Optimal Finite-State Monitoring of Partial Traces

Peeyush Kushwaha[1(✉)], Rahul Purandare[1,2], and Matthew B. Dwyer[3]

[1] IIIT-Delhi, New Delhi, India
peeyush16254@iiitd.ac.in
[2] University of Nebraska-Lincoln, Lincoln, USA
rahul@unl.edu
[3] University of Virginia, Charlottesville, USA
matthewbdwyer@virginia.edu

Abstract. Monitoring programs for finite state properties is challenging due to high memory and execution time overheads it incurs. Some events if skipped or lost naturally can reduce both overheads, but lead to uncertainty about the current monitor state. In this work, we present a theoretical framework to model traces that carry partial information (like number of events lost), and provide construction for a monitor capable of monitoring these *partial traces* without producing *false positives* while reporting violations. The constructed monitor optimally reports as many violations as possible for the partial traces. We model several loss types of practical relevance using our framework.

Keywords: Runtime verification · Finite state properties · Optimization

1 Introduction

Monitoring the execution behavior of software goes back to the dawn of programming and is a standard practice, e.g., through logging, programmer inserted print statements, and assertions. In the late 1990s, researchers began to explore the use of formal specifications to define run-time monitors [1] which brought the expressive power of formal methods to monitoring. Such *run-time verification* techniques rely on a set of defined *events* which denote the occurrence of program behavior relevant to a property specification, e.g., the invocation of a particular method, along with associated data, e.g., method parameters. A run-time monitor observes a *trace* generated by a program execution, incrementally updates the *state* of the specified property, and reports a property violation when a violating state is reached.

Run-time verification is attractive because it complements sound static verification approaches that cannot scale to modern software systems. However, monitoring occasionally incurs high memory and execution time overheads. Researchers have proposed a range of techniques to deal with the challenge of reducing these overheads, while preserving violation detection e.g., [2–6]. In this paper, we consider the additional challenge of *partial trace* that arises in the deployment of run-time verification in realistic system contexts such as: networked and distributed systems where message loss or reordering may be inherent, real-time systems which may shed monitoring workloads to meet scheduling constraints, or web-based systems with quality-of-service guarantees may lead to suppressed monitoring. In such systems, the original trace may be perturbed by dropping events, reordering events, or dropping or corrupting data correlated with events.

Partial traces are problematic for existing run-time verification approaches since treating a partial trace the same as the original trace may lead to missing a property violation or falsely declaring a violating execution. Partial traces do not, in general, permit the same degree of precision as the original trace. The information loss restricts us to two choices: reporting unconfirmed violations (including false positives), or reporting only confirmed violations (but missing some violations). Our framework is able to model both, but we present our results in the context of the second choice, for reduced run-time overhead at the cost of missed violations (e.g. [7,8]).

The first choice may be equally useful for ensuring compliance (e.g. [9,10]) and we show how it is enabled by our framework in Sect. 5. Existing work on partial traces [7,10] studies specific types of losses but does not contain a general model that could be used to study other loss types.

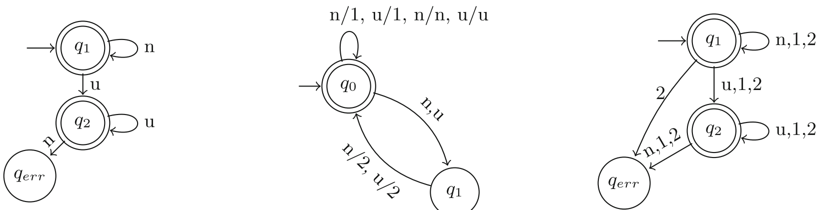
The paper makes foundational contributions to runtime verification by (a) *defining an expressive framework for modeling partial traces*, (b) *developing techniques for synthesizing provably complete and optimal verification monitors under those models*. Importantly, these results preclude the need for additional theory development for individual loss types and set the stage for more applied work and tool development. The main contributions of this paper are theoretical, complemented by (c) *a collection of diverse loss models*, including a discussion of how the event losses in current literature are specific instances of our general framework (Sects. 6, 7). For the expressiveness of our framework, we prove that *all* loss models are representable in our framework under these conditions: (i) the original property is a finite-state safety property, (ii) the desired monitor for partial traces is finite-state, and (iii) loss model can be represented as an arbitrary relation between original and partial strings.

2 Overview

We illustrate the problem by way of example and introduce the key insights behind the techniques we develop to address the problem.

Safety properties for run-time monitoring can be modelled using deterministic finite-state automata (DFA). An event is represented by a *symbol* and a trace by a *string* of symbols. Figure 1a shows the DFA for the *SafeIter* property which states that modification of a collection during iteration is not permitted. The DFA is expressed over the alphabet $\{n, u\}$ denoting accessing the (n)ext element in the iterator, and (u)pdating the collection being iterated.

Remark 1. Some literature uses an additional c "creation" event for this property. The interpretation of such events are implementation dependent. The implementation may use observance of this event to dynamically allocate memory for a monitor and instrument the code for other events just-in-time. In such a case, if the creation event is missed then the monitor is never created and the property is not monitored. For a statically allocated monitor, it may also be used as the fixed initial event – an effect equivalent to adding a state q_0 before q_1 with a transition c from q_0 and q_1. We model only the property after the creation event has happened and the monitor is ready to receive the first non-creation event, and leave implementation-dependent semantics out of our modelling. The framework, however, is trivially extensible to accommodate creation events, we discuss this extension in Sect. 6 along with an example where the role of creation events is more complicated.



(a) SafeIter Property as a DFA

(b) Loss Model as a NFT (ϵ outputs omitted for brevity)

(c) Alternate Property as a NFA

Fig. 1. Safe Iterator – after an iterator is created no updates (u) are permitted so long as next (n) elements remain to be accessed.

The state q_{err} is a trap state (self transitions ommitted for brevity) denoting the violation of the property. All violating strings include a subsequence $\ldots un \ldots$ indicating that an update was performed prior to accessing the next element. Statements that are free of such a subsequence end at one of the three accept states and are non-violating.

Loss may come in different forms. For example, symbols in a string may be erased (e.g., $n \rightarrow \epsilon$), reordered (e.g., $nu \rightarrow un$), or be modeled with only partial information (e.g., their count $nnn \rightarrow 3$).

To illustrate, we consider the case where symbols are dropped from the string, but the number of dropped symbols is recorded. This type of loss could be introduced intentionally as a means of mitigating excessive runtime overhead in monitoring, or it could occur naturally, for instance if an event arrives to the system strictly periodically, e.g., at 10 hz, but several of those inputs are missed in a row (due to load on the system, communication disruption, etc.).

We show how to continue monitoring while preserving fault detection capability. Consider a string $nnun$ where this loss is applied to the first two – we model the resulting string as $2un$. This could represent 4 possible strings starting with one of four $\{u, n\}^2$ and a suffix of un. For longer strings where sequences of length k are lost, the combinatorics of their possible replacements $\{u, n\}^k$ make it intractable to consider all of the possibilities. Despite this, the structure of Fig. 1a dictates that any string of the form kun violates the property, thereby illustrating that even with loss it is possible to perform accurate monitoring.

