Runtime Verification over Out-of-order Streams*

DAVID BASIN, ETH Zürich, Switzerland
FELIX KLAEDTKE, NEC Laboratories Europe GmbH, Germany
EUGEN ZĂLINESCU, Technische Universität München, Germany

We present an approach for verifying systems at runtime. Our approach targets distributed systems whose components communicate with monitors over unreliable channels, where messages can be delayed, reordered, or even lost. Furthermore, our approach handles an expressive specification language that extends the real-time logic MTL with freeze quantifiers for reasoning about data values. The logic's main novelty is a new three-valued semantics that is well suited for runtime verification as it accounts for partial knowledge about a system's behavior. Based on this semantics, we present online algorithms that reason soundly and completely about streams where events can occur out of order. We also evaluate our algorithms experimentally. Depending on the specification, our prototype implementation scales to out-of-order streams with hundreds to thousands of events per second.

CCS Concepts: • **Theory of computation → Logic and verification**; **Modal and temporal logics**; **Verification by model checking**; *Streaming models*; Timed and hybrid models;

Additional Key Words and Phrases: Runtime verification, temporal logic, Kleene logic, stream processing, distributed systems

ACM Reference Format:
David Basin, Felix Klaedtke, and Eugen Zălinescu. 2019. Runtime Verification over Out-of-order Streams. *ACM Trans. Comput. Logic* V, N, Article A (May 2019), 42 pages. https://doi.org/10.1145/3355609

1 INTRODUCTION

Distributed systems are omnipresent and complex, and they can malfunction for many reasons including software bugs and hardware or network failures. Monitoring is an attractive option for verifying at runtime whether a system behavior is correct with respect to a given specification. But distribution opens new challenges. The monitors themselves become components of the (extended) system and like any other system component they may exhibit delays, finite or even infinite, when communicating with other components.

Various runtime-verification approaches exist for different kinds of systems, including distributed systems [Barringer et al. 2004; Basin et al. 2015b; Bauer and Falcone 2016; Bauer et al. 2011; Falcone et al. 2014; Maler and Nickovic 2004; Meredith et al. 2012; Mostafa and Bonakdarbour 2015; Sen et al. 2004]. The specification languages used in these approaches are typically based on temporal logics or finite-state machines, which describe the correct system behavior

*Parts of the work described in this paper have been previously published in the conference papers [Basin et al. 2015a] and [Basin et al. 2017].
This is the authors' version of the work. It is posted here for your personal use. Not for redistribution. The definitive version was published in the ACM Transactions on Computational Logic, https://doi.org/10.1145/3355609.

Authors' addresses: David Basin, ETH Zürich, Department of Computer Science, Universitätstrasse 6, 8092, Zurich, Switzerland, basin@inf.ethz.ch; Felix Klaedtke, NEC Laboratories Europe GmbH, Kurfürsten-Anlage 36, 69115, Heidelberg, Germany, felix.klaedtke@neclab.eu; Eugen Zălinescu, Technische Universität München, Institut für Informatik, Boltzmanstraße 3, 85748, Garching, Germany, eugen.zalinescu@in.tum.de.

1529-3785/2019/5-ARTA $15.00
https://doi.org/10.1145/3355609

in terms of *infinite* streams of system actions. However, at any point in time, a monitor has only partial knowledge about the system's behavior. In particular, a monitor can at best only be aware of the actions the system performed so far, which correspond to a finite prefix of the infinite action stream. For this reason, many of the runtime-verification approaches rely on an extension of the standard Boolean semantics of the linear-time temporal logic LTL with a third truth value, as proposed by Bauer et al. [2010]. Namely, an LTL formula evaluates to the Boolean truth value b on a finite stream of actions σ if the formula evaluates to b on all infinite streams that extend σ; otherwise, the formula's truth value is unknown on σ.

This three-valued semantics, however, only accounts for settings where monitors are always aware of all previously performed actions. It is insufficient to reason soundly and completely about system behavior at runtime when, for example, unreliable channels are used to inform the monitors about the actions performed. In fact, the existing runtime-verification approaches are of limited use for distributed systems where components might crash or network failures occur, for example, when a component is temporarily unreachable and a monitor therefore cannot learn the component's behavior during this time period. Even in the absence of failures, monitors can receive messages about the system behavior in any order due to network delays. A naive solution for coping with out-of-order message delivery is to have the monitor buffer messages and reorder them prior to processing them. However, this can delay reporting a violation when the violation is already detectable on some of the buffered messages. This is undesirable for applications where one cannot afford to wait and the monitor should promptly output its verdict. Moreover, the verdict should remain correct when some of the monitor's knowledge gaps are subsequently closed. Another limitation concerns the expressivity of the specification languages used by the existing runtime-verification approaches for distributed systems. It is not possible to express real-time constraints, which are common requirements for distributed systems. Such constraints specify, for example, deadlines to be met. Furthermore, the supported specification languages cannot handle data values.

In this paper, we present a runtime-verification approach that overcomes these limitations. Our approach handles specifications that are given as formulas in an extension of the real-time logic MTL [Alur and Henzinger 1992; Koymans 1990]. Namely, we extend MTL with a freeze quantifier [Henzinger 1990] to extract data values from events and bind these values to logical variables. We call this extension $\text{MTL}^{\downarrow}$ (pronounced "MTL freeze"), where $\downarrow$ is the symbol for the freeze quantifier. Our runtime-verification approach accounts for out-of-order message deliveries and soundly operates in the presence of failures, such as components crashing. We also provide completeness guarantees for our approach, roughly meaning that in the absence of failures but with arbitrary finite message delays, violations and satisfactions of specifications are eventually reported. We build upon a timed model for distributed systems [Cristian and Fetzer 1999]. The system components use their local clocks to timestamp observations, which they send to the monitors. The monitors use these timestamps to determine the elapsed time between observations, for example, to check whether real-time constraints are met. Furthermore, the timestamps totally order the observations. This is in contrast to a time-free model [Fischer et al. 1985], where the events of a distributed system can only be partially ordered, for example using Lamport timestamps [Lamport 1978]. However, since the accuracy of existing clocks is limited, the monitors' conclusions might only be valid for the provided timestamps. See Section 7.2 where we elaborate on this point.

A cornerstone of our monitoring approach is a new three-valued semantics for $\text{MTL}^{\downarrow}$ that is well suited to reason in settings where system components communicate with the monitors over unreliable channels. Specifically, we define $\text{MTL}^{\downarrow}$'s semantics over the three truth values t, f, and $\bot$. We interpret these truth values as in Kleene logic [Kleene 1950] and conservatively extend the logic's standard Boolean semantics, where t and f stand for "true" and "false," respectively, and the third truth value $\bot$ stands for "unknown" and accounts for the monitor's knowledge gaps. The

models of MTL$^{\downarrow}$ are finite words where knowledge gaps are explicitly represented. Intuitively, a finite word corresponds to a monitor's knowledge about the system behavior at a given time and the knowledge gaps may result from message delays, losses, crashed components, and the like. Critically in our setting, reasoning is monotonic with respect to the partial order on truth values, where $\bot$ is less than t and f, and t and f are incomparable. This monotonicity property guarantees that closing knowledge gaps does not invalidate previously obtained Boolean truth values.

We also present online algorithms for verifying systems at runtime with respect to MTL$^{\downarrow}$ specifications. Our algorithms' output is sound and complete for MTL$^{\downarrow}$'s three-valued semantics and with respect to the monitor's partial knowledge about the actions performed at each point in time. In a nutshell, the algorithms work as follows. They receive as input timestamped messages from the system components, which describe the actions these components perform. No assumptions are made on the order in which these messages are received. The algorithms update their state for each received message. This state comprises an acyclic graph structure for reasoning about the system behavior, that is, computing verdicts about the monitored specification's fulfillment. The graph's nodes store the truth values of the subformulas for the different times that data values are frozen to quantified variables, including the times with no or only partial knowledge. The graph is refined when the monitor receives knowledge about a specific point in time, whereby the nodes representing the knowledge gap are split and instantiated. In each such update, the algorithms propagate data values down to the graph's leaves and propagate Boolean truth values for subformulas up along the graph's edges. When a Boolean truth value is propagated to a root node of the graph, the algorithms output a verdict.

Overall, our main contributions are as follows. First, we define a new three-valued semantics for a temporal logic, which is well suited for runtime verification, in particular, for reasoning about incomplete traces. Second, we present online algorithms to reason soundly and completely about incomplete traces. Moreover, these algorithms output verdicts promptly. Third, we experimentally evaluate the performance of our algorithms and explore the performance impact on handling messages that arrive out of order. Finally, we describe the deployment of our online algorithms for verifying distributed systems at runtime.

The remainder of this paper is structured as follows. In Section 2, we provide preliminaries. In Section 3, we present our new three-valued semantics for monitoring. In the Sections 4 and 5, we present our monitoring algorithms, including a proof of their correctness. We evaluate our algorithms in Section 6. In Section 7, we describe the deployment of our runtime-verification approach for distributed systems. Finally, in the Sections 8 and 9, we discuss related work and draw conclusions.

2 PRELIMINARIES

In this section, we recall standard notation and terminology that will be used throughout the paper.

Intervals. An *interval* I is a nonempty subset of the positive rationals $\mathbb{Q}_{\geq 0}$ such that if $a, b \in I$ and $a \leq c \leq b$ then $c \in I$, for all $a, b, c \in \mathbb{Q}_{\geq 0}$. We use standard notation and terminology for intervals. For example, $(a, b]$ denotes the interval that is left-open with bound a and right-closed with bound b. Note that an interval I with cardinality $|I| = 1$ is a singleton $I = \{\tau\} = [\tau, \tau]$, for some $\tau \in \mathbb{Q}_{\geq 0}$. An interval I is *unbounded* if its right bound is ∞, and *bounded* otherwise. With less-than, $<$, we denote the partial order on intervals, that is $I < J$ iff $I \cap J = \emptyset$ and I's right bound is not greater than J's left bound. Let $I - J := \{\tau - \tau' \mid \tau \in I \text{ and } \tau' \in J\} \cap \mathbb{Q}_{\geq 0}$.

Partial Functions. For a partial function $f : A \nrightarrow B$, let $\mathrm{def}(f) := \{a \in A \mid f(a) \text{ is defined}\}$. If $\mathrm{def}(f) = \{a_1, \ldots, a_n\}$, for some $n \in \mathbb{N}$, we also write $[a_1 \mapsto f(a_1), \ldots, a_n \mapsto f(a_n)]$ for f, when f's domain A and its codomain B are irrelevant or clear from the context. Note that $[\,]$ denotes the partial function that is undefined everywhere. We also

Table 1. Truth tables for three-valued logical operators (strong Kleene logic).

$\neg$	
t	f
f	t
$\bot$	$\bot$

$\vee$	t	f	$\bot$
t	t	t	t
f	t	f	$\bot$
$\bot$	t	$\bot$	$\bot$

$\wedge$	t	f	$\bot$
t	t	f	$\bot$
f	f	f	f
$\bot$	$\bot$	f	$\bot$

$\rightarrow$	t	f	$\bot$
t	t	f	$\bot$
f	t	t	t
$\bot$	t	$\bot$	$\bot$

carry over the notation for set comprehension, for instance, $[a \mapsto a + 1 \mid a \geq 0$ and a is even$]$ denotes the partial function that is defined on the nonnegative even integers and returns their successor. Furthermore, we write $f[a \mapsto b]$ to denote the update of a partial function $f : A \nrightarrow B$ at $a \in A$, i.e., $f[a \mapsto b]$ equals f, except that a is mapped to b if $b \in B$, and $a \notin \mathrm{def}(f[a \mapsto b])$ if $b \notin B$. With $f[a \mapsto \bot]$ we denote the restriction of f to the domain $\mathrm{def}(f) \setminus \{a\}$. Finally, for partial functions $f, g : A \nrightarrow B$, we write $f \sqsubseteq g$ if $\mathrm{def}(f) \subseteq \mathrm{def}(g)$ and $f(a) = g(a)$, for all $a \in \mathrm{def}(f)$.

Truth Values. Let 3 be the set $\{\mathsf{t}, \mathsf{f}, \bot\}$, where t (true) and f (false) denote the standard Boolean values, and $\bot$ denotes the truth value "unknown." Table 1 shows the truth tables of some standard logical operators over 3. Observe that these operators coincide with their Boolean counterparts when restricted to the set $2 := \{\mathsf{t}, \mathsf{f}\}$. We partially order the elements in 3 by their knowledge: $\bot \prec \mathsf{t}$, $\bot \prec \mathsf{f}$, and t and f are incomparable as they carry the same amount of knowledge. Note that $(3, \prec)$ is a lower semilattice where $\wedge$ denotes the meet. We remark that the operators in Table 1 are monotonic, which ensures that reasoning is monotonic in knowledge. Intuitively, when closing a knowledge gap, represented by $\bot$, with t or f, we never obtain a truth value that disagrees with the previous one.

Timed Words. Let Σ be an alphabet. A *timed word* over Σ is an infinite word $(\tau_0, a_0)(\tau_1, a_1) \ldots \in (\mathbb{Q}_{\geq 0} \times \Sigma)^\omega$, where the sequence of τ_is is strictly monotonic and nonzeno, that is, $\tau_i < \tau_{i+1}$, for every $i \in \mathbb{N}$, and for every $t \in \mathbb{Q}_{\geq 0}$, there is some $i \in \mathbb{N}$ such that $\tau_i > t$. Note that we use a dense time domain and assume a nonfictitious clock semantics, that is, there is no stuttering of equal timestamps.

Metric Temporal Logic. Let P be a finite set of predicate symbols, where $\iota(p)$ denotes the arity of $p \in P$. Furthermore, let V be a set of variables and R a finite set of registers. The syntax of the real-time logic $\mathrm{MTL}^{\downarrow}$ is given by the grammar:

$$\varphi \ ::= \ \mathsf{t} \mid p(x_1, \ldots, x_{\iota(p)}) \mid \downarrow^r x.\, \varphi \mid \neg\varphi \mid \varphi \vee \varphi \mid \bullet_I \varphi \mid \bigcirc_I \varphi \mid \varphi \, \mathsf{S}_I \, \varphi \mid \varphi \, \mathsf{U}_I \, \varphi,$$

where $p \in P$, $x, x_1, x_2 \ldots, x_{\iota(p)} \in V$, $r \in R$, and I is an interval. We remark that $\mathrm{MTL}^{\downarrow}$ extends the standard propositional metric temporal logic (MTL) [Alur and Henzinger 1992; Koymans 1990] with a freeze quantifier $\downarrow$. We call a formula an *MTL formula* if all the predicate symbols occurring in it have arity 0 and the freeze quantifier does not occur in it.

A formula is *closed* if each variable occurrence is bound by a freeze quantifier. A formula is *temporal* if the connective at the root of the formula's syntax tree is $\bullet_I$, $\bigcirc_I$, S_I, or U_I. We denote by $sub(\varphi)$ the set of φ's subformulas. We employ standard syntactic sugar. For example, $\varphi \wedge \psi$ abbreviates $\neg(\neg\varphi \vee \neg\psi)$, $\varphi \rightarrow \psi$ abbreviates $\neg\varphi \vee \psi$, and $\Diamond_I \varphi$ ("eventually") and $\Box_I \varphi$ ("always") abbreviate $\mathsf{t} \, \mathsf{U}_I \, \varphi$ and $\neg \Diamond_I \neg\varphi$, respectively. The past-time counterparts $\blacklozenge_I \varphi$ ("once") and $\blacksquare_I \varphi$ ("historically") are defined as expected. The nonmetric variants of the temporal connectives are also easily defined, e.g., $\Box \varphi := \Box_{[0,\infty)} \varphi$. We also use standard conventions concerning the connectives' binding strength to omit parentheses. For example, $\neg$ binds stronger than $\wedge$, which binds stronger than $\vee$, and the connectives $\neg$, $\vee$, etc. bind stronger than the temporal connectives, which bind stronger than the freeze quantifier. Finally, to simplify notation, we omit the superscript r in formulas like $\downarrow^r x.\, \varphi$ whenever $r \in R$ is irrelevant or clear from the context.

Example 2.1. Before defining MTL$^{\downarrow}$'s semantics, we provide some intuition. The following formula formalizes the policy that whenever a customer executes a transaction that exceeds some threshold (e.g. \$2,000), then this customer must not execute any other transaction for a fixed time period (e.g. 3 days).

$$\square \downarrow^{cid} c.\ \downarrow^{tid} t.\ \downarrow^{sum} a.\ trans(c,t,a) \wedge a > 2000 \rightarrow \square_{(0,3]} \downarrow^{tid} t'.\ \downarrow^{sum} a'.\ \neg trans(c,t',a')$$

Note that in the formula, we take the liberty to deviate slightly from the given grammar, which does not include constant and function symbols. Such an extension would be straightforward, but we omit it for the sake of brevity. In particular, the formula contains the constant symbol 2000, interpreted as expected. Furthermore, the binary predicate symbol $>$, also with its expected rigid interpretation, is written in infix.

We assume that the predicate symbol *trans* is interpreted as a singleton relation or the empty set at any point in time. For instance, the interpretation $\{(Alice, 42, 99)\}$ of *trans* at time τ describes the action of *Alice* executing a transaction with identifier 42 with the amount \$99 at time τ. When the interpretation is the empty set, no transaction is executed. We further assume that when the interpretation of the predicate symbol *trans* is nonempty, the registers *cid*, *tid*, and *sum* store (a) the transaction's customer, (b) the transaction identifier, and (c) the transferred amount, respectively. If the interpretation is the empty set, the registers store a dummy value, representing undefinedness.

The variables c, t, a, t', and a' are frozen to the respective register values. For example, c is frozen to the value stored in the register *cid* at each point in time and is used to identify subsequent transactions from this customer. Also note that, e.g., the variables t and t' are frozen to values stored in the registers *tid* at different times. The freeze quantifier can be seen as a weak form of the standard first-order quantifiers [Henzinger 1990]. Since each register stores exactly one value at any time, it is irrelevant whether we quantify existentially or universally over a register's value. ◁

Let D—the *data domain*—be a nonempty set of values. Furthermore, let Σ be the set of the pairs (σ, ϱ), where σ is a function over P with $\sigma(p) \subseteq D^{\iota(p)}$ for $p \in P$ and ϱ is a function over R with $\varrho(r) \in D$ for $r \in R$. Intuitively, σ interprets the predicate symbols at the given time point and ϱ provides the values of the registers in R. MTL$^{\downarrow}$'s Boolean semantics is defined inductively over the formula structure. We define a function $\varphi \mapsto [\![w, i, v \models \varphi]\!] \in 2$, for a given timed word w over Σ, $i \in \mathbb{N}$, and a valuation $v : V \to D$. Let $w = (\tau_0, (\sigma_0, \varrho_0))\ (\tau_1, (\sigma_1, \varrho_1)) \ldots$.

$$[\![w, i, v \models \mathsf{t}]\!] := \mathsf{t}$$

$$[\![w, i, v \models p(\overline{x})]\!] := \begin{cases} \mathsf{t} & \text{if } v(\overline{x}) \in \sigma_i(p) \\ \mathsf{f} & \text{otherwise} \end{cases}$$

$$[\![w, i, v \models \downarrow^{r} x.\ \varphi]\!] := [\![w, i, v[x \mapsto \varrho_i(r)] \models \varphi]\!]$$

$$[\![w, i, v \models \neg\varphi]\!] := \neg [\![w, i, v \models \varphi]\!]$$

$$[\![w, i, v \models \varphi \vee \psi]\!] := [\![w, i, v \models \varphi]\!] \vee [\![w, i, v \models \psi]\!]$$

$$[\![w, i, v \models \bullet_I \varphi]\!] := i > 0 \wedge \tau_i - \tau_{i-1} \in I \wedge [\![w, i-1, v \models \varphi]\!]$$

$$[\![w, i, v \models \bigcirc_I \varphi]\!] := \tau_{i+1} - \tau_i \in I \wedge [\![w, i+1, v \models \varphi]\!]$$

$$[\![w, i, v \models \varphi\ \mathsf{S}_I\ \psi]\!] := \bigvee_{j \in \mathbb{N}, j \leq i} \left(\tau_i - \tau_j \in I \wedge [\![w, j, v \models \psi]\!] \wedge \bigwedge_{j < k \leq i} [\![w, k, v \models \varphi]\!]\right)$$

$$[\![w, i, v \models \varphi\ \mathsf{U}_I\ \psi]\!] := \bigvee_{j \in \mathbb{N}, j \geq i} \left(\tau_j - \tau_i \in I \wedge [\![w, j, v \models \psi]\!] \wedge \bigwedge_{i \leq k < j} [\![w, k, v \models \varphi]\!]\right)$$

Note that we abuse notation here and identify the logic's constant symbol t with the Boolean value t, and the connectives $\neg$ and $\vee$ with the corresponding logical operators. Furthermore, we use standard conventions, for example, $p(\overline{x})$ abbreviates $p(x_1, \ldots, x_{\iota(p)})$ and $v(\overline{x}) \in \sigma_i(p)$ abbreviates $\big(v(x_1), \ldots, v(x_{\iota(p)})\big) \in \sigma_i(p)$. Finally, note that the disjunction in the U_I case is infinite.

3 METRIC TEMPORAL LOGIC FOR MONITORING

In this section, we present a three-valued semantics for $\mathrm{MTL}^{\downarrow}$ that conservatively approximates the logic's standard Boolean semantics. Our new semantics is defined with monitoring in mind in that it accounts for knowledge gaps that arise during monitoring, which may be fully or partially filled later. We first introduce in Section 3.1 the models of our semantics, which support reasoning about incomplete traces. Afterwards, in Section 3.2, we present the semantics and establish basic properties about it in Section 3.3. We conclude by defining correctness requirements for monitoring in Section 3.4.

3.1 Observations

A monitor usually has only partial knowledge about the behavior of the system it monitors. For instance, for nonterminating systems, a monitor is only aware of a finite prefix of the system's behavior. Thus, when modeling this behavior as a timed word, the monitor only knows a finite prefix of this word. Moreover, when communication to the monitor is unreliable or delayed, the monitor may not even have the entire finite prefix, but only portions thereof. In the following, we introduce a notion of observations that supports reasoning based on partial information about the system behavior.

Throughout this section, we fix an alphabet Σ. We require that Σ is partially ordered and denote the partial order by $\sqsubset$. Intuitively, $a \sqsubset b$ means that a carries less information than b. Furthermore, we require that Σ has a least element a_0.

Definition 3.1. The set of *observations* $Obs(\Sigma)$ is inductively defined.

- The word $\big([0, \infty), a_0\big)$ of length 1 is in $Obs(\Sigma)$.
- If the word w is in $Obs(\Sigma)$, then the word obtained by applying one of the following transformations to w is in $Obs(\Sigma)$.
  - (T1) Some letter (I, a) of w, with $|I| > 1$, is replaced by the three-letter word
  $$\big(I \cap [0, \tau), a\big)\ \big(\{\tau\}, a\big)\ \big(I \cap (\tau, \infty), a\big)\,,$$
  where $\tau \in I$ and $\tau > 0$. If $\tau = 0$, then (I, a) is replaced by the two-letter word $\big(\{\tau\}, a\big)\ \big(I \cap (\tau, \infty), a\big)$.
  - (T2) Some letter (I, a) of w, with $|I| > 1$ and I bounded, is removed.
  - (T3) Some letter (I, a) of w, with $|I| = 1$, is replaced by (I, a') with $a \sqsubset a'$.

For an observation w of length $n \in \mathbb{N}$, let $pos(w) := \{0, \ldots, n-1\}$. We call $i \in pos(w)$ a *time point* in w if the interval I_i of the letter at position i in w is a singleton. In this case, the element of I_i is the *timestamp* of the time point i, denoted by $\mathrm{ts}_w(i)$.

Given the inductive definition of the set $Obs(\Sigma)$, the partial order over Σ naturally extends to a partial order on observations. We thereby obtain the following refinement relation on observations.

Definition 3.2. For $w, w' \in Obs(\Sigma)$, let $w \sqsubset_1 w'$ iff w' is obtained from w by one of the transformations (T1), (T2), or (T3). The observation w' *refines* the observation w if $w \sqsubseteq w'$, where $\sqsubseteq$ is the reflexive-transitive closure of $\sqsubset_1$.

Example 3.3. Recall the set of predicates symbols $P = \{trans\}$ and the set of registers $R = \{cid, tid, sum\}$ from Example 2.1. For brevity, we ignore here the rigid interpretations of the constant symbol 2000 and the binary predicate

symbol $\geq$. Furthermore, recall the data domain D that contains all customers and the positive integers. Let Σ be the alphabet consisting of the pairs (σ, ϱ) with $\sigma : P \nrightarrow 2^{D \times D \times D}$ and $\varrho : R \nrightarrow D$. Note that the partial orders on the two sets of partial functions extend to a partial order on Σ and that $([\,],[\,])$ is Σ's least element.

A monitor's knowledge can be represented by observations over Σ. A monitor's initial knowledge is represented by the observation $w_0 = \big([0,\infty),([\,],[\,])\big)$. Suppose that a transaction of \$99 with identifier 42 from *Alice* is executed at time 3.0. The monitor's initial knowledge w_0 is then updated by the transformations (T1) and (T3) to $w_1 = \big([0,3.0),([\,],[\,])\big)\,\big(\{3.0\},(\sigma,\varrho)\big)\,\big((3.0,\infty),([\,],[\,])\big)$, where $\sigma(\mathit{trans}) = \{(\mathit{Alice}, 42, 99)\}$ and $\varrho = [\mathit{cid} \mapsto \mathit{Alice}, \mathit{tid} \mapsto 42, \mathit{sum} \mapsto 99]$. Note that $w_0 \sqsubseteq w_1$.

