{\tau}) \in \mathbf{T}$ *and some valuation* v*, it holds that* $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \phi$*.*

(2) $\mathbf{T}$ strongly violates *the formula* ϕ *at time point* $i \in \mathbb{N}$ *if for all* $(\bar{\mathcal{D}}, \bar{\tau}) \in \mathbf{T}$*, there is some valuation* v *such that* $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \phi$*.*

Unfortunately, even in a propositional setting, determining whether the set of interleavings weakly or strongly violates a formula is intractable.

Theorem 3.1.3. *Let* $(\bar{\mathcal{D}}^1, \bar{\tau}^1)$ *and* $(\bar{\mathcal{D}}^2, \bar{\tau}^2)$ *be temporal structures,* $i \in \mathbb{N}$*, and* ϕ *a quantifier-free sentence with only Boolean and non-metric past operators that neither contains the equality symbol* $\approx$ *nor the ordering symbol* $\prec$*.*

1. *Determining whether the set of interleavings* $(\bar{\mathcal{D}}^1, \bar{\tau}^1) \between (\bar{\mathcal{D}}^2, \bar{\tau}^2)$ *weakly violates* ϕ *at* i *is NP-complete.*

2. *Determining whether the set of interleavings* $(\bar{\mathcal{D}}^1, \bar{\tau}^1) \between (\bar{\mathcal{D}}^2, \bar{\tau}^2)$ *strongly violates* ϕ *at* i *is coNP-complete.*

Note that both decision problems are well defined as ϕ does not contain future operators. We therefore only need to examine the finite prefixes with length $i + 1$ of the interleavings to determine whether ϕ is weakly or strongly violated at the time point i.

Before proving Theorem 3.1.3, we remark that related intractability results for linear-time temporal logic (LTL) on so-called partially ordered traces are given in [MMV08] and [GMM06]. The setting in [MMV08] is different from ours. In particular, it is unclear how to describe the set of interleavings of two timestamped traces using partially ordered traces as defined in [MMV08]. Moreover, we reduce the checking of the satisfiability and validity of formulas in propositional logic, respectively, to the respective decision problem for proving its hardness. In [MMV08], the global-predicate-detection decision problem [CG98] is used. The setting in [GMM06] allows for arbitrary partial orders and hence could be used to describe the set of interleavings of two timestamped traces. The authors reduce the decision problem 3-SAT to the problem of determining whether all possible interleavings satisfy a formula.

Next, we prove Theorem 3.1.3.

Membership. The decision problem in Theorem 3.1.3(1) is in NP as a nondeterministic Turing machine can first guess the violating interleaving up to the given time point and then verify its guess in polynomial time [MS03]. Note that the Turing machine does not need to guess a valuation, as the input formula is a quantifier-free sentence and thus contains no variables. The decision problem in Theorem 3.1.3(2) is in coNP since its complement is in NP for the negation of the input formula.

Hardness. Hardness of the decision problem in Theorem 3.1.3(1) is established by polynomially reducing SAT to it. Analogously, the coNP-hardness of the decision problem in Theorem 3.1.3(2) is shown by polynomially reducing TAUT to it.

Reduction from SAT. We show NP-hardness of the decision problem in Theorem 3.1.3(1) by a reduction from SAT. The SAT problem asks whether a given propositional formula is satisfiable. SAT is NP-hard.

To fix notation, we recall that a propositional formula α over a set of atomic propositions P is satisfiable iff there is an assignment θ of propositions to truth values $\perp$ (denoting false) and $\top$ (denoting true), that is, $\theta : P \to \{\perp, \top\}$, such that $\theta(\alpha) = \top$, where θ is homomorphically extended from atomic propositions to formulas.

Suppose $P = \{p_0, \dots, p_{n-1}\}$, with $n \geq 0$, is a set of atomic propositions. Let S be the signature (C, R, ι) with $C = \{c\}$, $R = \{q_0, r_0, \dots, q_{n-1}, r_{n-1}\}$, and $\iota(q_i) = \iota(r_i) = 1$, for any $0 \leq i < n$. The two temporal structures $(\bar{\mathcal{D}}^1, \bar{\tau}^1)$ and $(\bar{\mathcal{D}}^2, \bar{\tau}^2)$ over S are given by $|\bar{\mathcal{D}}| = \{c\}$, $c^{\bar{\mathcal{D}}} = c$, $\tau^1_i = \tau^2_i = i$, and

$$q_i^{\mathcal{D}^k_j} = \begin{cases} \{c\} & \text{if } k = 1 \text{ and } i = j, \\ \emptyset & \text{otherwise,} \end{cases}$$

$$r_i^{\mathcal{D}^k_j} = \begin{cases} \{c\} & \text{if } k = 2 \text{ and } i = j, \\ \emptyset & \text{otherwise,} \end{cases}$$

for any $i \in \mathbb{N}$, $k \in \{1, 2\}$, and $i, j \in \mathbb{N}$ with $0 \leq i < n$,

Given a propositional formula α over P, the MFOTL formula $\ulcorner\alpha\urcorner$ is obtained by replacing each occurrence of a proposition p_i in α with $\blacklozenge\,(r_i(c) \wedge \blacklozenge\, q_i(c))$. Thus, given a propositional formula α, the reduction constructs the two prefixes of length n of $(\bar{\mathcal{D}}^1, \bar{\tau}^1)$ and $(\bar{\mathcal{D}}^2, \bar{\tau}^2)$ and the MFOTL formula $\ulcorner\alpha\urcorner$. This reduction is linear in the length of α. Its correctness is shown by Lemma 3.1.6. The following remarks and lemma will be needed.

Remark 3.1.4. For any interleaving $(\bar{\mathcal{D}}, \bar{\tau}) \in (\bar{\mathcal{D}}^1, \bar{\tau}^1) \between (\bar{\mathcal{D}}^2, \bar{\tau}^2)$, the functions f_1 and f_2 in Definition 3.1.1 satisfy $f_k(i) \in \{2i, 2i+1\}$ where $k \in \{1, 2\}$. Moreover, these functions are unique, that is, if $g_1, g_2 : \mathbb{N} \to \mathbb{N}$ are strictly monotonic functions satisfying conditions (1)–(3) in Definition 3.1.1, then either $g_1 = f_1$ and $g_2 = f_2$, or $g_1 = f_2$ and $g_2 = f_1$. Furthermore, for any strictly monotonic functions f_1 and f_2 satisfying conditions (1) and (2) in Definition 3.1.1 and with $f_1(i), f_2(i) \in \{2i, 2i+1\}$ for $0 \leq i < n$, there is a unique temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$ such that f_1 and f_2 also satisfy condition (3). In other words, the functions f_1 and f_2 determine an interleaving of $(\bar{\mathcal{D}}^1, \bar{\tau}^1)$ and $(\bar{\mathcal{D}}^2, \bar{\tau}^2)$.

Lemma 3.1.5. *Let α be a propositional formula, θ a truth value assignment, v a valuation, and $(\bar{\mathcal{D}}, \bar{\tau})$ an interleaving of $(\bar{\mathcal{D}}^1, \bar{\tau}^1) \between (\bar{\mathcal{D}}^2, \bar{\tau}^2)$ given by the functions f_1 and f_2 such that $\theta(p_i) = \top$ iff $f_1(i) = 2i$, for any i with $0 \leq i < n$. Then $\theta(\alpha) = \top$ iff $(\bar{\mathcal{D}}, \bar{\tau}, v, 2n) \models \ulcorner\alpha\urcorner$.*

Proof. We use structural induction on the form of α. The only interesting case is the base case; the other cases follow directly from the induction hypotheses. Thus let $\alpha = p_i \in P$.

Suppose that $(\bar{\mathcal{D}}, \bar{\tau}, v, 2n) \models \blacklozenge(r_i(c) \wedge \blacklozenge\, q_i(c))$. That is, there is a time point $j \leq 2n$ such that $(\bar{\mathcal{D}}, \bar{\tau}, v, j) \models r_i(c)$ and such that there is a time point $j' \leq j$ for which $(\bar{\mathcal{D}}, \bar{\tau}, v, j') \models q_i(c)$. Then $c \in r_i^{\mathcal{D}_j}$ and $c \in q_i^{\mathcal{D}_{j'}}$. From the definition of an interleaving and the definitions of the interpretations of the predicate symbols q_i and r_i, it follows that $j = f_2(i)$ and $j' = f_1(i)$. Then, as $f_1(i), f_2(i) \in \{2i, 2i+1\}$, $f_1(i) \neq f_2(i)$, and $j' \leq j$, we have that $f_1(i) = 2i$ and $f_2(i) = 2i+1$. Thus $\theta(p_i) = \top$.

Suppose that $\theta(\alpha) = \top$. Then $f_1(i) = 2i$ and $f_2(i) = 2i+1$. We have $(\bar{\mathcal{D}}, \bar{\tau}, v, 2i) \models q_i(c)$ and $(\bar{\mathcal{D}}, \bar{\tau}, v, 2i+1) \models r_i(c)$. Thus $(\bar{\mathcal{D}}, \bar{\tau}, v, 2i+1) \models r_i(c) \wedge \blacklozenge\, q_i(c)$ and clearly $(\bar{\mathcal{D}}, \bar{\tau}, v, 2n) \models \blacklozenge\,(r_i(c) \wedge \blacklozenge\, q_i(c))$. □

Lemma 3.1.6. *Let α be a propositional formula. It holds that α is satisfiable iff $(\bar{\mathcal{D}}^1, \bar{\tau}^1) \between (\bar{\mathcal{D}}^2, \bar{\tau}^2)$ weakly violates $\neg\ulcorner\alpha\urcorner$ at time point $2n$.*

Proof. Suppose first that α is satisfiable. Then there is a truth value assignment θ such that $\theta(\alpha) = \top$. Let $(\bar{\mathcal{D}}, \bar{\tau})$ be the interleaving determined by the functions f_1 and f_2 given by

$$f_1(i) = \begin{cases} 2i & \text{if } \theta(p_i) = \top, \\ 2i+1 & \text{otherwise,} \end{cases}$$

and

$$f_2(i) = \begin{cases} 2i & \text{if } \theta(p_i) = \bot, \\ 2i+1 & \text{otherwise.} \end{cases}$$

Let v be an arbitrary valuation. From Lemma 3.1.5, we obtain that $(\bar{\mathcal{D}}, \bar{\tau}, v, 2n) \models \ulcorner\alpha\urcorner$, that is, $(\bar{\mathcal{D}}, \bar{\tau}, v, 2n) \not\models \neg\ulcorner\alpha\urcorner$.

Suppose now that $(\bar{\mathcal{D}}^1, \bar{\tau}^1) \between (\bar{\mathcal{D}}^2, \bar{\tau}^2)$ weakly violates $\neg\ulcorner\alpha\urcorner$ at time point $2n$. Then there is an interleaving $(\bar{\mathcal{D}}, \bar{\tau})$ and a valuation v such that $(\bar{\mathcal{D}}, \bar{\tau}, v, 2n) \not\models \neg\ulcorner\alpha\urcorner$. Let f_1 and f_2 be the functions determined by $(\bar{\mathcal{D}}, \bar{\tau})$ as in Definition 3.1.1. Let θ be a truth value assignment such that $\theta(p_i) = \top$ if $f_1(i) = 2i$. Using again Lemma 3.1.5, we get that θ is a satisfying assignment for α. □

Reduction from TAUT. We show coNP-hardness of the decision problem in Theorem 3.1.3(2) by reduction from TAUT. The TAUT problem asks whether a given propositional formula is a tautology. TAUT is coNP-hard.

We recall that a propositional formula α over a set of atomic propositions P is a tautology if $\theta(\alpha) = \top$ for any assignment θ of propositions to truth values.

We use the same reduction that was used in the decision problem in Theorem 3.1.3(1). The correctness of the reduction follows from the following lemma.

Lemma 3.1.7. *Let α be a propositional formula. Then α is a tautology iff $(\bar{\mathcal{D}}^1, \bar{\tau}^1) \between (\bar{\mathcal{D}}^2, \bar{\tau}^2)$ strongly violates $\neg\ulcorner\alpha\urcorner$ at time point $2n$.*

Proof. Suppose first that α is a tautology. Let $(\bar{\mathcal{D}}, \bar{\tau})$ be an arbitrary interleaving in $(\bar{\mathcal{D}}^1, \bar{\tau}^1) \between (\bar{\mathcal{D}}^2, \bar{\tau}^2)$ and f_1 and f_2 be functions as in Definition 3.1.1. Let θ be a truth value assignment such that $\theta(p_i) = \top$ iff $f_1(i) = 2i$. Let v be an arbitrary valuation. Using Lemma 3.1.5, we obtain that $(\bar{\mathcal{D}}, \bar{\tau}, v, 2n) \not\models \neg\ulcorner\alpha\urcorner$. Hence $(\bar{\mathcal{D}}^1, \bar{\tau}^1) \between (\bar{\mathcal{D}}^2, \bar{\tau}^2)$ strongly violates $\neg\ulcorner\alpha\urcorner$ at time point $2n$.

Suppose now that $(\bar{\mathcal{D}}^1, \bar{\tau}^1) \between (\bar{\mathcal{D}}^2, \bar{\tau}^2)$ strongly violates $\neg\ulcorner\alpha\urcorner$ at time point $2n$. Let θ be an arbitrary truth value assignment. Let $(\bar{\mathcal{D}}, \bar{\tau})$ be the interleaving determined by the functions f_1 and f_2 given by

$$f_1(i) = \begin{cases} 2i & \text{if } \theta(p_i) = \top, \\ 2i+1 & \text{otherwise,} \end{cases}$$

and

$$f_2(i) = \begin{cases} 2i & \text{if } \theta(p_i) = \bot, \\ 2i+1 & \text{otherwise.} \end{cases}$$

There is a valuation v such $(\bar{\mathfrak{D}}, \bar{\tau}, v, 2n) \not\models \neg\ulcorner\alpha\urcorner$. Using again Lemma 3.1.5, we have that θ is a satisfying assignment for α. Hence α is a tautology. □

3.2 Sufficient Logs

In this section, we motivate two solutions where only a single representative log is monitored. We first give conditions with respect to an arbitrary set of temporal structures for when it suffices to monitor a single temporal structure.

Definition 3.2.1. *The temporal structure* $(\bar{\mathfrak{C}}, \bar{\kappa})$ *is* sufficient *for the formula* ϕ *on the set* $\mathbf{T}$ *of temporal structures if for all valuations* v*, the following conditions are fulfilled:*

(S1) If $(\bar{\mathfrak{C}}, \bar{\kappa}, v, 0) \models \phi$ *then* $(\bar{\mathfrak{D}}, \bar{\tau}, v, 0) \models \phi$*, for all* $(\bar{\mathfrak{D}}, \bar{\tau}) \in \mathbf{T}$.

(S2) If $(\bar{\mathfrak{C}}, \bar{\kappa}, v, 0) \not\models \phi$ *then* $(\bar{\mathfrak{D}}, \bar{\tau}, v, 0) \not\models \phi$*, for all* $(\bar{\mathfrak{D}}, \bar{\tau}) \in \mathbf{T}$.

Note that the actual ordering of actions logged with equal timestamps in a concurrent system cannot be known unless there is an additional mechanism to order these events. Instead of adding such mechanisms, we identify two classes of policies, which are indifferent to the ordering of equally timestamped actions, the *interleaving-sufficient* and *collapse-sufficient* policies. Formulas in both classes can be efficiently monitored by inspecting just a single temporal structure instead of all the possible interleavings. For both classes, the set $\mathbf{T}$ in Definition 3.2.1 is the set of all interleavings of two temporal structures. For an interleaving-sufficient policy, inspecting an arbitrary interleaving is sufficient to determine whether the policy is strongly violated. With collapse-sufficient policies we exploit the inability to distinguish the ordering of events logged with equal timestamps to make monitoring more efficient and inspect the so-called collapse of an interleaving:

Definition 3.2.2. *Let* $(\bar{\mathfrak{D}}, \bar{\tau})$ *and* $(\bar{\mathfrak{C}}, \bar{\kappa})$ *be temporal structures.* $(\bar{\mathfrak{C}}, \bar{\kappa})$ *is a* collapse *of* $(\bar{\mathfrak{D}}, \bar{\tau})$ *if there is a monotonic surjective function* $f : \mathbb{N} \to \mathbb{N}$ *such that*

(1) if $\tau_i = \tau_j$ *then* $f(i) = f(j)$*, for all* $i, j \in \mathbb{N}$*,*

(2) $\kappa_{f(i)} = \tau_i$*, for all* $i \in \mathbb{N}$*, and*

(3) $r^{\mathfrak{C}_j} = \bigcup_{i \in f^{-1}(j)} r^{\mathfrak{D}_i}$*, for all* $j \in \mathbb{N}$ *and* $r \in R$.

Intuitively, the structures of the temporal structure $(\bar{\mathfrak{D}}, \bar{\tau})$ with equal timestamps are collapsed into a single structure. Figure 3.1 depicts an example of collapsing. The collapse is uniquely defined and we denote it by $col(\bar{\mathfrak{D}}, \bar{\tau})$. Furthermore, the collapses of temporal structures in the set of interleavings of two given temporal structures are all isomorphic. Note that the set of interleavings is strictly included in the set of collapse pre-images, that is, $(\bar{\mathfrak{D}}, \bar{\tau}) \bowtie (\bar{\mathfrak{D}}', \bar{\tau}') \subsetneq col^{-1}(\bar{\mathfrak{C}}, \bar{\kappa})$, where $(\bar{\mathfrak{C}}, \bar{\kappa})$ is the collapse of an interleaving of the temporal structures $(\bar{\mathfrak{D}}, \bar{\tau})$ and $(\bar{\mathfrak{D}}', \bar{\tau}')$.

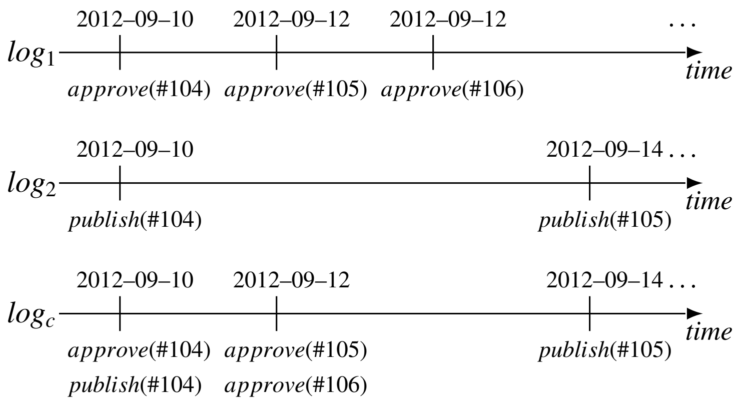

Figure 3.1: Example of a Collapsed Interleaving, log_c, of the Temporal Structures log_1 and log_2

3.3 Monitoring an Interleaving

In this section we describe an interleaving-sufficient fragment. Intuitively, interleaving-sufficient formulas are those formulas that yield neither false positives nor false negatives when monitoring an interleaving. This is because they either satisfy all possible interleavings of two temporal structures or they violate all possible interleavings.

Definition 3.3.1. *Let ϕ be a formula. For $k \in \{1,2\}$, we say that ϕ* has the property (Ik) *if $(\bar{\mathfrak{C}}, \bar{\kappa})$ fulfills the condition (Sk) in Definition 3.2.1 with respect to ϕ and $(\bar{\mathfrak{D}}, \bar{\tau}) \between (\bar{\mathfrak{D}}', \bar{\tau}')$, for every $(\bar{\mathfrak{D}}, \bar{\tau})$, $(\bar{\mathfrak{D}}', \bar{\tau}')$, and $(\bar{\mathfrak{C}}, \bar{\kappa})$, where $(\bar{\mathfrak{C}}, \bar{\kappa})$ is an interleaving of $(\bar{\mathfrak{D}}, \bar{\tau})$ and $(\bar{\mathfrak{D}}', \bar{\tau}')$. Moreover, ϕ is* interleaving-sufficient *if it has the properties (I1) and (I2).*

Note that we define interleaving-sufficiency only as a property of the formula. We could alternatively consider a refined notion that limits the interleavings on which the formula must hold. For example, if the relations of certain predicate symbols are logged by a single logging mechanism then we can impose conditions that the temporal structures $(\bar{\mathfrak{D}}, \bar{\tau})$ and $(\bar{\mathfrak{D}}', \bar{\tau}')$ in the above definition must fulfill. This would enlarge the set of interleaving-sufficient formulas. However, for the ease of exposition, we restrict ourselves here to the property as defined in Definition 3.3.1.

Monitoring an arbitrary interleaving with respect to an interleaving-sufficient formula is correct for strong violations. Since the formula has property (I2), violations found in $(\bar{\mathfrak{C}}, \bar{\kappa})$ imply that the set of interleavings strongly violates the formula. The converse is ensured by the property (I1): if no violation is found in $(\bar{\mathfrak{C}}, \bar{\kappa})$, then all interleavings are policy compliant. Furthermore, by monitoring $(\bar{\mathfrak{C}}, \bar{\kappa})$ we also detect when the set of interleavings both weakly and strongly violates the given formula. The reason is that if a formula is strongly violated by a set of interleavings then it is also weakly violated, since the set of interleavings is always nonempty.

Example 3.3.2. *Recall the formula $\Box \forall x. publish(x) \rightarrow \blacklozenge_{[0,11)} approve(x)$ from the example in Section 2.2. It is not interleaving-sufficient. Suppose that a report x is published in $(\bar{\mathfrak{D}}^1, \bar{\tau}^1)$ at time point i, that is, $x \in publish^{\mathfrak{D}^1_i}$ and only approved in $(\bar{\mathfrak{D}}^2, \bar{\tau}^2)$ at the equally timestamped* *time point j, that is, $x \in approve^{\mathcal{D}^2_j}$ with $\tau^2_j = \tau^1_i$. Then there is an interleaving $(\bar{\mathcal{D}}, \bar{\tau}) \in (\bar{\mathcal{D}}^1, \bar{\tau}^1) \between (\bar{\mathcal{D}}^2, \bar{\tau}^2)$ where the approval action comes (pointwise) strictly after the publish action. We cannot handle this formula correctly by monitoring just a single interleaving of the given temporal structures $(\bar{\mathcal{D}}^1, \bar{\tau}^1)$ and $(\bar{\mathcal{D}}^2, \bar{\tau}^2)$.*

A slightly stronger policy however can be efficiently monitored. Namely, the policy that requires that an approval action must happen timewise strictly before the publish action, that is, $\Box\, \forall x.\, publish(x) \to \blacklozenge_{[1,11)}\, approve(x)$. This formula is interleaving-sufficient. Similarly, $\Box\, \forall x.\, publish(x) \to \Diamond_{[0,1)} \blacklozenge_{[0,11)}\, approve(x)$ is also interleaving-sufficient. It formalizes the slightly weaker policy where every publish action must be approved at a time point with a timestamp that is less than or equal to the timestamp of the time point when the publish action happens.

In Theorem 3.3.4, we show the undecidability of checking whether a formula has the interleaving-sufficient property. The theorem follows from the undecidability of the tautology problem for first-order temporal logic (FOTL) formulas, that is, MFOTL formulas where the temporal operators do not have any metric constraints, established in Lemma 3.3.3.

Lemma 3.3.3. *Given a FOTL formula ϕ, it is undecidable whether ϕ is a tautology.*

Proof. We reduce the halting problem of a deterministic Turing machine (DTM) with the empty word as input to the FOTL tautology problem. We first introduce notation for a DTM and then proceed with the reduction.

Different types of Turing machines are used in the literature, so we briefly describe the one we use. For a more detailed introduction to Turing machines, see [HMU00]. Our DTM has a tape and a head to read and write the tape. The tape consists of cells and is infinite in both directions. The cells of the tape are indexed with the indexes coming from $\mathbb{Z}$. A single tape symbol is written in each cell. Initially, the input to the DTM is written on the tape starting at cell 0. The rest of the tape is filled with the blank symbol. We consider only the empty word as input, so the whole tape is filled with the blank symbol. The head of the DTM is initially positioned at cell 0.

The DTM is always in one of finitely many states and executes in steps. In each step, the DTM reads the symbol on the tape at the cell where the head is positioned, writes a new symbol into the cell, moves the head to the left, right, or not at all, and finally, the DTM may make a transition into a new state. When the DTM reaches a final state, it continues to loop forever in this state, always writes the same symbol onto the tape, and does not move the head. We say that it halts.

Formally, the DTM is described by a tuple $(Q, \Gamma, \Sigma, \delta, q_0, B, F)$, where the components are as follows.

- Q is the finite set of states in which the DTM can be.
- Γ is the finite set of tape symbols.
- $\Sigma \subseteq \Gamma$ is the finite set of input symbols.
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\text{left}, \text{right}, \text{none}\}$ is the transition function. The arguments of $\delta(q, x)$ are a state q in which the DTM is and a symbol x read from the tape where the head is positioned. The value of $\delta(q, x)$ is a triple (p, y, d), where:

  - p is the next state into which the DTM transitions,
  - y is the symbol to be written in the cell where the head is positioned, and
  - d is left, right, or none indicating whether the head should move to the left, to the right, or not move at all, respectively.

