LTL-Grey : An LTL semantics for quantitative evaluation of LTL specifications

Raphaël Khoury
Department of Computer Science and Mathematics
Université du Québec à Chicoutimi

Sylvain Hallé
Department of Computer Science and Mathematics
Université du Québec à Chicoutimi

October 3, 2017

Abstract

When monitoring a trace using a LTL specification, the verdict returned by the monitor can often be insufficiently informative to be actionable. In this paper, we propose a generalization of LTL that allows formulae to evaluate to natural or real values, thus yielding quantitative information about the underlying trace. We illustrate with examples how this logic can be used to verify meaningful properties of traces. We provide an automata-based representation of this new logic as well as an implementation using the BeepBeep 3 complex event processor.

1 Introduction

There has recently been an interest in providing a quantitative, rather than qualitative verdict, on the evaluation of a specification, or security policy. As several authors have observed, a reductionist 3-valued judgement on the evaluation of a formula on a trace is often insufficiently informative. A more practical procedure that provides indication to the manner, or degree at which a property is satisfied or violated could be much more practical. For instance, if the specification states that any opened file is eventually closed, it is useful to distinguish between those sequences that trivially respect the property since no files are opened during the execution, from those that do so because any opened file is promptly closed. Likewise, can also be useful to distinguish the number of execution steps that elapse from the moment a file is opened, until it is eventually closed, or the number of completed open-close pairs that are present in otherwise invalid executions.

In this paper, we suggest an extension of LTL, called LTL-grey, with associated semantics, which provides such quantitative judgement. LTL-grey allows users to specify formulas that evaluate to numerical properties of traces, such as the numbers of occurrences of specific formal patterns, or the distance in the trace between such patterns. We argue that this formalism allows for a more informative judgement.

Consider the following two motivating examples

- Every *open* action is eventually followed by a corresponding *close* action. This property can be stated in LTL as $open \rightarrow \mathbf{F}\, close$. Observe that evaluating this property on a trace with the LTL's usual 3-valued semantics will always always return the uninformative '?' verdict, for any trace. Using the proposed logic, we may be able to state or verify questions such as :
  - How many completed open/closed pairs occur in the trace?
  - What are the longest, shortest and average times from open to close?
  - How long after a file is closed until it is eventually reopened?
  - Considering the entire execution trace, during how much of it are there open files waiting to be closed?
- A *request* action must occurs before any *allocate* action occurs. This property can be stated in LTL as $\neg request\, \mathbf{U}\, allocate$. Once again we can state finer questions such as:
  - How long before *allocate* does does *request* occurs?
  - How many *request*s occurs before *allocate*?

The remainder of this paper is organised as follows. Section 2, we examines related works and Section 3 presents some preliminary notions related to traces and temporal logic. In Section 4, we sketch out the syntax and semantics of our proposed extension to LTL. Section 5 provides case use examples that illustrate the flexibility and versatility of the proposed logic. Next, in section 6, we provide an automata-based representation for the proposed logic. Concluding remarks are given in section 8.

2 Related Work

There has recently been much interest in a finer judgement of property validation that an elementary 2 or 3 valued judgement. Bauer et al. suggest a 4-valued interpretation of LTL over finite traces, which distinguishes between a sequence that not yet respect the property but might, if the execution is allowed to proceed with no intervening event, and conversely one that does not yet irremediably violate the property, but likely will if corrective action does not occur. The two added judgements, possibly true possible false, are refinements of the undecided judgement '?', with $\top$ and $\bot$ retaining their usual semantic meaning. In [8], Medhat et al. extend this idea further by proposing a 6-valued semantics for LTL over finite traces. Once again, the additional judgements are refinements of '?' and $\top$ and $\bot$ retaining their usual semantic meaning. As the authors note "*We claim that these truth values provide us with informative verdicts about the status of different components of properties [...] at run time* ". The insight that a multi-valued verdict is more informative and actionable is the driving motivation behind the development of LTL-grey.

Our work can be seen as a generalization of theirs. In particular, the logic proposed in this paper provides refinements to the basic judgement $\bot$ and $\top$, which can indicate the degree to which a specification is respected by a trace or the easy at which the specification is met.

Security policies naturally lend themselves to a quantitative judgement. Security policies naturally lend themselves to a quantitative judgement. Such a judgment captures the intuitive notion of expressing the severity of a violation, rather than simply its occurrence, an information that is necessary when selecting a security policy enforcement mechanism. In [9], Ligatti et al. argue that a quantitative judgement indicating the degree to which a security policy is respected would provide multiple benefits. They further generalize the well-established notions of safety and liveness to such a quantitative enforcement context. The extension of LTL proposed in this paper allows users to define such quantitative security properties in a formal manner.