If the monitor also receives the information that no action occurred in the interval $[0, 3.0)$, then its updated knowledge is represented by the observation $\big(\{3.0\},(\sigma,\varrho)\big)\,\big((3.0,\infty),([\,],[\,])\big)$, obtained from w_1 by the transformation (T2). The information that no action has occurred in an interval can be communicated explicitly or implicitly by the monitored system to the monitor, for instance, by attaching a sequence number to each action. See Section 7.2.1 for details. $\lhd$

We remark that the interval associated with the last letter of an observation is always unbounded. This reflects that a monitor is unaware of what it will observe in the future. More generally, a letter (I, a) of an observation with $|I| > 1$ represents a knowledge gap of the monitor. In particular, a is the alphabet's least element a_0, meaning that nothing is known about the interpretation of the predicate symbols and the register values during the time period I. Finally, note that according to Definition 3.1, knowledge gaps (I, a) can completely disappear (T2), or can be partially resolved by adding a new time point where the interval is split (T1), where (T3) can add additional knowledge to the new time point by replacing a with a letter that is larger with respect to the alphabet's partial order. For simplicity, we do not include a transformation in Definition 3.1 that allows one to shrink nonsingleton intervals, that is, a transformation that replaces a letter (I, a) with $|I| > 1$ by a letter (I', a) with $|I'| > 1$, $I' \subsetneq I$, and I' is unbounded if I is unbounded.

3.2 Three-valued Semantics

MTL$^{\downarrow}$'s models under the three-valued semantics are observations, which represent a monitor's partial knowledge about the system behavior at a given point in time. This is in contrast to the models for the standard Boolean semantics for MTL, which are timed words and capture the complete system behavior in the limit.

For defining MTL$^{\downarrow}$'s three-valued semantics, we fix a *data domain* D, which is a nonempty set of values with $\bot \notin D$. Furthermore, let Σ be the alphabet consisting of the letters (σ, ϱ), where σ and ϱ are partial functions, namely, $\sigma : P \nrightarrow \bigcup_{p \in P} 2^{D^{\iota(p)}}$ and $\varrho : R \nrightarrow D$. Note that Σ is partially ordered and its least element is $([\,],[\,])$. Analogous to the definition of MTL$^{\downarrow}$'s Boolean semantics in Section 2, we define the logic's three-valued semantics by a function $\varphi \mapsto [\![w, i, \nu \models\!\!\approx \varphi]\!] \in 3$, for a given observation $w \in \mathit{Obs}(\Sigma)$, $i \in \mathit{pos}(w)$, and a partial valuation $\nu : V \nrightarrow D$. We define this function inductively over the formula structure. In the following, we assume that $w = \big(I_0,(\sigma_0,\varrho_0)\big) \ldots \big(I_{n-1},(\sigma_{n-1},\varrho_{n-1})\big)$ and abuse notation by identifying the logic's constant symbol t with the Boolean value $\mathsf{t} \in 3$, and the connectives $\neg$ and $\vee$ with the corresponding three-valued logical operators in Table 1. The nontemporal cases are as expected.

$$[\![w, i, \nu \models\!\!\approx \mathsf{t}]\!] := \mathsf{t}$$

$$[\![w, i, \nu \models\!\!\approx p(\overline{x})]\!] := \begin{cases} \mathsf{t} & \text{if } \overline{x} \in \mathrm{def}(\nu),\ p \in \mathrm{def}(\sigma_i), \text{ and } \nu(\overline{x}) \in \sigma_i(p) \\ \mathsf{f} & \text{if } \overline{x} \in \mathrm{def}(\nu),\ p \in \mathrm{def}(\sigma_i), \text{ and } \nu(\overline{x}) \notin \sigma_i(p) \\ \bot & \text{otherwise} \end{cases}$$

$$[\![w, i, v \models\!\approx \downarrow^r x.\, \varphi]\!] := \begin{cases} [\![w, i, v[x \mapsto \varrho_i(r)] \models\!\approx \varphi]\!] & \text{if } x \in \mathrm{def}(\varrho_i) \\ [\![w, i, v[x \mapsto \bot] \models\!\approx \varphi]\!] & \text{otherwise} \end{cases}$$

$$[\![w, i, v \models\!\approx \neg\varphi]\!] := \neg[\![w, i, v \models\!\approx \varphi]\!]$$

$$[\![w, i, v \models\!\approx \varphi \vee \psi]\!] := [\![w, i, v \models\!\approx \varphi]\!] \vee [\![w, i, v \models\!\approx \psi]\!]$$

The temporal cases are less straightforward. In particular, the definition must account for letters in w where a nonsingleton interval represents knowledge gaps that may either disappear or may be replaced by multiple letters in a refinement. We make use of the auxiliary functions $\mathrm{tp}_w : pos(w) \to 3$ and $\mathrm{mc}_{w,I} : pos(w) \times pos(w) \to 3$, which are as follows for the observation w and an interval I.

$$\mathrm{tp}_w(i) := \begin{cases} \mathsf{t} & \text{if } |I_i| = 1 \\ \bot & \text{otherwise} \end{cases} \qquad \text{and} \qquad \mathrm{mc}_{w,I}(i,j) := \begin{cases} \mathsf{t} & \text{if } I_i - I_j \neq \emptyset \text{ and } I_i - I_j \subseteq I \\ \mathsf{f} & \text{if } (I_i - I_j) \cap I = \emptyset \\ \bot & \text{otherwise} \end{cases}$$

We use tp_w to check whether a position is a *time point* (hence, the name "tp"), and we use $\mathrm{mc}_{w,I}$ to check whether the *metric constraint* I of a temporal connective is valid or unsatisfiable between two positions in w (hence, the name "mc"). Note that if $\mathrm{mc}_{w,I}(i,j) = \bot$, then the metric constraint between the positions i and j could either be satisfied or violated, depending on some timestamps $\tau \in I_i$ and $\tau' \in I_j$.

The semantics of the temporal connectives S_I and U_I is defined as follows.

$$[\![w, i, v \models\!\approx \varphi\ \mathsf{S}_I\ \psi]\!] := \bigvee_{j \in pos(w), j \leq i} \Big(\mathrm{tp}_w(j) \wedge \mathrm{mc}_{w,I}(i,j) \wedge [\![w, j, v \models\!\approx \psi]\!] \wedge \bigwedge_{j < k \leq i} \big(\mathrm{tp}_w(k) \to [\![w, k, v \models\!\approx \varphi]\!]\big)\Big)$$

$$[\![w, i, v \models\!\approx \varphi\ \mathsf{U}_I\ \psi]\!] := \bigvee_{j \in pos(w), j \geq i} \Big(\mathrm{tp}_w(j) \wedge \mathrm{mc}_{w,I}(j,i) \wedge [\![w, j, v \models\!\approx \psi]\!] \wedge \bigwedge_{i \leq k < j} \big(\mathrm{tp}_w(k) \to [\![w, k, v \models\!\approx \varphi]\!]\big)\Big)$$

We comment on the definitions for $\varphi\ \mathsf{S}_I\ \psi$ and $\varphi\ \mathsf{U}_I\ \psi$. First, note that j ranges over so-called "anchor" positions and k ranges over so-called "continuation" positions. For a position j to be a "valid" anchor position, j must be a time point, which is the case when $\mathrm{tp}_w(j) = \mathsf{t}$. Otherwise, $\mathrm{tp}_w(j) = \bot$. Using the truth value f instead of $\bot$ would be incorrect since it is not yet known whether a refinement of w will contain a time point with a timestamp in I_j. Furthermore, note that the function $\mathrm{mc}_{w,I}$ returns $\bot$ if it is unknown in w whether the formula's metric constraint is always satisfied or never satisfied for the positions i and j. Finally, suppose that a position k between j and i is an "invalid" continuation position, that is, φ's truth value at k is f. If the interval I_k is not a singleton, $\mathrm{tp}_w(k)$ "downgrades" this truth value to $\bot$, since it will be irrelevant in refinements of w that do not contain any time points with a timestamp in I_k.

Finally, we define the semantics of the temporal connectives $\bullet_I$ and $\bigcirc_I$ as

$$[\![w, i, v \models\!\approx \bullet_I\, \varphi]\!] := c_0 \vee c_{-1} \vee c_{-2} \qquad \text{and} \qquad [\![w, i, v \models\!\approx \bigcirc_I\, \varphi]\!] := c_0 \vee c_1 \vee c_2$$

with

$$c_k := \begin{cases} \mathrm{mc}_{w,I}(i,i) \wedge [\![w,i,v \mid\approx \varphi]\!] \wedge \neg \mathrm{tp}_w(i) & \text{if } k = 0 \text{ and } I \neq \{0\}, \\ \mathrm{mc}_{w,I}(i,i-1) \wedge [\![w,i-1,v \mid\approx \varphi]\!] \wedge \mathrm{tp}_w(i-1) \wedge \mathrm{tp}_w(i) & \text{if } k = -1 \text{ and } i \geq 1, \\ \mathrm{mc}_{w,I}(i+1,i) \wedge [\![w,i+1,v \mid\approx \varphi]\!] \wedge \mathrm{tp}_w(i+1) \wedge \mathrm{tp}_w(i) & \text{if } k = 1 \text{ and } i < n-1, \\ \mathrm{mc}_{w,I}(i,i-2) \wedge [\![w,i-2,v \mid\approx \varphi]\!] \wedge \neg \mathrm{tp}_w(i-1) & \text{if } k = -2 \text{ and } i \geq 2, \\ \mathrm{mc}_{w,I}(i+2,i) \wedge [\![w,i+2,v \mid\approx \varphi]\!] \wedge \neg \mathrm{tp}_w(i+1) & \text{if } k = 2 \text{ and } i < n-2, \\ \mathsf{f} & \text{otherwise.} \end{cases}$$

We comment on the definition for $\bigcirc_I \varphi$ with $i \leq n-2$; the other cases are analogous or restricted cases of this one. One might expect that the conjunct c_1 is already sufficient. However, having only c_1 could result in the wrong truth value f for $\bigcirc_I \varphi$ at i when $[\![w,i+1,v \mid\approx \varphi]\!] = \mathsf{f}$. If, for example, $|I_i| > 1$ then it is still possible to satisfy $\bigcirc_I \varphi$ when refining the observation w at I_i. A refinement of w may consist of two time points with the timestamps τ and τ' in I_i, where $\tau' - \tau \in I$ and φ is true at the time point with timestamp τ'. The conjunct c_0 takes care of such a refinement at i. The conjunct c_2 is necessary when $|I_{i+1}| > 1$. In this case i and $i+2$ are time points in w. The observation w may be refined by removing the letter at position $i+1$, resulting in an observation where w's letter at position $i+2$ is the successor of w's letter at position i. Note that c_0 and c_2 can be f or $\bot$ but never t because of the negative tp_w literals occurring in c_0 and c_2. Furthermore, again because of the tp_w literals, we have that $c_0 = c_2 = \mathsf{f}$ whenever $c_1 = \mathsf{t}$. Finally, observe that $\mathrm{mc}_{w,\{0\}}(i,i) \neq \mathsf{f}$. However, the metric constraint $\{0\}$ is only satisfiable for time points that have equal timestamps and we require that timestamps are strictly increasing. Hence, the additional constraint $I \neq \{0\}$ is needed when $k = 0$.

Observe that it may be the case that $[\![w,i,v \mid\approx \varphi]\!] \in 2$ when i is not a time point in w. A trivial example is when $\varphi = \mathsf{t}$. In a refinement of w, it might turn out that there are no time points with timestamps in I_i, and hence a monitor should not output a verdict for the specification φ at position i in w. We address this artifact by downgrading (with respect to the partial order $\prec$) a Boolean truth value $[\![w,i,v \mid\approx \varphi]\!]$ to $\bot$ when i is not a time point. To this end, we introduce the following variant of the semantics.

Definition 3.4. For a formula φ, an observation w, $\tau \in \mathbb{Q}_{\geq 0}$, and a partial valuation v, we define

$$[w,\tau,v \mid\approx \varphi] := \begin{cases} [\![w,i,v \mid\approx \varphi]\!] & \text{if } \tau \text{ is the timestamp of some time point } i \in \mathit{pos}(w), \text{ and} \\ \bot & \text{otherwise.} \end{cases}$$

3.3 Properties

Our first theorem shows that MTL$^{\downarrow}$'s three-valued semantics conservatively approximates its standard Boolean semantics. Intuitively speaking, if a formula φ evaluates to a Boolean value for an observation at time $\tau \in \mathbb{Q}_{\geq 0}$, then φ has the same Boolean value at time τ for any timed word that refines the observation. To state the theorem, we need the following definitions. A timed word w' *refines* an observation w, written $w \sqsubseteq w'$ for short, if for every $j \in \mathbb{N}$, there is some $i \in \mathit{pos}(w)$, such that $\tau_j \in I_i$, $\sigma_i \sqsubseteq \sigma'_j$, and $\varrho_i \sqsubseteq \varrho'_j$, where $(I_\ell, (\sigma_\ell, \varrho_\ell))$ and $(\tau_k, (\sigma'_k, \varrho'_k))$, for $\ell \in \mathit{pos}(w)$ and $k \in \mathbb{N}$, are the letters of w and w', respectively. Furthermore, similar to Definition 3.4, we define for $\tau \in \mathbb{Q}_{\geq 0}$, a timed word w, a valuation v, and a formula φ,

$$[w,\tau,v \models \varphi] := \begin{cases} [\![w,j,v \models \varphi]\!] & \text{if the } j\text{th letter of } w \text{ is } (\tau,(\sigma,\varrho)), \text{ and} \\ \bot & \text{otherwise.} \end{cases}$$

Theorem 3.5. *Let φ be a formula, μ a partial valuation, ν a total valuation, u an observation, v a timed word, and $\tau \in \mathbb{Q}_{\geq 0}$. If $u \sqsubseteq v$ and $\mu \sqsubseteq \nu$ then $[u, \tau, \mu \models_{\approx} \varphi] \preceq [v, \tau, \nu \models \varphi]$.*

Proof. Let $(I_i, (\sigma_i, \varrho_i))$ and $(\tau_j, (\sigma'_j, \varrho'_j))$, for $i \in pos(u)$ and $j \in \mathbb{N}$, be the letters of u and v, respectively. Since $u \sqsubseteq v$, there is a function $\pi : \mathbb{N} \to pos(u)$ such that (R1) $\tau_j \in I_{\pi(j)}$, (R2) $\sigma_{\pi(j)} \sqsubseteq \sigma'_j$, and (R3) $\varrho_{\pi(j)} \sqsubseteq \varrho'_j$, for every $j \in \mathbb{N}$. It is easy to see that π is monotonic.

We prove by structural induction on φ that for every $i' \in \mathbb{N}$ and partial valuations μ and ν with $\mathrm{def}(\nu) = V$ and $\mu \sqsubseteq \nu$, it holds that $[\![u, \pi(i'), \mu \models_{\approx} \varphi]\!] \preceq [\![v, i', \nu \models \varphi]\!]$. The theorem easily follows from this statement. Let $i' \in \mathbb{N}$, and let μ and ν be partial valuations with $\mathrm{def}(\nu) = V$ and $\mu \sqsubseteq \nu$. Furthermore, let $i = \pi(i')$. Note that the statement clearly holds for $[\![u, i, \mu \models_{\approx} \varphi]\!] = \bot$. Hence, it suffices to show that $[\![u, i, \mu \models_{\approx} \varphi]\!] = [\![v, i', \nu \models \varphi]\!]$, provided that $[\![u, i, \mu \models_{\approx} \varphi]\!] \in 2$.

Base cases. The case $\varphi = \mathrm{t}$ is trivial. Consider the case $\varphi = p(\overline{x})$, for some $p \in P$. As $[\![u, i, \nu \models_{\approx} p(\overline{x})]\!] \in 2$, it holds that $\overline{x} \in \mathrm{def}(\nu)$ and $p \in \mathrm{def}(\sigma_i)$. It follows from the theorem's premise that $\mu(\overline{x}) = \nu(\overline{x})$ and from (R2) that $\sigma_i(p) = \sigma'_{i'}(p)$. Thus $[\![u, i, \nu \models_{\approx} p(\overline{x})]\!] = [\![v, i', \nu \models p(\overline{x})]\!]$.

Inductive cases. The cases where φ is of the form $\neg\alpha$ or $\alpha \vee \beta$ are straightforward and are omitted. We also omit the cases for $\bullet_I\, \alpha$, $\bigcirc_I\, \alpha$, and $\alpha\ \mathsf{S}_I\ \beta$, since they are similar to the case $\alpha\ \mathsf{U}_I\ \beta$.

First, assume that φ is of the form $\downarrow^r x.\ \psi$. Let $\eta = \mu[x \mapsto \varrho_i(r)]$ if $r \in \mathrm{def}(\varrho_i)$, and $\eta = \mu[x \mapsto \bot]$ otherwise. Similarly, let $\eta' = \nu[x \mapsto \varrho'_{i'}(r)]$ if $r \in \mathrm{def}(\varrho'_{i'})$, and $\eta' = \mu[x \mapsto \bot]$ otherwise. By (R3), we have that if $r \in \mathrm{def}(\varrho_i)$, then $r \in \mathrm{def}(\varrho'_{i'})$ and $\varrho_i(r) = \varrho'_{i'}(r)$, and thus $\eta(x) = \eta'(x)$. Furthermore, if $r \notin \mathrm{def}(\varrho_i)$, then $x \notin \mathrm{def}(\eta)$. Hence $\eta \sqsubseteq \eta'$. It follows from the induction hypothesis that $[\![u, i, \eta \models_{\approx} \psi]\!] \preceq [\![v, i', \eta' \models \psi]\!]$ and therefore $[\![u, i, \mu \models_{\approx} \varphi]\!] = [\![v, i', \nu \models \varphi]\!]$.

Assume that φ is of the form $\alpha\ \mathsf{U}_I\ \beta$. We consider first the case $[\![u, i, \mu \models_{\approx} \alpha\ \mathsf{U}_I\ \beta]\!] = \mathrm{t}$. By definition, there is some $j \in pos(u)$ with $j \geq i$ such that $\mathrm{tp}_u(j) = \mathrm{t}$, $\mathrm{mc}_{u,I}(i, j) = \mathrm{t}$, $[\![u, j, \mu \models_{\approx} \beta]\!] = \mathrm{t}$, and $\mathrm{tp}_u(k) \to [\![u, k, \mu \models_{\approx} \alpha]\!] = \mathrm{t}$, for all k with $i \leq k < j$. As j is a time point in u, $\pi(j') = j$, for some $j' \in \mathbb{N}$. From (R1) we have that $\tau_{j'} = \mathrm{ts}_u(j)$. As $\mathrm{mc}_{u,I}(i, j) = \mathrm{t}$ and $I_j = \{\tau_{j'}\}$, we have that $\tau_{j'} - \tau \in I$, for all $\tau \in I_i$. From (R1), we have that $\tau_{i'} \in I_i$. Thus $\tau_{j'} - \tau_{i'} \in I$ (I1). From the induction hypothesis, $[\![u, j, \mu \models_{\approx} \beta]\!] \preceq [\![v, j', \nu \models \beta]\!]$. Hence $[\![v, j', \nu \models \beta]\!] = \mathrm{t}$ (I2). We also have that $[\![u, \pi(k'), \mu \models_{\approx} \alpha]\!] \preceq [\![v, k', \nu \models \alpha]\!]$, for any $k' \in \mathbb{N}$. Let $k' \in \mathbb{N}$ such that $i' \leq k' < j'$, and let $k = \pi(k')$. By the monotonicity of π we have that $i \leq k \leq j$. Since j is a time point in u we also have that $k < j$. As $\mathrm{tp}_u(k) \to [\![u, k, \mu \models_{\approx} \alpha]\!] = \mathrm{t}$ and tp_u is never f by definition, we have that $[\![u, k, \mu \models_{\approx} \alpha]\!] = \mathrm{t}$. Then $[\![v, k', \mu \models \alpha]\!] = \mathrm{t}$ (I3). Summing up, from (I1), (I2), (I3), and as k' was chosen arbitrarily, we obtain that $[\![v, i', \nu \models \alpha\ \mathsf{U}_I\ \beta]\!] = \mathrm{t}$.

The case $[\![u, i, \mu \models_{\approx} \alpha\ \mathsf{U}_I\ \beta]\!] = \mathrm{f}$ is as follows. Note that each disjunct in the definition of $[\![u, i, \mu \models_{\approx} \alpha\ \mathsf{U}_I\ \beta]\!]$ is f. We fix an arbitrary $j' \geq i'$ and let $j = \pi(j')$. It holds that $\mathrm{tp}_u(j) \wedge \mathrm{mc}_{u,I}(j, i) \wedge [\![u, j, \mu \models_{\approx} \beta]\!] \wedge \bigwedge_{i \leq k < j} (\mathrm{tp}_u(k) \to [\![u, k, \mu \models_{\approx} \alpha]\!]) = \mathrm{f}$. Since $\mathrm{tp}_u(j) \neq \mathrm{f}$, one of the remaining conjuncts must be f.

(1) If $\mathrm{mc}_{u,I}(i, j) = \mathrm{f}$, then $\tau' - \tau'' \notin I$, for all $\tau'' \in I_i$ and $\tau' \in I_j$. From (R1), $\tau_{i'} \in I_i$ and $\tau_{j'} \in I_j$, it follows that $\tau_{j'} - \tau_{i'} \notin I$.
(2) If $[\![u, j, \mu \models_{\approx} \beta]\!] = \mathrm{f}$, then $[\![v, j', \nu \models \beta]\!] = \mathrm{f}$, by the induction hypothesis.
(3) If $\mathrm{tp}_u(k) \to [\![u, k, \mu \models_{\approx} \alpha]\!] = \mathrm{f}$, for some k with $i \leq k < j$, then $\mathrm{tp}_u(k) = \mathrm{t}$ and $[\![u, k, \mu \models_{\approx} \alpha]\!] = \mathrm{f}$. It follows as before that there is a k' with $i' \leq k' < j'$ such that $[\![v, k', \nu \models \alpha]\!] = \mathrm{f}$.

We have thus obtained that either $\tau_{i'} - \tau_{j'} \notin I$ or one of the conjuncts of $[\![v, j', \nu \models \beta]\!] \wedge \bigwedge_{i' \leq k' < j'} [\![v, k', \nu \models \alpha]\!]$ is f. In other words, if j' is such that $\tau_{j'} - \tau_{i'} \in I$, then $[\![v, j', \nu \models \beta]\!] \wedge \bigwedge_{i' \leq k' < j'} [\![v, k', \nu \models \alpha]\!] = \mathrm{f}$. As j' was chosen arbitrarily, we conclude that $[\![v, i', \nu \models \alpha\ \mathsf{U}_I\ \beta]\!] = \mathrm{f}$. □

The next theorem states that MTL$^{\downarrow}$'s three-valued semantics is monotonic in $\sqsubseteq$ (on observations and partial valuations) and $\preceq$ (on truth values). This property is crucial for monitoring since it guarantees that a verdict output for an observation stays valid for refined observations.

Theorem 3.6. *Let φ be a formula, μ and ν partial valuations, u and v observations, and $\tau \in \mathbb{Q}_{\geq 0}$. If $u \sqsubseteq v$ and $\mu \sqsubseteq \nu$ then $[u, \tau, \mu \models\!\!\approx \varphi] \preceq [v, \tau, \nu \models\!\!\approx \varphi]$.*

Proof. The proof is similar to that of Theorem 3.5 and details are thus omitted. We just note that we make use of the following properties (R1′), (R2′), and (R3′), which correspond to the ones used in the proof of Theorem 3.5.

Let w and w' be observations with letters $\big(I_i, (\sigma_i, \varrho_i)\big)$ and respectively $\big(I'_j, (\sigma'_j, \varrho'_j)\big)$, for $i \in pos(w)$ and $j \in pos(w')$. We claim that if $w \sqsubseteq w'$ then there is a monotonic function $\pi : pos(w') \to pos(w)$ with the following properties.

(R1′) $I'_j \subseteq I_{\pi(j)}$, for all $j \in pos(w')$.
(R2′) $\sigma_{\pi(j)} \sqsubseteq \sigma'_j$, for all $j \in pos(w')$.
(R3′) $\varrho_{\pi(j)} \sqsubseteq \varrho'_j$, for all $j \in pos(w')$.

If $w = w'$ then take π to be the identity. If w' is obtained from w using one of the transformations, that is, if $w \sqsubset_1 w'$, then, for each transformation it is easy to construct a function π that has the stated properties. If $w \sqsubsetneq w'$, then there is a sequence $(w_i)_{0 \leq i \leq n}$ of observations, with $n > 1$, such that $w = w_0 \sqsubset_1 w_1 \sqsubset_1 \cdots \sqsubset_1 w_n = w'$. From the previous observation, there is a sequence of functions $\pi_i : pos(w_i) \to pos(w_{i-1})$, with $1 \leq i \leq n$, each having the stated properties. The functions' composition $\pi = \pi_1 \circ \cdots \circ \pi_n$ also has these properties. □

We next investigate the decision problem that underlies monitoring. Note that we do not require that the interpretations of the predicate symbols are finite relations. However, for monitoring, the relations must be decidable and a monitor needs an algorithm for performing membership checks. For the following theorem, we assume that the membership of a tuple in a predicate symbol's interpretation at a time point can be checked in PSPACE.