- $q_0 \in Q$ is the initial state of the DTM.
- $B \in \Gamma \setminus \Sigma$ is the blank symbol.
- $F \subseteq Q$ is the set of final states.

To ensure that the DTM loops in final states, we require that $\delta(q, x) = (q, x, \text{none})$ for all $q \in F$ and all $x \in \Gamma$.

We now reduce the undecidable problem of deciding whether a DTM halts on the empty word to the problem of deciding whether a FOTL formula is unsatisfiable. To describe a run of the DTM we use the following predicate symbols.

- $H(i)$ iff the head is positioned at cell i.
- $T_x(i)$ iff the tape contains symbol x, other than the blank symbol, at position i.
- I_q iff the DTM is in state q.

To check for an empty symbol at position i on the tape, we use the syntactic sugar $T_B(i) := \bigwedge_{x\in\Gamma\setminus\{B\}} \neg T_x(i)$.

We represent positions on the tape with an index $i \in \mathbb{Z}$. The left-most symbol of the input is at position 0, where also the head is initially positioned.

We also need a successor function on $\mathbb{Z}$. We define it as

$$S(i, j) := i \prec j \wedge \forall k. (k \prec i \vee k \approx i \vee k \approx j \vee j \prec k)\,.$$

We describe a non-halting run of the DTM M with the FOTL formula

$$\rho_M := \Box\, \mathit{WELLFORMED} \wedge (\blacklozenge\, \mathit{INIT}) \wedge \mathit{STEP} \wedge \neg \mathit{FINAL}\,,$$

where its subformulas are as follows.

- *WELLFORMED* ensures that the DTM is in a proper configuration. We define it as

$$\begin{aligned}\mathit{WELLFORMED} :=\, &(\forall i. \forall j. H(i) \wedge H(j) \to i \approx j) \wedge \\ &(\forall i. \bigwedge_{x,y\in\Gamma} (T_x(i) \wedge T_y(i) \to x \approx y)) \wedge \\ &\bigwedge_{p,q\in Q} (I_p \wedge I_q \to p \approx q)\,.\end{aligned}$$

  It ensures that the head is positioned at exactly one tape cell, that there is exactly one symbol written on each tape cell, and that the DTM is in exactly one state.

- *INIT* describes the initial configuration of the DTM. We define it as

$$\mathit{INIT} := H(0) \wedge I_{q_0} \wedge \forall i.\, T_B(i)\,.$$

Note that we consider only the empty word as the input, so the whole tape is initially filled with the blank symbol.

- *STEP* describes one step of the DTM. Let the tuples (q, x, p, y, d) represent the transition function δ where $(p, y, d) = \delta(q, x)$. *STEP* is a conjunction of formulas representing all those tuples. For tuples with d = left, the formula is

$$\forall i.\, \forall j.\, I_q \wedge H(j) \wedge T_x(j) \wedge S(i, j) \to \bigcirc\, I_p \wedge H(i) \wedge T_y(j)\,.$$

For tuples with d = right, the formula is

$$\forall i.\, \forall j.\, I_q \wedge H(i) \wedge T_x(i) \wedge S(i, j) \to \bigcirc\, I_p \wedge H(j) \wedge T_y(i)\,.$$

For tuples with d = none, the formula is

$$\forall i.\, I_q \wedge H(i) \wedge T_x(i) \to \bigcirc\, I_p \wedge H(i) \wedge T_y(i)\,.$$

In addition, the conjunction also contains the formula

$$\forall i.\, \forall j.\, \bigwedge_{z \in \Gamma} (H(i) \wedge i \not\approx j \wedge T_z(j) \to \bigcirc\, T_z(j))$$

expressing the fact that the tape can change only at the position where the head is positioned.

- *FINAL* describes the DTM entering a final state. We define

$$\mathit{FINAL} := \bigvee_{q \in F} I_q\,.$$

Every configuration of the DTM M in a run is represented by a time point in the model of the FOTL formula ρ_M. Note that this is a valid MFOTL model. At any step it holds that M has written at most a finite number of cells on the tape, so the relations of the predicate symbols T_x for $x \in \Gamma \setminus \{B\}$ are always finite. There is no predicate symbol to directly represent the blank symbol.

The DTM M does not halt if there is a model where ρ_M is satisfied. Since the formula ρ_M can be effectively constructed from the description of M, the undecidability of the halting problem implies the undecidability of the unsatisfiability problem for FOTL formulas. By considering the negation of a FOTL formula, it follows that the tautology problem is also undecidable. $\square$

Theorem 3.3.4. *Given an MFOTL formula ϕ, it is undecidable whether ϕ is interleaving-sufficient.*

Proof. We restrict ourselves without loss of generality to FOTL formulas. From Lemma 3.3.3 we know that the problem whether a FOTL formula ϕ is a tautology is undecidable. Hence, also determining whether ϕ is unsatisfiable is an undecidable problem. We proceed by reducing the problem of deciding whether ϕ is unsatisfiable to deciding whether ϕ is interleaving-sufficient. To this end, we show the following equivalence: ϕ is unsatisfiable iff the formula $\phi \land \Box\, p \to \blacklozenge\, q$ is interleaving-sufficient, where the predicate symbols p and q do not occur in ϕ.

Note that if ϕ falls into the fragment of MFOTL that we can monitor, then it is of the form $\Box\,\psi$. The formula $\phi \land \Box\, p \to \blacklozenge\, q$ can then be rewritten to $\Box\,\psi \land (p \to \blacklozenge\, q)$ and hence also falls into the fragment of MFOTL that we can monitor.

We first show the direction from left to right. As ϕ is unsatisfiable, $\phi \land \Box\, p \to \blacklozenge\, q$ is unsatisfiable. Hence, it is interleaving-sufficient.

Next, we show the direction from right to left and prove that if ϕ is satisfiable then $\phi \land \Box\, p \to \blacklozenge\, q$ is not interleaving-sufficient. If ϕ is satisfiable, there is a temporal structure $(\bar{\mathcal{D}}, \bar{\tau})$ on which ϕ is satisfied. As ϕ's temporal operators do not have any temporal constraints, we can pick $\bar{\tau}$ so that the first two time points have an equal timestamp. That is, $\tau_0 = \tau_1$. Furthermore, because the predicate symbols p and q do not occur in the formula ϕ, we can pick $\bar{\mathcal{D}}$ so that the predicate symbol q is satisfied only at the first time point and the predicate symbol p is satisfied only at the second time point. That is, $(\bar{\mathcal{D}}, \bar{\tau}, v, 0) \models q$, but $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models q$, for all $i \neq 0$ and any valuation v. Similarly, $(\bar{\mathcal{D}}, \bar{\tau}, v, 1) \models p$, but $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models p$, for all $i \neq 1$. Clearly, this temporal structure satisfies our formula, that is, $(\bar{\mathcal{D}}, \bar{\tau}, v, 0) \models \phi \land \Box\, p \to \blacklozenge\, q$.

Let $(\bar{\mathcal{D}}_1, \bar{\tau}_1)$ and $(\bar{\mathcal{D}}_2, \bar{\tau}_2)$ be two temporal structures such that $(\bar{\mathcal{D}}, \bar{\tau}) \in (\bar{\mathcal{D}}_1, \bar{\tau}_1) \between (\bar{\mathcal{D}}_2, \bar{\tau}_2)$ and for all $i \in \mathbb{N}$ we have that $(\bar{\mathcal{D}}_1, \bar{\tau}_1, v, i) \not\models q$ and $(\bar{\mathcal{D}}_2, \bar{\tau}_2, v, i) \not\models p$. Clearly, there is another interleaving $(\bar{\mathcal{D}}', \bar{\tau}') \in (\bar{\mathcal{D}}_1, \bar{\tau}_1) \between (\bar{\mathcal{D}}_2, \bar{\tau}_2)$, where the first two time points are in the opposite order as in $(\bar{\mathcal{D}}, \bar{\tau})$. That is, p is satisfied only on the first time point and q is satisfied only on the second time point. Then $(\bar{\mathcal{D}}', \bar{\tau}', v, 0) \not\models \Box\, p \to \blacklozenge\, q$ and $(\bar{\mathcal{D}}', \bar{\tau}', v, 0) \not\models \phi \land \Box\, p \to \blacklozenge\, q$. As the formula $\phi \land \Box\, p \to \blacklozenge\, q$ is satisfied on one interleaving, but not on another one, the formula is not interleaving-sufficient. □

Given undecidability, we proceed by providing sufficient conditions for ϕ being interleaving-sufficient. We do this by identifying a subset of formulas using a labeling algorithm.

Our algorithm labels the atomic subformulas of the given formula and propagates these labels bottom-up to the formula's root using a fixed set of labeling rules. We use two labels: ONE and ALL. They represent properties that capture the relationship between violations found in one interleaving and violations found in other interleavings. If a formula with the label ONE is satisfied at a time point in one interleaving, then the formula is also satisfied in all other interleavings at the corresponding time point. If a formula with the label ALL is satisfied at a time point with timestamp τ in one interleaving, then the formula is also satisfied in all other interleavings at all time points with the timestamp τ. We formally state these properties in the following definition.

Definition 3.3.5. *Let $(\bar{\mathcal{D}}^1, \bar{\tau}^1)$ and $(\bar{\mathcal{D}}^2, \bar{\tau}^2)$ be two temporal structures and $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ be two arbitrary interleavings from the set $(\bar{\mathcal{D}}^1, \bar{\tau}^1) \between (\bar{\mathcal{D}}^2, \bar{\tau}^2)$.*

– *We say that the formula ϕ* has the property ONE *when the following holds: If $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \phi$, for some valuation v and time point $i \in \mathbb{N}$ then $(\bar{\mathcal{D}}', \bar{\tau}', v, i') \models \phi$ for the time point $i' \in \mathbb{N}$, where i' is the time point corresponding to i. That is, there are $k \in \{1, 2\}$ and $j \in \mathbb{N}$ with*

$$\frac{\phi : \mathsf{ALL}}{\phi : \mathsf{ONE}}$$

$$\frac{}{t \approx t' : \mathsf{ALL}} \qquad \frac{}{t \prec t' : \mathsf{ALL}} \qquad \frac{}{r(t_1, \dots, t_{\iota(r)}) : \mathsf{ONE}}$$

$$\frac{\phi : \mathsf{ONE}}{\neg\phi : \mathsf{ONE}} \qquad \frac{\phi : \mathsf{ALL}}{\neg\phi : \mathsf{ALL}} \qquad \frac{\phi : \mathsf{ONE} \quad \psi : \mathsf{ONE}}{\phi \lor \psi : \mathsf{ONE}} \qquad \frac{\phi : \mathsf{ALL} \quad \psi : \mathsf{ALL}}{\phi \lor \psi : \mathsf{ALL}}$$

$$\frac{\exists x.\, \phi : \mathsf{ONE}}{\exists x.\, \phi : \mathsf{ONE}} \qquad \frac{\exists x.\, \phi : \mathsf{ALL}}{\exists x.\, \phi : \mathsf{ALL}}$$

$$\frac{\phi : \mathsf{ALL} \quad \psi : \mathsf{ALL}}{\phi \,\mathsf{S}_I\, \psi : \mathsf{ALL}} \qquad \frac{\phi : \mathsf{ALL} \quad \psi : \mathsf{ALL}}{\phi \,\mathsf{U}_I\, \psi : \mathsf{ALL}}$$

$$\frac{\phi : \mathsf{ONE}}{\blacklozenge_I\, \phi : \mathsf{ALL}}\ 0 \notin I \qquad \frac{\phi : \mathsf{ONE}}{\lozenge_I\, \phi : \mathsf{ALL}}\ 0 \notin I \qquad \frac{\phi : \mathsf{ONE}}{\blacklozenge_I \lozenge_J\, \phi : \mathsf{ALL}} \qquad \frac{\phi : \mathsf{ONE}}{\lozenge_I \blacklozenge_J\, \phi : \mathsf{ALL}}$$

Figure 3.2: Labeling Rules (Interleaving)

$i = f_k(j)$ and $i' = f'_k(j)$ with f_1, f_2 being the functions used in the interleaving $(\bar{\mathcal{D}}, \bar{\tau})$, and f'_1, f'_2 the functions used in the interleaving $(\bar{\mathcal{D}}', \bar{\tau}')$.

– *We say that the formula ϕ* has the property ALL *when the following holds: If $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \phi$, for some valuation v and time point $i \in \mathbb{N}$ then for all time points $i' \in \mathbb{N}$ with $\tau_i = \tau'_{i'}$, it holds that $(\bar{\mathcal{D}}', \bar{\tau}', v, i') \models \phi$.*

We overload notation and identify each label with its corresponding property. The labeling is done using the rules in Figure 3.2. To improve readability, we use syntactic sugar in the rules. When applying the rules, we assume that syntactic sugar is unfolded in both the rules and the formula. Note that multiple rules may be applicable to a subformula. In this case, multiple labels may be assigned to the subformula. We use the notation $\phi : \ell$ as shorthand for "ϕ's label includes ℓ." By labeling the subformula bottom-up and by attempting to apply all rules before proceeding up to the next subformula, we ensure that a formula is assigned with all possible labels.

Lemma 3.3.6. *Let ϕ be a formula. If ϕ can be labeled with ℓ, then ϕ has the property ℓ, where $\ell \in \{\mathsf{ONE}, \mathsf{ALL}\}$.*

Lemma 3.3.6 shows the soundness of our labeling rules. Before we formally show its correctness, we intuitively explain the most representative rules. The first line in Figure 3.2 shows the weakening rule. The property corresponding to the label ALL implies the property corresponding to ONE.

The next line shows rules for atomic formulas. An atomic formula $t \approx t'$ or $t \prec t'$ depends only on the valuation and therefore can be labeled ALL. An atomic formula of the form $r(t_1, \dots, t_{\iota(r)})$ can be labeled ONE. If a predicate symbol is satisfied at some time point in an interleaving, then it is also satisfied at the corresponding time point in another interleaving. Hence, we label it with ONE. However, we cannot label it with ALL because we do not know whether it is satisfied at other time points with equal timestamps.

Next, we consider labeling rules for the temporal operator $\blacklozenge_I$. If the formula $\blacklozenge_I\, \phi$ is satisfied at time point i, then the subformula ϕ is satisfied at some time point $j \leq i$. If ϕ is labeled ONE, then in any other interleaving the time point corresponding to j also satisfies ϕ. However, if the time points i and j have equal timestamps, then their relative ordering can be exchanged in another interleaving. In this case, the formula $\blacklozenge_I \phi$ would not be satisfied at the time point corresponding to i.

If $0 \notin I$ then the time points i and j must have different timestamps. Therefore, their relative ordering cannot be changed in any interleaving. In this case, not only the time point corresponding to i satisfies the formula $\blacklozenge_I \phi$, but all time points with an equal timestamp as i satisfy this formula. We can therefore propagate the label ONE as ALL if $0 \notin I$, but cannot propagate it if $0 \in I$.

We also consider the case when the subformula ϕ is labeled ALL. In this case, all time points with an equal timestamp as j satisfy ϕ. But then, independent of the relative ordering of these time points, all time points with an equal timestamp as i satisfy $\blacklozenge_I \phi$. Hence, we can propagate ϕ's label ALL to $\blacklozenge_I \phi$ without any restrictions on I. The rule for ALL is not shown in Figure 3.2, but can be derived from the rule for the operator S_I after unfolding the syntactic sugar of $\blacklozenge_I \phi$ into $(\exists x.\, x \approx x)\ \mathsf{S}_I\ \phi$.

We can try to label a formula solely based on labeling rules that involve only a single Boolean or temporal operator. However, by using more specialized labeling rules like the one for $\blacklozenge_I \lozenge_J \psi$, we are more likely to succeed in propagating a label to the formula's root. Intuitively, with the nesting of the operators $\blacklozenge_I$ and $\lozenge_J$, the ordering of equally timestamped time points becomes irrelevant since, from a given time point, we can freely choose any of these time points that satisfy the metric constraints given by the intervals I and J. Hence, a labeling ONE for ψ allows us to label $\blacklozenge_I \lozenge_J \psi$ with ALL.

Finally, there are no labeling rules for the temporal operators $\bullet_I$ and $\bigcirc_I$ because these operators inherently rely on the relative ordering of time points.

Proof. We prove Lemma 3.3.6 by induction on the size of the derivation tree assigning label ℓ to ϕ. We make a case distinction based on the rule 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 two arbitrary interleavings $(\bar{\mathfrak{D}}, \bar{\tau})$ and $(\bar{\mathfrak{D}}', \bar{\tau}')$ of two given temporal structures. We also fix an arbitrary valuation v, an arbitrary time point i in $(\bar{\mathfrak{D}}, \bar{\tau})$, and the time point i' in $(\bar{\mathfrak{D}}', \bar{\tau}')$ corresponding to the time point i.

We first consider the weakening rule. Suppose that $(\bar{\mathfrak{D}}, \bar{\tau}, v, i) \models \phi$. By the induction hypothesis, ϕ has the property ALL, so for all time points $j \in \mathbb{N}$ with $\tau_i = \tau'_j$ we have that $(\bar{\mathfrak{D}}', \bar{\tau}', v, j) \models \phi$. But then the time point i' is among those j's, so $(\bar{\mathfrak{D}}', \bar{\tau}', v, i') \models \phi$ and ϕ has the property ONE.

Next, we make a case distinction on the form of the formula. Consider formulas of the form:

- $t \approx t'$, where t and t' are variables or constants. Suppose that $(\bar{\mathfrak{D}}, \bar{\tau}, v, i) \models t \approx t'$. It follows that $v(t) = v(t')$. As this only depends on the valuation v, we have $(\bar{\mathfrak{D}}', \bar{\tau}', v, j) \models t \approx t'$ for all $j \in \mathbb{N}$ and hence we can add the label ALL to the formula $t \approx t'$.

- $t \prec t'$, where t and t' are variables or constants. This case is similar to the previous one.

- $r(t_1, \dots, t_{\iota(r)})$, where $t_1, \dots, t_{\iota(r)}$ are variables or constants. Suppose that $(\bar{\mathfrak{D}}, \bar{\tau}, v, i) \models r(t_1, \dots, t_{\iota(r)})$. As i' is the time point corresponding to i, it follows that $r^{\mathfrak{D}_i} = r^{\mathfrak{D}'_{i'}}$.

Hence, $(\bar{\mathfrak{D}}', \bar{\tau}', v, i') \models r(t_1, \dots, t_{\iota(r)})$ and $r(t_1, \dots, t_{\iota(r)})$ has the property ONE.

- $\neg\phi$.
  - We first show why the label ONE can be propagated. Suppose that ϕ : ONE and $(\bar{\mathfrak{D}}, \bar{\tau}, v, i) \models \neg\phi$, from which it follows that $(\bar{\mathfrak{D}}, \bar{\tau}, v, i) \not\models \phi$. We claim that from ϕ : ONE it follows that $(\bar{\mathfrak{D}}', \bar{\tau}', v, i') \not\models \phi$. To achieve a contradiction, suppose that $(\bar{\mathfrak{D}}', \bar{\tau}', v, i') \models \phi$. By the induction hypothesis, ϕ has the property ONE, so it follows that $(\bar{\mathfrak{D}}, \bar{\tau}, v, i) \models \phi$, which is a contradiction. Hence, $(\bar{\mathfrak{D}}', \bar{\tau}', v, i') \not\models \phi$, so that $(\bar{\mathfrak{D}}', \bar{\tau}', v, i') \models \neg\phi$ and $\neg\phi$ has the property ONE.
  - Next we show why the label ALL can be propagated. Suppose that ϕ : ALL and $(\bar{\mathfrak{D}}, \bar{\tau}, v, i) \models \neg\phi$, from which it follows that $(\bar{\mathfrak{D}}, \bar{\tau}, v, i) \not\models \phi$. We claim that from ϕ : ALL it follows that $(\bar{\mathfrak{D}}', \bar{\tau}', v, j) \not\models \phi$ for all $j \in \mathbb{N}$ with $\tau_i = \tau'_j$. To achieve a contradiction, suppose that $(\bar{\mathfrak{D}}', \bar{\tau}', v, k) \models \phi$ for some k with $\tau_i = \tau'_k$. By the induction hypothesis, ϕ has the property ALL, so it follows that $(\bar{\mathfrak{D}}, \bar{\tau}, v, \ell) \models \phi$ for all $\ell \in \mathbb{N}$ with $\tau_\ell = \tau'_k = \tau_i$ and hence $(\bar{\mathfrak{D}}, \bar{\tau}, v, i) \models \phi$, which is a contradiction. Therefore, $(\bar{\mathfrak{D}}', \bar{\tau}', v, j) \not\models \phi$, so that $(\bar{\mathfrak{D}}', \bar{\tau}', v, j) \models \neg\phi$ and $\neg\phi$ has the property ALL.
- $\phi \vee \psi$.
  - We first show why the label ONE can be propagated. Suppose that ϕ : ONE, ψ : ONE, and $(\bar{\mathfrak{D}}, \bar{\tau}, v, i) \models \phi \vee \psi$. It follows that 1) $(\bar{\mathfrak{D}}, \bar{\tau}, v, i) \models \phi$ or 2) $(\bar{\mathfrak{D}}, \bar{\tau}, v, i) \models \psi$. If 1), then by the induction hypothesis ϕ has the property ONE and it follows that $(\bar{\mathfrak{D}}', \bar{\tau}', v, i') \models \phi$. If 2), then by the induction hypothesis ψ has the property ONE and it follows that $(\bar{\mathfrak{D}}', \bar{\tau}', v, i') \models \psi$. Therefore, in both cases we have that $(\bar{\mathfrak{D}}', \bar{\tau}', v, i') \models \phi \vee \psi$, so ϕ has the property ONE.
  - The argument why the label ALL can be propagated is analogous.
- $\exists x.\, \phi$.
  - We first show why the label ONE can be propagated. Suppose that ϕ : ONE and $(\bar{\mathfrak{D}}, \bar{\tau}, v, i) \models \exists x.\, \phi$. It follows that $(\bar{\mathfrak{D}}, \bar{\tau}, v[x \hookleftarrow d], i) \models \phi$, for some $d \in |\bar{\mathfrak{D}}|$. Since $|\bar{\mathfrak{D}}| = |\bar{\mathfrak{D}}'|$, d is also in $|\bar{\mathfrak{D}}'|$. By the induction hypothesis, ϕ has the property ONE, and hence $(\bar{\mathfrak{D}}', \bar{\tau}', v[x \hookleftarrow d], i') \models \phi$ and $(\bar{\mathfrak{D}}', \bar{\tau}', v, i') \models \exists x.\, \phi$. Therefore, $\exists x.\, \phi$ has the property ONE.
  - The argument why the label ALL can be propagated is analogous.
- $\phi \mathsf{S}_I \psi$. Suppose that ϕ : ALL, ψ : ALL, and $(\bar{\mathfrak{D}}, \bar{\tau}, v, i) \models \phi \mathsf{S}_I \psi$. It follows that there is a $j \leq i$ such that $\tau_i - \tau_j \in I$, $(\bar{\mathfrak{D}}, \bar{\tau}, v, j) \models \psi$, and $(\bar{\mathfrak{D}}, \bar{\tau}, v, k) \models \phi$ for all k with $j < k \leq i$. By the induction hypothesis, ψ has the property ALL. It follows that $(\bar{\mathfrak{D}}', \bar{\tau}', v, j) \models \psi$ for all time points j' with $\tau_j = \tau'_{j'}$. Let $j'_{\max}$ be the largest of such j'. By the induction hypothesis, ϕ also has the property ALL. It follows that $(\bar{\mathfrak{D}}', \bar{\tau}', v, k') \models \phi$ for all k' with $\tau'_{j'_{\max}} < \tau'_{k'} \leq \tau'_{i'}$. Hence, for every time point i'' with $\tau'_{i''} = \tau'_{i'}$ there is a $j' \leq i''$ such that $\tau'_{i''} - \tau'_{j'} \in I$, $(\bar{\mathfrak{D}}', \bar{\tau}', v, j') \models \psi$, and $(\bar{\mathfrak{D}}', \bar{\tau}', v, k'') \models \phi$ for all k'' with $j'' < k'' \leq i''$. Therefore, $(\bar{\mathfrak{D}}', \bar{\tau}', v, k') \models \phi \mathsf{S}_I \psi$ and $\phi \mathsf{S}_I \psi$ has the property ALL.
- $\phi \mathsf{U}_I \psi$. This case is similar to the previous one.

- $\blacklozenge_I \phi$ with $0 \notin I$.

