Trace-Length Independent Runtime Monitoring of Quantitative Policies

Xiaoning Du, Alwen Tiu, Kun Cheng, *Student Member, IEEE*, and Yang Liu

Abstract—Metric linear-time logic (MTL) has been widely used to specify runtime policies. Traditionally this use of MTL is to capture the qualitative aspects of the monitored systems, but recent developments in its extensions with aggregate operators allow some quantitative policies to be specified. Our interest in MTL-based policy languages is driven by applications in runtime malware or intrusion detection in platforms like Android and autonomous vehicles, which requires the monitoring algorithm to be independent of the length of the system event traces so that its performance does not degrade as the traces grow. We propose a policy language based on a past-time variant of MTL, extended with an aggregate operator called the metric temporal counting quantifier to specify a policy based on the number of times some sub-policies are satisfied in the specified past time interval. We show that a broad class of policies, but not all policies, specified with our language can be monitored in a trace-length independent way, and provide a concrete algorithm to do so. We implement and test our algorithm in both an existing Android monitoring framework and an autonomous vehicle simulation platform, and show that our approach can effectively specify and monitor quantitative policies drawn from real-world studies.

Index Terms—Runtime monitoring, counting quantifier, MTL, trace-length independent, runtime attack detection

1 Introduction

Metric linear-time logic (MTL) [1] and the non-metric linear temporal logic (LTL) [2] have been widely used as a specification language to specify runtime properties of systems and languages. Traditionally, this use of temporal logic is concerned mainly with qualitative properties, such as relative ordering of events, or eventuality of events, etc. MTL is an extension of LTL, in which operators are bounded with time intervals, and is able to specify timing-dependent properties. MTL-based specification language is widely utilized to describe policies for runtime malware detection in Android [3]. Such application in runtime monitoring is mainly because efficient monitoring algorithms are usually available for such logic specification language, and the monitoring scripts can be automatically generated for any given policies. However, there exist some attack patterns that cannot be stated as pure MTL formulas. For example, plenty of malware will turn infected phones into bots to launch botnet attack [4], [5], where the phones are forced to conduct certain activities with abnormal high frequency. An effective way to detect such attack at runtime is to monitor the frequency of events of interests, like SMS sending and socket opening. Currently, MTL lacks such quantitative semantics in policy specification.

One approach to solve this problem is to build into MTL a notion of counting of events [6], or more generally, aggregate operators [7]. But monitoring algorithms for such extensions have not been extensively studied in the literature. Particularly, they are expected to be time and space efficient considering the restricted storage and computation resources in Android. Extension of expressiveness always brings more challenges to the design of efficient algorithms. The monitoring problem for even a simple extension of LTL with the counting quantifier, as studied in [6], is already PSPACE complete (in the size of policy and the trace), and PTIME (in the size of the trace) when the policy is fixed. In the online monitoring, where near real-time decisions need to be made, the dependence of the monitor on the size of the trace would make it impractical even if its complexity is PTIME (assuming the policy is fixed), as its performance would degrade as the trace length grows. Our goal is thus to balance the expressivity of the policy specification language and the complexity of the monitoring algorithm.

In this work, we propose an extension of past-time MTL (ptMTL) [8], called MTL_{cnt}, to support a *metric temporal counting quantifier*, or metric counting quantifier for short, which is motivated by and extended from the counting quantifier proposed in [6]. In addition, we also introduce arithmetic relations to allow various restrictions on the count. MTL_{cnt} considers only the fragment of MTL with past time operators, as this is sufficient for our purpose to enforce history-sensitive access control. For monitoring on resource restricted platforms, like Android, once we fix the policy to be monitored, the space requirement of the monitoring algorithm is expected to be constant, i.e., independent of the length of the system event trace. Such type of monitoring algorithms are called as *trace-length independent* (TLI) monitoring algorithms [9]. In this work, we call a policy whose

- *X. Du, K. Cheng, and Y. Liu are with the School of Computer Science and Engineering, Nanyang Technological University, Singapore 639798, Singapore. E-mail: {duxi0002, chengkun, yangliu}@ntu.edu.sg.*
- *A. Tiu is with the Research School of Computer Science, Australian National University, Canberra ACT 0200, Australia. E-mail: alwen.tiu@anu.edu.au.*

Manuscript received 28 Jan. 2019; revised 7 May 2019; accepted 25 May 2019. Date of publication 29 May 2019; date of current version 13 May 2021. (Corresponding author: Yang Liu.)
Digital Object Identifier no. 10.1109/TDSC.2019.2919693

satisfiability can be checked by a TLI monitoring algorithm as a TLI-monitorable policy. Note that we require the TLI algorithm generated from a policy to be complete w.r.t. the policy, that is, every trace of events that adheres to the policy should also be accepted by the monitor, and every trace of events that violates the policy specification should be rejected. Otherwise the problem would be trivial as one could simply make various ad hoc restrictions such as restricting the time window for the monitoring.

A complete TLI monitoring algorithm for past-time LTL (ptLTL) is developed in [10]. For more expressive logics, such as first-order logics [6], [9] and our MTL$_{cnt}$, trace-length independence is not always possible, i.e., there are formulas for which the monitor needs to store the entire history of events and hence the space used to store critical information will increase as the traces grow. For example, with MTL$_{cnt}$, one can write a formula that compares the numbers of occurrences of two events, say e_1 and e_2. Let x and y denote the number of past occurrences of events of e_1 and e_2, respectively. To check the relation $x < y$ at any state, we would need to keep track of the difference between x and y; such a difference may grow as the trace grows, so the space requirement for monitoring this formula is not bounded. However, there are also instances of complex quantitative policies that are TLI-monitorable (see Section 7). One main contribution of our work is in deriving classes of policies that are TLI-monitorable.

To the best of our knowledge, there has been so far no study on TLI monitoring for MTL with aggregate operators like the counting quantifier. Our monitoring algorithm extends the monitoring algorithm for MTL [11], which uses a dynamic programming approach in determining the truth values of MTL formulas at any given state. Such a dynamic programming approach is derived from the recursively defined semantics of MTL, where the truth values of a formula at a state is computed compositionally from truth values of its subformulas at either the same state or a previous state. When designing the monitoring algorithm for MTL with counting quantifier, one main challenge is to make sure that one does not need to store explicit counters associated with events in the history. To solve this, we seek for the recursive definition of the semantics of counting quantifier. An important fact we leverage is that events happen incrementally result the count of events irreversibly grows unless some reset conditions are triggered, i.e., the count n is always observed earlier than $n + 1$. This is also true for counting with time interval constrains when properly resolving the time intervals. In the process, we realize not all of the counting policies are TLI-monitorable and determine the characteristics of the TLI-monitorable subgroup with proofs. Further, we deliver the recursive semantics for that subgroup of counting policies, and show how to automatically construct a TLI monitoring algorithm for them.

We have performed a number of security case studies on Android phones and autonomous vehicles to show the practicality of our specification language for malware detection and intrusion detection, respectively. The monitoring algorithm is implemented and evaluated on both an Android monitoring framework called LogicDroid [3] and an autonomous vehicle simulation platform based on Robot Operating System (ROS) [12]. The experimental results show that our approach can effectively specify and monitor a wide range of quantitative policies drawn from real-world applications.

We summarize the contributions of this paper as follows:

1) We extend MTL with an expressive and intuitive metric temporal counting quantifier and formally define its syntax and semantics. The extended logic MTL$_{cnt}$ can be used to describe a broad class of policies where quantitative semantics is needed.
2) We study and prove that a subgroup of counting policies can be monitored in a trace-length independent way. Based on that, the recursive semantics of the non-metric counting quantifier and the metric counting quantifier are developed and proved to facilitate the design of TLI monitoring algorithms.
3) We propose a complete TLI monitoring algorithm for the TLI monitorable fragment of MTL$_{cnt}$ with the soundness proof, and discuss on the worst case time and space complexity.
4) We perform several case studies on the runtime monitoring of software, hardware, and cyber-physical systems, where quantitative policies are needed and also our efficient TLI monitoring can be applied. With prototypes implemented on Android OS and autonomous vehicle simulators, we show that in practice our monitoring algorithm performs efficiently with low computation costs and memory footprint.

A preliminary version of this article has appeared in [13], with the focus on non-metric LTL with counting quantifier. In this work, we extend the policy language and the monitoring algorithms with the metric temporal counting quantifier. For the newly introduced metric temporal counting quantifier, the syntax, semantics, recursive semantics, monitoring algorithm as well as implementation and evaluation are all particularly re-designed and investigated. This is the first attempt to investigate the TLI monitoring of metric counting quantifiers. For the contributions above, other than the non-metric part in (2), which is from [13], (1), (3) and (4) are new contributions of this article. This article also contains detailed proofs, some of which are missing in [13].

2 Specification Language MTL$_{cnt}$

In this section, we propose a logic specification language MTL$_{cnt}$ by extending ptMTL [8] with a *metric temporal counting quantifier*. The counting quantifier helps to specify the times that a sub-policy has been satisfied in a past time period and checks whether the count satisfies some constraints. The syntax and semantics of MTL$_{cnt}$ are presented in Sections 2.1 and 2.2, respectively. The difference between the counting quantifier in MTL$_{cnt}$ and those from existing works, including [6], [7], [14], is discussed in Section 2.3.

2.1 Syntax of MTL$_{cnt}$

We denote with AP the set of atomic propositional variables. Elements in AP are ranged over by p, q and s. We assume a countably infinite set of constants and use a, b, c and d to range over integer constants. We assume an infinite set $\mathcal{V}$ of variables of integers, whose elements are ranged over by x, y and z. MTL$_{cnt}$ also admits common arithmetic operators such as $+$, $-$ and $\times$, and comparison operators like $=$, $<$, $>$, $\leq$

and $\geq$. Terms are built from constants, variables and arithmetic operators, and denoted by s, t, u and v.

The time model of MTL$_{cnt}$ follows the point-based time model adopted in [8]. We denote the time domain with $\mathbb{T}$, and in this work, we consider only discrete time domain with $\mathbb{T} = \mathbb{N}$.$\mathbb{T}$ is assumed with the standard ordering $\leq$. An *interval* is a non-empty set $I \subseteq \mathbb{T}$ such that, for all $\tau, \tau' \in I$ and $\kappa \in \mathbb{T}$, if $\tau < \kappa < \tau'$ then $\kappa \in I$. We denote with $\mathbb{I}$ the set of all time intervals. The lower bound and the upper bound of an interval I are denoted by $\ell(I)$ and $r(I)$. To simplify exposition, we require the time intervals to be left-closed and right-open. For a natural number n, we define $I \pm n = \{y \pm n | y \in I\} \cap [0, +\infty)$. A *time sequence* $(\tau_i)_{i \in \mathbb{N}_{\geq 1}}$ is a sequence of elements $\tau_i \in \mathbb{T}$ with $\tau_k \leq \tau_j$ for all $k \leq j$ and $k, j \in \mathbb{N}_{\geq 1}$. Here the time sequence is not required to be strictly increasing, that is successive elements in the sequence can have identical timestamps.

The syntax of MTL$_{cnt}$ is defined via the grammar

$$\phi := \bot \,|\, p \,|\, (t > 0) \,|\, \neg\phi \,|\, \phi \vee \phi \,|\, \odot_I \phi \,|\, \phi \mathcal{S}_I \phi \,|\, \mathcal{C}_I x : \langle \phi, \phi \rangle . \phi,$$

where ϕ is any MTL$_{cnt}$ formula, $p \in AP$, $I \in \mathbb{I}$, t ranges over terms and x ranges over variables. The operators appearing above are those of ptMTL except for the counting quantifier $\mathcal{C}$ and the relation $t > 0$. The operators subscripted with I are *metric temporal operators* and the subscripts can be omitted when $I = [0, +\infty)$ for succinctness. We assume the reader is familiar with the notion of free and bound variables. In the formula $\mathcal{C}_I x : \langle \alpha, \beta \rangle . \phi$, the variable x is a bound variable, whose scope is over ϕ, so x is not free in either formulas α or β. Roughly, the variable x is a placeholder that would contain a number associated with how many times β is satisfied within the interval I since α is satisfied. The formula α here acts as a counter *reset* condition, i.e., we only start counting once α is satisfied. The counting quantifier is generalized from a similar modality introduced in [6], and a detailed discussion about the difference will be given in Section 2.3. We also assume that bound variables in a formula are pairwise distinct. We write $\phi(x_1, \ldots, x_n)$ to mean that the free variables of ϕ are in $\{x_1, \ldots, x_n\}$ and we write $\phi(c_1, \ldots, c_n)$ to denote the instance of $\phi(x_1, \ldots, x_n)$ where c_i is the substitution for x_i.