Other researchers focused on enforcement [7, 3, 2], and argue that a quantitative judgement on security policy satisfaction can aid in selecting the most adequate reactive measure when faced with a potential violation of the security policy. In The work proposed in this paper provides a logical framework to assign a numerical value to each trace, thus allowing the selection of the appropriate corrective action to be based on solid theoretical footing.

3 Preliminaries

Let Σ be a finite or countably infinite set of atomic events. A *trace* is finite sequence of events from Σ. We write ϵ for the empty sequence and Σ^* for the set of all finite sequences. We let σ, τ range over sequences. We write $\tau;\sigma$ for the concatenation of τ and σ. We say that τ is a prefix of σ noted $\tau \preceq \sigma$ iff there exists a sequence σ' such that $\tau;\sigma' = \sigma$. The ith event in a sequence σ is given as σ_i, with the initial event being given as σ_1. The length of σ is written $|\sigma|$. We write $\sigma_{[i..]}$ for the suffix of σ starting on its ith event, and $\sigma_{[..i]}$ prefix of σ starting at its initial event and ending with event i.

LTL's syntax is based on classical propositional logic, using the connectives $\neg$ ("not"), $\vee$ ("or"), $\wedge$ ("and"), $\rightarrow$ ("implies"), to which five temporal operators have been added. An LTL formula is a well-formed combination of these operators and connectives, according to the usual construction rules:

Definition 1 (LTL Syntax)

1. *If x and y are variables or constants, then $x = y$ is an LTL formula;*
2. *If φ and ψ are LTL formulæ, then $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \rightarrow \psi$, $\mathbf{G}\,\varphi$, $\mathbf{F}\,\varphi$, $\mathbf{X}\,\varphi$, $\varphi\,\mathbf{U}\,\psi$ are LTL formulæ;*

Boolean connectives carry their usual meaning, and the following identities allow some connectors to be defined from proceeding ones : $\varphi \wedge \psi \equiv \neg(\neg\varphi \vee \neg\psi)$, $\varphi \rightarrow \psi \equiv \neg\varphi \vee \psi$, $\mathbf{G}\,\varphi \equiv \neg\mathbf{F}\,\neg\varphi$.

We build upon a three-valued semantics for LTL over finite sequences, in which a $\sigma \models \varphi$ evaluates to $\top$ iff every possible continuation of σ models φ; $\sigma \models \varphi$ evaluates to $\bot$ iff every possible continuation of σ does not respect φ, and $\sigma \models \varphi$ evaluates to '?' otherwise. We let v range over this set of truth values $V = \{\top, \bot, ?\}$. V forms a 3-valued lattice with $\top$ as supremum and $\bot$ as infimum, with $\bot \sqsubset ? \sqsubset \top$. When interpreted over this values from this lattice, $\wedge$ is equivalent to join and $\vee$ is equivalent to meet.

Definition 2 (LTL Semantics(from [1]))

$$\sigma \models p \;= \begin{cases} \top, & \textit{if } p \in \sigma_0; \\ \bot, & \textit{otherwise;} \end{cases}$$

$$\sigma \models \neg\varphi \;= \begin{cases} \top, & \textit{if } \sigma \models \varphi = \bot; \\ \bot, & \textit{if } \sigma \models \varphi = \top; \\ ?, & \textit{otherwise;} \end{cases}$$

$$\sigma \models \varphi \vee \psi = \begin{cases} \top, & \textit{if } \sigma \models \varphi \vee = \top \sigma \models \psi = \top; \\ \bot, & \textit{if } \sigma \models \varphi = \bot \vee \sigma \models \psi = \bot; \\ ?, & \textit{otherwise;} \end{cases}$$

$$\sigma \models \mathbf{X}\,\varphi \;= \begin{cases} \sigma_{[2..]} \models \varphi & \textit{if } |\sigma| \geq 2 \\ ? & \textit{otherwise;} \end{cases}$$

$$\sigma \models \mathbf{F}\,\varphi \;= \begin{cases} \top & \textit{if } \exists\, 1 \leq j \leq |\sigma| : \sigma_{..j} \models \varphi = \top \\ ? & \textit{otherwise;} \end{cases}$$

$$\sigma \models \mu\,\mathbf{U}\,\eta = \begin{cases} \top & \textit{if } \exists\, 1 \leq j \leq |\sigma| : ((\sigma_{..j} \models \eta = \top) \wedge (\forall\, 1 \leq k < j \sigma_{..k} \models \mu = \top)) \\ \bot & \textit{if } \exists\, 1 \leq j \leq n : ((\sigma_{..j} \models \mu = \bot) \wedge (\exists\, 1 \leq k \leq j \sigma_{..k} \models \eta = \bot)) \\ ? & \textit{otherwise;} \end{cases}$$

