# Monitorability Over Unreliable Channels[⋆]

Sean Kauffman[1], Klaus Havelund[2], and Sebastian Fischmeister[1]

[1] University of Waterloo, Canada
[2] Jet Propulsion Laboratory, California Inst. of Technology, USA

**Abstract.** In Runtime Verification (RV), monitoring a system means checking an execution trace of a program for satisfactions and violations of properties. The question of which properties can be effectively monitored over ideal channels has mostly been answered by prior work. However, program monitoring is often deployed for remote systems where communications may be unreliable. In this work, we address the question of what properties are monitorable over an unreliable communication channel. We describe the different types of mutations that may be introduced to an execution trace and examine their effects on program monitoring. We propose a fixed-parameter tractable algorithm for determining the immunity of a finite automaton to a trace mutation and show how it can be used to classify $\omega$-regular properties as monitorable over channels with that mutation.

## 1 Introduction

In Runtime Verification (RV) the correctness of a program execution is determined by another program, called a monitor. In many cases, monitors run remotely from the systems they monitor, either due to resource constraints or for dependability. For example, ground stations monitor a spacecraft, while an automotive computer may monitor emissions control equipment. In both cases, the program being monitored must transmit data to a remote monitor.

Communication between the program and monitor may not always be reliable, however, leading to incorrect or incomplete results. For example, data from the Mars Science Laboratory (MSL) rover is received out-of-order, and some low priority messages may arrive days after being sent. Even dedicated debugging channels like ARM Embedded Trace Macrocell (ETM) have finite bandwidth and may lose data during an event burst [1]. Some works in the field of RV have begun to address the challenges of imperfect communication, but the problem has been largely ignored in the study of monitorability.

In this work, we propose a definition for a property to be considered monitorable over an unreliable communication channel. To reach our definition, we must determine what constitutes a monitorable property and whether monitorability is affected by a mutation of the property's input. We first examine the

[⋆] The research performed by the second author was carried out at Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

concept of uncertainty in monitoring and four common notions of monitorability in Sections 3 and 4. We then define possible trace mutations due to unreliable channels and describe what makes a property immune to a trace mutation in Sections 5 and 6. The combination of immunity to a trace mutation and monitorability (under an existing definition) is what defines the monitorability of a property under that mutation. To reach a decision procedure for the immunity of an $\omega$-regular property, we map the definition of immunity to a property of derived monitor automata in Section 7. We finally present a decision procedure for the immunity of an automaton to a mutation and prove it correct in Section 8.

## 2 Notation

We use $\mathbb{N}$ to denote the set of all natural numbers and $\infty$ to denote infinity. We write $\bot$ to denote *false* and $\top$ to denote *true*. A finite sequence $\sigma$ of $n$ values, is written $\sigma = \langle v_1, \cdots, v_n \rangle$ where both $v_i$ and $\sigma(i)$ mean the $i$'th item in the sequence. A value $x$ is in a sequence $\sigma$, denoted by $x \in \sigma$, iff $\exists\ i \ \in\ \mathbb{N}$ such that $\sigma(i) = x$. The length of a sequence $\sigma$ is written $|\sigma| \in \mathbb{N} \cup \{\infty\}$. The suffix of a sequence $\sigma$ beginning at the $i$'th item in the sequence is written $\sigma^i$. The concatenation of two sequences $\sigma, \tau$ is written $\sigma \cdot \tau$ where $\sigma$ is finite and $\tau$ is either finite or infinite.

We denote the cross product of $A$ and $B$ as $A \times B$ and the set of total functions from $A$ to $B$ as $A \to B$. Given a set $S$, $S^*$ denotes the set of finite sequences over $S$ where each sequence element is in $S$, $S^\omega$ denotes the set of infinite sequences of such elements, and $S^\infty = S^* \cup S^\omega$. Given a set $S$, we write $2^S$ to mean the set of all subsets of $S$. The cardinality of a set $S$ is written $|S|$. A map is a partial function $M : K \nrightarrow V$ where $K$ is a finite domain of keys mapped to the set $V$ of values. We write $M(k) \leftarrow v$ to denote $M$ updated with $k$ mapped to $v$. AP is a finite, non-empty set of *atomic propositions*. An *alphabet* is denoted $\Sigma = 2^{\mathrm{AP}}$, and an element of the alphabet is a symbol $s \in \Sigma$. A *trace*, *word*, or *string* is a sequence of symbols.

In this work, we use *Finite Automata (sFAs)* to represent both regular and $\omega$-regular languages. We use Non-deterministic Büchi Automata (sNBAs) to represent $\omega$-regular languages, which accept infinite strings, and Non-deterministic Finite Automata (sNFAs) to represent regular languages, which accept finite strings. Both an NBA and an NFA are written $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$, where $Q$ is the set of states, $\Sigma$ is the alphabet, $q_0 \in Q$ is the initial state, $\delta : Q \times \Sigma \to 2^Q$ is the transition function, and $F \subseteq Q$ is the set of accepting states. The two types of FAs differ in their accepting conditions. An NFA is a Deterministic Finite Automaton (DFA) iff $\forall q \in Q,\ \forall \alpha \in \Sigma,\ |\delta(q, \alpha)| = 1$.

A *path* (or *run*) through an FA $\mathcal{A}$ from a state $q \in Q$ over a word $\sigma \in \Sigma^\infty$ is a sequence of states $\pi = \langle q_1, q_2, \cdots \rangle$ such that $q_1 = q$ and $q_{i+1} \in \delta(q_i, \sigma_i)$. We write $\mathcal{A}(q, \sigma)$ to denote the set of all runs on $\mathcal{A}$ starting at state $q$ with the word $\sigma$. The set of all *reachable states* in an FA $\mathcal{A}$ from a starting state $q_0$ is denoted $\mathcal{R}each(\mathcal{A}, q_0) = \{q \in \pi : \pi \in \mathcal{A}(q_0, \sigma),\ \sigma \in \Sigma^\infty\}$. Given a DFA $(Q, \Sigma, q_0, \delta, F)$, a state $q \in Q$, and a finite string $\sigma \in \Sigma^* : |\sigma| = n$, $\delta^* : Q \times \Sigma^* \to Q$ denotes the terminal ($n$th) state of the run over $\sigma$ beginning in $q$.

A finite run on an NFA $\pi = \langle q_1, q_2, \cdots, q_n \rangle$ is considered *accepting* if $q_n \in F$. For an infinite run on an NBA $\rho$, we use $\mathit{Inf}(\rho) \subseteq Q$ to denote the set of states that are visited infinitely often, and the run is considered *accepting* when $\mathit{Inf}(\rho) \cap F \neq \emptyset$. $\mathcal{L}(\mathcal{A})$ denotes the language accepted by an FA $\mathcal{A}$. The complement or negation of an FA $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ is written $\overline{\mathcal{A}}$ where $\mathcal{L}(\overline{\mathcal{A}}) = \Sigma^* \setminus \mathcal{L}(\mathcal{A})$ for NFAs and $\mathcal{L}(\overline{\mathcal{A}}) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$ for NBAs.

We use Linear Temporal Logic (LTL) throughout the paper to illustrate examples of properties because it is a common formalism in the RV area. The syntax of these formulae is defined by the following inductive grammar where $p$ is an atomic proposition, $U$ is the *Until* operator, and $X$ is the *Next* operator.