We formalize the intuition above in a *loss model* that maps symbols from the property to an alternative symbol set. For example, the loss model described above is defined by the mapping $\{u, n\}^k \mapsto k$. In Sect. 5, we show that all mappings of interest are a restricted class of relations on strings called rational relations. Rational relations can be represented by non-deterministi finite-state transducers (NFTs). An NFT maps strings in an alphabet, Σ, to strings in an alternative alphabet, Σ_a. Figure 1b shows the NFT with the mapping for alternate symbols 1 and 2 that lose the identity of symbols in a subsequence but retain its length, e.g. as in the $2un$ example we just discussed.

Retaining partial information about an event string might be insufficient to conclude that a violation occurred (or did not occur). We report a violation only when the partial information is sufficient to conclude that there must be a violation – such monitoring is *complete*[1] since it never reports a false violation. Consider the original string (O_1) in Fig. 2 and the set of 3 partial strings (L_1) induced by the NFT in Fig. 1b. Tracing through Fig. 1a on the first two partial strings by interpreting 1 and 2 as any individual or pair of symbols, respectively, leads only to the error state – since they preserve the fact that an n follows a u. These strings would be reported as violations. On the other hand, the string $nnuu$ (O_2) is non-violating and of the set of 3 partial strings (L_2) none reach *only* the q_{err} state. Completeness assures that no partial string can have u followed by a n and monitor in Fig. 1b won't report a false violation. The automaton which would observe these partial strings of alternate alphabet and give the states we discussed is shown in Fig. 1c.

Assuring completeness in violation reporting means, however, that the reporting of some violations may be missed. For example, the third partial string for L_1 suppresses all nu making it impossible to definitively conclude that the observed string is a violation. Our goal is to report violations on as many strings of alternate symbols as possible while maintaining completeness. When we refer to *optimality* of monitoring the partial trace, we are referring to this goal (discussed in Sect. 5).

	String	State(s)
O_1	nnunnun	q_{err}
L_1	2nun2n	$\{ q_{err} \}$
	n1unnun	$\{ q_{err} \}$
	nnu2u1	$\{ q_2, q_{err} \}$
O_2	nnuu	q_2
L_2	2uu	$\{ q_2, q_{err} \}$
	nnu1	$\{ q_2, q_{err} \}$
	2n2	$\{ q_1, q_2, q_{err} \}$

Fig. 2. Partial strings

In Sect. 4 we will show how loss models (such as the one in Fig. 1b) are defined, and how the property of interest (e.g. Fig. 1a) and the loss model are used to construct the alternate monitor (e.g. Fig. 1c) which would observe the partial trace and optimally monitor them. Discussion

[1] See Remark 5 for a discussion on the terms *soundness* and *completness*.

and significance for implementation of the theory developed in Sect. 3–Sect. 5 is given in Sect. 5.1.

3 Basic Definitions

3.1 Notation

1. $x \prec y$: string x is a proper prefix of string y.
2. $f : X \to Y$ is *lifted* to 2^X as given by $f(S) = \{ f(x) \mid x \in S \}$ $\forall S \subseteq X$.
3. $(\cdot)$ is string concatenation. $(\cdot)$ is lifted to sets of strings.
4. $\times$ is cartesian product of two sets. For a relation $R \subseteq X \times Y$, $R(x) = \{ y \mid xRy \}$ and $R^{-1}(y) = \{ x \mid xRy \}$
5. DOMAIN$(R) = \{ x \mid \exists y \textit{ s.t. } xRy \}$, RANGE$(R) = \{ y \mid \exists x \textit{ s.t. } xRy \}$.
6. A *partition* $\mathcal{P}$ of a set S is a set $\{ P_1, P_2, \dots \}$ such that P_i are pairwise disjoint nonempty sets (*equivalence classes*) with union S.
7. A *class representative* of P_i is a distinguished element in P_i.
8. For $s \in S$ in $\mathcal{P}$, $[s]_\mathcal{P}$: equivalence class, and $rep^\mathcal{P}(s)$: class representative.
9. $\blacksquare$: proof omitted (due to triviality/limited space/relevance).

Finite Automata and Properties. Familiarity with regular languages and their properties is assumed. A finite set of symbols is called an *alphabet*. $REG(\Sigma)$ is the set of all regular languages over an alphabet Σ. ϵ is the empty string, and Σ_ϵ is the alphabet $\Sigma \cup \{ \epsilon \}$. A *trace* is a (possibly infinite) sequence of events, and an *execution* is a finite prefix of a trace. A trace x is a *continuation* of an execution x' if $x' \prec x$.

3.2 Definitions

Definition 1 (Finite Automata). *A finite automaton is a 5-tuple* $(Q, \Sigma, \delta, q_0, F)$ *with a finite set of states* Q*, the alphabet* Σ*, a specified initial state* $q_0 \in Q$*, and the set of final states* $F \subseteq Q$*. A deterministic finite automaton (DFA) has the transition function* $\delta : Q \times \Sigma \to Q$ *and a nondeterministic finite automaton (NFA) has the transition function* $\delta : Q \times \Sigma \to 2^Q$*. The transition function* δ *is lifted to strings, sets of strings, and sets of states. We call* $L(A) = \{ x \in \Sigma^* \mid \delta(q_0, x) \in F \}$ *the* language *of the finite automaton.*

Definition 2 (Nondeterministic Finite-State Transducers (NFTs)). *Defined as a NFA* $(Q, \Sigma, \delta, q_0, F)$ *with an extra alphabet* Γ *(labelled the "output" alphabet), where* $\delta : Q \times \Sigma \to 2^{Q \times \Gamma_\epsilon}$*. After observing a symbol* $\sigma \in \Sigma$*, the NFT in state* q *transitions to a choice of* q' *with output* $\gamma \in \Gamma_\epsilon$ *where* (q', γ) *is one of the pairs in* $\delta(q, \sigma)$*.*

Definition 3 (Finite-state safety property). *A finite-state property* ϕ *is the minimal-state DFA* $\phi = (Q, \Sigma, \delta, q_0, Q \setminus \{ q_{err} \})$ *with the specified error state* q_{err}*. The error state* q_{err} *must be a* trap state*, i.e.* $\forall \sigma \in \Sigma$, $\delta(q_{err}, \sigma) = q_{err}$*. For a property* ϕ*, the notation* $Q^\phi, \Sigma^\phi, \delta^\phi, q_0^\phi$*, and* q_{err}^ϕ *is used to refer to* Q, Σ, δ, q_0*, and* q_{err} *respectively. An execution* $x \in \Sigma^*$ violates *the property* ϕ *if* $\delta(q_0, x) = q_{err}$*. An execution* x *that does not violate the property is* non-violating.