4 Syntax and Semantics

LTL-grey extends 3-valued LTL with two sets of semantic structures, namely counters and quantifiers. Counters, which range over $\mathcal{C}, \mathcal{D}$ and $\mathcal{L}$ and serve to count the number of events in a sequence that evaluate to a given truth value, usually $\top$, or the index in the trace at which a condition holds. For example, the counter $\mathcal{C}^{v}_{\varphi}$, where φ is an LTL formula and v ranges over the truth values of LTL, returns the number of prefixes of the input traces for which the evaluation of φ evaluates to v. The values returned by counters ranges over $\mathbb{N}$, but the outputs of multiples counters over the same sequence can be freely applied to the arithmetic operators or functions to compute information about the trace, yielding a value in $\mathbb{R}$. Quantifiers, such as $\forall_{\sim k}\mathcal{C}$ or $\exists_{\sim k}\mathcal{C}$, verify if the values returned by a counter over the trace meets a given condition c, and return a verdict in V. This allows unlimited recursion of alternating LTL formulae, counters and quantifiers.

Definition 3 *(LTL-Grey Syntax)*
$\varphi ::= \top \,|\, \bot \,|\, F\varphi \,|\, G\varphi \,|\, \ldots \,|\, \mathcal{Q}$
$\mathcal{C} ::= \mathcal{C}^v_\varphi \,|\, \mathcal{D}^v_\varphi \,|\, {}_\varphi\mathcal{D}^v_\phi \,|\, \mathcal{L}^\top_\varphi \,|\, \mathcal{C} \star \mathcal{C} \,|\, f(\mathcal{C})$
$\mathcal{Q} ::= \mathcal{P}_{\sim k}\mathcal{C} \,|\, \forall_{\sim k}\mathcal{C} \,|\, \exists_{\sim k}\mathcal{C}$
where v ranges over the values of V, f ranges over unary functions in $\mathbb{R} \times \mathbb{R}$, $\star$ ranges over arithmetic operators, $\sim \in \{<, >, \leq, \geq, =, \neq\}$ and $k \in \mathbb{R}$.

Since we expect the evaluation of a formula to yield a result in $\{\top, \bot, ?\} \cup \mathbb{R}$ we redefine the notion of satisfaction so that it yields a value in this range: $\Vdash\; : \Sigma^* \times \{\varphi \cup \mathcal{C} \cup \mathcal{Q}\} \times \{\top, \bot, ?\} \cup \mathbb{R}$. For clarity, we continue to freely use $\sigma \models \varphi$ and its negation $\sigma \not\models \varphi$ when φ is an LTL property with a truth value in $\{\top, \bot, ?\}$.

The counter $\mathcal{C}^v_\varphi$, where φ is an LTL formula and v ranges over the truth values of LTL, returns the number of prefixes of the input traces for which the evaluation of φ evaluates to v. We write $\mathcal{C}^{\geq ?}_\varphi$ as a stand-in for $\mathcal{C}^?_\varphi + \mathcal{C}^\top_\varphi$ and $\mathcal{C}^{\leq ?}_\varphi$ for $\mathcal{C}^?_\varphi + \mathcal{C}^\bot_\varphi$.

Definition 4 *(Semantics of $\mathcal{C}$)*
$\sigma \Vdash \mathcal{C}^v_\varphi = |\{\tau | \tau \preceq \sigma \wedge (\tau \models \varphi = v)\}|$.

The token $\mathcal{C}^\top_\top$ naturally returns the length of the trace.

Lemma 1 $\forall \sigma \in \Sigma^* : \sigma \Vdash \mathcal{C}^\top_\top = |\sigma|$.

The values returned by counters can easily be compared using arithmetic operators. For instance $\frac{\mathcal{C}^\top_p}{\mathcal{C}^\top_\top}$ returns the percentage of events in the trace that respect p.