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid X\varphi \mid \varphi\, U \varphi$$

The symbols $\neg$ and $\vee$ are defined as expected and the following inductive semantics are used for $X$ and $U$, where $\sigma \in \Sigma^\omega$.

- $\sigma \models X\varphi$ *iff* $\sigma^2 \models \varphi$
- $\sigma \models \varphi\, U \phi$ *iff* $\exists k \geq 1 : \sigma^k \models \phi \wedge \forall j : 1 \leq j < k,\ \sigma^j \models \varphi$

We also define the standard notation: $\mathit{true} = p \vee \neg p$ for any proposition $p$, $\mathit{false} = \neg \mathit{true}$, $\varphi \wedge \phi = \neg(\neg\varphi \vee \neg\phi)$, $\varphi \to \phi = \neg\varphi \vee \phi$, $F\varphi = \mathit{true}\, U \varphi$ (eventually $\varphi$), $G\varphi = \neg F \neg \varphi$ (globally $\varphi$), and $\overline{X}\varphi = X\varphi \vee \neg X \mathit{true}$ (weak-next $\varphi$, true at the end of a finite trace).

## 3 Uncertainty

Program properties are typically specified as languages of infinite length strings, for example by writing LTL formulae. However, in RV, a finite prefix of an execution trace must be checked. We say a finite string *determines* inclusion in (or exclusion from) a language of infinite words only if all infinite extensions of the prefix are in (or out of) the language. If some infinite extensions are in the language and some are out, then the finite prefix does not determine inclusion and the result is uncertainty. The problem appears with an LTL property such as $Fa$, which is satisfied if an $a$ appears in the string. However, if no $a$ has yet been observed, and the program is still executing, it is unknown if the specification will be satisfied in the future.

To express notions of uncertainty in monitoring, extensions to the Boolean truth domain $\mathbb{B}_2 = \{\top, \bot\}$ have been proposed. $\mathbb{B}_3$ adds a third verdict of ***?*** to the traditional Boolean notion of *true* or *false* to represent the idea that the specification is neither satisfied nor violated by the current finite prefix [5]. $\mathbb{B}_4$ replaces ***?*** with *presumably true* ($\top_p$) and *presumably false* ($\bot_p$) to provide more information on what has already been seen [6].

The verdicts $\top_p$ and $\bot_p$ differentiate between prefixes that would satisfy or violate the property interpreted with finite trace semantics. The intuition is that $\bot_p$ indicates that something is required to happen in the future, while $\top_p$ means there is no such outstanding event. For example, if the formula $a \rightarrow Fb$ is interpreted as four-value LTL ($\text{LTL}_4$) (also called Runtime Verification LTL (RV-LTL) [6], which uses $\mathbb{B}_4$), the verdict on a trace $\langle b \rangle$ is $\top_p$ because $a$ has not occurred, and therefore no $b$ is required, while the verdict on $\langle a \rangle$ is $\bot_p$ because there is an $a$ but as yet no $b$. If the same property is interpreted as three-value LTL ($\text{LTL}_3$) (which uses $\mathbb{B}_3$) the verdicts on both traces would be *?*.

The above intuitions are formalized in Definition 1, which is based on notation from [13]. Here, $\varphi$ is a language that includes both finite and infinite traces.

**Definition 1.** (Evaluation Functions) *Given a property $\varphi \subseteq \Sigma^\infty$ (here understood as the language it accepts) for each of the truth domains $\mathbb{B} \in \{\mathbb{B}_3, \mathbb{B}_4\}$, we define evaluation functions of the form $[\![\cdot]\!]_{\mathbb{B}}(\cdot) : 2^{\Sigma^\infty} \times \Sigma^* \rightarrow \mathbb{B}$ as the following. For $\mathbb{B}_3 = \{\bot, ?, \top\}$,*

$$[\![\varphi]\!]_{\mathbb{B}_3}(\sigma) = \begin{cases} \bot & \text{if } \sigma \cdot \mu \notin \varphi \ \forall \mu \in \Sigma^\omega \\ \top & \text{if } \sigma \cdot \mu \in \varphi \ \forall \mu \in \Sigma^\omega \\ ? & \text{otherwise} \end{cases}$$

*For $\mathbb{B}_4 = \{\bot, \bot_p, \top_p, \top\}$,*

$$[\![\varphi]\!]_{\mathbb{B}_4}(\sigma) = \begin{cases} [\![\varphi]\!]_{\mathbb{B}_3}(\sigma) & \text{if } [\![\varphi]\!]_{\mathbb{B}_3}(\sigma) \neq \ ? \\ \bot_p & \text{if } [\![\varphi]\!]_{\mathbb{B}_3}(\sigma) = \ ? \text{ and } \sigma \notin \varphi \\ \top_p & \text{if } [\![\varphi]\!]_{\mathbb{B}_3}(\sigma) = \ ? \text{ and } \sigma \in \varphi \end{cases}$$

Introducing the idea of uncertainty in monitoring causes the possibility that some properties might never reach a definite, *true* or *false* verdict. A monitor that will only ever return a *?* result does not have much utility. The *monitorability* of a property captures on this notion of the reachability of definite verdicts.

## 4 Monitorability

In this section, we examine the four most common definitions of monitorability. To define monitorability for properties over unreliable channels, we must first define monitorability for properties over ideal channels. Rather than choose one definition, we introduce four established definitions and allow the reader to select that of their preference.

### 4.1 Classical $\sigma$-Monitorability

Pnueli and Zaks introduced the first formal definition of monitorability in their work on Property Specification Language (PSL) model checking in 2006 [24]. They define what languages are monitorable given a trace prefix $\sigma$.

**Definition 2.** (Classical $\sigma$-Monitorability) *Given an alphabet $\Sigma$, and a finite sequence $\sigma \in \Sigma^*$ a language $\varphi \subseteq \Sigma^\infty$ is $\sigma$-monitorable iff $\exists \eta \in \Sigma^* : \sigma \cdot \eta \cdot s \models \varphi \ \forall s \in \Sigma^\infty \ \lor \ \sigma \cdot \eta \cdot s \not\models \varphi \ \forall s \in \Sigma^\infty$.*

That is, there exists another finite sequence $\eta$ such that $\sigma \cdot \eta$ determines inclusion in or exclusion from $\varphi$.

For example, $GFp$ is non-monitorable for any finite prefix, because the trace needed to determine the verdict must be infinite. If a reactive system is expected to run forever, then it is useless to continue monitoring after observing $\sigma$ such that $\varphi$ is not monitorable.

## 4.2 Classical Monitorability

Bauer, Leuker, and Schallhart restated this definition of monitorability and proved that safety and guarantee (co-safety) properties represent a proper subset of the class of monitorable properties [7]. It was already known that the class of monitorable properties was not limited to safety and guarantee properties from the work of d'Amorim and Roşu on monitoring $\omega$-regular languages [10], however that work did not formally define monitorability.

The definition of monitorability given by Bauer et al. is identical to Definition 2 except that it considers all possible trace prefixes instead of a specific prefix [12, 13] and it excludes languages with finite words. The restriction to infinite words is due to their interest in defining monitorable LTL$_3$ properties, which only considers infinite traces.

