Runtime Verification Under Access Restrictions

Rania Taleb, Raphaël Khoury, Sylvain Hallé

rania.taleb1@uqac.ca, raphael.khoury@uqac.ca, shalle@acm.org
Laboratoire d'informatique formelle
Université du Québec à Chicoutimi, Canada

***Abstract*—We define a logical framework that permits runtime verification to take place when a monitor has incomplete or uncertain information about the underlying trace. Uncertainty is modeled as a stateful access control proxy that has the capacity to turn events into sets of possible events, resulting in what we call a "multi-trace". We describe a model of both proxy and monitor as extensions of Mealy machines, and provide an algorithm to lift a classical monitor into a sound, loss-tolerant monitor. Experiments on various scenarios show that the approach can account for various types of data degradation and access limitations, provides a tighter verdict than existing works in some cases, and preserves scalable performance of the model.**

***Index Terms*—stateful, proxy, multi-trace, loss-tolerant monitor, degradation, access limitation**

I. Introduction

Runtime Verification (RV) is the process of observing a sequence of events produced by a running software system, and determining whether this sequence satisfies a given property expressed in a formal notation [15]. Depending on the context, the source of events given to the *monitor* may come from instrumentation instructions manually inserted inside the system's code, logging statements normally produced by the system, external readings from devices such as sensors, packets sniffed from communication links, or a combination thereof. Despite the variety of event types and event sources that can be used for analysis, a widely held assumption in RV is that the monitor has complete and error-free access to the set of events against which to evaluate a given property [7], [10], [11].

However, it has been acknowledged that this assumption is not completely warranted, as there exist multiple situations where the monitor may operate with some level of uncertainty about the content of the underlying trace. As a matter of fact, a recent Dagstuhl seminar report has emphasized the importance of dealing with incomplete, imprecise and faulty sources of events [3], as did a recent survey of challenges related to Runtime Verification [17].

In this paper, we present a formal model to account for access restrictions in a monitoring context. We group under the term "access restrictions" the conditions that cause a source of events to become imperfectly known by a monitor. First, in Section II, we enumerate various such situations, including new use cases not studied in existing literature. We also review the few works that have already studied the problem of monitoring over incomplete event sources or event sources containing uncertain events. These approaches focus on designing monitoring algorithms that are *tolerant* to missing or uncertain events; however, we shall see that they present some limitations in the kind of information degradation they can account for.

In Section III, we define an abstract model of access restrictions over event traces. Our contribution stands out from related works in that it incorporates a formal model of the degradation of events, in the form of an *access proxy*, interposed between a source of events and a monitor. Each concrete event is modeled as a completely defined "possible world"; the action of the proxy has the effect of potentially turning a unique world into a *set* of such worlds, or deleting events altogether. Obviously, the presence of the proxy and the degradation it causes on the input events have an impact on the verdict produced by the monitor: for instance, it can result in multiple possible verdicts, a phenomenon we call *ambiguity*. A first advantage of our model is that it makes possible, for a given input trace and a monitor, to study the effect of various kinds of degradation on the monitor's verdict. It is also flexible: the manipulations made to the input trace can be stateful (i.e. the alteration applied to an event, if any, may depend on the past), and the "multi-events" resulting from an input event can account for various types of data degradation and access limitations, including some that cannot be modeled by existing related works.

Yet another advantage of this abstract model is that, contrary to existing works, it is agnostic to the concrete way in which the proxy and the monitor are specified. In Section IV, we present one such possible way, by defining an extension of Mealy machines where symbols for transitions and outputs are replaced by logical formulas. We describe a construction that lifts a loss-tolerant "multi-monitor" from a classical monitor, and show that it runs in linear time in the size of the trace and the size of the underlying monitor. In the case where an imprecise trace leads to more than one possible verdict, it quantifies the likelihood of each possible verdict on-the-fly.

These concepts have been concretely implemented as an extension to an event stream processing engine called BeepBeep [9], which is described in Section V. Experiments with a number of different scenarios show that the multi-monitor adds constant memory overhead and linear time overhead over an input trace, which means that it can scale to large traces and large monitors (10^6 events and more than 10^9 states).

Furthermore, we show that some types of data degradation can only be accounted for in related works by an over-approximation of uncertainty, which has a significant negative impact on the precision of a monitor's verdict and its performance, compared to the finer modeling presented in this paper. Finally, this model opens the door to numerous exciting theoretical questions, which we briefly enumerate in Section VI as future work.

II. RV with Partial Information

Typically, Runtime Verification (RV) is used for testing purposes by verifying formal properties on the execution of a software system. A program is instrumented to periodically relay information about its state to a monitor; the data objects produced by this instrumentation are called events. The properties that are monitored generally involve conditions on the sequence of such events, as well as the data inside these events. This analysis can take place on-the-fly while the system executes ("online" monitoring), or occur *a posteriori* on pre-recorded logs of the system ("offline" monitoring). RV is a generic process that has been applied on various sources of events: system calls, method calls, database logs, sensor readings, and logging statements manually inserted by a developer.

A. Causes of Incomplete and Uncertain Events in Logs

We first shed some light on the most commonly occurring causes and situations that lead to the logging of incorrect information or to the corruption of the existing information in the logs, in a way that affects the monitoring process.

1) Log Corruption: The first obvious cause for the presence of incomplete and uncertain log information is the fortuitous corruption of logs for various reasons, such as hardware failure. Mechanisms such as error correcting codes can be used to detect the presence of corrupted data. However, in some cases, the error can only be detected, but not corrected. A monitor receiving such a piece of corrupted log could, for example, know that some event occurred, without being able to ascertain for sure what event happened due to its corrupted nature. The same can be said of values within an event: some parameters in an event could be detected as missing or corrupted using the same mechanism.

A stronger form of corruption happens when an event, or an interval of events, is dropped from the stream, for example because of a temporary disruption of a communication link. In such a case, assuming that each event is given a unique and incrementing ID, a monitor connected to the event source can uncover the occurrence of such a drop by the presence of non-successive IDs. This makes it possible to determine how many events occurred, but not what these events were.

2) Incorrect Instrumentation or Logging: In the case where the source of events is the execution of a software system, the correct monitoring of a property is dependent on the fact that the system provides all the relevant events to the monitor. Indeed, Bartocci *et al.*, in their introduction to Runtime Verification [4], describe various instrumentation techniques, but assume this instrumentation to be complete and correct.

However, this hypothesis may be unwarranted in cases when logging statements are manually inserted by the developers [21]; moreover, each logging statement is typically assigned a severity level, and studies have shown that usage of these levels by developers can be highly unreliable [16]. Consequently, a monitor that analyzes a log containing only events of a specific level can miss statements that are relevant to the property being monitored. Such manually-generated logging statements can also be imprecise in themselves, with multiple distinct events being assigned the same general error message.

3) Imprecision and Uncertainty: The previous example brings forward the fact that an event source can be imprecise and/or uncertain. For example, a temperature reading produced by a sensor can be associated with an error range, such as $T = 20° \pm 0.5$. In this case, a monitor evaluating a property that produces different outcomes depending on whether $T \leq 20$ or $T > 20$ may produce an incorrect verdict for a range of values of T. A finer description of imprecision can be employed, such as the work of Heintz *et al.* [19] where the altitude a of the drone is modeled as a probability distribution instead of a flat range. In such a model, definite statements such as $a > 3$ must be turned into statements about a probability, such as $Pr(a > 3)$.

4) Load Shedding and Throttling: In many cases, feeding an event to a monitor involves an additional amount of work for the executing system. A possible solution is to make *load shedding* which consists of the deliberate deletion of events in order to reduce resource consumption. For example, Joshi *et al.* [12] describe a scenario where a media player software is instrumented with a library given a fixed time budget. In a given time interval T, the instrumentation can produce at most B events; any event exceeding this threshold within this interval is replaced by a special "non-event" called χ.

5) Impedance Mismatch: Finally, incomplete knowledge about event parameters can also be caused by the fact that the monitor watches a property expressed using parameters that do not perfectly align with those provided by the event source it is connected to. For example, a property φ may involve conditions on individual values of parameters x and y, while the source of events, for whatever reason, can only provide their sum s. We call this situation *impedance mismatch*. This may be the case if the log to be analyzed contains information that is of confidential nature, such as database transaction logs, or events coming from an instrumented program whose parameters reveal sensitive information. In such a case, access control mechanisms may block the reading of some individual data elements, but allow queries on *aggregations* of multiple elements. Here, s can be seen as an aggregation of the individual values of x and y. This can also occur in the case of log repurposing, when one wishes to evaluate a new property over a log that has been recorded in the past for another purpose, and contains attributes that do not directly align with those expressed in the property. A final cause of mismatch can be found in the manual logging instructions discussed above: for example, the distinct events "open socket" and "open file" could both have the message "open", making them undistinguishable.