In the course of experimenting with grey-LTL, it was frequently useful to identify the initial point in the input trace where a given property holds. We write $\mathcal{D}^v_\varphi$ for the counter that returns the initial position in σ where φ evaluates to v, or 0 if σ exhibits no prefix at which φ evaluates to v.

Definition 5 *(Semantics of $\mathcal{D}$)*
$$\sigma \Vdash \mathcal{D}^\top_\varphi = \begin{cases} i, & \textit{if } \exists i < |\sigma| : \sigma_i \models \varphi \wedge \forall j < i : \sigma_j \not\models \varphi; \\ 0, & \textit{otherwise.} \end{cases}$$

It is also convenient to identify the first position at which a condition holds, starting not from the beginning of the trace, but from the satisfaction of another condition. The binary counter ${}_\phi\mathcal{D}^v_\varphi$ returns this information, or 0 if σ does not exhibit such a prefix. ${}_\phi\mathcal{D}^v_\varphi$ is syntactic sugar and is equivalent to $max(\mathcal{D}^\top_\varphi - \mathcal{D}^\top_\phi, 0)$.

Definition 6 *(Semantics of binary $\mathcal{D}$)*
$$\sigma \Vdash {}_\phi\mathcal{D}^\top_\varphi = \begin{cases} k, & \textit{if } \exists i < |\sigma| : \sigma_i \models \phi \wedge \sigma_{i+k} \models \varphi \wedge \nexists j : j > i \wedge j < i + k \wedge \not\models \varphi; \\ 0, & \textit{otherwise.} \end{cases}$$

Finally, the syntax of LTL-grey also includes the counter *local*, given as $\mathcal{L}^v_\varphi$, which returns the index of the last occurrence of an event for which the property φ evaluates to v, or 0 if σ has no prefix for which that is the case.

Definition 7 *(Semantics of $\mathcal{L}$)*
$$\sigma \Vvdash \mathcal{L}^{\top}_{\varphi} = \begin{cases} i, & \textit{if } \exists i < |\sigma| : \sigma_{..i} \models \varphi = v \wedge \neg \exists j > i : \sigma_{..j} \models \varphi = v; \\ 0, & \textit{otherwise.} \end{cases}$$

In addition to counters, we enrich the semantics of LTL-grey with quantifiers. Quantifiers examine the value returned by a counter, and return a value from the same 3-valued truth domain as an LTL-property according to a condition subscripted to the quantifier. LTL-grey defines three quantifiers: the first two are the existential and an universal quantifiers, with natural semantics. For instance, the formula $\exists_{=5}\mathcal{C}^{\top}_{p}$ returns $\top$ if the atomic proposition p holds on at least 5 prefixes of the input trace, and returns $'?'$ otherwise. Conversely, the formula $\forall_{<0}\mathcal{C}^{\top}_{p} - \mathcal{C}^{\top}_{q}$ returns $\top$ if there exists a prefix of the input trace in which the atomic proposition q holds more often than p. The thirds quantifier is termed the propositional quantifier, and is written as $\mathcal{P}$. $\sigma \Vvdash \mathcal{P}_{\sim k}\mathcal{C}$ evaluates to $\top$ if the comparison $n \sim k$ holds where $n$is the value returned by $\sigma \Vvdash \mathcal{C}$. For example, let $\sigma = a; a; a; b; a;$ be a trace. the formula $\sigma_i \Vvdash \mathcal{P}_{=3}\mathcal{C}^{\top}_{a} = \top$ for $i = 3$ and $i = 4$, and it returns $\bot$ in all other cases.

Definition 8 *(Semantics of $\forall_{\sim k}\mathcal{C}$)*
$$\sigma \Vvdash \forall_{\sim k}\mathcal{C}^{v}_{\varphi} = \begin{cases} \bot, & \textit{if } \exists i < |\sigma| : \sigma_i \Vvdash \mathcal{C}^{v}_{\varphi} \nsim k; \\ ?, & \textit{otherwise.} \end{cases}$$

Definition 9 *(Semantics of $\exists_{\sim k}\mathcal{C}$)*
$$\sigma \Vvdash \exists_{\sim k}\mathcal{C}^{v}_{\varphi} = \begin{cases} \top, & \textit{if } \exists i < |\sigma| : \sigma_i \Vvdash \mathcal{C}^{v}_{\varphi} \sim k; \\ ?, & \textit{otherwise.} \end{cases}$$