  Suppose that ϕ : ONE and $(\bar{\mathfrak{D}}, \bar{\tau}, v, i) \models \blacklozenge_I \phi$. There is then a time point j with $j \leq i$, $\tau_i - \tau_j \in I$ such that $(\bar{\mathfrak{D}}, \bar{\tau}, v, j) \models \phi$. By the induction hypothesis, ϕ has the property ONE. It follows that there is a time point j' in $(\bar{\mathfrak{D}}', \bar{\tau}')$ corresponding to j with $(\bar{\mathfrak{D}}', \bar{\tau}', v, j') \models \phi$. From $0 \notin I$ it follows that $\tau_j < \tau_i$, so that $\tau'_{j'} < \tau'_{i'}$, and hence $j' < i''$ for all $i'' \in \mathbb{N}$ with $\tau_i = \tau'_{i''}$. Therefore, $(\bar{\mathfrak{D}}', \bar{\tau}', v, i'') \models \blacklozenge_I \phi$ for all such i'' and $\blacklozenge_I \phi$ has the property ALL.

- $\lozenge_I \phi$ with $0 \notin I$. This case is similar to the previous one.

- $\lozenge_I \blacklozenge_J \phi$.

  Suppose that ϕ : ONE and $(\bar{\mathfrak{D}}, \bar{\tau}, v, i) \models \lozenge_I \blacklozenge_J \phi$, so there are time points j and k with $j \geq i$, $\tau_j - \tau_i \in I$, $k \leq j$, $\tau_j - \tau_k \in J$, and $(\bar{\mathfrak{D}}, \bar{\tau}, v, k) \models \phi$. By the induction hypothesis, ϕ has the property ONE, so there is a time point k' in $(\bar{\mathfrak{D}}', \bar{\tau}')$ corresponding to k with $(\bar{\mathfrak{D}}', \bar{\tau}', v, k') \models \phi$. To satisfy the formula $\lozenge_I \blacklozenge_J \phi$ on $(\bar{\mathfrak{D}}', \bar{\tau}')$ we pick the maximal j' with $\tau'_{j'} = \tau_j$. To see that this satisfies the formula we need to show that 1) $j' \geq i'$, 2) $\tau'_{j'} - \tau'_{i'} \in I$, 3) $k' \leq j'$, and 4) $\tau'_{j'} - \tau'_{k'} \in J$.

  1) From $\tau_j \geq \tau_i$, $\tau'_{j'} = \tau_j$, $\tau'_{i'} = \tau_i$ we see that $\tau'_{j'} \geq \tau'_{i'}$. From j' being the maximal time point with the timestamp $\tau'_{j'}$ it follows that $j' \geq i'$.
  2) From $\tau_j - \tau_i \in I$, $\tau'_{i'} = \tau_i$, and $\tau'_{j'} = \tau_j$ it follows that $\tau'_{j'} - \tau'_{i'} \in I$.
  3) From $\tau_k \leq \tau_j$, $\tau'_{k'} = \tau_k$, $\tau'_{j'} = \tau_j$ we see that $\tau'_{k'} \leq \tau'_{j'}$. From j' being the maximal time point with the timestamp $\tau'_{j'}$, it follows that $k' \leq j'$.
  4) From $\tau_j - \tau_k \in J$, $\tau'_{j'} = \tau_j$, and $\tau'_{k'} = \tau_k$ it follows that $\tau'_{j'} - \tau'_{k'} \in J$.

  Therefore, $(\bar{\mathfrak{D}}', \bar{\tau}', v, i') \models \lozenge_I \blacklozenge_J \phi$ and $\lozenge_I \blacklozenge_J \phi$ has the property ALL.

- $\blacklozenge_I \lozenge_J \phi$. This case is similar to the previous one, but we pick the minimal time point for the temporal operator $\blacklozenge_I$.

$\square$

Based on the labels at a formula's root, we can determine if the formula is interleaving sufficient, as shown in Theorem 3.3.8. To prove this theorem, we first establish in Lemma 3.3.7 a relation between the properties ONE and ALL and the properties (I1) and (I2).

Lemma 3.3.7. *Let ϕ be a formula.*

1. If ϕ has the property ALL*, then ϕ has the properties (I1) and (I2).*

2. If ϕ has the property ONE*, then $\square \phi$ has the properties (I1) and (I2).*

Proof. We fix two arbitrary interleavings $(\bar{\mathfrak{D}}, \bar{\tau})$ and $(\bar{\mathfrak{D}}', \bar{\tau}')$ of two given temporal structures.

1. We first show that ϕ has (I1). Suppose that ϕ has the property ALL and that $(\bar{\mathfrak{D}}, \bar{\tau}, v, 0) \models \phi$ for some valuation v. Since ϕ has the property ALL, it follows that $(\bar{\mathfrak{D}}', \bar{\tau}', v, i') \models \phi$ for all time points i' with $\tau_0 = \tau'_{i'}$. Since $\tau_0 = \tau'_0$, it follows that $(\bar{\mathfrak{D}}', \bar{\tau}', v, 0) \models \phi$ and hence ϕ has (I1).

Next, we show that ϕ has (I2). Suppose that ϕ has the property ALL and that $(\bar{\mathcal{D}}, \bar{\tau}, v, 0) \not\models \phi$ for some valuation v. To achieve a contradiction, suppose that $(\bar{\mathcal{D}}', \bar{\tau}', v, 0) \models \phi$. From this and from ϕ : ALL, it follows that $(\bar{\mathcal{D}}, \bar{\tau}, v, 0) \models \phi$, which is a contradiction. Hence, $(\bar{\mathcal{D}}', \bar{\tau}', v, 0) \not\models \phi$ and ϕ has (I2).

2. We first show that $\Box\,\phi$ has (I1). Suppose that ϕ has the property ONE and that $(\bar{\mathcal{D}}, \bar{\tau}, v, 0) \models \Box\,\phi$ for some valuation v. Then $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \phi$ for all time points $i \in \mathbb{N}$. Since ϕ has the property ONE, it follows that $(\bar{\mathcal{D}}', \bar{\tau}', v, i') \models \phi$ for all corresponding time points $i' \in \mathbb{N}$. Hence $(\bar{\mathcal{D}}', \bar{\tau}', v, 0) \models \Box\,\phi$ and $\Box\,\phi$ has (I1).

   We continue to show that $\Box\,\phi$ has also (I2). Suppose that ϕ has the property ONE and that $(\bar{\mathcal{D}}, \bar{\tau}, v, 0) \not\models \Box\,\phi$ for some valuation v. Then $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \phi$ for some time point $i \in \mathbb{N}$.

   To achieve a contradiction, suppose that $(\bar{\mathcal{D}}', \bar{\tau}', v, i') \models \phi$, where i' is the time point corresponding to i. But from this and ϕ : ONE it follows that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \phi$, which is a contradiction. Hence, $(\bar{\mathcal{D}}', \bar{\tau}', v, i') \not\models \phi$, so that $(\bar{\mathcal{D}}', \bar{\tau}', v, 0) \not\models \Box\,\phi$ and $\Box\,\phi$ has (I2).

□

Theorem 3.3.8. *Let ϕ be a formula.*

1. If ϕ is labeled ALL*, then ϕ is interleaving-sufficient.*

2. If ϕ is labeled ONE*, then $\Box\,\phi$ is interleaving-sufficient.*

Moreover, we can determine in linear time in the formula's length whether ϕ can be labeled by ONE *or* ALL*.*

Proof. The implications in Theorem 3.3.8 follow directly from the correctness of the labeling rules (Lemma 3.3.6) and from Lemma 3.3.7.

Next, now prove that a formula ϕ can be labeled in time linear in its length. We start with some definitions and then present a simple labeling algorithm and analyze its complexity.

For a formula ϕ, we define its immediate subformulas $isub(\phi)$ to be: (i) $\{\psi\}$ if $\phi = \neg\psi$, $\phi = \exists x.\,\psi$, $\phi = \CIRCLE_I\,\psi$, or $\phi = \Circle_I\,\psi$; (ii) $\{\psi,\chi\}$ if $\phi = \psi \wedge \chi$, $\phi = \psi\ \mathsf{S}_I\,\chi$, or $\phi = \psi\ \mathsf{U}_I\,\chi$; and (iii) $\emptyset$ otherwise. For a rule r, we denote $\ell(r)$ the label of the rule's conclusion.

We assume that the data structure used to represent formulas is a tree corresponding to the formula's syntax tree and that each node in the tree also stores two bits to represent the two different labels. Initially these bits are set to 0, meaning that no label is associated with the corresponding subformula.

```
add_labels($\phi$)
  foreach $\psi \in isub(\phi)$
    add_labels($\psi$)
  foreach rule $r$
    if $matches(\phi, r)$ then
      add_label($\phi$, $\ell(r)$)
```

The function $matches(\phi, r)$ checks if the formula ϕ pattern matches a rule r. The order of rules is arbitrary, with the exception that the weakening rules are checked last. So, for instance if ϕ received label ALL, then ϕ will match the appropriate weakening rule and it will also be labeled with ONE. For each rule, at most the first four levels of the tree representing the formula ϕ need to be inspected. For example, after unfolding syntactic sugar in the rule for the formula $\blacklozenge_I \lozenge_J \phi$, the formula becomes $(\exists x.\, x \approx x)\ \mathsf{S}_I\ (\exists y.\, y \approx y\ \mathsf{U}_I\ \phi)$. As rules have constant size, we conclude that the function executes in constant time.

The function *add_label*(ϕ, ℓ) simply adds the label ℓ to ϕ. Clearly, this operation can be performed in constant time.

Note that the execution of the lines 2 and 4–6 takes constant time: $|isub(\phi)| \leq 2$ for any ϕ, there is a fixed, constant number of rules, and the functions *matches* and *add_label* execute in constant time. Furthermore, the function *add_labels* is executed once for each subformula of ϕ. Hence the whole labeling procedure of ϕ takes time linear in the length of ϕ. □

Example 3.3.9. *We illustrate our algorithm by applying it to the formula* $\Box\, \forall x.\, publish(x) \to \blacklozenge_{[0,11)}\, approve(x)$. *We first remove some syntactic sugar and obtain the formula* $\Box\, \forall x.\, \neg publish(x) \vee \blacklozenge_{[0,11)}\, approve(x)$. *We start by labeling the atomic subformulas. Both* $publish(x)$ *and* $approve(x)$ *are labeled with* ONE. *Hence, the subformula* $\neg publish(x)$ *is also labeled with* ONE. *However, we cannot propagate the label* ONE *to the subformula* $\blacklozenge_{[0,11)}\, approve(x)$ *because the interval of the operator* $\blacklozenge$ *includes* 0. *We therefore cannot propagate any labels to the subformulas* $\neg publish(x) \vee \blacklozenge_{[0,11)}\, approve(x)$ *and* $\forall x.\, \neg publish(x) \vee \blacklozenge_{[0,11)}\, approve(x)$. *We conclude that the formula* $\Box\, \forall x.\, \neg publish(x) \vee \blacklozenge_{[0,11)}\, approve(x)$ *is not in the interleaving-sufficient fragment. It is not even interleaving-sufficient, as explained in Example 3.3.2.*

The formula $\Box\, \forall x.\, publish(x) \to \blacklozenge_{[1,11)}\, approve(x)$ *is interleaving-sufficient. The labeling starts similarly but* $\blacklozenge_{[1,11)}\, approve(x)$ *can be labeled with* ALL *since the interval of the temporal operator does not contain* 0. *This label is weakened to* ONE *and propagates to the formula* $\forall x.\, \neg publish(x) \vee \blacklozenge_{[1,11)}\, approve(x)$. *We conclude that* $\Box\, \forall x.\, \neg publish(x) \vee \blacklozenge_{[1,11)}\, approve(x)$ *is interleaving-sufficient.*

Note that the defined fragment is sound, but incomplete. In particular, the converse of statements 1 and 2 in Theorem 3.3.8 is false. For example, the formula $\Box\, \forall x.\, publish(x) \to (\lozenge_{[0,1)}\, approve(x)) \vee \blacklozenge_{[0,11)}\, approve(x)$ is interleaving-sufficient, but cannot be labeled as required by Theorem 3.3.8. However, the semantically equivalent formula $\Box\, \forall x.\, publish(x) \to \lozenge_{[0,1)} \blacklozenge_{[0,11)}\, approve(x)$ is recognized as interleaving-sufficient by the rules in Figure 3.2. We could enlarge the fragment by adding rules that handle this particular formula. However, since it is undecidable whether a formula is interleaving-sufficient (Theorem 3.3.4) we cannot make the syntactically-defined fragment decidable, sound, and complete.

3.4 Monitoring the Collapse

In this section we describe a collapse-sufficient fragment. Intuitively, collapse-sufficient formulas are those formulas that do not yield false positives and false negatives when monitoring the collapse of an interleaving:

Definition 3.4.1. *Let* ϕ *be a formula. For* $k \in \{1,2\}$, *we say that* ϕ has the property (Ck) *if* $(\bar{\mathfrak{C}}, \bar{\kappa})$ *fulfills the condition (Sk) in Definition 3.2.1 with respect to* ϕ *and* $(\bar{\mathfrak{D}}, \bar{\tau}) \between (\bar{\mathfrak{D}}', \bar{\tau}')$, *for every* $(\bar{\mathfrak{D}}, \bar{\tau})$, $(\bar{\mathfrak{D}}', \bar{\tau}')$, *and* $(\bar{\mathfrak{C}}, \bar{\kappa})$, *where* $(\bar{\mathfrak{C}}, \bar{\kappa})$ *is the collapse of an interleaving of* $(\bar{\mathfrak{D}}, \bar{\tau})$ *and* $(\bar{\mathfrak{D}}', \bar{\tau}')$. *Moreover,* ϕ *is* collapse-sufficient *if it has the properties (C1) and (C2).*

Monitoring the collapse with respect to a collapse-sufficient formula is correct for strong violations. Since strong violations are trivially also weak violations, we detect some weak violations as well. However, we may miss violations that are weak, but not strong.

Note that the formula from the example in Section 2.2 is not collapse-sufficient, but the weaker and stronger formulas from Example 3.3.2 are collapse-sufficient. Also note that stutter-invariance [Lam83] is a necessary condition for collapse-sufficiency. However, it is not a sufficient condition. For example, the formula $\Box\, \forall x.\, p(x) \land q(x)$ is stuttering-invariant but not collapse-sufficient.

As with interleaving-sufficient formulas, it is undecidable whether a formula is collapse-sufficient, as stated in Theorem 3.4.2.

Theorem 3.4.2. *Given an MFOTL formula ϕ, it is undecidable whether ϕ is collapse-sufficient.*

The proof of Theorem 3.4.2 is analogous to the proof of Theorem 3.3.4 (the interleaving-sufficient case), but we consider the formula $\phi \land \Box\, \exists x\, .p(x)$. We omit the proof.

Our collapse-sufficient fragment is, similar to the interleaving-sufficient fragment in Section 3.3, defined by a labeling algorithm. The labels represent properties, which capture the relation between violations found in a collapsed temporal structure at some time point and violations found in pre-images of the collapsing at a time point with an equal timestamp. We formally state these properties in the following definition.

Definition 3.4.3. *Let $(\bar{\mathfrak{C}}, \bar{\kappa})$ be a collapsed temporal structure and let $col^{-1}(\bar{\mathfrak{C}}, \bar{\kappa})$ denote the pre-images of collapsing, that is, the set of temporal structures $(\bar{\mathfrak{D}}, \bar{\tau})$ with $col(\bar{\mathfrak{D}}, \bar{\tau}) = (\bar{\mathfrak{C}}, \bar{\kappa})$.*

- *The formula ϕ* has the property $(\models \forall)$ *when the following holds: For all valuations v and all $i \in \mathbb{N}$, if $(\bar{\mathfrak{C}}, \bar{\kappa}, v, i) \models \phi$ then for every $(\bar{\mathfrak{D}}, \bar{\tau}) \in col^{-1}(\bar{\mathfrak{C}}, \bar{\kappa})$ and every $j \in \mathbb{N}$ with $\kappa_i = \tau_j$, it holds that $(\bar{\mathfrak{D}}, \bar{\tau}, v, j) \models \phi$.*
- *The formula ϕ* has the property $(\models \exists)$ *when the following holds: For all valuations v and all $i \in \mathbb{N}$, if $(\bar{\mathfrak{C}}, \bar{\kappa}, v, i) \models \phi$ then for every $(\bar{\mathfrak{D}}, \bar{\tau}) \in col^{-1}(\bar{\mathfrak{C}}, \bar{\kappa})$, there is some $j \in \mathbb{N}$ with $\kappa_i = \tau_j$ such that $(\bar{\mathfrak{D}}, \bar{\tau}, v, j) \models \phi$.*
- *The formula ϕ* has the property $(\not\models \forall)$ *when the following holds: For all valuations v and all $i \in \mathbb{N}$, if $(\bar{\mathfrak{C}}, \bar{\kappa}, v, i) \not\models \phi$ then for every $(\bar{\mathfrak{D}}, \bar{\tau}) \in col^{-1}(\bar{\mathfrak{C}}, \bar{\kappa})$ and every $j \in \mathbb{N}$ with $\kappa_i = \tau_j$, it holds that $(\bar{\mathfrak{D}}, \bar{\tau}, v, j) \not\models \phi$.*
- *The formula ϕ* has the property $(\not\models \exists)$ *when the following holds: For all valuations v and all $i \in \mathbb{N}$, if $(\bar{\mathfrak{C}}, \bar{\kappa}, v, i) \not\models \phi$ then for every $(\bar{\mathfrak{D}}, \bar{\tau}) \in col^{-1}(\bar{\mathfrak{C}}, \bar{\kappa})$, there is some $j \in \mathbb{N}$ with $\kappa_i = \tau_j$ such that $(\bar{\mathfrak{D}}, \bar{\tau}, v, j) \not\models \phi$.*

The first symbol ($\models$ or $\not\models$) in a property indicates whether the formula is satisfied in the collapsed temporal structure $(\bar{\mathfrak{C}}, \bar{\kappa})$. The second symbol ($\exists$ or $\forall$) states whether the formula is satisfied at all equally timestamped time points or at some equally timestamped time point in all temporal structures $(\bar{\mathfrak{D}}, \bar{\tau}) \in col^{-1}(\bar{\mathfrak{C}}, \bar{\kappa})$.

Again, we overload notation and identify each label with its corresponding property. Figure 3.3 lists the labeling rules. In addition, Figure 3.4 lists rules for the Boolean operator $\land$, the quantifier $\forall$, and the temporal operators trigger T_I and release R_I. These rules are used for formulas in positive normal form, which we require in Section 3.5. Recall that formulas in this normal form are obtained by pushing negation inside until it appears only in front of atomic formulas. When considering these formulas, the operators $\land$, $\forall$, T_I, and R_I are seen as primitives, instead of being defined as syntactic sugar. We recall that $\psi\ \mathsf{T}_I\ \chi$ abbreviates $\neg(\neg\psi\ \mathsf{S}_I\ \neg\chi)$ and $\psi\ \mathsf{R}_I\ \chi$ abbreviates $\neg(\neg\psi\ \mathsf{U}_I\ \neg\chi)$.

$$\frac{\phi : (\models \forall)}{\phi : (\models \exists)} \qquad \frac{\phi : (\not\models \forall)}{\phi : (\not\models \exists)}$$

$$\frac{}{t \approx t' : (\models \forall)} \qquad \frac{}{t \approx t' : (\not\models \forall)} \qquad \frac{}{t \prec t' : (\models \forall)} \qquad \frac{}{t \prec t' : (\not\models \forall)}$$

$$\frac{}{r(t_1, \dots, t_{\iota(r)}) : (\models \exists)} \qquad \frac{}{r(t_1, \dots, t_{\iota(r)}) : (\not\models \forall)}$$

$$\frac{\psi : (\models \exists)}{\neg\psi : (\not\models \exists)} \qquad \frac{\psi : (\models \forall)}{\neg\psi : (\not\models \forall)} \qquad \frac{\psi : (\not\models \exists)}{\neg\psi : (\models \exists)} \qquad \frac{\psi : (\not\models \forall)}{\neg\psi : (\models \forall)}$$

$$\frac{\psi : (\not\models \forall) \quad \chi : (\not\models \forall)}{\psi \vee \chi : (\not\models \forall)} \qquad \frac{\psi : (\not\models \forall) \quad \chi : (\not\models \exists)}{\psi \vee \chi : (\not\models \exists)}$$

$$\frac{\psi : (\models \forall) \quad \chi : (\models \forall)}{\psi \vee \chi : (\models \forall)} \qquad \frac{\psi : (\models \exists) \quad \chi : (\models \exists)}{\psi \vee \chi : (\models \exists)}$$

$$\frac{\psi : (\models \forall)}{\exists x. \psi : (\models \forall)} \qquad \frac{\psi : (\models \exists)}{\exists x. \psi : (\models \exists)} \qquad \frac{\psi : (\not\models \forall)}{\exists x. \psi : (\not\models \forall)} \qquad \frac{\psi : (\not\models \exists)}{\exists x. \psi : (\not\models \exists)}$$

$$\frac{\psi : (\models \forall) \quad \chi : (\models \forall)}{\psi \,\mathsf{S}_I\, \chi : (\models \forall)} \qquad \frac{\psi : (\not\models \forall) \quad \chi : (\not\models \forall)}{\psi \,\mathsf{S}_I\, \chi : (\not\models \forall)} \qquad \frac{\psi : (\not\models \exists) \quad \chi : (\not\models \forall)}{\psi \,\mathsf{S}_I\, \chi : (\not\models \exists)}$$

$$\frac{\psi : (\models \forall) \quad \chi : (\models \forall)}{\psi \,\mathsf{U}_I\, \chi : (\models \forall)} \qquad \frac{\psi : (\not\models \forall) \quad \chi : (\not\models \forall)}{\psi \,\mathsf{U}_I\, \chi : (\not\models \forall)} \qquad \frac{\psi : (\not\models \exists) \quad \chi : (\not\models \forall)}{\psi \,\mathsf{U}_I\, \chi : (\not\models \exists)}$$

$$\frac{\psi : (\not\models \exists) \quad \chi : (\not\models \forall)}{(\psi \,\mathsf{S}_I\, \chi) \wedge (\Box_J\, \psi) : (\not\models \forall)}\ 0 \notin I, 0 \in J \qquad \frac{\psi : (\not\models \exists) \quad \chi : (\not\models \forall)}{(\psi \,\mathsf{U}_I\, \chi) \wedge (\blacksquare_J\, \psi) : (\not\models \forall)}\ 0 \notin I, 0 \in J$$

$$\frac{\psi : (\models \exists)}{\blacklozenge_I\, \psi : (\models \exists)} \qquad \frac{\psi : (\models \exists)}{\blacklozenge_I\, \psi : (\models \forall)}\ 0 \notin I \qquad \frac{\psi : (\models \exists)}{\Diamond_I\, \psi : (\models \exists)} \qquad \frac{\psi : (\models \exists)}{\Diamond_I\, \psi : (\models \forall)}\ 0 \notin I$$

$$\frac{\psi : (\models \exists)}{\blacklozenge_I \Diamond_J\, \psi : (\models \forall)}\ 0 \in I \cap J \qquad \frac{\psi : (\models \exists)}{\Diamond_I \blacklozenge_J\, \psi : (\models \forall)}\ 0 \in I \cap J$$

Figure 3.3: Labeling Rules (Collapse)

$$\frac{\psi : (\models\forall) \quad \chi : (\models\forall)}{\psi \wedge \chi : (\models\forall)} \qquad \frac{\psi : (\models\forall) \quad \chi : (\models\exists)}{\psi \wedge \chi : (\models\exists)}$$

$$\frac{\psi : (\not\models\forall) \quad \chi : (\not\models\forall)}{\psi \wedge \chi : (\not\models\forall)} \qquad \frac{\psi : (\not\models\exists) \quad \chi : (\not\models\exists)}{\psi \wedge \chi : (\not\models\exists)}$$

$$\frac{\psi : (\models\forall)}{\forall x.\, \psi : (\models\forall)} \qquad \frac{\psi : (\models\exists)}{\forall x.\, \psi : (\models\exists)} \qquad \frac{\psi : (\not\models\forall)}{\forall x.\, \psi : (\not\models\forall)} \qquad \frac{\psi : (\not\models\exists)}{\forall x.\, \psi : (\not\models\exists)}$$

$$\frac{\psi : (\not\models\forall) \quad \chi : (\not\models\forall)}{\psi \,\mathsf{T}_I\, \chi : (\not\models\forall)} \qquad \frac{\psi : (\models\forall) \quad \chi : (\models\forall)}{\psi \,\mathsf{T}_I\, \chi : (\models\forall)}$$

$$\frac{\psi : (\models\exists) \quad \chi : (\models\forall)}{\psi \,\mathsf{T}_I\, \chi : (\models\exists)} \qquad \frac{\psi : (\models\exists) \quad \chi : (\models\forall)}{(\psi \,\mathsf{T}_I\, \chi) \vee (\Diamond_J\, \psi) : (\models\forall)}\; 0 \notin I, 0 \in J$$

$$\frac{\psi : (\not\models\forall) \quad \chi : (\not\models\forall)}{\psi \,\mathsf{R}_I\, \chi : (\not\models\forall)} \qquad \frac{\psi : (\models\forall) \quad \chi : (\models\forall)}{\psi \,\mathsf{R}_I\, \chi : (\models\forall)}$$

$$\frac{\psi : (\models\exists) \quad \chi : (\models\forall)}{\psi \,\mathsf{R}_I\, \chi : (\models\exists)} \qquad \frac{\psi : (\models\exists) \quad \chi : (\models\forall)}{(\psi \,\mathsf{R}_I\, \chi) \vee (\blacklozenge_J\, \psi) : (\models\forall)}\; 0 \notin I, 0 \in J$$

Figure 3.4: Labeling Rules for Formulas in Positive Normal Form

Lemma 3.4.4. *Let ϕ be a formula. If ϕ can be labeled with ℓ, then ϕ has the property ℓ, where $\ell \in \{(\models\forall), (\not\models\forall), (\models\exists), (\not\models\exists)\}$.*

Lemma 3.4.4 shows the soundness of our labeling rules. Before we formally show its correctness, we intuitively explain the most representative rules. The first two rules in Figure 3.3 express that the properties corresponding to the labels $(\models\forall)$ and $(\not\models\forall)$ imply the properties corresponding to $(\models\exists)$ and $(\not\models\exists)$, respectively.