Theorem 3.7. *For a formula φ, an observation w, a partial valuation ν, $\tau \in \mathbb{Q}_{\geq 0}$, and a truth value $b \in 2$, the problem of whether $[w, \tau, \nu \models\!\!\approx \varphi]$ equals b is* PSPACE*-complete.*

Proof. We first show that the problem is PSPACE-hard by reducing the satisfiability problem for quantified Boolean logic (QBL) to it. Let α be a closed QBL formula over the propositions $p_1, \ldots, p_n$. We define the set P of predicate symbols as $\{P_1, \ldots, P_n\}$, where each predicate symbol has arity 1. Moreover, let $R := \{r\}$ and $D := \{0, 1\}$, and let w be the observation

$$\big(\{0\}, \sigma, \varrho_0\big)\ \big(\{1\}, (\sigma, \varrho_1)\big)\ \big(\{3\}, ([\,], [\,])\big)\ \big((3, \infty), ([\,], [\,])\big)\,,$$

with $\sigma(P_i) = \{1\}$, for each $i \in \{1, \ldots, n\}$, and $\varrho_i(r) = i$, for $i \in \{0, 1\}$. Finally, we translate the QBL formula α to an MTL$^{\downarrow}$ formula α^* as follows.

$$p_i^* := P_i(x_i) \quad (\neg\beta)^* := \neg\beta^* \quad (\beta \vee \gamma)^* := \beta^* \vee \gamma^* \quad (\exists p_i\,.\,\beta)^* := \blacklozenge\, \Diamond_{[0,1]} \downarrow x_i\,.\,\beta^* \quad (\forall p_i\,.\,\beta)^* := \blacksquare\, \Box_{[0,1]} \downarrow x_i\,.\,\beta^*$$

It is easy to see that α is satisfiable iff $[w, 0, [\,] \models\!\!\approx \alpha^*] = \mathrm{t}$.

We only sketch the problem's membership in PSPACE. Note that w is finite. If there is no time point in w with timestamp τ, then $[w, \tau, \nu \models\!\!\approx \varphi] = \bot$. Suppose that $i \in pos(w)$ is a time point in w with timestamp τ. A computation of φ's truth value at position i can be easily obtained from the inductive definition of the satisfaction relation $\models\!\!\approx$. Note, however, that the space consumed by naively unfolding the semantic definitions would in general not be polynomially bounded. One reason is that subformulas may occur multiple times in the unfolding for different time points and valuations.

Instead, we must carry out this computation by a depth-first traversal when unfolding the semantic definitions to stay in PSPACE. Furthermore, note that our additional assumption on the membership checks allows us to determine in PSPACE the truth value of an atomic formula at a time point. □

In a propositional setting, the corresponding decision problem can be solved in polynomial time using dynamic programming, where the truth values at the positions of an observation are propagated up the formula structure. Note that the truth value of a proposition at a position is given by the observation's letter at that position. This is in contrast to MTL$^{\downarrow}$, where atomic formulas can have free variables and their truth values at the positions in an observation w may depend on the data values stored in the registers and frozen to these variables at different time points of w. Before truth values are propagated up, the bindings of variables to data values must be propagated down.

3.4 Monitoring Correctness Requirements

A monitor for a specification iteratively receives information about the system behavior. Abstractly speaking, the monitor's input is an infinite sequence $(in_i)_{i\in\mathbb{N}}$, where in_i describes a part of the system behavior and is received by the monitor in its ith iteration. The monitor's output is an infinite sequence $(out_i)_{i\in\mathbb{N}}$, where out_i is the output in iteration i describing when the monitor's specification is satisfied or violated. In the following, we concretize a monitor's input and output for our setting and define correctness requirements for monitoring. Note that we assume that a monitor never terminates and that it infinitely often receives information about the system behavior. This assumption is invalid if, for instance, the system observed by the monitor ever terminates. Nevertheless, we make this assumption to simplify matters and it is easy to adapt our definitions and results to the general case.

We first turn to a monitor's input, which is a sequence of observations $(w_i)_{i\in\mathbb{N}}$. That is, we view the observation w_i as the input to the monitor at iteration $i \in \mathbb{N}$. In practice, a monitor would receive at iteration $i > 0$ a message that describes just the differences between w_{i-1} and w_i. Furthermore, note that the w_is can be understood as abstract descriptions of the monitor's state over time, representing the monitor's knowledge about the system behavior, where w_0 represents the monitor's initial knowledge. Also note that if the timed word v is the system behavior in the limit, then $w_i \sqsubseteq v$, for all $i \in \mathbb{N}$, assuming that components do not send bogus messages. However, for every $i \in \mathbb{N}$, there are infinitely many timed words u with $w_i \sqsubseteq u$. Since messages sent to the monitor can be lost, it can even be the case that there is a timed word u with $u \neq v$ and $w_i \sqsubseteq u$, for all $i \in \mathbb{N}$.

Definition 3.8. The infinite sequence $\bar{w} = (w_i)_{i\in\mathbb{N}}$ of observations is *valid* if $w_0 = \big([0,\infty), ([\,], [\,])\big)$ and $w_i \sqsubsetneq w_{i+1}$, for all $i \in \mathbb{N}$.

We turn to a monitor's output. Based on the input $(w_i)_{i\in\mathbb{N}}$, the monitor outputs in each iteration $i \in \mathbb{N}$ a set V_i of *verdicts*, which is a finite set of pairs (τ, b) with $\tau \in \mathbb{Q}_{\geq 0}$ and $b \in 2$. Intuitively, τ is the time at which the specification has the Boolean value b.

Definition 3.9. Let φ be a closed formula, $\bar{w} = (w_i)_{i\in\mathbb{N}}$ a valid observation sequence, and $\bar{V} = (V_i)_{i\in\mathbb{N}}$ a sequence of verdict sets.

(i) $\bar{V}$ is *observationally sound* for $\bar{w}$ and φ if for all partial valuations ν and $i \in \mathbb{N}$, whenever $(\tau, b) \in V_i$ then $[w_i, \tau, \nu \models\!\!\approx \varphi] = b$.

(ii) $\bar{V}$ is *observationally complete* for $\bar{w}$ and φ if for all partial valuations ν, $i \in \mathbb{N}$, and $\tau \in \mathbb{Q}_{\geq 0}$, if $[w_i, \tau, \nu \models\!\!\approx \varphi] \in 2$ then $(\tau, b) \in \bigcup_{j\leq i} V_j$, for some $b \in 2$.

We say that a monitor is *observationally sound* if for all valid observation sequences $\bar{w}$ and closed formulas φ, its sequence of verdict sets is observationally sound for $\bar{w}$ and φ. The definition of a monitor being *observationally complete* is analogous.

It follows from Theorem 3.7 that monitors for $\mathrm{MTL}^{\downarrow}$ exist that are both observationally sound and complete. In Sections 4 and 5, we present such monitoring algorithms in detail. In the remainder of this section, we relate the correctness requirements from Definition 3.9 to requirements that demand that a monitor outputs a verdict as soon as the specification has the same Boolean value on every extension of the monitor's current knowledge. Such requirements are stronger and achieving them can be hard or even impossible for nontrivial specification languages. In particular, we show that monitors satisfying such a requirement do not exists for $\mathrm{MTL}^{\downarrow}$. We start with an example that illustrates the differences on the verdicts for monitoring.

Example 3.10. Consider the formula $\varphi = \Box(p \wedge \Diamond \neg p)$. Note that under the classical Boolean semantics, φ is logically equivalent to f, however not under the three-valued semantics. For example, $[\![w, 0, v \mid\approx \varphi]\!] = \bot$, for $w = \big([0,\infty), ([\,], [\,])\big)$ and any valuation v. Given a valid observation sequence $\bar{w}$, an observationally sound and complete monitor for $\bar{w}$ and φ first outputs the verdict $(0, \mathrm{f})$ for the minimal i such that w_i contains a letter that assigns p to false. In contrast, a sound and complete monitor for the classical Boolean semantics (see Definition 3.11 below) must immediately output the verdict f. ◁

For an observation w, we define $U_w := \{v \mid v \text{ a timed word with } w \sqsubseteq v\}$. Intuitively, U_w contains the timed words that are compatible with the reported system behavior that a monitor received so far, represented by w.

Definition 3.11. Let φ be a closed formula, $\bar{w}$ a valid observation sequence, and $\bar{V}$ a sequence of verdict sets.

(i) $\bar{V}$ is *sound* for $\bar{w}$ and φ if for all valuations v and $i \in \mathbb{N}$, whenever $(\tau, b) \in V_i$ then $\bigwedge_{v \in U_{w_i}} [v, \tau, v \models \varphi] = b$, that is, the meet $\wedge$ of the truth values in the lower semilattice $(3, \prec)$ is b.

(ii) $\bar{V}$ is *complete* for $\bar{w}$ and φ if for all valuations v, $i \in \mathbb{N}$, and $\tau \in \mathbb{Q}_{\geq 0}$, whenever $\bigwedge_{v \in U_{w_i}} [v, \tau, v \models \varphi] \in 2$ then $(\tau, b) \in \bigcup_{j \leq i} V_j$, for some $b \in 2$.

We say that a monitor is *sound* if for all valid observation sequences $\bar{w}$ and closed formulas φ, its sequence of verdict sets is sound for $\bar{w}$ and φ. The definition of a monitor being *complete* is analogous.

With the help of Theorem 3.5 we prove that the completeness requirement from Definition 3.9 is indeed a weaker notion than the completeness requirement from Definition 3.11, while the soundness requirement from Definition 3.11 offers the same correctness guarantees as the one from Definition 3.9.

Theorem 3.12. *Let M be a monitor.*

(a) If M is observationally sound, then M is sound.

(b) If M is complete, then M is observationally complete.

Proof. Let $\bar{V}$ be the sequence of verdict sets that M iteratively outputs for φ and $\bar{w}$.

We first prove (a). Assume that M is observationally sound. Let v be a total valuation, $i \in \mathbb{N}$, $\tau \in \mathbb{Q}_{\geq 0}$, and $b \in 2$ such that $(\tau, b) \in V_i$. Then, by definition, $[w_i, \tau, v \mid\approx \varphi] = b$. For $v \in U_{w_i}$, we have that $w_i \sqsubseteq v$. By Theorem 3.5, we obtain that $[v, \tau, v \models \varphi] = b$. It follows that $\bigwedge_{v \in U_{w_i}} [v, \tau, v \models \varphi] = b$. We conclude that M is sound.

It remains to prove (b). Assume that M is complete. Let v be a partial valuation, $i \in \mathbb{N}$, and $\tau \in \mathbb{Q}_{\geq 0}$ such that $[w_i, \tau, v \models \varphi] = b'$, for some $b' \in 2$. Let v' be a total valuation with $v \sqsubseteq v'$ and $v \in U_{w_i}$. As $w_i \sqsubseteq v$, we obtain from Theorem 3.5 that $[v, \tau, v' \models \varphi] = b'$. As v was chosen arbitrarily, we get $\bigwedge_{v \in U_{w_i}} [v, \tau, v' \models \varphi] = b'$. From

M's completeness, it follows that there are $b \in 2$ and $j \in \mathbb{N}$ with $j \leq i$ such that $(\tau, b) \in V_j$. We conclude that M is observationally complete. □

The correctness requirements in Definition 3.11 are related to the use of a three-valued "runtime-verification" semantics for a specification language as introduced by Bauer et al. [2011] for LTL and adopted by other runtime-verification approaches, for example, the one by Bauer et al. [2015]. Both a sound and complete monitor, and a monitor implementing the three-valued "runtime-verification" semantics output a verdict as soon as the specification has the same Boolean value on every extension of the monitor's current knowledge. However, as we explain next, efficient monitors can be hard to achieve or may not even exist for nontrivial specification languages.

Remark 3.13. Having a sound and complete monitor M for a specification language is at least as hard as checking satisfiability for this language. For instance, we can use a sound and complete monitor M to check satisfiability for $\text{MTL}^{\downarrow}$ as follows. We run M for the closed formula φ whose satisfiability we want to check. We refine M's initial knowledge by the transformations (T1) and (T2) and add the first time point with the timestamp 0.0. The formula φ is unsatisfiable under the standard Boolean semantics iff M's verdict set V_1 contains $(0.0, \mathrm{f})$. Already MTL with the standard Boolean semantics is undecidable [Ouaknine and Worrell 2006] and many of its nontrivial decidable fragments have a high complexity. Recall that the satisfiability problem for LTL is PSPACE-complete [Sistla and Clarke 1985].

Some monitoring approaches try to compensate for this complexity burden with a preprocessing step. For instance, Bauer et al. [2011] translates an LTL formula into an automaton prior to monitoring. The resulting automaton can be directly used for sound and complete monitoring in environments where messages are neither delayed nor lost. However, there are no obvious extensions that handle out-of-order message delivery. Furthermore, not every specification language has such a corresponding automaton model and, for those where translations are known, the automaton construction can be very costly. For LTL, the size of the automaton is already in the worst case doubly exponential in the size of the formula [Bauer et al. 2011]. ◁

4 MONITORING IN THE PROPOSITIONAL SETTING

In this section, we present an observationally sound and complete online algorithm for MTL. We extend the algorithm in the next section to $\text{MTL}^{\downarrow}$, where we also provide the proof details. To support scalable monitoring, the verdict computation is incremental in that the results from previous computations are reused whenever observations are refined by the transformations (T1), (T2), and (T3) from Definition 3.1. We start with the algorithm's main procedure (Section 4.1). Afterwards, we describe the state the algorithm maintains (Section 4.2) and further algorithmic details (Section 4.3).

4.1 Main Procedure

The pseudocode of the monitor's top-level procedure is shown in Listing 1. In a nutshell, after the monitor initializes its state, it enters a nonterminating loop. In each loop iteration, the monitor receives a message, updates its state according to the information extracted from the message, and outputs the computed verdicts. Recall from Section 3.4 that each message received describes the "delta" between two subsequent observations in a valid observation sequence $(w_i)_{i \in \mathbb{N}}$. The message format and therefore how the monitor obtains the necessary information from a message and its current state are system dependent. A possible realization is given in Section 7.

We provide a brief description of the procedures used by the monitor's top-level procedure. The procedure Init initializes the monitor's state; see Section 4.2 for details. The procedure ReceiveMessage receives a message, for instance, over a channel or from a log file. The procedure UpdateKnowledge updates the monitor's knowledge about

```
procedure MonitorMTL(φ)
  Init(φ)
  loop
    m := ReceiveMessage()
    ts := UpdateKnowledge(m)
    foreach t with t in ts do
      case (T1): J, τ := DeltaT1(t)
                 AddTimePoint(φ, J, τ)
      case (T2): K := DeltaT2(t)
                 RemoveInterval(K)
      case (T3): τ, σ := DeltaT3(t)
                 foreach p with p ∈ def(σ) do
                   PropagateTruthValue(p, {τ}, σ(p))
```

Listing 1. The monitor's main loop for MTL.

the system behavior. This procedure also returns a list of the transformations that transform the observation w_{i-1} into the observation w_i in the ith iteration. The monitor uses the procedures DeltaT1, DeltaT2, and DeltaT3 to learn how the observation is updated. Concretely, DeltaT1 returns the timestamp τ of a new time point and the interval J that is split at τ. DeltaT2 returns the interval K of the letter that is removed from the observation. DeltaT3 returns the Boolean values $\sigma(p)$ of the newly assigned propositions $p \in \mathrm{def}(\sigma)$ at the time point with the timestamp τ. The procedures AddTimePoint, RemoveInterval, and PropagateTruthValue are central to the monitor. They update the monitor's state. For instance, PropagateTruthValue propagates the Boolean values of newly assigned propositions. Section 4.3 provides algorithmic details for these three procedures.

Before we proceed, we introduce the following conventions that we use in the remainder of this section. Let φ be the MTL formula that is monitored with propositions in P. The letter I ranges over the metric constraints of the temporal connectives that occur in φ. The letters α, β, and γ range over elements in $sub(\varphi)$. Furthermore, let w be an observation. It ranges over the elements in the valid observation sequence $(w_i)_{i\in\mathbb{N}}$. The letters J, K, and H range over the intervals that occur in letters of w. The lower case letters j, k, and h are the indexes of the letters in w with the intervals J, K, and H, respectively. We also simplify notation. We omit the partial valuation v in $[\![w, i, v \mid\approx \gamma]\!]$, that is, we only write $[\![w, i \mid\approx \gamma]\!]$. Note that v is irrelevant for MTL. We also assume that φ is not an atomic formula and subformulas of φ are pairwise distinct. Both assumptions are without loss of generality. For example, the second one is met when representing formulas as abstract syntax trees.

4.2 Monitor State

4.2.1 Reduction to Propositional Logic. At the core of the monitor is a mapping of MTL's three-valued semantics into propositional logic with the standard two-valued semantics. From a high-level perspective, the monitor's state comprises a representation of propositional formulas, which the monitor refines and simplifies whenever it receives information about the system behavior. For readability, we start with a variant of these propositional formulas that is close to the definition of MTL's three-valued semantics.

The propositional formula $\Phi_w^{\gamma,J}$ over propositions of the form α^K, tp^K, and $mc_I^{H,K}$ is defined as follows. Its inductive definition follows the definition of MTL's three-valued semantics in Section 3.2, where the propositions tp^K and $mc_I^{H,K}$

take the role of the corresponding functions.

$$\Phi_w^{\gamma,J} := \begin{cases} \mathsf{t} & \text{if } \gamma = \mathsf{t} \\ p^J & \text{if } \gamma = p \text{ with } p \in P \\ \neg\alpha^J & \text{if } \gamma = \neg\alpha \\ \alpha^J \vee \beta^J & \text{if } \gamma = \alpha \vee \beta \\ \bigvee_{K \leq J} \left(tp^K \wedge mc_I^{J,K} \wedge \beta^K \wedge \bigwedge_{K < H \leq J} (tp^H \rightarrow \alpha^H) \right) & \text{if } \gamma = \alpha \, \mathsf{S}_I \, \beta \\ \bigvee_{K \geq J} \left(tp^K \wedge mc_I^{K,J} \wedge \beta^K \wedge \bigwedge_{J \leq H < K} (tp^H \rightarrow \alpha^H) \right) & \text{if } \gamma = \alpha \, \mathsf{U}_I \, \beta \\ C_w^{J,0} \vee C_w^{J,-1} \vee C_w^{J,-2} & \text{if } \gamma = \bullet_I \, \alpha \\ C_w^{J,0} \vee C_w^{J,+1} \vee C_w^{J,+2} & \text{if } \gamma = \circ_I \, \alpha \end{cases}$$

with

$$C_w^{J,\pm\ell} := \begin{cases} mc_I^{J,J} \wedge \alpha^J \wedge \neg tp^J & \text{if } \ell = 0 \text{ and } I \neq \{0\} \\ mc_I^{\max\{J,J\pm1\},\min\{J,J\pm1\}} \wedge \alpha^{J\pm1} \wedge tp^J \wedge tp^{J\pm1} & \text{if } \ell = \pm1 \text{ and } j \pm 1 \in pos(w) \\ mc_I^{\max\{J,J\pm2\},\min\{J,J\pm2\}} \wedge \alpha^{J\pm2} \wedge \neg tp^{J\pm1} & \text{if } \ell = \pm2 \text{ and } j \pm 2 \in pos(w) \\ \mathsf{f} & \text{otherwise} \end{cases}$$

where $J \pm \ell$ denotes the interval of w's letter at the position $j \pm \ell$, provided that $j \pm \ell \in pos(w)$. We also define the substitution θ_w over the propositions of $\Phi_w^{\gamma,J}$ as follows.

$$\alpha^K \mapsto \begin{cases} \mathsf{t} & \text{if } [\![w, k \models\!\approx \alpha]\!] = \mathsf{t} \\ \mathsf{f} & \text{if } [\![w, k \models\!\approx \alpha]\!] = \mathsf{f} \end{cases} \qquad tp^K \mapsto \mathsf{t} \quad \text{if } |K| = 1 \qquad mc_I^{H,K} \mapsto \begin{cases} \mathsf{t} & \text{if } H - K \neq \emptyset \text{ and } H - K \subseteq I \\ \mathsf{f} & \text{if } (H - K) \cap I = \emptyset \end{cases}$$

For the propositions not listed, θ_w is undefined. In general, a substitution θ is a partial function from propositions to propositional formulas. Its homomorphic extension to propositional formulas is as expected, in particular, $\theta(\Psi)$ is the propositional formula in which the occurrences of propositions $p \in \mathrm{def}(\theta)$ within the propositional formula Ψ are replaced by $\theta(p)$, and the occurrences of propositions not in $\mathrm{def}(\theta)$ are unaltered.

Let $\equiv$ denote semantic equivalence between propositional formulas. The following lemma connects γ's truth value under MTL's three-valued semantics with the propositional formula $\theta_w(\Phi_w^{\gamma,J})$. Its proof is straightforward and omitted.

Lemma 4.1. *The following two statements hold.*

(i) If $[\![w, j \models\!\approx \gamma]\!] \in 2$ *then* $\theta_w(\Phi_w^{\gamma,J}) \equiv [\![w, j \models\!\approx \gamma]\!]$.
(ii) If $[\![w, j \models\!\approx \gamma]\!] = \bot$ *then* $\theta_w(\Phi_w^{\gamma,J}) \not\equiv \mathsf{t}$ *and* $\theta_w(\Phi_w^{\gamma,J}) \not\equiv \mathsf{f}$.

Note that the propositional formula $\theta_w(\Phi_w^{\gamma,J})$ tells us more than the truth value $[\![w, j \models\!\approx \gamma]\!]$. When $\theta_w(\Phi_w^{\gamma,J}) \not\equiv b$, for $b \in 2$, we also know, in addition to $[\![w, j \models\!\approx \gamma]\!] = \bot$, what causes the uncertainty, namely, the corresponding counterparts of the propositions that are not replaced by Boolean constants.

Next, we provide a tailored version of $\Phi_w^{\gamma,J}$ that is better suited for monitoring. Note that for the cases $\gamma = \alpha \, \mathsf{S}_I \, \beta$ and $\gamma = \alpha \, \mathsf{U}_I \, \beta$, at an anchor position K, the truth value $[\![w, k \models\!\approx \alpha]\!]$ is irrelevant for $[\![w, j \models\!\approx \gamma]\!]$. However, when $|K| > 1$ and when refining w by splitting K at some $\kappa \in K$, we obtain new anchor and continuation positions for which the truth value $[\![w, k \models\!\approx \alpha]\!]$ becomes relevant. With the tailored version $\Psi_w^{\gamma,J}$ of $\Phi_w^{\gamma,J}$ we keep track of α's truth value at

anchor positions K (cf. Example 4.2). The definition of $\Psi_w^{\gamma,J}$ is as follows.

$$\Psi_w^{\gamma,J} := \begin{cases} \bigvee_{K \leq J} \left(tp^K \wedge mc_I^{J,K} \wedge \beta^K \wedge (\overline{tp}^K \rightarrow \alpha^K) \wedge \bigwedge_{K<H\leq J} (tp^H \rightarrow \alpha^H) \right) & \text{if } \gamma = \alpha \, \mathsf{S}_I \, \beta \\ \bigvee_{K \geq J} \left(tp^K \wedge mc_I^{K,J} \wedge \beta^K \wedge (\overline{tp}^K \rightarrow \alpha^K) \wedge \bigwedge_{J\leq H<K} (tp^H \rightarrow \alpha^H) \right) & \text{if } \gamma = \alpha \, \mathsf{U}_I \, \beta \\ \Phi_w^{\gamma,J} & \text{otherwise} \end{cases}$$

For the new propositions $\overline{tp}^K$, we extend the substitution θ_w by $\overline{tp}^K \mapsto \mathsf{f}$ if $|K| = 1$.

Example 4.2. We illustrate the definitions of the propositional formulas $\Phi_w^{\gamma,J}$ and $\Psi_w^{\gamma,J}$, with $\gamma = \alpha \mathsf{U} \beta$, and the reason for using $\theta_w(\Psi_w^{\gamma,J})$ for monitoring. Let w be an observation with the intervals $J_0 = [0, \tau)$, $J_1 = \{\tau\}$, and $J_2 = (\tau, \infty)$, and where α's and β's truth values are everywhere $\bot$, except for position 1, where $[\![w, 1 \models\!\approx \alpha]\!] = [\![w, 1 \models\!\approx \beta]\!] = \mathsf{f}$. By definition,

$$\begin{aligned} \Phi_w^{\gamma,J_0} &= \left(tp^{J_0} \wedge mc_I^{J_0,J_0} \wedge \beta^{J_0} \right) \vee \left(tp^{J_1} \wedge mc_I^{J_1,J_0} \wedge \beta^{J_1} \wedge (tp^{J_0} \rightarrow \alpha^{J_0}) \right) \vee \\ &\quad \left(tp^{J_2} \wedge mc_I^{J_2,J_0} \wedge \beta^{J_2} \wedge (tp^{J_0} \rightarrow \alpha^{J_0}) \wedge (tp^{J_1} \rightarrow \alpha^{J_1}) \right), \\ \Psi_w^{\gamma,J_0} &= \left(tp^{J_0} \wedge mc_I^{J_0,J_0} \wedge \beta^{J_0} \wedge (\overline{tp}^{J_0} \rightarrow \alpha^{J_0}) \right) \vee \left(tp^{J_1} \wedge mc_I^{J_1,J_0} \wedge \beta^{J_1} \wedge (\overline{tp}^{J_1} \rightarrow \alpha^{J_1}) \wedge (tp^{J_0} \rightarrow \alpha^{J_0}) \right) \vee \\ &\quad \left(tp^{J_2} \wedge mc_I^{J_2,J_0} \wedge \beta^{J_2} \wedge (\overline{tp}^{J_2} \rightarrow \alpha^{J_2}) \wedge (tp^{J_1} \rightarrow \alpha^{J_1}) \wedge (tp^{J_0} \rightarrow \alpha^{J_0}) \right), \end{aligned}$$

and

$$\theta_w = [\alpha^{J_1} \mapsto \mathsf{f}, \beta^{J_1} \mapsto \mathsf{f}, tp^{J_1} \mapsto \mathsf{t}, \overline{tp}^{J_1} \mapsto \mathsf{f}, mc_I^{J_0,J_0} \mapsto \mathsf{t}, mc_I^{J_1,J_0} \mapsto \mathsf{t}, mc_I^{J_2,J_0} \mapsto \mathsf{t}] \,.$$

Furthermore, let w' be the observation that is obtained from w by the transformation (T1), where the interval J_0 is split at $\kappa \in J_0$. That is, the intervals of w' are $K_0 = [0, \kappa)$, $K_1 = \{\kappa\}$, $K_2 = (\kappa, \tau)$, J_1, and J_2. Note that $[\![w', 3 \models\!\approx \alpha]\!] = [\![w', 3 \models\!\approx \beta]\!] = \mathsf{f}$ and $\bot$ anywhere else.