B. Existing Approaches

Bartocci *et al.* [5] and Stroller *et al.* [18] proposed to use statistical models to learn the underlying application's expected behavior, and use it to add values to the trace when they are found missing. Such an analysis could form the basis for the creation of the proxies introduced in Section III, allowing both methods to work in tandem. In such a case, the segments of the trace for which this model guesses values correspond to the parts of the trace where the proxy produces multi-events instead of uni-events. Kalajdzic *et al.* [13] likewise use a statistical model to monitor a trace that contains "observation gaps" during which the program was not monitored.

Wang *et al.* [20] consider a model where changes of values for multiple variables are recorded independently. In this model, the values are assumed correct, and the uncertainty lies in their actual interleaving. For their part, Aceto *et al.* [1] study the monitoring of systems that exhibit silent actions, a common feature of formal specifications that captures the occurrence of an internal action, whose specific nature is not observable by the monitor. They study the monitorability of branching-time logic that employs silent actions and also considered the possibility that the information received by the monitor about silent actions may be incomplete. Our approach is more granular, since we can model different levels of uncertainty about the actions that occur.

Joshi *et al.* [12] presented a novel approach to verify LTL formulas on lossy traces. Moreover, it determines if there exists a loss-tolerant monitor for the LTL property, and builds one when it does. The approach only accounts for complete loss of events, and not uncertain events. In contrast, in our approach the multi-monitor lifted from π_M is loss-tolerant by construction. If the presence of lossy events has no impact on the monitor, $\hat{\pi}_M$ will produce a single verdict.

A different angle of attack is taken by works on probabilistic specifications. In such a case, ground statements are typically associated with probabilities, and the truth value of a specification is similarly associated with a quantitative measure of its likelihood. Probabilistic Signal Temporal Logic (ProbSTL) [**?**] has been introduced to deal with imprecise measurements.

Basin *et al.* [6] use a model that represents uncertainties over event occurrences, by means of what they call a *logging knowledge base*. This knowledge base is a sequence $\overline{\mathcal{D}} = \mathcal{D}_0, \mathcal{D}_1, \dots$ of first-order structures defined over the set of ternary Boolean values $\{\top, \bot, ?\}$, where "?" represents the unknown truth value. Each first-order structure represents a discrete time point, and totally defines the (ternary) truth value of each event predicate. Informally, for some predicate r of input arity n, $r(a_1, \dots, a_n) = \top$ in a given time point τ indicates that the event $r(a_1, \dots, a_n)$ happened at τ. Conversely, $r(a_1, \dots, a_n) = \bot$ in a given time point τ indicates that the event $r(a_1, \dots, a_n)$ did not happen at τ. Finally, $r(a_1, \dots, a_n) =?$ represents a *knowledge gap* with regard to whether $r(a_1, \dots, a_n)$ happened at τ.

Abstract TeSSLa is an extension of the TeSSLa specification language that accounts for known "gaps" in timed event streams [14]. Our approach does not deal with time, and is hence simpler. However, gaps can be represented in our model by a series of events that the proxy replaces by the "total" multi-event Ω. TeSSLa can model complex operations on input streams and produce intervals of possible values in the presence of gaps; our framework produces a multi-verdict, but quantifies the likelihood of each possible verdict.

These last two approaches are the closest to the model we propose in this paper. However, as we shall see in Section III-C, they have in common that some forms of imprecision and uncertainty handled by our framework are difficult to model and result in over-approximations in these models.

III. Trace Proxies

In this section, we describe a formal framework in which the various situations described in Section II can be modeled. Since our modeling of traces must account for access restrictions in the contents of events in a trace, we must first define an appropriate logical framework for dealing with it. Then, we show how the traditional definition of trace and monitor can be generalized to uncertain or "lossy" traces.

A. Multi-Traces and Proxies

Let $\mathbb{B} = \{\bot, \top\}$ be the set of Boolean truth values; let $\mathcal{A} = \{a, b, \dots\}$ be a finite set of atomic propositions. A valuation is a total function $\omega : \mathcal{A} \to \mathbb{B}$ that assigns a truth value to every atomic proposition. For $b \in \mathbb{B}$ and $a \in \mathcal{A}$, we note by $\omega[a \mapsto b]$ the valuation ω' such that $\omega'(x) = \omega(x)$ when $x \neq a$, and $\omega'(a) = b$. We denote by Ω the set of all valuations over $\mathcal{A}$.

In our context, a *uni-trace* is a finite sequence of valuations $\overline{\omega} = \omega_0, \omega_1, \dots, \omega_n$; we denote the set of uni-traces as Ω^*. Valuations, when seen as elements of a trace, will be called "events". The notation $\overline{\omega}[i]$ will be used to denote the event at the i-th position in a trace $\overline{\omega}$. The length of a trace $\overline{\omega}$ will be noted $|\overline{\omega}|$. The concatenation of two finite traces $\overline{\omega}$ and $\overline{\omega}'$ is noted as $\overline{\omega}{\cdot}\overline{\omega}'$. We shall use a special symbol, ϵ, to designate the empty event, which behaves in the usual way: $\epsilon \cdot \overline{\omega} = \overline{\omega} \cdot \epsilon = \overline{\omega}$. Uni-traces correspond to the concept commonly referred to as a *trace* in the literature on runtime verification. However, our introduction of a more general concept of trace in the following necessitates that the distinction be made explicit.

The modeling of events as sets of Boolean variables may seem primitive at first sight. However, we shall remind the reader that the same argument applies here as for SAT solving, and that such a setting is sufficient to model a very wide range of finite structures, given the proper amount of syntactical sugar. Case in point, one of our implemented scenarios in Section V models manipulations of a virtual shopping cart containing a set of purchased items, while another models temperature readings by a sensor.

Let Φ be the set of Boolean propositions that can be built using the classical Boolean connectives over $\mathcal{A}$. The definition of a valuation can be extended to propositional formulas by interpreting Boolean connectives in the usual way. For a given formula φ, a valuation ω is said to be *positive* if $\omega(\varphi) = \top$, and

negative if $\omega(\varphi) = \bot$. We shall define the positive interpretation of a propositional formula $\varphi \in \Phi$, noted $[\![\varphi]\!]_\top$, as the set of its positive valuations. A formula is said to be *uniquely positive* if it has a single positive valuation. The set $\Phi_\top$ represents the subset of Φ composed of uniquely positive formulas.

Traces can be generalized to *multi-traces*, where each element is not a single valuation, but rather a set of valuations; the set of multi-traces is noted $(2^\Omega)^*$. Given a multi-trace $\overline{v} \in (2^\Omega)^*$, a uni-projection is a uni-trace $\overline{\omega} \in \Omega^*$ such that $\overline{\omega}[i] \in \overline{v}[i]$ for all i. We shall denote by $\mathcal{U}(\overline{v})$ the set of all uni-projections of a multi-trace. The intuition behind uni-traces and multi-traces is that each event of a uni-trace represents a single, completely defined "world". In contrast, an event of a multi-trace may represent multiple, alternate "possible worlds". This vehicle will allow us to represent uncertainty and imprecision about the contents of an event.

As a convention, we shall use the symbol ω to represent a valuation (a uni-event), and v to represent a set of valuations (a multi-event). Given a formula $\varphi \in \Phi$ and a set of valuations $v \in 2^\Omega$ we say that v *supports* φ, and note $v \vDash \varphi$, if $v \subseteq [\![\varphi]\!]_\top$. Intuitively, a set of valuations supports a formula φ if it only contains possible worlds where φ holds.

Our next step in the management of uncertainty is to define a special type of transducer on multi-traces.

Definition 1. *Let $\overline{v}, \overline{v}', \overline{v}'' \in (2^\Omega)^*$ be three multi-traces and $v \in 2^\Omega$ be a multi-event. A* multi-trace proxy *is a relation $\pi \subseteq (2^\Omega)^* \times (2^\Omega)^*$, such that, if $(\overline{v}, \overline{v}') \in \pi$ and $(\overline{v} \cdot v, \overline{v}'') \in \pi$, then $\overline{v}'$ is a prefix of $\overline{v}''$.*

A multi-trace proxy is defined as a special type of transducer $\pi : (2^\Omega)^* \to (2^\Omega)^*$ that does not rewrite the past: that is, if $\overline{v}$ is a prefix of $\overline{v}'$, then $\pi(\overline{v})$ is a prefix of $\pi(\overline{v}')$. When defined in this manner, it is possible to treat a proxy as a stateful relation that can be fed input (multi-)events one by one, and which produces zero or more output (multi-)events for each of these inputs. We shall abuse notation and also accept that a proxy reads uni-traces by taking each input uni-event as a singleton multi-event.