The next two lines in Figure 3.3 are rules for atomic formulas. An atomic formula $t \approx t'$ or $t \prec t'$ depends only on the valuation and therefore can be labeled $(\models\forall)$ and $(\not\models\forall)$. An atomic formula of the form $r(t_1, \dots, t_{\iota(r)})$ can be labeled $(\models\exists)$ and $(\not\models\forall)$. We only explain the labeling $(\models\exists)$. The explanation for the label $(\not\models\forall)$ is analogous. The interpretation of a predicate symbol in a collapsed temporal structure $(\bar{\mathfrak{C}}, \bar{\kappa})$ at a time point i is the union of the predicate symbol's interpretations at all time points j in a temporal structure $(\bar{\mathfrak{D}}, \bar{\tau}) \in col^{-1}(\bar{\mathfrak{C}}, \bar{\kappa})$ for which τ_j equals κ_i. Therefore, if $\bar{a} \in r^{\mathfrak{C}_i}$ then $\bar{a} \in r^{\mathfrak{D}_j}$, for some $j \in \mathbb{N}$ with $\tau_j = \kappa_i$. Note that $\bar{a} \in r^{\mathfrak{D}_j}$ does not necessarily hold for all of these js; hence, we cannot label $r(t_1, \dots, t_{\iota(r)})$ with $(\models\forall)$.

We next consider the labeling rules for the temporal operator S_I. To ease our explanation, we just consider the special case $\blacklozenge_I\, \psi = \mathit{true}\ \mathsf{S}_I\, \psi$. We first justify the rule that propagates the label $(\models\forall)$ from ψ to $\blacklozenge_I\, \psi$. It is not shown in Figure 3.3, but can be derived from the rule for the operator S_I after unfolding the syntactic sugar $\blacklozenge_I\, \phi$, by observing that *true*, which unfolds to $\exists x.\, x \approx x$, can be labeled with $(\models\forall)$. If $\blacklozenge_I\, \psi$ is satisfied in the collapsed temporal structure $(\bar{\mathfrak{C}}, \bar{\kappa})$ at time point i then ψ is satisfied at some previous time point $j \le i$ in $(\bar{\mathfrak{C}}, \bar{\kappa})$ with $\kappa_i - \kappa_j \in I$. Because ψ is labeled with $(\models\forall)$, all time points with timestamp κ_j in the temporal structure $(\bar{\mathfrak{D}}, \bar{\tau}) \in col^{-1}(\bar{\mathfrak{C}}, \bar{\kappa})$ also satisfy ψ, and hence, all time points with timestamp κ_i satisfy $\blacklozenge_I\, \psi$ in $(\bar{\mathfrak{D}}, \bar{\tau})$. When ψ is labeled with $(\models\exists)$, possibly only a single time point k in $(\bar{\mathfrak{D}}, \bar{\tau})$ with $\tau_k = \kappa_j$

satisfies ψ. If $0 \in I$ then $\blacklozenge_I \psi$ might not be satisfied at time points before k, even if these time points have the timestamp κ_i. So, we can label $\blacklozenge_I$ with $(\models \exists)$ but not with $(\models \forall)$. However, if $0 \notin I$ then ψ is satisfied in $(\bar{\mathcal{C}}, \bar{\kappa})$ at a time point j with the timestamp $\kappa_j < \kappa_i$. Hence $\blacklozenge_I \psi$ is satisfied in $(\bar{\mathcal{D}}, \bar{\tau})$ at all time points with the timestamp κ_i. This allows us to label $\blacklozenge_I \psi$ with $(\models \forall)$. Finally, when ψ is labeled $(\not\models \forall)$, then $\blacklozenge_I \psi$ can also be labeled with $(\not\models \forall)$. This rule is not shown in Figure 3.3, but it can be derived from the rule for the operator S_I, like the rule for the label $(\models \forall)$. If $\blacklozenge_I \psi$ is violated in the collapsed temporal structure $(\bar{\mathcal{C}}, \bar{\kappa})$ at timestamp κ_i, then ψ is violated at all previous points in the temporal structure $(\bar{\mathcal{D}}, \bar{\tau}) \in col^{-1}(\bar{\mathcal{C}}, \bar{\kappa})$ that satisfy the metric constraints given by I. But then $\blacklozenge_I \psi$ is also violated in $(\bar{\mathcal{D}}, \bar{\tau})$ at all time points with the timestamp κ_i. Hence we can label $\blacklozenge_I \psi$ with $(\not\models \forall)$.

We can try to label a formula solely based on labeling rules that involve only a single Boolean or temporal operator. However, with additional specialized labeling rules like the one for $\blacklozenge_I \Diamond_J \psi$, we are more likely to succeed in propagating labels to the root of the formula. Intuitively, with the nesting of the operators $\blacklozenge_I$ and $\Diamond_J$, and when $0 \in I \cap J$, the ordering of equally timestamped time points becomes irrelevant since from a given time point, we can freely choose any of the time points that satisfy the metric constraints given by the intervals I and J. Hence, a labeling $(\models \exists)$ for ψ allows us to label $\blacklozenge_I \Diamond_J \psi$ with $(\models \forall)$.

Proof. We first show the correctness of the labeling rules from Figure 3.3. We proceed by induction on the size of the derivation tree assigning label ℓ to ϕ. We make a case distinction based on the rule 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.

Let $(\bar{\mathcal{C}}, \bar{\kappa})$ be the collapse of an interleaving of two given temporal structures. For readability, and without loss of generality, we already fix an arbitrary valuation v, an arbitrary time point i, and an arbitrary temporal structure $(\bar{\mathcal{D}}, \bar{\tau}) \in col^{-1}(\bar{\mathcal{C}}, \bar{\kappa})$.

We first consider the weakening rules:

- ϕ is labeled with $(\models \forall)$ and $(\models \exists)$. Suppose that $(\bar{\mathcal{C}}, \bar{\kappa}, v, i) \models \phi$. By the induction hypothesis, ϕ has the property $(\models \forall)$, thus $(\bar{\mathcal{D}}, \bar{\tau}, v, j) \models \phi$ for any j with $\tau_j = \kappa_i$. By the definition of $(\bar{\mathcal{C}}, \bar{\kappa})$, there is at least one j with $\tau_j = \kappa_i$. Hence ϕ has the property $(\models \exists)$.
- ϕ is labeled with $(\not\models \forall)$ and with $(\not\models \exists)$. This case is analogous to the previous one.

Next, we make a case distinction on the form of the formula. Consider formulas of the form:

- $\phi = t \approx t'$, where t and t' are variables or constants. In this case ϕ is labeled with $(\models \forall)$ and $(\not\models \forall)$.
  - ϕ is labeled with $(\models \forall)$. Suppose that $(\bar{\mathcal{C}}, \bar{\kappa}, v, i) \models \phi$. Then $v(t) = v(t')$. Clearly, $(\bar{\mathcal{D}}, \bar{\tau}, v, j) \models \phi$ for any time point j, as ϕ only depends on the valuation. The property $(\models \forall)$ is hence satisfied.
  - ϕ is labeled with $(\not\models \forall)$. This case is analogous to the previous one.
- $\phi = t \prec t'$, where t and t' are variables or constants. This case is analogous to the previous one.
- $\phi = r(t_1, \dots, t_{\iota(r)})$, where $t_1, \dots, t_{\iota(r)}$ are variables or constants. In this case ϕ is labeled with $(\models \exists)$ and $(\not\models \forall)$.

- ϕ is labeled with ($\models\exists$). Suppose that $(\bar{\mathfrak{C}}, \bar{\kappa}, v, i) \models \phi$. Then $(v(t_1), \ldots, v(t_{\iota(r)})) \in r^{\mathfrak{C}_i}$. As $r^{\mathfrak{C}_i} = \bigcup_{\{j \mid \tau_j = \kappa_i\}} r^{\mathfrak{D}_j}$, there is a j with $\tau_j = \kappa_i$ such that $(v(t_1), \ldots, v(t_{\iota(r)})) \in r^{\mathfrak{D}_j}$. Therefore $(\bar{\mathfrak{D}}, \bar{\tau}, v, j) \models \phi$. Thus ϕ has the property ($\models\exists$).
- ϕ is labeled with ($\not\models\forall$). Suppose that $(\bar{\mathfrak{C}}, \bar{\kappa}, v, i) \not\models \phi$. Then for any j with $\tau_j = \kappa_i$ we have that $(v(t_1), \ldots, v(t_{\iota(r)})) \notin r^{\mathfrak{D}_j}$, that is, $(\bar{\mathfrak{D}}, \bar{\tau}, v, j) \not\models \phi$. Thus ϕ has the property ($\not\models\forall$).

- $\phi = \neg\psi$. If ψ is labeled with ℓ, then ϕ is labeled with $\neg\ell$, where $\neg\ell$ is ($\models\forall$), ($\not\models\forall$), ($\not\models\exists$), or ($\models\exists$) when ℓ is ($\not\models\forall$), ($\models\forall$), ($\models\exists$), or ($\not\models\exists$), respectively.
  - ϕ is labeled with ($\models\forall$). Suppose that $(\bar{\mathfrak{C}}, \bar{\kappa}, v, i) \models \neg\psi$. By the induction hypothesis, ψ has the property ($\not\models\forall$). As $(\bar{\mathfrak{C}}, \bar{\kappa}, v, i) \not\models \psi$, we have that $(\bar{\mathfrak{D}}, \bar{\tau}, v, k) \not\models \psi$, that is, $(\bar{\mathfrak{D}}, \bar{\tau}, v, k) \models \phi$, for all k with $\tau_k = \kappa_i$. Thus ϕ has the property ($\models\forall$).
  - The other cases are similar.

- $\phi = \psi \vee \chi$. There are four rules to be analyzed.
  - ϕ, ψ, and χ are labeled with ($\not\models\forall$). Suppose that $(\bar{\mathfrak{C}}, \bar{\kappa}, v, i) \not\models \psi \vee \chi$. Then $(\bar{\mathfrak{C}}, \bar{\kappa}, v, i) \not\models \psi$ and $(\bar{\mathfrak{C}}, \bar{\kappa}, v, i) \not\models \chi$. By the induction hypothesis, ψ and χ have the property ($\not\models\forall$). Hence, for all j with $\tau_j = \kappa_i$, we have $(\bar{\mathfrak{D}}, \bar{\tau}, v, j) \not\models \psi$ and $(\bar{\mathfrak{D}}, \bar{\tau}, v, j) \not\models \chi$. Thus $(\bar{\mathfrak{D}}, \bar{\tau}, v, j) \not\models \phi$ for all j with $\tau_j = \kappa_i$. Hence, ϕ has the property ($\not\models\forall$).
  - The other cases are similar.

- $\phi = \exists x.\psi$. There are four rules, one for each label: if ψ is labeled with ℓ, then ϕ is labeled with ℓ.
  - ℓ is ($\models\forall$). Suppose that $(\bar{\mathfrak{C}}, \bar{\kappa}, v, i) \models \exists x.\psi$. Then there is a $d \in |\bar{\mathfrak{D}}|$ such that $(\bar{\mathfrak{C}}, \bar{\kappa}, v[x \hookleftarrow d], i) \models \psi$. As ψ has the property ($\models\forall$), we have $(\bar{\mathfrak{D}}, \bar{\tau}, v[x \hookleftarrow d], j) \models \psi$ for all j with $\tau_j = \kappa_i$. That is, $(\bar{\mathfrak{D}}, \bar{\tau}, v, j) \models \exists x.\psi$ for all j with $\tau_j = \kappa_i$. Hence ϕ has the property ($\models\forall$).
  - The other cases are similar.

- $\phi = \psi \, \mathsf{S}_I \, \chi$. We have three rules to analyze.
  - ϕ, ψ, and χ are each labeled with ($\models\forall$). By the induction hypothesis, ψ and χ have the property ($\models\forall$). Suppose that $(\bar{\mathfrak{C}}, \bar{\kappa}, v, i) \models \phi$. Then, for some $j \le i$ with $\kappa_i - \kappa_j \in I$, we have $(\bar{\mathfrak{C}}, \bar{\kappa}, v, j) \models \chi$ and $(\bar{\mathfrak{C}}, \bar{\kappa}, v, k) \models \psi$ for all $k \in [j+1, i+1)$. Let i' be an arbitrary time point such that $\tau_{i'} = \kappa_i$. As χ has the property ($\models\forall$), for the largest j' with $\tau_{j'} = \kappa_j$ we have $(\bar{\mathfrak{D}}, \bar{\tau}, v, j') \models \chi$. Clearly, $\tau_{i'} - \tau_{j'} \in I$. From the definition of $(\bar{\mathfrak{C}}, \bar{\kappa})$, for any $k' \in [j'+1, i'+1)$, there is a $k \in [j+1, i+1)$ such that $\tau_{k'} = \kappa_k$. Then, as ψ has the property ($\models\forall$), for any $k' \in [j'+1, i'+1)$, we have $(\bar{\mathfrak{D}}, \bar{\tau}, v, k') \models \psi$. As ψ has the property ($\models\forall$), for all $k \in [j+1, i+1)$ and all k' with $\tau_{k'} = \kappa_k$, we have $(\bar{\mathfrak{D}}, \bar{\tau}, v, k') \models \psi$. Hence $(\bar{\mathfrak{D}}, \bar{\tau}, v, i') \models \psi \, \mathsf{S}_I \, \chi$, and thus ϕ has the property ($\models\forall$).
  - ϕ, ψ, and χ are each labeled with ($\not\models\forall$). By the induction hypothesis, ψ and χ have the property ($\not\models\forall$). Suppose that $(\bar{\mathfrak{C}}, \bar{\kappa}, v, i) \not\models \phi$. Furthermore, to achieve a

contradiction, suppose that ϕ does not have the property ($\not\models\forall$). That is, there is an i' with $\tau_{i'} = \kappa_i$ such that $(\bar{\mathcal{D}}, \bar{\tau}, v, i') \models \phi$. Then there is a $j' \leq i'$ with $\tau_{i'} - \tau_{j'} \in I$ such that $(\bar{\mathcal{D}}, \bar{\tau}, v, j') \models \chi$ and for all $k' \in [j'+1, i'+1)$ we have $(\bar{\mathcal{D}}, \bar{\tau}, v, k) \models \psi$. By the definition of $(\bar{\mathcal{C}}, \bar{\kappa})$, there is a j with $\kappa_j = \tau_{j'}$. As χ has the property ($\not\models\forall$), we have that $(\bar{\mathcal{C}}, \bar{\kappa}, v, j) \models \chi$. Similarly, we have that $(\bar{\mathcal{C}}, \bar{\kappa}, v, k) \models \psi$ for all $k \in [j+1, i+1)$. That is, $(\bar{\mathcal{C}}, \bar{\kappa}, v, i) \models \phi$, which is a contradiction.

– ϕ and ψ are labeled with ($\not\models\exists$), and χ is labeled by ($\not\models\forall$). By the induction hypothesis, ψ and χ have the properties ($\not\models\exists$) and ($\not\models\forall$), respectively. As before, suppose that $(\bar{\mathcal{C}}, \bar{\kappa}, v, i) \not\models \phi$. Furthermore, to achieve a contradiction, suppose that ϕ does not have the property ($\not\models\exists$). That is, for all i' with $\tau_{i'} = \kappa_i$ we have $(\bar{\mathcal{D}}, \bar{\tau}, v, i') \models \phi$. Consider the largest such i'. Then there is a $j' \leq i'$ with $\tau_{i'} - \tau_{j'} \in I$ such that $(\bar{\mathcal{D}}, \bar{\tau}, v, j') \models \chi$ and for all $k' \in [j'+1, i'+1)$ we have $(\bar{\mathcal{D}}, \bar{\tau}, v, k') \models \psi$. By the definition of $(\bar{\mathcal{C}}, \bar{\kappa})$, there is a j with $\kappa_j = \tau_{j'}$. As χ has the property ($\not\models\forall$), we have that $(\bar{\mathcal{C}}, \bar{\kappa}, v, j) \models \chi$. Take $k \in [j+1, i+1)$ arbitrarily. If $(\bar{\mathcal{C}}, \bar{\kappa}, v, k) \not\models \psi$, as ψ has the property ($\not\models\exists$), then there is a k' with $\tau_{k'} = \kappa_k$ such that $(\bar{\mathcal{D}}, \bar{\tau}, v, k') \not\models \psi$. This contradicts our assumption that $(\bar{\mathcal{D}}, \bar{\tau}, v, i') \models \phi$, since k' must be in the interval $[j'+1, i'+1)$. We thus have that $(\bar{\mathcal{C}}, \bar{\kappa}, v, k) \models \psi$ for all $k \in [j+1, i+1)$. Hence $(\bar{\mathcal{C}}, \bar{\kappa}, v, i) \models \phi$, which is a contradiction.

- $\phi = \psi \, \mathsf{U}_I \, \chi$. This case is analogous to the previous one.

- $\phi = (\psi \, \mathsf{S}_I \, \chi) \wedge (\Box_J \, \psi)$ with $0 \notin I$ and $0 \in J$. ϕ and χ are labeled with ($\not\models\forall$), and ψ is labeled by ($\not\models\exists$). By the induction hypothesis, ψ and χ have the properties ($\not\models\exists$) and ($\not\models\forall$), respectively. Suppose that $(\bar{\mathcal{C}}, \bar{\kappa}, v, i) \not\models \phi$. Furthermore, to achieve a contradiction, suppose that ϕ does not have the property ($\not\models\forall$). That is, there is an i' with $\tau_{i'} = \kappa_i$ such that $(\bar{\mathcal{D}}, \bar{\tau}, v, i') \models \phi$. Then there is a $j' \leq i'$ with $\tau_{i'} - \tau_{j'} \in I$ such that $(\bar{\mathcal{D}}, \bar{\tau}, v, j') \models \chi$ and for all $k' \in [j'+1, i'+1)$ we have $(\bar{\mathcal{D}}, \bar{\tau}, v, k') \models \psi$; and for all $j'' \geq i'$ with $\tau_{j''} - \tau_{i'} \in J$ we have $(\bar{\mathcal{D}}, \bar{\tau}, v, j'') \models \psi$.

  By the definition of $(\bar{\mathcal{C}}, \bar{\kappa})$, there is a j with $\kappa_j = \tau_{j'}$. As χ has the property ($\not\models\forall$), we have that $(\bar{\mathcal{C}}, \bar{\kappa}, v, j) \models \chi$. Take $k \in [j+1, i)$ arbitrarily. If $(\bar{\mathcal{C}}, \bar{\kappa}, v, k) \not\models \psi$, as ψ has the property ($\not\models\exists$), then there is a k' with $\tau_{k'} = \kappa_k$ such that $(\bar{\mathcal{D}}, \bar{\tau}, v, k') \not\models \psi$. This contradicts our assumption that $(\bar{\mathcal{D}}, \bar{\tau}, v, i') \models \phi$. Indeed, k' must be in the interval $[j'+1, i''+1)$, where i'' is the largest time point such that $\tau_{i''} = \kappa_i$. If $k' \leq i'$ then $(\bar{\mathcal{D}}, \bar{\tau}, v, i') \not\models \psi \, \mathsf{S}_I \, \chi$. If $k' > i'$ then $(\bar{\mathcal{D}}, \bar{\tau}, v, i') \not\models \Box_J \, \psi$, as $0 \in J$. We thus have that $(\bar{\mathcal{C}}, \bar{\kappa}, v, k) \models \psi$ for all $k \in [j+1, i+1)$. Hence $(\bar{\mathcal{C}}, \bar{\kappa}, v, i) \models \psi \, \mathsf{S}_I \, \chi$.

  As $(\bar{\mathcal{D}}, \bar{\tau}, v, i') \models \Box_J \, \psi$ and $0 \in J$, it follows that for all $k' \geq i'$ with $\tau_{k'} = \tau_{i'}$ we have $(\bar{\mathcal{D}}, \bar{\tau}, v, k') \models \psi$. We have seen that $(\bar{\mathcal{D}}, \bar{\tau}, v, k') \models \psi$ for all $k' \in [j'+1, i'+1)$. Because $\tau_{j'} < \tau_{i'}$ (as $0 \notin I$), it also follows that for all $k' \leq i'$ with $\tau_{k'} = \tau_{i'}$ we have $(\bar{\mathcal{D}}, \bar{\tau}, v, k') \models \psi$. Hence $(\bar{\mathcal{D}}, \bar{\tau}, v, k') \models \psi$ for all k' with $\tau_{k'} = \tau_{i'}$. As ψ has the property ($\not\models\exists$), we obtain that $(\bar{\mathcal{C}}, \bar{\kappa}, v, i) \models \psi$. Similarly, we obtain that $(\bar{\mathcal{C}}, \bar{\kappa}, v, k) \models \psi$ for all $k > i$ such that $\kappa_k - \kappa_i \in J$. Hence $(\bar{\mathcal{C}}, \bar{\kappa}, v, i) \models \Box_J \, \psi$.

  We showed that $(\bar{\mathcal{C}}, \bar{\kappa}, v, i) \models \phi$, which is a contradiction. Thus ϕ has the property ($\not\models\forall$).

- $\phi = (\psi \, \mathsf{U}_I \, \chi) \wedge (\blacksquare_J \, \psi)$ with $0 \notin I$ and $0 \in J$. This case is analogous to the previous one.

- $\phi = \blacklozenge_I \psi$. There are two rules to analyze. For both rules, ψ is labeled with $(\models\exists)$. Suppose that $(\bar{\mathfrak{C}}, \bar{\kappa}, v, i) \models \phi$. Then there is a $j \le i$ with $\kappa_i - \kappa_j \in I$ such that $(\bar{\mathfrak{C}}, \bar{\kappa}, v, j) \models \psi$. As, by the induction hypothesis, ψ has the property $(\models\exists)$, there is a j' with $\tau_{j'} = \kappa_j$ such that $(\bar{\mathfrak{D}}, \bar{\tau}, v, j') \models \psi$.
  - ϕ is labeled with $(\models\exists)$. Take i' to be the largest k such that $\tau_k = \kappa_i$. Clearly, $\tau_{i'} - \tau_{j'} \in I$ and $j' \le i'$. Hence $(\bar{\mathfrak{D}}, \bar{\tau}, v, i') \models \blacklozenge_I \psi$ and ϕ has the property $(\models\exists)$.
  - $0 \notin I$ and ϕ is labeled with $(\models\forall)$. Take i' arbitrarily such that $\tau_{i'} = \kappa_i$. Clearly, $\tau_{i'} - \tau_{j'} \in I$ and, as $0 \notin I$, $\tau_{i'} - \tau_{j'} > 0$, thus $j' < i'$. Hence $(\bar{\mathfrak{D}}, \bar{\tau}, v, i') \models \blacklozenge_I \psi$. Thus ϕ has the property $(\models\forall)$.
- $\phi = \Diamond_I \psi$. This case is analogous to the previous one.
- $\phi = \blacklozenge_I \Diamond_J \psi$ with $0 \in I \cap J$. There is only one rule to consider: ψ is labeled with $(\models\exists)$ and ϕ is labeled by $(\models\forall)$. Suppose that $(\bar{\mathfrak{C}}, \bar{\kappa}, v, i) \models \phi$. Then there is a $j \le i$ with $\kappa_i - \kappa_j \in I$ and there is a $k \ge j$ with $\kappa_k - \kappa_j \in J$ such that $(\bar{\mathfrak{C}}, \bar{\kappa}, v, k) \models \psi$. As, by the induction hypothesis, ψ has the property $(\models\exists)$, there is a k' with $\tau_{k'} = \kappa_k$ such that $(\bar{\mathfrak{D}}, \bar{\tau}, v, k') \models \psi$. Take i' arbitrarily such that $\tau_{i'} = \kappa_i$. If $k' \ge i'$ then $0 \le \tau_{k'} - \tau_{i'} = \kappa_k - \kappa_i \le \kappa_k - \kappa_j \in J$. As $0 \in J$, we have $\tau_{k'} - \tau_{i'} \in J$. Thus $(\bar{\mathfrak{D}}, \bar{\tau}, v, i') \models \Diamond_J \psi$ and, as $0 \in I$, $(\bar{\mathfrak{D}}, \bar{\tau}, v, i') \models \blacklozenge_I \Diamond_J \psi$. The case when $k' < i'$ is similar. Hence ϕ has the property $(\models\forall)$.