Let $n \in Z^+$, $\Gamma = \Sigma \cup \{1 \ldots n\}$, $R \subseteq \Sigma \times \Gamma$ s.t.:

$$aRb \text{ iff } \begin{cases} a = b & b \in \Sigma \\ |a| = b & b \in \{1 \ldots n\} \end{cases}$$

(a) Dropped-count Loss. The formal definition of loss type from Sect. 2.

Let $\Delta \subseteq \Sigma$ be symbols which may be skipped.

$$\text{Let } \Gamma = \{ x' \mid x \in \Sigma \}$$
$$R = \{ (a, b') \mid a \in \Delta^* b \text{ and } b \in \Sigma \}$$

(b) Silent Drop Loss: several dropped Δ symbols may precede an observed Γ symbol b. The marker $'$ is used here just to visually distinguish symbols in Γ from Σ.

Fig. 3. Formally specified loss types

Remark 2. Complement of language of automaton ϕ: $L(\phi)^C = \Sigma^* \setminus L(\phi)$ are all the strings violating ϕ. If an execution violates a property, then so do all its continuations (because q_{err} is a trap state).

Definition 4 (Monitors). *A* monitor M_ϕ *observes events for a property* ϕ. M_ϕ *has a* current state q_{curr}, *initialized as* $q_{curr} = q_0$, *and updated as* $q_{curr} \leftarrow \delta(q, \sigma)$ *on observing* $\sigma \in \Sigma$. *The* verdict *of a monitor on a trace is* true *if the property* ϕ *cannot be violated in any continuation,* false *if the property has been violated,* inconclusive *if neither.*

Remark 3 (Valuedness). Our definition of monitors considers 3 verdicts. However, in theoretical development of our result we only care about "violation" or "no violation", i.e. "false" verdict is distinguished from "inconclusive" and "true" but the latter two are not distinguished from each other.

The terms "monitor" and "property" are used interchangeably when clear from the context (e.g. language of a monitor). The analysis in the following sections is not affected by the existence of multiple monitors. Therefore we omit any discussion of it till our discussion of a particular loss type in Sect. 6.

4 Losses, Alternate Monitors, Soundness and Optimality

This section and the next contain our theoretical results. We do not *assume* a loss model R must be an NFT, but rather prove it (in the next section). That's why we begin with a general definition of a loss model with minimal restrictions (only to exclude cases without valid interpretation[2]).

Definition 5 (Loss Model). *A* loss model *for an original alphabet* Σ *to an alternate alphabet* Γ *is a relation* $R \subseteq \Sigma^* \times \Gamma$ *satisfying:*

1. No spurious alternate symbols condition: $\epsilon \notin \text{DOMAIN}(R)$

[2] The first condition excludes partial symbols produced not corresponding to any original information. The second condition excludes existence of full traces which are completely lost, i.e. for which no partial information is ever observed.

2. *Prefix existence condition: There is an upper-bound $n \in \mathbb{Z}^+$ such that for all strings $x \in \Sigma^*$ of length n or more (i.e. $|x| \geq n$) at least one prefix $x' \preceq x$ is a part of* $\text{DOMAIN}(R)$ *(i.e. $x' \in \text{DOMAIN}(R)$)*

We assign the loss model R the interpretation that $(x, \gamma) \in R$ means that if a symbol γ is observed in alternate trace $y \in \Gamma*$, then it was produced in lieu of one of the strings $x \in R^{-1}(\gamma)$.

We use a related term *loss type* to roughly denote a family of related loss models. As an example, the loss type for the loss model from Sect. 2 is given in Fig. 3a, which is parameterized over various choices of Σ and n (and where for a specific choice of Σ and n we'd obtain a *loss model* of that *loss type*).

To motivate our next definition, consider the partial traces $y_1 = 2n2$ and $y_2 = 3uu$ from Fig. 2 for the corresponding original trace $x = nnnuu$ (O_2). For y_1, as the program runs the original monitor observes n and alternate monitor observes nothing. On the next event original monitor again observes n and alternate monitor observes 2. For every prefix of x we have a corresponding prefix of y_1, and we may relate them with a *filter* function f_1 such that $f_1(n) = \epsilon, f_1(nn) = 2, \dots f_1(nnnuu) = 2n2$. f_1 will be a partial function defined only on prefixes of x. We could similarly define filter f_2 for the relation between prefixes of x and prefixes of y_2. Not all functions between Σ^* and Γ^* would satisfy the loss model R – one obvious limitation would be that filter f must produce outputs for a string x which are consistent with what R relates to prefixes of x. We formalize these requirements on a function and call a function that satisfies them a *filter*, as defined next:

Definition 6 (Filter, Partial Traces, Segments, and Replacements). *Let Σ and Γ be finite alphabets, and $x \in \Sigma^*$ be a trace, and $P_x = \{ x' \mid x' \preceq x \}$ be the set of all finite prefixes of x. Consider a* loss model $R \subseteq \Sigma^* \times \Gamma$. *Then a function $f : P_x \to \Gamma^*$ is called a* filter *under R if it satisfies the* monotonicity *property, defined below:*

if $f(x) = y$ and $f(x') \neq y$ for all proper prefixes x' of x, then:

$$f(x \cdot s) = \begin{cases} y \cdot \gamma & \textit{if } sR\gamma \\ y & \textit{otherwise} \end{cases}$$

In the first case γ is called a replacement *(symbol) for the* segment *s (sub-string) of the string $x \cdot s$.*

$\mathcal{F}_R$ is defined as the set of all possible functions which are filters under R.

Definition 7 (Completions). *We're also interested in possible strings $x \in \Sigma^*$ which could have lead to the observation y. If $\exists f \in \mathcal{F}_R$ such that the last symbol in x marks end of a segment and produces the last symbol in y, i.e. $f(x) = y \wedge \nexists x' \prec x, f(x') = y$, then we call x a* completion *for y. We define $\mathfrak{C}(y)$ as the set of all completions for y:*

$$\mathfrak{C}(y) = \{ x \in \Sigma^* \mid \exists f \in \mathcal{F}_R \textit{such that } f(x) = y \wedge \nexists x' \prec x \wedge f(x') = y \}$$

Next, we show how to compute the set of completions $\mathfrak{C}(y)$ using R^{-1}.

Theorem 1. *For a string $y \in \Gamma^*$, $y = \gamma_1\gamma_2 \dots \gamma_k$:*

$$\mathfrak{C}(y) = R^{-1}(\gamma_1) \dots R^{-1}(\gamma_k)$$

Proof. (LHS $\subseteq$ RHS) Let $x \in \mathfrak{C}(y)$. Let $y = y'\gamma_k$. From definition of filters and the condition $\nexists x' \prec x, f(x') = y$ in definition of $\mathfrak{C}(y)$ we can partition x into two substrings $x = x'x_k$ where x_k is the corresponding segment for the replacement γ i.e. $x_k R \gamma_k$. Repeatedly applying this reasoning (e.g. in the next step for y' and x'), we conclude $\exists$ a partition $x = x_1 \dots x_k$ such that $\gamma_1, \dots, \gamma_k$ are a replacements for respective x_i. Then $x_i \in R^{-1}(\gamma_i)$ and thus $x \in R^{-1}(\gamma_1) \dots R^{-1}(\gamma_k)$. *(RHS $\subseteq$ LHS)* Let $x \in R^{-1}(\gamma_1) \dots R^{-1}(\gamma_k) \implies x = x_1 \dots x_k \implies f(x) = \gamma_1 \dots \gamma_k = y$. □

We now give definitions of alternate monitors and their soundness, completeness, and optimality. We present the construction in the next section.