To this basic definition, we can further qualify various kinds of proxies depending on additional properties. If $\pi : (2^\Omega)^* \to (2^\Omega)^*$ is a trace proxy, and $\overline{v} \in (2^\Omega)^*$ is a multi-trace, we say that π is *deterministic* if there exists a single $\overline{v}' \in (2^\Omega)^*$ such that $(\overline{v}, \overline{v}') \in \pi$. Similarly, π is *k-bounded* if for every $v \in 2^\Omega$ and $\overline{v}, \overline{v}', \overline{v}'' \in (2^\Omega)^*$, if $(\overline{v}, \overline{v}') \in \pi$ and $(\overline{v} \cdot v, \overline{v}'') \in \pi$, $|\overline{v}''| - |\overline{v}'| \leq k$.

When a proxy is deterministic, we shall use the notation $\pi(\overline{v})$ to designate the unique multi-trace $\overline{v}'$ such that $(\overline{v}, \overline{v}') \in \pi$. A proxy is called *world-preserving* when it produces exactly one output event for each input event, and all valuations of the input multi-event are still valuations of the output multi-event, i.e. possible worlds are not removed.

B. Monitors for Multi-Traces

A multi-trace proxy generalizes the notion of a monitor for some abstract property P. In what follows, we override the definition of ϵ to represent an empty trace. Our (propositional) monitor $\pi_P : \Omega^* \to \{\Omega, \emptyset, \epsilon\}$ is a deterministic transducer on uni-traces; each event of the trace is a valuation that can be seen as the binary encoding of a symbol of an input alphabet. It produces in return the empty trace (ϵ), or the multi-trace made of a single multi-event, Ω or $\emptyset$. These three outcomes represent the usual verdicts produced by a monitor: Ω represents the true verdict, $\emptyset$ the false verdict, and ϵ the inconclusive verdict. One can define a proxy such that for every (uni-)trace $\overline{\omega}$, we have that $(\overline{\omega}, \Omega) \in \pi_P$ if and only if $\overline{\omega}$ satisfies P, $(\overline{\omega}, \emptyset) \in \pi_P$ if and only if $\overline{\omega}$ violates P, and $(\overline{\omega}, \epsilon) \in \pi_P$ otherwise. It is also required that for every $x \in \{\Omega, \emptyset\}$ and every uni-trace $\overline{\omega} \in \Omega^*$, if $\pi_P(\overline{\omega}) = x$, then $\pi_P(\overline{\omega} \cdot \overline{\omega}') = x$ for every $\overline{\omega}' \in \Omega^*$. This corresponds to the intuition that a monitor producing a conclusive verdict for an input trace does not change its verdict when appending events to that trace.

We devise a construction to turn a monitor for uni-traces (a "uni-monitor") into one for multi-traces (a "multi-monitor").

Definition 2. *Let $\pi_P : \Omega^* \to \{\Omega, \emptyset, \epsilon\}$ be a uni-monitor for some property P. The multi-monitor lifted from π_P, noted $\hat{\pi}_P$, is the multi-trace proxy $\hat{\pi}_M : (2^\Omega)^* \to 2^{\{\Omega, \emptyset, \epsilon\}}$ such that, for every multi-trace $\overline{v} \in (2^\Omega)^*$: $\hat{\pi}_P(\overline{v}) \triangleq \bigcup_{\overline{\omega} \in \mathcal{U}(\overline{v})} \{\pi_P(\overline{\omega})\}$.*

For a given multi-trace, the output of the multi-monitor is the set of outputs obtained by running the underlying uni-monitor on every possible uni-projection. This set of outputs will be called the *multi-verdict*.

The fact that events fed to a monitor can now contain multiple valuations has an impact on the possible verdicts produced by the monitor. We say that a uni-monitor π_P is *ambiguous* for a multi-trace $\overline{v}$ if $|\hat{\pi}_P(\overline{v})| > 1$. This corresponds to the fact that two uni-projections of $\overline{v}$ result in two different verdicts. The monitor is *strongly ambiguous* for $\overline{v}$ if $\{\Omega, \emptyset\} \subseteq \hat{\pi}_P(\overline{v})$; in such a case, the monitor produces two contradictory verdicts. Otherwise, the monitor is called *weakly* ambiguous (for example, if $\hat{\pi}_P(\overline{v}) = \{\Omega, \epsilon\}$). In the case where a monitor is ambiguous for a given multi-trace, we can provide a measure of this ambiguity; each verdict can be associated to the fraction of all uni-traces that yield this verdict, and hence be used as a quantitative indication of its likelihood.

For $v \in \{\Omega, \emptyset, \epsilon\}$, we define a function $\rho^v_{\pi_P} : (2^\Omega)^* \to [0, 1]$ as:

$$\rho^v_{\pi_P}(\overline{v}) \triangleq \frac{|\{\overline{\omega} \in \mathcal{U}(\overline{v}) : \pi_P(\overline{\omega}) = v\}|}{|\mathcal{U}(\overline{v})|}$$

The function $\rho^v_{\pi_P}$ represents the fraction of all uni-projections of $\overline{v}$ for which the monitor π_P produces the verdict v.

We shall now consider a binary system composed of an access proxy and a monitor, in such a way that the output of the first is given as the input to the second.

Definition 3. *Let $\pi_A : \Omega^* \to (2^\Omega)^*$ be a proxy that turns uni-traces into multi-traces, and $\pi_P : \Omega^* \to \{\Omega, \emptyset, \epsilon\}$ is a uni-monitor as defined earlier. The access-controlled monitor $\mathcal{M}(\pi_A, \pi_P) : \Omega^* \to 2^{\{\Omega, \emptyset, \epsilon\}}$ is the trace proxy defined as $\mathcal{M}(\pi_A, \pi_P) \triangleq \hat{\pi}_P \circ \pi_A$.*

The intuition between an access-controlled monitor is that uni-events are produced from some abstract source; these

uni-events are then transformed by the action of the proxy π_A, resulting in a multi-trace. Hence, π_A represents the "degradation" of the original uni-trace. This multi-trace is then fed to the multi-monitor lifted from π_P, and its set of verdicts represents the output of the access-controlled monitor.

We can then extend the definition of ambiguity to an access-controlled monitor. Given a uni-monitor π_P for some property P and an access control proxy π_A, we say that π_P is *strongly* (resp. weakly) *affected* by π_A if there exists a uni-trace for which $\mathcal{M}(\pi_A, \pi_P)$ is strongly (resp. weakly) ambiguous. Finally, a monitor π_P is called *sound under* π_A if, for every uni-trace $\overline{\omega}$, the verdict of $\mathcal{M}(\pi_A, \pi_P)$ contains the verdict of π_P. It is easy to see that world preservation is a sufficient condition for soundness:[1]

Theorem 1. *If π_A is world-preserving, then $\hat{\pi}_P$ is sound under π_A.*

C. Modeling Access Restrictions with Proxies

Equipped with these basic definitions, we can now illustrate how the concept of access proxy can be used to model the various use cases about imprecise and uncertain data enumerated in Section II, including situations that cannot be accounted for in existing models of "lossy RV" discussed earlier.

1) Missing, Corrupted or Encrypted Values and Events: Missing values can first be modeled by altering the set of valuations of an input event. Consider for example the proxy π_1 defined as $\pi_1(v_1, \ldots, v_n) \triangleq \pi_1(v_1, \ldots, v_{n-1}) \cdot f_1(v_n)$, where $f_1 : 2^\Omega \to 2^\Omega$ is defined as:

$$f_1(v) \triangleq \bigcup_{\omega \in v} \{\omega[a \mapsto \top], \omega[a \mapsto \bot]\}$$

The action of function f_1 can be explained as follows: whatever the input multi-event $v \in 2^\Omega$, in the output event $f_1(v)$ we have neither $f_1(v) \models a$ nor $f_1(v) \models \neg a$ (all other variables being left unchanged).[2] In other words, in the output event, we can no longer conclude anything about the value of a in the input event. This is equivalent to a representation of uncertainty using a third "unknown" Boolean value [6]. It can be used to represent the fact that one of the readings inside an event is corrupted, missing, or encrypted with a key that is not in the possession of the recipient. In the case where each event represents a set of observations at a given time point, this proxy can also represent the fact that it is unknown whether a occurred or not in a time point.