We continue to show the soundness of the rules for the Boolean operator $\wedge$, the quantifier $\forall$, and the temporal operators trigger T_I and release R_I, shown in Figure 3.4. The soundness of these rules follows from the soundness of the rules in Figure 3.3 and the mentioned equivalences. For instance, the correctness of the rule

$$\frac{\psi : (\models\exists) \quad \chi : (\models\forall)}{(\psi \,\mathsf{T}_I\, \chi) \vee (\Diamond_J \psi) : (\models\forall)}\; 0 \notin I, 0 \in J$$

follows from unfolding the abbreviation $(\psi \,\mathsf{T}_I\, \chi) \vee (\Diamond_J \psi)$, which is $\neg((\neg\psi \,\mathsf{S}_I\, \neg\chi) \wedge (\Box_J \neg\psi))$, and the following derivation:

$$\cfrac{\cfrac{\cfrac{\psi : (\models\exists)}{\neg\psi : (\not\models\exists)} \quad \cfrac{\chi : (\models\forall)}{\neg\chi : (\not\models\forall)}}{(\neg\psi \,\mathsf{S}_I\, \neg\chi) \wedge (\Box_J \neg\psi) : (\not\models\forall)}\; 0 \notin I, 0 \in J}{\neg((\neg\psi \,\mathsf{S}_I\, \neg\chi) \wedge (\Box_J \neg\psi)) : (\models\forall)}$$

□

Based on the labels at a formula's root, we can determine if the formula has the properties (C1) and (C2) and hence whether it is collapse-sufficient, as shown in Theorem 3.4.6. In order to prove Theorem 3.4.6, we first establish a relation between the properties corresponding to the labels and the properties (I1) and (I2) in Lemma 3.3.7.

Lemma 3.4.5. *Let ϕ be a formula.*

1. *If ϕ has the property $(\models\forall)$, then ϕ has property (C1).*
2. *If ϕ has the property $(\not\models\forall)$, then ϕ has property (C2).*

3. *If ϕ has the property $(\models\exists)$, then $\Diamond\,\phi$ has property (C1).*

4. *If ϕ has the property $(\not\models\exists)$, then $\Box\,\phi$ has property (C2).*

Proof. We fix a temporal structure $(\bar{\mathfrak{C}}, \bar{\kappa})$.

1. Suppose ϕ has the property $(\models\forall)$ and that $(\bar{\mathfrak{C}}, \bar{\kappa}, v, 0) \models \phi$ for some valuation v. Then, for any $(\bar{\mathfrak{D}}, \bar{\tau}) \in col^{-1}(\bar{\mathfrak{C}}, \bar{\kappa})$ and every $j \in \mathbb{N}$ with $\kappa_0 = \tau_j$, it holds that $(\bar{\mathfrak{D}}, \bar{\tau}, v, j) \models \phi$. By the definition of collapsed temporal structure, we have $\kappa_0 = \tau_0$. Hence ϕ has (C1).

2. This case is analogous to the previous one.

3. Suppose ϕ has the property $(\models\exists)$ and that $(\bar{\mathfrak{C}}, \bar{\kappa}, v, 0) \models \Diamond\,\phi$ for some arbitrary valuation v. Then $(\bar{\mathfrak{C}}, \bar{\kappa}, v, i) \models \phi$ for some $i \in \mathbb{N}$. Because ϕ has the property $(\models\exists)$, for every $(\bar{\mathfrak{D}}, \bar{\tau}) \in col^{-1}(\bar{\mathfrak{C}}, \bar{\kappa})$, there is some $j \in \mathbb{N}$ with $\kappa_i = \tau_j$ such that $(\bar{\mathfrak{D}}, \bar{\tau}, v, j) \models \phi$. It follows that $(\bar{\mathfrak{D}}, \bar{\tau}, v, 0) \models \Diamond\,\phi$. Hence $\Diamond\,\phi$ has (C1).

4. This case is analogous to the previous one.

□

Based on the correctness of the derivation rules (Lemma 3.4.4) and Lemma 3.4.4, we obtain the following theorem.

Theorem 3.4.6. *If the formula ϕ can be labeled by $(\models\forall)$ and $(\not\models\forall)$, then it is collapse-sufficient. If ϕ can be labeled by $(\models\forall)$ and $(\not\models\exists)$, then $\Box\,\phi$ is collapse-sufficient. If ϕ can be labeled by $(\models\exists)$ and $(\not\models\forall)$, then $\Diamond\,\phi$ is collapse-sufficient. Moreover, we can determine in linear time in the formula's length whether ϕ can be labeled by $(\models\forall)$, $(\models\exists)$, $(\not\models\forall)$, or $(\not\models\exists)$.*

Proof. The first implications follow directly from Lemma 3.4.4 and Lemma 3.4.5. For the second implication, note that, based on our labeling rules, the label $(\models\forall)$ propagates through the operator $\Box$. For the third implication, note that, based on our labeling rules, the label $(\not\models\forall)$ propagates through the operator $\Diamond$. The proof for the complexity of the labeling procedure is analogous to the proof for Theorem 3.3.8. The only difference is in using four bits for the four different labels instead of using two bits for two labels. □

Note that formulas of the form $\Box\,\psi$ are already collapse-sufficient if ψ can be labeled by $(\not\models\exists)$ and $\Box\,\psi$ can be labeled by $(\models\forall)$. Even if only one of these labellings can be derived, monitoring $\Box\,\psi$ on the collapsed temporal structure of an interleaving is still useful. For example, if ψ is labeled by $(\not\models\exists)$ then violations that are found on the collapsed temporal structure relate to strong violations on the set of interleavings. However, we might miss some violations.

Example 3.4.7. *We illustrate our algorithm by applying it to the formulas from Example 3.3.2. Unlike in Example 3.3.2, we use the labels $(\models\forall)$, $(\models\exists)$, $(\not\models\forall)$, and $(\not\models\exists)$ and the rules shown in Figure 3.3. First, we consider the formula $\Box\,\forall x.\, \neg publish(x) \vee \blacklozenge_{[0,11)}\, approve(x)$. Both of its atomic subformulas $publish(x)$ and $approve(x)$ are labeled with $(\models\exists)$ and $(\not\models\forall)$. We label the subformula $\blacklozenge_{[0,11)}\, approve(x)$ with $(\models\exists)$ and $(\not\models\forall)$. We cannot label it with $(\models\forall)$ since the interval contains 0. The subformula $\neg publish(x)$ is labeled with $(\not\models\exists)$ and $(\models\forall)$. The subformulas $\neg publish(x) \vee \blacklozenge_{[0,11)}\, approve(x)$ and $\forall x.\, \neg publish(x) \vee \blacklozenge_{[0,11)}\, approve(x)$ are* *labeled* $(\models \exists)$ *and* $(\not\models \exists)$. *We conclude that the formula* $\Box\, \forall x.\, \neg \mathit{publish}(x) \lor \blacklozenge_{[0,11)}\, \mathit{approve}(x)$ *has the property (C2). It does not have the property (C1), as explained in Example 3.3.2.*

The formula $\Box\, \forall x.\, \mathit{publish}(x) \to \blacklozenge_{[1,11)}\, \mathit{approve}(x)$ *has both properties (C1) and (C2). The labeling starts similarly but* $\blacklozenge_{[1,11)}\, \mathit{approve}(x)$ *is additionally labeled with* $(\models \forall)$ *since the interval of the temporal operator does not contain* 0. *This label propagates to the formula's root. We conclude that* $\Box\, \forall x.\, \neg \mathit{publish}(x) \lor \blacklozenge_{[1,11)}\, \mathit{approve}(x)$ *also has property (C1).*

As in the interleaving-sufficient case, the defined fragment is incomplete, which is again witnessed by the formula $\Box\, \forall x.\, \mathit{publish}(x) \to (\Diamond_{[0,1)}\, \mathit{approve}(x)) \lor \blacklozenge_{[0,11)}\, \mathit{approve}(x)$. It is collapse-sufficient, but cannot be labeled as required by Theorem 3.4.6. Note that the semantically equivalent formula $\Box\, \forall x.\, \mathit{publish}(x) \to \Diamond_{[0,1)} \blacklozenge_{[0,11)}\, \mathit{approve}(x)$ is recognized as collapse-sufficient using the rules shown in Figure 3.3.

3.5 Sufficient Fragments

In this section, we compare the interleaving-sufficient and collapse-sufficient fragments and present a generic recipe that approximates policies to obtain formulas in these fragments.

3.5.1 Comparison

The interleaving-sufficient fragment is larger than the collapse-sufficient fragment. In contrast, the collapse-sufficient fragment is more efficient to monitor. We explain these two aspects in more detail.

Intuitively, a collapse-sufficient formula is satisfied either on all pre-images of a collapse or on none. An interleaving-sufficient formula is satisfied either on all interleavings or on none. As the set of all interleavings is a strict subset of all collapse pre-images, the interleaving-sufficient property is a weaker requirement than the collapse-sufficient property. Theorem 3.5.1 shows that a collapse-sufficient formula is indeed always also interleaving-sufficient.

Theorem 3.5.1. *If a formula ϕ is collapse-sufficient then ϕ is also interleaving-sufficient.*

Proof. Suppose that ϕ is a collapse-sufficient MFOTL formula. Moreover, let $(\bar{\mathcal{D}}_1, \bar{\tau}_1)$, $(\bar{\mathcal{D}}_2, \bar{\tau}_2)$, $(\bar{\mathcal{D}}, \bar{\tau})$, and $(\bar{\mathcal{C}}, \bar{\kappa})$ be temporal structures where $(\bar{\mathcal{D}}, \bar{\tau}) \in (\bar{\mathcal{D}}_1, \bar{\tau}_1) \between (\bar{\mathcal{D}}_2, \bar{\tau}_2)$ and $(\bar{\mathcal{C}}, \bar{\kappa}) = \mathit{col}(\bar{\mathcal{D}}, \bar{\tau})$. We fix an arbitrary valuation v. There are two cases to consider: either ϕ is satisfied on $(\bar{\mathcal{D}}, \bar{\tau})$ or it is not.

First, if $(\bar{\mathcal{D}}, \bar{\tau}, v, 0) \models \phi$, then also $(\bar{\mathcal{C}}, \bar{\kappa}, v, 0) \models \phi$. To see this, suppose that $(\bar{\mathcal{C}}, \bar{\kappa}, v, 0) \not\models \phi$. As ϕ has the property (C2), it would follow that $(\bar{\mathcal{D}}, \bar{\tau}, v, 0) \not\models \phi$, which is a contradiction. From $(\bar{\mathcal{C}}, \bar{\kappa}, v, 0) \models \phi$ and ϕ having the property (C1), it follows that $(\bar{\mathcal{D}}', \bar{\tau}', v, 0) \models \phi$ for all $(\bar{\mathcal{D}}', \bar{\tau}') \in (\bar{\mathcal{D}}_1, \bar{\tau}_1) \between (\bar{\mathcal{D}}_2, \bar{\tau}_2)$. Hence, ϕ has the property (I1).

Second, if $(\bar{\mathcal{D}}, \bar{\tau}, v, 0) \not\models \phi$, then since ϕ has the property (C1), it follows that $(\bar{\mathcal{C}}, \bar{\kappa}, v, 0) \not\models \phi$. Since ϕ has the property (C2), it follows that $(\bar{\mathcal{D}}', \bar{\tau}', v, 0) \not\models \phi$, for all $(\bar{\mathcal{D}}', \bar{\tau}') \in (\bar{\mathcal{D}}_1, \bar{\tau}_1) \between (\bar{\mathcal{D}}_2, \bar{\tau}_2)$. Hence, ϕ has the property (I2).

Since ϕ has the properties (I1) and (I2), it is interleaving-sufficient. □

The converse does not hold. There are interleaving-sufficient formulas that are not collapse-sufficient. Intuitively, the interleaving-sufficient formulas allow us to check individual time points from the original traces, but collapse-sufficient formulas are restricted to checking the collapsed time points. For example, the policy requiring that if p happens, then q must happen at the same time point, formalized as $\Box\, p \rightarrow q$, is interleaving-sufficient, but not collapse-sufficient. Inspecting only collapsed time points still allows us to check that an event never occurs. That is, $\Box\, \neg p$ is collapse-sufficient. However, $\Box\, p$ is interleaving-sufficient but not collapse-sufficient.

A practical advantage of the collapse-sufficient fragment is that monitoring a collapsed temporal structure is more efficient than monitoring an interleaving, as our case study described in Chapter 4 demonstrates. The main reason is that time points with equal timestamps are merged to a single time point in a collapsed temporal structure. Hence, the monitor processes the logged actions with equal timestamps in a single invocation.

The structure of the collapsed log can be further exploited to increase monitor performance by rewriting the monitored formulas. In particular, if we are monitoring the collapsed log, rather than an interleaving, then we can rewrite formulas of the form $\Diamond_{[0,1)}\, \phi$, $\blacklozenge_{[0,1)}\, \phi$, $\Box_{[0,1)}\, \phi$, and $\blacksquare_{[0,1)}\, \phi$ to ϕ. The reason is that in a collapsed trace, there is at most one time point for each timestamp. We call the rewritten formulas *collapse-optimized*.

3.5.2 Policy Approximation

In Example 3.3.2, we have seen that we can obtain an interleaving-sufficient policy by strengthening or weakening the original policy. We now generalize this observation.

Let ϕ be a formula in positive normal form. That is, negations in ϕ are pushed inside and occur only in front of atomic formulas. We obtain a *weakened* formula ϕ^w by replacing each atomic subformula $r(t_1, \ldots, t_{\iota(r)})$ that occurs positively in ϕ by $\blacklozenge_I \Diamond_{I'}\, r(t_1, \ldots, t_{\iota(r)})$, for some intervals I and I' with $0 \in I \cap I'$. Analogously, in a *strengthened* formula ϕ^s, we replace each negative occurrence of an atomic subformula $r(t_1, \ldots, t_{\iota(r)})$ by $\blacklozenge_I \Diamond_{I'}\, r(t_1, \ldots, t_{\iota(r)})$ for some intervals I, I'.

Theorem 3.5.2. *Let ϕ^w and ϕ^s be weakened and strengthened formulas of the formula ϕ in positive normal form. The formulas $\phi \rightarrow \phi^w$ and $\phi^s \rightarrow \phi$ are valid. Moreover,*

1. *if ϕ^s is collapse-sufficient then ϕ has property (C1), and*

2. *if ϕ^w is collapse-sufficient then ϕ has property (C2).*

Proof. We first show that ϕ^w is weaker than ϕ, or more precisely, that the formula $\phi \rightarrow \phi^w$ is valid. We proceed by structural induction on ϕ.

- $\phi = t \approx t'$, $\phi = t \prec t'$, $\phi = \neg(t \approx t')$, $\phi = \neg(t \prec t')$, or $\neg r(t_1, \ldots, t_{\iota(r)})$, where t, t', and t_i with $1 \leq i \leq \iota(r)$ are variables or constants. Then $\phi^w = \phi$, and the statement clearly holds.

- $\phi = r(t_1, \ldots, t_{\iota(r)})$. Then $\phi^w = \blacklozenge_J \Diamond_{J'}\, r(t_1, \ldots, t_{\iota(r)})$, for some intervals J and J' with $0 \in J \cap J'$. Let $(\bar{\mathcal{D}}, \bar{\tau})$ be a temporal structure, v a valuation, and i a time point. Suppose that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \phi$. As $0 \in I \cap J$, we clearly have $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \blacklozenge_J \Diamond_{J'}\, \phi$, that is, $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \phi'$.

- $\phi = \psi \wedge \chi$, $\phi = \exists x. \psi$, $\phi = \bullet_I\, \psi$, $\phi = \bigcirc_I\, \psi$, $\phi = \psi\, \mathsf{S}_I\, \chi$, or $\phi = \psi\, \mathsf{U}_I\, \chi$. These cases follow directly from the induction hypotheses. We only present the case $\phi = \psi\, \mathsf{S}_I\, \chi$. We have $\phi^w = \psi^w\, \mathsf{S}_I\, \chi^w$. Let $(\bar{\mathcal{D}}, \bar{\tau})$ be a temporal structure, v a valuation, and i a time point. Suppose that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \phi$. Then there is a $j \leq i$ with $\tau_i - \tau_j \in I$ such that $(\bar{\mathcal{D}}, \bar{\tau}, v, j) \models \chi$ and $(\bar{\mathcal{D}}, \bar{\tau}, v, k) \models \psi$ for any $k \in [i+1, j+1)$. Using the induction hypotheses for ψ and χ, we obtain that $(\bar{\mathcal{D}}, \bar{\tau}, v, j) \models \chi^w$ and $(\bar{\mathcal{D}}, \bar{\tau}, v, k) \models \psi^w$ for any $k \in [i+1, j+1)$. Hence $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \phi^w$.

The proof of the dual case, that is, that the formula $\phi^s \to \phi$ is valid, is similar. It is based on the remark that the formula $(\neg \blacklozenge_J \lozenge_{J'} r(t_1, \dots, t_{\iota(r)})) \to \neg r(t_1, \dots, t_{\iota(r)})$ is valid.

Finally, we prove Statement (1). Statement (2) is similar. Let $(\bar{\mathcal{C}}, \bar{\kappa})$ be the collapse of two temporal structures $(\bar{\mathcal{D}}^1, \bar{\tau}^1)$ and $(\bar{\mathcal{D}}^2, \bar{\tau}^2)$. Suppose that ϕ^s is collapse-sufficient and that $(\bar{\mathcal{C}}, \bar{\kappa}, v, 0) \models \phi^s$, for some arbitrary valuation v. It follows that $(\bar{\mathcal{D}}, \bar{\tau}, v, 0) \models \phi^s$ for any $(\bar{\mathcal{D}}, \bar{\tau}) \in (\bar{\mathcal{D}}^1, \bar{\tau}^1) \between (\bar{\mathcal{D}}^2, \bar{\tau}^2)$. As $\phi^s \to \phi$ is valid, we have that $(\bar{\mathcal{D}}, \bar{\tau}, v, 0) \models \phi$, for any $(\bar{\mathcal{D}}, \bar{\tau}) \in (\bar{\mathcal{D}}^1, \bar{\tau}^1) \between (\bar{\mathcal{D}}^2, \bar{\tau}^2)$. □

Weakened and strengthened formulas are more likely to be collapse-sufficient, since their subformulas of the form $\blacklozenge_I \lozenge_{I'} r(t_1, \dots, t_{\iota(r)})$ can be labeled with $(\models \forall)$, while $r(t_1, \dots, t_{\iota(r)})$ can only be labeled with the weaker label $(\models \exists)$. Simultaneously weakening and strengthening always results in a collapse-sufficient formula. However, the resulting formula does not necessarily relate to the original formula.

Since a collapse-sufficient formula is also interleaving-sufficient (Theorem 3.5.1), the above rewriting can also be used when an interleaving-sufficient formula is desired.

Note that by inserting the temporal operators $\blacklozenge_{[0,1)}$ and $\lozenge_{[0,1)}$ around positively occurring atomic subformulas, the ordering of equally timestamped actions becomes irrelevant. This is desirable in systems where the clocks used to timestamp the actions are synchronized but too coarse-grained to capture the relative ordering of events occurring almost concurrently. Taking this idea further, by putting temporal operators $\blacklozenge_{[0,b)}$ and $\lozenge_{[0,b)}$ around these subformulas with $b \geq 1$, we take into account that the timestamps in a temporal structure are inaccurate and might differ from their actual value by the threshold b—a situation that occurs in practice.

4 The Nokia Case Study

In this chapter, we describe the deployment of our approach to monitoring concurrently logged actions within Nokia's Data-collection Campaign [AN10, KBD$^+$10, LGPA$^+$12], which is a real-world application with realistic data-usage policies. Furthermore, we report on the monitor's performance and our findings.

4.1 Nokia's Data-collection Campaign

Scenario. The campaign collects contextual information from cell phones of about 180 participants. This sensitive data includes phone locations, call and SMS information, and the like. The data collected by a participant's phone is propagated into the databases db1, db2, and db3. The phones use WLAN to periodically upload their data to database db1. Every night, the synchronization script script1 copies the data from db1 to db2. Furthermore, triggers running on db2 anonymize and copy the data to db3, where researchers can access and analyze the anonymized data. The participants can access and delete their own data using a web interface to db1. Deletions are propagated to all databases: from db1 to db2 by the synchronization script script2, which also runs every night, and from db2 to db3 by database triggers. Figure 4.1 summarizes the various data usages.

Within the campaign, data is organized by records and can easily be identified. When uploading data from a phone into db1, a unique identifier is generated for each record. This identifier, together with an identifier of the participant who contributed the data, is attached to the record.

Policies. The collected data is subject to various policies that protect the participants' privacy. For example, there are access-control policies and policies governing the process of propagating the data between databases. In particular, insertions and deletions of data must be propagated within a given time limit. Furthermore, the latest version of the synchronization scripts must be used and their running times are restricted. Finally, access to the databases is restricted to selected user accounts and the account used by the script script1 may be used only while the script is running.

We now describe these policies in more detail and present their formalization in MFOTL. We start with the predicate symbols used for formalizing the policies. We represent system actions as elements in relations interpreting the predicate symbols at the time points. The elements of the relations for the predicate symbols *select*, *insert*, *delete*, and *update* correspond to database operations with equally-named SQL commands. The parameters are the user executing the operation, the name of the database, and an identifier of the involved data. The predicate symbols *start* and *stop* indicate the starting and finishing of a synchronization script and include the script's name. After the script script1 starts, it logs additional details about its SVN status using the predicate symbol *svn*. The parameters are the script's name, its

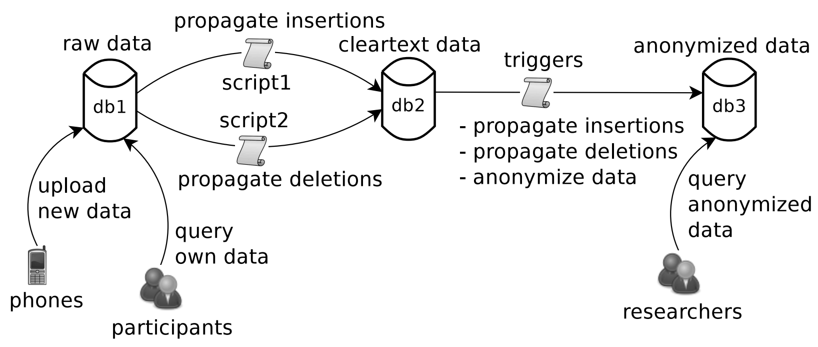

Figure 4.1: Nokia's Data-collection Campaign

SVN status determined by the command `svn status -u -v`, the SVN URL, and the SVN revision number. When the script is the latest version, we use the value latest for the SVN status. The predicate symbol *commit* represents committing a new script version into the subversion repository. The parameters are the SVN URL and revision number.

The MFOTL formalization of the policies uses the predicate symbols just described. The formulas are shown in Table 4.1. In the following, we informally state the detailed policies in natural language and for the more involved policies, we provide additional explanations:

- *delete:* Only user script2, representing the synchronization script script2, may delete data in db2 by executing the SQL delete command.

- *insert:* Only user script1, representing the synchronization script script1, may insert data in db2 by executing the SQL insert command.

- *select:* Only a limited set of users, namely script1, script2, and triggers, may read data from db2 by executing the SQL select command.

- *update:* No SQL update commands are allowed in db2.

- *script1:* Database operations may be executed under the user account script1 only while the script script1 is running. The motivation for this policy is that the account script1 should only be used by the script, so if the account is used while the script is not running, the account may have been compromised. The database operation can happen while the script is running, including when the script starts or finishes. That is, the time points when an operation happens and when the script starts or ends may have equal timestamps. The semantics of the S operator includes the script start, but excludes the script end. Therefore, the script end is allowed with the additional disjunct at the end of the formula.

- *runtime:* The synchronization scripts must run for at least 1 second and for no longer than 6 hours.