Definition 8 (Alternate monitor). *Given a* primary *monitor M_ϕ and a loss model $R \subseteq \Sigma^* \times \Gamma$, an alternate monitor M_ψ is any finite state monitor over the alphabet Γ that observes the partial execution $f(e)$ for any $f \in \mathcal{F}_R$ when M_ϕ observes the execution e. We call $(M_\phi, M_\psi)_R$ a* primary-alternate monitor pair *and $(\phi, \psi)_R$ a* primary-alternate property pair*. We also refer to these as just* monitor pair *or* property pair*.*

Remark 4. It is useful to consider the monitors in $(M_\phi, M_\psi)_R$ as monitoring together for the purposes of theoretical analysis and for definitions. In practice, we want to monitor using just M_ψ.

Definition 9 (Soundness and Completeness for a property pair). *For $(\phi, \psi)_R$, with the definition of $\mathfrak{C}$ lifted to the set of strings, we define:*

Soundness: *A non-violating trace must not have any violating completions, i.e. $y \in L(\psi) \implies \mathfrak{C}(y) \subseteq L(\phi)$, equivalently $\mathfrak{C}(L(\psi)) \subseteq L(\phi)$.*

Completeness: *A violating trace must have all violating completions, i.e. $y \notin L(\psi) \implies \mathfrak{C}(y) \subseteq L(\phi)^C$, equivalently $\mathfrak{C}(L(\psi)^C) \subseteq L(\phi)^C$.*

Remark 5. Our definitions of soundness and completeness of a monitor are consistent with the definitions commonly used in the literature on program analysis and verification [11]. There also exist other definitions, particularly in the literature on runtime verification, which essentially interchange the interpretations of soundness and completeness of a monitor as defined here.

Definition 10 (Optimality of a property pair). *For $(\phi, \psi^*)_R$ where ψ^* is complete, ψ^* is called optimal if for any other* complete *property pair $(\phi, \psi)_R$, $L(\psi^*) \subseteq L(\psi)$, or equivalently $\mathfrak{C}(L(\psi^*)) \subseteq \mathfrak{C}(L(\psi))$. We call $(\phi, \psi^*)_R$ an* optimal pair*.*

Remark 6. Our definition of Optimality is a strong definition. An alternate definition for an optimal monitor might be to count the number of strings up to any given length and define a monitor which reports a violation on maximum number of strings for every length as the optimal monitor, but optimality by our definition would imply optimality in this alternate definition.

Definition 11 ($L_{opt}(\phi, R)$). *For a property ϕ and loss model R, $L_{opt}(\phi, R) = \mathcal{F}_R(L(\phi))$, i.e. L_{opt} is the set of partial traces in Γ^* produced by a non-error execution in Σ^*. L_{opt} is the smallest set of strings on which a complete alternate monitor cannot reach a false verdict. i.e. $L_{opt} = \mathcal{F}_R(L(\phi)) = \{\, y \mid \mathfrak{C}(y) \cap L(\phi) \neq \varnothing \,\}$.*

There are several definitions of monitorability in runtime verification literature [12]. The most suitable to our problem is the ability to report a violation. Since the alternate monitor is allowed to miss violations, the verdict that there are no more errors is not useful.

Definition 12 (Monitorability). *If there exists a continuation of an execution which leads to the* false *verdict, then the monitor's current state is* monitorable. *In a finite automaton, monitorability of a state q can be checked by path-reachability from q to the error state.*

5 Optimal Monitor and Losses as Transducers

In this section, we first construct an optimal monitor using the definition of R introduced in the previous section. We then use that construction and its properties to prove that loss models beyond what can be represented by a finite-state transducer can be removed from our consideration. Finite-state transducers have a regular structure to deal with which makes impelmentation more feasible, and theoretical treatement easier. So removing other loss models from our consideration restricting R to the space of loss models to these transducers is good news for us, as long as we can get the assurance that we're not missing any expressivity (which we'll show).

We begin with the construction of an optimal alternate monitor and proof of optimality. The key idea for the construction is for the optimal alternate monitor to keep track of the set of states that the primary monitor could possibly be in.

Theorem 2. *For a property ϕ and a loss model R, we construct the NFA ψ with δ^{ψ} as $\forall q \in Q,\ \delta^{\psi}(q, y) = \delta^{\phi}(q, R^{-1}(y))$[3]. After determinizing and minimizing this NFA, we obtain the optimal alternate monitor ψ^* which recognizes L_{opt}.*

In order to prove this theorem, we first need to establish correspondence of states between the NFA constructed and its determinized and minimized version. We already know that using powerset construction [13] for NFA determinization, the states of determinized DFA are labelled by subset of NFA states. We use the result that this labelling is well-defined even after minimization.

[3] Note that $R^{-1}(y)$ must be decidable for this construction to be well-defined. This is not an issue as we later prove that R must represent a NFT, and for NFTs computing $R^{-1}(y)$ is decidable.

$$\delta^{\psi*}(q,x) = \begin{cases} \delta(q,\Sigma^x) & if\ x \in \{1 \dots n\} \\ \delta(q,x) & otherwise \end{cases}$$

(a) Disabling monitoring for up to n events, defined on filter from Fig. 3a

$$\delta^{\psi*}(S,x) = C_\Delta(\delta(C_\Delta(S),x))$$

C_Δ is the Δ-closure of the set of states S in M, i.e. set of all states which can be reached from states in S by following 0 or more y-transitions, where $y \in \Delta$

(b) Silent drop monitor, defined on Δ and the filter from Fig. 3b

Fig. 4. Example constructions of δ^{ψ^*} for optimal pair $(\phi, \psi^*)_R$ Theorem 2

Lemma 1. *In DFA minimization [13] of a determinized NFA, let $\mathcal{P}$ be the paritition of 2^Q where $\mathcal{S} \in \mathcal{P}$ represents a set of states merged together. If the states S_1 and S_2 merge, then the state $S_1 \cup S_2$ merges with them. i.e. $\forall\ \mathcal{S} \in \mathcal{P}, \forall\ S_1, S_2 \in \mathcal{S} \implies S_1 \cup S_2 \in \mathcal{S}$.* ■

Remark 7. Using the previous lemma, for each class $[S]_\mathcal{P}$ of states, the class representative $rep^\mathcal{P}(S)$ of S is defined as $\bigcup_{T \in [S]_P} T$. We label the resultant state from merged states in $[S]_P$ in the minimized DFA by $rep^P(S)$.