An extreme case is a known missing event —that is, an event whose occurrence is known or has been deduced (for example by observing gaps in indexes, or after a database request has been denied), but whose content is completely missing. This can be represented in our model by an event that contains all valuations, i.e. Ω. The case of load shedding discussed in Section II-A4 can be modeled using such a mechanism, which is equivalent to the non-event χ used by [12] precisely to account for this situation.

For example, consider the proxy π'_1 defined as

$$\begin{aligned} \pi'_1(v) &\triangleq v \\ \pi'_1(v_1, \ldots, v_n) &\triangleq \pi'_1(v_1, \ldots, v_{n-1}) \text{ if } v_{n-1} \neq v_n \\ \pi'_1(v_1, \ldots, v_n) &\triangleq \pi'_1(v_1, \ldots, v_{n-1}) \cdot \Omega \text{ otherwise} \end{aligned}$$

This proxy compresses an input trace by replacing any stuttering events by Ω, symbolizing a deleted event. Alternately, a proxy could emit and discard events in an alternating fashion, to represent a form of systematic preemptive load shedding. One can also imagine variations over this basic mechanism: for example, the proxy could output all events until the occurrence of some trigger that activates load shedding, while some other trigger returns the proxy to normal operation.

2) Uncertainty and Fuzziness: Values inside an event may not be completely unknown, and only involve some amount of fuzziness. This is especially the case for sensor readings, where numerical values are typically accompanied by a precision interval. A discrete set of numerical values $\mathbb{D}$ can be modeled with Boolean variables in various ways —an easy one being to associate each value $d \in \mathbb{D}$ to a Boolean variable b_d. Uncertainty can then be represented as a function $\gamma : \mathbb{D} \to 2^{\mathbb{D}}$.

A proxy π_2 can be defined as $\pi_2(v_1, \ldots, v_n) \triangleq \pi_2(v_1, \ldots, v_{n-1}) \cdot f_2(v_n)$, where $f_2 : 2^\Omega \to 2^\Omega$ is defined as:

$$f_2(v) \triangleq \bigcup_{\omega \in v} \bigcup_{b \in \gamma(b_v)} \{\omega[b_v \mapsto \bot, b \mapsto \top]\}$$

where b_v is the unique b_i in ω such that $b_i = \top$. In other words, the proxy turns each valuation where b_d is true into the set of valuations where each $b_i \in \gamma(d)$ is made successively true (and leaves any other variables unchanged). This corresponds to the intuition that in any possible world, the numerical value can be any one of $\gamma(d)$, but only one of them at a time and none of the other values. Stated in this way, it is the *discrete* equivalent of the notion of *abstract data domain* in Abstract TeSSLa's modeling of uncertainty [14].

Note however that this form of imprecision cannot be accounted for in a model where each event is a single possible world with ternary Boolean values (i.e. [6]). Giving the value "unknown" to all temperature variables in $\gamma(d)$ misses the fact that in any possible interpretation, exactly one of them *must* be the value. In other words, this modeling is an over-approximation that introduces the spurious possible world where the event can contain "no value", or many values. This situation cannot be modeled either by Joshi *et al.*'s approach [12], where events are atomic and are either completely known or completely unknown (except for their occurrence).

3) Correlated Uncertainty and Fuzziness: So far, the examples of degradation we have shown apply in an independent manner to a single input variable or signal at a time. Correlated uncertainty occurs when deterioration of information is applied in a way that depends on more than one input variable.

Consider the proxy π_3 defined in the same way as π_2, but with f_2 replaced by $f_3(v) \triangleq v \cup \bigcup_{\omega \in v} \{\omega_{a \leftrightarrow b}\}$. The notation $\omega_{a \leftrightarrow b}$

[1] Due to lack of space, proofs of theorems have been moved into an appendix at the end of the paper.

[2] The condition is actually even stronger: for any formula φ such that $v \models \varphi$, we have neither $\varphi \to a$ nor $\varphi \to \neg a$.

designates the valuation that swaps the assignments of a and b in ω. This has the effect of making a and b indistinguishable: an input multi-event that supports a is transformed into an output multi-event that only supports the weaker proposition $a \vee b$ (and similarly for events that support b). In other words, it is no longer possible to conclude precisely that a is true or that b is true, only that at least one of them is true. This is a simple form of the impedance mismatch use case we discussed in Section II.

This situation cannot be accounted for in any of the models we surveyed in Section II-B. In three-valued logics, the reasoning is the same as before: the best one can do in such a model is to over-approximate uncertainty by stating that the occurrence of both a and b is unknown (this abstraction is still precise for events where neither a nor b are true). For abstract data domains [14], the situation becomes even less desirable: since these domains are defined for each variable separately, and must remain the same for the entire trace, the only world-preserving abstraction is the one that replaces values of a and b with both their possible values at all time points, which is an even greater over-approximation.

IV. Automata-Based Trace Proxies

As one can see, the multi-event model and the definition of a trace proxy is very flexible in its ability to model various forms of imprecision, uncertainty and missing or incorrect values. In the following, we shall focus on one concrete representation of multi-trace proxies, by providing an extension of Mealy machines.

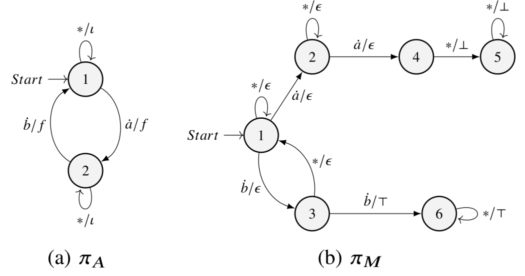
A. Propositional Mealy Machines

In the following, we shall assume that the representation of trace proxies is based on a special type of finite-state machine called a *propositional machine*. It is formally defined as follows.

Definition 4. *A* propositional machine *is a triplet $M = \langle S, s^0, \mu \rangle$ consisting of a finite set of states S, a unique start state $s^0 \in S$, and a marking $\mu \subseteq S \times \Phi \times S$ associating propositional formulas to pairs of states.*

Figure 1 shows two examples of propositional machines represented graphically. As one can see, the main difference between a propositional machine and a traditional finite-state machine is the fact that input symbols and transitions are replaced by a marking with *propositional formulas*. Note that there can be two formulas φ, φ' associated with the same two states; these would be represented as two distinct edges in the graph representation of the machine.

The figure illustrates a few notational shortcuts we shall use in the following. Given propositional variables $\mathcal{A} = \{a_1, \dots, a_m\}$, the notation $\dot{a}_k$ represents the propositional formula $(\bigwedge_{i \neq k} \neg a_i) \wedge a_k$. Moreover, the special symbol $\star$ is meant as a notation shortcut meaning "otherwise": in a given state, it corresponds to the propositional formula made by the conjunction of the negation of all formulas in the other outgoing edges. For example, assuming that $\mathcal{A} = \{a, b, c\}$, the



(a) π_A (b) π_M

Figure 1: Access-controlled monitor represented as a pair of propositional machines.

$\star$ symbol in state 1 of Figure 1a corresponds to the formula $\neg a \vee b \vee c$ (which is the negation of $\dot{a} = a \wedge \neg b \wedge \neg c$).

Definition 5. *Let $\mu \subseteq S \times \Phi \times S$ be a marking over a propositional machine M. The transition relation induced by μ, is the relation $\tilde{\mu} \subseteq S \times 2^{\Omega} \times S$ such that, for every $s, s' \in S$ and every $v \in 2^{\Omega}$, we have that $(s, v, s') \in \tilde{\mu}$ if and only if $v \cap [\![\varphi]\!]_\top \neq \emptyset$.*

Intuitively, in a state s and given an input multi-event $v \in 2^{\Omega}$, the transition $s - \varphi \rightarrow s'$ is possible if the positive valuations of φ contain at least one valuation of v. In other words, there exists one "possible world" admitted by φ that is compatible with v. In turn, this definition of a transition relation can be lifted to multi-traces expressed as sequences of Boolean formulas; given a multi-event $\varphi' \in \Phi$, the transition $s - \varphi \rightarrow s'$ is possible if $[\![\varphi]\!]_\top \cap [\![\varphi']\!]_\top \neq \emptyset$. This corresponds to the situation where both φ and φ' share at least one positive valuation.

Boolean formulas are a useful shortcut to succinctly represent sets of valuations. In the following, we shall concentrate on multi-traces viewed as elements of Φ^*. Given a multi-trace $\overline{\varphi}$ of length n, a possible *run* of M is any sequence $s_0 - \psi_0 \rightarrow s_1 - \psi_1 \rightarrow \cdots - \psi_{n-1} \rightarrow s_n$ such that $[\![\varphi[i]]\!]_\top \cap [\![\psi_i]\!] \neq \emptyset$ and $(s_i, \psi_i, s_{i+1}) \in \mu$ for every $i \in [0, n-1]$. In such a case, the complexity of determining if a sequence of transitions is a run for some $\overline{\varphi} \in \Phi^*$ can be precisely established.