They use Kupferman and Vardi's definitions of *good* and *bad* prefixes of an infinite trace [17] to define what they call an *ugly* prefix. That is, given an alphabet $\Sigma$ and a language of infinite strings $\varphi \subseteq \Sigma^\omega$,

- a finite word $b \in \Sigma^*$ is a *bad prefix* for $\varphi$ iff $\forall s \in \Sigma^\omega$, $b \cdot s \notin \varphi$, and
- a finite word $g \in \Sigma^*$ is a *good prefix* for $\varphi$ iff $\forall s \in \Sigma^\omega$, $g \cdot s \in \varphi$.

Bauer et al. use good and bad prefixes to define *ugly* prefixes and then use ugly prefixes to define Classical Monitorability.

**Definition 3.** (Ugly Prefix) *Given an alphabet $\Sigma$ and a language of infinite strings $\varphi \subseteq \Sigma^\omega$, a finite word $u \in \Sigma^*$ is an* ugly prefix *for $\varphi$ iff $\nexists s \in \Sigma^* : u \cdot s$ is either a good or bad prefix.*

**Definition 4.** (Classical Monitorability) *Given a language of infinite strings $\varphi \subseteq \Sigma^\omega$, $\varphi$ is* classically monitorable *iff $\nexists u \in \Sigma^* : u$ is an ugly prefix for $\varphi$.*

## 4.3 Weak Monitorability

Recently, both Chen et al. and Peled and Havelund proposed a weaker definition of monitorability that includes more properties than either the Classical or Alternative definitions [9, 22]. They observed that there are properties that are classically non-monitorable, but that are still useful to monitor. For example, the property $a \wedge GFa$ is non-monitorable under Definition 4 because any trace that begins with $a$ must then satisfy or violate $GFa$, which is not possible. However, $a \wedge GFa$ is violated by traces that do not begin with $a$, so it may have some utility to monitor.

**Definition 5.** (Weak Monitorability) *Given a property $\varphi \subseteq \Sigma^\infty$, $\varphi$ is* weakly monitorable *iff $\exists p \in \Sigma^* : p$ is not an ugly prefix for $\varphi$.*

### 4.4 Alternative Monitorability

Falcone et al. observed that the class of monitorable properties should depend on the truth domain of the monitored formula. However, they noticed that changing from $\mathbb{B}_3$ to $\mathbb{B}_4$ does not influence the set of monitorable properties under classical monitorability [12, 13]. To resolve this perceived shortcoming, the authors of [12, 13] introduce an *alternative* definition of monitorability. They introduce the notion of an *r-property* (runtime property) which separates the property's language of finite and infinite traces into disjoint sets. We do not require this distinction and treat the language of a property as a single set containing both finite and infinite traces. Falcone et al. then define an alternative notion of monitorability for a property using a variant of Definition 1.

**Definition 6.** (Alternative Monitorability) *Given a truth domain* $\mathbb{B}$ *and an evaluation function for* $\mathbb{B}$ $[\![\cdot]\!]_{\mathbb{B}}(\cdot) : 2^{\Sigma^\infty} \times \Sigma^* \to \mathbb{B}$*, a property* $\varphi \subseteq \Sigma^\infty$ *is alternatively monitorable iff* $\forall \sigma_g \in \varphi \cap \Sigma^*,\ \forall \sigma_b \notin \varphi \cap \Sigma^* \quad [\![\varphi]\!]_{\mathbb{B}}(\sigma_g) \neq [\![\varphi]\!]_{\mathbb{B}}(\sigma_b)$

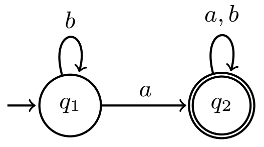
Definition 6 says that, given a truth domain, a language with both finite and infinite words is monitorable if evaluating the finite strings in the language always yield different verdicts from evaluating the finite strings out of the language.

## 5 Unreliable Channels

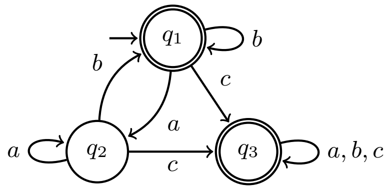
For a property to be monitorable over an unreliable channel it must be monitorable over ideal channels, and it must reach the correct verdict despite the unreliable channel. To illustrate this idea, we introduce an example.

### 5.1 An Example with Unreliable Channels

Consider the LTL property $Fa$ over the alphabet $\Sigma = \{a, b\}$. That is, all traces that contain at least one $a$ satisfy $\varphi$. We assume that the trace is monitored remotely, and, for this example, we will adopt a $\mathbb{B}_3$ truth domain. With LTL$_3$ semantics, the verdict on finite prefixes without an $a$, is *?*, while the verdict when an $a$ is included is $\top$. Figure 1a shows the NBA for such a property.



(a) NBA for $Fa$



(b) NBA for $(G(a \to Fb)) \vee Fc$

**Monitorability under reordering** Suppose that the channel over which the trace is transmitted may reorder events. That is, events are guaranteed to be delivered, but not necessarily in the same order in which they were sent.

We argue that $Fa$ should be considered monitorable over a channel that reorders the trace. First, the property is monitorable over an ideal channel (see Section 4). Second, given any trace prefix, reordering the prefix would not change the verdict of a monitor. Any $a$ in the trace will cause a transition to state $q_2$, regardless of its position.

**Monitorability under loss** Now suppose that, instead of reordering, the channel over which the trace is transmitted may lose events. That is, the order of events is guaranteed to be maintained, but some events may be missing from the trace observed by the monitor.

We argue that $Fa$ should not be considered monitorable over a channel that loses events, even though the property is deemed to be monitorable over an ideal channel. It is possible for the verdict from the monitor to be different from what it would be given the original trace. For example, assume a trace $\langle a, b\rangle$. For this trace, the verdict from an LTL$_3$ monitor would be $\top$. However, if the $a$ is lost, the verdict would be **?**.

### 5.2 Trace Mutations

To model unreliable channels, we introduce *trace mutations*. A mutation represents the possible modifications to traces from communication over unreliable channels. These mutations are defined as relations between unmodified original traces and their mutated counterparts. Trace mutations include only finite traces because only finite prefixes may be mutated in practice.

There are four trace mutations $\mathcal{M}^k \subseteq \Sigma^* \times \Sigma^*$ where $\mathcal{M}$ denotes any of the relations in Definitions 7, 8, 9, and 10 or a union of any number of them, and $k$ denotes the number of inductive steps.