We have the following semantic equivalences.

$$\begin{aligned} &\theta_w(\Phi_w^{\gamma,J_0}) \equiv tp^{J_0} \wedge \beta^{J_0} && \theta_{w'}(\Phi_{w'}^{\gamma,K_1}) \equiv \beta^{K_1} \vee \left(tp^{K_2} \wedge \beta^{K_2} \wedge \alpha^{K_1} \right) \\ &\theta_w(\Psi_w^{\gamma,J_0}) \equiv tp^{J_0} \wedge \beta^{J_0} \wedge (\overline{tp}^{J_0} \rightarrow \alpha^{J_0}) && \theta_{w'}(\Psi_{w'}^{\gamma,K_1}) \equiv \beta^{K_1} \vee \left(tp^{K_2} \wedge \beta^{K_2} \wedge (\overline{tp}^{K_2} \rightarrow \alpha^{K_2}) \wedge \alpha^{K_1} \right) \end{aligned}$$

Observe that the proposition α^{J_0} does not occur in $\theta_w(\Phi_w^{\gamma,J_0})$. In contrast, α^{J_0} occurs in $\theta_w(\Psi_w^{\gamma,J_0})$. With the subformula $\overline{tp}^{J_0} \rightarrow \alpha^{J_0}$, we store information about α's truth value in J_0. In this example, since $\theta_w(\overline{tp}^{J_0} \rightarrow \alpha^{J_0}) \equiv \overline{tp}^{J_0} \rightarrow \alpha^{J_0}$ we know that α's truth value in J_0 is $\bot$. If $\theta_w(\overline{tp}^{J_0} \rightarrow \alpha^{J_0}) \equiv \mathsf{t}$, we infer that α's truth value in J_0 is t, and if $\theta_w(\overline{tp}^{J_0} \rightarrow \alpha^{J_0}) \equiv \neg\overline{tp}^{J_0}$, α's truth value in J_0 is f. This information is relevant when splitting J_0. In particular, it allows us to obtain $\theta_{w'}(\Psi_{w'}^{\gamma,K_1})$ from $\theta_w(\Psi_w^{\gamma,J_0})$ because all propositions that occur in $\theta_{w'}(\Psi_{w'}^{\gamma,K_1})$ originate from propositions that already occur in $\theta_w(\Psi_w^{\gamma,J_0})$. ◁

The following lemma shows that Lemma 4.1 carries over to $\theta_w(\Psi_w^{\gamma,J})$. We omit its straightforward proof.

Lemma 4.3. *For $b \in 2$,*

$$\theta_w(\Phi_w^{\gamma,J}) \equiv b \qquad \textit{iff} \qquad \theta_w(\Psi_w^{\gamma,J}) \equiv b \,.$$

4.2.2 State Variables. The monitor's state consists of the global variable observation and the global variables $\mathsf{gate}^{\gamma,J}$, where γ is a nonatomic subformula of the monitored formula φ and J is an interval. The monitor stores in the state variable observation its knowledge about the system behavior. This variable is updated in each iteration according to the message

received by the procedure UpdateKnowledge (cf. Section 4.1). More concretely, in the monitor's ith iteration, the state variable observation equals the observation w_i of the valid observation sequence $(w_i)_{i\in\mathbb{N}}$. The state variables $\text{gate}^{\gamma,J}$ are used for the verdict computation. In particular, the monitor maintains the invariant $\text{gate}^{\gamma,J} \equiv \theta_{w_i}(\Psi_{w_i}^{\gamma,J})$. Because of the assumption that φ is not an atomic formula, the monitor only needs to maintain state variables $\text{gate}^{\gamma,K}$, where γ is not atomic. Atomic formulas p only occur as propositions p^J in the propositional formulas. We remark that we chose the variable name gate since the propositional formulas can be seen as logic gates in a combinational circuit. Each such gate computes a Boolean operation, where the input signals are the formula's propositions.

For the sake of simplicity, we do not explicitly remove irrelevant state variables. Instead, we assume that they are automatically "garbage collected." For instance, a state variable $\text{gate}^{\gamma,J}$ with $|J| = 1$ becomes irrelevant when it is semantically equivalent to a Boolean constant and its truth value has been propagated and, when γ is the monitored formula φ, the verdict for J has been output. For simplicity, we also do not discard any knowledge about the system behavior. In practice, one would remove irrelevant information from the state variable observation, for example, isolated time points for which the monitor has already output a verdict.

Instead of fixing a concrete representation of the propositional formulas that are stored in the state variables $\text{gate}^{\gamma,J}$, we provide an abstract interface for accessing and updating $\text{gate}^{\gamma,J}$. In Section 4.3, we use this interface to describe the monitor's central algorithmic details, which are independent from an actual representation of the propositional formulas. In Section 5.2, we describe a graph-based data structure for implementing the interface for the generalized setting with the freeze quantifier. Note that this presentation-independent description also allows us to separate concerns in the monitor's correctness proof (cf. Section 5.3). The interface comprises the following procedures.

- Clone(gate): returns a copy of gate.
- IsBool(gate): returns true iff gate is semantically equivalent to a Boolean constant.
- ToBool(gate): returns the Boolean value $b \in 2$, provided that gate is semantically equivalent to the corresponding Boolean constant.
- Contains(gate, p): returns true iff gate depends on the proposition p, that is, $[p \mapsto \mathsf{t}](\text{gate}) \not\equiv [p \mapsto \mathsf{f}](\text{gate})$. We shall abuse terminology in the following by also saying that p occurs in gate, although the occurrence of a proposition in a formula can be representation dependent.
- Eval(gate, θ): applies the substitution θ to gate, where θ only replaces propositions p with one of the Boolean constants t or f.
- Instantiate(gate): substitutes Boolean constants for the propositions of the form tp^L, $\overline{tp}^L$, $mc_I^{L,L'}$, and $(\mathsf{t})^L$ in gate, wherever possible. Note that the Boolean constants for these propositions can be determined by their name. For instance, $mc_I^{L,L'}$ is replaced by t iff $L - L' \neq \emptyset$ and $L - L' \subseteq I$, and by f iff $I \cap (L - L') = \emptyset$. Furthermore, note that Instantiate is a special case of Eval.
- Rename(gate, θ): applies the substitution θ to gate, where θ only renames propositions p with some proposition p'.

For the following last two interface procedures Add and Remove, we first introduce the following additional notion. The propositional formulas $\theta_w(\Psi_w^{\gamma,J})$ and hence also $\text{gate}^{\gamma,J}$ can be grouped into subformulas with respect to an interval and a direct subformula of γ. For instance, note that $\text{gate}^{\alpha \mathsf{U}_I \beta,J}$ is semantically equivalent to $\bigvee_{K \geq J}(\mathit{anchor}_K \wedge \bigwedge_{H \leq L < K} \mathit{continuation}_L)$, with $\mathit{anchor}_K = tp^K \wedge mc_I^{K,J} \wedge \beta^K \wedge (\overline{tp}^K \rightarrow \alpha^K)$ and $\mathit{continuation}_L = tp^L \rightarrow \alpha^L$, possibly with some of their literals replaced by Boolean constants, and where the intervals K and L range over the intervals of the letters in observation. The (β, K)-*relevant* part of $\text{gate}^{\alpha \mathsf{U}_I \beta,J}$ is the propositional formula anchor_K. The (α, L)-*relevant* part of $\text{gate}^{\alpha \mathsf{U}_I \beta,J}$ is the propositional formula $\mathit{continuation}_L$ if $L \neq J$, and, if $L = J$, the subformula $\overline{tp}^J \rightarrow \alpha^J$ of

```
procedure Init(φ)
  observation := w_0
  foreach γ with γ ∈ sub(φ) and not IsAtom(γ) do
    # Iterate top down with respect to φ's formula structure.
    gate^{γ,[0,∞)} := Ψ_{w_0}^{γ,[0,∞)}
    Instantiate(gate^{γ,[0,∞)})
    if IsBool(gate^{γ,[0,∞)}) then
      PropagateTruthValue(γ, [0, ∞), ToBool(gate^{γ,[0,∞)}))
```

Listing 2. Initialization procedure.

$anchor_J$, possibly with some propositions replaced by Boolean constants. Note that the relevant part can be t or f. For example, for the formula $\theta_{w'}(\Psi_{w'}^{\gamma,K_1})$ in Example 4.2, the (β, K_1)-relevant part is β^{K_1}, the (β, K_2)-relevant part is $tp^{K_2} \wedge \beta^{K_2} \wedge (\overline{tp}^{K_2} \rightarrow \alpha^{K_2})$, and the (β, H)-relevant part is f, for H being K_0, J_1, or J_2. Furthermore, the (α, K_2)-relevant part is $\overline{tp}^{K_2} \rightarrow \alpha^{K_2}$ and the (α, H)-relevant part is t, for H being K_0, K_1, J_1, or J_2. The definition of the relevant parts for other formulas γ is as expected and omitted. For instance, the (α, J)-relevant part of $\text{gate}^{\alpha \vee \beta, J} = \alpha^J$ is α^J and its (β, J)-relevant part is f. Note that for temporal formulas the relevant parts are always defined for their direct subformulas. For nontemporal formulas, the relevant parts are only defined for their direct subformulas and when the intervals match.

- Add(gate, α, K, Θ): replaces the (α, K)-relevant part of gate with the propositional formula Θ. We require that Θ is of the form of the relevant parts of gate.
- Remove(gate, α, K): returns the (α, K)-relevant part of gate and "removes" it from gate. For anchors, the removal corresponds to a replacement with the Boolean constant f. For continuations, the relevant part is replaced by the Boolean constant t.

4.2.3 Initialization. The procedure Init, shown in Listing 2, initializes the state variables. Initially, observation is the word w_0. Recall that $w_0 = \big([0, \infty), ([\,], [\,])\big)$. Furthermore, for $\gamma \in sub(\varphi)$, Init initializes $\text{gate}^{\gamma,[0,\infty)}$ with the propositional formula $\theta_{w_0}(\Psi_{w_0}^{\gamma,[0,\infty)})$. For this, the Init procedure uses the interface procedure Instantiate and the procedure PropagateTruthValue, which we present in Section 4.3.3, for propagating Boolean values up the formula structure. Since the formula is traversed top down, Boolean truth values are always propagated to already initialized state variables.

4.3 Algorithmic Details

In the following, we provide algorithmic details for the monitor's central procedures AddTimePoint, RemoveInterval, and PropagateTruthValue. Recall from Section 4.1 that each of these procedures updates the monitor's state, in particular, the propositional formulas stored in the gate variables according to one of the transformations (T1), (T2), and (T3).

4.3.1 Adding a Time Point. The pseudocode of the procedure AddTimePoint is given in Listing 3. Suppose that the respective transformation splits the interval J at τ. For $\tau > 0$, we obtain the new intervals $L := J \cap [0, \tau)$, $T := \{\tau\}$, and $R := J \cap (\tau, \infty)$. For brevity, we do not present the details for the corner case $\tau = 0$, where we only obtain two new intervals, $\{0\}$ and $J \setminus \{0\}$. We first create for each $\gamma \in sub(\varphi)$ three copies of $\text{gate}^{\gamma,J}$. Namely, we create the propositional formulas $\text{gate}^{\gamma,L}$, $\text{gate}^{\gamma,T}$, and $\text{gate}^{\gamma,R}$. Afterwards, we remove all the propositional formulas for the interval J from the monitor's state, that is, we delete $\text{gate}^{\gamma,J}$, for all $\gamma \in sub(\varphi)$. We then handle the special case where $\text{gate}^{\varphi,\{\tau\}}$ is semantically equivalent to a Boolean constant, which results in outputting a verdict for the new time point.

```
procedure AddTimePoint(φ, J, τ)
  foreach gate^{γ,J} and K ∈ {J ∩ [0, τ), {τ}, J ∩ (τ, ∞)} do
    gate^{γ,K} := Clone(gate^{γ,J})
  Delete(J)
  if IsBool(gate^{φ,{τ}}) then
    OutputVerdict({τ}, ToBool(gate^{φ,{τ}}))
  foreach gate^{γ,H} with Contains(gate^{γ,H}, p),
                          for some proposition p with the interval J do
    # Iterate top down with respect to φ's formula structure.
    case γ = ¬α:       Rename(gate^{γ,H}, [α^J ↦ α^H])
    case γ = α ∨ β:    Rename(gate^{γ,H}, [α^J ↦ α^H, β^J ↦ β^H])
    case γ = ●_I α:    ... # Omitted; analogous to the next case.
    case γ = ○_I α:    RefineNext(γ, H, J, τ)
                       Instantiate(gate^{γ,H})
    case γ = α S_I β:  ... # Omitted; analogous to the next case.
    case γ = α U_I β:  RefineUntil(γ, H, J, τ)
                       Instantiate(gate^{γ,H})
    if IsBool(gate^{γ,H}) then
      PropagateTruthValue(γ, H, ToBool(gate^{γ,H}))
```

Listing 3. Procedure for transformation (T1).

```
procedure RemoveInterval(K)
  foreach gate^{γ,J} with J ≠ K and Contains(gate^{γ,J}, tp^K) do
    Eval(gate^{γ,J}, [tp^K ↦ f])
    if IsBool(gate^{γ,J}) then
      PropagateTruthValue(γ, J, ToBool(gate^{γ,J}))
  Delete(K)
```

Listing 4. Procedure for transformation (T2).

```
procedure PropagateTruthValue(α, J, b)
  if IsRoot(α) and |J| = 1 then
    OutputVerdict(J, b)
  else if not IsRoot(α) then
    foreach gate^{γ,K} with Contains(gate^{γ,K}, α^J) do
      Eval(gate^{γ,K}, [α^J ↦ b])
      if IsBool(gate^{γ,J}) then
        PropagateTruthValue(γ, J, ToBool(gate^{γ,J}))
```

Listing 5. Procedure for transformation (T3).

```
procedure RefineNext(○_I α, H, J, τ)
  γ, L, T, R := ○_I α, J ∩ [0, τ), {τ}, J ∩ (τ, ∞)
  C_0, C_1 := Remove(gate^{γ,H}, α, H), Remove(gate^{γ,H}, α, H + 1)
  Remove(gate^{γ,H}, α, H + 2)
  if H ⊈ J then # Note that J = H + 1, |H| = 1, and C_0 ≡ f.
    Add(gate^{γ,H}, α, L, [mc_I^{J,H} ↦ mc_I^{L,H}, tp^J ↦ tp^L, α^J ↦ α^L](C_1))
    Add(gate^{γ,H}, α, T, [mc_I^{J,H} ↦ mc_I^{T,H}, tp^J ↦ ¬tp^L, α^J ↦ α^T](C_1))
  else if H = L then
    Add(gate^{γ,H}, α, L, [mc_I^{J,J} ↦ mc_I^{L,L}, tp^J ↦ tp^L, α^J ↦ α^L](C_0))
    Add(gate^{γ,H}, α, T, [mc_I^{J,J} ↦ mc_I^{T,L}, tp^J ↦ ¬tp^L, α^J ↦ α^T](C_0))
  else if H = T then
    Add(gate^{γ,H}, α, R, [mc_I^{J,J} ↦ mc_I^{R,T}, tp^J ↦ ¬tp^R, α^J ↦ α^R](C_0))
    Add(gate^{γ,H}, α, H + 2, [mc_I^{H+2,J} ↦ mc_I^{H+2,T}, tp^J ↦ ¬tp^R](C_1))
  else if H = R then
    Add(gate^{γ,H}, α, R, [mc_I^{J,J} ↦ mc_I^{R,R}, tp^J ↦ tp^R, α^J ↦ α^R](C_0))
    Add(gate^{γ,H}, α, H + 2, [mc_I^{J+1,J} ↦ mc_I^{H+2,R}, tp^J ↦ tp^R](C_1))
```

Listing 6. Auxiliary procedure for (T1) and the connective $\bigcirc_I$.

```
procedure RefineUntil(α U_I β, H, J, τ)
  γ, L, T, R := α U_I β, J ∩ [0, τ), {τ}, J ∩ (τ, ∞)
  anchor, continuation := Remove(gate^{γ,H}, β, J), Remove(gate^{γ,H}, α, J)
  foreach K with K ∈ {L, T, R} and H ≤ K do
    θ := [α^J ↦ α^K, β^J ↦ β^K, tp^J ↦ tp^K] ∪
                 [mc_I^{J,L} ↦ mc_I^{K,L} | for some interval L]
    Add(gate^{γ,H}, β, K, θ[tp̄^J ↦ tp̄^K](anchor))
    Add(gate^{γ,H}, α, K, θ[tp̄^J ↦ tp^K](continuation))
  if H ⊆ J then
    Rename(gate^{γ,H}, [mc_I^{L,J} ↦ mc_I^{L,H} | for some interval L])
```

Listing 7. Auxiliary procedure for (T1) and the connective U_I.

Finally, we update the propositional formulas $\text{gate}^{\gamma,H}$ in which a proposition with the interval J occurs. Note that H can be different from L, T, and R when γ is a temporal formula. We make a case split on γ's form. The cases p, $\neg\alpha$, and $\alpha \vee \beta$ are obvious. We replace any proposition with the interval J by the corresponding proposition with the interval H. Let us turn to the case where γ is of the form $\alpha \,\mathsf{U}_I\, \beta$. We omit the dual case $\alpha \,\mathsf{S}_I\, \beta$. The procedure RefineUntil, shown in Listing 7, updates the anchor and continuation subformulas of the propositional formula $\text{gate}^{\gamma,H}$, that is, the (β, J)-relevant and (α, J)-relevant parts of $\text{gate}^{\gamma,H}$.

- If $H > J$, we replace the (β, J)-relevant part with the three relevant parts for the intervals L, T, and R. They originate from the (β, J)-relevant part. Similarly, we replace the (α, J)-relevant parts.
- If H is one of the intervals L, T, or R, we replace the relevant parts with the interval J up to the interval H. Furthermore, we need to adjust the J interval in the *mc* propositions in $\text{gate}^{\gamma,H}$.

After RefineUntil, AddTimePoint calls Instantiate to replace propositions with Boolean constants where possible. Note that after the instantiation, $\text{gate}^{\gamma,H}$ can be semantically equivalent to a Boolean constant. The **if** statement at the end of the second **foreach** loop performs the corresponding check and triggers the propagation.

Finally, let us consider the case where γ is of the form $\bigcirc_I \alpha$. We omit the dual case $\bullet_I \alpha$. The procedure RefineNext, shown in Listing 6, updates a propositional formula $\text{gate}^{\gamma,H}$ as follows. It first removes all its relevant parts. Note that these have one of the three intervals H, $H+1$, and $H+2$. Then, depending on whether $H \not\subseteq J$ or H is one of the intervals L, T, or R, RefineNext adds the new relevant parts to $\text{gate}^{\gamma,H}$. These parts originate from the old relevant parts with the intervals H and $H+1$.

4.3.2 Removing an Interval. The pseudocode of the procedure RemoveInterval is given in Listing 4. Since K does not contain any time points, we replace any occurrence of the proposition tp^K by f. It suffices to only update propositional formulas $\text{gate}^{\gamma,J}$, where γ is a temporal formula and $J \neq K$. Note that this replacement could trigger the propagation of Boolean values. For instance, we propagate f from $\text{gate}^{\alpha \mathsf{U}_I \beta,J}$, if K is the only anchor in $\text{gate}^{\alpha \mathsf{U}_I \beta,J}$, that is, if for all $H \neq K$, the (β,H)-relevant part of $\text{gate}^{\alpha \mathsf{U}_I \beta,J}$ is f. Afterwards, we delete all the propositional formulas $\text{gate}^{\gamma,K}$ with $\gamma \in \mathit{sub}(\varphi)$ from the monitor's state.

4.3.3 Propagating a Boolean Value. The pseudocode of the procedure PropagateTruthValue is given in Listing 5. The procedure is called whenever a propositional formula $\text{gate}^{\alpha,J}$ simplifies to a Boolean constant b. If $|J| = 1$ and $\alpha = \varphi$, we output a verdict. Otherwise, for $\alpha \neq \varphi$, we substitute the proposition α^J with its Boolean value b in all the propositional formulas $\text{gate}^{\gamma,K}$. Note that γ must be the parent formula of α. However, for temporal formulas, K can be different from J. We continue the propagation whenever the updated $\text{gate}^{\gamma,K}$ propositional formula is semantically equivalent to a Boolean constant.

5 MONITORING WITH DATA VALUES

In this section, we extend the online algorithm from Section 4 for MTL to $\text{MTL}^{\downarrow}$. The handling of the freeze quantifier is orthogonal to the core ideas already used for monitoring MTL specifications. In Section 5.1, we present the extension. In Section 5.2, we describe the graph-based data structure for representing and manipulating the propositional formulas of the monitor's state. Finally, in Section 5.3, we establish the algorithm's correctness. Note that the data structure and the correctness proof also apply to the restricted setting of Section 4, that is, the online algorithm for MTL.

5.1 Algorithmic Details

Throughout this section, we reuse the conventions that we introduced in Section 4 for MTL. Analogous to Section 4, we also assume that φ is not an atomic formula and subformulas of φ are pairwise distinct. Furthermore, we require that the monitored formula φ is closed. Finally, we assume that variables are frozen at most once in φ. This assumption is also without loss of generality and it allows us to identify a frozen variable with the respective subformula.