- *svn, svn2:* The synchronization scripts are maintained in an SVN repository. We require that when started, the synchronization scripts are the latest version available in the repository (largest SVN revision number). We use two different formalizations, *svn* and *svn2*. The policy *svn* uses the status parameter of the predicate symbol *svn*. The policy *svn2* compares

Table 4.1: Policy Formalizations in MFOTL

policy	MFOTL formalization
delete	$\Box\, \forall user.\, \forall data.\, delete(user, \mathsf{db2}, data) \rightarrow user \approx \mathsf{script2}$
insert	$\Box\, \forall user.\, \forall data.\, insert(user, \mathsf{db2}, data) \rightarrow user \approx \mathsf{script1}$
select	$\Box\, \forall user.\, \forall data.\, select(user, \mathsf{db2}, data) \rightarrow$ $user \approx \mathsf{script1} \vee user \approx \mathsf{script2} \vee user \approx \mathsf{triggers}$
update	$\Box\, \forall user.\, \forall data.\, \neg update(user, \mathsf{db2}, data)$
script1	$\Box\, \forall db.\, \forall data.\, select(\mathsf{script1}, db, data) \vee insert(\mathsf{script1}, db, data) \vee$ $delete(\mathsf{script1}, db, data) \vee update(\mathsf{script1}, db, data) \rightarrow$ $((\neg \blacklozenge_{[0,1s)} \Diamond_{[0,1s)}\, end(\mathsf{script1}))\ \mathsf{S}\ (\blacklozenge_{[0,1s)} \Diamond_{[0,1s)}\, start(\mathsf{script1}))) \vee$ $\blacklozenge_{[0,1s)} \Diamond_{[0,1s)}\, end(\mathsf{script1})$
runtime	$\Box\, \forall script.\, start(script) \rightarrow$ $(\neg \blacklozenge_{[0,1s)} \Diamond_{[0,1s)}\, end(script)) \wedge \Diamond_{[1s,6h)}\, end(script)$
svn	$\Box\, \forall script.\, start(script) \rightarrow$ $\blacklozenge_{[0,1s)} \Diamond_{[0,10s)}\, \exists url.\, \exists rev.\, svn(script, \mathsf{latest}, url, rev)$
svn2	$\Box\, \forall script.\, \forall status.\, \forall url.\, \forall rev.\, svn(script, status, url, rev) \rightarrow$ $\blacksquare_{[1s,\infty)}\, (commit(url, rev') \rightarrow rev' \leq rev)$
ins-1-2	$\Box\, \forall user.\, \forall data.\, insert(user, \mathsf{db1}, data) \wedge data \not\approx \mathsf{unknown} \rightarrow$ $\blacklozenge_{[0,1s)} \Diamond_{[0,30h]}\, \exists user'.\, insert(user', \mathsf{db2}, data) \vee$ $delete(user', \mathsf{db1}, data)$
ins-2-3	$\Box\, \forall user.\, \forall data.\, insert(user, \mathsf{db2}, data) \wedge data \not\approx \mathsf{unknown} \rightarrow$ $\blacklozenge_{[0,1s)} \Diamond_{[0,60s)}\, \exists user'.\, insert(user', \mathsf{db3}, data)$
ins-3-2	$\Box\, \forall user.\, \forall data.\, insert(user, \mathsf{db3}, data) \wedge data \not\approx \mathsf{unknown} \rightarrow$ $\blacklozenge_{[0,60s)} \Diamond_{[0,1s)}\, \exists user'.\, insert(user', \mathsf{db2}, data)$
del-1-2	$\Box\, \forall user.\, \forall data.\, delete(user, \mathsf{db1}, data) \wedge data \not\approx \mathsf{unknown} \rightarrow$ $(\blacklozenge_{[0,1s)} \Diamond_{[0,30h)}\, \exists user'.\, delete(user', \mathsf{db2}, data)) \vee$ $((\Diamond_{[0,1s)} \blacklozenge_{[0,30h)}\, \exists user'.\, insert(user', \mathsf{db1}, data)) \wedge$ $(\blacksquare_{[0,30h)} \Box_{[0,30h)}\, \neg\exists user'.\, insert(user', \mathsf{db2}, data)))$
del-2-3	$\Box\, \forall user.\, \forall data.\, delete(user, \mathsf{db2}, data) \wedge data \not\approx \mathsf{unknown} \rightarrow$ $\blacklozenge_{[0,1s)} \Diamond_{[0,60s)}\, \exists user'.\, delete(user', \mathsf{db3}, data)$
del-3-2	$\Box\, \forall user.\, \forall data.\, delete(user, \mathsf{db3}, data) \wedge data \not\approx \mathsf{unknown} \rightarrow$ $\blacklozenge_{[0,60s)} \Diamond_{[0,1s)}\, \exists user'.\, delete(user', \mathsf{db2}, data)$

the revision number parameter of the predicate symbol *svn* with the committed revision numbers obtained from the subversion log via the predicate symbol *commit*. For the policy *svn* we let the logging mechanism compute the latest revision number, while for the policy *svn2* we compute it using the monitor. Monitoring both policies allows us to compare how efficiently the monitor copes with these different formalizations and to observe the impact of offloading the monitor by doing pre-computations in the logging mechanisms.

- *ins-1-2, ins-2-3, ins-3-2:* Data uploaded by the phone into db1 must be propagated to all databases. In particular, *ins-1-2* requires that data uploaded into db1 must be inserted into db2 within 30 hours after the upload, unless it has been deleted from db1 in the meantime. Furthermore, *ins-2-3* and *ins-3-2* require that data may be inserted into db2 iff it is inserted into db3 within 1 minute. The time limit from db1 to db2 is 30 hours because the synchronization scripts run once every 24 hours and can run for up to 6 hours. The time limit from db2 to db3 is only 60 seconds as this synchronization is implemented by database triggers that start immediately upon a change in db2. Note that these policies require propagating new data between db2 and db3 in both directions. However, between db1 and db2 only one direction is required. The reason is the incomplete logging for db1.
- *del-1-2, del-2-3, del-3-2:* Data deleted from db1 must be consistently deleted from all databases. The policies *del-2-3* and *del-3-2* are analogous to the policies *ins-2-3* and *ins-3-2*, respectively. The formalization of the policy *del-1-2* is more involved: If data is deleted from db1, then this data must also be deleted from db2 within 30 hours. However, if the data has just been uploaded to db1 and not yet propagated to db2, then it should not be propagated to db2 in the future either. Since the propagation would happen within at most 30 hours, we can simply consider the past and the future 30 hours to determine whether data has been or will be propagated to db2.

Note that all formulas in Table 4.1 are collapse-sufficient. However, some policies have slightly weaker or stronger variants that are not collapse-sufficient. For example, we obtained *ins-2-3* from the policy "all data inserted into db2 must also be inserted into db3 within 60 seconds" by weakening the formula $\Box\, \forall users.\, \forall data.\, insert(user, \mathsf{db2}, data) \land data \not\approx \mathsf{unknown} \rightarrow \Diamond_{[0,60s)}\, \exists user'.\, insert(user', \mathsf{db3}, data)$. Intuitively, *ins-2-3* is the policy formalization that does not distinguish the relative ordering of the insertions into db2 and db3 when they are logged with equal timestamps. This is because the 1 second timestamp granularity that is used may not be fine enough: the database triggers may be activated within milliseconds.

Logging Mechanisms. We extended the data-collection setup with mechanisms to log policy-relevant actions. We installed logging mechanisms for the three databases, the script script1, and the SVN repository, assuming synchronized clocks for timestamping. Adding logging mechanisms to the databases was not straightforward, so we discuss them in more detail.

As logs for the database db1 were not available, we implemented a proxy to inspect interactions of participants and phones with db1. The proxy logs what data is inserted and deleted. To observe the insertion of new data, we monitor the network traffic when the phone uploads data. For deletions, we use a custom front-end that logs the requests for deleting data. For practical reasons, we could deploy these mechanisms only for 2 out of the 180 participants.

Hence, we have only partial logging for db1. However, the partial logging affects only 2 out of the 14 policies.

The databases db2 and db3 reside physically on a single PostgreSQL server, which logs the SQL queries. We extract relevant actions from these PostgreSQL logs. The main challenge is to determine what data is processed in a query since only the query itself is logged. Fortunately, most relevant queries are made by automated scripts or database triggers and contain enough information to determine what data is used. For example, an insert or delete query initiated by a synchronization script includes the identifier of the used data record. Hence, a simple syntactic analysis of these queries suffices to log the relevant actions in sufficient detail. When the analysis failed to extract the data, we identified the data with the constant unknown.

4.2 Evaluation

Performance. We evaluated the performance of the monitor on logs from the data-collection campaign. We now describe the logs, the optimizations we made when monitoring the logs, and the monitor's performance.

We monitored all formulas shown in Table 4.1 on a log file covering approximately one year of the data-collection campaign. We obtained this log file by interleaving logs from the different log producers to produce one interleaved log that we subsequently collapsed. Note that all monitored formulas are collapse-sufficient, so the monitor correctly reports all violations after inspecting the collapsed log.

We now describe the collapsed log. It contains approximately 5 million time points and 218 million actions (the total number of tuples in all relations). The major part consists of insertions into the three databases: more than 107 million insertions into db2 and db3 each, and 360,000 insertions into db1. The smaller number of logged insertions for db1 is due to the incomplete logging. There are about 3 million select actions and 700,000 update actions on db2 and db3. All other types of actions occurred less than 1,000 times in the whole log.

For comparison, before collapsing, the interleaved log was larger. It contained approximately 400 million time points and a similar number of log actions. The collapsing reduced the number of time points because not all time points in the interleaving had a unique timestamp. The main reason why the total number of log actions was decreased by collapsing is that for most SQL select queries on database db3 we could not determine what data was used. These were logged as using unknown data and therefore could not be distinguished from each other. As there were multiple such indistinguishable actions logged per time unit (second), they were always preserved as only one logged action per timestamp.

For the evaluation, we used the MONPOLY tool [BHKZ12] running on a desktop computer with an Intel Core i5 2.67 GHz CPU and 8 GB of RAM. To monitor all policies we made two optimizations.

The first optimization was collapsing the interleaving. Note that all monitored formulas are interleaving-sufficient, so the monitor would correctly report all violations after inspecting an arbitrary interleaving. However, it turned out that monitoring an interleaving was computationally infeasible for four policies: *del-1-2*, *ins-1-2*, *ins-2-3*, and *ins-3-2*. Monitoring the policy *del-1-2* exceeded the memory available on our computer and monitoring the policies *ins-1-2*, *ins-2-3*, and *ins-3-2* took too long. For example, monitoring the policy *ins-2-3* only on the first two months of the one year log file took 17 days. Monitoring the collapsed interleaving

Table 4.2: Monitor Performance

	collapse, rewritten formulas		collapse		interleaving	
policy	running time	memory used	running time	memory used	running time	memory used
delete	17 min	14 MB	17 min	14 MB	31 min	12 MB
insert	21 min	14 MB	21 min	14 MB	32 min	12 MB
select	17 min	15 MB	17 min	15 MB	34 min	12 MB
update	17 min	14 MB	17 min	14 MB	33 min	12 MB
script1	21 min	14 MB	22 min	14 MB	57 min	13 MB
runtime	18 min	30 MB	19 min	30 MB	52 min	2866 MB
svn	17 min	14 MB	17 min	14 MB	38 min	22 MB
svn2	17 min	14 MB	17 min	14 MB	33 min	12 MB
ins-1-2	34 min	1014 MB	14 days	1387 MB	-	-
ins-2-3	49 min	20 MB	52 min	20 MB	-	-
ins-3-2	49 min	15 MB	49 min	15 MB	-	-
del-1-2	57 min	3313 MB	69 min	3248 MB	-	-
del-2-3	18 min	14 MB	17 min	14 MB	38 min	26 MB
del-3-2	17 min	14 MB	17 min	14 MB	37 min	12 MB

was computationally feasible for all policies. For policies, which we could monitor already on the interleaving, the monitor was up to three times faster. However, monitoring *ins-1-2* still took a long time, namely 2 weeks.

The second optimization was rewriting formulas into collapse-optimized formulas. The time needed to monitor the policy *ins-1-2* improved from 2 weeks to 34 minutes and for *del-1-2* it improved from 69 minutes to 57 minutes. For other policies, the difference was negligible.

Table 4.2 shows the monitor's running times and memory usage for each policy with the different optimizations. A missing value in the table signifies that we could not monitor the policy.

We now report in detail on the performance of the monitor after the two optimizations: monitoring performance-optimized formulas on the collapsed log. Monitoring invariants like the policy *delete* is fast: the monitor needed around 20 minutes. The more complex formulas with temporal operators were similarly fast when the formulas matched only a small number of events from the log file. For example, monitoring the policy *svn2* also took less than 20 minutes. Finally, formulas involving temporal operators with large time windows and matching a large part of the events in the log were the most expensive ones to monitor. This was the case for the policies *ins-1-2*, *ins-2-3*, and *ins-3-2* because the log consists mainly of insert events. The policies *ins-2-3* and *ins-3-2* took 49 minutes each. The policy *ins-1-2* took only 34 minutes because there are significantly fewer inserts into db1 than into db2 or db3. The most expensive policy for monitoring was *del-1-2*. It took 57 minutes because its formalization includes multiple temporal operators with large time windows and the subformulas of these temporal operators match the abundant insert events. *ins-1-2* has only one such operator.

We monitored the logs offline. That is, we first collected the complete logs and then monitored them. However, an online monitoring approach, where the logs are monitored as they are generated, seems possible because the running times are orders of magnitude smaller than the time period covered by the logs.

For comparison, we have also monitored the simple access control policies on the collapsed log with the Unix tool grep. The policies *update* and *delete* each took only 5 minutes, which is 3 times faster than the monitor. However, for *insert* grep needed 4 hours, which is 10 times slower than the monitor. We suspect that the automaton used by grep to represent a regular expression was inefficient in dealing with the many insert actions.

The monitor's memory requirements are also modest. For most policies, the monitor does not require more than 30 MB of RAM. The only exceptions are *ins-1-2* with 1 GB and *del-1-2* with 3.3 GB. Again, the reason is temporal operators with large time windows matching a large number of log events.

We now describe in more detail how the monitor coped with the most difficult policy, *del-1-2*. The dashed line in Figure 4.2 shows the monitor's accumulated running time. The solid line shows the number of log actions since the beginning of the log. The steepness of the solid line indicates the amount of data logged at the corresponding time on the x-axis. We see several flat parts in this curve. During these flat parts, no logs were produced due to server migration and upgrades of the logging infrastructure. We can also see a steep part at the end of August 2010. A new version of the synchronization scripts was copying additional data into the databases for several days. This resulted in an increased number of logged actions. The running time (dashed line) closely followed the number of logged actions (solid line), except for a noticeable slow-down of the monitor during the steep part. The dashed line in Figure 4.3 shows the amount of memory used by the monitor. Most of the time the monitor needed less than 0.5 GB of memory, but peaked at 3.3 GB at the point where the log curve was steep. Due to the increased log density, more log actions fell into the time windows of the temporal operators, causing the monitor to use larger auxiliary relations. Also note that both the amount of new actions logged and the memory used by the monitor decreased towards the log's end. The campaign ended in 2011 and the participants gradually stopped contributing data towards the campaign's end.

Findings. To our surprise, the monitor reported a number of policy violations. First, some access control policies like *delete* were violated. These violations were due to testing, debugging, and other improvement activities going on while the system was running. Second, the policy *runtime* was violated several times, such as when synchronizing the databases after the server migration. Third, an earlier version of one of the synchronization scripts contained a bug, which was not detected in previous tests. Only a subset of the insertions were propagated between the databases. Fourth, while the campaign was running, the infrastructure was migrated to another server. After the migration, the deployment of the scripts was delayed, which caused policy violations.

Overall, the main reason for these violations is that we monitored an experimental system still under development. It is worth pointing out that the privacy of the participants was guaranteed at all times during the campaign and no data elements were unintendedly lost. However, as this case study shows, the monitor can be a powerful debugging tool. For commercial systems, it can detect policy violations thereby protecting the users' privacy and increasing users' trust in the systems. Our findings also show that policy monitoring makes sense even in systems where users and system administrators are honest and interested in honoring the policies.

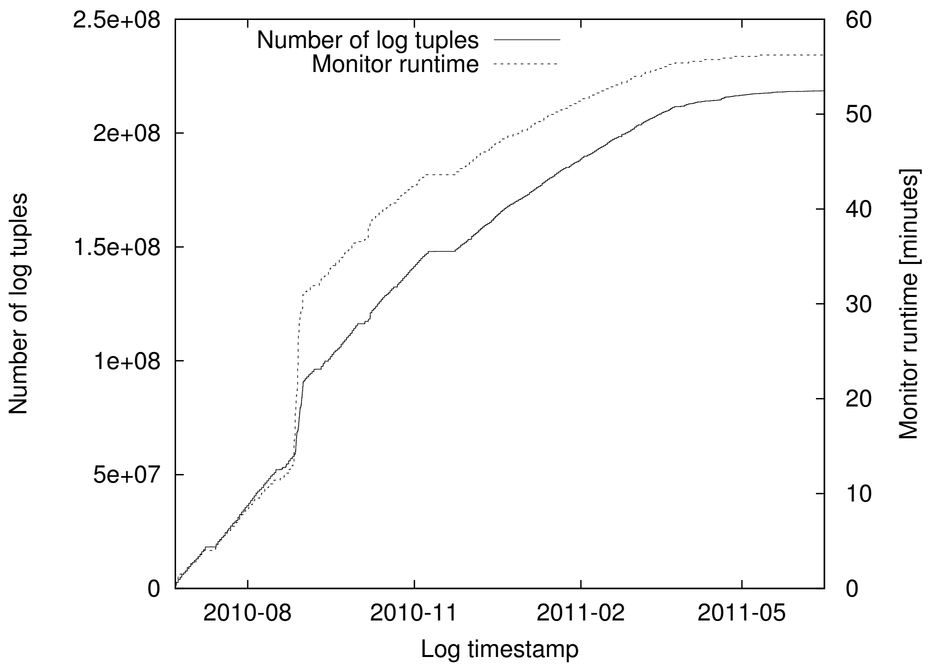

Figure 4.2: Monitor Running Time on Policy *del-1-2*

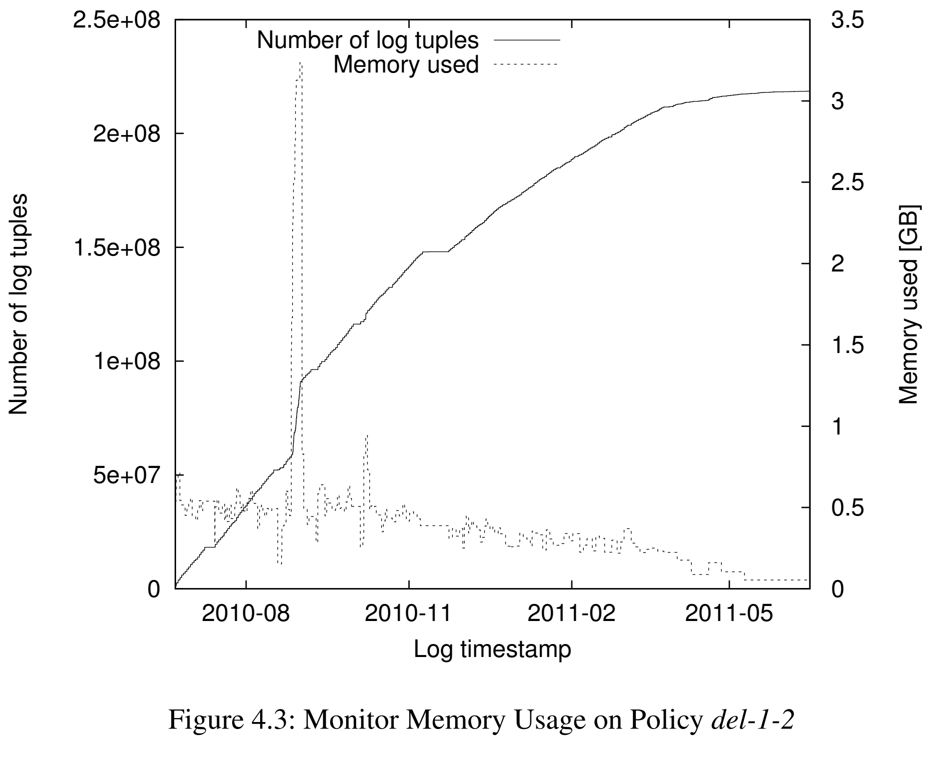

Figure 4.3: Monitor Memory Usage on Policy *del-1-2*

5 Monitoring Large Logs

In this chapter, we introduce the theory underpinning our approach of splitting logs into slices and monitoring the slices separately and in parallel. In particular, in Section 5.1, we lay the foundations for generating sound and complete slices. In the Sections 5.2 and 5.3, we present the slicing and filtering methods based on data parameters and timestamps.

We begin with a motivating example. We assume that a system logs relevant system actions together with the time when they are carried out. 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 $\mathit{ssh\_login}(c, s)@\tau$. We monitor these logs to check compliance with policies. An example of 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: $\mathit{ssh\_login}(c, s)@\tau$ and $\mathit{ssh\_logout}(c, s)@\tau$, 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.

5.1 Slicing Framework

In 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.

Recall that V is the set of variables. In the remainder of this chapter, 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 5.1.1.

Definition 5.1.1. *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 5.1.2. *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 5.1.3. *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 5.1.4 we give conditions that the refined restrictions must fulfill.

Definition 5.1.4. *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, 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. 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 5.1.5 we give requirements that the slices and the associated restrictions produced by a slicer must fulfill. In Theorem 5.1.6 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 5.2 and 5.3, 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 Chapter 6.

Definition 5.1.5. *A* slicer $\mathfrak{s}_\phi$ *for the formula* ϕ *is a function that takes as input a temporal structure* $(\bar{\mathfrak{D}}, \bar{\tau})$ *and a restriction* (D, T)*. It returns a family of temporal structures* $(\bar{\mathfrak{D}}^k, \bar{\tau}^k)_{k \in K}$ *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) $(\bar{\mathfrak{D}}^k, \bar{\tau}^k)$ *is* (D^k, T^k)-sound *for* $(\bar{\mathfrak{D}}, \bar{\tau})$ *and* ϕ*, for each* $k \in K$*.*

(S3) $(\bar{\mathfrak{D}}^k, \bar{\tau}^k)$ *is* (D^k, T^k)-complete *for* $(\bar{\mathfrak{D}}, \bar{\tau})$ *and* ϕ*, for each* $k \in K$*.*

Theorem 5.1.6. *Let* $\mathfrak{s}_\phi$ *be a slicer for the formula* ϕ*. Let* $\mathfrak{s}_\phi$*, on input temporal structure* $(\bar{\mathfrak{D}}, \bar{\tau})$ *and restriction* (D, T)*, return the family of temporal structures* $(\bar{\mathfrak{D}}^k, \bar{\tau}^k)_{k \in K}$ *and the family of restrictions* $(D^k, T^k)_{k \in K}$ *as output. The following conditions are equivalent:*

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

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

Proof. (2) follows from (1) because $(D^k, T^k)_{k \in K}$ refines (D, T) (condition (S1) in Definition 5.1.5) and because $(\bar{\mathfrak{D}}^k, \bar{\tau}^k)$ is (D^k, T^k)-sound for $(\bar{\mathfrak{D}}, \bar{\tau})$ and ϕ, for each $k \in K$ (condition (S2) in Definition 5.1.5).

To show that (2) implies (1), we show the contrapositive. Let v be a valuation and $i \in \mathbb{N}$ such that $(\bar{\mathfrak{D}}, \bar{\tau}, v, i) \not\models \phi$, 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 5.1.5), there is a $k \in K$ such that v is permitted by (D^k, T^k) and $\tau_i \in T^k$. Because $(\bar{\mathfrak{D}}^k, \bar{\tau}^k)$ is (D^k, T^k)-complete for $(\bar{\mathfrak{D}}, \bar{\tau})$ and ϕ (condition (S3) in Definition 5.1.5), $(\bar{\mathfrak{D}}^k, \bar{\tau}^k, v, j) \not\models \phi$, for some $j \in \mathbb{N}$ with $\tau^k_j = \tau_i$. □

Note that Theorem 5.1.6 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 5.1.7. Afterwards, in Theorem 5.1.8, we prove that the combination of slicers is again a slicer.

Definition 5.1.7. *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 5.1.8. *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 5.1.5. 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). □

5.2 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 5.1.1 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.

5.2.1 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 5.2.1. *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 5.2.1. 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 5.2.2. *Consider the formula $\phi = snd(src, msg) \rightarrow \Diamond_{[0,6)}\, rcv(\mathsf{0}, msg)$, where src 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*

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

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

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

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 5.2.1 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 5.2.5 that a data slicer is a slicer, and therefore Theorem 5.1.6 applies.

Lemma 5.2.3. *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 $r(t_1, \dots, t_{\iota(r)})$.

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 $r(t_1, \dots, t_{\iota(r)})$ we show the two directions of the equivalence separately.

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

2. Next, we show that $(\bar{\mathfrak{D}}', \bar{\tau}, v, i) \not\models r(t_1, \ldots, t_{\iota(r)})$ implies $(\bar{\mathfrak{D}}, \bar{\tau}, v, i) \not\models r(t_1, \ldots, t_{\iota(r)})$. From $(\bar{\mathfrak{D}}', \bar{\tau}, v, i) \not\models r(t_1, \ldots, t_{\iota(r)})$ it follows that $(v(t_1), \ldots, v(t_{\iota(r)})) \notin r^{\mathfrak{D}'_i}$. We first show that one of the Conditions (a)–(c) in Definition 5.2.1 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 $v(t_j) = c^{\bar{\mathfrak{D}}}$ and hence Condition (c) is satisfied.