In the definition of formulas, we have kept a minimum number of logical operators. The omitted operators can be derived using the given operators, e.g., propositional operators such as $\top$ (truth), $\wedge$ (conjunction), $\rightarrow$ (implication), and modal operators such as $\diamondsuit_I$ (sometimes in the past I), which is defined as $\diamondsuit_I \phi \equiv \top \mathcal{S}_I \phi$, and $\boxdot_I$ (always in the past I), which can be defined as $\boxdot_I \phi \equiv \neg \diamondsuit_I \neg \phi$. Note also that all other arithmetic relations can be derived from the relation of the form $(t > 0)$ and logical connectives: $s > t \equiv (s - t) > 0$, $s \leq t \equiv \neg(s > t)$, $s = t \equiv (s \leq t) \wedge (t \leq s)$, $s \geq t \equiv s > t \vee s = t$, and $s < t \equiv s \leq t \wedge \neg(s = t)$.

2.2 Semantics of MTL$_{cnt}$

A *trace* is a finite sequence of *states*, where each state itself consists of a set of atomic propositions. These atomic propositions correspond to events of interests that are being monitored in a system. We assume an *interpretation function* ξ which maps constant symbols to integers, and arithmetic operators and relation symbols to their corresponding semantic counterparts. We assume the usual arithmetic operators, and in addition, depending on applications, we may assume a fixed set of function symbols denoting computable functions over the integer domain. Since terms can contain variables, we additionally need to interpret these variables. This is done via a *valuation function*, i.e., a function from variables to integers. Formally, given an interpretation function ξ and a valuation function ν, the interpretation of a term t, written $t^{\xi,\nu}$ is defined as in first-order logic [15]. For example, $(5 + x)^{\xi,\nu}$ is defined as $5^{\xi} +^{\xi} \nu(x)$. If ν maps x to 2, then $(5 + x)^{\xi,\nu}$ is 7. Relations in integer arithmetic are interpreted similarly, e.g., $(x > 0)^{\xi,\nu}$ is interpreted as $(\nu(x) >^{\xi} 0^{\xi})$. To simplify the presentation, we drop the superscript ξ for familiar arithmetic symbols, e.g., instead of writing $+^{\xi}$ and $(5 + x)^{\xi,\nu}$, we simply write $+$ and $(5 + x)^{\nu}$ respectively.

A *model* for MTL$_{cnt}$ is a triple (ρ, τ, ν), where ρ is a trace, τ a time sequence, and ν is a valuation function, with $|\rho| = |\tau|$. For ρ (or τ), we write ρ_i (or τ_i) to denote its ith state (or timestamp). Each state ρ_i is timestamped with τ_i. For a valuation ν, we write $\nu[x \mapsto n]$ to denote the function which is identical to ν except for the valuation of x, i.e., $\nu[x \mapsto n](y) = \nu(y)$, when $y \neq x$, and $\nu[x \mapsto n](x) = n$.

Definition 1. *The satisfiability relation between a model (ρ, τ, ν), a* world *$i \in \mathbb{N}_{\geq 1}$ and a formula ψ, written as $\rho, \tau, \nu, i \models \psi$, is defined by induction on ψ as below, where $\rho, \tau, \nu, i \not\models \psi$ if $\rho, \tau, \nu, i \models \psi$ is false.*

- *$\rho, \tau, \nu, i \not\models \psi$ if $i < 1$ or $i > |\rho|$.*
- *$\rho, \tau, \nu, i \not\models \bot$.*
- *$\rho, \tau, \nu, i \models p$ iff $p \in \rho_i$.*
- *$\rho, \tau, \nu, i \models t > 0$ iff $t^{\nu} > 0$ is true.*
- *$\rho, \tau, \nu, i \models \neg\phi$ iff $\rho, \tau, \nu, i \not\models \phi$.*
- *$\rho, \tau, \nu, i \models \phi_1 \vee \phi_2$ iff $\rho, \tau, \nu, i \models \phi_1$ or $\rho, \tau, \nu, i \models \phi_2$.*
- *$\rho, \tau, \nu, i \models \odot_I \phi$ iff $\rho, \tau, \nu, i - 1 \models \phi$ and $\tau_i - \tau_{i-1} \in I, i \geq 2$.*
- *$\rho, \tau, \nu, i \models \phi_1 \mathcal{S}_I \phi_2$ iff there exists $j \leq i$ such that $\tau_i - \tau_j \in I$, $\rho, \tau, \nu, j \models \phi_2$ and $\rho, \tau, \nu, k \models \phi_1$ for all $j < k \leq i$.*
- *$\rho, \tau, \nu, i \models \mathcal{C}_I x : \langle \alpha, \beta \rangle . \phi$ iff $\rho, \tau, \nu[x \mapsto n], i \models \phi$ where $n = |\{j \,|\, \rho, \ \tau, \nu, j \models \beta \text{ for } \tau_i - \tau_j \in I \text{ and } m < j \leq i\}|$ and $m = max(\{j \,|\, \rho, \ \tau, \nu, j \models \alpha, \tau_i - \tau_j \in I, 1 \leq j \leq i\} \cup \{0\})$.*

We simply write $i \models \psi$ when the trace ρ, the time sequence τ and the valuation function ν are clear from the context.

Intuitively, the meaning of $\mathcal{C}_I x : \langle \alpha, \beta \rangle . \phi$ is as follows: suppose that β is true at exactly n states within the past time period I since the latest state where α holds; then the instance of ϕ with x mapped to n must also be true. Here is an example to give a more intuitive explanation about the semantics of the counting quantifier.

Example 1. For an authentication server (e.g., bank) which validates a user's credential, a common login policy can be that if a user fails to enter the correct password three times in a row, then the user's account is temporarily disabled. Let us consider only two system events: a correct password was entered (cp), and a wrong password was entered (wp) by a particular user. The logic policy can be specified as follows:

$$\boxdot[\neg(cp \wedge wp) \wedge (\mathcal{C}x : \langle cp, wp \rangle . x < 3)]. \tag{1}$$

The first conjunct expresses a consistency property, i.e., a password entered cannot be both correct and wrong at the same time. The variable x stores the number of times a wrong password was entered since the last time a correct password was entered (or since the beginning of the trace, if no correct password has been entered so far). Consider the event trace

$$\rho = \langle\{wp\};\{cp\};\{wp\};\{wp\};\{cp\};\{wp\}\rangle.$$

Since the formula is free of time intervals, there is no need to consider the time sequence. Then Formula (1) above is true at every state. For example, at ρ_4, the variable x gets instantiated to 2, since there are exactly two wp events since the latest cp event at this point. The wp event at ρ_1 is not counted in this case since it happened before the reset condition cp.

Intuitively, the counting quantifier can be used to express quantitative properties within a 'session' (e.g., an authentication session, a life cycle of a process, etc). One could introduce two events, $start$ and end, to mark the beginning and the end of a session. Then to check that the number of occurrences of an event e within a session is less than n, for example, one can simply use the formula $\mathcal{C}x : \langle start, e \wedge \neg end\rangle.\ x < n$ in conjunction with other formulas expressing the well-formedness of a session (e.g., every end corresponds to a $start$, etc). If e is a simple event (e.g., the wp event in Example 1), one could encode this in MTL using standard temporal operators, but at the expense of conciseness, i.e., one needs to expand the parameter n into n instances of $e \wedge \neg end$. For example, Example 1 can be alternatively specified as

$$\boxdot[\neg(cp \wedge wp) \wedge \neg(wp \wedge \odot(wp \wedge \odot wp)))].$$

That is, there cannot be three consecutive wp events any time in the past. However, this is the case only when there are no events being monitored other than cp and wp. When other events are possible, then we need to specify that events other than cp can happen in between two consecutive wp events. In general, in a formula $\mathcal{C}x : \langle\alpha, \beta\rangle.\phi$, any of the α, β and ϕ could be a complicated temporal formula, e.g., it could involve nested counting quantifiers and other temporal operators. In such a case, the encoding into pure MTL becomes less obvious and less concise. Therefore, it is necessary to introduce the counting quantifier for not only a simple way to compose formulas but also the possibility of designing an efficient monitoring algorithm.

2.3 Language Comparison

In [6], Bauer et al. introduce a counting quantifier $\mathcal{N}$ (defined in Definition 2) to LTL, which is similar to the nonmetric fragment of $\mathcal{C}$. We prove that the two quantifiers are with equivalent expressiveness, when $\mathcal{C}$ is with interval $[0, +\infty)$, as shown in Proposition 1.

Definition 2 ($\mathcal{N}$ counting quantifier). *Given a trace ρ, valuation v, a world i and the formula $\mathcal{N}x : \psi.\varphi$, the satisfiability is defined as*

$$\rho, v, i \models \mathcal{N}x : \psi.\varphi \ iff\ \rho, v[x \mapsto n], i \models \varphi \ where$$
$$n = |\{j \mid \rho, v, j \models \psi \ and\ 1 \le j \le i\}|.$$

Proposition 1. *The counting quantifiers $\mathcal{C}$ (with interval $[0, +\infty)$) and $\mathcal{N}$ are equivalent, i.e., one can be defined in terms of the other.*

Proof. Obviously, the quantifier $\mathcal{N}$ can be encoded using $\mathcal{C}$ as follows:

$$\mathcal{N}x : \beta.\phi \equiv \mathcal{C}x : \langle\bot, \beta\rangle.\phi. \tag{2}$$

That is, for $\mathcal{C}$, we can set the time interval to $[0, +\infty)$ and disable the reset condition to achieve the same semantics with $\mathcal{N}$. In this case, the time interval constraints in the semantic definition will not have any effect.

By Definition 1, we have

$$\begin{aligned}\rho, \tau, v, i \models \mathcal{C}x : \langle\bot, \beta\rangle.\phi \text{ iff } &\rho, \tau, v[x \mapsto n], i \models \phi, \text{ where}\\ &n = |\{j \mid \rho, \tau, v, j \models \beta \text{ and } m < j \le i\}| \text{ and}\\ &m = max(\{j \mid \rho, \tau, v, j \models \bot, 1 \le j \le i\} \cup \{0\}).\end{aligned}$$

Since $\{j \mid \rho, \tau, v, j \models \bot, 1 \le j \le i\} = \emptyset$, we have $m = 0$. Thus,

$$\begin{aligned}\rho, \tau, v, i \models \mathcal{C}x : \langle\bot, \beta\rangle.\phi \text{ iff } &\rho, \tau, v[x \mapsto n], i \models \phi,\\ &\text{where } n = |\{j \mid \rho, \tau, v, j \models \beta \text{ and } 0 < j \le i\}|.\end{aligned}$$

And with a little adjustment, $n = |\{j \mid \rho, \tau, v, j \models \psi \text{ and } 1 \le j \le i\}|$. This exactly follows the semantic definition of $\mathcal{N}$.

Conversely, $\mathcal{C}$ can also be encoded with $\mathcal{N}$ as follows:

$$\mathcal{C}x : \langle\alpha, \beta\rangle.\phi \ \equiv\ \mathcal{N}z : \alpha.\ \mathcal{N}x : (\beta \wedge \mathcal{N}y : \alpha.y = z).\ \phi. \tag{3}$$

Note that the subformula $\mathcal{N}y : \alpha.y = z$ actually acts as a counter reset condition. According to the definition of $\mathcal{N}$, we have

$$\begin{aligned}&\rho, v, i \models \mathcal{N}z : \alpha.\ \mathcal{N}x : (\beta \wedge \mathcal{N}y : \alpha.y = z).\ \phi\\ &\text{iff } \rho, v[z \mapsto n_1, x \mapsto n_2], i \models \phi, \text{ where}\\ &\quad n_1 = |\{h \mid \rho, v, h \models \alpha, 1 \le h \le i\}| \text{ and}\\ &\quad n_2 = |\{j \mid \rho, v, j \models \beta \text{ and } \rho, v, j \models \mathcal{N}y : \alpha.y = n_1, 1 \le j \le i\}|.\end{aligned} \tag{4}$$

Let $m_j = |\{k \mid \rho, v, k \models \alpha, 1 \le k \le j\}|$. Then n_2 can be equivalently stated as

$$\begin{aligned}n_2 = &|\{j \mid \rho, v, j \models \beta, \rho, v[y \mapsto n_3], j \models (m_j = n_1),\\ &\qquad and\ 1 \le j \le i\}|.\end{aligned}$$

We define $m = max(\{h \mid \rho, v, h \models \alpha, 1 \le h \le i\} \cup \{0\})$, and given the same trace, if $m_j = n_1$ with $j \le i$, i.e.,

$$|\{k \mid \rho, v, k \models \alpha, 1 \le k \le j\}| = |\{h \mid \rho, v, h \models \alpha, 1 \le h \le i\}|,$$

obviously, we can get $j \ge m$.

Because with the definition of m, we know

$$\begin{aligned}&\{h \mid \rho, v, h \models \alpha, m < h \le i\} = \emptyset,\\ &\{h \mid \rho, v, h \models \alpha, 1 \le h \le i\} = \{h \mid \rho, v, h \models \alpha, 1 \le h \le m\}.\end{aligned}$$

Given $j \geq m$, since $j \leq i$, we can also get

$$\begin{aligned}&\{k \mid \rho, \nu, k \models \alpha, 1 \leq k \leq j\}\\ &= \{k \mid \rho, \nu, k \models \alpha, 1 \leq k \leq m\} \cup \{k \mid \rho, \nu, k \models \alpha, m < k \leq j\}\\ &= \{k \mid \rho, \nu, k \models \alpha, 1 \leq k \leq m\} \cup \emptyset\\ &= \{k \mid \rho, \nu, k \models \alpha, 1 \leq k \leq m\}.\end{aligned} \tag{5}$$

That is $m_j = n_1$.