Theorem 2. *Let $\mathcal{A} = \{a_1, \dots, a_m\}$ be a set of m propositional variables. Let $\overline{\varphi} = \varphi_0, \dots, \varphi_{n-1}$ be a finite multi-trace over $\mathcal{A}$ and M be a propositional machine. Let $s_0, \dots, s_n$ be a sequence of states, and $\psi_0, \dots, \psi_{n-1}$ a sequence of transition labels in μ. Determining if $s_0 - \psi_0 \rightarrow s_1 - \psi_1 \rightarrow \cdots - \psi_{n-1} \rightarrow s_n$ is a run of M for $\overline{\varphi}$ is NP-complete.*

In the general case, a propositional machine run is not necessarily uniquely defined, even when the formulas on each outgoing transition in a state are mutually exclusive. Case in point, consider the machine of Figure 1b; in state 1, if the machine receives for its input event the proposition $a \vee b$, one can see that the input event can fire both the transitions $\dot{a}$ and $\dot{b}$, and therefore both 2 and 3 are possible next states.

This basic definition can be extended by allowing the propositional machine to produce output symbols. To achieve this, we define an output function $\gamma : S \times \Phi \times S \rightarrow (\Phi \rightarrow \Phi)$. To each transition $s - \psi \rightarrow s'$ in M, the function γ associates

a function $\ell : \Phi \to \Phi$ which transforms an input multi-event into another output multi-event. Given an input multi-trace that induces a run in M, the resulting output multi-trace is defined as follows:

Definition 6. *Let $\overline{\varphi} = \varphi_0, \dots, \varphi_{n-1}$ be a finite multi-trace and let $s_0 - \psi_0 \to s_1 - \psi_1 \to \cdots - \psi_{n-1} \to s_n$ be a run of M for $\overline{\varphi}$. The output multi-trace produced by M is the sequence $\overline{\varphi}' = \varphi'_0, \dots, \varphi'_{n-1}$, where $\varphi'_i = \gamma(s_i, \psi_i, s_{i+1})(\varphi_i)$ for each $i \in \{0, \dots, n-1\}$.*

Simply put, an input formula is replaced by applying to it the function $\ell : \Phi \to \Phi$ associated by γ to the corresponding transition in M. This function effectively turns a propositional machine into an extended version of a Mealy machine. We can further extend the definition of ℓ, and allow its image to be $\Phi \cup \{\epsilon\}$. In such a case, the machine may produce no output (ϵ) for an input event. This representation makes it possible to model the class of *1-bounded* trace proxies.

In the following, we will represent a few functions $\ell : \Phi \to \Phi$ by special symbols. Function ι will designate the identity, i.e. $\iota(\varphi) = \varphi$ for all $\varphi \in \Phi$. For a given formula φ, we will abuse notation and use φ to designate the constant function that turns any input formula into φ (the most notable ones being the constants $\top$ and $\bot$). Although ℓ accepts and returns a Boolean formula, it is not excluded that its definition be made in terms of sets of valuations. For example, the proxy π_3, defined in Section III-A could be defined by first converting an input formula φ into its set of positive valuations $[\![\varphi]\!]_\top$, performing the transformations on that set, and converting this set back into a Boolean formula φ'.[3]

To illustrate this point, Figure 1 shows a simple example of a pair of access proxy and monitor, on the input alphabet $\mathcal{A} = \{a, b, c\}$. In this case, the function f in π_A is the function f_3 already given as an example in Section III-B, which makes it impossible to know which one of a or b is true in an input event, only that at least one of them is true. This has an impact on the verdict that can be produced by π_M for some of the traces it receives. For example, the uni-trace $\dot{a}, \dot{b}, \dot{c}, \dot{a}$ is transformed by the access proxy into $\dot{c}, \dot{a} \vee \dot{b}, \dot{a} \vee \dot{b}, \dot{c}, \dot{a} \vee \dot{b}$. There are 8 possible runs in π_M for this input multi-trace, including one that visits the states 1–2–4–5, and produces the verdict $\bot$, and another that visits the states 1–3–6–6, and produces the verdict $\top$. Therefore, the verdict becomes ambiguous.

B. A Monitoring Algorithm

Equipped with such definitions, we can now define an algorithm which, given an access-controlled monitor $\mathcal{M}(\pi_A, \pi_M)$ expressed as a pair of propositional machines and a finite multi-trace prefix $\overline{\varphi} \in \Phi^*$, computes the multi-verdict associated to this prefix. Furthermore, this multi-verdict is quantified —that is, if its output set contains more than one value, the fraction computed by ρ, as defined in Section III-B, will be associated to each value.

The procedure is defined in Algorithm 1, for an access proxy $\pi_A = \langle s^0_A, S_A, \mu_A\rangle$ and a uni-monitor $\pi_P = \langle s^0_P, S_P, \mu_P\rangle$.

[3]One easy way being by creating the disjunction of each valuation.

Algorithm 1 Access-controlled update algorithm

1: **procedure** UPDATE(φ, s_A, σ)
2: $\quad\sigma' \leftarrow \emptyset$ $\qquad\triangleright\ \sigma' : S_P \to \mathbb{N}$
3: $\quad\beta \leftarrow \emptyset$ $\qquad\triangleright\ \beta : \{\Omega, \emptyset, \epsilon\} \to \mathbb{N}$
4: $\quad(\psi_A, s'_A) \leftarrow$ the unique ψ_A, s'_A s.t. $(s_A, \psi_A, s'_A) \in \tilde{\mu}_{\pi_A}$
5: $\quad\ell_A \leftarrow \gamma_{\pi_A}(s_A, \psi_A, s'_A)$
6: $\quad\varphi' \leftarrow \ell_A(\varphi)$
7: $\quad$**for** $(s_P, n) \in \sigma$ **do**
8: $\qquad$**for** $(s_P, \psi_P, s'_P) \in \tilde{\mu}_{\pi_P}$ **do**
9: $\qquad\quad c \leftarrow |[\![\psi_P]\!]_\top \cap [\![\varphi']\!]_\top|$
10: $\qquad\quad$**if** $c > 0$ **then**
11: $\qquad\qquad\sigma'(s'_P) \leftarrow \sigma'(s'_P) + (n * c)$
12: $\qquad\qquad l \leftarrow \gamma_{\pi_P}(s_P, \psi_P, s'_P)$
13: $\qquad\qquad\beta(l) \leftarrow \beta(l) + (n * c)$
14: $\qquad\quad$**end if**
15: $\qquad$**end for**
16: $\quad$**end for**
17: $\quad$**return** $\langle s'_A, \sigma', \beta\rangle$
18: **end procedure**

We assume that π_A and π_P are both deterministic and that their transition relation is total. The algorithm takes as input a multi-event $\varphi \in \Phi$, a state $s_A \in S_A$, and a partial function $\sigma : S_P \to \mathbb{N}$. Intuitively, φ is the new input event to ingest, s_A is the current state in π_A reached after reading a trace prefix $\overline{\varphi}$, and for some $s \in S_P$, $\sigma(s)$ designates the number of uni-projections of $\overline{\varphi}$ that result in a run of π_P ending in state s ($\sigma(s)$ being undefined can be assimilated to the case $\sigma(s) = 0$, which indicates that no uni-projection ends in s).

Lines 2–3 initialize an empty partial function $\sigma' : S_P \to \mathbb{N}$, and an empty partial function $\beta : \{\Omega, \emptyset, \epsilon\} \to \mathbb{N}$. Function σ' stores the update of σ after ingesting the input event; β maps each of the three verdicts to the number of uni-projections being mapped by π_P to that verdict. Line 4 identifies the transition $(s_A, \psi_A, s'_A) \in \tilde{\mu}_{\pi_A}$ that can be taken from s_A and input event φ; since we assumed that φ is a uni-event and that π_A is total and deterministic, this transition exists and is unique. Lines 5–6 then apply the output function $\gamma_{\pi_A}(s_A, \psi_A, s'_A)$ that associates the transformation function ℓ_A to the transition producing the resulting output multi-event φ'.