Definition 10 *(Semantics of $\mathcal{P}_{\sim k}\mathcal{C}$)*
$$\sigma \Vvdash \mathcal{P}_{\sim k}\mathcal{C}^{v}_{\varphi} = \begin{cases} \top, & \textit{if } \sigma \Vvdash \mathcal{C}^{v}_{\varphi} \sim k; \\ \bot, & \textit{otherwise.} \end{cases}$$

Since quantifiers evaluate to a value in V, they can be freely used alongside with other constructs in complex LTL formulae. For example, the following formula states that if there are 4 or more requests for a resource waiting for a response then this fact must be logged : $(\exists_{>3}(\mathcal{C}^{\top}_{req} - \mathcal{C}^{\top}_{)} resp \rightarrow \mathbf{F}\ log.$

The counter $\mathcal{L}$ is also syntactic sugar, and can be defined in terms of $\mathcal{C}$ and $\exists$ as follows:$\mathcal{L}^{\top}_{\varphi} \equiv \mathcal{D}^{\top}_{\exists_{=\mathcal{C}^{\top}_{\varphi}}(\mathcal{C}^{\top}_{\varphi})}$.

The following identities over the relationship between LTL formulae, counters and quantifiers hold:

5 Examples

We illustrate the uses of LTL-grey using sample LTL formulae taken from *Spec Patterns*, an online repository of commonly used LTL patterns[1]. For each property, we will give a few examples of the types of quantitative properties that can

[1]http://patterns.projects.cs.ksu.edu/documentation/patterns/ltl.shtml

$$\begin{array}{ccc}
\mathbf{F}\,\exists_{\sim k}\mathcal{C}^{v}_{\varphi} \equiv \exists_{\sim k}\mathcal{C}^{v}_{\varphi} & \mathbf{G}\,\forall_{\sim k}\mathcal{C}^{v}_{\varphi} \equiv \forall_{\sim k}\mathcal{C}^{v}_{\varphi} & \mathcal{C}^{?}_{F\varphi} \equiv \mathcal{C}^{\top}_{\top} - \mathcal{C}^{\top}_{F\varphi} \\
\mathcal{C}^{\bot}_{\mathbf{G}\ \varphi} \equiv \mathcal{C}^{\top}_{\top} - \mathcal{C}^{\top}_{G\varphi} & \mathcal{C}^{?}_{\phi\ \ \mathbf{U}\ \ \varphi} \equiv \mathcal{C}^{\top}_{\phi} & |\mathcal{C}^{?}_{\phi\ \ \mathbf{U}\ \ \varphi} - \mathcal{C}^{\top}_{\top}| \equiv \mathcal{D}^{\top}_{\phi\ U\ \varphi} \\
\neg\mathcal{C}^{\top}_{\varphi} \equiv \mathcal{C}^{\leq ?}_{\varphi} & \neg\mathcal{C}^{\bot}_{\varphi} \equiv \mathcal{C}^{\geq ?}_{\varphi} & \neg\exists_{\sim k}\mathcal{C}^{v}_{\varphi} \equiv \forall_{\nsim k}\mathcal{C}^{v}_{\varphi} \\
\neg\forall_{\sim k}\mathcal{C}^{v}_{\varphi} \equiv \exists_{\nsim k}\mathcal{C}^{v}_{\varphi} & \mathcal{L}^{\top}_{\mathbf{F}\ \varphi} \equiv \mathcal{C}^{\top}_{\varphi} & \mathcal{L}^{\bot}_{\mathbf{G}\ \varphi} \equiv \mathcal{C}^{\bot}_{\varphi}
\end{array}$$

Figure 1: Identities over LTL-grey formulae

be stated using LTL-grey. As these examples illustrate, LTL-grey allows the properties that capture useful information about the trace under consideration.

In what follows p,q and r are atomic events.