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

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 5.2.4 and Theorem 5.2.5 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 5.2.4. *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 $(\bar{\mathfrak{D}}, \bar{\tau})$ and a restriction (D, T). It returns a family of temporal structures $(\bar{\mathfrak{D}}^k, \bar{\tau}^k)_{k\in K}$ and a family of restrictions $(D^k, T^k)_{k\in K}$, where $(\bar{\mathfrak{D}}^k, \bar{\tau}^k)$ is the $(S^k \cap D(x), x, \phi)$-slice of $(\bar{\mathfrak{D}}, \bar{\tau})$ and $(D^k, T^k) = (D[x \hookleftarrow S^k \cap D(x)], T)$, for each $k \in K$.*

Theorem 5.2.5. *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 5.1.5.

For (S1), we show that the family $(D^k, T^k)_{k\in K}$ fulfills the conditions (R1)–(R3) in Definition 5.1.4: (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 $D^k = D[x \hookleftarrow S^k \cap D(x)]$ and $T^k = T$, for each $k \in K$.

(S2) and (S3) follow directly from Lemma 5.2.3. □

5.2.2 Data Filter

We can speed up the monitoring of temporal structures by filtering them before monitoring. Filtering removes some parts of the temporal structure, which 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 5.2.6. *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 5.2.3 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 5.2.5.

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 5.3.1, so it makes sense to data filter slices in general.

5.2.3 A Non-Restrictive Fragment

In the following, we describe a fragment where no restrictions are needed. By Lemma 5.2.3 the slices from Definition 5.2.1 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 5.2.7 and *variable overlap* in Definition 5.2.9. 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 5.2.7. *The set* S *is a* valid *slicing set for the temporal structure* $(\bar{\mathfrak{D}}, \bar{\tau})$ *if* $c^{\bar{\mathfrak{D}}} \in S$*, for all* $c \in C$*.*

Example 5.2.8. *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 5.2.1,* 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 5.2.9. *Two distinct* variables x *and* y overlap *in the formula* ϕ *if for some predicate symbol* $r \in R$*,* ϕ *contains atomic subformulas* $r(s_1, \dots, s_{\iota(r)})$ *and* $r(t_1, \dots, t_{\iota(r)})$ *where* $s_j = x$ *and* $t_j = y$*, for some* j *with* $1 \le j \le \iota(r)$*,*

Example 5.2.10. *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 $\Box(p(x) \to q(x)) \vee \blacksquare_{[0,3)} \neg p(y)$ 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 5.2.11. *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 $(\bar{\mathcal{D}}, \bar{\tau})$.*

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 5.1. Note that any syntactic sugar must be unfolded before applying the rules.

Remark 5.2.12. From the labeling rules for *true* and the operators S and U we see that the formulas $\blacklozenge_I \psi$, $\Diamond_I \psi$, $\blacksquare_I \psi$, and $\Box_I \psi$ 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 $\psi : \mathsf{DE}_x$.

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

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

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

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

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

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

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

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

Figure 5.1: Labeling Rules (Slicing by Data)

Remark 5.2.13. We cannot propagate the label DT_x over the operators $\bullet_I$ and $\bigcirc_I$ 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 5.2.14. *Consider the formula $\Box\, \mathit{snd}(\mathit{src}, \mathit{msg}) \to \Diamond_{[0,6)}\, \mathit{rcv}(0, \mathit{msg})$. After unfolding the syntactic sugar for $\to$ we obtain the formula $\Box\, \neg \mathit{snd}(\mathit{src}, \mathit{msg}) \lor \Diamond_{[0,6)}\, \mathit{rcv}(0, \mathit{msg})$. We explain the labeling for the variables msg. The atomic subformulas $\mathit{snd}(\mathit{src}, \mathit{msg})$ and $\mathit{rcv}(0, \mathit{msg})$ are labeled $\mathsf{DF}_{\mathit{msg}}$. The subformula $\neg \mathit{snd}(\mathit{src}, \mathit{msg})$ is labeled $\mathsf{DT}_{\mathit{msg}}$. The label $\mathsf{DT}_{\mathit{msg}}$ propagates through the operator $\lor$ and through the temporal operator $\Box$, so $\Box\, \neg \mathit{snd}(\mathit{src}, \mathit{msg}) \lor \Diamond_{[0,6)}\, \mathit{rcv}(0, \mathit{msg})$ is labeled $\mathsf{DT}_{\mathit{msg}}$. The labeling for the variable src is analogous and $\Box\, \neg \mathit{snd}(\mathit{src}, \mathit{msg}) \lor \Diamond_{[0,6)}\, \mathit{rcv}(0, \mathit{msg})$ is labeled $\mathsf{DT}_{\mathit{src}}$.*

Example 5.2.15. *Consider the formula $\Box\, \mathit{ssh\_login}(c, s) \to \Diamond_{[0,25)}\, \mathit{ssh\_logout}(c, s)$. After unfolding the syntactic sugar for $\to$ we obtain the formula $\Box\, \neg \mathit{ssh\_login}(c, s) \lor \Diamond_{[0,25)}\, \mathit{ssh\_logout}(c, s)$. We explain the labeling for the variable c. The atomic subformulas $\mathit{ssh\_login}(c, s)$ and $\mathit{ssh\_logout}(c, s)$ are labeled DF_c. The formula $\neg \mathit{ssh\_login}(c, s)$ is labeled DT_c. The label DT_c propagates through the operator $\lor$ and then through the temporal operator $\Box$, so $\neg \mathit{ssh\_login}(c, s) \lor \Diamond_{[0,25)}\, \mathit{ssh\_logout}(c, s)$. and $\Box\, \neg \mathit{ssh\_login}(c, s) \lor \Diamond_{[0,25)}\, \mathit{ssh\_logout}(c, s)$ are labeled DT_c. The labeling for the variable s is analogous and $\Box\, \neg \mathit{ssh\_login}(c, s) \lor \Diamond_{[0,25)}\, \mathit{ssh\_logout}(c, s)$ is labeled DT_s.*

Theorem 5.2.16. *For all formulas ϕ and all variables $x \in V$, if the derivation rules shown in Figure 5.1 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 5.2.16 establishes the correctness of our labeling rules. Before we formally show its correctness, we intuitively explain 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 5.1 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 5.1. Consider the formula $\phi \mathrel{\mathsf{S}_I} \psi$. 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 $\phi \mathrel{\mathsf{S}_I} \psi$ 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 $\phi \mathrel{\mathsf{S}_I} \psi$ and it is in DE_x.

Proof. We prove Theorem 5.2.16 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:

- $r(t_1, \dots, t_{\iota(r)})$ where $x \in \{t_1, \dots, t_{\iota(r)}\}$. 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 $(v(t_1), \dots, v(t_{\iota(r)}))$, none of the Conditions (a)–(c) in Definition 5.2.1 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 $(v(t_1), \dots, v(t_{\iota(r)}) \notin r^{\mathcal{P}_i}$, $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \not\models r(t_1, \dots, t_{\iota(r)})$, and hence $r(t_1, \dots, t_{\iota(r)})$ is in DF_x.

- $r(t_1, \dots, t_{\iota(r)})$ where $x \notin \{t_1, \dots, t_{\iota(r)}\}$. 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 valuation 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 $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models r(t_1, \dots, t_{\iota(r)})$ iff $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models r(t_1, \dots, t_{\iota(r)})$, so that $r(t_1, \dots, t_{\iota(r)})$ is in DE_x.

  We first show the implication from right to left. Suppose that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models r(t_1, \dots, t_{\iota(r)})$, that is, $(v(t_1), \dots, v(t_{\iota(r)}) \in r^{\mathcal{D}_i}$. Because the variable x is not among the terms $t_1, \dots, t_{\iota(r)}$, 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 5.2.1 is satisfied for every such tuple $(v(t_1), \dots, v(t_{\iota(r)}))$. Hence, $(v(t_1), \dots, v(t_{\iota(r)}) \in r^{\mathcal{P}_i}$ and thus $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models r(t_1, \dots, t_{\iota(r)})$.

  We show the implication from left to right by contradiction. Suppose that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models r(t_1, \dots, t_{\iota(r)})$, that is, $(v(t_1), \dots, v(t_{\iota(r)}) \notin r^{\mathcal{D}_i}$. It follows that $(v(t_1), \dots, v(t_{\iota(r)}) \notin r^{\mathcal{P}_i}$ and $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \not\models r(t_1, \dots, t_{\iota(r)})$.

- $t \prec t'$. 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, $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models$ *true* and hence *true* is in DT_x.

  It also trivially holds that $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models$ *true* and hence $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models$ *true* iff $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models$ *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.

- $\CIRCLE_I \psi$ 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 $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \not\models \CIRCLE_I \psi$. For $i > 0$ with $\tau_i - \tau_{i-1} \notin I$ and for $i = 0$ it follows from the definition of the operator $\CIRCLE_I$ that $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \not\models \CIRCLE_I \psi$. Therefore, $\CIRCLE_I \psi$ 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 $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \models \CIRCLE_I \psi$ iff $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models \CIRCLE_I \psi$. For $i > 0$ with $\tau_i - \tau_{i-1} \notin I$ and for $i = 0$ it follows from the definition of the operator $\CIRCLE_I$ that $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \not\models \CIRCLE_I \psi$ and $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \CIRCLE_I \psi$. It follows trivially that $(\bar{\mathcal{P}}, \bar{\tau}, v, i) \not\models \CIRCLE_I \psi$ iff $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models \CIRCLE_I \psi$. Therefore, $\CIRCLE_I \psi$ is in DE_x.

- $\bigcirc_I \psi$. 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.

□

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 5.2.18 to show that in this case the data slicer does not need to tighten the restrictions.

Lemma 5.2.17. *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 5.2.18. *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 5.2.17. □

Finally, we show that membership in the sets DT_x, DF_x, or DE_x is undecidable. We first establish Lemma 5.2.19 that we use in the undecidability proof.

Lemma 5.2.19. *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 $r(t_1, \dots, t_{\iota(r)})$. Since ϕ does not contain the variable x, x is not in $\{t_1, \dots, t_{\iota(r)}\}$ and hence all atomic formulas can be labeled with DE_x.

The step case follows straightforwardly from the fact that all atomic sub-formulas are labeled with DE_x and from the fact that, according to the labeling rules in Figure 5.1, 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 5.2.16 that ϕ is in the set DE_x. □

Theorem 5.2.20 establishes undecidability of set membership. The theorem follows from the undecidability of the tautology problem for FOTL formulas, established in Lemma 3.3.3.

Theorem 5.2.20. *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 5.2.19 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 $(\bar{\mathcal{D}}', \bar{\tau}', v[x \hookleftarrow d], 0) \not\models \phi$, for every $d \in \mathbb{D} \setminus S$. There is at least one such value d because $S \subsetneq \mathbb{D}$. From $(\bar{\mathcal{D}}', \bar{\tau}', v[x \hookleftarrow d], 0) \not\models \bullet\, \mathit{true}$ it follows that $(\bar{\mathcal{D}}', \bar{\tau}', v[x \hookleftarrow d], 0) \not\models \phi \vee \bullet\, \mathit{true}$. 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 $\phi \vee p(x) \vee \CIRCLE\, \mathit{true}$ 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 $(\bar{\mathcal{D}}', \bar{\tau}', v[x \hookleftarrow d], 0) \models p(x)$ and hence $(\bar{\mathcal{D}}', \bar{\tau}', v[x \hookleftarrow d], 0) \models \phi \vee p(x) \vee \CIRCLE\, \mathit{true}$. Let S be a slicing set with $d \notin S$. Then $(\bar{\mathcal{D}}'', \bar{\tau}'', v[x \hookleftarrow d], 0) \not\models p(x)$, where $(\bar{\mathcal{D}}'', \bar{\tau}'')$ is the $(S, x, \phi \vee p(x) \vee \CIRCLE\, \mathit{true})$-slice of $(\bar{\mathcal{D}}', \bar{\tau}')$. From Lemma 5.2.19 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 $(\bar{\mathcal{D}}'', \bar{\tau}'', v[x \hookleftarrow d], 0) \not\models \phi$. Finally, from $(\bar{\mathcal{D}}'', \bar{\tau}'', v[x \hookleftarrow d], 0) \not\models \CIRCLE\, \mathit{true}$ it follows that $(\bar{\mathcal{D}}'', \bar{\tau}'', v[x \hookleftarrow d], 0) \not\models \phi \vee p(x) \vee \CIRCLE\, \mathit{true}$. Since $(\bar{\mathcal{D}}', \bar{\tau}', v[x \hookleftarrow d], 0) \models \phi \vee p(x) \vee \CIRCLE\, \mathit{true}$, but $(\bar{\mathcal{D}}'', \bar{\tau}'', v[x \hookleftarrow d], 0) \not\models \phi \vee p(x) \vee \CIRCLE\, \mathit{true}$, the formula $\phi \vee p(x) \vee \CIRCLE\, \mathit{true}$ is not in the set DE_x. □

5.3 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 5.1.1 such that $r^{\mathcal{D}'_i} = r^{\mathcal{D}_{s(i)}}$, for all $r \in R$ and $i \in [0, \ell)$.

5.3.1 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 5.3.1 we show how we determine the relevant time range for a formula. We formalize the slicing of a log by time in Definition 5.3.2 and in Definition 5.3.3 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 $\{a \in \mathbb{Z} \mid b \leq a \leq b'\}$ and $(b, b']$ denotes the set $\{a \in \mathbb{Z} \mid b < a \leq b'\}$. 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 5.3.1. *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 $\psi \vee \chi$.*

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

We give intuition for Definition 5.3.1. 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 considering the interval $[0, b)$. 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 5.3.2. *Let $(\bar{\mathcal{D}}, \bar{\tau})$ be a temporal structure 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 5.2 illustrates Definition 5.3.2, 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 $(\bar{\mathcal{D}}, \bar{\tau})$ whose timestamp is in T. In total, there are ℓ time points in $(\bar{\mathcal{D}}, \bar{\tau})$ whose timestamps fall into T. All these time points are included in the T-slice and are identical as in $(\bar{\mathcal{D}}, \bar{\tau})$. To ensure the soundness and completeness of time slices, the $(\ell + 1)$st time point [1] in the T-slice must have a timestamp that lies outside of T, just like the corresponding time point in $(\bar{\mathcal{D}}, \bar{\tau})$.

[1]Note that the index of the $(\ell + 1)$st time point is ℓ because we use a zero-based indexing for time points.

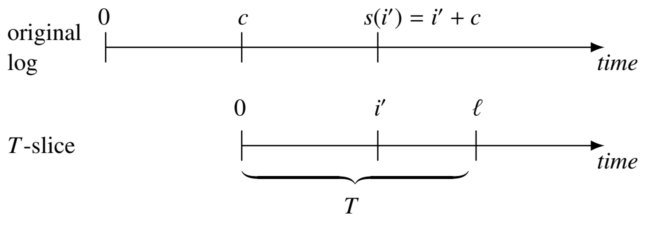

Figure 5.2: Illustration of a T-slice.

Definition 5.3.3. *A* time slicer $\mathrm{t}_{\phi,(I^k)_{k\in K}}$ *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 $((I^k \cap T) \oplus \mathrm{RI}(\phi))$-slice of $(\bar{\mathcal{D}}, \bar{\tau})$ and $(D^k, T^k) = (D, I^k \cap T)$, for each $k \in K$.*

Theorem 5.3.9 shows that a time slicer is a slicer. However, the proof needs additional machinery to show that a time slicer satisfies the criteria (S2) and (S3) in Definition 5.1.5. We present this machinery in the following.

We first state a definition about overlapping temporal structures and several lemmas that we use in the proof of Theorem 5.3.9.

Definition 5.3.4. *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 5.3.2. 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 5.3.5 establishes that time slices overlap and Lemma 5.3.6 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 5.3.5. *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 5.3.2. $(\bar{\mathcal{D}}', \bar{\tau}')$ 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 5.3.4 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 $c = \min\{k \in \mathbb{N} \mid \tau_k \in T \oplus I\}$ in Definition 5.3.2 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 5.3.6. *Let $(\bar{\mathfrak{D}}, \bar{\tau})$ and $(\bar{\mathfrak{D}}', \bar{\tau}')$ 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 $(\bar{\mathfrak{D}}, \bar{\tau})$ and $(\bar{\mathfrak{D}}', \bar{\tau}')$ 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 $(\bar{\mathfrak{D}}, \bar{\tau})$ and $(\bar{\mathfrak{D}}', \bar{\tau}')$ are (I, c, i)-overlapping, Condition 1 in Definition 5.3.4 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 5.3.4 holds. □

Lemma 5.3.7 establishes that 0 is included in the relative interval of every formula. This guarantees that the satisfaction relation in Lemma 5.3.8 is defined.

Lemma 5.3.7. *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 $r(t_1, \dots, t_{\iota(r)})$, where t, t', and $t_1, \dots, t_{\iota(r)}$ are variables or constants. It follows trivially from Definition 5.3.1 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, $0 \in \mathrm{RI}(\psi) \Cup \mathrm{RI}(\chi)$ 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 5.3.1 that $0 \in \mathrm{RI}(\phi)$.

□

Lemma 5.3.8 establishes the soundness and completeness of the slices.

Lemma 5.3.8. *Let ϕ be a formula and $(\bar{\mathfrak{D}}, \bar{\tau})$ and $(\bar{\mathfrak{D}}', \bar{\tau}')$ temporal structures. If $(\bar{\mathfrak{D}}, \bar{\tau})$ and $(\bar{\mathfrak{D}}', \bar{\tau}')$ are $(\mathrm{RI}(\phi), c, i)$-overlapping, for some c and i, then for all valuations v it holds that $(\bar{\mathfrak{D}}, \bar{\tau}, v, i) \models \phi$ iff $(\bar{\mathfrak{D}}', \bar{\tau}', v, i - c) \models \phi$.*

Proof. We prove Lemma 5.3.8 by structural induction on the form of the formula ϕ, as the corresponding Lemma 5.2.3. In contrast to Lemma 5.2.3, the arguments for the atomic subformulas are trivial and the arguments for the non-atomic subformulas are complex.

Note that for all cases, $(\bar{\mathfrak{D}}', \bar{\tau}', v, i - c) \models \phi$ is defined: it follows from Lemma 5.3.7 that $0 \in \mathrm{RI}(\phi)$ and from Condition 1 in Definition 5.3.4 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 $(\bar{\mathfrak{D}}, \bar{\tau}, v, i) \models t \approx t'$ iff $v(t) = v(t')$ iff $(\bar{\mathfrak{D}}', \bar{\tau}', v, i - c) \models t \approx t'$, for all valuations v.
- $t \prec t'$, where t and t' are variables or constants. This case is similar to the previous one.

- $r(t_1, \dots, t_{\iota(r)})$, where $t_1, \dots, t_{\iota(r)}$ are variables or constants. Since $(\bar{\mathcal{D}}, \bar{\tau})$ and $(\bar{\mathcal{D}}', \bar{\tau}')$ are $(RI(r(t_1, \dots, t_{\iota(r)}), c, i)$-overlapping and $0 \in RI(r(t_1, \dots, t_{\iota(r)}))$, it also follows from Condition 1 in Definition 5.3.4 that $\mathcal{D}_i = \mathcal{D}'_{i-c}$ and hence $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \models r(t_1, \dots, t_{\iota(r)})$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v, i - c) \models r(t_1, \dots, t_{\iota(r)})$, 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 $(\mathrm{RI}(\psi) \Cup \mathrm{RI}(\chi), c, i)$-overlapping. From $RI(\psi) \subseteq \mathrm{RI}(\psi) \Cup \mathrm{RI}(\chi)$, $RI(\chi) \subseteq \mathrm{RI}(\psi) \Cup \mathrm{RI}(\chi)$, and Lemma 5.3.6 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 $(\bar{\mathcal{D}}, \bar{\tau}, v[x \hookleftarrow d], i) \models \psi$ iff $(\bar{\mathcal{D}}', \bar{\tau}', v[x \hookleftarrow d], i - c) \models \psi$. 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 $\mathrm{RI}(\bullet_{[a,b)} \psi) = (-b, 0] \Cup ((-b, -a] \oplus \mathrm{RI}(\psi))$.

  From $0 \in \mathrm{RI}(\bullet_{[a,b)} \psi)$ and from Condition 1 in Definition 5.3.4 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 5.3.4 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 5.3.4 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 5.3.6 $(\bar{\mathcal{D}}, \bar{\tau})$ 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 5.3.4 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 5.3.4 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 $\mathrm{RI}(\bigcirc_{[a,b)} \psi) = [0,b) \uplus ([a,b) \oplus \mathrm{RI}(\psi))$.

  From $0 \in \mathrm{RI}(\bigcirc_{[a,b)} \psi)$ and from Condition 1 in Definition 5.3.4 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 5.3.4 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 5.3.6 $(\bar{\mathcal{D}}, \bar{\tau})$ 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 5.3.4 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 $\mathrm{RI}(\psi \,\mathsf{S}_{[a,b)} \chi) = (-b, 0] \uplus ((-b,0] \oplus \mathrm{RI}(\psi)) \uplus ((-b,-a] \oplus \mathrm{RI}(\chi))$.

  Note that $0 \in \mathrm{RI}(\psi \,\mathsf{S}_{[a,b)} \chi)$, so from Condition 1 in Definition 5.3.4 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 5.3.6, $(\bar{\mathcal{D}}, \bar{\tau})$ 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 5.3.6 $(\bar{\mathcal{D}}, \bar{\tau})$ 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 5.3.4 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 5.3.4 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 5.3.4 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 5.3.4 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.

□

Finally, we show in Theorem 5.3.9 that a time slicer is a slicer.

Theorem 5.3.9. *The time slicer* $\mathsf{t}_{\phi,(I^k)_{k\in K}}$ *is a slicer for the formula* ϕ *if* $\bigcup_{k\in K} I^k = \mathbb{N}$.

Proof. The proof has the same structure as the proof of Theorem 5.2.5. We show that a time slicer $\mathsf{t}_{\phi,(I^k)_{k\in K}}$ satisfies the criteria (S1)-(S3) in Definition 5.1.5 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 5.1.4: (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 5.3.5 and from Lemma 5.3.8. □

For RI(ϕ) 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 5.3.9 will do, but, as illustrated by Example 5.3.10, the choice can influence the overhead of monitoring the slices.

Example 5.3.10. *Consider the formula $\square\, p \rightarrow \blacklozenge_{[0,15)}\, q$ and assume a log, where the timestamps are given as days. We have that* $\mathrm{RI}(p \rightarrow \blacklozenge_{[0,15)}\, q) = (-15, 0]$. *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. As a consequence, each time point is monitored three times. Namely, once when monitoring the 1-week equivalent period and once 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 as there are less slices.*

5.3.2 Filtering Time Points

After a data slicer removes tuples from the relations in a temporal structure, there may be many time points left 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 5.3.13). However, for some formulas, the filtered temporal structure is not sound and complete. We identify a fragment of formulas for which such filtering can be safely used (Theorem 5.3.17).

Definition 5.3.11. *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 5.3.12. *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 $s : [0, \ell) \rightarrow \mathbb{N}$ 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 5.3.12 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 5.3.13. *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 5.3.17. 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.

Definition 5.3.14. *The sets* FT, FF, *and* FE *are defined such that for every formula ϕ the following holds:*

- $\phi \in$ 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 $(\bar{\mathcal{D}}, \bar{\tau})$.*
- $\phi \in$ 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 $(\bar{\mathcal{D}}, \bar{\tau})$.*
- $\phi \in$ 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 5.3. Afterwards, it assigns the label FE according to the rules in Figure 5.4.

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

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

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

Figure 5.3: Labeling Rules 1 (Empty-time-point Filter)

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

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

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

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

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

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

Figure 5.4: Labeling Rules 2 (Empty-time-point Filter)

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 5.3.15.

Theorem 5.3.15. *For all formulas ϕ, if the derivation rules shown in Figures 5.3 and 5.4 assign the label* FT, FF, *or* FE *to ϕ then ϕ is in the set* FT, FF, *or* FE, *respectively.*

Proof. We first show the soundness of the rules assigning 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 $r(t_1,\dots,t_{r(l)})$ is labeled FF. If i is an empty time point in $\mathcal{D}$ then clearly $(\bar{\mathcal{D}}, \bar{\tau}, v, i) \not\models r(t_1,\dots,t_{r(l)})$.

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 show the soundness of the rules assigning 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 $r(t_1, \dots, t_{r(l)})$, $t \approx t'$, 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 $j' \le i'$ such that $\tau'_{i'} - \tau'_{j'} \in I$ and $(\bar{\mathcal{D}}', \bar{\tau}', v, j') \models \psi$ and, for every k' with $j' < k' \le i'$, 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 $s(j') < k \le s(i')$ 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 $j' < k' \le i'$ 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 $j \le s(i')$ with $\tau_{s(i')} - \tau_j \in I$ and $(\bar{\mathcal{D}}, \bar{\tau}, v, j) \models \psi$, and that, for every k with $j < k \le s(i')$, 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 $j' \le i'$ 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 $j' < k' \le i'$ 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 $s(j') < k \le s(i')$. 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.