The second part of the algorithm is made of the lines 7–16, and corresponds to the update of both the uni-monitor's states, and the count of uni-projections for each verdict. The algorithm takes in succession each state $s_P \in S_P$ of the uni-monitor reached after processing one of the uni-projections of $\overline{\varphi}$. From each such state s_P, it computes the count c of uni-projections that can fire the condition ψ_P on each outgoing transition. When this is the case, the number of uni-projections reaching state s'_P, stored in σ', is incremented by nc (line 11), where n is the number of uni-projections of $\overline{\varphi}$ reaching s_P. This calculation can be explained by the fact that, if there are n uni-projections of $\overline{\varphi}$ that reach s_P, and c uni-projections of φ allow us to take the transition $s_P - \psi \to s'_P$, then there are nc uni-projections of $\overline{\varphi} \cdot \varphi$ whose last two visited states are s_P and s'_P.

The verdict l produced by π_P on taking the transition $s_P - \psi \to s'_P$ is fetched (line 12). Then, mapping β is updated: the number $\beta(l)$ of uni-projections producing verdict l is

incremented by nc (line 13), by the same reasoning as for the update of σ'. The process repeats for all states $s_P \in S_P$ defined in σ. Upon termination, the algorithm returns the triplet $\langle s'_A, \sigma', \beta \rangle$: s'_A is the new current state of π_A, σ' maps each possible current state of π_M with a number of uni-projections, and β does the same with each verdict.

In order to compute the verdict of an access-controlled monitor $\mathcal{M}(\pi_A, \pi_M)$ on a uni-trace $\overline{\varphi}$, it suffices to call the procedure UPDATE repeatedly. A straightforward procedure called MONITOR (not shown due to lack of space) can iterate over each event in $\overline{\sigma}$, call UPDATE repeatedly, and output the current mapping β associating each of the three verdicts to the corresponding number of uni-projections. The start configuration of this procedure is simply the unique initial state s^0_A and the mapping that stipulates that a single uni-projection of the empty trace reaches the unique initial state of π_P. The following theorem states that the output map produced by MONITOR on a uni-trace $\overline{\varphi}$ does correspond to the number of uni-projections of $\pi_A(\overline{\varphi})$ that result in each of the three possible verdicts.

Theorem 3. *Let $\mathcal{M}(\pi_A, \pi_M)$ be an access-controlled monitor expressed as two propositional machines, $\overline{\varphi}$ be a uni-trace and $\beta : \{\Omega, \emptyset, \epsilon\} \to \mathbb{N}$ be the mapping produced by calling MONITOR($\overline{\varphi}$). For $v \in \{\Omega, \emptyset, \epsilon\}$, we have that:*

$$\frac{\beta(v)}{\sum_{v' \in \{\Omega, \emptyset, \epsilon\}} \beta(v')} = \rho^v_{\pi_P}(\pi_A(\overline{\varphi}))$$

C. Discussion

A few remarks must be made about this algorithm. First, it operates "on-the-fly": each new input event is handled by updating states and uni-projection counts obtained on the previous computation step. In other words, the algorithm does not need to recalculate everything from the start, which makes it possible to operate in streaming fashion. In particular, it does not explicitly enumerate all uni-projections. Second, it merges the operation of the access proxy π_A and the monitor π_P. An algorithm only for the multi-monitor lifted from π_P (i.e. $\hat{\pi}_P$) can easily be obtained by removing lines 2–6 of Algorithm 1 and using φ' as the input to UPDATE.

In terms of complexity, a call to UPDATE is dominated by the loop in lines 7–16; one can easily see that the number of iterations of the inner loop of lines 9–14 is bounded by $|S_P| \cdot |\tilde{\mu}_{\pi_P}|$. However, each such iteration involves an execution of line 9, which computes the number of positive valuations that are common to two Boolean formulas. A naïve way of obtaining this count it is to enumerate all $2^{|\mathcal{A}|}$ valuations. From this, we can conclude that the complexity of MONITOR is linear in the length of the input trace, the number of states and the number of transitions of π_P, and exponential in the number of propositional variables encoding each event.

The fact that each call to the monitor involves solving multiple NP-complete problems may seem alarming. However, this is mitigated by two observations. First, SAT instances typically involve very large numbers of variables. It is expected that the Boolean encoding of events, in a monitoring context, will be much smaller. For example, a monitoring problem with an alphabet of 1,000 different events can be encoded using only 10 Boolean variables. Therefore, the model counting and SAT problems involved are expected to be, in comparison, very small.

V. EXPERIMENTAL RESULTS

An implementation of propositional machines has been realized in the form of a Java library that extends the BeepBeep event stream processing engine [9]. The library is open source and publicly available[4]. In this library, multi-events exist in two flavors: the `ConcreteMultiEvent` is implemented as a set of valuations, while the `SymbolicMultiEvent` is implemented as a propositional formula. Both classes implement the same methods to enumerate their valuations and determine if two events have a non-empty intersection. Hence, a trace of events and a propositional machine can use either of these two multi-event types interchangeably. Propositional Mealy machines are implemented as an object that descends from BeepBeep's `Processor` class; this means that once they are instantiated, they can be connected to any other BeepBeep processor to form a potentially complex pipeline. Similarly, propositional formulas are descendants of BeepBeep's `Function` class. A downloadable instance containing all the experiments of this paper can be obtained online[5]. All the experiments were run on a Intel CORE i5-7200U 2.5 GHz running Ubuntu 18.04, inside a Java 8 virtual machine with 1746 MB of memory.

A. Overhead Experiments

A first set of experiments is meant to assess the overhead, both in terms of running time and memory consumption, incurred by the presence of an access proxy and the lifting of a uni-monitor into a multi-monitor. Our experiments are made of a number of "scenarios", where each scenario corresponds to a source of uni-events, an access proxy and a property to monitor, the latter two expressed as propositional Mealy machines:

Simple: the running example discussed in this paper, and represented in Figure 1.

MPlayer: a generated sequence of operations (play, pause, etc.) of the operation of a media player (cf. [12]), with an access proxy applying the load shedding strategy discussed in Section II-A4. The monitor verifies the correct ordering of the operations; it has 5 states and 20 transitions.

Temperature Threshold: a scenario made of CPU temperature readings from a cyber-physical system, adapted from [2]. Temperatures are encoded using 20 Boolean variables representing intervals of 1 degree. The access proxy applies a transformation that adds an uncertainty of ±2 degree. The monitor checks that for the first 100 units of time, whenever the temperature falls below a certain threshold T, it will again be above the threshold within 5 units of time. This monitor has 486 states and 966 transitions.

Shopping Cart: a scenario made of Boolean-encoded sequences of shopping cart manipulation operations. The monitor

[4]https://github.com/liflab/propositional-machines
[5]https://github.com/liflab/propositional-machines-lab

Table I: Global impact of the presence of an access proxy for the tested scenarios.

(a) Throughput (Hz)

Scenario	With	Without
MPlayer	42158.516	1428571.4
Shopping Cart	33277.87	609756.1
Simple	452488.7	1086956.5
Tempreature Threshold	53966.54	431034.47

(b) Memory (B)

Scenario	With	Without
MPlayer	23622	9214
Shopping Cart	124344	119696
Simple	10722	7714
Tempreature Threshold	87600	85508

verifies properties on the sequence of operations based on a study of an Amazon web service [8]; it has 1,154 states and 7,267 transitions. The access proxy replaces 5% of events by the multi-event Ω symbolizing data loss.

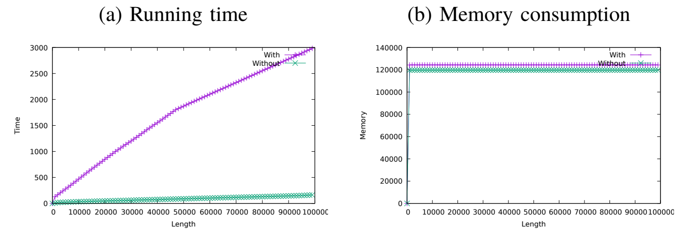
CPU Load: a scenario made of Boolean-encoded CPU load values. The monitor uses the same property as in [14], which checks that the average load over a sliding window of five readings does not exceed some arbitrary threshold T. The access proxy adds an uncertainty of $\pm 1\%$ to each reading. Due to the presence of a sliding window and the use of arithmetic, this last example has a very large state space, consisting of 2×10^9 states and more than 10^{10} transitions.

For each of these scenarios, we ran a randomly-generated input trace of 100000 uni-events into the uni-monitor alone, and then into the access-controlled monitor made of the access proxy and the multi-monitor. We measured the difference in terms of throughput (number of events ingested per unit of time) and memory consumption. The global impact of the presence of the access proxy is summarized in Table I. In terms of throughput, it can be observed that the inclusion of an access proxy induces a slowdown on the monitoring process, since the monitor must handle multi-events instead of uni-events, and track the various possible states the uni-monitor can be in. However, for the traces and properties included in our tests, this slowdown ranges between 2× and 8×, which seems to indicate that the handling of multi-events does not impose too big an overhead on the performance of the monitor.