- Consider the property stating that if p occurs, then q must occur at some point afterwards. This property is given as $G(p \to \mathbf{F}\,q)$, and is similar to the *open* $\to$ *close* example mentioned in the introduction. Recall that for this property, the usual 3-valued semantics of LTL always return the uninformative verdict '?', regardless of the input trace. Using the specification language proposed in this paper, we can state, and answer, questions that provide meaningful and information about the underlying execution. Many of these formulas evaluate to a numeric value, and thus provide a much more precise assessment of the trace than can be provided using a 3-values logic.
  - How long after the initial p does q first occur: ${}_{p}\mathcal{D}^{\top}_{q}$.
  - How many qs occur after the initial p, notwithstanding any q that occurs before: $max(\mathcal{C}^{\top}_{q} - \mathcal{C}^{>?}_{q \wedge \mathbf{G}\ \neg p}, 0)$
  - How many qs occur before the first occurrence of p: $\mathcal{C}^{?}_{q \wedge \mathbf{G}\ \neg p}$
  - How many event occur in situations where an action p has occurred, and has not yet been followed by a corresponding q action: $\mathcal{C}^{\top}_{\mathcal{P}_{>0}(\mathcal{L}^{\top}_{p} - \mathcal{L}^{\top}_{q})}$.
  - What percentage of the sequence consists in events that follow the occurrence of an p action, but precede the occurrence of the corresponding q: $\frac{\mathcal{C}^{\top}_{\mathcal{P}_{>0}(\mathcal{L}^{\top}_{p} - \mathcal{L}^{\top}_{q})}}{\mathcal{C}^{\top}_{\top}}$
  - Is there a interval of length more than n between the occurrence of an p action and the corresponding q: $(\exists_{>n}\mathcal{L}^{\top}_{q} - \mathcal{L}^{\top}_{p}) \wedge \mathbf{F}\,p$ The $\mathbf{F}$ p at the end ensures that a case in which a q occurs without any p preceding is not counted.
  - How many completed pairs of the form (p, q) occur in the sequence: $\mathcal{C}^{\top}_{(\exists_{=1}\mathcal{L}^{\top}_{q} - \mathcal{L}^{\top}_{p}) \wedge \mathbf{F}\ p}$.
- p becomes true before q, if it ever holds. This property is given in LTL as: $\neg p\,\mathbf{U}\,q$.
  - How many ps occur before the first occurrence of q : $\mathcal{C}^{\top}_{p \wedge \mathbf{G}\ \neg q}$.

- What percentage of the events occurring before the first q are ps : $min(\mathcal{D}^{\top}_{q}, \frac{\mathcal{C}^{\top}_{p \wedge \mathbf{G}\, \neg q}}{|\mathcal{D}^{\top}_{q} - 1|})$. (0 otherwise)
- Starting from the first p, how many events elapse until the first q is reached (not counting any proceeding ones): ${}_{p}\mathcal{D}^{\top}_{q}$.

• p holds at some point between q and r. This property is given in LTL as: $\mathbf{G}\,(q \wedge \neg r \rightarrow (p\,\mathbf{W}\,r))$.

- How many occurrences of p are there between the first q and the first r: $\mathcal{C}^{\top}_{p \wedge \mathbf{F}\, q} - \mathcal{C}^{\top}_{p \wedge \mathbf{F}\, r}$.
- What percentage of the events occurring q and r are ps : (0 if q has not occurred) $min({}_{q}\mathcal{D}^{\top}_{r}, \frac{\mathcal{C}^{\top}_{p \wedge \mathbf{F}\, q} - \mathcal{C}^{\top}_{p \wedge \mathbf{F}\, r}}{|{}_{q}\mathcal{D}^{\top}_{r} - 1|})$.
- Stating from the first occurrence of q, how many events elapse until the first occurrence of p is reached : $\mathcal{C}^{\top}_{\mathbf{F}\, q \wedge \neg p}$

As can easily be seen, these properties correspond to meaningful program behaviors, and an evaluation of these formulae will provide actionable information about the target program. Consider the first example, $G(p \rightarrow \mathbf{F}\, q)$, an important response property, can captures an essential requirement in resource management, indicating that if a given resource is acquired, that resource must eventually be released. The first LTL-grey property we propose, ${}_{p}\mathcal{D}^{\top}_{q}$ computes the number of steps that occur between acquisition and release, and informative refinement of the previous property. The next two properties capture a mismatch between the number of requests and the number of responses, which may indicate an inefficiency in the allocation process. The fourth property, $\mathcal{C}^{\top}_{\mathcal{P}_{>0}(\mathcal{L}^{\top}_{p} - \mathcal{L}^{\top}_{q})}$, indicate counts the numbers of events in the sequence that occur while the resource is in use, and the fifth property calculates the portion of the execution during which the resource is allocated. The penultimate property indicates whether or not the resource is ever acquired and held for more than n execution steps. The final property counts the number of times the resource and acquired and then released during the execution.

6 Automata Model

In this section, we propose an automata model for LTL-Grey, which draws upon the edit automaton[6]. For a given LTL-gey property φ, we build a chain of sequential edit automaton $\mathcal{E}_0, \mathcal{E}_1, \mathcal{E}_2, ...$ such that the output of one automaton is fed as input to the next.The initial automaton $\mathcal{E}_0$ takes as input σ. The output of the final edit automaton is the result of the evaluation of $\sigma \Vdash \varphi$.