**Definition 7.** (Loss Mutation)
$$\mathit{Loss} = \{(\sigma, \sigma') : \sigma = \sigma' \vee \exists \alpha, \beta \in \Sigma^*, \exists x \in \Sigma : \sigma = \alpha \cdot \langle x\rangle \cdot \beta \wedge \sigma' = \alpha \cdot \beta\}$$

**Definition 8.** (Corruption Mutation)
$$\mathit{Corruption} = \{(\sigma, \sigma') : \exists \alpha, \beta \in \Sigma^*, \exists x, y \in \Sigma : \sigma = \alpha \cdot \langle x\rangle \cdot \beta \wedge \sigma' = \alpha \cdot \langle y\rangle \cdot \beta\}$$

**Definition 9.** (Stutter Mutation)
$$\mathit{Stutter} = \{(\sigma, \sigma') : \sigma = \sigma' \vee \exists \alpha, \beta \in \Sigma^*, \exists x \in \Sigma : \sigma = \alpha \cdot \langle x\rangle \cdot \beta \wedge \sigma' = \alpha \cdot \langle x, x\rangle \cdot \beta\}$$

**Definition 10.** (Out-of-Order Mutation)
$$\mathit{OutOfOrder} = \{(\sigma, \sigma') : \exists \alpha, \beta \in \Sigma^*, \exists x, y \in \Sigma : \sigma = \alpha \cdot \langle x, y\rangle \cdot \beta \wedge \sigma' = \alpha \cdot \langle y, x\rangle \cdot \beta\}$$

**Definition 11.** (Inductive k-Mutations) *Given any mutation or union of mutations $\mathcal{M}^k$, we define $\mathcal{M}^{k+1}$ inductively as the following.*
$$\mathcal{M}^1 \in \{\bigcup m : m \in 2^{\{\mathit{Loss}, \mathit{Corruption}, \mathit{Stutter}, \mathit{OutOfOrder}\}}, m \neq \varnothing\}$$
$$\mathcal{M}^{k+1} = \{(\sigma_1, \sigma_3) : \exists (\sigma_1, \sigma_2) \in \mathcal{M}^k, \exists (\sigma_2, \sigma_3) \in \mathcal{M}^1\} \cup \mathcal{M}^k$$

These mutations are based on Lozes and Villard's interference model [21]. Other works on the verification of unreliable channels, such as [8], have chosen to include *insertion* errors instead of *Corruption* and *OutOfOrder*. We prefer to define *Corruption* and *OutOfOrder* because the mutations more closely reflect our real-world experiences. For example, packets sent using the User Datagram Protocol (UDP) may be corrupted or arrive out-of-order, but packets must be sent before these mutations occur.

We say a mutation $M$ is *prefix closed* when $\forall(\sigma, \sigma') \in M$ such that $|\sigma| > 1$, $\exists(\sigma_p, \sigma'_p) \in M$, where $\sigma_p$ is a prefix of $\sigma$ and $\sigma'_p$ is a prefix of $\sigma'$. All mutations $\mathcal{M}^1$ are prefix closed. Combining mutations is possible under Definition 11, and it is possible to form any combination of strings by doing so. This capability is important to ensure the mutation model is complete.

**Theorem 1.** (Completeness of Mutations) *Given any set of non-empty traces* $S \subseteq \Sigma^* \setminus \{\varepsilon\}$*,* $(\mathit{Loss} \cup \mathit{Corruption} \cup \mathit{Stutter})^\infty = S \times \Sigma^*$*.*

*Proof:* First, Definition 8 allows an arbitrary symbol in a string to be changed to any other symbol. Thus, $\forall\sigma' \in \Sigma^*,\ \exists\sigma : (\sigma, \sigma') \in \mathit{Corruption}^n,\ |\sigma| = |\sigma'|$ where $n \geq |\sigma|$. A string can also be lengthened or shortened arbitrarily, so long as it is non-empty. Definition 9 allows lengthening, because $\mathit{Stutter}(\sigma, \sigma') \implies |\sigma| < |\sigma'|$, while Definition 7 allows shortening, because $\mathit{Loss}(\sigma, \sigma') \implies |\sigma| > |\sigma'|$. □

These mutations are general and it may be useful for practitioners to define their own, more constrained mutations based on domain knowledge. For example, even Definition 10 is unnecessary for the completeness of the mutation model, but the combination of Definitions 7, 8, and 9 cannot completely specify the *OutOfOrder* relation. That is, $\mathit{OutOfOrder}^n \subset \mathit{Corruption}^{2n}\ \forall n \in \mathbb{N}$.

## 6 Immunity to Trace Mutations

The two requirements for a property to be monitorable over an unreliable channel are that the property is monitorable over an ideal channel and that the property is *immune* to the effects of the unreliable channel. A monitor must be able to reach a meaningful, actionable verdict for a trace prefix, and the verdict must also be *correct*. If a monitored property is immune to a mutation then we can trust the monitor's verdict whether or not the observed trace is mutated.

Definition 12 characterizes properties where the given trace mutation will have no effect on the evaluation verdict. For example, the LTL property $Fa$ from Figure 1a is immune to $\mathit{OutOfOrder}^\infty$ with truth domain $\mathbb{B}_3$ or $\mathbb{B}_4$ because reordering the input trace cannot change the verdict.

**Definition 12.** (Full Immunity to Unreliable Channels) *Given a trace alphabet* $\Sigma$*, a property* $\varphi \subseteq \Sigma^\infty$*, a trace mutation* $\mathcal{M}^k \subseteq \Sigma^* \times \Sigma^*$*, a truth domain* $\mathbb{B}$*, and an evaluation function* $[\![\cdot]\!]_{\mathbb{B}}(\cdot) : 2^{\Sigma^\infty} \times \Sigma^* \to \mathbb{B}$*,* $\varphi$ *is immune to* $\mathcal{M}^k$ *iff* $\forall(\sigma, \sigma') \in \mathcal{M}^k,\ [\![\varphi]\!]_{\mathbb{B}}(\sigma) = [\![\varphi]\!]_{\mathbb{B}}(\sigma')$*.*

Definition 12 specifies a k-Mutation from Definition 11, but a property that is immune to a mutation for some $k$ is immune to that mutation for *any* $k$. This significant result forms the basis for checking for mutation immunity in Section 8. The intuition is that, since we assume any combination of symbols in the alphabet is a possible ideal trace, and a mutation could occur at any time, one mutation is enough to violate immunity for any vulnerable property.

**Theorem 2.** (Single Mutation Immunity Equivalence) *Given a trace alphabet $\Sigma$, a property $\varphi \subseteq \Sigma^\infty$, a trace mutation $\mathcal{M} \subseteq \Sigma^* \times \Sigma^*$, and a number of applications of that mutation $k$, $\varphi$ is immune to $\mathcal{M}^k$ iff $\varphi$ is immune to $\mathcal{M}^1$.*

*Proof:* Since k-Mutations are defined inductively, Theorem 2 is equivalent to the statement that $\varphi$ is immune to $\mathcal{M}^{k+1}$ iff $\varphi$ is immune to $\mathcal{M}^k$. Now assume by way of contradiction a property $\varphi_{\text{bad}} \subseteq \Sigma^\infty$ such that $\varphi_{\text{bad}}$ is immune to some k-Mutation $M^k$ but not to $M^{k+1}$. That is, given a truth domain $\mathbb{B}$, $\exists(\sigma_1, \sigma_3) \in M^{k+1} : [\![\varphi_{\text{bad}}]\!]_{\mathbb{B}}(\sigma_1) \neq [\![\varphi_{\text{bad}}]\!]_{\mathbb{B}}(\sigma_3)$. From Definition 11, either $(\sigma_1, \sigma_3) \in M^k$, or $\exists(\sigma_1, \sigma_2) \in \mathcal{M}^k, \exists(\sigma_2, \sigma_3) \in \mathcal{M}^1 : [\![\varphi_{\text{bad}}]\!]_{\mathbb{B}}(\sigma_1) \neq [\![\varphi_{\text{bad}}]\!]_{\mathbb{B}}(\sigma_3)$. It cannot be true that $(\sigma_1, \sigma_3) \in M^k$ since $\varphi_{\text{bad}}$ is immune to $M^k$ so there must exist pairs $(\sigma_1, \sigma_2) \in \mathcal{M}^k$ and $(\sigma_2, \sigma_3) \in \mathcal{M}^1$. Since $\varphi_{\text{bad}}$ is immune to $\mathcal{M}^k$, $[\![\varphi_{\text{bad}}]\!]_{\mathbb{B}}(\sigma_1) = [\![\varphi_{\text{bad}}]\!]_{\mathbb{B}}(\sigma_2)$ so it must be true that $[\![\varphi_{\text{bad}}]\!]_{\mathbb{B}}(\sigma_2) \neq [\![\varphi_{\text{bad}}]\!]_{\mathbb{B}}(\sigma_3)$. However, it is clear from Definition 11 that $M^k \subseteq M^{k+1}$, so $M^1 \subseteq M^k$ for any $k$, which is a contradiction.