Therefore, $j \geq m$ is equivalent to the condition $m_j = n_1$. Thus,

$$\begin{aligned}&\rho, \nu, i \models \mathcal{N}z : \alpha.\ \mathcal{N}x : (\beta \wedge \mathcal{N}y : \alpha.y = z).\ \phi\\ &\equiv \rho, \nu[x \mapsto n_2], i \models \phi \ with\ n_2 = |\{j \mid \rho, \nu, j \models \beta\ and\ m \leq j \leq i\}|,\end{aligned}$$

which equals to $\mathcal{C}x : \langle \alpha, \beta \rangle.\phi$.

Finally, the proposition follows. □

Although $\mathcal{C}$ can be encoded using $\mathcal{N}$ as in Equation (3), it introduces nested occurrences of $\mathcal{N}$ and one needs to compare at least two counting variables. In general, policies involving two or more counting variables are hard or impossible to be monitored in a trace-length independent way, like $x < y$, where we need to keep track of the difference between two variables. We could have simply used $\mathcal{N}$, but we would then have to use the encoding above to compare two or more variables. Such encodings would thus obscure the underlying structure of the problem, and makes it harder to systematically generate TLI monitors from a given specification. For instance, the policy described in Example 1 uses only one counting variable when expressed using $\mathcal{C}$, and results in Section 4 would guarantee the existence of TLI monitors for that particular policy. Had we chosen to encode it using $\mathcal{N}$, we would have to make more efforts in order to show that the policy is in fact TLI monitorable. Moreover, the counting quantifier we propose is capable of specifying metric temporal counting policies while $\mathcal{N}$ fails to. We also mention that counting operators defined in [7], [14] are encountered with the same problem of complicating the encoding, though they are able to specify metric temporal constraints.

The quantifier $\mathcal{C}$ achieves equivalent expressiveness but with a more concise representation owing to the reset component. In [16], a similar design of counting operator with reset is given but under the non-metric semantics. To the best of our knowledge, $\mathcal{C}$ is the first metric counting quantifier with an elegant and beneficial design of reset condition, especially for the study of TLI monitoring algorithms.

3 Preliminaries

3.1 Trace-Length Independent Monitoring

Under the scenario of runtime monitoring, we assume a runtime monitor runs in parallel with the system to be monitored, and the monitor has access to certain events emitted by the system. The intention is to monitor the *violation* of specified properties at every state of the trace. Whenever a new event arrives, the monitor would check whether the required property is violated in the trace resulting from extending the history with the new event. Possibly, it needs to keep the entire trace to check the satisfaction.

Any algorithm that implements the satisfiability relation in Definition 1 can be used as a monitoring algorithm. In general, an algorithm that checks the satisfiability relation $(\rho, \tau, \nu, i \models \psi)$ would need to walk through the event trace ρ from the beginning up to the world i. In a setting with limited storage and computation resource (e.g., an OS kernel or embedded devices), a runtime monitoring algorithm that requires storage and inspection of an entire event trace may not be practical, even if its complexity is linear in the size of the trace, since the performance of the monitor will degrade as the event trace grows. The dependency on the size of the policies (i.e., formulas) is not so much of a problem in practice, since the policies are usually fixed for a given system, or at least updated less frequently compared to event traces. Ad hoc restrictions such as limiting the time window or enforcing bounded storage of events are not desirable as they may introduce incompleteness with respect to the policies being monitored, i.e., there may be violations to the policies that can only be detected on a trace of events longer than what could fit in the storage. In early work [10], monitoring algorithms are designed so that they require constant memory and take linear time to the size of the trace for a given formula to report the satisfiability relation in each world. Their algorithm is complete with formulas specified in ptLTL, and one needs to maintain only two states of all the subformulas of a policy. Such monitoring algorithms are termed as *trace-length independent* monitoring algorithms in [9]:

Definition 3. *A monitoring algorithm $\mathcal{M}$ is trace-length independent for a specification language $\mathcal{L}$ if for any trace and formula specified in $\mathcal{L}$, the interpretation for all worlds can be done in linear time in the size of trace, with its constant depending only on the size of formula, and with constant memory space which also depends only on the size of formula.*

In designing the algorithm, trace-length independence is achieved when only a bounded history is required for checking at each state, i.e., in a recursive paradigm. Formally, there exists a fixed k such that the interpretation of a formula in current world i depends only on the interpretations of its strict subformulas in the same world, or the truth values of its subformulas in at most k previous worlds. i.e., the interpretations in the following set:

$$\begin{aligned}&\{i - j \vdash \varphi \mid \varphi \text{ is a subformula of } \psi \text{ and } 1 \leq j \leq k\}\\ &\bigcup \{i \vdash \varphi \mid \varphi \text{ is a } \textit{strict} \text{ subformula of } \psi.\}.\end{aligned}$$

For example, the TLI monitoring algorithm for ptLTL in [10] follows such routine, with $k = 1$.

In a monitoring scenario where one needs to process events one at a time as they arrive, a TLI algorithm has the advantage that it does not need to store the entire trace of events seen so far. Rather, it only has to remember the truth values of every subformulas of the policy up to at most k previous worlds.

3.2 Trace-Length Independent Monitoring of ptMTL

Beyond ptLTL, TLI monitoring algorithms for its metric extension, ptMTL, have also been investigated in [8], [11]. Before exploring the TLI monitoring algorithm for MTL_{cnt}, we first prepare the readers with basic ideas on the TLI monitoring of the vanilla ptMTL. Recursive definitions of satisfiability relations typically lead to efficient dynamic

programming based algorithms for checking membership of a trace in the set of traces defined by a formula [17]. For the TLI-monitoring of ptMTL, the core problem is to devise a recursive semantics for the metric *since* operator, and it is first investigated and solved in [11]

$$i \models \phi_1 \mathcal{S}_I \phi_2 \text{ iff } 0 \in I \text{ and } i \models \phi_2, \text{ or}$$
$$i > 1, i \models \phi_1 \text{ and } i \models \phi_1 \mathcal{S}_{I'} \phi_2 \text{ where } I' = I - (\tau_i - \tau_{i-1}).$$

As can be seen that the recursive semantics is dependent on the interpretation of a new formula with the same structure, but a different time interval, which shifts by the difference between the current and the previous timestamps. In a general setting, difference between two successive timestamps (e.g., τ_i and τ_{i-1}) in a time sequence varies, so we need to simultaneously monitor the set of formulas arising from all possible interval shifts. Such formulas are termed as *"interval-skewed subformulas"* in [18]. Let $\mathcal{SF}(\phi)$ denote the set of subformulas of ϕ that are defined in a usual manner, the set of its interval-skewed subformulas is defined as

$$\mathcal{SF}^k(\phi) = \{\phi_1 \mathcal{S}_{I-n} \phi_2 \,|\, \phi_1 \mathcal{S}_I \phi_2 \in \mathcal{SF}(\phi), n \in [1, max(I)]\}.$$

Example 3 is given in Appendix A to demonstrate the interval-skewed subformula concept.

We denote the expanded set of subformulas with $\mathcal{SF}^+(\phi)$, and define it as

$$\mathcal{SF}^+(\phi) = \mathcal{SF}(\phi) \cup \mathcal{SF}^k(\phi).$$

For metric logic, timestamps can be infinitely large and recording any of them will interfere with the TLI property. As this is a characteristic of the logic itself and beyond our topic, following [8] we assume that timestamps can be represented by bounded number of bits. In practice, this assumption is often reasonable since arbitrarily precise clocks do not exist physically. We stick to the assumption in this article, not only to restrain bound space consumption, but also guarantee calculations on timestamps to be carried out in constant time. Our monitoring algorithm is extended from the one presented in [11].

3.3 Isolated Variables

For a counting formula, e.g., $\mathcal{C}_I x : \langle \alpha, \beta \rangle . \phi$, the constraint part ϕ may consist of multiple complicated components, like arithmetic relations, *since* operators, counting operators, conjunction operators and etc, thus x can be used in more than one arithmetic relations. In this case, these relations must be considered as a whole and complicate the analysis. To ease the exposition and make the description more succinct, we define isolated variables as in Definition 4.

Definition 4. *Given a counting formula $\mathcal{C}_I x : \langle \alpha, \beta \rangle . \phi$, if variable x is used in exactly one arithmetic relation, we call it an isolated variable.*

In reality, this is not a limitation to the formulas. Particularly, if there is no variable in ϕ, the counting formula is logically equivalent to ϕ. In case ϕ is with more than one arithmetic relations involving x, it can be rewritten into an equivalent formula where x is isolated as shown in Lemma 1.

Lemma 1. *Given a counting formula $\mathcal{C}_I x : \langle \alpha, \beta \rangle . \phi$ with x occurring in more than one arithmetic relations, it can be encoded to an equivalent formula where all variables are isolated.*

Proof. According to the syntax of MTL$_{cnt}$, all direct occurrences of variables are in arithmetic relations, and we assume x is used in terms $t_1, \ldots, t_m$. The rewriting is done as follows:

1) We substitute the quantifier $\mathcal{C}_I x : \langle \alpha, \beta \rangle$ with $\mathcal{C}_I x_1 : \langle \alpha, \beta \rangle \ldots \mathcal{C}_I x_m : \langle \alpha, \beta \rangle$.
2) For t_j with $1 \leq j \leq m$, x is substituted with x_j.

With the rewriting, it is guaranteed any x_j $(1 \leq j \leq m)$ occurs in only one term. More importantly, formulas before and after the rewriting are equivalent because the quantification conditions—target condition, trigger condition and time interval—for x and its substitutions are the same, thus keeping the same valuation for these variables, and further the same valuation for the terms. □

With Lemma 1, we can assume without loss of generality that all the variables in MTL$_{cnt}$ formulas are isolated and restricted by a particular counting quantifier. Furthermore, given an execution trace, the valuation of a variable is determined by the counting quantifier, and this is different from the traditional usage of first-order quantifiers. From this point of view, we do not deem MTL$_{cnt}$ as an extension of the first-order temporal logic, where variables always have alternative valuations.

4 Trace-Length Independent Monitoring of MTL$_{cnt}$

With the extension of counting quantifier and arithmetic relation, MTL$_{cnt}$ is more expressive and designing a TLI monitoring algorithm for it is desirable but challenging. From experiences on TLI monitoring of other past time temporal logics, it is profitable to explore the recursive semantics of the counting formula. Afterwards, algorithms can be designed accordingly. In the following, all variables are required to be bound, which means for any occurrence of an arithmetic relation in a formula, we can always identify the corresponding counting quantifiers. For instance, if a relation $t(x_1, \ldots, x_n) > 0$ occurs as a subformula, then there must exist quantifiers $\mathcal{C}_{I_1} x_1 : \langle \alpha_1, \beta_1 \rangle, \ldots, \mathcal{C}_{I_n} x_n : \langle \alpha_n, \beta_n \rangle$ restricting the valuation of its variables. Moreover, we assume all variables are isolated unless otherwise stated.

There are two parts in the semantics of counting formula: the valuation of the variable and the interpretation of the arithmetic relation under this valuation. In runtime monitoring, the valuation of a variable is calculated via interpreting the counting quantifier against the execution trace. For non-metric counting quantifiers (i.e., those with interval $[0, +\infty)$), the number of times a formula is satisfied in the history can never decrease, unless the reset condition is triggered, which means $t^{v[x \mapsto d]} > 0$ is always evaluated in worlds behind $t^{v[x \mapsto (d-1)]} > 0$. This statement is still true for metric temporal counting quantifiers when interpreted on the *absolute* time intervals, and details can be find in Section 4.4. This observation is the key inspiration for us to unveil the recursive semantics of counting formulas. Here we need to mention that recording the actual count and

updating it according to the coming event would break the TLI property as the space needed grows. On the other hand, storing partial history is also not practical, as we do not specify any upper bound of the number of events that can emerge within a time period. As a result, we focus our attention on how to interpret the arithmetic relations recursively, assuming the counts grow irreversibly unless reset.

In this section, we first identify the class of arithmetic relations that are TLI-monitorable and then give the recursive semantics for counting formulas with such arithmetic relations. Specifically, in Section 4.1 we look at the univariate case where relations are with arity 1, and then generalize it to the multivariate case in Section 4.2. If all relations in a counting formula are TLI-monitorable, it is straightforward that formula itself is TLI-monitorable. For such TLI-monitorable counting formulas, we develop the recursive semantics to interpret the counting quantifiers and derive the valuation of the formulas based on results from bounded previous worlds. In Section 4.3, we present the recursive description of the semantics of TLI-monitorable counting formulas with non-metric counting quantifiers, and make a generalization to metric temporal counting quantifiers in Section 4.4.