This should be put in perspective with the extremely large number of uni-traces handled by the multi-monitor. In the *Simple* scenario, the access proxy generates a multi-trace that corresponds to 10^{8631} distinct uni-traces; in the *Shopping Cart* scenario, this number reaches 10^{26400}. However, the complexity of Algorithm 1 does not depend on the number of uni-traces, but rather on a much simpler metric, which is the number of multi-events produced for each input uni-event; moreover, only a count of uni-traces needs to be maintained, and uni-events are discarded after processing. This is why our approach scales despite the large number of "possible worlds" introduced by the insertion of uncertainty by the proxy.

The impact is less noticeable in terms of memory (Table

Figure 2: Impact of the presence of an access proxy on the throughput and memory consumption of the monitor, for the *Shopping Cart* scenario.



Ib). Even though the presence of multi-events does increase the maximum amount of memory consumed by the access-controlled monitor with respect to the single uni-monitor, this increase is relatively negligible and never exceeds a factor 1.5.

To get further details on the actual behavior of the access-controlled monitor, we also measured the evolution of time and memory consumption across a trace. Running time is shown in Figure 2a for the *Shopping Cart* scenario (plots for the remaining scenarios are not shown but exhibit very similar trends). An important point is that processing time per event is higher, yet constant. This feature is important for an access-controlled monitor to be usable for long-running systems.

Memory consumption is plotted in Figure 2b. One can observe that the memory consumption of both the uni-monitor and the access-controlled monitor is constant throughout the whole trace. This observation is not surprising, as Algorithm 1 uses data structures (the mappings β and σ) of constant or bounded size. More importantly, although each input uni-event may result in multiple uni-events being processed, these events are discarded at the end of the processing and only their count needs to be kept.

B. *Comparison to Over-Approximations*

As we mentioned earlier, our modeling of imprecision and uncertainty can account for finer-grained restrictions that can only be expressed as conservative (i.e. world-preserving) over-approximations in the state of the art. To better highlight the impact that such approximations can have on the performance of a monitor, we designed a second set of experiments that revisits three of the scenarios. In each case, we describe the operation of two access proxies: the first is the one used by our approach, and the second is an over-approximation of this proxy that is the "best effort" that can be modeled by one of the related approaches mentioned in Section II-B.

Simple: to symbolize impedance mismatch, our access proxy replaces values of $\dot{a}$ and $\dot{b}$ by the less precise assertion $\dot{a} \vee \dot{b}$. Models that do not handle correlated uncertainty (e.g. [12], [14]) must rather resort to an over-approximation where all occurrences of $\dot{a}$ and $\dot{b}$ are replaced by possible worlds where they can be either true or false whenever one of them is true in the original event.

Temperature Threshold: our proxy is left unchanged from the original set of experiments; the over-approximation replaces

Table II: Impact of using an over-approximation for the various scenarios.

(a) Throughput (Hz)

Scenario	Our approach	Over-approx.
MPlayer	1666.6666	666.6667
Simple	1818.1818	2857.1428
Tempreature Threshold	229.88506	175.4386

(b) Verdict precision

Scenario	TP	IP	FP	TB	IB	FB
MPlayer	0	4.85	4.85	0	4.85	11.08
Simple	0	0.0	1.94	5.83	10.25	8.28
Temperature Threshold	0	0	28.82	0	59.07	60.83

all variables in the temperature interval of each event by the possible worlds where they can hold any value. As discussed in Section III-C2, this approximation is necessary in a framework where all variables must be given a single ternary Boolean value (e.g. [6]).

MPlayer: to show the impact of impedance mismatch, our proxy has the *Stop* and *Pause* events conflated into a fuzzier *Interrupted* event that stands for both of them. Similar to the *Simple* scenario, the over-approximation replaces their value into possible worlds where they can both be either true or false (as would be required in e.g. [12], [14]).

The results are summarized in Table II. Table IIa shows that the use of a coarser-grained modeling of imprecision generally has a negative impact on throughput (with the exception of *Simple*), mostly caused by the larger number of possible worlds that must be handled by the over-approximation. More interestingly, Table IIb shows that, as we already hinted earlier, this over-approximation also impacts the precision of the verdict returned by the underlying monitor. For each scenario, it shows the base-10 logarithm of the number of uni-projections mapped to each verdict T(rue), F(alse) and I(conclusive), for both our access-controlled proxy (P) and the "best effort" over-approximation (B).

In all scenarios, the over-approximation cannot produce a definite verdict. For *Temperature*, about 10% of all uni-projections are mapped to the "unknown" verdict instead of the correct false verdict. In comparison, our access-controlled proxy produces a single clear false verdict. For scenarios such as *Simple*, the over-approximation fares even worse: it causes all three verdicts to be possible, whereas our proposed access controlled monitor still produces a single (false) verdict. Although it can be observed that, in the over-approximation, only 0.1% of all uni-traces are mapped to the incorrect verdict, we argue there is nevertheless a fundamental qualitative gap between a definite correct verdict and a merely *likely* one, especially in the context of safety-critical systems, where monitoring is commonly employed.

The case of the *MPlayer* scenario also deserves discussion. The correct verdict of the original uni-trace should be "?". In this case, both our proxy and the over-approximation produce an equivocal verdict. However, the over-approximation makes the false verdict many orders of magnitude more likely than the (correct) inconclusive verdict, while in our proxy, both verdicts are relatively nose-to-nose. This shows that, in some cases, an over-approximation can not only result in a clear verdict being turned into an uncertain one, it can also be such that the verdict given as the most likely is the incorrect one.

VI. Conclusion and Future Work

In this work, we presented a flexible framework to deal with access restrictions on the events in a trace. A stateful proxy is used to model the known gaps and imprecise values in the events and impose other types of uncertainty on the events before feeding it to the monitor. We introduced a construction of a loss-tolerant multi-monitor from a uni-monitor that runs in linear time and quantifies the multi-verdict, and adds constant memory and time overhead on each input event.

Our proposed framework paves the way to a large number of ancillary research questions centered on the notion of ambiguity. First is the question of *deciding* ambiguity: given an access proxy π_A and a monitor π_P, determine if there exists a trace for which $\mathcal{M}(\pi_A, \pi_P)$ is ambiguous; ties to existing results on monitorability could be exploited. Second is the question of *fixing* ambiguity: find the minimal modifications required to π_A such that ambiguity is lifted for a given monitor. Finally, the reverse question of *introducing* ambiguity could also be considered: given a monitor π_P, find the "least disturbing" proxy π_A such that $\mathcal{M}(\pi_A, \pi_P)$ becomes ambiguous. This latter question could be studied to determine what access restrictions should be introduced in order to prevent an attacker from deducing some property π_P from a sensitive log.

The access restrictions presented in the paper were all world-preserving; however, our model of an access proxy can also be used to inflict transformations to an input trace that do not satisfy this condition. It highlights an interesting side effect of the presence of an explicit model of event degradation, as it can be used to study the impact of feeding a monitor with a trace of events that is not only imprecise, but also *incorrect* according to some systematic pattern. For example, one could define a proxy that removes events according to some pattern (to study the impact of dropped events going undetected), or that adds events according to some pattern (for example, to study the impact of inserting stuttering into the trace). Obviously, in such cases, a multi-monitor receiving such traces can no longer guarantee the soundness of its multi-verdict. Restoring soundness of the multi-monitor without the hypothesis of world-preservation is left as future work.

A possible refinement of this scenario could address the notion of throttling we touched upon in Section II. A monitor could be given an access "budget" to the events of a log, with each access request imputing a charge on this budget. Depending on the property and its current state, a monitor could decide whether or not to make an access request for an event, in order to save as much access budget as possible. This could turn monitoring into a form of optimization problem.

A direct extension of the model would be the symbolic manipulation of infinite or continuous variables. This would allow the convenient expression of a wider range of event types and access restrictions. Moreover, the notion of uncertainty

and loss tolerance could be extended to other formal notations apart from Mealy machines, such as Linear Temporal Logic.

Another angle of future work can be in the field of runtime enforcement, where the proxy can be modeled to make the modifications so that each trace it produces is a replacement of the input trace that satisfies the given property, and all output traces can be evaluated to choose the most optimal trace that can replace the input trace with minimal amount of modifications.