For the reverse case, assume a property $\varphi_{\text{sad}} \subseteq \Sigma^\infty$ such that $\varphi_{\text{sad}}$ is not immune to some k-Mutation $M^k$ but immune to $M^{k+1}$. However, as we saw before, $M^k \subseteq M^{k+1}$ so $\varphi_{\text{sad}}$ must not be immune to $M^{k+1}$, a contradiction. □

Immunity under Definition 12 is too strong to be a requirement for monitorability over an unreliable channel, however. Take, for example, the property $(G(a \to Fb)) \vee Fc$, as shown in Figure 1b. By Definition 12 with truth domain $\mathbb{B}_4$ this property is vulnerable (not immune) to *OutOfOrder*$^1$ because reordering symbols may change the verdict from $\top_p$ to $\bot_p$ and vice versa. However, this property is monitorable under all definitions in Section 4, so we would like to weaken the definition of immunity only to consider the parts of a property that affect its monitorability.

To weaken the definition of immunity we consider only the determinization of the property to be crucial. Definition 13 characterizes properties for which satisfaction and violation are unaffected by a mutation. We call this *true-false immunity*, and it is equivalent to immunity with truth domain $\mathbb{B}_3$. The intuition is that $\mathbb{B}_3$ treats all verdicts outside $\{\top, \bot\}$ as the symbol *?* so immunity with this truth domain does not concern non-*true-false* verdicts.

**Definition 13.** (True-False Immunity to Unreliable Channels) *Given a trace alphabet $\Sigma$, a property $\varphi \subseteq \Sigma^\infty$, a trace mutation $\mathcal{M}^k \subseteq \Sigma^* \times \Sigma^*$, and the evaluation function $[\![\cdot]\!]_{\mathbb{B}_3}(\cdot) : 2^{\Sigma^\infty} \times \Sigma^* \to \mathbb{B}_3$, $\varphi$ is true-false immune to $\mathcal{M}^k$ iff $\forall(\sigma, \sigma') \in \mathcal{M}^k,\ [\![\varphi]\!]_{\mathbb{B}_3}(\sigma) = [\![\varphi]\!]_{\mathbb{B}_3}(\sigma')$.*

The true-false immunity of a property to a mutation is necessary but not sufficient to show that the property is monitorable over an unreliable channel.

For example, $G(a \rightarrow Fb)$ is true-false immune to all mutations because the verdict will be *?* for any prefix, but the property is not monitorable. We can now define monitorability over unreliable channels in the general case.

**Definition 14.** (Monitorability over Unreliable Channels) *Given a trace alphabet $\Sigma$, a property $\varphi \subseteq \Sigma^\infty$, a trace mutation $\mathcal{M}^k \subseteq \Sigma^* \times \Sigma^*$, and a definition of monitorability $\mathcal{V}$, $\varphi$ is monitorable over $\mathcal{M}^k$ iff $\varphi$ is considered monitorable by $\mathcal{V}$, and $\varphi$ is true-false immune to $\mathcal{M}^k$.*

By Rice's Theorem, monitorability over unreliable channels is undecidable in the general case, but we now provide a decision procedure for properties expressible by an NBA. As decision procedures for the monitorability of $\omega$-regular languages exist, we focus on determining the true-false immunity of a property to a given mutation.

## 7 Deciding Immunity for $\omega$-Regular Properties

To determine the immunity of an $\omega$-regular property to a trace mutation, we must construct automata that capture the notion of uncertainty from $\mathbb{B}_3$. Bauer et al. defined a simple process to build a $\mathbb{B}_3$ monitor using two DFAs in their work on LTL$_3$ [5].

The procedure begins by complementing the language. A language of infinite words $\varphi$ is represented as an NBA $\mathcal{A}_\varphi = (Q, \Sigma, q_0, \delta, F_\varphi)$, for example, LTL can be converted to an NBA by tableau construction [25]. The NBA is then complemented to form $\overline{\mathcal{A}_\varphi} = (\overline{Q}, \Sigma, \overline{q_0}, \overline{\delta}, \overline{F_\varphi})$. *Remark:* The upper bound for NBA complementation is $2^{O(n \log n)}$, so it is cheaper to complement an LTL property and construct its NBA if starting from temporal logic [18].

To form the monitor, create two NFAs based on the NBAs and then convert them to DFAs. The two NFAs are defined as $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ and $\overline{\mathcal{A}} = (\overline{Q}, \Sigma, \overline{q_0}, \overline{\delta}, \overline{F})$ The new accepting states are the states from which an NBA accepting state is reachable. That is, we make $F = \{q \in Q : \mathcal{R}each(\mathcal{A}_\varphi, q) \cap F_\varphi \neq \varnothing\}$, and $\overline{F} = \{q \in \overline{Q} : \mathcal{R}each(\overline{\mathcal{A}_\varphi}, q) \cap \overline{F_\varphi} \neq \varnothing\}$. The two NFAs are then converted to DFAs via powerset construction. The verdict for a finite trace $\sigma$ is then given as the following:

**Definition 15.** ($\mathbb{B}_3$ Monitor Verdict) *Given an alphabet $\Sigma$ and a language $\varphi \subseteq \Sigma^\omega$ , derive $\mathbb{B}_3$ monitor DFAs $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ and $\overline{\mathcal{A}} = (\overline{Q}, \Sigma, \overline{q_0}, \overline{\delta}, \overline{F})$. The $\mathbb{B}_3$ verdict for a string $\sigma \in \Sigma^*$ is the following.*

$$[\![\varphi]\!]_{\mathbb{B}_3}(\sigma) = \begin{cases} \bot & \text{if } \sigma \notin \mathcal{L}(\mathcal{A}) \\ \top & \text{if } \sigma \notin \mathcal{L}(\overline{\mathcal{A}}) \\ ? & \text{otherwise} \end{cases}$$

We can now restate Definition 13 using monitor automata. This new definition will allow us to construct a decision procedure for a property's immunity to a mutation.