4.1 TLI-Monitorable Arithmetic Relations

Intuitively, TLI-monitorable arithmetic relations are those for which the valuation of $t(x_1, \ldots, x_n) > 0$ can be solved incrementally as events keep coming. In this subsection, we look into univariate relations, i.e., $t(x) > 0$, and leave the multivariate cases to the next subsection. If a variable x is quantified with the time interval $[0, +\infty)$, we can infer that whenever an event occurs, the valuation of x either remains the same or increases by 1, or is cleared to 0 if the reset condition is triggered. This means the interpretation of $t^{v[x \mapsto (d-1)]} > 0$ is always calculated ahead of the interpretation of $t^{v[x \mapsto d]} > 0$ for any count number d. For counting quantifiers with general time intervals, we also observe such an interpretation order, after some transformation on the time intervals, as we shall discuss Section 4.4. If we can determine the truth values of $t^{v[x \mapsto d]} > 0$ based on a finite number of interpretations of $t^{v[x \mapsto d-1]} > 0, \ldots, t^{v[x \mapsto d-k]} > 0$, then it only needs to maintain the previous k status of this interpretation. Moreover, there is no need to store the actual value of the variable x nor the value of $t(x)$ during the monitoring; all that matters is the truth value of $t(x) > 0$. Thus the space required for monitoring such relations remains constant irrespective of the value of x during the monitoring. The formal definition of TLI-monitorable relations is given in Definition 5. In this definition, we do not require the valuation of $t(x) > 0$ to be recursive from 0 because it is when x becomes large that the constant space restriction is hard to fulfill.

Definition 5. *Given a term $t : \mathbb{N} \to \mathbb{R}, t > 0$ is* TLI-monitorable *if there are two constants $c, k \in \mathbb{N}$, with $c \geq k \geq 1$, and a total computable boolean function H, such that for any $x \geq c$*

$$f_t(x) = H(f_t(x-1), \ldots, f_t(x-k)).$$

Definition 6. *Let $t : \mathbb{N}^n \to \mathbb{R}$. We define its characteristic function f_t as follows:*

$$f_t(x_1, \ldots, x_n) = \begin{cases} 0 & \text{if } t(x_1, \ldots, x_n) \leq 0, \\ 1 & \text{otherwise.} \end{cases}$$

Definition 7. *Given a function $f : \mathbb{N} \to \mathbb{R}$, it is said to be* periodic *over interval $I \subseteq \mathbb{N}$ with period $T \in \mathbb{N}_{>0}$ if we have*

$$f(x) = f(x+T),$$

for all values of $x \in I$, such that $(x+T) \in I$.

Definition 8. *A total function $f : \mathbb{N} \to \mathbb{R}$ is said to be* lower-bounded periodic, *or* lb-periodic *for short, if there is a $b \in \mathbb{N}$ such that f is periodic on interval $[b, +\infty)$.*

Lemma 2. *Given a term $t : \mathbb{N} \to \mathbb{R}, t > 0$ is TLI-monitorable if f_t is lb-periodic.*

Proof. Since f_t is lb-periodic, there is a b such that f_t is periodic on $[b, +\infty)$. Suppose the period is k, and then for any $x \geq b + k$ we have $f_t(x) = f_t(x-k)$. So,

$$f_t(x) = \begin{cases} t(x) > 0 & \text{if } x < b+k, \\ f_t(x-k) & \text{otherwise.} \end{cases}$$

Hence, the boolean function H can be easily constructed as: for $x \geq b + k$,

$$H(f_t(x-1), \ldots, f_t(x-k)) = f_t(x-k).$$

Obviously, H is a standard primitive recursive projection function and according to Definition 5, $t > 0$ is TLI-monitorable. □

We characterize the class of relations which are TLI-monitorable according to Definition 5. Some auxiliary definitions are presented in Definitions 6, 7, and 8 to define the characteristic function and to describe the *lb-periodic* property of a function. These definitions help to specify the characterizations of TLI-monitorable relations. The sufficiency condition for a relation to be TLI-monitorable is shown and proved in Lemma 2. Example 2 in Appendix A is presented to illustrate the definitions and lemma above.

In the next, we present in Lemma 3 the necessary condition for a relation to be TLI-monitorable and give the proofs.

Lemma 3. *Given a term $t : \mathbb{N} \to \mathbb{R}$, f_t is lb-periodic if $t > 0$ is TLI-monitorable.*

Proof. As f_t is TLI-monitorable, there exist two constants c and k, with $c \geq k \geq 1$ and $c, k \in \mathbb{N}$, such that

$$f_t(x) = \begin{cases} t(x) > 0 & \text{if } x < c, \\ H(f_t(x-1), \ldots, f_t(x-k)) & \text{otherwise.} \end{cases}$$

for some given funciton H.

We denote the value of $f_t(x)$ as a_x for $x \geq c$, and then we can get a string $a = a_c a_{c+1} a_{c+2} \cdots$ with infinite length. We define a subsequence of a as $A = a a_{i+1} \cdots a_{i+k-1}$, with $i \geq c$ and $i \in \mathbb{N}$. A is of length k. It is obvious that A is composed of only 0 and 1, so $A \in \{0,1\}^k$.

Now we can get an infinite list of sequences $A_c, A_{c+1}, A_{c+2}, \ldots$. For the first $(2^k + 1)$ sequences, there must be at least one pair of identical sequences according to the *Pigeonhole Principle*. We pick out the first sequence A_p which has a repeat, and then there must exist $q > p$ so that $A_p = A_q$; and for any $p < t < q$, $A_p \neq A_t$. More explicitly, $A_p = A_q$ means that $a_p = a_q, \ldots, a_{p+k-1} = a_{q+k-1}$.

With some adaption on representation, we have $a_x = H(a_{x-1}, \ldots, a_{x-k})$, then it can be inferred that

$$H(a_{p+k-1}, \ldots, a_p) = H(a_{q+k-1}, \ldots, a_q) \Rightarrow a_{p+k} = a_{q+k}.$$

Along with $a_{p+k} = a_{q+k}$, we get

$$H(a_{p+k}, \ldots, a_{p+1}) = H(a_{q+k}, \ldots, a_{q+1}) \Rightarrow a_{p+k+1} = a_{q+k+1},$$

and so on and so forth.

Via the deduction we get the conclusion that for $x \geq q$, $a_x = a_{x-(q-p)}$, i.e., $f_t(x) = f_t(x-(q-p))$. That is, $f_t(x) = f_t(x-(q-p))$ on $[q, +\infty)$. Therefore, f_t is lb-periodic on $[p, +\infty)$, and this lemma follows. □

With the sufficient and necessary conditions proved in Lemmas 2 and 3 separately, we get the following theorem:

Theorem 1. *Given a term $t : \mathbb{N} \to \mathbb{R}$, $t > 0$ is TLI-monitorable iff f_t is lb-periodic.*

The abstract characterization in Theorem 1 is in a way quite obvious from the definition of TLI-monitorable relations. The important part is that monitorability is associated with periodic characteristic function f_t rather than the term t itself. That is, t may not be periodic yet still be TLI-monitorable. In concrete applications, since we are usually only given the term t but not the characteristic function f_t, the difficulty is in deciding whether f_t is lb-periodic. In the following, we show several classes of functions for which their characteristic functions are lb-periodic.

Theorem 2. *Given a term $t : \mathbb{N} \to \mathbb{R}$, $t > 0$ is TLI-monitorable if t satisfies one of the following conditions:*

1) *t is lb-periodic.*
2) *t is monotonically increasing/decreasing.*
3) *t is a univariate polynomial function.*

Proof. We show the proofs for each case.

Case(1): Supposing t is lb-periodic on interval $[b, +\infty)$ with period T, we have, for $\forall m \in [b, +\infty)$, $t(m) = t(m+T)$. If $t(m) \leq 0$, naturally $t(m+T) \leq 0$, and then

$$f_t(m) = f_t(m+T) = 0.$$

Else if $t(m) > 0$, it can be inferred that $t(m+T) > 0$ and

$$f_t(m) = f_t(m+T) = 1.$$

Therefore, $f_t(x)$ is lb-periodic on interval $[b, +\infty)$. According to Theorem 1, $t > 0$ is TLI-monitorable.

Case(2): We first consider the scenario where term t is monotonically increasing. If $t(x) \leq 0$ for $\forall m \in \mathbb{N}$, then $f_t(x)$ remains 0, i.e., it is lb-periodic on $\mathbb{N}$ with period 1. Else, if $\exists m_0 \in \mathbb{N}$, we have $t(x) > 0$. Then for any $m > m_0$, we have $t(m) > t(m_0) > 0$, that is $f_t(x)$ remains 1 on $[m_0, +\infty)$. According to Theorem 1, $t > 0$ is TLI-monitorable.

In the other case F is monotonically decreasing, with similar proofs we can get to the same results.

Case(3): Suppose t is a univariate polynomial of degree n and $t(x)$ has at most n roots. If $t(x)$ has no roots, according to the Continuity of polynomial functions, t remains positive or negative on the domain. Else, we assume the roots are $c_1, \ldots, c_k$, for some $1 \leq k \leq n$. (Note that these roots are not necessary to be integers.) Let c be a positive integer larger than $\lceil max(c_1, \ldots, c_k) \rceil$. Since all polynomial functions are continuous, we have either $t(x) > 0$ for $\forall x \geq c$ or $t(x) < 0$ for $\forall x \geq c$. In either case, f_t is lb-periodic on $[c, \infty)$ with period 1, and therefore $t > 0$ is TLI-monitorable. □

There also exist univariate relations that are not TLI-monitorable, like $t(x) = sin\sqrt{x}$, for which the characteristic function is not lb-periodic.

4.2 TLI-Monitorable Multivariate Relations

We now look at the case where the arithmetic relations can be multivariate. The formal definition of TLI-monitorable multivariate relations is given in Definition 9. Theorem 3 implies that φ is TLI-monitorable if the period of a projection of F into one of its parameter is independent of the other parameter, once the value of that parameter exceeds a certain threshold. This allows us to quotient the values of each parameters into their own equivalence classes independently of each other.

Definition 9. *Let $t : \mathbb{N}^n \to \mathbb{R}$ be a term, then $t > 0$ is TLI-monitorable if there are constants $c_1, \ldots, c_n$ and $k_1, \ldots, k_n$, with $c_i \geq k_i \geq 1$ and $c_i, k_i \in \mathbb{N}$ for all $1 \leq i \leq n$, such that $f_t(x_1, \ldots, x_n) =$*

$$\begin{cases} t(x_1, \ldots, x_n) > 0, \text{ if } \forall i : x_i < c_i, \\ H(\{f_t(x_1 - u_1, \ldots, x_n - u_n) \mid \forall i : 0 \leq u_i \leq k_i\} \setminus \\ \qquad f_t(x_1, \ldots, x_n)), \text{otherwise.} \end{cases}$$

Here H is a total computable boolean function.

Definition 10. *Let $t : \mathbb{N}^n \to \mathbb{R}$, we define the set of the ith dimension slices of t as*

$$\Pi_i(t) = \{x \to t(a_1, \ldots, a_{i-1}, x_i, a_{i+1} \ldots, a_n) \mid a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n \in \mathbb{N}\}.$$

Theorem 3. *Given a term $t : \mathbb{N}^n \to \mathbb{R}$, for any $1 \leq i \leq n$, if the characteristic functions of functions in $\Pi_i(t)$ are all lb-periodic with the same lower bound and period, then $t > 0$ is TLI-monitorable.*

Proof. For any $1 \leq i \leq n$, we assume the characteristic functions of functions in $\Pi_i(t)$ are lb-periodic and share the same period k_i and lower bound c_i. If $x_i < k_i + c_i$ for any $1 \leq i \leq n$, then $t(x_1, \ldots, x_n)$ is calculated accordingly and get the interpretation of $t > 0$. Otherwise, supposing there exists j such that $x_j \geq k_j + c_j$ and $x_m < k_m + c_m$ for all $1 \leq m < j$, since the characteristic functions of functions in $\Pi_j(t)$ are lb-periodic, we have,

$$f_t(x_1, \ldots, x_j, \ldots, x_n) = f_t(x_1, \ldots, x_j - k_j, \ldots, x_n).$$

Following similar lines, function H can be constructed as $f_t(x_1, \ldots, x_n) =$

$$\begin{cases} t(x_1,\ldots,x_n) > 0 & \text{if } x_i < k_i + c_i \text{ for all } 1 \le i \le n, \\ f_t(x_1 - k_1,\ldots,x_n) & \text{if } x_1 \ge k_1 + c_1. \\ \cdots & \cdots \\ f_t(x_1,\ldots,x_j - k_j,\ldots,x_n) & \text{if } x_j \ge k_j + c_j \text{ and } x_i < k_i + c_i \\ & \qquad \text{for all } 1 \le i < j \le n. \\ \cdots & \cdots \\ f_t(x_1,\ldots,x_n - k_n) & \text{if } x_n \ge k_n + c_n \text{ and } x_i < k_i + c_i \\ & \qquad \text{for all } 1 \le i < n. \end{cases}$$

According to the Definition 9, $t > 0$ is TLI-monitorable.□

4.3 Recursive Semantics of Non-Metric Temporal Counting Quantifier

For a counting formula, the counting semantics comes from the counting quantifiers and the limitation semantics comes from the arithmetic relations. If the counting quantifier is with time interval $[0, +\infty]$, we call it non-metric temporal counting quantifier. In this section, we investigate how to interpret the counting formula with non-metric temporal counting quantifiers in a recursive way.