5.1.1 Reduction to Propositional Logic. Similar to MTL, at the core of the monitor for $\text{MTL}^{\downarrow}$ is a mapping of $\text{MTL}^{\downarrow}$'s three-valued semantics into propositional logic. To this end, we first extend the propositional formulas $\Psi_w^{\gamma,J}$ from Section 4.2 to $\Psi_w^{\gamma,J,\nu}$ to capture $\text{MTL}^{\downarrow}$'s freeze quantifier. We remark that the only change in the definition below is

that each proposition for a subformula $\alpha \in sub(\varphi)$ now also carries a partial valuation ν in addition to an interval K.

$$\Psi_w^{\gamma,J,\nu} := \begin{cases} \mathsf{t} & \text{if } \gamma = \mathsf{t} \\ p(\overline{x})^{J,\nu} & \text{if } \gamma = p(\overline{x}) \text{ with } p \in P \\ \alpha^{J,\nu[x\mapsto\bot]} & \text{if } \gamma = \downarrow^r x.\ \alpha \text{ and } r \notin \mathrm{def}(\varrho_j) \\ \alpha^{J,\nu[x\mapsto\varrho_j(r)]} & \text{if } \gamma = \downarrow^r x.\ \alpha \text{ and } r \in \mathrm{def}(\varrho_j) \\ \neg\alpha^{J,\nu} & \text{if } \gamma = \neg\alpha \\ \alpha^{J,\nu} \vee \beta^{J,\nu} & \text{if } \gamma = \alpha \vee \beta \\ \bigvee_{K\le J} \left(tp^K \wedge mc_I^{J,K} \wedge \beta^{K,\nu} \wedge (\overline{tp}^K \to \alpha^{K,\nu}) \wedge \bigwedge_{K<H\le J}(tp^H \to \alpha^{H,\nu})\right) & \text{if } \gamma = \alpha\ \mathsf{S}_I\ \beta \\ \bigvee_{K\ge J} \left(tp^K \wedge mc_I^{K,J} \wedge \beta^{K,\nu} \wedge (\overline{tp}^K \to \alpha^{K,\nu}) \wedge \bigwedge_{J\le H<K}(tp^H \to \alpha^{H,\nu})\right) & \text{if } \gamma = \alpha\ \mathsf{U}_I\ \beta \\ C_w^{J,0,\nu} \vee C_w^{J,-1,\nu} \vee C_w^{J,-2,\nu} & \text{if } \gamma = \bullet_I\,\alpha \\ C_w^{J,0,\nu} \vee C_w^{J,+1,\nu} \vee C_w^{J,+2,\nu} & \text{if } \gamma = \circ_I\,\alpha \end{cases}$$

with

$$C_w^{J,\pm\ell,\nu} := \begin{cases} mc_I^{J,J} \wedge \alpha^{J,\nu} \wedge \neg tp^J & \text{if } \ell = 0 \text{ and } I \neq \{0\} \\ mc_I^{\max\{J,J\pm1\},\min\{J,J\pm1\}} \wedge \alpha^{J\pm1,\nu} \wedge tp^J \wedge tp^{J\pm1} & \text{if } \ell = \pm1 \text{ and } j \pm 1 \in pos(w) \\ mc_I^{\max\{J,J\pm2\},\min\{J,J\pm2\}} \wedge \alpha^{J\pm2,\nu} \wedge \neg tp^{J\pm1} & \text{if } \ell = \pm2 \text{ and } j \pm 2 \in pos(w) \\ \mathsf{f} & \text{otherwise} \end{cases}$$

In the following, let θ_w be the substitution that maps a proposition of the form $\alpha^{K,\nu}$ to $[\![w,k,\nu \models\!\approx \alpha]\!]$, provided that $[\![w,k,\nu \models\!\approx \alpha]\!] \in 2$, and for the other propositions, θ_w is as in Section 4, namely,

$$tp^K \mapsto \mathsf{t} \quad \text{if } |K| = 1, \qquad mc_I^{H,K} \mapsto \begin{cases} \mathsf{t} & \text{if } H - K \neq \emptyset \text{ and } H - K \subseteq I, \\ \mathsf{f} & \text{if } (H-K) \cap I = \emptyset, \text{ and} \end{cases} \qquad \overline{tp}^K \mapsto \mathsf{f} \quad \text{if } |K| = 1.$$

Example 5.1. We illustrate the definition of the propositional formulas $\Psi_w^{\gamma,J,\nu}$. Consider the formula $\downarrow^r x.\ \alpha$ with $\alpha = \Diamond_{(0,1]}\, p(x)$. For readability, we use β for the subformula $p(x)$. Let w be the observation $\big([0,\tau),([\,],[\,])\big)\big(\{\tau\},([\,],[r\mapsto d])\big)\big((\tau,\infty),([\,],[\,])\big)$. We obtain the following propositional formulas, where $J_0 = [0,\tau)$, $J_1 = \{\tau\}$, $J_2 = (\tau,\infty)$, and $\bar\beta^{J,H,\nu}$ abbreviates the conjunction $tp^H \wedge mc_I^{J,H} \wedge \beta^{H,\nu}$.

$$\theta_w(\Psi_w^{\downarrow^r x.\alpha,J_0,[\,]}) \equiv \alpha^{J_0,[\,]} \qquad \theta_w(\Psi_w^{\downarrow^r x.\alpha,J_1,[\,]}) \equiv \alpha^{J_1,[x\mapsto d]} \qquad \theta_w(\Psi_w^{\downarrow^r x.\alpha,J_2,[\,]}) \equiv \alpha^{J_2,[\,]}$$
$$\theta_w(\Psi_w^{\alpha,J_0,[\,]}) \equiv \bar\beta^{J_0,J_0,[\,]} \vee \bar\beta^{J_1,J_0,[\,]} \vee \bar\beta^{J_2,J_0,[\,]} \qquad \theta_w(\Psi_w^{\alpha,J_1,[x\mapsto d]}) \equiv \bar\beta^{J_2,J_1,[x\mapsto d]} \qquad \theta_w(\Psi_w^{\alpha,J_2,[\,]}) \equiv \bar\beta^{J_2,J_2,[\,]}$$

First, note that as $\Diamond_{(0,1]}\,\beta$ is syntactic sugar for $\mathsf{t}\,\mathsf{U}_{(0,1]}\,\beta$, we can ignore the continuation subformulas in the propositional formulas since they simplify to t. Furthermore, note that $\theta_w(\Psi_w^{\alpha,J_1,[x\mapsto d]})$ has only one anchor subformula (different from a propositional constant), since α's temporal constraint $(0,1]$ is unsatisfiable for J_1, that is, $\theta_w(mc_{(0,1]}^{J_1,J_1}) = \mathsf{f}$ and hence $\theta_w(\bar\beta^{J_1,J_1,[x\mapsto d]}) \equiv \mathsf{f}$. Finally, note that for β and J_2, we have the two propositions $\beta^{J_2,[\,]}$ and $\beta^{J_2,[x\mapsto d]}$, where $\beta^{J_2,[\,]}$ occurs in both $\theta_w(\Psi_w^{\alpha,J_0,[\,]})$ and $\theta_w(\Psi_w^{\alpha,J_2,[\,]})$, and $\beta^{J_2,[x\mapsto d]}$ occurs once in $\theta_w(\Psi_w^{\alpha,J_1,[x\mapsto d]})$. ◁

Lemma 4.1 for MTL carries over to MTL$^\downarrow$ and its propositional formulas $\Psi_w^{\gamma,J,\nu}$.

Lemma 5.2. *The following two statements hold.*

(1) If $[\![w,j,\nu \models\!\approx \gamma]\!] \in 2$ *then* $\theta_w(\Psi_w^{\gamma,J,\nu}) \equiv [\![w,j,\nu \models\!\approx \gamma]\!]$.

```
procedure MonitorMTL↓(φ)
  Init(φ)
  loop
    m := ReceiveMessage()
    ts := UpdateKnowledge(m)
    foreach t in ts do
      case (T1):  τ, J := DeltaT1(t)
                  AddTimePoint(φ, J, τ)
      case (T2):  K := DeltaT2(t)
                  RemoveInterval(K)
      case (T3.1): τ, σ := DeltaT31(t)
                  foreach p(x̄)^{τ},ν with p ∈ def(σ) and x̄ ∈ def(ν) do
                    PropagateTruthValue(p(x̄), {τ}, ν, ν(x̄) ∈ σ(p))
      case (T3.2): τ, ϱ := DeltaT32(t)
                  foreach α^{τ},ν with ↓^r x. α ∈ sub(φ) and r ∈ def(ϱ) do
                    Rename(gate^{↓^r x.α,{τ},ν}, [α^{τ},ν ↦ α^{τ},ν[x↦ϱ(r)]])
                    PropagateDataValue(α, {τ}, ν, [x ↦ ϱ(r)])
```

Listing 8. The monitor's main loop for MTL$^{\downarrow}$.

```
procedure Init(φ)
  observation := w0
  foreach γ ∈ sub(φ) with not IsAtom(γ) do
    # Iterate top down with respect to φ's formula structure.
    gate^{γ,[0,∞),[]} := Ψ_{w0}^{γ,[0,∞),[]}
    Instantiate(gate^{γ,[0,∞),[]})
    if IsBool(gate^{γ,[0,∞),[]}) then
      PropagateTruthValue(γ, [0, ∞), [], ToBool(gate^{γ,[0,∞),[]}))
```

Listing 9. Initialization procedure.

(2) If $[\![w, j, \nu \models\!\approx \gamma]\!] = \bot$ *then* $\theta_w(\Psi_w^{\gamma,J,\nu}) \not\equiv \mathsf{t}$ *and* $\theta_w(\Psi_w^{\gamma,J,\nu}) \not\equiv \mathsf{f}$.

Proof. We prove the lemma by a case split on γ. The case $\gamma = \mathsf{t}$ is obvious and omitted. For the case $\gamma = p(\overline{x})$, we have by definition that $\Psi_w^{p(\overline{x}),J,\nu} = p(\overline{x})^{J,\nu}$. Furthermore, when $[\![w, j, \nu \models\!\approx p(\overline{x})]\!] \in 2$, we have that $\theta_w(p(\overline{x})^{J,\nu}) = [\![w, j, \nu \models\!\approx p(\overline{x})]\!]$; otherwise, θ_w is not defined for $p(\overline{x})^{J,\nu}$. We conclude that both implications (1) and (2) hold. The cases for $\neg\alpha$ and $\alpha \vee \beta$ are similar and omitted. We also omit the details of the case for $\gamma = \downarrow^r x.\ \alpha$ as it is also similar to the case $p(\overline{x})$. Note that for $r \notin \mathrm{def}(\varrho_j)$, we have that $\Psi_w^{\downarrow^r x.\alpha,J,\nu} = \alpha^{J,\nu[x\mapsto\bot]}$ and $\Psi_w^{\downarrow^r x.\alpha,J,\nu} = \alpha^{J,\nu[x\mapsto\varrho_j(r)]}$, for $r \in \mathrm{def}(\varrho_j)$.

Finally, we provide proof details for the case $\gamma = \alpha\ \mathsf{S}_I\ \beta$. The cases for the other three temporal connectives are similar and omitted. For K with $K \le J$, the disjunction of the propositional formulas $\Psi := tp^K \wedge mc_I^{J,K} \wedge \beta^{K,\nu} \wedge (\overline{tp}^K \rightarrow \alpha^{K,\nu}) \wedge \bigwedge_{K<H\le J}(tp^H \rightarrow \alpha^{H,\nu})$ of $\Psi_w^{\alpha \mathsf{S}_I \beta,J,\nu}$ follows closely the semantic definition of $[\![w, j, \nu \models\!\approx \alpha\ \mathsf{S}_I\ \beta]\!]$ in Section 3.2, except that each Ψ contains the additional propositional subformula $\overline{tp}^K \rightarrow \alpha^{K,\nu}$. First, observe that the propositions tp^K and $mc_I^{J,K}$ together with the substitution θ_w take the role of $\mathrm{tp}_w(k)$ and $\mathrm{mc}_{w,I}(j,k)$. Furthermore, θ_w replaces the propositions $\beta^{K,\nu}$ and $\alpha^{H,\nu}$ in Ψ with the corresponding Boolean constants whenever the respective formulas evaluate to Boolean truth values under the three-valued semantics $[\![\cdot]\!]$. Finally, we observe that the additional propositional subformula $\overline{tp}^K \rightarrow \alpha^{K,\nu}$ in Ψ simplifies to t when $|K| = 1$, since $\theta_w(\overline{tp}^K) = \mathsf{f}$. For $|K| > 1$, $\overline{tp}^K \rightarrow \alpha^{K,\nu}$ simplifies either to $\overline{tp}^K \rightarrow \alpha^{K,\nu}$, $\overline{tp}^K$, or t. Since θ_w is also not defined for tp^K, when $|K| > 1$, $\theta_w(\Psi) \not\equiv \mathsf{t}$. Furthermore, $\theta_w(\Psi) \not\equiv \mathsf{f}$ if $\theta_w(\beta^{K,\nu}) \not\equiv \mathsf{f}$ and $\theta_w(\alpha^{H,\nu}) \not\equiv \mathsf{f}$, for all H with $|H| = 1$ and $K < H \le J$. With these observations, it is easy to see that the implications (1) and (2) hold. □

5.1.2 Main Procedure. The monitor's main procedure for MTL$^{\downarrow}$ is shown in Listing 8 and the initialization procedure in Listing 9. Both procedures are similar to their counterparts for MTL (see the Listings 1 and 2). The main difference is that the case (T3) now comprises two subcases. The first subcase (T3.1) handles new interpretations for predicate symbols at a time point and is similar to the (T3) case for MTL in Listing 1. The second subcase (T3.2) handles the freezing of variables at a time point to data values. Note that in the **foreach** loops in both subcases, the propositions $p(\overline{x})^{\{\tau\},\nu}$ and $\alpha^{\{\tau\},\nu}$ range over propositions that occur in some propositional formula of the monitor's state. In the following, we use (T3.1) and (T3.2) to refer to the transformation of the corresponding subcase, respectively.

```
procedure AddTimePoint(φ, J, τ)
  foreach gate^{γ,J,ν} and K ∈ {J ∩ [0, τ), {τ}, J ∩ (τ, ∞)} do
    gate^{γ,K,ν} := Clone(gate^{γ,J,ν})
  Delete(J)
  if IsBool(gate^{φ,{τ},[]}) then
    OutputVerdict({τ}, ToBool(gate^{φ,{τ},[]}))
  foreach gate^{γ,H,ν} with Contains(gate^{γ,H,ν}, p),
                            for some proposition p with the interval J do
    # Iterate top down with respect to φ's formula structure.
    case γ = ¬α:        Rename(gate^{γ,H,ν}, [α^{J,ν} ↦ α^{H,ν}])
    case γ = α ∨ β:     Rename(gate^{γ,H,ν}, [α^{J,ν} ↦ α^{H,ν}, β^{J,ν} ↦ β^{H,ν}])
    case γ = ↓^s y. α:  Rename(gate^{γ,H,ν},
                                 [α^{J,μ} ↦ α^{H,μ} | μ some partial valuation])
    case γ = ●_I α:     ... # Omitted; analogous to the next case.
    case γ = ○_I α:     RefineNext(γ, H, ν, J, τ)
                        Instantiate(gate^{γ,H,ν})
    case γ = α S_I β:   ... # Omitted; analogous to the next case.
    case γ = α U_I β:   RefineUntil(γ, H, ν, J, τ)
                        Instantiate(gate^{γ,H,ν})
    if IsBool(gate^{γ,H,ν}) then
      PropagateTruthValue(γ, H, ν, ToBool(gate^{γ,H,ν}))
```

Listing 10. Procedure for transformation (T1).

```
procedure RemoveInterval(K)
  foreach gate^{γ,J,ν} with J ≠ K and Contains(gate^{γ,J,ν}, tp^K) do
    Eval(gate^{γ,J,ν}, [tp^K ↦ f])
    if IsBool(gate^{γ,J,ν}) then
      PropagateTruthValue(γ, J, ν, ToBool(gate^{γ,J,ν}))
  Delete(K)
```

Listing 11. Procedure for transformation (T2).

```
procedure PropagateTruthValue(α, J, ν, b)
  if IsRoot(α) and |J| = 1 then
    OutputVerdict(J, b)
  else if not IsRoot(α) then
    foreach gate^{γ,K,μ} with Contains(gate^{γ,K,μ}, α^{J,ν}) do
      Eval(gate^{γ,K,μ}, [α^{J,ν} ↦ b])
      if IsBool(gate^{γ,K,μ}) then
        PropagateTruthValue(γ, K, μ, ToBool(gate^{γ,K,μ}))
```

Listing 12. Procedure for transformation (T3.1).

```
procedure PropagateDataValue(γ, K, ν, [x ↦ d])
  if γ = p(x̄) then
    σ := PredicateInterpretations(observation, K)
    if p ∈ def(σ) and x̄ ∈ def(ν[x ↦ d]) then
      PropagateTruthValue(γ, K, ν[x ↦ d], ν[x ↦ d](x̄) ∈ σ(p))
  else if gate^{γ,K,ν[x↦d]} does not exist then
    gate^{γ,K,ν[x↦d]} := Clone(gate^{γ,K,ν})
    if IsBool(gate^{γ,K,ν[x↦d]}) then
      PropagateTruthValue(γ, K, ν[x ↦ d], ToBool(gate^{γ,K,ν[x↦d]}))
    Rename(gate^{γ,K,ν[x↦d]}, [α^{L,μ} ↦ α^{L,μ[x↦d]} | α ∈ sub(γ),
                                 L some interval, and μ some partial valuation])
    foreach α^{L,μ[x↦d]} with Contains(gate^{γ,K,ν[x↦d]}, α^{L,μ[x↦d]}) do
      PropagateDataValue(α, L, μ, [x ↦ d])
  else if IsBool(gate^{γ,K,ν[x↦d]}) then
    PropagateTruthValue(γ, K, ν[x ↦ d], ToBool(gate^{γ,K,ν[x↦d]}))
```

Listing 13. Procedure for the transformation (T3.2).

5.1.3 State Updates. The central procedures for updating the monitor's state are the procedures AddTimePoint, RemoveInterval, PropagateTruthValue, and PropagateDataValue. Their pseudocode is given in the Listings 10–13. The first three procedures extend their counterparts for MTL from Section 4. The last one is new and propagates data values down the formula structure. As in Section 4, we do not fix the representation of the propositional formulas of the monitor's state. Instead, we use the same abstract interface for accessing and updating the state variables $\text{gate}^{\gamma,J,\nu}$ as described in Section 4.2.2.

If γ is an atomic formula of the form $p(\overline{x})$, PropagateDataValue first obtains the interpretation of the predicate symbols at the position k of observation. It starts the propagation of the truth value, if $p(\overline{x})$ can be evaluated for the extended partial valuation $\nu[x \mapsto d]$. If γ is not an atomic formula and the propositional formula $\text{gate}^{\gamma,K,\nu[x\mapsto d]}$ for the extended partial valuation $\nu[x \mapsto d]$ does not exist yet, PropagateDataValue creates it from $\text{gate}^{\gamma,K,\nu}$. When $\text{gate}^{\gamma,K,\nu[x\mapsto d]}$ is semantically equivalent to a Boolean constant, PropagateDataValue starts the propagation of the truth value. Otherwise, PropagateDataValue continues the propagation of the new data value down the formula structure. Finally, if $\text{gate}^{\gamma,K,\nu[x\mapsto d]}$ already exists and is semantically equivalent to a Boolean constant, then—as in the case where $\text{gate}^{\gamma,K,\nu[x\mapsto d]}$ is newly created—PropagateDataValue starts the propagation of the truth value.

5.2 Data Structure

We briefly describe a graph-based data structure for representing and updating the monitor's state variables $\text{gate}^{\gamma,J,\nu}$. The *nodes* of the data structure are tuples of the form (γ, J, ν), with γ a subformula of the monitored formula φ, J an interval, and ν a partial valuation. When γ is not atomic, the node corresponds to the state variable $\text{gate}^{\gamma,J,\nu}$. A

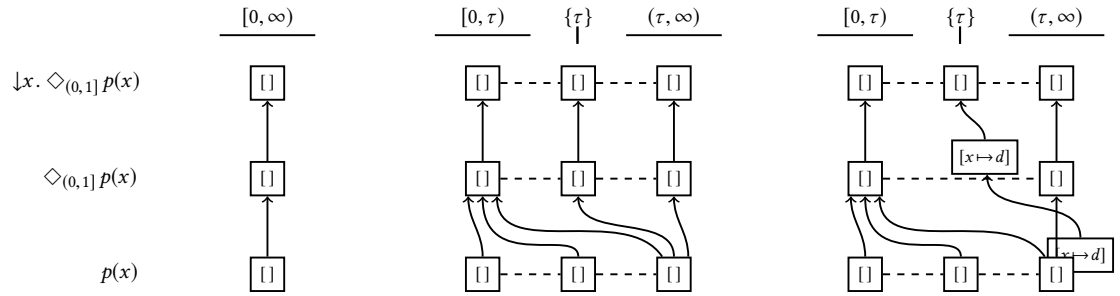
node (γ, J, ν) stores a truth value $b \in 3$, where the monitor maintains the invariant $b = [\![\text{observation}, j, \nu \models\!\approx \gamma]\!]$. If γ is of the form $\alpha \mathsf{U}_I \beta$ or $\alpha \mathsf{S}_I \beta$, then the node also stores the interval K of the closest valid anchor (i.e., for $\alpha \mathsf{U}_I \beta$, $k \geq j$ with $\mathrm{tp}_{\text{observation}}(k) = \mathrm{mc}_{\text{observation},I}(k, j) = [\![\text{observation}, k, \nu \models\!\approx \beta]\!] = \mathsf{t}$ and $[\![\text{observation}, h, \nu \models\!\approx \alpha]\!] \neq \mathsf{f}$, for all h with $j \leq h < k$ and $\mathrm{tp}_{\text{observation}}(h) = \mathsf{t}$), if it exists. Furthermore, nodes with the same formula γ and partial valuation ν are stored in a doubly-linked list, ordered by their intervals. The *edges* of the data structure are as follows. There is an edge from the node (α, K, μ) to the node (γ, J, ν) if a proposition of the (α, K)-relevant part of $\mathrm{gate}^{\gamma,J,\nu}$ occurs in the propositional formula $\mathrm{gate}^{\gamma,J,\nu}$. The edges are bidirectional. To simplify the exposition, we use an upward directed reading, namely, from nodes with the formula α to nodes with α's parent formula γ. For instance, both nodes (α, J, ν) and (β, J, ν) have an outgoing edge to the node $(\alpha \vee \beta, J, \nu)$, provided that the truth value of both nodes (α, J, ν) and (β, J, ν) is $\bot$.

We sketch how this data structure realizes the interface specified in Section 4.2.2. We first note that the graph-based data structure does not represent the propositional formulas $\mathrm{gate}^{\gamma,J,\nu}$ explicitly. However, an explicit representation of them can be obtained from its nodes and edges. From the incoming edges of a node (γ, J, ν), we can obtain the relevant parts of $\mathrm{gate}^{\gamma,J,\nu}$, in particular, the propositions occurring in them. Their arrangement, including the Boolean connectives between the propositions and the relevant parts, is given through γ's main connective and its direct subformulas. For example, for $\gamma = \alpha \mathsf{U}_I \beta$, whether the proposition $\beta^{K,\nu}$ occurs in the (β, K)-relevant part of $\mathrm{gate}^{\gamma,J,\nu}$ can be determined from the node's (β, K, ν) truth value and the interval of the valid anchor in the node (γ, J, ν). Note that $\mathrm{gate}^{\gamma,J,\nu}$ does not depend on $\beta^{K,\nu}$ when the node (γ, J, ν) has a closest valid anchor with the interval K' and $K' < K$. Furthermore, whether the propositions tp^K and $\overline{tp}^K$ occur in the (β, K)-relevant part of $\mathrm{gate}^{\gamma,J,\nu}$ can be determined from the interval of the node (β, K, ν). Similarly, whether the proposition $mc_I^{K,J}$ occurs in the (β, K)-relevant part of $\mathrm{gate}^{\gamma,J,\nu}$ can be determined from the intervals of the nodes (β, K, ν) and (γ, J, ν).

The realization of the interface procedures is not difficult. For instance, the procedures Add and Remove simply add and remove edges. However, some care must be taken for the procedure Eval. Assume that the arguments of Eval are $\mathrm{gate}^{\gamma,J,\nu}$ and the substitution $[\alpha^{H,\nu} \mapsto \mathsf{f}]$, where $\gamma = \alpha \mathsf{U}_I \beta$ and $H > J$ with $|H| = 1$. Obviously, Eval deletes the edge from the node (α, H, ν) to the node (γ, J, ν). This deletion may trigger the deletion of other incoming edges to the node (γ, J, ν). First, Eval deletes the incoming edges from the "anchor" nodes (β, K, ν), with $K > H$. Additionally, Eval deletes the interval L of the node's (γ, J, ν) valid anchor, provided it exists and $L > H$. Furthermore, Eval deletes the incoming edges from the "continuation" nodes (α, K, ν) that have no anchor anymore. These "continuation" nodes may arise when deleting the node's valid anchor or an incoming edge from an anchor node. Finally, Eval sets the node's (γ, J, ν) truth value to f, if there are no remaining incoming edges.

Example 5.3. We illustrate the data structure and its updates. Figure 1 shows the data structures associated with the formula $\downarrow^r x.\Diamond_{(0,1]} p(x)$ and the observations (a) $w_0 = \big([0,\infty), ([\,],[\,])\big)$, (b) $w_1 = \big([0,\tau), ([\,],[\,])\big)\big(\{\tau\}, ([\,],[\,])\big)\big((\tau,\infty), ([\,],[\,])\big)$, and (c) $w = \big([0,\tau), ([\,],[\,])\big)\big(\{\tau\}, ([\,],[r \mapsto d])\big)\big((\tau,\infty), ([\,],[\,])\big)$. A box in Figure 1 corresponds to a node of the graph-based data structure, where the node's formula is given by the row of the box, the interval by the column of the box, and the partial valuation is given inside the box. The edges are depicted as solid lines between boxes. The dashed lines are the links of the ordered doubly-linked lists. Note that the three boxes in Figure 1(a) and the two boxes in Figure 1(c) with the partial valuation $[x \mapsto d]$ are all stored in singleton lists.

Note that w_1 is obtained from w_0 by a (T1) transformation that splits the interval $[0,\infty)$ at τ, and w is obtained from w_1 by a (T3.2) transformation that freezes the data value d to the variable x at τ. Observe that Figure 1(c) does not contain the node $(\Diamond_{(0,1]} p(x), \{\tau\}, [\,])$. This node is irrelevant, since it has no outgoing edges. Irrelevant nodes



(a) initial observation w_0 (b) observation w_1 (c) observation w

Fig. 1. Graph-based data structure (Example 5.3).

are removed from the data structure. Furthermore, note that the data structure shown in Figure 1(c) represents the propositional formulas $\theta_w(\Psi_w^{\gamma,J,\nu})$ from Example 5.1. The nonexistence of the node $(\Diamond_{(0,1]}\, p(x), \{\tau\}, [\,])$ corresponds to the fact that the proposition $\Diamond_{(0,1]}\, p(x)^{\{\tau\},[\,]}$ does not occur in any of the propositional formulas. ◁

We remark that the data structure allows us to easily determine the propositional formulas $\mathsf{gate}^{\gamma,J,\nu}$ in which a given proposition $\alpha^{K,\mu}$ occurs. We just need to follow the node's (α, K, μ) outgoing edges, provided that the node's truth value is $\bot$. Analogously, by following a node's (γ, J, ν) incoming edges we can determine the propositions that occur in $\mathsf{gate}^{\gamma,J,\nu}$. Hence, the **foreach** loops in the procedures RemoveInterval and PropagateTruthValue, and the second one in AddTimePoint can be implemented efficiently. The data structure can also be further optimized. For example, to reduce the number of edges, a node only stores at most one outgoing edge. The other outgoing edges are implicit and computed on demand by following the links of the doubly-linked lists to the neighboring nodes. In particular, the procedure AddTimePoint needs to update significantly fewer outgoing edges when splitting an interval. We omit such implementation details.

5.3 Correctness

This section is dedicated to the monitor's correctness and we prove the following theorem.