**Theorem 3.** (True-False Immunity to Unreliable Channels for $\omega$-Regular Properties) *Given an alphabet $\Sigma$ and an $\omega$-regular language $\varphi \subseteq \Sigma^\omega$, derive $\mathbb{B}_3$ monitor DFAs $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ and $\overline{\mathcal{A}} = (\overline{Q}, \Sigma, \overline{q_0}, \overline{\delta}, \overline{F})$. $\varphi$ is true-false immune to a trace mutation $\mathcal{M}^k \subseteq \Sigma^* \times \Sigma^*$ iff $\forall (\sigma, \sigma') \in \mathcal{M}^k, (\sigma \notin \mathcal{L}(\mathcal{A}) \Leftrightarrow \sigma' \notin \mathcal{L}(\mathcal{A})) \wedge (\sigma \notin \mathcal{L}(\overline{\mathcal{A}}) \Leftrightarrow \sigma' \notin \mathcal{L}(\overline{\mathcal{A}}))$.*

*Proof:* By Definition 13 it is only necessary to show that $[\![\varphi]\!]_{\mathbb{B}_3}(\sigma) = [\![\varphi]\!]_{\mathbb{B}_3}(\sigma')$ is equivalent to $(\sigma \notin \mathcal{L}(\mathcal{A}) \Leftrightarrow \sigma' \notin \mathcal{L}(\mathcal{A})) \wedge (\sigma \notin \mathcal{L}(\overline{\mathcal{A}}) \Leftrightarrow \sigma' \notin \mathcal{L}(\overline{\mathcal{A}}))$. There are three cases: $\bot$, $\top$, and *?*. For $\bot$ and $\top$ it is obvious from Definition 15 that the verdicts are derived from exclusion from the languages of $\mathcal{A}$ and $\overline{\mathcal{A}}$. As there are only three possible verdicts, this also shows the *?* case. □

We say that an automaton is immune to a trace mutation in a similar way to how a property is immune. To show that a property is true-false immune to a mutation, we only need to show that its $\mathbb{B}_3$ monitor automata are also immune to the property. Note that, since the implication is both directions, we can use either language inclusion or exclusion in the definition.

**Definition 16.** (Finite Automaton Immunity) *Given a finite automaton $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ and a trace mutation $\mathcal{M}^k \subseteq \Sigma^* \times \Sigma^*$, $\mathcal{A}$ is immune to $\mathcal{M}^k$ iff $\forall (\sigma, \sigma') \in \mathcal{M}^k,\ \sigma \in \mathcal{L}(\mathcal{A}) \Leftrightarrow \sigma' \in \mathcal{L}(\mathcal{A})$.*

With this definition we can provide a decision procedure for the monitorability of an $\omega$-regular property over an unreliable channel. The procedure will check the immunity of the $\mathbb{B}_3$ monitor automata to the mutations from the channel, as well as the property's monitorability. If the DFAs are both immune to the mutations and the property is monitorable, then the property is monitorable over the unreliable channel.

## 8 Decision Procedure for Finite Automaton Immunity

We propose Algorithm 1 for deciding whether a DFA is immune to a trace mutation. The algorithm is loosely based on Hopcroft and Karp's near-linear algorithm for determining the equivalence of finite automata [15].

The parameters to Algorithm 1 are the DFA to check ($\mathcal{A}$) and the mutation ($M$) which is a relation given by $\mathcal{M}^1$ in Definition 11. The intuition behind Algorithm 1 is to follow transitions for pairs of unmutated and corresponding mutated strings in $M$ and verify that they lead to the same acceptance verdicts. More specifically, Algorithm 1 finds sets of states which must be equivalent for the DFA to be immune to a given mutation. The final verdict of IMMUNE is found by checking that no equivalence class contains both final and non-final states. If an equivalence class contains both, then there are some strings for which the verdict will change due to the given mutation.

If all mutations required only a string of length one, the step at Line 7 could follow transitions for pairs of single symbols. However, mutations like *OutOfOrder* require strings of at least two symbols, so we must follow transitions for short strings. We express this idea of a minimum length for a

**Algorithm 1** Determine if a DFA is immune to a given trace mutation.

```
1: procedure IMMUNE( $\mathcal{A} = (\Sigma, Q, q_0, \delta, F), M$ )
2:     for $q \in Q$ do $E(q) \leftarrow \{q\}$                         ▷ $E$ is a map $E : Q \nrightarrow 2^Q$
3:     $R \leftarrow \mathcal{R}each(\mathcal{A}, q_0)$                   ▷ $R$ is the reachable states
4:     $T \leftarrow \{\ \}$                         ▷ $T$ is a set of pairs, used like a worklist
5:     for $(\sigma, \sigma') \in M$ where $|\sigma| = minLength(M)$ do    ▷ $M$ is a mutation relation
6:         for $q \in R$ do
7:             $q_1 \leftarrow \delta^*(q, \sigma); q_2 \leftarrow \delta^*(q, \sigma')$         ▷ Follow mutated strings
8:             $E(q_1) \leftarrow E(q_2) \leftarrow \{q_1, q_2\}$           ▷ Update $E$ for both states
9:             $T \leftarrow T \cup \{(q_1, q_2)\}$                          ▷ Add the pair to $T$
10:    while $T$ is not empty do
11:        let $(q_1, q_2) \in T$                            ▷ Get a pair from the worklist
12:        $T \leftarrow T \setminus \{(q_1, q_2)\}$                      ▷ Remove the pair from $T$
13:        for $\alpha \in \Sigma$ do
14:            $n_1 \leftarrow \delta(q_1, \alpha); n_2 \leftarrow \delta(q_2, \alpha)$     ▷ Follow transitions to the next states
15:            $C \leftarrow \{E(n_1), E(n_2)\}$                         ▷ $C$ is a set of two sets
16:            if $|C| > 1$ then                               ▷ If those sets weren't equal
17:                $E(n_1) \leftarrow E(n_2) \leftarrow \bigcup C$                     ▷ Merge sets in $E$
18:                $T \leftarrow T \cup \{(n_1, n_2)\}$                   ▷ The new pair is added to $T$
19:    if Any set in $E$ contains both final and non-final states then return False
20:    else return True
```

mutation in the *minLength* : $2^{\Sigma^* \times \Sigma^*} \to \mathbb{N}$ function. For mutations in Section 5, $minLength(Loss) = minLength(Corruption) = minLength(Stutter) = 1$ and $minLength(OutOfOrder) = 2$. Note that *minLength* for unions must increase to permit the application of both mutations on a string. For example, $minLength(Loss \cup Corruption) = 2$. This length guarantees that each string has at least one mutation, which is sufficient to show immunity by Theorem 2.

The algorithm works as follows. We assume a mutation can occur at any time, so we begin by following transitions for pairs of mutated and unmutated strings from every reachable state (stored in the set $R$). On Lines 5-9, for each pair $(\sigma, \sigma')$ in $M$ and for each reachable state, we compute the states $q_1$ and $q_2$ reached from $\sigma$ (respectively $\sigma'$). The map $E$ contains equivalence classes, which we update for $q_1$ and $q_2$ to hold the set containing both states. The pair of states is also added to the worklist $T$, which contains equivalent states from which string suffixes must be explored.

The loop on Lines 10-18 then explores those suffixes. It takes a pair of states $(q_1, q_2)$ from the worklist and follows transitions from those states to reach $n_1$ and $n_2$. If $n_1$ and $n_2$ are already marked as equivalent to other states in $E$ or aren't marked as equivalent to each other, those states are added to the worklist, and their equivalence classes in $E$ are merged. If at the end, there is an equivalence class with final and non-final states, then $\mathcal{A}$ is not immune to $M$.

**Theorem 4.** (Immunity Procedure Correctness) *Algorithm 1 is sound and complete for any DFA and prefix-closed mutation. That is, given a DFA* $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$, *and a mutation,* $M$, IMMUNE$(\mathcal{A}, M) \Leftrightarrow \mathcal{A}$ *is immune to* $M$.