Lemma 2. *Let (M_ϕ, M_{ψ^*}) be the optimal monitor pair. When M_{ψ^*} transitions to a state $S \subseteq Q$ and M_ϕ is in the state q then $q \in S$.*

Proof. We apply induction on the number of symbols observed by the alternate monitor.
Base Case holds because start state of M_ψ is $\{ q_0^\phi \}$.
IH: Assume that $q \in S$ after n symbols and $n+1$th replacement symbol γ is observed in lieu of segment x.
IS: M_ψ transitions to $S' = \delta^\psi(S,\gamma) = \delta^\phi(S, R^{-1}(\gamma))$ and M_ϕ transitions to $q' = \delta^\phi(q,x)$. But since $q \in S$ and $x \in R^{-1}(\gamma)$, $q' \in S'$ □

Proof (Theorem 2). Subproof 1: $y \notin L(\psi^*) \implies y \notin L_{opt}$. Using Lemma 2

$$\begin{aligned} y \notin L(M_\psi) &\implies \delta^\psi(q_0,y) = \{ q_{err} \} \implies q \in \{ q_{err} \} \implies q = q_{err} \\ &\implies \forall x \in \mathfrak{C}(y),\ \delta(q,x) = \{ q_{err} \} \implies \mathfrak{C}(y) \subseteq L(M)^C \end{aligned}$$ □

Subproof 2: $y \in L(\psi^*) \implies y \in L_{opt}$. Consider $y \in L(\psi^*)$.

$$\begin{aligned} &\implies \delta^\psi(\{ q_0 \}, y) \neq \{ q_{err} \} \\ &\implies \delta^\psi(\dots \delta^\psi(\delta^\psi(\{ q_0 \}, y_1), y_2) \dots), y_n) \neq \{ q_{err} \} \\ &\implies \delta^\phi(\dots \delta^\phi(\delta^\phi(\{ q_0 \}, R^{-1}(y_1)), R^{-1}(y_2)) \dots), R^{-1}(y_n)) \neq \{ q_{err} \} \\ &\implies \delta^\phi(\{ q_0 \}, \mathfrak{C}(y)) \neq \{ q_{err} \} \implies y \in L_{opt} \end{aligned}$$ □

Corollary 1. *Property ϕ is monitorable under R iff the state labeled with singleton error state $\{ q_{err} \}$ is reachable in ψ^*.*

Figure 4 and the next section show example optimal monitor constructions.

Remark 8. Our definition of ψ^* is *constructive*, so ψ^* always exists. I.e. given a property ϕ and loss model R we can always construct ψ^* using Theorem 2. However, this says nothing about usefulness of ϕ^*. For instance, it may be the case that ψ^* is unmonitorable, which would tell us that $L_{opt}(\phi, R) = \Gamma^*$. Due to being the *optimal* construction, this just demonstrates that it's *impossible* to monitor under the loss model R (if ϕ itself was monitorable). This is an indication that too much information is lost under the loss model R for the resulting partial string to be of any use for monitoring ϕ.

We have a construction for optimal alternate monitors under the loss model R. Recall that we defined a *loss model* with minimal constraints in Definition 5, which permits us to define loss models of arbitrary complexity by more complex loss types (e.g. a loss type which could be represented by a transducer with a stack). We find that as long as the monitor observing the trace is constrained to be a finite-state automaton, the partial information of loss computed by more complex loss types (e.g. a loss type which could be represented by a transducer with a stack) does not help in reporting more violations, and we show that by showing that for every loss model which isn't a NFT, there is a loss model which is an NFT and results in construction of the same ψ^*.

Theorem 3. *Let $(\phi, \psi^*)_R$ be the optimal property pair where R may not be representable by NFT. Then there exists a loss model R' which can be represented as a NFT for which the constructed alternate property is also ψ^*.*

The following lemma handles the major part of the proof:

Lemma 3. *For an alternate symbol $\gamma \in \Gamma$, if the set of strings $X = R^{-1}(\gamma)$ is not regular, then we can come up with a set Y such that $X \subset Y$ and Y is regular and $\forall q \in Q\ \delta^\phi(q, X) = \delta^\phi(q, Y)$*

Proof. We'll use the shorthand $f(q) : Q \to 2^Q$ for $\delta(q, R^{-1}(\gamma))$, since γ is fixed. Consider $l(q) : Q \to REG(\Sigma)$, the regular language taking us from q to $f(q)$, i.e. $l(q) = \{\, x \mid x \in \Sigma^* \wedge \delta(q, x) \in f(q) \,\}$. Let $Y = \cap_{q \in Q}\ l(q)$. It follows that Y is regular since regular languages are closed under intersection.

$$\begin{aligned}
&\implies \forall q \in Q\ R^{-1}(\gamma) \subseteq l(q) \implies R^{-1}(\gamma) \subseteq Y\\
&\implies R^{-1}(\gamma) \subset Y \qquad\qquad (R^{-1}(\gamma) \neq Y \because R^{-1} \text{ is not regular and } Y \text{is})\\
&\text{Now } \forall q \in Q\ \{Y \subseteq l(q) \implies \delta(q, Y) \subseteq f(q)\}\\
&\text{And } R^{-1}(y) \subseteq Y \implies \forall q \in Q\ \delta(q, Y) \supseteq f(q)\\
&\implies \delta(q, Y) = f(q) = \delta(q, R^{-1}(\gamma)) \qquad\qquad \square
\end{aligned}$$

Rest of the proof uses this lemma to construct the new R' for every γ. It requires additional definitions of generalized automata and transducers which allow regular expressions on transitions instead of symbols, which we now give:

Definition 13 (Generalized Nondeterministic Finite Automaton *[14]***).** *A generalized nondeterministic finite automaton* (GNFA) *is a 5-tuple* $(Q, \Sigma, \delta, q_0, f)$*, where* Q *is the finite set of states,* Σ *is the alphabet,* $\delta \subseteq (Q \setminus f) \times (Q \setminus \{ q_0 \}) \rightarrow REG(\Sigma)$ *is the transition function, and* $q_0, f \in Q$ *are the specified initial and final states.*

Definition 14. (Generalized Nondeterministic Finite-State Transducers (GNFTs)). *Defined as a GNFA* $(Q, \Sigma, \Gamma, \delta, q_0, f)$*, where* $\delta : (Q \setminus f) \times (Q \setminus q_0) \rightarrow 2^{REG(\Sigma) \times \Gamma}$*. After observing a string* $x \in \Sigma^*$*, the NFT in state* q *transitions to a choice of* q' *with output* $\gamma \in \Gamma$ *where* $(r,\ \gamma)$ *s.t.* $x \in L(r)$ *is one of the pairs in* $\delta(q,\ q')$*.*

Remark 9. GNFA can be converted to NFA [13], similarly GNFT to NFT. ■

Using these definitions and Lemma 3, we proceed to prove Theorem 3:

Proof (Theorem 3). The construction in Theorem 2 uses $\delta^{\phi}(q,\ R^{-1}(\gamma))$ for defining δ^{ψ}. Therefore it is sufficient to produce an R' representable by a NFT such that $\delta^{\phi}(q,\ R'^{-1}(\gamma)) = \delta^{\phi}(q,\ R^{-1}(\gamma))\ \forall \gamma \in \Gamma$. In rest of the proof, we use δ to denote δ^{ϕ}.