Theorem 5.4. MonitorMTL$^{\downarrow}$ *is observationally complete and sound.*

Proof. We first observe that the monitor only outputs verdicts with the procedure AddTimePoint(φ, J, τ) and the procedure PropagateTruthValue(α, J, ν, b) when $\alpha = \varphi$ and J is a singleton. In both cases, IsBool($\mathsf{gate}^{\varphi,J,[\,]}$) returns true. For the second case, observe that PropagateTruthValue is only called when IsBool($\mathsf{gate}^{\varphi,J,[\,]}$) returns true. Moreover, ν is $[\,]$ in these calls, since state variables $\mathsf{gate}^{\alpha,J,\nu}$ with new partial valuations $\nu \neq [\,]$ are only created by the procedure PropagateDataValue(γ, K, ν, $[x \mapsto d]$), which is never called with the argument $\gamma = \varphi$. Thus whenever the monitor outputs a verdict (J, b), then $J = \{\tau\}$ and $\mathsf{gate}^{\varphi,J,[\,]} \equiv b$, for some $\tau \in \mathbb{Q}_{\geq 0}$ and $b \in 2$.

Let $\bar{w}$ be a valid observation sequence that represents the monitor's input. Without loss of generality, we assume that a single transformation is applied in each iteration, that is, for each $i \in \mathbb{N}$, w_{i+1} is obtained from w_i by exactly one of the transformations (T1), (T2), (T3.1), or (T3.2). For an observation w of $\bar{w}$, we denote by $\mathsf{gate}_w^{\gamma,J,\nu}$ the value of the state variable $\mathsf{gate}^{\gamma,J,\nu}$ at the end of the iteration that processes the observation w, that is, w is the value of the monitor's state variable observation.

The equivalence below follows from Lemma 5.8, which is stated and proved later. For an observation w of $\bar{w}$, a time point in w with timestamp τ, and $b \in 2$, it holds that

$$\theta_w(\Psi_w^{\varphi,\{\tau\},[\,]}) \equiv b \quad \text{iff} \quad \mathsf{gate}_w^{\varphi,\{\tau\},[\,]} \equiv b\,. \tag{1}$$

Furthermore, we note that in the iteration w, the monitor's state contains the state variable $\mathsf{gate}^{\varphi,J,[\,]}$ for any interval J that occurs in a letter of w.

Observational soundness follows from the above observation on when the monitor output verdicts, the equivalence (1), and Lemma 5.2. To show observational completeness, suppose that $[w, \tau, [\,] \models\!\!\approx \varphi] = b \in 2$. We must show that the verdict $(\{\tau\}, b)$ is output in this iteration w or has already been output in a previous iteration of $\bar{w}$. From $[w, \tau, [\,] \models\!\!\approx \varphi] \in 2$, it follows that there is a time point $j \in \mathit{pos}(w)$ with the timestamp τ. Furthermore, $[\![w, j, [\,] \models\!\!\approx \varphi]\!] = b$. It follows from Lemma 5.2 that $\theta_w(\Psi_w^{\varphi,\{\tau\},[\,]}) \equiv b$, and by (1), we obtain that $\mathsf{gate}_w^{\varphi,\{\tau\},[\,]} \equiv b$. We are done when the procedure PropagateTruthValue outputs the verdict $(\{\tau\}, b)$. Otherwise, let $w' \sqsubseteq w$ be the first observation in $\bar{w}$ for which $\mathsf{IsBool}(\mathsf{gate}_{w'}^{\varphi,J,[\,]})$ returns true, for some interval J with $\tau \in J$. Furthermore, let w'' be the observation of $\bar{w}$ when $J' \subseteq J$ is split into $J' \cap [0, \tau)$, $\{\tau\}$, and $J' \cap (\tau, \infty)$. Clearly, $w' \sqsubseteq w'' \sqsubsetneq w$. In this iteration, $\mathsf{gate}_{w''}^{\varphi,\{\tau\},[\,]}$ is set to $\mathsf{gate}_{w'}^{\varphi,J',[\,]}$ by the call to Clone in the AddTimePoint procedure. Note that $\mathsf{gate}_{w''}^{\varphi,\{\tau\},[\,]} \equiv \mathsf{gate}_{w'}^{\varphi,J,[\,]} \equiv b$. After the creation of $\mathsf{gate}_{w''}^{\varphi,\{\tau\},[\,]}$, the monitor outputs the verdict $(\{\tau\}, b)$ by calling the procedure OutputVerdict. □

In the remainder of this section, we establish the monitor's key invariants (Lemma 5.7 and Lemma 5.8). The equivalence (1), used to prove Theorem 5.4, is a straightforward consequence of Lemma 5.8, and Lemma 5.7 is used to establish Lemma 5.8. To state the invariants, we introduce further notation. As in the proof of Theorem 5.4, let $\bar{w}$ be a valid observation sequence that represents the monitor's input. Again, we assume without loss of generality that w_{i+1} is obtained from w_i by exactly one of the transformations (T1), (T2), (T3.1), or (T3.2), for each $i \in \mathbb{N}$. Furthermore, $\mathsf{gate}_w^{\gamma,J,\nu}$ denotes the value of the state variable $\mathsf{gate}^{\gamma,J,\nu}$ at the end of the iteration that processes the observation w of $\bar{w}$. To simplify matters, we also assume that state variables are not garbage collected even when they are irrelevant. This assumption does not affect the monitor's correctness because for an irrelevant state variable $\mathsf{gate}_w^{\psi,J,\nu}$, the corresponding proposition $\psi^{J,\nu}$ does not occur in any relevant gate state variable. The monitor only does more work than necessary.

The following definition allows us to state which state variables the monitor maintains. For an observation w, we define inductively the set $\mathit{val}_w(\psi, J)$ of the *relevant valuations* for $\psi \in \mathit{sub}(\varphi)$ at interval J, where J ranges over the intervals that occur in the letters of w, as

$$\mathit{val}_w(\varphi, J) := \{[\,]\} \qquad \text{and} \qquad \mathit{val}_w(\psi, J) := \left\{\nu \;\middle|\; \theta_w(\Psi_w^{\gamma,K,\mu}) \text{ depends on } \psi^{J,\nu}, \text{ for some } K \text{ and } \mu \in \mathit{val}_w(\gamma, K)\right\},$$

for $\psi \neq \varphi$, with the parent formula γ. Recall that a propositional formula Ψ *depends on* the proposition p if $[p \mapsto \mathsf{t}](\Psi) \not\equiv [p \mapsto \mathsf{f}](\Psi)$.

Example 5.5. We revisit Example 5.1 with the formula $\downarrow^r x\,.\,\alpha$ and the observation $w = \big(J_0, ([\,], [\,])\big)\big(J_1, ([\,], [r \mapsto d])\big)\big(J_2, ([\,], [\,])\big)$. Recall that $\alpha = \Diamond_{(0,1]}\,\beta$, with $\beta = p(x)$, $J_0 = [0, \tau)$, $J_1 = \{\tau\}$, and $J_2 = (\tau, \infty)$. We have the following relevant valuations.

$$\begin{aligned}
\mathit{val}_w(\downarrow^r x\,.\,\alpha, J_0) &= \{[\,]\} & \mathit{val}_w(\downarrow^r x\,.\,\alpha, J_1) &= \{[\,]\} & \mathit{val}_w(\downarrow^r x\,.\,\alpha, J_2) &= \{[\,]\}\\
\mathit{val}_w(\alpha, J_0) &= \{[\,]\} & \mathit{val}_w(\alpha, J_1) &= \{[x \mapsto d]\} & \mathit{val}_w(\alpha, J_2) &= \{[\,]\}\\
\mathit{val}_w(\beta, J_0) &= \{[\,]\} & \mathit{val}_w(\beta, J_1) &= \{[\,]\} & \mathit{val}_w(\beta, J_2) &= \{[\,], [x \mapsto d]\}
\end{aligned}$$

For instance, $[x \mapsto d] \in val_w(\beta, J_2)$ because $\theta_w(\Psi_w^{\alpha, J_1, [x \mapsto d]})$ depends on $\beta^{J_2, [x \mapsto d]}$ and $[x \mapsto d] \in val_w(\alpha, J_1)$. The latter membership in turn holds because $\theta_w(\Psi_w^{\downarrow^r x.\alpha, J_1, [\,]})$ depends on $\alpha^{J_1, [x \mapsto d]}$ and $[\,] \in val_w(\downarrow^r x\,.\,\alpha, J_1)$, by the definition of the base case. We also point out the correspondence between the nodes in the graph-based data structure and the relevant valuations. Compare, for instance, Figure 1(c) and the relevant valuations from this example. ◁

Finally, we make the simplifying assumption that only a single variable is frozen to a data value by (T3.2) transformations. That is, we assume that a register occurs at most once in the formula φ. Note that for a register r that occurs twice in φ, we can replace one occurrence with a fresh register r' and assume that r' carries the same data value at a time point as r. Furthermore, we can split a (T3.2) transformation into multiple ones such that the register assignment of any of these transformations only maps a single register to a data value. Under this assumption, the following technical lemma holds, which states that when this transformation is used, only subformulas of the freeze subformula containing the involved register can have new relevant valuations.

Lemma 5.6. *Let w and w' be observations such that w' is obtained from w by the transformation (T3.2), with τ and ϱ the corresponding timestamp and register assignment, respectively. For any $\psi \in sub(\varphi)$, interval J in w, and partial valuation ν, it holds that if $\nu \in val_{w'}(\psi, J) \setminus val_w(\psi, J)$, then ψ is a proper subformula of some $\downarrow^r x\,.\,\alpha \in sub(\varphi)$ with $r \in \mathrm{def}(\varrho)$ and $\nu(x) = \varrho(r)$. Additionally, the following conditions hold for any partial valuation μ, if also $\mu \in val_{w'}(\gamma, K)$ and $\theta_{w'}(\Psi_{w'}^{\gamma, K, \mu})$ depends on $\psi^{J, \nu}$, where γ is ψ's parent formula and K an interval in w.*

(1) If $\mu \notin val_w(\gamma, K)$ then $\mu(x) = \varrho(r)$ and $\theta_w(\Psi_w^{\gamma, K, \mu[x \mapsto \bot]})$ depends on $\psi^{J, \nu[x \mapsto \bot]}$.
(2) If $\mu \in val_w(\gamma, K)$ then $\gamma = \downarrow^r x\,.\,\alpha$, $\{\tau\} = J = K$, and $\nu = \mu[x \mapsto \varrho(r)]$.

Proof. We prove the lemma's first part by contraposition. Namely, we show that if ψ is not a proper subformula of some $\downarrow^r x\,.\,\alpha \in sub(\varphi)$ with $r \in \mathrm{def}(\varrho)$ and $\nu(x) = \varrho(r)$, then $val_{w'}(\psi, J) \subseteq val_w(\psi, J)$. If $\psi = \varphi$ then, by definition, $val_{w'}(\psi, J) = val_w(\psi, J) = \{[\,]\}$. Let γ be ψ's parent formula. By assumption, γ is not a subformula of some $\downarrow^r x\,.\,\alpha$ with $r \in \mathrm{def}(\varrho)$ and $\nu(x) = \varrho(r)$. We have that if $\theta_{w'}(\Psi_{w'}^{\gamma, K', \mu'})$ depends on $\psi^{J', \nu'}$ then $\theta_w(\Psi_w^{\gamma, K', \mu'})$ also depends on $\psi^{J', \nu'}$, for any intervals K' and J' of w and partial valuations μ' and ν'. Note that $\theta_w \sqsubseteq \theta_{w'}$ and $\Psi_{w'}^{\gamma, K', \mu'} = \Psi_w^{\gamma, K', \mu'}$. It follows that $val_{w'}(\psi, J) \subseteq val_w(\psi, J)$.

We make a case split to prove the lemma's second part.

Case I: $\psi = \alpha$. That is, $\gamma = \downarrow^r x\,.\,\psi$. We first show that $\mu \in val_w(\gamma, K)$. If, for the sake of a contradiction, $\mu \notin val_w(\gamma, K)$, it follows from the lemma's first part for γ, K, and μ that γ is a proper subformula of some $\downarrow^{r'} x'\,.\,\alpha'$, with $r' \in \mathrm{def}(\varrho)$ and $\mu(x') = \varrho(r')$. This contradicts the assumption that only one variable is frozen to a data value by the transformation. Hence, $\mu \in val_w(\gamma, K)$, and (1) trivially holds. We prove (2). Note that since $\gamma = \downarrow^r x\,.\,\alpha$, we have that $\Psi_w^{\gamma, K, \mu} = \psi^{K, \nu'}$ and $\Psi_{w'}^{\gamma, K, \mu} = \psi^{K, \nu''}$, for some partial valuations ν' and ν''. From $\nu \in val_{w'}(\psi, J)$, it follows that $\theta_{w'}(\Psi_{w'}^{\gamma, K, \mu}) \equiv \psi^{J, \nu}$ and thus $J = K$. From $\nu \in val_{w'}(\psi, J) \setminus val_w(\psi, J)$, it follows that $\{\tau\} = J = K$. From the definition of $\Psi_{w'}^{\gamma, \mu, K}$, it follows that $\nu = \mu[x \mapsto \varrho(r)]$.

Case II: ψ is a proper subformula of α. As $\theta_{w'}(\Psi_{w'}^{\gamma, K, \mu})$ depends on $\psi^{J, \nu}$, we obtain that $\theta_w(\Psi_w^{\gamma, K, \mu})$ depends on $\psi^{J, \nu}$. If $\mu \in val_w(\gamma, K)$ then $\nu \in val_w(\psi, J)$, which contradicts the assumption $\nu \in val_{w'}(\psi, J) \setminus val_w(\psi, J)$. Hence, $\mu \notin val_w(\gamma, K)$, and (2) trivially holds. We prove (1). From the lemma's first part applied to γ, K, and μ, we obtain that $\mu(x) = \varrho(r)$, and therefore $\mu(x) = \nu(x)$. Furthermore, as $\psi^{J, \nu} \notin \mathrm{def}(\theta_{w'})$, we have that $\psi^{J, \nu[x \mapsto \bot]} \notin \mathrm{def}(\theta_w)$, and thus $\theta_w(\Psi_w^{\gamma, K, \mu[x \mapsto \bot]})$ depends on $\psi^{J, \nu[x \mapsto \bot]}$. □

The next lemma establishes the key invariant about the existence of the monitor's gate state variables.

Lemma 5.7. *Let w be an observation of $\bar{w}$, J an interval of w, $\psi \in sub(\varphi)$ a nonatomic formula, and $\nu \in val_w(\psi, J)$ a partial valuation. The monitor's state at the iteration that processes w contains the state variable* $\mathsf{gate}^{\psi,J,\nu}$.

Proof. We reason by induction on the position of w in the sequence $\bar{w}$. Recall that we assume, without loss of generality, that a single transformation is applied to an observation in $\bar{w}$. In the base case, the observation w is w_0. The interval $[0,\infty)$ is the only interval of a letter in w_0 and $val_{w_0}(\psi,[0,\infty)) = \{[\,]\}$, for any $\psi \in sub(\varphi)$. Since the monitor has not received any messages, only the procedure Init has been executed so far. Init creates in its **foreach** loop for every $\psi \in sub(\varphi)$ the state variable $\mathsf{gate}^{\psi,[0,\infty),[\,]}$. This concludes the base case.

For the step case, we assume that the statement holds for w and prove it for w', the observation after w in $\bar{w}$. Let $\psi \in sub(\varphi)$, J an interval of w', and $\nu \in val_{w'}(\psi, J)$. We must prove the existence of the state variable $\mathsf{gate}^{\psi,J,\nu}_{w'}$. We make a case distinction on the type of the transformation t that transforms w into w'. The cases (T1), (T2), and (T3.1) are similar and straightforward. We only sketch the (T1) case. Let J' be the interval that is returned by $\mathsf{DeltaT1}(t)$, that is, the interval that is split. If $J \not\subseteq J'$, it follows that $\nu \in val_w(\psi, J)$. By the induction hypothesis, we have that $\mathsf{gate}^{\psi,J,\nu}_{w}$ exists. Since this state variable is not deleted, we have that $\mathsf{gate}^{\psi,J,\nu}_{w'}$ exists. If $J \subseteq J'$, that is, J originates from the interval J', we have that $\nu \in val_w(\psi, J')$ and obtain by the induction hypothesis that $\mathsf{gate}^{\psi,J',\nu}_{w}$ exists. The procedure AddTimePoint creates in its first **foreach** loop the state variable $\mathsf{gate}^{\psi,J,\nu}$ by cloning $\mathsf{gate}^{\psi,J',\nu}$.

It remains to prove the (T3.2) case. Let τ be the timestamp and ϱ the partial register assignment returned by $\mathsf{DeltaT32}(t)$. If $\mathsf{gate}^{\psi,J,\nu}_{w}$ exists, the existence of $\mathsf{gate}^{\psi,J,\nu}_{w'}$ directly follows from the observation that no state variable is deleted in the (T3.2) case. For the remainder of the proof, suppose that $\mathsf{gate}^{\psi,J,\nu}_{w}$ does not exist, where ψ is a proper subformula of φ with the parent formula γ. Note that if $\psi = \varphi$ then $\nu = [\,]$, since φ is closed. It is easy see that $\mathsf{gate}^{\varphi,J,[\,]}_{w}$ exists and hence also $\mathsf{gate}^{\varphi,J,[\,]}_{w'}$. From the induction hypothesis, it follows that $\nu \notin val_w(\psi, J)$. From $\nu \in val_{w'}(\psi, J)$, it follows that $\theta_{w'}(\Psi^{\gamma,K,\mu}_{w'})$ depends on $\psi^{J,\nu}$, for some interval K in w' and $\mu \in val_{w'}(\gamma, K)$. From Lemma 5.6, we obtain that γ is a subformula of some $\downarrow^r x.\ \alpha \in sub(\varphi)$, $r \in \mathrm{def}(\varrho)$, and $\nu(x) = \varrho(r)$. We prove the existence of $\mathsf{gate}^{\psi,J,\nu}_{w'}$ by induction on the distance between α and ψ, that is, the formula length of ψ minus the formula length of α.

For the base case, we have that $\psi = \alpha$ and $\gamma = \downarrow^r x.\ \alpha$. For the sake of contradiction, suppose that $\mu \notin val_w(\gamma, K)$. From Lemma 5.6(1), it follows that $\mu(x) = \varrho(r)$. However, from the definitions of $val_{w'}(\gamma, K)$ and $\Psi^{\gamma,K,\mu}_{w'}$, we have that $x \notin \mathrm{def}(\mu)$, which contradicts $\mu(x) = \varrho(r)$. Hence $\mu \in val_w(\gamma, K)$. From the outer induction hypothesis, it follows that $\mathsf{gate}^{\gamma,K,\mu}_{w}$ exists. By Lemma 5.6(2), we have that $\nu = \mu[x \mapsto \varrho(r)]$ and $J = K = \{\tau\}$. Therefore $\mathsf{PropagateDataValue}(\psi, J, \mu, [x \mapsto \varrho(r)])$ is called from $\mathsf{MonitorMTL}^{\downarrow}$. The first **else if** branch of PropagateDataValue is executed, which creates the state variable $\mathsf{gate}^{\psi,J,\nu}$.

For the step case, we have that ψ is a proper subformula of α. By the inner induction hypothesis, $\mathsf{gate}^{\gamma,K,\mu}_{w'}$ exists.
Case I: $\mathsf{gate}^{\gamma,K,\mu}_{w}$ does not exist. Therefore $\mathsf{gate}^{\gamma,K,\mu}$ is created at w' within $\mathsf{PropagateDataValue}(\gamma, K, \mu', [x \mapsto \varrho(r)])$, for some partial valuation μ'. Note that $\mu = \mu'[x \mapsto \varrho(r)]$ and $x \notin \mathrm{def}(\mu')$. It also follows from the outer induction hypothesis that $\mu \notin val_w(\gamma, K)$. From Lemma 5.6(1), it follows that $\theta_w(\Psi^{\gamma,K,\mu'}_{w})$ depends on $\psi^{J,\nu'}$, where $\nu' = \nu[x \mapsto \bot]$. This means that $\mathsf{Contains}(\mathsf{gate}^{\gamma,K,\mu'}_{w}, \psi^{J,\nu'})$ returns true. As $\mathsf{gate}^{\gamma,K,\mu}_{w'}$ is obtained from $\mathsf{gate}^{\gamma,K,\mu'}_{w}$ by cloning and renaming its propositions, we obtain that also $\mathsf{Contains}(\mathsf{gate}^{\gamma,K,\mu}_{w'}, \psi^{J,\nu})$ returns true. Therefore PropagateDataValue is called with the parameters ψ, J, ν', and $[x \mapsto \varrho(r)]$. The state variable $\mathsf{gate}^{J,\psi,\nu}_{w'}$ is created within this call.
Case II: $\mathsf{gate}^{\gamma,K,\mu}_{w}$ exists. It must be the case that $\theta_w(\Psi^{\gamma,K,\mu}_{w})$ depends on $\psi^{J,\nu}$, since $\theta_{w'}(\Psi^{\gamma,K,\mu}_{w'})$ depends on $\psi^{J,\nu}$. It follows that $\nu \in val_w(\psi, J)$, which is a contradiction and hence this second case cannot occur. □

The final lemma establishes the key invariant about the semantic equivalence of the monitor's gate state variables for which we have shown the existence in Lemma 5.7.

Lemma 5.8. *Let w be an observation of $\bar{w}$, J an interval of w, $\psi \in sub(\varphi)$ a nonatomic formula, and $\nu \in val_w(\psi, J)$ a partial valuation. It holds that* $\mathsf{gate}_w^{\psi,J,\nu} \equiv \theta_w(\Psi_w^{\psi,J,\nu})$.

Proof. As in Lemma 5.7, we reason by induction on the position of w in the sequence $\bar{w}$. In the base case, the observation w is w_0. We have that $J = [0,\infty)$ and $\nu = [\,]$. Only the procedure Init is executed, which initializes $\mathsf{gate}^{\psi,[0,\infty),[\,]}$ with $\Psi_{w_0}^{\psi,[0,\infty),[\,]}$. The execution of the procedures Instantiate and PropagateTruthValue, which are called by Init, results in applying the substitution θ_{w_0} to $\mathsf{gate}^{\psi,[0,\infty),[\,]}$. This concludes the base case.

For the step case, we assume that the statement holds for w and prove it for w', the observation after w in $\bar{w}$. Let $\psi \in sub(\varphi)$, J an interval of w', and $\nu \in val_{w'}(\psi, J)$. We make a case distinction on the type of the transformation t that transforms w into w'. We start with the (T3.2) case.

Transformation (T3.2). We first note that a state variable is modified only by Rename (from PropagateDataValue) and by Eval (from PropagateTruthValue). Furthermore, a state variable is modified at most once by Rename. Indeed, the first modification happens just after creating the state variable, using Clone. A second modification cannot happen, because the **else if** branch in which the second call would hypothetically occur is executed only when the state variable does not exist already. Also, a call to Rename cannot be preceded by a call to PropagateTruthValue (for the same gate state variable). We conclude that the possible modification by Rename precedes the modifications by Eval in the sequence of modifications of a state variable during the processing of the current transformation. We denote by $\mathsf{gate}_{w\to w'}^{\psi,J,\nu}$ the value of the $\mathsf{gate}^{\psi,J,\nu}$ after the possible modification by Rename, and before the modifications by Eval. Note also that if $\mathsf{gate}_w^{\psi,J,\nu}$ exists, then $\mathsf{gate}_{w\to w'}^{\psi,J,\nu} = \mathsf{gate}_w^{\psi,J,\nu}$.

We have that $\mathsf{gate}_{w\to w'}^{\psi,J,\nu} \equiv \theta_w(\Psi_{w'}^{\psi,J,\nu})$. Note that the right-hand side of the semantic equivalence uses the substitution for w and the propositional formula for w'. The proof is by a straightforward induction on the length of φ minus the length of ψ. We omit it.

We now prove that $\mathsf{gate}_{w'}^{\psi,J,\nu} \equiv \theta_{w'}(\Psi_{w'}^{\psi,J,\nu})$. We reason by an inner induction on the size of γ (i.e. on the number of its connectives). The base case (when the size of γ is 1) is a special case of the step case, and therefore omitted. For the step case, consider an arbitrary call to Eval with parameters $\mathsf{gate}_{w'}^{\psi,J,\nu}$ and $[\alpha^{K,\mu} \mapsto b]$. Clearly, α is a direct subformula of ψ. If α is atomic, then $\alpha = p(\overline{x})$ for some $p \in P$, and PropagateTruthValue(α, K, μ, b) was called from the PropagateDataValue procedure. Therefore $b = [\![w', k, \mu \mid\approx \alpha]\!]$. If α is not atomic, then PropagateTruthValue(α, K, μ, b) was called either from PropagateDataValue or from PropagateTruthValue (recursively). From the conditions under which the call was made (namely, that IsBool($\mathsf{gate}_{w'}^{\alpha,K,\mu}$) returns true), we deduce in all cases that $b \equiv \mathsf{gate}_{w'}^{\alpha,K,\mu}$. From the induction hypothesis and Lemma 5.2, it follows that $b = [\![w', k, \mu \mid\approx \alpha]\!]$. Thus, in both cases, $b = [\![w', k, \mu \mid\approx \alpha]\!]$. This also tells us that, for different calls to Eval, a proposition $\alpha^{K,\mu}$ cannot be replaced with different Boolean values. That is, we have shown that $\mathsf{gate}_{w'}^{\psi,J,\nu} \equiv \theta(\mathsf{gate}_{w\to w'}^{\psi,J,\nu})$, for some substitution θ that replaces propositions $\alpha^{K,\mu}$ with $[\![w', k, \mu \mid\approx \alpha]\!] \in 2$.