*Proof:* By Definition 16, this is equivalent to showing that $\textsc{Immune}(\mathcal{A}, M) \Leftrightarrow (\forall(\sigma, \sigma') \in M,\ \sigma \in \mathcal{L}(\mathcal{A}) \Leftrightarrow \sigma' \in \mathcal{L}(\mathcal{A}))$.

We will prove the $\Rightarrow$ direction (soundness) by contradiction. Suppose at the completion of the algorithm that all sets in $E$ contain only final or non-final states, but that $\mathcal{A}$ is not immune to $M$. There is at least one pair $(\sigma_b, \sigma'_b) \in M$ where one leads to a final state, and one does not. If Algorithm 1 had checked this pair then these states would be in an equivalence class in $E$. Since the loop on Line 7 follows transitions for pairs in $M$ of length $\mathit{minLength}(M)$, the reason $(\sigma_b, \sigma'_b)$ was not checked must be because $|\sigma_b| \neq \mathit{minLength}(M)$. The length of $\sigma_b$ must be greater than $\mathit{minLength}(M)$ since strings shorter than $\mathit{minLength}(M)$ cannot be mutated by $M$. Since $M$ is prefix closed, there must be a pair $(\sigma, \sigma') : |\sigma| = \mathit{minLength}(M)$ that are prefixes of $(\sigma_b, \sigma'_b)$. The loop on Line 10 will check $(\sigma \cdot s, \sigma' \cdot s),\ \forall s \in \Sigma^*$. Therefore it must be the case that $\sigma_b = \sigma \cdot t,\ \sigma'_b = \sigma' \cdot u : t, u \in \Sigma^*,\ t \neq u$. However, if $t \neq u$ then $(\sigma_b, \sigma'_b) \in M^k : k > 1$, so $\mathcal{A}$ is immune to $M^1$ but not $M^k$, but from Theorem 2 this is a contradiction.

We prove the $\Leftarrow$ direction (completeness) by induction. We will show that if $\mathcal{A}$ is immune to $M$ then no set in $E$, and no pair in $T$ will contain both final and non-final states. The base case at initialization is obviously true since every set in $E$ contains only one state and $T$ is empty. The induction hypothesis is that at a given step $i$ of the algorithm if $\mathcal{A}$ is immune to $M$ then every set in $E$ and every pair in $T$ contains only final or non-final states.

At step $i + 1$, in the loop on Line 7, $E$ and $T$ are updated to contain states reached by following $\sigma$ and $\sigma'$. Clearly, if $\mathcal{A}$ is immune to $M$ then these states must be both final or non-final since we followed transitions from reachable states for a pair in $M$. In the loop on Line 10, $n_1$ and $n_2$ are reached by following the same symbol in the alphabet from a pair of states in $T$. If $\mathcal{A}$ is immune to $M$, the strings leading to that pair of states must both be in, or both be out of the language. So, extending both strings by the same symbol in the alphabet creates two strings that must both be in or out of the language. These states reached by following these strings are added to $T$ on Line 18.

On Lines 15 and 17, the two sets in $E$ corresponding to $n_1$ and $n_2$ are merged. Since both sets must contain only final or non-final states, and one-or-both of $n_1$ and $n_2$ are contained in them, the union of the sets must also contain only final or non-final states. □

**Theorem 5.** (Immunity Procedure Complexity) *Algorithm 1 is Fixed-Parameter Tractable. That is, given a DFA $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$, and a mutation, $M$, its maximum running time is $|Q|^{O(1)} f(k)$, where $f$ is some function that depends only on some parameter $k$.*

*Proof:* The run-time complexity of Algorithm 1 is $O(n)O(m^l\ f(M))$ where $n = |Q|$, $m = |\Sigma|$, $l = \mathit{minLength}(M)$, and $f$ is a function on $M$. First, Lines 4, 7, 8, 9, 11, 12, 14, 15, 16, 17, and 18 execute in constant time, while each of Lines 2, 3, and 19 run in time bounded by $n$.

The initialization loop at Line 5 runs once for each pair in the mutation where the length of $\sigma$ is bounded by $\mathit{minLength}(M)$. This count is $m^l$ times a factor $f(M)$ determined by the mutation. For example, $f(\mathit{Loss}) = l$ because each $\sigma$ is mutated to remove each symbol in the string. Critically, this factor $f(M)$ must be finite, which it is for the mutations $\mathcal{M}^1$. The loop at Line 6 runs in time bounded by $n$, so the body of the loop is reached at most $m^l f(M)n$ times.

The loop at Line 10 may run at most $m^l f(M) + n$ times. The loop continues while the worklist $T$ is non-empty. Initially, $T$ has $m^l f(M)$ elements. Each time Line 12 runs, an element is removed from the worklist. For an element to be added to $T$, it must contain states corresponding to sets in $E$ which are not identical. When this occurs, those two corresponding sets are merged, so the number of unique sets in $E$ is reduced by at least one. Therefore, the maximum number of times Line 18 can be reached and an element added to $T$ is $n$. □

Note that, in practice, *minLength*$(M)$ is usually small (often only one), so Algorithm 1 achieves near linear performance in the size of the FA. The size of the alphabet has an effect but it is still quadratic.

## 9 Discussion

The mutations from Definitions 7 to 10 are useful abstractions of common problems in communication. However, in many cases, they are stronger than is needed as practitioners may have knowledge of the channel that constrains the mutations. For example, in MSL, messages contain sequence numbers which can be used to narrow the range of missing symbols. An advantage of our method is that custom mutations can be easily defined and then tested using Algorithm 1. Custom mutations should avoid behavior that requires long strings to mutate, however, as this causes exponential slowdown.

Well designed mutations like those in Section 5 can be checked quickly. However, the method relies on $\mathbb{B}_3$ monitor construction to obtain DFAs, and the procedure to create them from an NBA is in 2EXPSPACE. We argue that this is an acceptable cost of using the procedure since a monitor must be derived to check the property in any case. Future work should explore ideas from the study of monitorability [11, 22] to find a theoretical bound on deciding immunity.

Another avenue for improving on our work is to characterize classes of properties that are immune to different mutations. The classes of monitorable properties under different definitions in Section 4 are mostly understood [13, 22]. Finding a similar classification for the immunity of properties to mutations would be useful. It is already understood that all LTL properties without the next $(X)$ operator are immune to *Stutter* [19, 23].

## 10 Related Work

Unreliable channels have been acknowledged in formal methods research for some time. One area where the unreliable communication channels are commonly modeled is where Communicating Finite State Machines (sCFSMs) are used to verify network protocols. Abdulla and Jonsson provided algorithms for deciding the termination problem for protocols on lossy first-in first-out (FIFO) buffers, as well as algorithms for some safety and eventuality properties [2]. Cécé et al. also considered channels with insertion errors and duplication errors [8].

Work has been done to show which properties are verifiable on a trace with mutations and to express degrees of confidence when they are not. Stoller et al. used Hidden Markov Models (sHMMs) to compute the probability of a property being satisfied on a lossy trace [26]. Their definition of lossy included a "gap" marker indicating where symbols were missing. They used HMMs to predict the missing states where gaps occurred and aided their estimations with a learned probability distribution of state transitions. Joshi et al. introduced an algorithm to determine if a specification could be monitored soundly in the presence of a trace with transient loss, meaning that eventually it contained successfully transmitted events [16]. They defined monotonicity to identify properties for which the verdicts could be relied upon once a decision was made.