The edit automaton is a finite or countably infinite state machine that receives an input sequence, and produces an alternate output sequence.

Definition 11 *Edit Automaton (from [6]) An edit automaton is a tuple* $\langle \Sigma, Q, q_0, \delta \rangle$

- Σ *is the input alphabet of the original input sequence;*
- Q *is a finite set of states;*
- $q_0 \in Q$ *is a distinguished initial state;*
- $\delta : (Q \times \Sigma) \to (Q \times \Sigma^*)$ *is a transition function.Given the current state and input action, it specifies the automatons output and successor state.*

The conversion from LTL-grey formula to automaton proceeds in three steps. First, any LTL subformula contained in the LTL property is translated to a finite state automaton using an appropriate translation algorithm (see [10] for eg.).

Definition 12 *Finite Deterministic Automaton A Finite State Automaton (FSA)* $\langle \Sigma, Q, q_0, \delta, F \rangle$ *is a finite deterministic state machine where:*

- Σ *is the input alphabet of the original input sequence;*
- Q *is a finite set of states;*
- $q_0 \in Q$ *is a distinguished initial state;*
- $\delta : Q \times \Sigma \times Q$ *is a transition function; and*
- $F \subseteq Q$ *is a (possibly empty) subset of* Q *that contains accepting states.*

Let $q \in Q$ be a state, We write $reach(q)$ for the set of states that are *reachable* from q, i.e., the set of states that can be reached from q be following any number of transition. For the sake of simplicity, the elements $\Sigma, Q, q_0, \delta ...$ defining an deterministic finite automaton or an edit automaton $\mathcal{A}$ are referred to using the formalism $\mathcal{M}.\Sigma, \mathcal{A}.Q, \mathcal{A}.q_0, \mathcal{A}.\delta ...$ or simply $\Sigma, Q, q_0, \delta ...$ when $\mathcal{A}$ is clear from the context.

Let $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, F \rangle$ be a finite state automaton that accepts language $\mathcal{L}_\varphi$ consisting of exactly those sequences for which φ evaluates to $\top$. We convert $\mathcal{A}$ into an edit automaton $\mathcal{E} = \langle \Sigma', Q', q'_0, \delta' \rangle$ that returns a value from V, in lockstep with each input action as follows:

- $\Sigma' = \Sigma \cup V$;
- $Q' = Q$ This allows a bijection between the states of $\mathcal{A}$ and those of $\mathcal{E}$. For any state $\mathcal{A}.q_i$ of $\mathcal{A}$ there is a corresponding state $\mathcal{E}.q_i$ of $\mathcal{E}$. We write $\mathcal{A}.q_i \leftrightharpoons \mathcal{E}.q_i$ to indicate this relationship.
- $q'_0 = q_0$, where $q'_0 \leftrightharpoons q_0$.
- $\mathcal{E}.\delta'(\mathcal{E}.q, a) = \begin{cases} (q', \top), & \text{if } \delta(q,a) = \mathcal{A}.q' \text{ and } \mathcal{A}.q' \leftrightharpoons \mathcal{E}.q' \text{ and } \mathcal{A}.q' \in F \\ (q', \bot), & \text{if } \mathcal{A}.\delta(q,a) = \mathcal{A}.q' \text{ and } \mathcal{A}.q' \leftrightharpoons \mathcal{E}.q' \\ & \quad \text{and } reach(\mathcal{A}.q') \cap \mathcal{A}.F = \emptyset \\ (q', ?), & \text{otherwise.} \end{cases}$

Let $\mathcal{E} = \langle \Sigma, Q, q_0, \delta \rangle$ be an edit automaton as constructed above, and capturing a 3-valued evaluation of a formula φ. We can constructed an edit automaton $\mathcal{E}' = \langle \Sigma', Q', q'_0, \delta' \rangle$ capturing the semantics of the counter $\mathcal{C}^v_\varphi$ as follows:

- $\Sigma' = \mathbb{N}$;
- $Q' = \mathbb{N}$;
- $q'_0 = 0$;
- $\mathcal{E}'.\delta(\mathcal{E}'.n, a) = \begin{cases} (n+1, n+1), & \text{if } a = v \\ (n, n), & \text{otherwise.} \end{cases}$

The counter for $\mathcal{D}$ is only slightly more involved. As discussed above, binary $\mathcal{D}$ and $\mathcal{L}$ are syntactic sugar.