To conclude the (T3.2) case, it suffices to show that for any proposition $\alpha^{K,\mu}$ of $\mathsf{gate}_{w\to w'}^{\psi,J,\nu}$ such that $\mathsf{gate}_{w\to w'}^{\psi,J,\nu}$ depends on $\alpha^{K,\mu}$ and $\alpha^{K,\mu} \in \mathrm{def}(\theta_{w'}) \setminus \mathrm{def}(\theta_w)$, we have $\alpha^{K,\mu} \in \mathrm{def}(\theta)$. That is, we have that PropagateTruthValue(α, K, μ, b) is called, where $b = [\![w', k, \mu \mid\approx \alpha]\!]$. As $\alpha^{K,\mu} \in \mathrm{def}(\theta_{w'}) \setminus \mathrm{def}(\theta_w)$, we have that either $[\![w, k, \mu \mid\approx \alpha]\!] \notin 2$ or $\mu \notin val_w(\alpha, K)$. Note first that as $\alpha^{K,\mu} \in \mathrm{def}(\theta_{w'})$ we have that $\mu \in val_{w'}(\alpha, K)$. We now make a case distinction.

Case I: $\mu \in val_w(\alpha, K)$. Then $[\![w, k, \mu \mid\approx \alpha]\!] \notin 2$. Therefore $\mathsf{gate}_w^{\alpha,K,\mu}$ exists (by Lemma 5.7); however $\mathsf{gate}_w^{\alpha,K,\mu}$ is not semantically equivalent to a Boolean constant. As $\mu \in val_{w'}(\alpha, K)$, $\mathsf{gate}_{w'}^{\alpha,K,\mu}$ exists, by Lemma 5.7. Also, from the inner

induction hypothesis, $\text{gate}_{w'}^{\alpha,K,\mu} \equiv b$. Therefore, Eval was called on $\text{gate}^{\alpha,K,\mu}$ while executing PropagateTruthValue. Thus PropagateTruthValue(α, K, μ, b) is called.

Case II: $\mu \notin val_w(\alpha, K)$. Since $\mu \in val_{w'}(\alpha, K)$, then, as in the proof of Lemma 5.7, we obtain that PropagateDataValue is called with parameters α, K, $\mu[x \mapsto \bot]$, $[x \mapsto d]$, where x and d are the variable frozen by the current transformation and the corresponding value, respectively. Again, since the $\text{gate}_{w'}^{\alpha,K,\mu} \equiv b$ by the inner induction hypothesis, we have that PropagateTruthValue(α, K, μ, b) is called from PropagateDataValue. This concludes the (T3.2) case.

Transformation (T1). Let τ and K be the timestamp and the interval returned by DeltaT1(t), respectively. Note that $\tau \in K$ and we assume that $\tau > 0$. For an interval H of w', we define $\hat{H} := H$ if $H \not\subseteq K$ and $\hat{H} := K$ if $H \subseteq K$. As $\nu \in val_{w'}(\psi, J)$, we have that $\nu \in val_w(\psi, \hat{J})$. From Lemma 5.7, we obtain the existence of $\text{gate}_w^{\psi,\hat{J},\nu}$.

We first remark that the procedure AddTimePoint creates $\text{gate}^{\psi,J,\nu}$ in its first **foreach** loop from $\text{gate}^{\psi,\hat{J},\nu}$ by Clone, if $\hat{J} = K$. In AddTimePoint's second **foreach** loop, the procedures Rename, RefineNext, RefineUntil, Instantiate, or Eval may modify $\text{gate}^{\psi,J,\nu}$. Note that Rename, RefineNext, RefineUntil, or Instantiate are directly called from AddTimePoint and at most once. In contrast, Eval is called from PropagateTruthValue and Eval may modify $\text{gate}^{\psi,J,\nu}$ multiple times. Furthermore, Eval's modifications happen after modifications by Instantiate, which in turn happen after modifications by Rename, RefineNext, or RefineUntil. The reason is that the loop iterates top down over φ's formula structure. This means, if Eval modifies $\text{gate}^{\psi,J,\nu}$ in the iteration for some state variable $\text{gate}^{\gamma,H,\mu}$, then γ is a subformula of ψ. In particular, modifications by Rename, RefineNext, or RefineUntil on $\text{gate}^{\psi,J,\nu}$ have been carried out in an earlier iteration, namely, the one for $\text{gate}^{\psi,J,\nu}$. We denote by $\text{gate}_{w\to w'}^{\psi,J,\nu}$ the value of the state variable $\text{gate}^{\psi,J,\nu}$ after modifications by Rename, RefineNext, or RefineUntil, and before modifications by Instantiate or Eval.

The proof of $\text{gate}_{w'}^{\psi,J,\nu} \equiv \theta_{w'}(\Psi_{w'}^{\psi,J,\nu})$ comprises two parts. The first part shows that $\text{gate}_{w\to w'}^{\psi,J,\nu} \equiv \delta(\Psi_{w'}^{\psi,J,\nu})$, where δ behaves like θ_w, except that it carries over the truth value assignment for propositions with the interval K to the propositions originating from splitting K. That is, we define

$$\delta(p) := \begin{cases} \theta_w(\gamma^{\hat{H},\mu}) & \text{if } p = \gamma^{H,\mu} \text{ and } \gamma^{\hat{H},\mu} \in \text{def}(\theta_w), \\ \theta_w(mc_I^{\hat{H},\hat{L}}) & \text{if } p = mc_I^{H,L} \text{ and } mc_I^{\hat{H},\hat{L}} \in \text{def}(\theta_w), \\ \theta_w(p) & \text{if } p \in \text{def}(\theta_w) \text{ and } p \text{ is of the form } tp^H \text{ or } \overline{tp}^H. \end{cases}$$

Note that if $p \in \text{def}(\theta_w)$, then $\theta_w(p) \in 2$. Also note that δ is undefined for propositions of the form tp^H and $\overline{tp}^H$ with $H \subseteq K$. The second part, which we omit, since it is analogous to the second part of the previous (T3.2) case, uses the first part to show that $\text{gate}_{w'}^{\psi,J,\nu} \equiv \theta_{w'}(\Psi_{w'}^{\psi,J,\nu})$.

For the first part, it suffices to show that the relevant parts of $\text{gate}_{w\to w'}^{\psi,J,\nu}$ are semantically equivalent to their relevant counterparts in $\delta(\Psi_{w'}^{\psi,J,\nu})$. Indeed, note that $\theta_w(\Psi_w^{\gamma,\hat{J},\nu})$ is determined by its relevant parts. As $\text{gate}_w^{\psi,\hat{J},\nu} \equiv \theta_w(\Psi_w^{\psi,\hat{J},\nu})$ by the induction hypothesis, $\text{gate}_w^{\psi,\hat{J},\nu}$ and therefore also $\text{gate}^{\psi,J,\nu}$ when newly created are determined by their relevant parts. Finally, $\text{gate}_{w\to w'}^{\psi,J,\nu}$ is determined by its relevant parts, as $\text{gate}^{\psi,J,\nu}$ is only altered through the procedures of the interface presented in Section 4.2.2 (page 18). In the following, let χ be a direct subformula of ψ and H an interval of w'. We assume that the $(\chi, \hat{H})$-relevant part in $\Psi_w^{\psi,\hat{J},\nu}$ exists. Otherwise, there is nothing to prove. Furthermore, $\Theta(\Psi)$ denotes the (χ, H)-relevant part of the propositional formula Ψ, and $\hat{\Theta}(\Psi)$ denotes its $(\chi, \hat{H})$-relevant part.

There is a substitution ζ such that $\Theta(\Psi_{w'}^{\psi,J,\nu}) = \zeta\big(\hat{\Theta}(\Psi_w^{\psi,\hat{J},\nu})\big)$. For instance, if $\psi = \neg\alpha$, then $\zeta = [\alpha^{\hat{H},\nu} \mapsto \alpha^{H,\nu}]$, and if $\psi = \alpha \,\mathsf{U}_I\, \beta$ and $\chi = \beta$, then $\zeta = \zeta_2 \circ \zeta_1$, where $\circ$ denotes function composition, and ζ_1 and ζ_2 are the substitutions $[\alpha^{\hat{H},\nu} \mapsto \alpha^{H,\nu}, \beta^{\hat{H},\nu} \mapsto \beta^{H,\nu}, tp^{\hat{H}} \mapsto tp^H, \overline{tp}^{\hat{H}} \mapsto \overline{tp}^H] \cup [mc_I^{\hat{H},L} \mapsto mc_I^{H,L} \mid L \text{ is an interval in } w]$ and

$$\Box \downarrow^{cid} c.\ \downarrow^{tid} t.\ \downarrow^{sum} a.\ trans(c, t, a) \wedge a > 2000 \rightarrow \Diamond_{[0,3]}\ report(t) \tag{P1}$$

$$\Box \downarrow^{cid} c.\ \downarrow^{tid} t.\ \downarrow^{sum} a.\ trans(c, t, a) \wedge a > 2000 \rightarrow \Box_{(0,5]} \downarrow^{tid} t'.\ \downarrow^{sum} a'.\ trans(c, t', a') \rightarrow a' \leq 2000 \tag{P2}$$

$$\Box \downarrow^{cid} c.\ \downarrow^{tid} t.\ \downarrow^{sum} a.\ trans(c, t, a) \wedge a > 2000 \rightarrow \left((\downarrow^{tid} t'.\ \downarrow^{sum} a'.\ trans(c, t', a') \rightarrow t = t')\ \mathsf{W}\ report(t)\right) \tag{P3}$$

$$\Box \downarrow^{cid} c.\ \downarrow^{tid} t.\ \downarrow^{sum} a.\ trans(c, t, a) \wedge a > 2000 \rightarrow \Box_{[0,6]} \downarrow^{tid} t'.\ \downarrow^{sum} a'.\ trans(c, t', a') \rightarrow \Diamond_{[0,3]}\ report(t') \tag{P4}$$

$$\Box\ transaction \wedge suspicious \rightarrow \Diamond_{[0,3]}\ report \tag{P1′}$$

$$\Box\ transaction \wedge suspicious \rightarrow \Box_{(0,5]}\ transaction \rightarrow \neg suspicious \tag{P2′}$$

$$\Box\ transaction \wedge suspicious \rightarrow \left((transaction \rightarrow \Diamond_{[0,3]}\ report)\ \mathsf{W}\ unflag\right) \tag{P3′}$$

$$\Box\ transaction \wedge suspicious \rightarrow \Box_{[0,6]}\ transaction \rightarrow \Diamond_{[0,3]}\ report \tag{P4′}$$

Fig. 2. Formulas used in the experimental evaluation.

$[mc_I^{L,\hat{H}} \mapsto mc_I^{L,H} \mid L \text{ is an interval in } w']$, respectively. We have that

$$\Theta\big(\delta(\Psi_{w'}^{\psi,J,\nu})\big) = \delta\big(\Theta(\Psi_{w'}^{\psi,J,\nu})\big) = \delta\Big(\zeta\big(\hat{\Theta}(\Psi_{w}^{\psi,\hat{J},\nu})\big)\Big) = \zeta\Big(\theta_w\big(\hat{\Theta}(\Psi_{w}^{\psi,\hat{J},\nu})\big)\Big) = \zeta\Big(\hat{\Theta}\big(\theta_w(\Psi_{w}^{\psi,\hat{J},\nu})\big)\Big)\,. \tag{2}$$

The first and the last equalities hold because a relevant part is determined even after some propositions have been replaced by Boolean constants. The other two equalities follow from the definitions.

We remark that ζ is the substitution applied by AddTimePoint to the relevant parts of $\text{gate}^{\psi,J,\nu}$ when this state variable depends on some proposition p with the interval K. For instance, if $\psi = \alpha\ \mathsf{U}_I\ \beta$ and $\chi = \beta$, then ζ_1 is the substitution applied to the anchor variable and ζ_2 is the substitution applied to the state variable, when $H \subseteq K$, in RefineUntil (cf. Listing 7). Therefore, we obtain the semantic equivalence

$$\Theta(\text{gate}_{w\to w'}^{\psi,J,\nu}) \equiv \zeta\big(\hat{\Theta}(\text{gate}_{w}^{\psi,\hat{J},\nu})\big)\,. \tag{3}$$

This equivalence holds even when $\text{gate}_{w}^{\psi,\hat{J},\nu}$ does not depend on a proposition p with the interval K. In this case, $\text{gate}_{w\to w'}^{\psi,J,\nu} = \text{gate}_{w}^{\psi,\hat{J},\nu}$ and there is no proposition in $\text{gate}_{w}^{\psi,\hat{J},\nu}$ for ζ to substitute.

The right-hand sides of the semantic equivalence in (3) and of the right-most equality in (2) are semantically equivalent by the induction hypothesis. We conclude that the left-hand sides in (3) and of the left-most equality in (2) are also semantically equivalent.

Transformation (T2). Let K be the interval returned by DeltaT2(t). The proof is similar to the (T3.2) case. We only remark that we use $\Psi_{w'}^{\gamma,J,\nu} \equiv [tp^K \mapsto \mathsf{f}](\Psi_{w}^{\gamma,J,\nu})$ in the base case of the corresponding induction.

Transformation (T3.1). The proof is similar to the (T3.2) case and therefore omitted. □

6 EXPERIMENTAL EVALUATION

We have implemented the online algorithms for monitoring from the Sections 4 and 5 in a prototype tool, written in the programming language Go (golang.org). In this section, we experimentally evaluate the performance of our prototype tool, focusing on the impact of different message orderings.

Setup. For our experimental evaluation, we use a standard desktop computer with a 3.3 GHz CPU (Intel Xeon E3-1230V2), 16 GB of RAM, and the Linux operating system (Ubuntu 16.04). The prototype was compiled with the Go compiler 1.10 and executed single threaded. Furthermore, we use the formulas in Figure 2, which vary in their temporal requirements and the data involved. (P1) to (P4) express compliance policies from the banking domain and are variants

of policies that have been used in previous case studies [Basin et al. 2015b]. (P1′) to (P4′) are propositional versions of (P1) to (P4), except (P3′), which has an additional temporal connective and accounts for the additional event *unflag*.

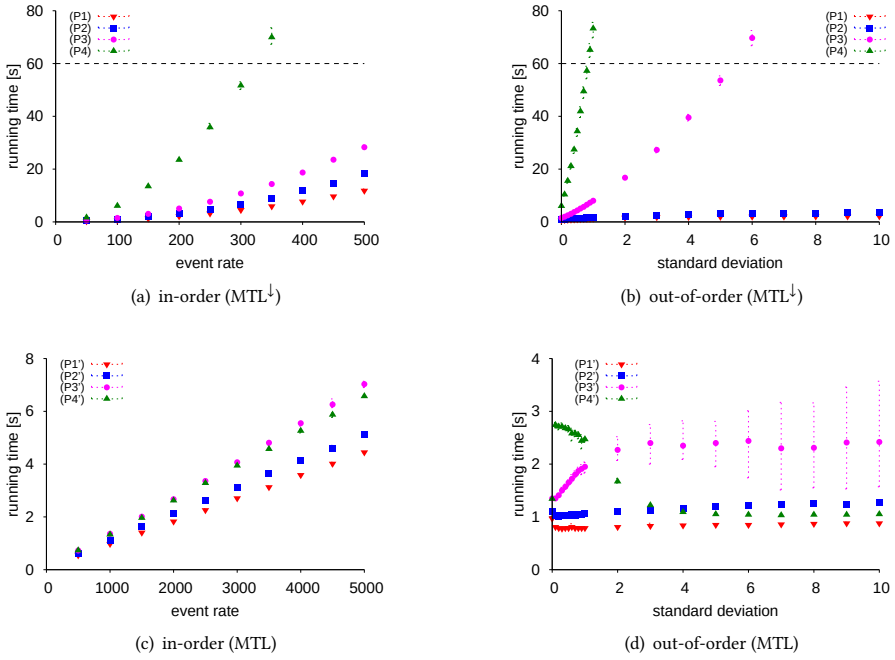
In the following, we provide some intuition on (P1) to (P4). We start by explaining the predicate symbols that model the events that the banking system is assumed to log or transmit to the monitor. The predicate $trans(c, t, a)$ represents the execution of the transaction t of the customer c transferring the amount a of money. The predicate $report(t)$ represents the reporting of the transaction t, that is, t is marked as suspicious. Note that a message sent to the monitor describes an event and the register values. For instance, when executing a transaction, the registers *tid* and *cid* store the identifiers of the transaction and the customer; the amount of the transaction is stored in the register *sum*. For a *report* event, the register *tid* stores the identifier of the transaction whereas the other registers for the customer and the amount store the default value 0.

The formula (P1) requires that a transaction t of a customer c must be reported within at most three seconds if the transferred amount a exceeds the threshold of \$2,000. (P2) to (P4) are variants of (P1). (P2) requires that whenever a customer c makes a transaction that exceeds the threshold, then any of c's future transactions within the next five seconds must not exceed the threshold. (P3) requires that whenever a customer c makes a transaction t that exceeds the threshold, then c is not allowed to make further transactions until the transaction t is reported. Note that the syntactic sugar W ("weak until") is used here instead of the primitive temporal connective U. We do not require that the transaction must eventually be reported. (P4) requires that whenever a customer c makes a transaction that exceeds the threshold, then any of c's transactions within the next six seconds must be reported within three seconds.

Finally, we synthetically generate log files. Each log spans over 60 seconds and contains one event per time point, for instance, corresponding to a single transaction. The number of events in a log is determined by the *event rate*, which is the approximate number of events per second. For each time point i, with $0 \leq i < 60$, the number of events with a timestamp in the time interval $[i, i + 1)$ is randomly chosen within ±10% of the event rate. For instance, a log with event rate 100 comprises approximately 6,000 events. The events and their parameters are randomly chosen such that the number of violations is in a provided range. Note that when the monitor processes an event it performs several state updates, which correspond to the transformations (T1), (T2), and (T3): (1) The monitor adds a new time point with the event's timestamp, (2) it may remove one or more nonsingleton intervals for which the monitor will not receive any events in the future, and (3) it propagates data and truth values. Since the messages can be received in any order by the monitor, it must determine whether all events within a time period have been received. To this end, we attach to each event a sequence number. The monitor removes the nonsingleton interval J if the event's sequence number for the time point before J is the predecessor of the event's sequence for the time point after J. In Section 7, we consider the general setting where the monitor receives events from different sources.

In-order Delivery. In our first setting, messages are received ordered by their timestamps and are never lost. Namely, all events of the log are processed in the order of their timestamps. Figure 3(a) shows the prototype's running times for different event rates. Note that each log spans 60 seconds and a running time below 60 seconds essentially means that the events in the log could have been processed online. The dashed horizontal lines mark this border. Memory usage does not exceed 50 MB, except for (P4) where it increases to around 300 MB.

Out-of-order Delivery. In our second setting, messages can arrive out of order, but they are not lost. We control the degree of message arrival disruption as follows. For the events in a generated log file, we choose their arrival times, which determine the order in which the monitor processes them. An event's arrival time is derived from the event's timestamp by offsetting it by a random delay with respect to the normal distribution with a mean of μ time units and



(a) in-order (MTL$^{\downarrow}$) (b) out-of-order (MTL$^{\downarrow}$)

(c) in-order (MTL) (d) out-of-order (MTL)

Fig. 3. Running times. Here each data point shows the average over five logs together with the minimum and maximum, which are very close to the average, except for (P3′) in (d).

a standard deviation σ. Intuitively, the degree of "out-of-orderness" increases for larger standard deviations. For the degenerate case $\sigma = 0$, the random delay is 0 and the reordered log is identical to the original log. For $\sigma > 0$, the random delay is, for example, between $\mu - \sigma$ and $\mu + \sigma$ with probability 0.68 and with probability 0.95 between $\mu - 2\sigma$ and $\mu + 2\sigma$. This means that for different standard deviations $\sigma, \sigma' > 0$, the random delays for σ are more likely spread over a larger range than for σ' when $\sigma > \sigma'$, which in the end results in reordered logs where the events are less ordered. Finally, we remark that the choice of μ does not impact the event reordering; with a large enough mean μ the random delays are (most likely) positive.

Figure 3(b) shows the prototype's running times on logs with the fixed event rate 100 for different deviations, where μ is fixed to 10 and σ ranges over different values between 0 and 10. For instance, for (P1), the logs are processed in under a second for $\sigma = 0$ and around two seconds for $\sigma = 10$. Memory usage stays moderate for small deviations (below 100 MB for $\sigma < 1$), but can increase significantly for larger deviations (almost 1 GB for (P3) with $\sigma = 5$ and (P4) with $\sigma = 1$). Reasons for this are the larger data structure and also the queued messages, since messages arrive faster than they can be processed by the monitor.

Interpretation. For (P1) to (P4), the running times are nonlinear in the event rate. This is expected from Theorem 3.7. The growth is mainly caused by the data values occurring in the events. A log with a higher event rate contains more different data values and the monitor's state must account for these. In particular, the graph-based data structure contains multiple nodes for a subformula γ and an interval J, but different partial valuations v. As expected, (P1) is the easiest to monitor. In addition to the outermost temporal connective $\Box$, it only has a single temporal connective with a

three second bound and a single block of freeze quantifiers. (P4) is hardest to monitor since it has two blocks of freeze quantifiers and two bounded temporal connectives, which are nested, resulting in a time window of nine seconds. The running times increase when messages are received out of order, which is also expected. For (P1) and (P2), the increase is however almost insignificant. In contrast, for (P3) and (P4), the running times increase rapidly. This can be traced back to the formulas' larger time window and the two blocks of freeze quantifiers.

In the propositional setting, the running times only increase linearly with respect to the event rate and logs are processed significantly faster. Furthermore, the out-of-order delivery of events has only a minor impact of the running times. See the Figures 3(c) and (d), where the event rate is one order of magnitude higher. Our prototype processes most events in a fraction of a millisecond and a noticeable amount of the computation time is actually spent in parsing the events. However, some care must be taken when comparing the figures of the propositional setting with the setting with data values. First, the formulas express different policies. For instance, in (P1′) and (P4′) a report might discharge multiple transactions. Second, the logs for the propositional settings differ from the logs for the formulas (P1) to (P4). In particular, the events in the log files generated for the propositional settings do not account for different customers. Overall, one pays a price at runtime for the expressivity gain given by the freeze quantifier. This price can be traced back to the number of nodes in the graph-based data structure that the monitor maintains. For MTL, the number of nodes in the data structure for an interval is bounded by the number of subformulas, whereas for $\text{MTL}^{\downarrow}$, the number of nodes for an interval is dominated by the different data values that occur in the messages.

To put the experimental results in perspective, we also compare our prototype with the MONPOLY tool [Basin et al. 2012]. MONPOLY's specification language is, like $\text{MTL}^{\downarrow}$, a point-based real-time logic. It is richer than $\text{MTL}^{\downarrow}$ in that it admits existential and universal quantification over domain elements. However, MONPOLY specifications are syntactically restricted in that temporal future connectives must be bounded (except for the outermost connective $\Box$). Thus, (P3) does not have a counterpart in MONPOLY's specification language. MONPOLY handles the counterparts of (P1), (P2), and (P4) significantly faster, up to three orders of magnitude. In the propositional setting, the running times only differ by a factor less than five. Comparing the performance of both tools should, however, be taken with a grain of salt. First, while MONPOLY has undergone several rounds of optimizations, our prototype is fairly unoptimized. More significant, MONPOLY only handles the restrictive setting where messages must be received in-order and MONPOLY outputs violations for specifications with (bounded) future only after all events in the relevant time window are available, whereas our prototype outputs verdicts promptly. For instance, for the formula $\Box_{[0,3]}\, p$, if p does not hold at the time point i with timestamp τ, then our prototype outputs the corresponding verdict directly after processing the time point i, whereas MONPOLY reports this violation at the first time point with a timestamp larger than $\tau + 3$.

In summary, our experimental evaluation shows that one pays a high price to handle an expressive specification language together with message delays. Nevertheless, our prototype's performance is sufficient to monitor systems that generate hundreds of events per second; in a propositional setting, the prototype already handles several thousand events per second. Furthermore, the prototype can be used as a starting point for more efficient implementations.

7 MONITORING APPLICATION

In this section, we describe a deployment of the online algorithms presented for verifying distributed systems at runtime. We first describe the system design and the underlying system assumptions. We also discuss some practical aspects and consequences of our deployment.

7.1 Deployment

We target distributed systems with multiple interacting components. The objective is to determine at runtime whether the system's behavior, as observed and reported by the components, satisfies or violates a given specification φ at some or all time points.

We sketch our system design, which extends the original system with an additional monitoring component for φ, where φ is a closed MTL$^{\downarrow}$ formula. The original system components are instrumented such that they report their performed actions to the monitoring component by sending dedicated messages over a unidirectional channel. Each such message also names the performing component and the time. Furthermore, the message contains a sequence number. That is, each component maintains a counter, which counts the actions it has performed so far, and includes the counter's value with every message sent to the monitor. With these numbers, the monitor can determine if no action has been performed in a given interval (see Section 7.2.1 for details). In addition to the messages that describe the performed actions, a component can send "alive" messages. They inform the monitoring component that the respective component has not performed any action for a while. In summary, there are two types of messages: $\mathit{action}(C, \tau, s, d)$ and $\mathit{alive}(C, \tau, s)$, where C is the component name, $\tau \in \mathbb{Q}_{\geq 0}$ the timestamp, s the component's sequence number, and d a description of the performed action.

Before providing further details and discussing the consequences of this deployment, we list and comment on the assumptions on the underlying system model.

A1*: The system is static.* This means that no system components are created or removed at runtime. Furthermore, the monitor is aware of the existence of all the system components. Note that this assumption can easily be eliminated by building into our approach a mechanism to register components before they become active and unsubscribing them when they become inactive. To register components we can, for example, use a simple protocol where a component sends a registration request and waits until it receives a message that confirms the registration.