Consider a symbol $\gamma \in \Gamma$.
Case 1: ($R^{-1}(\gamma)$ is regular) We define $xR'\gamma\ \forall xR\gamma$. So $R'^{-1}(\gamma) = R^{-1}(\gamma)$ and thus $\delta(q,\ R'^{-1}(\gamma)) = \delta(q,\ R^{-1}(\gamma))$
Case 2: ($R^{-1}(\gamma)$ is not regular) We construct $R'^{-1}(\gamma)$ by the rational set constructed in Lemma 3.

We construct a GNFT for R'. Consider a GNFT with states $\{ q_0, q, f \}$, ϵ-transitions from q_0 to q and q to f, and self loop edges on $q\ \forall \gamma \in \Gamma$ with input label as the regex of $R^{-1}(\gamma)$ and output label γ. This completes the construction. □

Next we show that an even more relaxed definition of loss model (a relation between Σ^* and Γ^* instead of Σ^* and Γ) does not increase loss models we can express.

Theorem 4. *For a primary-alternate pair* (ϕ, ψ) *with where* ψ*'s input on* $x \in \Sigma^*$ *is filtered by* f *from a loss model* $R \subseteq \Sigma^* \times \Gamma^*$*, we can define a finite state property* ψ'*, alternate symbol set* Γ'*, loss model* $R' \subseteq (\Sigma^* \times \Gamma')$ *with filter* f' *such that* $\forall q \in Q^{\phi}\ \delta^{\psi}(q, f(x)) = \delta^{\psi'}(q, f'(x))$

Proof. The proof exploits the observation that for a finite-state automaton ψ, the set of possible symbols which denote a unique transition are finite. For a symbol $\gamma \in \Gamma$, from each state there is a choice to transition to another state, resulting in an upper bound of $|Q|^{|Q|}$ unique transitions. RANGE(R) may be unbounded but we partition it into a finite number of equivalence classes such that a class representative may instead be used to denote the transition. Partition all strings in Γ^* using the relation $\sim$ defined as : $a \sim b \iff \forall q \in Q\ \delta(q, a) = \delta(q, b)$. It is easy to check that $\sim$ is reflexive, symmetric and transitive.

We will construct ψ' as $(Q^{\psi}, \Gamma', \delta', q_0^{\psi}, Q^{\psi} \setminus \{ q_{err}^{\psi} \})$, i.e. with a new alphabet and transition function. Let states of ψ be indexed by $i \in 1 \ldots |Q^{\psi}|$. Define Γ' to have symbol set of $|Q^{\psi}|$-tuples $(t_1, \ldots, t_{|Q^{\psi}|})$ and define δ' such when the tuple $\gamma' \in \Gamma'$ is encountered as a symbol by δ', the ith entry of γ' denotes the state transition from state i, i.e. $t_i \in 1 \ldots |Q^{\psi}|$ and $\delta'(i, (\ldots, t_i, \ldots) = t_i)$. We define a map m an equivalence class of $\sim$ to tuple by taking any member $y \in \Gamma^*$ of the equivalence class and mapping the class to $(\delta^{\psi}(1, y), \ldots, \delta^{\psi}(|Q^{\psi}|, y))$. Now we can define $R' = \{ (x, m(y)) \mid (x, y) \in R \}$, and $f'(x) = m(f(x))$ Then, for a $x \in \Sigma^*$, $\forall q \in Q^{\phi}$ $\delta^{\psi}(q, f(x)) = \delta^{\psi'}(q, m(f(x))) = \delta^{\psi'}(q, f'(x))$. □

Remark 10 (Sound alternate monitors). We can also construct a property pair $(\phi, \psi)_R$ which is sound and may be incomplete by using a construction similar to that in Theorem 2 by determinizing it and updating $\delta^{\psi}(S,\ \gamma) \leftarrow \{q_{err}\}$ if $q_{err} \in \delta(S,\ R^{-1}(\gamma))$. It can be argued in a similar fashion that this construction is optimally complete among all sound alternate monitors ■

5.1 Discussion and Significance for Implementation

Theorem 3 and Theorem 4 complete our claim that *any* loss type (arbitrary relation between original and partial strings) for which the final produced property has to be be finite-state is representable in our framework.

State Size. After determinization, number of states in DFA may be high. This is not a problem in practice, e.g. finite-state properties from the largest publicly available database of properties in [15] all have fewer than 10 states. Still, some existing techniques (e.g. [16]) may be used to further reduce number of states, at the expense of missing more violations. We leave it to future work for DFA size reduction techniques specific to monitoring.

Complexity. Because R can be arbitrary, we need to show $\delta^{\phi}(q, R^{-1}(y))$ is efficiently computable. We show that our construction of the NFA is polynomial, though determinizing the constructed NFA may be exponential in state size. It should also be noted that both these costs are incurred at static time, and once computed, there is no run-time overhead and each partial event is processed in $O(1)$ time by the determinized optimal alternate NFA.

Remark 11. Construction of alternate optimal NFA in Theorem 2 takes polynomial time when R is represented as a NFT.

Proof. To compute $\delta(q, R^{-1}(y)) = S$: the intersection of $R^{-1}(y)$ and the regex formed by the set of strings which go from q and some q' is non-empty thus $q' \in S$. These intersection and non-empty checks take polynomial time [13]. We loop over $O(n)$ states and check if each is in S. We repeat this for every (γ, q) pair making $O(|\Sigma \times Q|)$ iterations.

Monitorability (under a Loss Model, and at Runtime). Corollary 1 reduces the question of monitorability under a loss model to reachability of error state from q_0 in our constructed optimal monitor. i.e. since our constructed monitor is optimal, if it does not have capability of producing violations without false positives, then it is not possible to construct *any* DFA which consumes partial trace and can report a violation. In case of natural losses, this can serve as a test for checking if monitoring is even possible for a property under the loss, and provide hints to what partial information, if it could be recorded, would help monitorability. In case of artificial losses, this can serve as a way to discriminate between available loss types to exclude those which aren't monitorable. In context of Remark 3, Corollary 1 is also useful in finding the "true" verdict. Monitoring can be disabled at runtime as soon as the optimal monitor enters into a state from which the state $\{ q_{err} \}$ is unreachable in alternate monitor.

Sample Implementation. To provide a starting point for an implementation in a monitoring system, we provide a sample implementation at [17]. It contains an implementation of Theorem 2 which takes a property and a loss model as input and automatically constructs optimal alternate monitor as output. The implementation verifies some examples provided in this paper, allows new properties and loss models to be defined, and allows simulating original and partial traces against the primary as well as the constructed optimal alternate properties.

6 Framework Instantiations

We presented how our framework applies to loss types such as those in Fig. 3. In this section, we describe three more instantiations of the framework that illustrate the variety of realistic event loss models it can accommodate.