Garg et al. introduced a first-order logic with restricted quantifiers for auditing incomplete policy logs [14]. The authors used restricted quantifiers to allow monitoring policies that would, in principle, require iterating over an infinite domain. Basin et al. also specified a first-order logic for auditing incomplete policy logs [4]. Basin et al. also proposed a semantics and monitoring algorithm for Metric Temporal Logic (MTL) with freeze quantifiers that was sound and complete for unordered traces [3]. Their semantics were based on a three-value logic, and the monitoring algorithm was evaluated over ordered and unordered traces. All three of these languages used a three value semantics $(t, f, \perp)$ to model a lossy trace, where $\perp$ represented missing information.

Li et al. examined out-of-order data arrival in Complex Event Processing (CEP) systems and found that SASE [27] queries processed using the Active Instance Stack (AIS) data structure would fail in several ways [20]. They proposed modifications to AIS to support out-of-order data and found acceptable experimental overhead to their technique.

## 11 Conclusion

The ability to check properties expressible by NBAs for monitorability over unreliable channels allows RV to be considered for applications where RV would have previously been ignored. To arrive at this capability, we first needed to define monitorability over unreliable channels using both existing notions of monitorability and a new concept of mutation immunity. We proved that immunity to a single application of a mutation is sufficient to show immunity to any number of applications of that mutation, and we defined true-false immunity using $\mathbb{B}_3$ semantics. We believe unreliable communication is an important topic for RV and other fields that rely on remote systems, and we hope that this work leads to further examination of unreliable channels in the RV community.

## Acknowledgements

The authors would like to thank Rajeev Joshi for his contributions to the work.

## References

1. Embedded trace macrocell architecture specification (5 2019), Available online: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0014q/
2. Abdulla, P.A., Jonsson, B.: Verifying programs with unreliable channels. Information and Computation 127(2), 91–101 (1996)
3. Basin, D., Klaedtke, F., Zălinescu, E.: Runtime verification of temporal properties over out-of-order data streams. In: Majumdar, R., Kunčak, V. (eds.) Computer Aided Verification. pp. 356–376. Springer International Publishing, Cham (2017)
4. Basin, D.A., Klaedtke, F., Marinovic, S., Zalinescu, E.: Monitoring compliance policies over incomplete and disagreeing logs. In: RV. pp. 151–167. Springer (2012)
5. Bauer, A., Leucker, M., Schallhart, C.: Monitoring of real-time properties. In: Arun-Kumar, S., Garg, N. (eds.) FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science. pp. 260–272. Springer Berlin / Heidelberg, Berlin, Heidelberg (2006)
6. Bauer, A., Leucker, M., Schallhart, C.: Comparing LTL semantics for runtime verification. Journal of Logic and Computation 20(3), 651–674 (2010)
7. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. ACM Transactions on Software Engineering and Methodology (TOSEM) 20(4), 14:1–14:64 (9 2011)
8. Cécé, G., Finkel, A., Iyer, S.P.: Unreliable channels are easier to verify than perfect channels. Information and Computation 124(1), 20–31 (1996)
9. Chen, Z., Wu, Y., Wei, O., Sheng, B.: Deciding weak monitorability for runtime verification. In: Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings. pp. 163–164. ICSE '18, ACM, New York, NY, USA (2018)
10. d'Amorim, M., Roşu, G.: Efficient monitoring of $\omega$-languages. In: Etessami, K., Rajamani, S.K. (eds.) Computer Aided Verification. pp. 364–378. Springer Berlin / Heidelberg, Berlin, Heidelberg (2005)
11. Diekert, V., Muscholl, A., Walukiewicz, I.: A note on monitors and Büchi automata. In: Leucker, M., Rueda, C., Valencia, F.D. (eds.) Theoretical Aspects of Computing - ICTAC 2015. pp. 39–57. Springer International Publishing, Cham (2015)
12. Falcone, Y., Fernandez, J.C., Mounier, L.: Runtime verification of safety-progress properties. In: Bensalem, S., Peled, D.A. (eds.) Runtime Verification. pp. 40–59. Springer Berlin / Heidelberg, Berlin, Heidelberg (2009)
13. Falcone, Y., Fernandez, J.C., Mounier, L.: What can you verify and enforce at runtime? International Journal on Software Tools for Technology Transfer 14(3), 349–382 (6 2012)
14. Garg, D., Jia, L., Datta, A.: Policy auditing over incomplete logs: Theory, implementation and applications. In: Proceedings of the 18th ACM Conference on Computer and Communications Security. pp. 151–162. CCS '11, ACM, New York, NY, United States (2011)
15. Hopcroft, J.E., Karp, R.M.: A linear algorithm for testing equivalence of finite automata. Tech. rep., Cornell University (1971)
16. Joshi, Y., Tchamgoue, G.M., Fischmeister, S.: Runtime verification of LTL on lossy traces. In: Proceedings of the Symposium on Applied Computing. pp. 1379–1386. SAC '17, ACM, New York, NY, United States (2017)
17. Kupferman, O., Vardi, M.Y.: Model checking of safety properties. Formal Methods in System Design 19(3), 291–314 (11 2001)

18. Kupferman, O., Vardi, M.Y.: Weak alternating automata are not that weak. ACM Trans. Comput. Logic 2(3), 408–429 (07 2001)
19. Lamport, L.: What good is temporal logic? In: IFIP congress. vol. 83, pp. 657–668 (1983)
20. Li, M., Liu, M., Ding, L., Rundensteiner, E.A., Mani, M.: Event stream processing with out-of-order data arrival. In: Distributed Computing Systems Workshops, 2007. ICDCSW '07. 27th International Conference on. pp. 67–67 (June 2007)
21. Lozes, É., Villard, J.: Reliable contracts for unreliable half-duplex communications. In: Carbone, M., Petit, J.M. (eds.) Web Services and Formal Methods. pp. 2–16. Springer Berlin / Heidelberg, Berlin, Heidelberg (2012)
22. Peled, D., Havelund, K.: Refining the Safety–Liveness Classification of Temporal Properties According to Monitorability, pp. 218–234. Springer International Publishing, Cham (2019)
23. Peled, D., Wilke, T.: Stutter-invariant temporal properties are expressible without the next-time operator. Information Processing Letters 63(5), 243–246 (1997)
24. Pnueli, A., Zaks, A.: PSL model checking and run-time verification via testers. In: Misra, J., Nipkow, T., Sekerinski, E. (eds.) FM 2006: Formal Methods. pp. 573–586. Springer Berlin / Heidelberg, Berlin, Heidelberg (2006)
25. Sistla, A.P., Clarke, E.M.: The complexity of propositional linear temporal logics. J. ACM 32(3), 733–749 (Jul 1985)
26. Stoller, S.D., Bartocci, E., Seyster, J., Grosu, R., Havelund, K., Smolka, S.A., Zadok, E.: Runtime verification with state estimation. Runtime Verification 11, 193–207 (2011)
27. Wu, E., Diao, Y., Rizvi, S.: High-performance complex event processing over streams. In: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data. pp. 407–418. SIGMOD '06, ACM, New York, NY, United States (2006)