A2*: Communication between components is asynchronous and unreliable. However, messages are neither tampered with nor delivered to wrong components.* Asynchronous, unreliable communication means that messages may be received in an order different from which they were sent, and some messages may be lost and therefore never received. Note that message loss covers the case where a system component crashes without recovery. A component that stops executing is indistinguishable to other processes from one that stops sending messages or none of its messages are received. We explain in Section 7.2.3 that it is also straightforward to handle the case where crashed components can recover. The assumption ruling out tampering and improper delivery can be discharged in practice by adding information to each message, such as a recipient identifier and a cryptographic hash value, which are checked when receiving the message.

A3*: System components, including the monitor, are trustworthy.* This means, in particular, that the components correctly report their observations and do not send bogus messages.

A4*: Reported actions are consistent.* This means that messages from components to the monitor do not contradict each other. For instance, there are never two messages to the monitor such that one is saying that a proposition p is true at a time $\tau \in \mathbb{Q}_{\geq 0}$ and the other one is saying that p is false at τ.

A5*: The system components perform infinitely many actions in the limit.* This guarantees that the observable system behavior is in the limit a timed word. Note that MTL$^{\downarrow}$ specifies properties about infinite system behavior. In particular, MTL$^{\downarrow}$'s three-valued semantics over observations approximates infinite behavior as the interval of an observation's

last letter is unbounded and can always be refined. We would need to use another specification language if we want to express properties about finite system behavior. However, note that a monitor is always aware of only a finite part of the observed system behavior. Furthermore, since channels are unreliable and messages can be lost, a monitor might even, in the limit, be aware only of a finite part of the infinite system behavior.

7.2 Discussion

7.2.1 State Updates. Each message may result in multiple updates of the monitor's state. A message $action(C, \tau, s, d)$ results in adding a time point with the timestamp τ, the propagation of data and truth values, and also the removal of nonsingleton intervals. A message $alive(C, \tau, s)$ may result in the removal of nonsingleton intervals, which in turn may trigger the propagation of truth values.

With the messages' sequence numbers, the monitor can infer which intervals can be removed. When monitoring a single system component, this inference is obvious. We sketch the general case when monitoring a system with the components $C_0, \ldots, C_m$. Let $J_0, \ldots, J_n$ be the nonsingleton intervals of the letters in an observation. The monitor labels each of these intervals with a set S_{J_j} of the components from which it may receive an *action* message with a timestamp in J_j in the future. Additionally, the monitor maintains for each component C_i triples of the form (s, I, s'), where I is an interval and $s, s' \in \mathbb{N}$ with $s \leq s'$. The intuition is that all *action* messages from C_i with a timestamp in I have been received by the monitor, and s and s' are the smallest and largest sequence number of these messages, respectively. The monitor adds a triple $(t, \{\tau\}, t)$ when receiving from C_i a message with the timestamp τ and the sequence number t. The monitor also merges triples when possible. For example, the triples (s, I, s') and $(t, \{\tau\}, t)$ with $t = s - 1$ or $t = s' + 1$ are merged into the triple $(\min\{t, s\}, I \cup \{\tau\}, \max\{t, s'\})$, where $I \cup \{\tau\}$ is the smallest interval that contains I and $\{\tau\}$. Whenever one of the intervals J_j is a subset of the interval of such a triple, the monitor removes C_i from the set S_{J_j}. When S_{J_j} becomes empty, the monitor removes the letter with the interval J_j from the observation. Note that the intervals $J_0, \ldots, J_n$ can be ordered and stored in a balanced search tree. Analogously, the triples can be ordered and also stored in balanced search trees with pointers to their predecessors and successors.

7.2.2 Accuracy of Timestamps. The monitor's verdicts are computed with respect to the information in messages that the monitor receives from the system components. Even though we assume trustworthy system components (A3), their observations might not match with the actual system behavior. In particular, the timestamp τ in a message may be inaccurate because τ comes from the clock of a system component that has drifted from the actual time. One may wonder in what sense are the verdicts meaningful.

Consider first the guarantees we have under the additional system assumption that timestamps are precise and from the domain $\mathbb{Q}_{\geq 0}$. Under this assumption, $w_i \sqsubseteq w$, for all $i \in \mathbb{N}$, where the w_i's are observation describing the reported system behavior and w is a timed word represents the real system behavior. It follows from Theorem 3.5 that the verdicts computed from the reported system behavior w_i are also valid for the system behavior w.

Assuming precise timestamps is however a strong assumption, which does not hold in practice since real clocks are imprecise. Moreover, each system component uses its local clock to timestamp its messages and these clocks might differ due to clock drifts. In fact, assuming synchronized clocks boils down to having a synchronized system at hand. Nevertheless, we argue that for many kinds of specifications and systems, relying on timestamps from existing clocks in monitoring is good enough in practice. First, under stable conditions (like temperature), state-of-the-art hardware clocks already achieve a high accuracy and their drifts are, even over a longer time period, rather small [Cristian and Fetzer 1999]. Moreover, there are protocols like the Network Time Protocol (NTP) (see www.ntp.org) for synchronizing

clocks in distributed systems that work well in practice. For local area networks, NTP can maintain synchronization of clocks within one millisecond [Mills 1995]. Overall, with state-of-the-art techniques, we can obtain timestamps that are "accurate enough" for many monitoring applications, for instance, for checking whether deadlines are met when the deadlines are in the order of seconds or even milliseconds. Furthermore, if the monitored system guarantees an upper bound on the imprecision of timestamps, we can often account for this imprecision in the specification. For example, for checking at runtime that requests are acknowledged within 100 milliseconds, when the imprecision between two clocks is always less than a millisecond, we can use the formula $\Box\, req \rightarrow \blacklozenge_{[0,1)} \Diamond_{[0,101)} ack$ to avoid false alarms.

7.2.3 Component Crashes. When a system component crashes, its state is lost. For recovery, we must bring the component into a state that is safe for the system. To safely restart a system component that is not the monitor, we must restore its sequence number. We can use any persistent storage available to store this number. In case the component crashes while storing this number, we can increment the restored number by one. This might results in knowledge gaps for the monitor, since some intervals will never be identified as complete. However, the computed verdicts are still sound. For the recovery of a crashed monitor, we just need to initialize it. A recovered monitor corresponds to a monitor that has not yet received any message. This is safe in the sense that the recovered monitor will only output sound verdicts. When the monitor also logs received messages in a persistent storage, it can replay them to close some of its knowledge gaps. Note that the order in which these messages are replayed is irrelevant and they can even be replayed whenever the recovered monitor is idle.

8 RELATED WORK

In this section, we examine related work. Our focus is on system verification, in particular, runtime verification, a well-established area for checking at runtime whether a system's execution fulfills a given specification. We structure our discussion along the aspects of multiple truth values, data values, and distributed systems.

Multi-valued Semantics. Multi-valued semantics for temporal logics are widely used in runtime verification, see for example, [Bauer and Falcone 2016; Bauer et al. 2011; Mostafa and Bonakdarbour 2015; Scheffel and Schmitz 2014]. Their semantics extend the classical LTL semantics by also assigning non-Boolean truth values to finite prefixes of infinite words [Bauer et al. 2010]. The additional truth values differentiate when neither some nor all extensions of a finite word satisfy a formula. However, in contrast to the presented three-valued semantics of $\text{MTL}^{\downarrow}$ used in this paper, the Boolean and temporal connectives are not extended over the additional truth values. Furthermore, the partial order $\prec$ on the truth values, which orders them in knowledge, is not considered. Note that having the third truth value $\bot$ at the logic's object level and the partial order $\prec$ is at the core of our three-valued semantics for $\text{MTL}^{\downarrow}$ and our monitoring approach; namely, it is used to account for a monitor's knowledge gaps. Another difference is that a formula's truth value is not defined by the possible extensions of a finite word. As pointed out in Remark 3.13, including the possible extensions can render monitoring infeasible.

The monitoring approaches by Garg et al. [2011] and Basin et al. [2013], both targeting the auditing of policies on system logs, also account for knowledge gaps, i.e., logs that may not contain all the actions performed by a system. Both approaches handle rich policy specification languages with first-order quantification and a three-valued semantics. Garg et al.'s approach [2011], which is based on formula rewriting, is however, not suited for online use since it does not process logs incrementally. It also only accounts for knowledge gaps in a limited way, namely, the interpretation of a predicate symbol cannot be partially unknown, e.g., for certain time periods. Furthermore, their approach is not complete. Basin et al.'s approach [2013], which is based on their prior work [Basin et al. 2015b], can be used online. However, the

problem of how to output verdicts incrementally as prior knowledge gaps are resolved is not addressed, and thus it does not deal with out-of-order events. Moreover, the semantics of the specification language handled does not reflect a monitor's partial view about the system behavior. Instead, it is given for infinite data streams that represent system behavior in the limit. The runtime-verification approach by Stoller et al. [2011] also accounts for gaps in traces. These gaps are however caused by sampling the state of the monitored system to reduce the runtime-verification overhead and trace elements are processed ordered. Furthermore, their approach is not based on a multi-valued semantics for a temporal logic. Instead, an a priori trained model (namely, a hidden Markov model) for estimating the likelihood of missing trace elements is used to compute the probability of the specification's satisfaction.

Multi-valued semantics for temporal logics have been also considered in other areas of system verification. For instance, Chechik et al. [2003] describe a model-checking approach for a multi-valued extension for the branching-time temporal logic CTL. Their CTL extension is similar to our MTL$^{\downarrow}$ extension in that it allows one to reason about uncertainty at the logic's object level. However, the task they consider is different from ours. Namely, in model checking, the system model is given—usually finite-state—and correctness is checked offline with respect to the model's described executions; in contrast, in runtime verification, one checks online the correctness of the observed system behavior. The three-valued semantics for LTL provided by Godefroid and Piterman [2011] is also related to our three-valued semantics for MTL$^{\downarrow}$. It is, however, based on infinite words, not observations (Definition 3.1). Similar to (T3) of Definition 3.1, a proposition with the truth value $\bot$ at a position can be refined by t or f. In contrast, their semantics does not support refinements that add and delete letters, cf. (T1) and (T2) of Definition 3.1.

Data Values. Havelund et al. [2018] overview and compare different runtime-verification approaches that allow one to reason online about data values in event streams. Among them are parametric runtime-verification approaches [Barringer et al. 2012; Roşu and Chen 2012] and approaches that handle first-order extensions of temporal logics [Basin et al. 2015b; Hallé and Villemaire 2012]. Those approaches share some similarities to our approach, in particular, how the freeze quantifier is used to reason about data values. As explained in Example 2.1, the freeze quantifier can be seen as a weak form of the standard first-order quantifiers. Although the first-order extensions are more expressive than MTL$^{\downarrow}$, the expressiveness of MTL$^{\downarrow}$ seems sufficient for many runtime-verification applications because the data values often appear uniquely in the events. Handling specification languages with first-order quantification like MFOTL [Basin et al. 2015b] in settings with only partial knowledge and out-of-order event streams is nontrivial and various restrictions seem to be necessary [Basin et al. 2013]. In a nutshell, in parametric runtime verification, one slices a single event stream according to the events' data values in multiple streams, which are then monitored separately and checked against propositional specifications [Roşu and Chen 2012] or nonpropositional specifications, as for instance, quantified event automata [Barringer et al. 2012]. The bindings of the data values within the sliced event streams are implicit in most of those approaches and the slicing criteria is hard-coded in the monitoring algorithm. In contrast, the freeze quantifier explicitly binds the data values to logical variables. Furthermore, our monitoring algorithm for MTL$^{\downarrow}$ processes a single event stream.

Feng et al. [2017] define a similar extension of MTL with the freeze quantifier as in MTL$^{\downarrow}$. Their analysis focuses on the computational complexity of the path-checking problem. However, they use a finite trace semantics, which is less suitable for runtime verification. Brim et al. [2014], and Ryckbosch and Diwan [2014] also provide extensions of LTL with the freeze quantifier together with monitoring algorithms. Note that the rule-based runtime-verification approach EAGLE [Barringer et al. 2004] already allowed one, similar to the freeze quantifier, to freeze data values in events to variables. Neither Feng et al. [2017], Brim et al. [2014], Ryckbosch and Diwan [2014], nor Barringer et al. [2004] consider

out-of-order messages and knowledge gaps in event streams. Finally, Demri and Lazić [2009] analyze the complexity of the satisfiability problem of LTL extended with the freeze quantifier. In particular, they provide translations of restricted fragments to register automata. Applications to runtime verification are not explored.

Distributed Systems. Several runtime-verification approaches have been developed for distributed systems. Francalanza et al. [2018] provide an overview and we limit ourselves here to those approaches that are closely related to ours. Overall, all of them make different assumptions on the system model and thus target different kinds of distributed systems. Furthermore, they handle different specification languages. We are not aware of any approach in the literature that handles specifications with real-time constraints or accounts for network failures.

Colombo and Falcone [2016] propose a runtime-verification approach, based on formula rewriting, that also allows the monitor to receive messages out of order. Their approach only handles the propositional temporal logic LTL with the three-valued semantics proposed by Bauer et al. [2010]. In a nutshell, their approach unfolds temporal connectives as time progresses and special propositions act as placeholders for subformulas. The subsequent assignment of these placeholders to Boolean truth values triggers the reevaluation and simplification of the formula. Their approach only guarantees soundness but not completeness, since the simplification rules used for formula rewriting are incomplete. Finally, its performance with respect to out-of-order messages is not evaluated.

Sen et al. [2004] use an LTL variant with epistemic operators to express distributed knowledge. The verdicts output by the monitors are correct with respect to the local knowledge the monitors obtained about the systems' behavior. Since their LTL variant only has temporal connectives that refer to the past, only safety properties are expressible. Scheffel and Schmitz [2014] extend this work to handle also some liveness properties by working with a richer fragment of LTL that includes temporal connectives that refer to the future. The algorithm by Bauer and Falcone [2016] assumes a lock-step semantics and thus only applies to synchronous systems. Falcone et al. [2014] weaken this assumption. However, each component must still output its observations at each time point, which is determined by a global clock. The observations are then received by the monitors at possibly later time points. The algorithm by Mostafa and Bonakdarbour [2015] assumes lossless FIFO channels for asynchronous communication. Logical clocks are used to partially order messages.

Miscellaneous. The problem of processing streams in which events may appear out-of-order has also considered in other contexts than runtime verification, namely, in stream processing. For example, Srivastava and Widom [2004] use buffering and heartbeats so that continuous queries are evaluated correctly under the assumption that the heartbeats are sufficiently large. Various parameters are considered to generate the heartbeats. However, queries are not processed promptly, but always with a delay. Li et al. [2008] propose a stream-processing architecture with a global mechanism that reports progress and allows one to finalize a partial evaluation of a query on a time window. Events are processed promptly. The messages' sequence numbers, which we use to determine whether the monitor may be missing a message from a system component in some time period, can been seen as such a global mechanism.

9 CONCLUSION

We have presented a runtime-verification approach based on three truth values to checking real-time specifications given as $\text{MTL}^{\downarrow}$ formulas. Our approach targets distributed systems and handles the practically-relevant setting where the messages sent to monitors can be delayed, reordered, or lost, and it provides soundness and completeness guarantees. Although our experimental evaluation is promising, our approach does not yet scale to monitoring systems that generate thousands or even millions of events per second. This requires additional research, including algorithmic optimizations.

We plan to investigate this in future work, as well as to deploy and evaluate our approach in realistic, large-scale case studies.

ACKNOWLEDGMENTS

This work received funding from the European Union's Horizon 2020 research and innovation programme under the grant agreement No 779852.

REFERENCES

Rajeev Alur and Thomas A. Henzinger. 1992. Logics and Models of Real Time: A Survey. In *Proceedings of the 1991 REX Workshop on Real Time: Theory in Practice (Lect. Notes Comput. Sci.)*, Vol. 600. Springer, Berlin, Heidelberg, 74–106.

Howard Barringer, Yliès Falcone, Klaus Havelund, Giles Reger, and David E. Rydeheard. 2012. Quantified Event Automata: Towards Expressive and Efficient Runtime Monitors. In *Proceedings of the 18th International Symposium on Formal Methods (FM) (Lect. Notes Comput. Sci.)*, Vol. 7436. Springer, Berlin, Heidelberg, 68–84.

Howard Barringer, Allen Goldberg, Klaus Havelund, and Koushik Sen. 2004. Rule-Based Runtime Verification. In *Proceedings of the 5th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI) (Lect. Notes Comput. Sci.)*, Vol. 2937. Springer, Berlin, Heidelberg, 44–57.

David Basin, Matúš Harvan, Felix Klaedtke, and Eugen Zălinescu. 2012. MONPOLY: Monitoring Usage-control Policies. In *Proceedings of the 2nd International Conference on Runtime Verification (RV) (Lect. Notes Comput. Sci.)*, Vol. 7186. Springer, Berlin, Heidelberg, 360–364.

David Basin, Felix Klaedtke, Srdjan Marinovic, and Eugen Zălinescu. 2013. Monitoring Compliance Policies over Incomplete and Disagreeing Logs. In *Proceedings of the 3rd International Conference on Runtime Verification (RV) (Lect. Notes Comput. Sci.)*, Vol. 7687. Springer, Berlin, Heidelberg, 151–167.

David Basin, Felix Klaedtke, Samuel Müller, and Eugen Zălinescu. 2015b. Monitoring Metric First-Order Temporal Properties. *J. ACM* 62, 2, Article 15 (2015), 45 pages.

David Basin, Felix Klaedtke, and Eugen Zălinescu. 2015a. Failure-aware Runtime Verification of Distributed Systems. In *Proceedings of the 35th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS) (Leibniz International Proceedings in Informatics (LIPIcs))*, Vol. 45. Leibniz Center for Informatics, Schloss Dagstuhl, 590–603.

David Basin, Felix Klaedtke, and Eugen Zălinescu. 2017. Runtime Verification of Temporal Properties over Out-of-order Data Streams. In *Proceedings of the 29th International Conference on Computer Aided Verification (CAV) (Lect. Notes Comput. Sci.)*, Vol. 10426. Springer, Cham, 356–376.

Andreas Bauer and Yliès Falcone. 2016. Decentralised LTL Monitoring. *Form. Methods Syst. Des.* 48, 1–2 (2016), 46–93.

Andreas Bauer, Jan-Christoph Küster, and Gil Vegliach. 2015. The ins and outs of first-order runtime verification. *Form. Methods Syst. Des.* 46, 3 (2015), 286–316.

Andreas Bauer, Martin Leucker, and Christian Schallhart. 2010. Comparing LTL Semantics for Runtime Verification. *J. Logic Comput.* 20, 3 (2010), 651–674.

Andreas Bauer, Martin Leucker, and Christian Schallhart. 2011. Runtime Verification for LTL and TLTL. *ACM Trans. Softw. Eng. Meth.* 20, 4, Article 14 (2011), 64 pages.

Lubos Brim, Petr Dluhos, David Safránek, and Thomas Vejpustek. 2014. STL*: Extending signal temporal logic with signal-value freezing operator. *Inf. Comput.* 236 (2014), 52–67.

Marsha Chechik, Benet Devereux, Steve Easterbrook, and Arie Gurfinkel. 2003. Multi-valued symbolic model-checking. *ACM Trans. Softw. Eng. Meth.* 12, 4 (2003), 371–408.

Christian Colombo and Yliès Falcone. 2016. Organising LTL Monitors over Distributed Systems with a Global Clock. *Form. Methods Syst. Des.* 49, 1 (2016), 109–158.

Flaviu Cristian and Christof Fetzer. 1999. The Timed Asynchronous Distributed System Model. *IEEE Trans. Parallel Distrib. Syst.* 10, 6 (1999), 642–657.

Stéphane Demri and Ranko Lazić. 2009. LTL with the Freeze Quantifier and Register Automata. *ACM Trans. Comput. Log.* 10, 3, Article 16 (2009), 30 pages.

Yliès Falcone, Tom Cornebize, and Jean-Claude Fernandez. 2014. Efficient and Generalized Decentralized Monitoring of Regular Languages. In *Proceedings of the 34th IFIP WG 6.1 International Conference on Formal Techniques for Distributed Objects, Components, and Systems (FORTE) (Lect. Notes Comput. Sci.)*, Vol. 8461. Springer, Berlin, Heidelberg, 66–83.

Shiguang Feng, Markus Lohrey, and Karin Quaas. 2017. Path Checking for MTL and TPTL over Data Words. *Log. Methods Comput. Sci.* 13, 3, Article 19 (2017), 34 pages.

Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. 1985. Impossibility of Distributed Consensus with One Faulty Process. *J. ACM* 32, 2 (1985), 374–382.

Adrian Francalanza, Jorge A. Pérez, and César Sánchez. 2018. Runtime Verification for Decentralised and Distributed Systems. In *Lectures on Runtime Verification - Introductory and Advanced Topics*, Ezio Bartocci and Yliès Falcone (Eds.). Lect. Notes Comput. Sci., Vol. 10457. Springer, Cham, Chapter 6, 176–210.

Deepak Garg, Limin Jia, and Anupam Datta. 2011. Policy Auditing over Incomplete Logs: Theory, Implementation and Applications. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS)*. ACM Press, New York, 151–162.

Patrice Godefroid and Nir Piterman. 2011. LTL generalized model checking revisited. *Int. J. Softw. Tools Technol. Trans.* 13, 6 (2011), 571–584.

Sylvain Hallé and Roger Villemaire. 2012. Runtime Enforcement of Web Service Message Contracts with Data. *IEEE Trans. Serv. Comput.* 5, 2 (2012), 192–206.

Klaus Havelund, Giles Reger, Daniel Thoma, and Eugen Zălinescu. 2018. Monitoring Events that Carry Data. In *Lectures on Runtime Verification - Introductory and Advanced Topics*, Ezio Bartocci and Yliès Falcone (Eds.). Lect. Notes Comput. Sci., Vol. 10457. Springer, Cham, Chapter 3, 61–102.

Thomas A. Henzinger. 1990. Half-order modal logic: how to prove real-time properties. In *Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing (PODC)*. ACM Press, New York, 281–296.

Stephen C. Kleene. 1950. *Introduction to Metamathematics*. D. Van Nostrand, Princeton.

Ron Koymans. 1990. Specifying Real-Time Properties with Metric Temporal Logic. *Real-Time Syst.* 2, 4 (1990), 255–299.

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

Jin Li, Kristin Tufte, Vladislav Shkapenyuk, Vassilis Papadimos, Theodore Johnson, and David Maier. 2008. Out-of-order processing: a new architecture for high-performance stream systems. *Proceedings of the VLDB Endowment* 1, 1 (2008), 274–288.

Oded Maler and Dejan Nickovic. 2004. Monitoring Temporal Properties of Continuous Signals. In *Proceedings of the Joint International Conferences on Formal Modelling and Analysis of Timed Systems (FORMATS) and on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT) (Lect. Notes Comput. Sci.)*, Vol. 3253. Springer, Berlin, Heidelberg, 152–166.

Patrick O'Neil Meredith, Dongyun Jin, Dennis Griffith, Feng Chen, and Grigore Roşu. 2012. An overview of the MOP runtime verification framework. *Int. J. Softw. Tools Technol. Trans.* 14, 3 (2012), 249–289.

David L. Mills. 1995. Improved algorithms for synchronizing computer network clocks. *IEEE/ACM Trans. Netw.* 3, 3 (1995), 245–254.

Menna Mostafa and Borzoo Bonakdarbour. 2015. Decentralized Runtime Verification of LTL Specifications in Distributed Systems. In *Proceedings of the 29th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE Computer Society, Los Alamitos, 494–503.

Joël Ouaknine and James Worrell. 2006. On Metric Temporal Logic and Faulty Turing Machines. In *Proceedings of the 9th International Conference on Foundations of Software Science and Computation Structures (FOSSACS) (Lect. Notes Comput. Sci.)*, Vol. 3921. Springer, Berlin, Heidelberg, 217–230.

Grigore Roşu and Feng Chen. 2012. Semantics and Algorithms for Parametric Monitoring. *Log. Methods Comput. Sci.* 8, 1, Article 9 (2012), 47 pages.

Frederick Ryckbosch and Amer Diwan. 2014. Analyzing performance traces using temporal formulas. *Softw. Pract. Exper.* 44, 7 (2014), 777–792.

Torben Scheffel and Malte Schmitz. 2014. Three-valued asynchronous distributed runtime verification. In *Proceedings of the 12th ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMCODE)*. IEEE Computer Society, Los Alamitos, 52–61.

Koushik Sen, Abhay Vardhan, Gul Agha, and Grigore Roșu. 2004. Efficient decentralized monitoring of safety in distributed systems. In *Proceedings of the 26th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, Los Alamitos, 418–427.

A. Prasad Sistla and Edmund M. Clarke. 1985. The complexity of propositional linear temporal logics. *J. ACM* 32, 3 (1985), 733–749.

Utkarsh Srivastava and Jennifer Widom. 2004. Flexible Time Management in Data Stream Systems. In *Proceedings of the 23rd ACM Symposium on Principles of Database Systems (PODS)*. ACM Press, New York, 263–274.

Scott D. Stoller, Ezio Bartocci, Justin Seyster, Radu Grosu, Klaus Havelund, Scott A. Smolka, and Erez Zadok. 2011. Runtime Verification with State Estimation. In *Proceedings of the 2nd International Conference on Runtime Verification (RV) (Lect. Notes Comput. Sci.)*, Vol. 7186. Springer, Berlin, Heidelberg, 193–207.