□

Lemma 5.3.16 establishes the soundness and completeness of an empty-time-point-filtered slice for formulas that are in both sets FE and FT.

Lemma 5.3.16. *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 (D, T)-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 $(\bar{\mathcal{D}}', \bar{\tau}', v, i') \not\models \phi$. □

Theorem 5.3.17. *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.

Proof. We show that the empty-time-point filter satisfies the criteria (S1)-(S3) in Definition 5.1.5 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 5.3.16.

□

It follows from Theorem 5.3.15 and from Theorem 5.3.17 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.

6 The Google Case Study

In this chapter, we describe our deployment of compliance monitoring based on log slicing 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.

6.1 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*:** SSH session must not last longer than 24 hours. The motivation is that long-running SSH sessions present a security risk.

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

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

***P6*:** 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.
- $upd\_connect(c : string)$: 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 6.1. 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 *P4*, because of the subformula $\blacklozenge_{[1d,2d]}\, alive(c)$ we only consider computers that have recently been used. This

Table 6.1: Policy formalizations in MFOTL

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

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 *P5*, 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 (see Section 3.4), 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 (see Definition 3.2.2), 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 P6 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 5.3.2 and 5.2.3 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 in the policy *P1* 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

Table 6.2: 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)

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. The collapsed temporal structure contains approximately 77.2 million time points and 26 billion log events, that is, 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 6.2. The collapsed temporal structure encoded in a protocol buffer format [Goo13] 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.

6.2 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 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 [DG04] 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 6.1, for checking compliance with the policies, we used the MONPOLY tool [BHKZ12] together with Google's MapReduce framework [DG04]. 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 5.2.1 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 5.2.1. 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 [Wik13] 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 6.1. 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 6.1, the filtering and slicing does not introduce any spurious violations that must be removed from the monitor output in a post-processing step.

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.

6.3 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 6.1 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 6.1 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 6.3 we show the time it took to monitor each policy using our monitoring setup. [1] For most policies, the monitoring took up to two and a half hours. Monitoring the policy *P3*

[1]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.

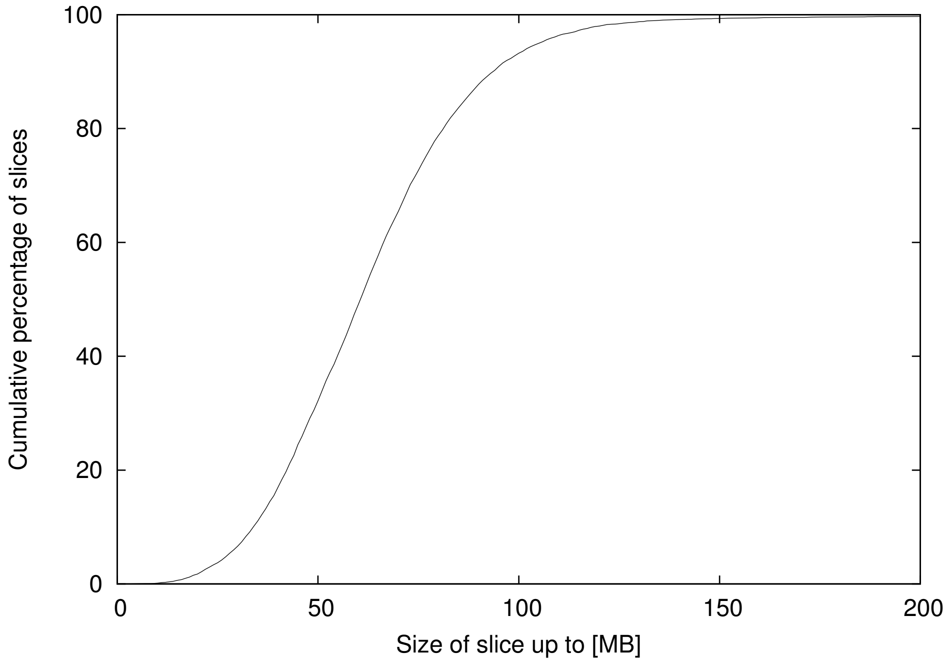

Figure 6.1: Distribution of the size of log slices in MONPOLY format

Table 6.3: Monitor performance

Policy	Runtime [hh:mm]
P1	2:04
P2	2:10
P3	11:56
P4	2:32
P5	2:28
P6	2:13

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 6.4 presents details about the monitoring of the individual slices. For the policy *P3*, Figure 6.2 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

Table 6.4: 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
P4	169	1:06	21.3	9.2	13.1
P5	168	1:01	21.3	6.1	6.1
P6	168	0:48	21.1	6.1	7.1

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 6.3 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 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

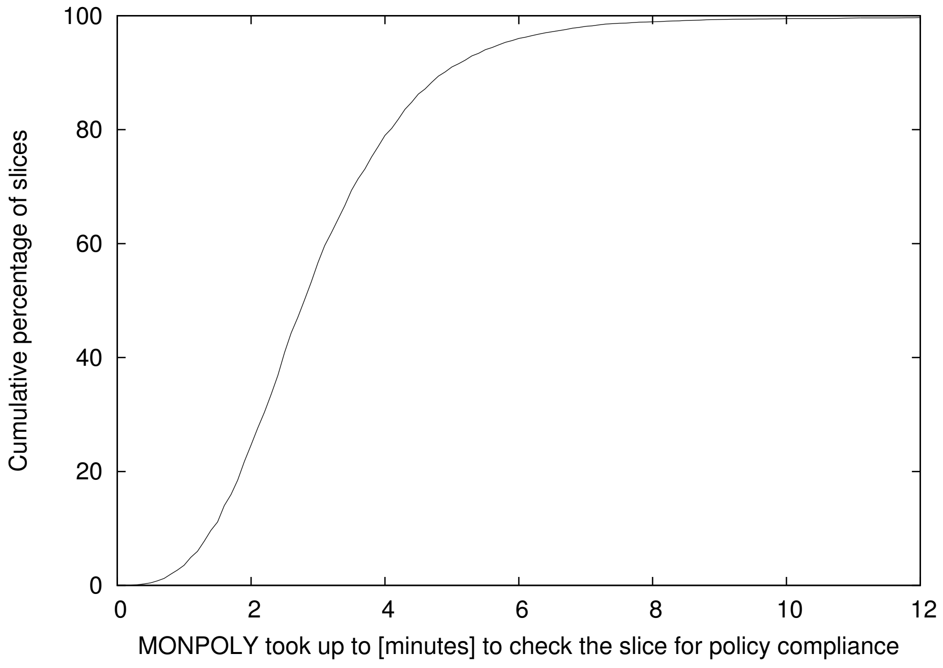

Figure 6.2: Distribution of time to monitor individual slices for policy *P3*

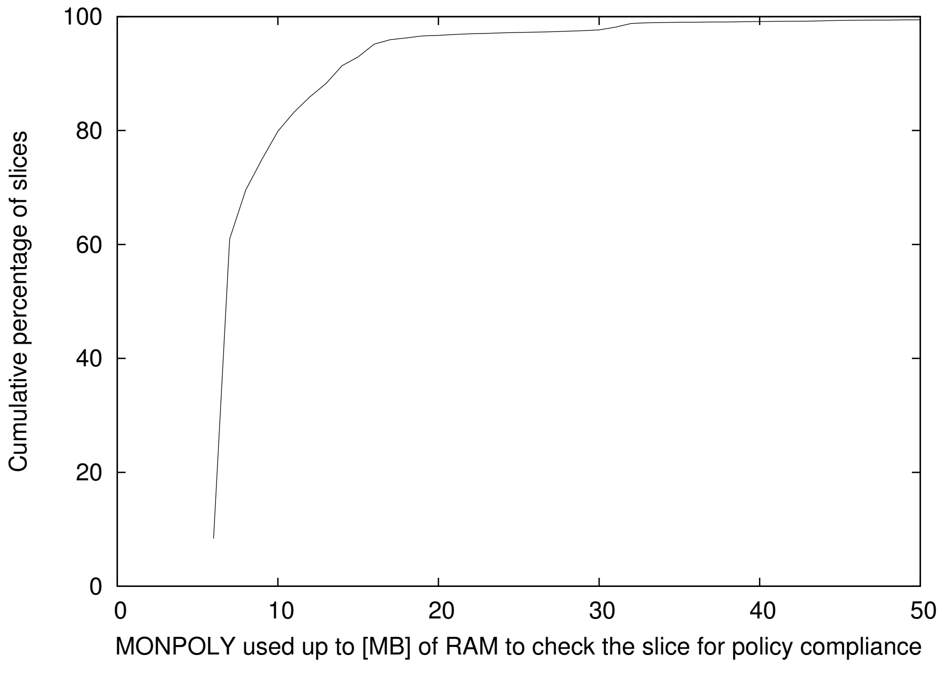

Figure 6.3: Distribution of memory used to monitor individual slices for policy *P3*

identifying seemingly unrelated changes in the configuration of other components in the IT infrastructure.

7 Related Work

Temporal logics have been widely used to formalize and analyze security and privacy policies. For example, Zhang et al. [ZPPSP05] formalize the UCON_{ABC} model [PS04] for usage control in the temporal logic of actions [Lam94]. The Obligation Specification Language, presented by Hilty et al. [HPB$^+$07], includes temporal operators. Barth et al. [BDMN06] present a framework for specifying privacy policies in a first-order temporal logic and DeYoung et al. [DGJ$^+$10] 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 logs containing concurrently logged actions and 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 [DJLS08,GHS10,BGHS10,HV12,BKMP08,BHKZ12]. However, these approaches cannot directly monitor concurrently logged actions and they do not scale to large logs due to a lack of parallelization. We discuss related work that addresses these two issues in the following two sections.

7.1 Monitoring Concurrently Logged Actions

In this section, we discuss work related to monitoring concurrently logged actions.

A broader overview on the state of the art of monitoring distributed systems can be found in the survey by Goodloe and Pike [GP10]. Sen et al. [SVAR04] present a distributed monitoring approach, where multiple monitors which communicate with each other are implemented locally. The authors use a propositional past linear-time distributed temporal logic with epistemic operators [Ram96] 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 [Lam78] 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 and a single central monitor determines whether a global property holds. In addition, note that distributed monitoring entails communication overhead between the monitors whereas we must merge distributed logs.

Genon et al. [GMM06] present a monitoring algorithm for propositional LTL, where events are partially ordered. Whereas we restrict ourselves to formulas for which monitoring a single interleaving is sufficient, their approach checks a formula on all interleavings using symbolic exploration methods. These methods can decrease the number of interleavings considered but, in the worst case, exponentially many must still be examined. Furthermore, it is unclear how their algorithm for the propositional setting extends to a timed and first-order setting.

Wang et al. [WASL12] consider a problem similar to that of Genon et al. [GMM06]. Their monitoring algorithm for past-only propositional LTL with a three-valued semantics explicitly explores the possible interleavings of a partially ordered trace. Matching our notion of strong violations (Definition 3.1.2(2)), their algorithm returns the truth value false only if the formula is violated on all interleavings. However, their algorithm is not complete in the sense that it might return an inconclusive answer, represented as the third truth value, although all possible interleavings violate the given formula. The third truth value is also returned if some interleavings violate the formula and others satisfy it. Note that in our approach, the formulas in the syntactically-defined fragments either satisfy all interleavings or violate all interleavings.

Several monitoring approaches [BF12, BLS06, ZSLL09] have been proposed where actions logged with equal timestamps are considered to happen simultaneously. This corresponds to defining their semantics with respect to the collapsed log in our setting and thereby restricting the expressiveness of the policy specification language. Therefore different possible interleavings need not be considered and the monitoring can be more efficient. We discuss these approaches in more detail below.

Bauer et al. [BLS06] assume a setting where the observed system actions are totally ordered, thereby abstracting away distributivity and concurrency. In their setting, system requirements are given in a propositional linear-time temporal logic. Their monitoring architecture additionally includes a component that analyzes the cause of a failure, which is fed back into the system.

Bauer and Falcone [BF12] assume a distributed system with synchronized clocks where observations of the system are done simultaneously in lock-step. This leads to a totally ordered trace of system actions, which corresponds to the collapsed log in our setting. They 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. Each monitor evaluates the subformulas for which it can observe the relevant system actions. Note that their monitoring algorithm is based on rewriting of formulas so, technically speaking, partially rewritten formulas are exchanged.

Zhou et al. [ZSLL09] present a distributed monitoring framework aimed at monitoring properties of network protocols. They check the properties against a totally ordered log with exactly one time point per timestamp. Again, this corresponds to the collapsed log in our setting. 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.

Mazurkiewicz traces [DR95] provide an abstract view on partially ordered logs. With this view, the problem of checking whether a policy is strongly violated on a partially ordered log can be stated as checking whether all linearizations of a Mazurkiewicz trace satisfy a temporal property. We are not aware of any work that solves this problem by inspecting a single sequence representing all linearizations. In Mazurkiewicz traces, an independence relation on actions specifies which actions can be reordered. This is independent of the timestamps of actions, whereas in our setting the possibility of reordering depends on the timestamp and not the action.

Also related to our work is partial-order reduction [Pel98]. Partial-order techniques aim at reducing the number of interleavings that are sufficient for checking whether a temporal property is satisfied on all possible interleavings. Partial-order reduction techniques have successfully been used in finite-state model checking where one checks all possible system executions. In contrast, we check compliance of all linearizations of a single observed system execution. Nevertheless, our approach for the interleaving-sufficient fragment can be seen as a special case of partial-order reduction. Namely, we restrict the logical formulas so that it is sufficient to inspect a single interleaving to determine compliance of all possible interleavings. For the collapse-sufficient fragment, we additionally compress the inspected interleaving.

7.2 Monitoring Large Logs

In this section, we discuss work related to monitoring log slices.

Similar to our approach, Barre et al. [BKSB$^+$13] monitor logs with a MapReduce framework. While we split the log into multiple slices and evaluate the whole formula on these slices in parallel in a single MapReduce job, 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 does not clarify the scalability of their approach. Finally, they evaluated their approach only for a propositional temporal logic, which is limited in expressing realistic policies.

Roşu and Chen [RC12] present a general extension for different property specification languages and associated monitoring algorithms where logged events have parameters. Considering log slices based on these parameters 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, actions corresponding to the logged 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. [SVAR04], Bauer and Falcone [BF12], and Zhou et al. [ZSLL09] 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. This is in contrast to our approach, where different slices can be monitored independently and at different speeds. 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. We already discussed these three approaches [SVAR04, BF12, ZSLL09] in more detail in Section 7.1.

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 [DG92, OV11], are related our work. In the terminology of the DBMS community [OV11], the slicing of logs can be seen as horizontal 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. [BKSB+13], 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 [DG85] to merge the results of evaluating the join operator on subsets of the data. The underlying reason for this is that we choose the slices in a way that is suitable for the monitored formula. On the contrary, the "slicing" in DBMSs is chosen independently of the evaluated query and the database is not "re-sliced" before evaluating other queries.

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. Several techniques overcome this limitation. For instance, Condie et al. [CCA+10] 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 such extensions of MapReduce allow 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 [ABW06] on rapidly evolving data streams in an online setting. We refer to [CM12] 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.

8 Conclusion

We conclude this thesis with a summary of our main results and a discussion of directions for future work.

8.1 Summary

We presented a scalable solution to monitor security policies in concurrent distributed systems. The main obstacle is that these systems provide only partially ordered logs. We showed the intractability of monitoring an arbitrary linear-time temporal logic formula on such logs and we overcame this obstacle by identifying two fragments, which can be monitored efficiently by inspecting a single, representative totally ordered log. We also showed that the semantically-defined properties of a formula are undecidable and we approximated them with sound, but incomplete, syntactically-defined fragments.

The scalability of our solution to monitor large logs 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 parallelizing the monitoring process: First, it allows us to efficiently reorganize a huge log 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.

We deployed and evaluated our monitoring approach in real-world applications. Our two case studies demonstrate the feasibility and benefits of monitoring security policies, and the usage of sensitive data in particular. While a single monitor could cope with the volume of logs in the Nokia case study, the Google case study shows the scalability of our solution for monitoring large logs. The case studies also show that the identified fragments are sufficiently rich to capture real-world policies. Intuitively, this is because one naturally formulates policies that are insensitive to the ordering of concurrently logged actions when their actual ordering cannot be determined easily.

A major practical challenge in deploying our monitoring solution in the case studies was the lack of accurate and reliable logging mechanisms. The underlying cause for this is that the systems were not originally designed to log in sufficient detail the system actions that are relevant for the monitored policies.

8.2 Future Work

Our solution for monitoring concurrently logged actions focuses on detecting strong policy violations, that is those violations which occur on all possible log interleavings. It falls short for detecting weak policy violations, that is those violations which occur on at least one, but not necessarily all possible log interleaving. Our solution is based only on analyzing the policy formalization, but not the logs themselves. Although a similar technique could identify policy formalizations for which there exists a set of logs whose interleaving violates the policy, it remains as future work to develop techniques that determine whether the actual monitored set of logs allows for such a violating interleaving.

As observed in the Google case study, slices that take significantly different amounts of time to monitor unnecessarily prolong the total time needed to check policy compliance. 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. Either heuristics can be used before the monitoring starts, for example, based on the size of the log slices. Alternatively, we could stop the monitoring of a slice after a time-out period, reslice this slice into smaller slices and monitor the smaller slices in parallel. Since our theoretical framework allows one to slice in multiple dimensions by composing different slicing methods, a different slicing method can be used for the reslicing.

It also remains open how to utilize and extend the framework for obtaining a scalable solution for checking system compliance online and for enforcing policies, that is, preventing policy violations rather than just detecting them.

Bibliography

[ABW06] A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: semantic foundations and query execution. *VLDB Journal*, 15(2):121–142, 2006.

[AH91] R. Alur and T. A. Henzinger. Logics and models of real time: A survey. In *Proceedings of the 1991 REX Workshop on Real Time: Theory in Practice*, volume 600 of *Lect. Notes Comput. Sci.*, pages 74–106. Springer, 1991.

[AHV94] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases: The Logical Level*. Addison Wesley, 1994.

[AN10] I. Aad and V. Niemi. NRC data collection campaign and the privacy by design principles. In *Proceedings of the International Workshop on Sensing for App Phones (PhoneSense)*, 2010.

[BDMN06] 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.

[BF12] 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.

[BGHS10] H. Barringer, A. Groce, K. Havelund, and M. Smith. Formal analysis of log files. *J. Aero. Comput. Inform. Comm.*, 7:365–390, 2010.

[BHKZ12] 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.

[BKM10] D. Basin, F. Klaedtke, and S. Müller. Monitoring security policies with metric first-order temporal logic. In *Proceeding of the 15th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 23–34. ACM Press, 2010.

[BKMP08] 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.

[BKSB$^+$13] 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.

[BLS06] A. Bauer, M. Leucker, and C. Schallhart. Model-based runtime analysis of distributed reactive systems. In *Proceedings of the 2006 Australian Software Engineering Conference (ASWEC)*. IEEE Computer Society, 2006.

[CCA$^+$10] 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.

[CG98] C. M. Chase and V. K. Garg. Detection of global predicates: Techniques and their limitations. *Distributed Computing*, 11:191–201, 1998.

[CM12] G. Cugola and A. Margara. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.*, 44(3), 2012.

[DG85] 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.

[DG92] D. DeWitt and J. Gray. Parallel database systems: the future of high performance database systems. *Commun. ACM*, 35(6):85–98, June 1992.

[DG04] 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.

[DGJ$^+$10] 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.

[DJLS08] 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.

[DR95] V. Diekert and G. Rozenberg, editors. *The Book of Traces*. World Scientific Publishing Co., Inc., 1995.

[GHS10] 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.

[GJD11] 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.

[GMM06] A. Genon, T. Massart, and C. Meuter. Monitoring distributed controllers: When an efficient LTL algorithm on sequences is needed to model-check traces. In *Proceedings of the 14th International Symposium on Formal Methods (FM)*, volume 4085 of *Lect. Notes Comput. Sci.*, pages 557–572. Springer, 2006.

[Goo13] Google. Protocol Buffers: Googles Data Interchange Format, 2013. [Online; accessed 6-June-2013].

[GP10] A. Goodloe and L. Pike. Monitoring distributed real-time systems: A survey and future directions. Technical Report NASA/CR-2010-216724, NASA Langley Research Center, July 2010.

[HIP96] The Health Insurance Portability and Accountability Act of 1996 (HIPAA), 1996. Public Law 104-191.

[HMU00] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley, 2nd edition, 2000.

[HPB$^+$07] M. Hilty, A. Pretschner, D. Basin, C. Schaefer, and T. Walter. A policy language for distributed usage control. In *Proceedings of the 12th European Symposium on Research in Computer Security (ESORICS)*, volume 4734 of *Lect. Notes Comput. Sci.*, pages 531–546. Springer, 2007.

[HV12] S. Hallé and R. Villemaire. Runtime enforcement of web service message contracts with data. *IEEE Trans. Serv. Comput.*, 5(2):192–206, 2012.

[KBD$^+$10] N. Kiukkonen, J. Blom, O. Dousse, D. Gatica-Perez, and J. Laurila. Towards rich mobile phone datasets: Lausanne data collection campaign. In *Proceedings of the 7th International Conference on Pervasive Services (ICPS)*, 7 2010.

[Lam78] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[Lam83] L. Lamport. What good is temporal logic? In *Proceedings of the IFIP 9th World Computer Congress*, volume 83 of *Information Processing*, pages 657–668. North-Holland, 1983.

[Lam94] L. Lamport. The temporal logic of actions. *ACM Trans. Program. Lang. Syst.*, 16(3):872–923, May 1994.

[LGPA$^+$12] J. Laurila, D. Gatica-Perez, I. Aad, J. Blom, O. Bornet, T.-M.-T. Do, O. Dousse, J. Eberle, and M. Miettinen. The mobile data challenge: Big data for mobile computing research. In *Pervasive Computing*, 2012.

[MMV08] T. Massart, C. Meuter, and L. Van Begin. On the complexity of partial order trace model checking. *Inform. Process. Lett.*, 106(3):120–126, 2008.

[MS03] N. Markey and P. Schnoebelen. Model checking a path. In *Proceedings of the 14th International Conference on Concurrency Theory (CONCUR)*, volume 2761 of *Lect. Notes Comput. Sci.*, pages 248–262. Springer, 2003.

[OV11] M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*. Springer, 3rd edition, 2011.

[Pel98] D. Peled. Ten years of partial order reduction. In *Proceedings of the 10th International Conference on Computer Aided Verification*, volume 1427 of *Lect. Notes Comput. Sci.*, pages 17–28. Springer, 1998.

[Pnu77] A. Pnueli. The temporal logic of programs. *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 46–57, 1977.

[PS04] J. Park and R. Sandhu. The uconabc usage control model. *ACM Trans. Inf. Syst. Secur.*, 7(1):128–174, February 2004.

[Ram96] 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.

[RC12] G. Roşu and F. Chen. Semantics and algorithms for parametric monitoring. *Log. Method. Comput. Sci.*, 8(1):1–47, 2012.

[RGL01] 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.

[SOX02] Sarbanes-Oxley Act of 2002, 2002. Public Law 107-204.

[SVAR04] 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.

[TvS02] A. S. Tanenbaum and M. van Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall, 2002.

[WASL12] S. Wang, A. Ayoub, O. Sokolsky, and I. Lee. Runtime verification of traces under recording uncertainty. In *Proceedings of the 2nd International Conference on Runtime Verification (RV)*, volume 7186 of *Lect. Notes Comput. Sci.*, pages 442–456. Springer, 2012.

[Wik13] Wikipedia. MurmurHash — Wikipedia, the free encyclopedia, 2013. [Online; accessed 14-April-2013].

[ZPPSP05] 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, November 2005.

[ZSLL09] 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.

Curriculum Vitae

Personal Information

Name:	**Matúš Harvan**
Citizenship:	Slovak Republic
Date of Birth:	29 November 1983

Education

9/2007 – 9/2013	**ETH Zurich, Switzerland** Doctoral student in the Institute of Information Security
9/2005 – 5/2007	**Jacobs University Bremen, Germany** MSc. degree in Computer Science MSc. Thesis: *Connecting Wireless Sensor Networks to the Internet: a 6lowpan Implementation for TinyOS 2.x*
9/2002 – 5/2005	**Jacobs University Bremen, Germany** BSc. degree in Electrical Engineering & Computer Science BSc. Thesis: *Prefix- and Lexicographical-order-preserving IP Address Anonymization*
9/2000 – 5/2002	**Gymnázium Jura Hronca, Bratislava, Slovakia** International Baccalaureate Diploma (41 IB points)

Professional Experience

4/2012 – 11/2012	Internship at Google Switzerland
9/2007 – 8/2013	Teaching Assistant at ETH Zurich
6/2007 – 8/2007	Google Summer of Code 2007 – *Magic Tunnel Daemon* (*mtund*) for FreeBSD
6/2004 – 8/2004	Internship at NEC Europe Ltd., Heidelberg, Germany
10/2002 – 7/2007	Teaching Assistant and Student Assistant at Jacobs University Bremen