Bounded Frequency Count of Missed Symbols. This loss model was considered in [10] for lossily compressing event traces over a slow network. It is a modification of the dropped-count filter in Fig. 3a where additional information about observed symbols is kept. For a bound $n \in Z^+$ and alphabet Σ with symbols σ_i indexed by $0 \leq i < |\Sigma|$, we'll define alternate symbol set Γ where each symbol is a $|\Sigma|$-tuple, tuple entry at an index i being number of dropped symbols σ_i. That is, $\Gamma = \{ (c_1, c_2, \ldots, c_{|\Sigma|}) \mid 0 < c_1 + \ldots + c_{|\Sigma|} \leq n \}$ Let $\#x(y)$ denote number of characters x in string y. The loss model is defined as $R = \{ (x, (c_1, c_2, \ldots c_{|\Sigma|})) \mid \bigwedge_{i \in I} c_i = \#\sigma_i(x) \}$.

As an example, we may have $f(babaab) = (a_1, b_2)(a_1, b_0), (a_1, b_1)$ for $\Sigma = \{ a, b \}, n = 3$.

There are two key differences between our formalization and that of [10]. First, the total size of the missed symbols is bounded in our case so that we have a finite alphabet with each transition taking $O(1)$ time in the determinized alternate DFA. [10] uses a constraint automata which accepts an infinite alphabet and each transition takes $O(|Q|)$ time. We note that even if more than n symbols are missed at a time, then up to mn missed symbols can produce m alternate symbols to transition to the correct set of states in our framework. The second difference is in consideration of soundness and completeness. While we construct a complete optimal monitor (without any false positives), they construct a sound monitor (without any false negatives). As Remark 10 shows, this is not an issue since we can easily construct a sound monitor instead.

Merged Objects. Here we look at a new loss type which loses information about which object an event belongs to in a multi-object monitor. Let $O = \{ o_1 \dots o_m \}$ be a set of objects with parametric events $E = e_1 \dots e_n$, i.e. $e_1(o_1)$ is a distinct event from $e_1(o_2)$. This means that $\Sigma = \{ e(o) \mid e \in E \wedge o \in O \}$.

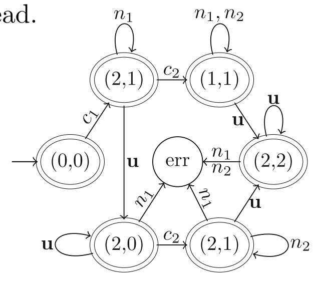


Fig. 5. A composite monitor for SafeIter on two iterators.

For $\sigma \in \Sigma, \gamma \in \Gamma$, let $\sigma R \gamma$ iff $\sigma = \gamma(o) \wedge o \in O$. In the partial trace, we lose information about which object the event belongs to within O. For the general case we can build an optimal monitor using the construction in Theorem 2. We give an example of the multi-object property "SafeIter" shown in Fig. 1a but including additional creation events with symbol c as discussed in Remark 1.

The fact that the event c will not be observed in any state is not a part of the property but a guarantee of the environment. We model the transitions for c from any non-initial state other to as special state q_{im} (not shown) representing "impossible" state.

A composite monitor for the SafeIter property from [6] is shown in Fig. 5 for two iterators I_1 and I_2 ((i, j) represents states (q_i, q_j) for the two iterators, assuming I_1 is created first). The loss model R merges events for I_1 and I_2, and using Theorem 2 to construct the optimal alternate monitor, and delete the q_{im} state from the NFA produced (as we're guaranteed by the environment to never enter it) to obtain the monitor in Fig. 6. For example, the obtained monitor only misses the violation for the traces like $c_1 u c_2 {n_2}^* n_1$, i.e. when we actually need the information that event n happened on object 1, but can still report violations for traces matching $c_1 n_1 c_2 (n_1|n_2) u u^* (n_1|n_2)$ or $c_1 u c_2 (n_2|n_1)^* u u^* (n_1|n_2)$.

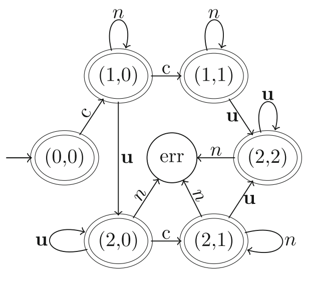


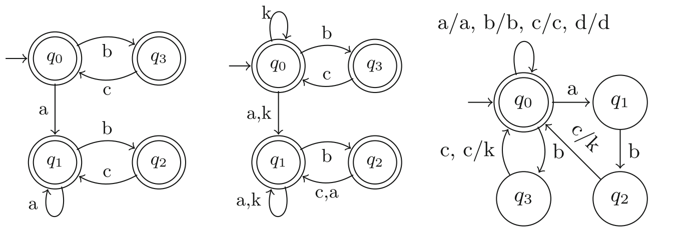
Fig. 6. Optimal complete alternate monitor.

Missing Loop Events. Significant number of events can be generated within loops in a program. [5] addresses this by eliminating instrumentation losslessly within loops when monitoring the first few iterations is sufficient.

We consider an extension of this idea in Fig. 7 where the program structure is used to obtain the loss model. Instrumentation from the loop is replaced with a single symbol k at the end of the loop. If instrumentation is disabled for all iterations of the loop, the monitor is in states $\{ q_0, q_1 \}$ after the event k. If the

```
while cond do
    if x <y then
        x.a();
    x.b();
    x.c();
```

(a) Loop targeted for instrumentation removal.



(b) Program property DFA. All missing edges go to error state (not shown). (c) Alternate monitor NFA. Error state not shown. (d) A filter to remove instrumentation from loop and replace it with a single symbol "k".

Fig. 7. Missing Events in Loops to be able to remove instrumentation in them.

first few iterations are monitored and event a is generated, the monitor will be in states $\{ q_1 \}$ after the event k.

The loss model can be calculated using a method from [4]. It presents a static analysis which finds the set of states that are possible after a program region, e.g., a loop body, for any given starting state if monitoring were to be disabled in that region. We can use this information directly instead of $R^{-1}(k)$ in Theorem 2 for computing $\delta^{\psi}(q,\ k)\ \forall q$. This is equivalent to mapping the set of strings which go from q to $\delta^{\psi}(q,\ k)$ to the new symbol for the loss model.

7 Related Work

Runtime monitoring has been an active research area over the past few decades. A significant part of the research in this area has focused on optimizing monitors and controlling the runtime overhead to make monitoring employable in practice.

A line of research [2,4,18,19] focuses on lossless partial evaluation of the finite state property to build residual monitors which process fewer events during runtime. [4] and [19] can be modelled in our framework using loss models where $R^{-1}(y)$ is a singleton set. Another line of research proposes purely dynamic optimizations where resources at run-time are constrained [8]. Purandare et al. [6] combine multiple monitors which share events into a single monitor to reduce the number of monitors updated. Schneider et al. execute monitors in parallel by enabling exchange of states between them to scale up monitoring [20].