Recall (from Section 3.3) that all variables in a counting formula are isolated, which means each counting variable appears in exactly one term. We start the analysis with a particular arithmetic relation, and see how to translate the counting semantics for all its bound counting quantifiers. Generally, the count for a target formula never decreases unless the reset condition is triggered. As the count is finally checked in an arithmetic relation, if the arithmetic relation is TLI-monitorable, we can track the valuation recursively instead of keeping its real value which can be arbitrarily large. For a univariate TLI-monitorable relation $t(x) > 0$, its truth value repeats periodically when x exceeds the lower bound of its characteristic function, therefore we can quotient its domain based on the period to form a finite set of equivalence classes. As for multivariate relations, e.g., $t(x_1,\ldots,x_n) > 0$, surprisingly we can get the equivalence classes following similar line. The main difference between the univariate and the multivariate case is finding the right lower bound and the periods for a particular variable, then we can quotient one dimension of its domain to form a finite set of equivalence classes. The definition of equivalence classes can be found in Definition 11, and the number equivalence classes is finite according to Proposition 2.

Definition 11. *Given a term $t : \mathbb{N}^n \to \mathbb{R}$, and for any $1 \le i \le n$ the characteristic functions of functions in $\Pi_i(t)$ are lb-periodic over $[b_i, +\infty)$ with period T_i, we define the* equivalence class *on the ith dimension domain of t as*

$$[e] = \begin{cases} \{e\} & \text{if } e < b, \\ \{a \mid (a - b_i) \bmod T_i + b_i = e, a \ge b_i\} & \text{otherwise.} \end{cases}$$

Proposition 2. *For equivalence classes defined in Definition 11, $[e]$ is empty iff $e \ge b_i + T_i$.*

Proof. Please refer to Appendix A. □

Proposition 3. *For equivalence classes of the ith dimension defined in Definition 11, $t > 0$ has the same interpretation for elements in the same equivalence class.*

Proof. For the equivalence class $[e]$, and $\forall a \in [e]$:

(1) If $e < b_i$, since $[e] = \{e\}$, obviously, $a = e$ and $t^{\nu[x_i \mapsto a]} > 0 \equiv t^{\nu[x_i \mapsto e]} > 0$.

(2) If $e \ge b_i$,since

$$[e] = \{a \mid (a - b_i) \bmod T_i + b_i = e, a \ge b_i, a \in \mathbb{N}\},$$

thus we have $(a - b_i) \bmod T_i + b_i = e$, and with some equivalent transformation, we have

$$\begin{aligned} & (a - b_i) \bmod T_i + b_i = e \\ \Leftrightarrow & (a - b_i) \bmod T_i = e - b_i \\ \Leftrightarrow & a - b_i = (e - b_i) + m * T_i (\exists m \in \mathbb{N}) \\ \Leftrightarrow & a = e + m * T_i. \end{aligned}$$

Since the characteristic functions of functions in $\Pi_i(t)$ is lb-periodic over $[b_i, +\infty)$ with period T_i, we assume f'_t the characteristic function of the mapping

$$x \to t(x_1,\ldots,x_{i-1},x,x_{i+1},\ldots,x_n),$$

where $x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n$ can be instantiated by any valid valuation functions. Then we have

$$f'_t(a) = f'_t(e + m * T_i) = f'_t(e + (m-1) * T_i) = \cdots = f'_t(e).$$

That is, $t^{\nu[x_i \mapsto a]} > 0 \equiv t^{\nu[x_i \mapsto e]} > 0$.

Now, it is proved that for any $a \in [e]$, we have $t^{\nu[x_i \mapsto a]} > 0 \equiv t^{\nu[x_i \mapsto e]} > 0$. Thus the proposition follows. □

As proved in Proposition 3, the interpretation of $t > 0$ is the same for valuations in the same equivalence class; we can achieve a recursive semantics by managing the valuation of variables in t as indexes to equivalence classes. For simplicity, each equivalence class is indexed with the minimal element in the set. In this case the valuation of variables is limited to a finite range and make trace-length independence possible. The reset condition in the quantifier is orthogonal to the issue of quotienting the domain of corresponding variable; when it is satisfied, the count is reset to 0, i.e., the index is reset to 0. We write $\mathfrak{r}(\rho, \tau, \nu, i, \mathcal{C}_I x : \langle \alpha, \beta \rangle)$ to denote the index of the equivalence class indicated by the valuation x quantified by $\mathcal{C}_I x : \langle \alpha, \beta \rangle$, and the value is computed under model (ρ, τ, ν) and world i. When ρ, τ and ν are clear form the context, we also write as $\mathfrak{r}(i, \mathcal{C}_I x : \langle \alpha, \beta \rangle)$ for simplicity.

As in [3], [10], the key to achieve the trace-length independence property is to express the semantics of all logical operators in a recursive form, e.g., the interpretation of ϕ at state i is generated only based on the interpretation of its subformulas at current world and/or the interpretation of ϕ at world $i - 1$. Semantics of all operators in MTL_{cnt} except the counting formula is already in recursive form. Our aim is to calculate to which equivalence class the current valuation of the variable belongs based on the previous records. Theorem 4 shows that the semantics of a TLI-monitorable non-metric counting formula also admits a recursive form. For world 1, i.e., the base case, we uniformly give the semantics of general (temporal and non-temporal) counting quantifier in Theorem 5 (Section 4.4).

Theorem 4. *Given a model (ρ, τ, ν) and a counting formula $\mathcal{C}x : \langle \alpha, \beta \rangle.\phi(x)$, if every arithmetic relation in ϕ is TLI-monitorable, then for every $1 < j \le |\rho|$,*

$$j \models \mathcal{C}x : \langle \alpha, \beta \rangle.\phi(x) \text{ iff } j \models \phi^{v[x \mapsto \mathfrak{r}(j, \mathcal{C}x:\langle \alpha, \beta \rangle)]},$$

where for variable x, we assume the characteristic functions of slices on x are lb-periodic on $[b, +\infty]$ with period T, and the index of equivalence class is calculated via:

1) *If $j \models \alpha$, then $\mathfrak{r}(j, \mathcal{C}x : \langle \alpha, \beta \rangle) = 0$;*
2) *If $j \not\models \alpha$, and $j \models \beta, r < b$, where $r = \mathfrak{r}(j-1, \mathcal{C}x : \langle \alpha, \beta \rangle)$, then $\mathfrak{r}(j, \mathcal{C}x : \langle \alpha, \beta \rangle) = r + 1$;*
3) *If $j \not\models \alpha$, and $j \models \beta$, $r \geq b$, where $r = \mathfrak{r}(j-1, \mathcal{C}x : \langle \alpha, \beta \rangle)$, then $\mathfrak{r}(j, \mathcal{C}x : \langle \alpha, \beta \rangle) = (r + 1 - b) \bmod T + b$;*
4) *If $j \not\models \alpha$, and $j \not\models \beta$, then $\mathfrak{r}(j, \mathcal{C}x : \langle \alpha, \beta \rangle) = \mathfrak{r}(j-1, \mathcal{C}x : \langle \alpha, \beta \rangle)$.*

Proof. For the counting quantifier, based on whether the reset condition and target formula are satisfied at current stage, the valuation of x determined by $\mathcal{C}x : \langle \alpha, \beta \rangle$ is discussed under the following three scenarios with distinct conditions, among which the union is complete and the intersection of any two is empty:

(i) $j \models \alpha$;
(ii) $j \not\models \alpha$ and $j \models \beta$;
(iii) $j \not\models \alpha$ and $j \not\models \beta$.

Since we assume the counting formula is TLI-monitorable, x is used either in a TLI-monitorable univariate relation whose characteristic function is lb-periodic on $[b, +\infty)$ and with period T, or a TLI-monitorable multivariate relation for which the characteristic functions of slices on x are lb-periodic on $[b, +\infty)$ and with period T.

For a non-metric counting quantifier its semantics can be simplified as:

- $j \models \mathcal{C}x : \langle \alpha, \beta \rangle.\phi$ iff $v[x \mapsto l], j \models \phi$, where $l = |\{h \mid h \models \beta, m < h \leq j\}|, m = max(\{h \mid h \models \alpha, 1 \leq h \leq j\} \cup \{0\})$.

For $\mathcal{C}x : \langle \alpha, \beta \rangle$, we can define

$$\begin{aligned} \mathbb{B} &= \{h \mid h \models \beta, m < h \leq j\}, \\ \mathbb{A} &= \{h \mid h \models \alpha, 1 \leq h \leq j\}. \\ l &= |\mathbb{B}|, \\ m &= max(\mathbb{A} \cup \{0\}). \end{aligned}$$

Thus we only need to prove l always belongs to the equivalence class $[\mathfrak{r}(j, \mathcal{C}x : \langle \alpha, \beta \rangle)]$.

Under condition (i), if $j \models \alpha$, then $j \in \mathbb{A}$ and j is the largest value in $\mathbb{A}$. Naturally, $m = j$ and $\mathbb{B}$ is empty because there is no h greater than j. It follows that $l = 0$, and obviously $l \in [\mathfrak{r}(j, \mathcal{C}x : \langle \alpha, \beta \rangle)]$.

Under condition (ii), if $j \not\models \alpha$ and $j \models \beta$, it means that there is a new occurrence of β which should be updated to the count as the reset condition is not triggered. To deal with the update of the count, we need to check whether the index of the equivalence class to which the count belongs at the current state is beyond the lower bound. We define, at state $j-1$,

$$\begin{aligned} \mathbb{B}' &= \{h \mid h \models \beta, m' < h \leq j-1\}, \\ \mathbb{A}' &= \{h \mid h \models \alpha, 1 \leq h \leq j-1\}, \\ l' &= |\mathbb{B}'|, \\ m' &= max(\mathbb{A}' \cup \{0\}). \end{aligned}$$

With the condition, we can get

$$\begin{aligned} &j \not\models \alpha \Rightarrow j \notin \mathbb{A} \Rightarrow \mathbb{A} = \mathbb{A}' \Rightarrow m = m', \\ &j \models \beta \Rightarrow j \in \mathbb{B} \Rightarrow \mathbb{B} = \mathbb{B}' \cup \{j\}(\text{ with } m = m'). \end{aligned}$$

Since $j \notin \mathbb{B}'$, it can be implied that$|\mathbb{B}| = |\mathbb{B}'| + 1 \Rightarrow l = l' + 1$.

After getting the valuation l of x, we need to compute to which the equivalence class l belongs. If $l' < b$, we have $l' \in [l']$. It can also be inferred that $l' + 1 \leq b$, and also

$$\begin{aligned} &l' + 1 < b \Rightarrow l' + 1 \in [l' + 1]; \\ &l' + 1 = b \Rightarrow l' + 1 \in [(b - b) \bmod T + b] \Rightarrow l' + 1 \in [b]. \end{aligned}$$

So we can always get $l \in [l' + 1]$. With $r = \mathfrak{r}(j-1, \mathcal{C}x : \langle \alpha, \beta \rangle) = l'$, we have $\mathfrak{r}(j, \mathcal{C}x : \langle \alpha, \beta \rangle) = r + 1 = l' + 1$, that is $l \in [\mathfrak{r}(j, \mathcal{C}x : \langle \alpha, \beta \rangle)]$.

Otherwise, if $l' \geq b$, then $l' \in [(l' - b) \bmod T + b]$. Obviously, $l = l' + 1 \geq b$ and $l \in [(l - b) \bmod T + b]$. As $\mathfrak{r}(j, \mathcal{C}x : \langle \alpha, \beta \rangle) = ((l' - b) \bmod T + b + 1 - b) \bmod T + b$, if $T = 1$,

$$\mathfrak{r}(j, \mathcal{C}x : \langle \alpha, \beta \rangle) = b = (l - b) \bmod T + b,$$

thus we have $l \in [\mathfrak{r}(j, \mathcal{C}x : \langle \alpha, \beta \rangle)]$.

If $T > 1$, then

$$\begin{aligned} &((l' - b) \bmod T + b + 1 - b) \bmod T + b \\ =&((l' - b) \bmod T + 1) \bmod T + b \\ =&((l' - b) \bmod T + 1 \bmod T) \bmod T + b \\ =&(l' - b + 1) \bmod T + b \\ =&(l - b) \bmod T + b, \end{aligned}$$

that is $l \in [\mathfrak{r}(j, \mathcal{C}x : \langle \alpha, \beta \rangle)]$.