- $\Sigma' = \mathbb{N}$;
- $Q' = \langle \mathbb{N} \times \mathbb{B} \rangle$;
- $q'_0 = 0 \times \bot$;
- $\mathcal{E}'.\delta(\langle \mathcal{E}.n, b\rangle, a) = \begin{cases} (\langle n+1, \top\rangle, n+1), & \text{if } a = v \wedge b = \bot \\ (\langle n, \top\rangle, n), & \text{if } a = v \wedge b = \top \\ (\langle n+1, \bot\rangle, 0), & \text{otherwise.} \end{cases}$

Finally, an NFA with input alphabet $\mathbb{N}$ or $\mathbb{R}$, can receive as input the output of an edit automaton as described above, (or the results of applying arithmetic operators or functions to the results of multiple such automata in lockstep), and accepts a language equivalent to the LTL formula captured by an quantifier. Let $\mathcal{Q}_{\sim k}\mathcal{C}^v_\varphi$ be a quantifier and let σ be the input sequence. We generate a new input sequence σ' over the alphabet $\{\top, \bot\}$ where $\forall i < |\sigma| : \sigma'_i = (\sigma_i \Vdash \mathcal{C}^v_\varphi) \sim k$. The quantifiers $\mathcal{P}_{\sim k}\mathcal{C}, \forall_{\sim k}\mathcal{C}$ and $\exists_{\sim k}\mathcal{C}$ can then be stated as the LTL formulae v, $\mathbf{G}\, v$ and $\mathbf{F}\, v$ respectively.

7 Implementation with BeepBeep

LTL-grey was implemented using the event processing tool BeepBeep [5]. Each processor takes no more that 20 lines of code.

8 Conclusions

In this paper we proposed a generalization of LTL that allows formulae to evaluate to natural or real values, thus yielding quantitative information about the underlying trace. We posit that this language allows to users to formally state and verify meaningful properties about the underlying traces, and illustrate this with examples.

In the future, we are interested in using grey-LTL in the context of runtime enforcement. When confronted with a potential violation of the security policy, runtime enforcement requires selecting the appropriate corrective action, from within a set of possible reactions. By providing a metric for comparing executions that rests on solid theoretical founding, we beleive that LTL-grey can form the basis for this selection.

References

[1] A. Bauer, M. Leucker, and C. Schallhart. Monitoring of real-time properties. In S. Arun-Kumar and N. Garg, editors, *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science, 26th International Conference, Kolkata, India, December 13-15, 2006, Proceedings*, vol-

ume 4337 of *Lecture Notes in Computer Science*, pages 260–272. Springer, 2006.

[2] P. Drábik, F. Martinelli, and C. Morisset. Cost-aware runtime enforcement of security policies. In A. Jøsang, P. Samarati, and M. Petrocchi, editors, *Security and Trust Management - 8th International Workshop, STM 2012, Pisa, Italy, September 13-14, 2012, Revised Selected Papers*, volume 7783 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2012.

[3] P. Drábik, F. Martinelli, and C. Morisset. *A Quantitative Approach for Inexact Enforcement of Security Policies*, pages 306–321. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[4] Y. Falcone and C. Sánchez, editors. *Runtime Verification - 16th International Conference, RV 2016, Madrid, Spain, September 23-30, 2016, Proceedings*, volume 10012 of *Lecture Notes in Computer Science*. Springer, 2016.

[5] S. Hallé. When RV meets CEP. In Falcone and Sánchez [4], pages 68–91.

[6] J. Ligatti, L. Bauer, and D. Walker. Edit automata: enforcement mechanisms for run-time security policies. *Int. J. Inf. Sec.*, 4(1-2):2–16, 2005.

[7] F. Martinelli, I. Matteucci, and C. Morisset. From qualitative to quantitative enforcement of security policy. In *Proceedings of the 6th International Conference on Mathematical Methods, Models and Architectures for Computer Network Security: Computer Network Security*, MMM-ACNS'12, pages 22–35, Berlin, Heidelberg, 2012. Springer-Verlag.

[8] R. Medhat, B. Bonakdarpour, S. Fischmeister, and Y. Joshi. Accelerated runtime verification of LTL specifications with counting semantics. In Falcone and Sánchez [4], pages 251–267.

[9] D. Ray and J. Ligatti. A theory of gray security policies. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, Sept. 2015.

[10] M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Inf. Comput.*, 115(1):1–37, 1994.