Kauffman et al. [21] and Joshi et al. [22] consider monitorability of LTL formulas under losses. [21] considers natural losses such as loss, corruption, repetition, or out-of-order arrival of an event and gives an algorithm to find monitorability of a LTL formula. They do not construct a monitor to monitor the partial traces. [22] considers monitorability of LTL formulas in the presence of one loss type which is equivalent to our dropped-count loss (Fig. 3a) with $n = 1$. They only handle the formulas whose synthesized monitor has transitions that always lead to just one state, and it only recovers from losses after observing such a transition. In general, recurrence temporal properties [23] that can be modeled by Büchi automata are naturally immune to event losses due to loops in their structures. Our work primarily focuses on safety properties.

Falzon et al. [10] consider the construction of an alternate sound monitor when for some parts of the traces only aggregate information, such as the frequency of events but not their order, is available. We formalize this loss type in our framework in Sect. 6. Dwyer et al. [7] consider sub-properties formed when the alphabet is restricted to its subset to sample sub-properties from a given property. Their construction ensures completeness and is equivalent to our construction with $R = \{(x, y) \mid x \in \Delta^* y \ \forall y \in \Sigma \setminus \Delta\}$, where Δ is the set of symbols not observed as events. Figure 4b generalizes it with $R = \{\,(x, y) \mid x \in \Delta^* y \ \forall y \in \Sigma\,\}$.

Basin et al. [9] introduce a 3-valued timed logic to account for missing information in recorded traces for offline analysis. This allows them to report 3 results: if a violation occurred, if it did not occur, or if the knowledge is insufficient to report either. In the problem we consider, instead of having a single representation for missing information we can have multiple representations for different losses which can differ in their power to report an error.

Bartocci et al. [24] introduce statistical methods to inform overhead control and minimize the probability of missing a violation. For the monitors which are disabled, [25] introduces statistical methods to predict the missing information due to sampling, which is then used in [24] to get a probability that the violation occurred in an incomplete run. Instead of disabling monitoring altogether and predicting missing information, our approach records partial information about the events to report violations while maintaining completeness.

8 Conclusion and Future Work

We presented a framework for online finite-state monitoring of traces that carry partial information, where the losses could be natural or artificially induced. Our framework provides a general model that accommodates various losses present in the current research literature. We provide an efficient and automatic method to construct optimal monitors from a property specification and a loss model. Our optimality results provide hard limits to determine which property-loss and model combinations are *feasible*. We hope this makes it easier in future to study specific loss types. We also hope that this novel approach makes monitoring particularly attractive in the presence of high-frequency events and lossy channels, serving as a theoretical basis for implementations dealing with such constraints and for new optimizations inducing event losses. In the future, we would like to extend our framework to address infinite-state monitors and work on open questions about the characteristics of more restricted classes of loss models and their properties.

References

1. Kim, M., Viswanathan, M., Ben-Abdallah, H., Kannan, S., Lee, I., Sokolsky, O.: Formally specified monitoring of temporal properties. ECRTS **9–11**, 114–122 (1999)
2. Bodden, E., Lam, P., Hendren, L.: Clara: a framework for partially evaluating finite-state runtime monitors ahead of time. In: RV, pp. 183–197 (2010)
3. Bodden, E.: Efficient hybrid typestate analysis by determining continuation-equivalent states. In: ICSE, pp. 5–14 (2010)
4. Dwyer, M.B., Purandare, R.: Residual dynamic typestate analysis exploiting static analysis: results to reformulate and reduce the cost of dynamic analysis. In: ASE, pp. 124–133 (2007)
5. Purandare, R., Dwyer, M.B., Elbaum, S.: Monitor optimization via stutter-equivalent loop transformation. In: OOPSLA, pp. 270–285 (2010)
6. Purandare, R., Dwyer, M.B., Elbaum, S.: Optimizing monitoring of finite state properties through monitor compaction. In: ISSTA, pp. 280–290 (2013)
7. Dwyer, M.B., Diep, M., Elbaum, S.: Reducing the cost of path property monitoring through sampling. In: ASE, pp. 228–237 (2008)
8. Allabadi, G., Dhar, A., Bashir, A., Purandare, R.: METIS: resource and context-aware monitoring of finite state properties. In: Colombo, C., Leucker, M. (eds.) RV 2018. LNCS, vol. 11237, pp. 167–186. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03769-7_10
9. Basin, D., Klaedtke, F., Marinovic, S., Zălinescu, E.: Monitoring compliance policies over incomplete and disagreeing logs. In: RV, pp. 151–167 (2013)
10. Falzon, K., Bodden, E., Purandare, R.: Distributed finite-state runtime monitoring with aggregated events. In: RV, pp. 94–111 (2013)
11. Cook, S.A.: Soundness and completeness of an axiom system for program verification. SIAM J. Comput. **7**, 70–90 (1978)
12. Bartocci, E., Falcone, Y. (eds.): Lectures on Runtime Verification. LNCS, vol. 10457, Springer, Cham (2018). https://doi.org/10.1007/978-3-319-75632-5
13. Sipser, M.: Introduction to the Theory of Computation. Third ed. (2013)
14. Han, Y.-S., Wood, D.: The generalization of generalized automata: expression automata. In: CIAA, pp. 156–166 (2005)
15. Inc, R.V.: runtimeverification/property-db. https://github.com/runtimeverification/property-db (2013)
16. Gange, G., Ganty, P., Stuckey, P.J.: Fixing the state budget: approximation of regular languages with small DFAs. In: ATVA, pp. 67–83 (2017)
17. Kushwaha, P.: Accompanying sample implementation. https://gist.github.com/peey/1c2be77b05d00aa795e08d54ac2c1f9d (2022)
18. Bodden, E., Lam, P., Hendren, L.: Finding programming errors earlier by evaluating runtime monitors ahead-of-time. In: FSE, pp. 36–47 (2008)
19. Dwyer, M.B., Kinneer, A., Elbaum, S.: Adaptive online program analysis. In: ICSE, pp. 220–229 (2007)
20. Schneider, J., Basin, D., Brix, F., Krstić, S., Traytel, D.: Adaptive online first-order monitoring. In: ATVA, pp. 133–150 (2019)
21. Kauffman, S., Havelund, K., Fischmeister, S.: Monitorability over unreliable channels. In: Finkbeiner, B., Mariani, L. (eds.) RV 2019. LNCS, vol. 11757, pp. 256–272. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32079-9_15
22. Joshi, Y., Tchamgoue, G.M., Fischmeister, S.: Runtime verification of LTL on lossy traces. In: SAC, pp. 1379–1386 (2017)

23. Manna, Z., Pnueli, A.: A hierarchy of temporal properties (invited paper, 1989). In: PODC, pp. 377–410 (1990)
24. Bartocci, E., et al.: Adaptive runtime verification. In: RV, pp. 168–182 (2013)
25. Stoller, S.D., et al.: Runtime verification with state estimation. In: RV, pp. 193–207 (2012)