Under condition (iii), if $j \not\models \alpha$ and $j \not\models \beta$, then we know that $j \notin \mathbb{B}$ and $j \notin \mathbb{A}$, thus yielding

$$\begin{aligned} \mathbb{A} &= \{h \mid h \models \alpha, 1 \leq h \leq j-1\} = \mathbb{A}'. \\ m &= max(\mathbb{A} \cup \{0\}) = m', \\ \mathbb{B} &= \{h \mid h \models \beta, m < h \leq j-1\} = \mathbb{B}'. \end{aligned}$$

It follows that $l = l'$. Since $l' \in [\mathfrak{r}(j-1, \mathcal{C}x : \langle \alpha, \beta \rangle)]$, it is obviously $l \in [\mathfrak{r}(j, \mathcal{C}x : \langle \alpha, \beta \rangle)]$.

The theorem follows for cases (i) (ii) (iii). □

4.4 Recursive Semantics of Metric Temporal Counting Quantifier

In this section, we present the recursive semantics of counting quantifier bounded with general intervals. Since the time model used in this article is non-strictly increasing, which means there can be multiple occurrences of events with same timestamp, the count for certain formulas can be extremely large even in a short time interval. In this case, simply recording the exact count for metric counting quantifiers can still break the trace-length independence. Therefore, we need to seek for the recursive semantics of metric counting quantifiers with general time intervals in order to pave the way for TLI monitoring. We present the proposed formal definition in Theorem 6 and the base case (world 1) in Theorem 5. The key idea is to look at the time intervals in an absolute way. If there is a $\Delta\tau$ time shifting between the

current and previous timestamps, current observation of events within time interval I is consistent with previous observation in skewed interval $I - \Delta\tau$. In this way we can accumulate the count incrementally.

Theorem 5. *Given a model (ρ, τ, v) and a counting formula $\mathcal{C}_I x : \langle\alpha, \beta\rangle.\phi(x)$, at the world 1,*

$$1 \models \mathcal{C}_I x : \langle\alpha, \beta\rangle.\phi(x) \text{ iff } 1 \models \phi^{v[x \mapsto \mathfrak{r}(1, \mathcal{C}_I x:\langle\alpha,\beta\rangle)]},$$

where for variable x, we assume the characteristic functions of slices on x is lb-periodic on $[b, +\infty]$ with period T, then the index of equivalence class is calculated via:

1) *If $0 \notin I$, $\mathfrak{r}(1, \mathcal{C}_I x : \langle\alpha, \beta\rangle) = 0$;*
2) *If $0 \in I$,*
   a) *if $1 \models \alpha$, then $\mathfrak{r}(1, \mathcal{C}_I x : \langle\alpha, \beta\rangle) = 0$;*
   b) *if $1 \not\models \alpha$ and $1 \not\models \beta$, then $\mathfrak{r}(1, \mathcal{C}_I x : \langle\alpha, \beta\rangle) = 0$.*
   c) *if $1 \not\models \alpha$, $1 \models \beta$ and $b > 1$, then $\mathfrak{r}(1, \mathcal{C}_I x : \langle\alpha, \beta\rangle) = 1$;*
   d) *if $1 \not\models \alpha$, $1 \models \beta$ and $b \leq 1$, then $\mathfrak{r}(1, \mathcal{C}_I x : \langle\alpha, \beta\rangle) = (1 - b) \bmod T + b$.*

Proof. Please refer to Appendix A. □

Theorem 6. *Given a model (ρ, τ, v) and a counting formula $\mathcal{C}_I x : \langle\alpha, \beta\rangle.\phi(x)$, if every arithmetic relation in ϕ is TLI-monitorable, then for every $1 < j \leq |\rho|$,*

$$j \models \mathcal{C}_I x : \langle\alpha, \beta\rangle.\phi(x) \text{ iff } j \models \phi^{v[x \mapsto \mathfrak{r}(j, \mathcal{C}_I x:\langle\alpha,\beta\rangle)]}, \tag{6}$$

where for variable x, we assume the characteristic functions of slices on x is lb-periodic on $[b, +\infty]$ with period T, and $I' = I - (\tau_j - \tau_{j-1})$, then the index of equivalence class is calculated via:

1) *If $0 \notin I$ and $I' = \emptyset$, $\mathfrak{r}(j, \mathcal{C}_I x : \langle\alpha, \beta\rangle) = 0$;*
2) *If $0 \notin I$ and $I' \neq \emptyset$, $\mathfrak{r}(j, \mathcal{C}_I x : \langle\alpha, \beta\rangle) = \mathfrak{r}(j - 1, \mathcal{C}_{I'} x : \langle\alpha, \beta\rangle)$;*
3) *If $0 \in I$ and $I' = \emptyset$:*
   If $j \not\models \alpha$ and $j \models \beta$, $\mathfrak{r}(j, \mathcal{C}_I x : \langle\alpha, \beta\rangle) = 1$ if $b > 1$, and $\mathfrak{r}(j, \mathcal{C}_I x : \langle\alpha, \beta\rangle) = (1 - b) \bmod T + b$ if $b \leq 1$. Otherwise $\mathfrak{r}(j, \mathcal{C}_I x : \langle\alpha, \beta\rangle) = 0$;
4) *If $0 \in I$ and $I' \neq \emptyset$:*
   a) *if $j \models \alpha$, then $\mathfrak{r}(j, \mathcal{C}_I x : \langle\alpha, \beta\rangle) = 0$;*
   b) *if $j \not\models \alpha$ and $j \models \beta$ and $r = \mathfrak{r}(j - 1, \mathcal{C}_{I'} x : \langle\alpha, \beta\rangle) < b$, then $\mathfrak{r}(j, \mathcal{C}_I x : \langle\alpha, \beta\rangle) = r + 1$;*
   c) *if $j \not\models \alpha$ and $j \models \beta$ and $r = \mathfrak{r}(j - 1, \mathcal{C}_{I'} x : \langle\alpha, \beta\rangle) \geq b$, then $\mathfrak{r}(j, \mathcal{C}_I x : \langle\alpha, \beta\rangle) = (r + 1 - b) \bmod T + b$;*
   d) *if $j \not\models \alpha$ and $j \not\models \beta$, then $\mathfrak{r}(j, \mathcal{C}_I x : \langle\alpha, \beta\rangle) = \mathfrak{r}(j - 1, \mathcal{C}_{I'} x : \langle\alpha, \beta\rangle)$.*

Proof. In this proof, for each counting quantifier, the valuation of x determined by $\mathcal{C}_I x : \langle\alpha, \beta\rangle$ is analyzed under the four cases based on whether $0 \in I$ and I' is empty. Since we assume the counting formula is TLI-monitorable, x is used either in a TLI-monitorable univariate relation whose characteristic function is lb-periodic on $[b, +\infty)$ and with period T, or a TLI-monitorable multivariate relation for which the characteristic functions of slices on x are lb-periodic on $[b, +\infty)$ and with period T. According to the semantics of metric temporal counting quantifier defined in Section 2, for $\mathcal{C}_I x : \langle\alpha, \beta\rangle$ we denote

$$\begin{aligned}
\mathbb{B} &= \{h \mid h \models \beta, \text{ for } \tau_j - \tau_h \in I, m < h \leq j\},\\
\mathbb{A} &= \{h \mid h \models \alpha, \text{ for } \tau_j - \tau_h \in I, 1 \leq h \leq j\},\\
l &= |\mathbb{B}|,\\
m &= max(\mathbb{A} \cup \{0\}).
\end{aligned}$$

Thus we only need to prove that l always belongs to the equivalence class $[\mathfrak{r}(j, \mathcal{C}_I x : \langle\alpha, \beta\rangle)]$.

Case(1): If $0 \notin I$ and I' is empty,

$$\begin{aligned}
0 \notin I &\Rightarrow \tau_j - \tau_j \notin I \Rightarrow j \notin \mathbb{B},\\
I' = \emptyset &\Rightarrow (\tau_j - \tau_{j-1}) > max(I)\\
&\Rightarrow \forall h < j - 1, \tau_j - \tau_h > max(I)\\
&\Rightarrow \forall h < j - 1, \tau_j - \tau_h \notin I,
\end{aligned}$$

then we can conclude that $\mathbb{B} = \emptyset \Rightarrow l = 0$. Hence, $l \in [\mathfrak{r}(j, \mathcal{C}_I x : \langle\alpha, \beta\rangle)]$.

Case(2): If $0 \notin I$ and $I' \neq \emptyset$, for $\mathcal{C}_{I'} x : \langle\alpha, \beta\rangle$, we denote at state $j - 1$

$$\begin{aligned}
\mathbb{B}' &= \{h \mid h \models \beta, \text{ for } \tau_{j-1} - \tau_h \in I', m' < h \leq j - 1\},\\
\mathbb{A}' &= \{h \mid h \models \alpha, \text{ for } \tau_{j-1} - \tau_h \in I', 1 \leq h \leq j - 1\},\\
l' &= |\mathbb{B}'|,\\
m' &= max(\mathbb{A}' \cup \{0\}).
\end{aligned}$$

For $0 \notin I$, similar with Case(1), we know $j \notin \mathbb{A}$ and $j \notin \mathbb{B}$. Therefore, we can get

$$\begin{aligned}
\mathbb{A} &= \{h \mid h \models \alpha, \text{ for } \tau_j - \tau_h \in I, 1 \leq h \leq j - 1\}\\
&= \{h \mid h \models \alpha, \text{ for } \tau_j - \tau_h - (\tau_j - \tau_{j-1}) \in I - (\tau_j - \tau_{j-1}),\\
&\qquad 1 \leq h \leq j - 1\}\\
&= \{h \mid h \models \alpha, \text{ for } \tau_{j-1} - \tau_h \in I', 1 \leq h \leq j - 1\} = \mathbb{A}'\\
&\Rightarrow m = m',
\end{aligned}$$

$$\begin{aligned}
\mathbb{B} &= \{h \mid h \models \beta, \text{ for } \tau_j - \tau_h \in I, m < h \leq j - 1\}\\
&= \{h \mid h \models \beta, \text{ for } \tau_j - \tau_h - (\tau_j - \tau_{j-1}) \in I - (\tau_j - \tau_{j-1}),\\
&\qquad m < h \leq j - 1\}\\
&= \{h \mid h \models \beta, \text{ for } \tau_{j-1} - \tau_h \in I', m < h \leq j - 1\}\\
&= \{h \mid h \models \beta, \text{ for } \tau_{j-1} - \tau_h \in I', m' < h \leq j - 1\} = \mathbb{B}'\\
&\qquad (m = m')\\
&\Rightarrow l = l'.
\end{aligned}$$

Hence, $l \in [\mathfrak{r}(j, \mathcal{C}_I x : \langle\alpha, \beta\rangle)]$.

Case(3): If $0 \in I$ and $I' = \emptyset$, similar with Case(1),

$$I' = \emptyset \Rightarrow \forall h \in [1, j), h \notin \mathbb{B}, h \notin \mathbb{A}.$$

Thus the only possible member for $\mathbb{B}$ and $\mathbb{A}$ is j because $\tau_j - \tau_j = 0 \in I$, and further $l \leq 1$. Easily, we can get

$$\begin{aligned}
l = 1 \Leftrightarrow \mathbb{B} = \{j\} &\Leftrightarrow j \models \beta \text{ and } m < j \Leftrightarrow j \models \beta \text{ and } j \notin \mathbb{A}\\
&\Leftrightarrow j \models \beta \text{ and } j \not\models \alpha.
\end{aligned}$$

Hence, if $j \models \beta$ and $j \not\models \alpha$, then $l = 1$, and naturally $l \in [\mathfrak{r}(j, \mathcal{C}_I x : \langle\alpha, \beta\rangle)]$ no matter $b > l$ or $b \leq l$. Otherwise, $l = 0$, and also $l \in [\mathfrak{r}(j, \mathcal{C}_I x : \langle\alpha, \beta\rangle)]$.

Case(4): If $0 \in I$ and $I' \neq \emptyset$, following the proof of Theorem 4, cases are analyzed based on the satisfaction of $j \models \beta$ and $j \models \alpha$.

For case(a),

$$j \models \phi \Rightarrow j \in \mathbb{A} \Rightarrow m = j(0 \in I) \Rightarrow \mathbb{B} = \emptyset \Rightarrow l = 0.$$

Thus, $l \in [\mathfrak{r}(j, \mathcal{C}_I x : \langle \alpha, \beta \rangle)]$.

For case(d), similar with case(2),

$$\begin{aligned} j \not\models \alpha &\Rightarrow j \notin \mathbb{A} \\ &\Rightarrow \mathbb{A} = \{h \mid h \models \alpha, \text{ for } \tau_j - \tau_h \in I, 1 \leq h \leq j-1\} = \mathbb{A}' \\ &\Rightarrow m = m', \end{aligned}$$

and

$$\begin{aligned} j \not\models \beta &\Rightarrow j \notin \mathbb{B} \\ &\Rightarrow \mathbb{B} = \{h \mid h \models \beta, \text{ for } \tau_j - \tau_h \in I, m < h \leq j-1\} = \mathbb{B}' \\ &\Rightarrow l = l'. \end{aligned}$$

Hence, $l \in [\mathfrak{r}(j, \mathcal{C}_I x : \langle \alpha, \beta \rangle)]$.