References

[1] L. Aceto, A. Achilleos, A. Francalanza, and A. Ingólfsdóttir. Monitoring for silent actions. In S. V. Lokam and R. Ramanujam, editors, *FSTTCS*, volume 93 of *LIPIcs*, pages 7:1–7:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[2] A. Aerts, M. Reniers, and M. Mousavi. Chapter 19 - model-based testing of cyber-physical systems. In H. Song, D. B. Rawat, S. Jeschke, and C. Brecher, editors, *Cyber-Physical Systems*, Intelligent Data-Centric Systems, pages 287–304. Academic Press, Boston, 2017.

[3] A. Artikis, T. Eiter, A. Margara, and S. Vansummeren. Foundations of Composite Event Recognition (Dagstuhl Seminar 20071). *Dagstuhl Reports*, 10(2):19–49, 2020.

[4] E. Bartocci, Y. Falcone, A. Francalanza, and G. Reger. Introduction to runtime verification. In E. Bartocci and Y. Falcone, editors, *Lectures on Runtime Verification - Introductory and Advanced Topics*, volume 10457 of *Lecture Notes in Computer Science*, pages 1–33. Springer, 2018.

[5] E. Bartocci and R. Grosu. Monitoring with uncertainty. In L. Bortolussi, M. L. Bujorianu, and G. Pola, editors, *HAS*, volume 124 of *EPTCS*, pages 1–4, 2013.

[6] D. A. Basin, F. Klaedtke, S. Marinovic, and E. Zalinescu. Monitoring compliance policies over incomplete and disagreeing logs. In S. Qadeer and S. Tasiran, editors, *RV*, volume 7687 of *Lecture Notes in Computer Science*, pages 151–167. Springer, 2012.

[7] A. Bauer, M. Leucker, and C. Schallhart. Runtime verification for LTL and TLTL. *ACM Trans. Softw. Eng. Methodol.*, 20(4):14:1–14:64, 2011.

[8] S. Hallé, T. Bultan, G. Hughes, M. Alkhalaf, and R. Villemaire. Runtime verification of web service interface contracts. *IEEE Computer*, 43(3):59–66, 2010.

[9] S. Hallé. *Event Stream Processing with BeepBeep 3: Log Crunching and Analysis Made Easy*. Presses de l'Université du Québec, 2018. ISBN 978-2-7605-5101-5.

[10] K. Havelund and G. Rosu. Efficient monitoring of safety properties. *STTT*, 6(2):158–173, 2004.

[11] J. Huang, C. Erdogan, Y. Zhang, B. M. Moore, Q. Luo, A. Sundaresan, and G. Rosu. ROSRV: runtime verification for robots. In B. Bonakdarpour and S. A. Smolka, editors, *RV*, volume 8734 of *Lecture Notes in Computer Science*, pages 247–254. Springer, 2014.

[12] Y. Joshi, G. M. Tchamgoue, and S. Fischmeister. Runtime verification of LTL on lossy traces. In A. Seffah, B. Penzenstadler, C. Alves, and X. Peng, editors, *SAC*, pages 1379–1386. ACM, 2017.

[13] K. Kalajdzic, E. Bartocci, S. A. Smolka, S. D. Stoller, and R. Grosu. Runtime verification with particle filtering. In A. Legay and S. Bensalem, editors, *RV*, volume 8174 of *Lecture Notes in Computer Science*, pages 149–166. Springer, 2013.

[14] M. Leucker, C. Sánchez, T. Scheffel, M. Schmitz, and D. Thoma. Runtime verification for timed event streams with partial information. In B. Finkbeiner and L. Mariani, editors, *RV*, volume 11757 of *Lecture Notes in Computer Science*, pages 273–291. Springer, 2019.

[15] M. Leucker and C. Schallhart. A brief account of runtime verification. *J. Log. Algebr. Program.*, 78(5):293–303, 2009.

[16] H. Li, W. Shang, and A. E. Hassan. Which log level should developers choose for a new logging statement? *Empirical Software Engineering*, 22(4):1684–1716, 2017.

[17] C. Sánchez, G. Schneider, W. Ahrendt, E. Bartocci, D. Bianculli, C. Colombo, Y. Falcone, A. Francalanza, S. Krstic, J. M. Lourenço, D. Nickovic, G. J. Pace, J. Rufino, J. Signoles, D. Traytel, and A. Weiss. A survey of challenges for runtime verification from advanced application domains (beyond software). *Formal Methods Syst. Des.*, 54(3):279–335, 2019.

[18] S. Stoller, E. Bartocci, J. Seyster, R. Grosu, K. Havelund, S. Smolka, and E. Zadok. Runtime verification with state estimation. In S. Khurshid and K. Sen, editors, *RV*, volume 7186 of *Lecture Notes in Computer Science*, pages 193–207. Springer, 2011.

[19] M. Tiger and F. Heintz. Incremental reasoning in probabilistic signal temporal logic. *Int. J. Approx. Reason.*, 119:325–352, 2020.

[20] S. Wang, A. Ayoub, O. Sokolsky, and I. Lee. Runtime verification of traces under recording uncertainty. In S. Khurshid and K. Sen, editors, *RV*, volume 7186 of *Lecture Notes in Computer Science*, pages 442–456. Springer, 2011.

[21] D. Yuan, S. Park, P. Huang, Y. Liu, M. M. Lee, X. Tang, Y. Zhou, and S. Savage. Be conservative: Enhancing failure diagnosis with proactive logging. In C. Thekkath and A. Vahdat, editors, *OSDI*, pages 293–306. USENIX Association, 2012.

Appendix: Proofs

Proof of Theorem 2

Proof. Let ψ be a propositional formula. We create the propositional machine made of a single state s, with a single transition $s - \psi \longrightarrow s$. Let $\overline{\varphi} = \top$ be the multi-trace made of the single multi-event $\top$. By Definition 5, the sequence $s - \psi \longrightarrow s$ is a run of M for the trace made of the single multi-event ψ if and only if $[\![\top]\!]_\top \cap [\![\psi]\!]_\top \neq \emptyset$, i.e. if ψ is satisfiable. This shows that the problem is NP-hard.

Let $\psi_0, \ldots, \psi_{n-1}$ be the sequence of formulas such that ψ_i is the formula associated to the transition $s_i \longrightarrow s_{i+1}$ in M. For each i, define the set of propositional variables A_i as $\{a_1^i, \ldots, a_m^i\}$. Given a formula φ over $\mathcal{A}$, the renaming of $\mathcal{A}$ to $\mathcal{A}^i$, noted $\rho_i(\varphi)$, is the propositional formula obtained by replacing a_j by a_j^i for every $j \in [1, m]$. Let $\hat{\varphi}$ be the propositional formula defined as:

$$\hat{\varphi} \triangleq \bigwedge_{i=0}^{n-1} (\rho_i(\varphi_i) \wedge \rho_i(\psi_i))$$

The sequence $s_0 - \psi_0 \longrightarrow \cdots - \psi_{n-1} \longrightarrow s_n$ is a run of M for $\overline{\varphi}$ if and only if $\hat{\varphi}$ is satisfiable, hence the problem is in NP. □

Proof of Theorem 3

Proof. Let φ be a multi-event, $s_A \in S_A$, $s_P \in S_P$ be states in the access proxy and the multi-monitor, and $\sigma : S_P \longrightarrow \mathbb{N}$. For some multi-trace $\overline{\varphi} = \varphi_0, \ldots, \varphi_{n-1}$ and every state $s \in S_P$, $\sigma(s)$ is the number of runs of the form $s_0 - \varphi_0 \longrightarrow \ldots - \varphi_{n-1} \longrightarrow s$ in π_P. For every $s' \in S_P$, $\sigma'(s')$ is the number of runs of the form $s_0 - \varphi_0 \longrightarrow \ldots - \varphi_{n-1} \longrightarrow s' - \varphi \longrightarrow s''$ in π_P.

Moreover, we have that for every iteration of lines 8–15, $nc = \sigma(s_P) + |[\![\psi_P]\!]_\top \cap [\![\varphi']\!]_\top|$; i.e. nc is the number of runs of the form $s_0 - \varphi_0 \longrightarrow \ldots - \varphi_{n-1} \longrightarrow s_P - \varphi \longrightarrow s'_P$. We can finally observe that for a given verdict $l \in \{\Omega, \emptyset, \epsilon\}$, $\beta(l)$ is the sum of all values nc in iterations where $\gamma_{\pi_P}(s_P, \psi_P, s'_P) = l$. In other words, $\beta(l)$ is the number of runs of $\overline{\varphi} \cdot \varphi$ in π_P that end up in a state labeled with verdict l. Then $\frac{\beta(v)}{\sum_{v' \in \{\Omega, \emptyset, \epsilon\}} \beta(v')}$ is the fraction of all runs that end up in this verdict, which is equal to $\rho_{\pi_P}^v(\pi_A(\overline{\varphi}))$. □