For case(b) and case(c), it is easy to get $j \not\models \alpha \Rightarrow m = m'$, and

$$\begin{aligned} j \models \beta &\Rightarrow j \in \mathbb{B}(0 \in I) \\ &\Rightarrow \mathbb{B} = \{h \mid h \models \beta, \text{ for } \tau_j - \tau_h \in I, m < h \leq j-1\} \cup \{j\} \\ &\qquad = \mathbb{B}' \cup \{j\} \\ &\Rightarrow l = l' + 1. \end{aligned}$$

This is the same case as shown in the proof of Theorem 4 for condition (ii), so we do not repeat here.

In conclusion, the theorem follows. □

5 Monitoring Algorithm of MTL$_{cnt}$

Now each logical operators of MTL$_{cnt}$ can be handled recursively, and dynamic programming can be employed to design the TLI monitoring algorithm dealing with formulas composed by these operators. A key for the monitoring is to guarantee subformulas are always interpreted ahead of its root formula. Hence, the set of subformulas of a given MTL$_{cnt}$ formula must be formulated and organized in a well-order. For quantified counting formulas, however, its subformulas would contain free variables, and their truth values would thus depend on the valuation of variables. To avoid encoding valuation of variables explicitly in the monitoring algorithm, we need to instantiate variables to concrete values before computing their interpretation. Given a TLI-monitorable formula $\mathcal{C}_I x : \langle \alpha, \beta \rangle . \phi(x)$, we compute a finite set of equivalence classes for the domain, particularly on the dimension of x. Suppose this set has n elements $\{[e_1], \ldots, [e_n]\}$. We define its immediate subformulas as: α, β, and $\phi^{v[x \mapsto e_i]}$ for $1 \leq i \leq n$. Given a formula ψ, we use $\mathcal{SF}^*(\psi)$ to denote the extension of $\mathcal{SF}^+(\psi)$ to include the instantiated subformulas of counting formulas, and it is defined as

$$\begin{aligned} \mathcal{SF}^*(\psi) = \mathcal{SF}^+(\psi) \cup \{\phi^{v[x \mapsto e_i]} \mid \mathcal{C}_I x : \langle \alpha, \beta \rangle . \phi(x) \in \mathcal{SF} \\ \text{and } 1 \leq i \leq n\}. \end{aligned}$$

From the recursive semantics presented in Theorem 6, we can see the interpretation of the counting formulas depends not only on its immediate subformulas, but also the valuation of its interval-skewed counting quantifiers. We define the interval-skewed counting quantifiers of $\mathcal{C}_I x : \langle \alpha, \beta \rangle$ as

$$\mathcal{IS}_Q(\mathcal{C}_I x : \langle \alpha, \beta \rangle) = \{\mathcal{C}_{I-m} x : \langle \alpha, \beta \rangle \mid m \in [0, max(I)]\}.$$

These interval-skewed counting quantifiers are no included in the set of subformulas but their valuation of variables are maintained respectively. We define a well-order $\prec$ over the set of interval-skewed quantifiers that respects the following rule:

- if Q' is an interval-skewed counting quantifier of Q and $Q' \neq Q$, then $Q' \prec Q$.

And we assume the quantifiers in the set are always well ordered.

Now we present how monitoring can be done for a general MTL$_{cnt}$ formula ϕ, given a trace ρ and a time sequence τ, with $|\rho| = |\tau|$. Let $\phi_1, \phi_2, \ldots, \phi_m$ be an enumeration of $\mathcal{SF}^*(\phi)$ respecting the $\prec$ order. Simply, we deem ϕ_j $(1 \leq j \leq m)$ is with index j and the index can be referred to via $idx(\phi_j)$. We maintain two Boolean arrays $prev[1, \ldots, m]$ and $cur[1, \ldots, m]$ to record the interpretation of formulas in $\mathcal{SF}^*(\phi)$ at two successive states. Under the runtime monitoring scenario, the algorithm is required to output the interpretation of ϕ at each state. With dynamic programming approach, the monitoring algorithm outputs the verdict of all subformulas at a world following the $\prec$ order; and it outputs the verdict for all worlds incrementally from the beginning to the end. That is, with the results of world $i-1$ stored in $prev[1, \ldots, m]$, we can calculate the results of world i and keep them in $cur[1, \ldots, m]$. Array cur is updated with $prev$ at the start of next iteration, and the iteration stops when reaching the end of the trace.

For quantified counting subformulas, we assume they are with TLI-monitorable univariate arithmetic relations and/or TLI-monitorable multivariate relations as specified in Definition 9. In this case, we need to maintain the lower bound and period for each univariate relation and every slice of the multivariate relations. We also assume all the quantified subformulas in $\mathcal{SF}^*(\phi)$ to be variable-isolated. Since each variable occurs in exactly one relation, we use variable names to index the set of lower bounds and periods. These information is stored in array tli, whose size is equivalent to the number of variables in the formula. For variable x, the index can be referred via $idx_v(x)$, and its lower bound and period are $tli[idx_v(x)].lb$ and $tli[idx_v(x)].pd$. From the recursive semantics shown in Theorems 4 and 6, we know it is necessary to keep track of the valuation determined by counting quantifiers. For counting quantifiers in the subformulas, the corresponding valuations for the current and previous state are stored in arrays $c\_cnt[1, \ldots, l]$ and $p\_cnt[1, \ldots, l]$. Here l is equivalent to the number of interval-skewed quantifiers of all quantifiers in ϕ. Two counting quantifiers are deemed to be equivalent if they are with the same interval, variable, reset condition and target formula. For example, the valuation of $\mathcal{C}_I x : \langle \alpha, \beta \rangle$ can be accessed via $c\_cnt[idx_c(\mathcal{C}_I x : \langle \alpha, \beta \rangle)]/p\_cnt[idx_c(\mathcal{C}_I x : \langle \alpha, \beta \rangle)]$. Updating on the valuation arrays follows Theorems 4 and 6, and is done as a first step (see line 17-36 in Algorithm 3) when dealing with quantified subformulas.

The main monitoring algorithm (Algorithm 1) can be divided into two steps on the top level: the initialization procedure (Algorithm 2) and the for-loop calling iterative procedure (Algorithm 3). The initialization procedure works to initialize the cur and c_cnt arrays at the first state and the iterative procedure will help to maintain the arrays

as the state progressing. In the pseudocode of the algorithms, we overload some logical symbols to denote operators on boolean values. It is straightforward to see that, once the formula to be monitored is fixed, the space required to run the algorithm does not grow with the length of traces. Moreover, the correctness of our monitoring algorithm is stated in Theorem 7, with axillary Lemmas 4, 5, and 6. Proofs of them are included in Appendix A. The worst case time and space complexity of the monitoring algorithm is discussed and proved in Theorem 8.

Algorithm 1. $Monitor(\rho, \tau, \nu, i, \phi)$

1: Init($\rho, \nu, \phi, cur, c\_cnt$);
2: **for** $j = 2$ *to* i **do**
3: Iter($\rho, \tau, \nu, j, \phi, prev, cur, p\_cnt, c\_cnt$);
4: **return** $cur[idx(\phi)]$;

Algorithm 2. $Init(\rho, \nu, \phi, cur, c\_cnt)$

1: **for** $k = 1$ **to** m **do**
2: **switch** ϕ_k **do**
3: **case** $\bot$ **do** $cur[k] \leftarrow false$;
4: **case** p **do** $cur[k] \leftarrow p \in \rho_1$;
5: **case** $\neg\psi$ **do** $cur[k] \leftarrow \neg cur[idx(\psi)]$;
6: **case** $t > 0$ **do** $cur[k] \leftarrow t^\nu > 0$;
7: **case** $\psi_1 \vee \psi_2$ **do** $cur[k] \leftarrow cur[idx(\psi_1)] \vee cur[idx(\psi_2)]$;
8: **case** $\odot_I \psi$ **do** $cur[k] \leftarrow false$;
9: **case** $\psi_1 \mathcal{S}_I \psi_2$ **do** $cur[k] \leftarrow 0 \in I \wedge cur[idx(\psi_2)]$;
10: **case** $\mathcal{C}_I x : \langle \alpha, \beta \rangle . \varphi$ **do**
11: **for** $(Q = \mathcal{C}_{I'} x : \langle \alpha, \beta \rangle) \in \mathcal{IS}_{\mathcal{Q}}(\mathcal{C}_I x : \langle \alpha, \beta \rangle)$ **do**
12: **if** $!cur[idx(\alpha)] \wedge cur[idx(\beta)] \wedge 0 \in I'$ **then**
13: $lb \leftarrow tli[idx_v(x)].lowerBound$;
14: $pd \leftarrow tli[idx_v(x)].period$;
15: **if** $lb > 1$ **then** $c\_cnt[idx_c(Q)] \leftarrow 1$;
16: **else** $c\_cnt[idx_c(Q)] \leftarrow (1 - lb) \bmod pd + lb$;
17: **else** $c\_cnt[idx_c(Q)] \leftarrow 0$;
18: $cur[k] \leftarrow cur[idx(\varphi^{\nu[x \mapsto c\_cnt[idx_c(\mathcal{C}_I x:\langle\alpha,\beta\rangle)]]})$;
19: **return** $cur[idx(\phi)]$;

Lemma 4. *Given a model (ρ, τ, ν) with $(|\rho| = |\tau|) \geq 1$, after procedure $Init(\rho, \nu, \phi, cur, c\_cnt)$, for all $\phi_k \in \mathcal{SF}^*(\phi)$, we have $1 \models \phi_k$ iff $cur[k] \equiv$ true.*

Lemma 5. *Given a model (ρ, τ, ν) with $(|\rho| = |\tau|) \geq 1$, for all $i \geq 2$, if we assume $i - 1 \models \phi_k$ iff $cur[idx(\phi_k)] = true$ for any $\phi_k \in \mathcal{SF}^+(\phi)$, then after executing $Iter(\rho, \tau, \nu, i, \phi, prev, cur, p\_cnt, c\_cnt)$, for any $\phi_k \in \mathcal{SF}^*(\phi)$, we have $i \models \phi_k$ iff $cur[k] \equiv$ true.*

Lemma 6. *Given a model (ρ, τ, ν) with $(|\rho| = |\tau|) \geq 1$, after executing $Monitor(\rho, \tau, \nu, i, \phi)$, for all $\phi_k \in \mathcal{SF}^+(\phi)$, we have $i \models \phi_k$ iff $cur[k] =$ true.*

Theorem 7. *Given a model (ρ, τ, ν), world $i \geq 2$, and a TLI-monitorable formula ϕ, the procedure $Monitor(\rho, \tau, \nu, i, \phi)$ returns true iff $i \models \phi$.*

Theorem 8. *Given a model (ρ, τ, ν), the procedure $Monitor(\rho, \tau, \nu, i, \phi)$ takes space $m + \log(b + T)^n_{max} + log(max(b_{max}, T_{max}))^k$ and time $|\rho| ml$, where $m = |\mathcal{SF}^*(\phi)| \leq |\mathcal{SF}^+(\phi)| + (b + T)^k_{max}, n = \sum_{\phi_{\mathcal{C}}}, k = |\mathcal{SF}(\phi)|$, and $l = \max(\phi_{\mathcal{C}})$.*

TABLE 1
Standard CAN Frame Format

SOF	ID	Control	Data	CRC	ACK	EOF
1 bit	11 bits	7 bits	0-64 bits	16 bits	2 bits	7 bits

Proof. As can be seen from the monitoring algorithm, we need to store the array of the truth value of all subformulas of ϕ, the array of valuations of all the interval-skewed quantifiers, and the array of the lower bound and period of all the counting quantifiers occurring in ϕ. These are the three parts in the space complexity. Here k is number of usual subformulas of ϕ, m is the number of all subformulas of ϕ including not only the interval-skewed subformulas of the $\mathcal{S}$ formulas but also the instantiated subformulas of counting formulas. To record the valuations of counting quantifiers, we need an array of size n, where n is the number of all the interval-skewed quantifiers and it is calculated via summing up all the maximum numeric constants associated to each occurrence of $\mathcal{C}_I$ in ϕ. For example, if $I = [a, d]$, then the constant is d; if $I = [a, +\infty)$, then a. Particularly, the valuation is bounded by the number of corresponding equivalence classes, which is $b + T$ as shown in Proposition 2, and here we set it as the maximum one. We also need to store the lower bounds and periods for all variables used in ϕ. The size of the array is bounded by k and the element size is bounded by the maximum among all lower bounds and periods.

For the time complexity, we can see from the algorithm that the maximum nesting level of loops is three, and the number of iterations are $|\rho|, m$ and l, where l is the maximum number of interval-skewed counting quantifiers of a particular counting quantifier occurring in ϕ. □

6 Case Studies

Runtime monitoring provides effective realtime understanding and protection of the target system. Policies used in realtime monitoring may involve counting or frequency of some events. In this section, we first summarize some runtime monitoring applications in areas like anomaly detection, performance bug analysis and hardware level malware detection. Afterwards, we study and present some concrete security policies for runtime Android malware detection.

Anomaly detection is extensively used in cyber physical systems (CPSs) [19] and network intrusion detection [20]. In the anomaly detection scenario, various metrics are used to build the norm profile of the target system in order to detect anomalies. Frequency is among one of the most widely used metrics. Concerned frequencies include link utilization, CPU usage, login failure, as well as traffic rate and the rate of connections.

DoS attacks are the most common one [21], [22], [23] against intra-vehicle bus in automobile systems. A Controller Area Network (CAN) is a vehicle bus standard designed to allow microcontrollers and devices to communicate with each other in applications without a host computer. As shown in Table 1, each standard CAN frame consists of the following fields, 1-bit start of frame (SOF), 11-bit identifier (ID), 7-bit control field including data length code, up to

64-bit data field (Data), 16-bit CRC checksum, 2-bit ACK field and 7-bit end of frame (EOF). Each frame has a unique id and carries different information for such as acceleration, braking, engine revolutions per minute (RPM) and odometer readings. Frame ID is also used for arbitration when transmission conflicts, where the one with smaller ID always wins. That means, if an adversary injects smaller ID frames in a high frequency, it is possible to force transmission pending of normal frames (DoS attacks). Given the fact that most CAN traffics are transmitted at a fixed rate [24], [25] proposed a frequency check against anomaly communication on CAN bus by comparing historical statistics with the current data. Hyun Min Song et al. [26] analyzed the time interval of a single id traffic to detect anomaly changes when frames were injected in to CAN bus.

Algorithm 3. $Iter(\rho, \tau, \nu, i, \phi, prev, cur, p\_cnt, c\_cnt)$

$prev \leftarrow cur$;
$p\_cnt \leftarrow c\_cnt$;
for $k = 1$ **to** m **do**
  switch ϕ_k **do**
    case $\bot$ **do** $cur[k] \leftarrow false$;
    case p **do** $cur[k] \leftarrow p \in \rho_i$;
    case $\neg\psi$ **do** $cur[k] \leftarrow \neg cur[idx(\psi)]$;
    case $t > 0$ **do** $cur[k] \leftarrow t^{\nu} > 0$;
    case $\psi_1 \vee \psi_2$ **do** $cur[k] \leftarrow cur[idx(\psi_1)] \vee cur[idx(\psi_2)]$;
    case $\odot_I \psi$ **do** $cur[k] \leftarrow prev[idx(\psi)] \wedge (\tau_i - \tau_{i-1} \in I)$;
    case $\psi_1 \mathcal{S}_I \psi_2$ **do**
      $I' \leftarrow I - (\tau_i - \tau_{i-1})$;
      if $0 \in I \wedge cur[idx(\psi_2)]$ **then** $cur[k] \leftarrow true$;
      if $I' \neq \emptyset \wedge cur[idx(\psi_1)] \wedge prev[idx(\psi_1 \mathcal{S}_{I'} \psi_2)]$ **then** $cur[k] \leftarrow true$;
      else $cur[k] \leftarrow false$;
    case $\mathcal{C}_I x : \langle \alpha, \beta \rangle . \varphi(x)$ **do**
      for $Q = \mathcal{C}_{I'} x : \langle \alpha, \beta \rangle \in \mathcal{IS}_{\mathcal{Q}}(\mathcal{C}_I x : \langle \alpha, \beta \rangle)$ **do**
        $I'_{\Delta} \leftarrow I' - (\tau_i - \tau_{i-1})$;
        $Q' = \mathcal{C}_{I'_{\Delta}} x : \langle \alpha, \beta \rangle$;
        $c\_cnt[idx_c(Q)] \leftarrow 0$;
          if $0 \notin I' \wedge I'_{\Delta} \neq \emptyset$ **then** $c\_cnt[idx_c(Q)] \leftarrow p\_cnt[idx_c(Q')]$;
          else if $0 \in I' \wedge I'_{\Delta} = \emptyset$ **then**
            if $!cur[idx(\alpha)] \wedge cur[idx(\beta)]$ **then**
              $lb \leftarrow tli[idx_v(x)].lowerBound$;
              $pd \leftarrow tli[idx_v(x)].pediod$;
              if $lb > 1$ **then** $c\_cnt[idx_c(Q)] \leftarrow 1$; **else** $c\_cnt[idx_c(Q)] \leftarrow (1 - lb) \bmod pd + lb$;
          else if $0 \in I' \wedge I'_{\Delta} \neq \emptyset$ **then**
            if $!(cur[idx(\alpha)] \vee cur[idx(\beta)])$ **then** $c\_cnt[idx_c(Q)] \leftarrow p\_cnt[idx_c(Q')]$;
            else if $cur[idx(\beta)]$ **then**
              $n \leftarrow ++ p\_cnt[idx_c(Q')]$;
              $lb \leftarrow tli[idx_v(x)].lowerBound$;
              $pd \leftarrow tli[idx_v(x)].period$;
              if $n \geq lb + pd$ **then** $c\_cnt[idx_c(Q)] \leftarrow (n - lb) \bmod pd + lb$; **else** $c\_cnt[idx_c(Q)] \leftarrow n$;
      $cur[k] \leftarrow cur[idx(\varphi^{\nu[x \mapsto c\_cnt[idx_c(\mathcal{C}_I x:\langle\alpha,\beta\rangle)]]})]$;
return $cur[idx(\phi)]$;

Performance bugs [27], [28] are drawing more and more attention recently. Different from traditional functional bugs, performance bugs degrade performance and waste computation resource. GUI lagging [28] is one of the most commonly found performance bugs, where user events cannot be handled in a timely manner. A convenient and effective approach [29] to identify potential GUI lagging bugs is to measure the number of frames a graphics-rendering unit can produce per second. When the target application, e.g., Firefox explore, scores less than 60 frames per second, it is likely that there exist GUI lagging bugs and further optimizations have to be made.

RowHammer [30] is a kind of disturbance error found in dynamic random-access memory. Physically, it results from the increasingly denser memory cells and smaller cell isolation margins, and in this case higher-than-natural access rate of certain memory rows can possibly cause bit flips in adjacent rows. The first exploit of RowHammer was released by Google's Project Zero team in [31], which is able to gain kernel privileges. Recently, Gruss et al. [32] demonstrated an JavaScript-based exploit that can launch Rowhammer attack with the eviction rate up to 99.99 percent. This is also the first remote software-induced hardware-fault attack. A countermeasure to detect suspicious RowHammer attack with hardware performance counter is proposed by [33] in 2015. The core technique is to observe high numbers of last-level cache (LLC) miss events within short periods of time.

From the case studies above, we can see runtime monitoring of some frequency related policies is indispensable for detecting some anomaly or malicious behaviors. Malware, as a major threat to Android ecosystem [34], [35], is sometimes difficult to identify with current static detection approaches [36], [37]. However, runtime monitoring techniques can help to capture stealthy behaviors during the execution. Here, we put more attention on several security policies extensively discussed in Android system and some concrete policies are provided as case studies for MTL_{cnt}. In the rest of this article, we assume the following atomic propositions in Android OS.

- S_i (or E_i) : the app with UID i starts to run (or stops running).
- M_i : the app with UID i sends a message.
- I_i : the app with UID i opens an Internet connection socket.
- F_i : the app with UID i forks a new child process.

The following policies refer to the malicious access patterns that are forbidden in Android systems. As for the bounded time interval and the limit on specific count variable, we just give a rough estimation for illustration purpose. The unit adopted for the time intervals is second. Every policy given below takes the form $\boxdot\neg\phi$ where ϕ describes a violation of the policy at a given time. The prefix $\boxdot$ ensures that globally such a violation does not occur.

1) SMS-based Botnet Attack

$$\boxdot\neg[\mathcal{C}_{[0,1800)} x : \langle \bot, M_i \rangle . (x > 30)].$$

This policy describes that an app with UID i cannot send more than 30 SMS messages within any 30 minutes. This is a requirement for the SMS sending rate from Android 4.1 [38], and for older systems the limit is 100 SMS messages per hour. Arzt et al. [39] also

suggest to stop unintended SMS transmissions by limiting the SMS sending frequency. In [4], authors find that current Android botnets are exploiting SMS messages to gather money by sending SMS to premium-rate numbers. With the specified policy, it helps to discover possible tainted Android bots.

2) DDos Attack

$$\boxdot\neg[\mathcal{C}_{[0,3)}x : \langle \bot, I_i\rangle.(x > 5)].$$

This policy specifies that an app cannot open the Internet connection socket for more than 5 times in 3 seconds. If an Android app aims at flooding a targeted server to launch a DDoS attack, it usually tries to open massive Internet connections. This policy can help to control the amount of Internet connections, thus preventing some potential malware.

3) RageAgainstTheCage

$$\boxdot\neg[\mathcal{C}x : \langle S_i, F_i \wedge \neg E_i\rangle.(x > 2^{16})].$$

This policy says that during the life cycle of an app, it is not allowed to create more than 2^{16} child processes to exhaust the domain of *pid*, i.e., the process identifier, in Linux kernel. *RageAgainstTheCage* [40] is a well-known exploit on Android, which can perform unauthorized privileged actions by gaining the root access. This malware uses a vulnerability in Android kernel to get the root privilege by keeping forking the child process to 2^{16}. With this policy, a constraint is set on how many child processes an app can fork during a single run.

7 Evaluation

7.1 Application on Android

We implement the monitoring algorithm for MTL_{cnt} and evaluated it on LogicDroid platform [3], which is a modified version of Android 4.1. Instrumentation on IPC (Inter-process communication) calls, like opening Internet socket, sending SMS and accessing contact database, is deployed in LogicDroid. These IPC calls compose the event trace, and whenever they happen during the system execution, the monitoring module in the kernel is notified and runtime examination will proceed. Aiming to demonstrate the trace-length independence property, we measure the time used during the monitoring. We also measure the overhead caused by the RM mechanism under different event throughput and it turns out the delays are negligible for normal use.

For the benchmarks used, since hooks on the start or end of a process or an app have not been implemented in LogicDroid, some of the reset conditions in the example policies are not applicable. Policy 3) in Section 6 is also excluded from the evaluation due to the lack of support for process forking instrumentation in LogicDroid. Consequently, the benchmark policies used in our experiments are:

P1: $\boxdot\neg[\mathcal{C}_{[0,1800)}x : \langle \bot, M_i\rangle.(x > 30)]$
P2: $\boxdot\neg[\mathcal{C}_{[0,3)}x : \langle \bot, I_i\rangle.(x > 5)]$
P3: $\boxdot\neg[\mathcal{C}_{[0,30)}x : \langle \bot, I_i\rangle.(x > 5)]$
P4: $\boxdot\neg[\mathcal{C}_{[0,300)}x : \langle \bot, I_i\rangle.(x > 5)]$
P5: $\boxdot\neg[\mathcal{C}_{[0,3)}x : \langle \bot, I_i\rangle.(x > 50)]$
P6: $\boxdot\neg[\mathcal{C}_{[0,3)}x : \langle \bot, I_i\rangle.(x > 500)]$

TABLE 2
Details of Policies for Android

Policy	Module Size(KB)	$\lvert\mathcal{SF}^*(\phi)\rvert$	$\max(\phi_{\mathcal{C}})$	Quantifiers		
				variable	*lb*	*pd*
P1	407	35	1800	x	31	1
P2	405	10	3	x	6	1
P3	405	10	30	x	6	1
P4	405	10	300	x	6	1
P5	408	55	3	x	51	1
P6	439	505	3	x	501	1
P7	407	28	3	x	1	3
				y	3	2

P7: $\boxdot\neg[\mathcal{C}_{[0,3)}x : \langle \bot, M_i\rangle.\mathcal{C}_{[0,3)}y : \langle \bot, I_i\rangle.K(x,y) > 0]$ where function K is defined as follows:

$$K(x,y) = \begin{cases} 3x - 4y & \text{if } x < 4 \text{ and } y < 5, \\ K(x-3,y) & \text{if } x \geq 4 \text{ and } y < 5, \\ K(x,y-2) & \text{if } x < 4 \text{ and } y \geq 5, \\ K(x-3,y-2) & \text{otherwise.} \end{cases}$$

Note that each projection of function K becomes periodic once $x \geq 1$ and $y \geq 3$, so it is easy to show that φ_K is TLI-monitorable.

Note that in the implementation in LogicDroid, the runtime monitor is used to prevent execution that could lead to violation of a given security policy. This means that in practice, at every stage, the runtime monitor can assume that no violation of the given policy has occurred in the past. In this setting, the job of the runtime monitor, when enforcing a given policy of the form $\boxdot\neg\phi$ in the current world and given an event generated by the app (i.e., a request to some resource), is to check whether the formula ϕ would be satisfied or not in the (hypothetical) next world where the event is allowed to happen. If ϕ would be satisfed in this next world, which means the policy would be violated, the monitor would terminate the request. So in practice, only the truth value of ϕ needs to be computed, and the truth value of $\boxdot\neg\phi$ is implicit in the way the runtime monitor is implemented and deployed.

The first two policies P1 and P2 are those introduced in Section 6, and others are artifacts used to evaluate the effectiveness of our approach, as interval ranges and arithmetic relations (with different *period* and *lower bound*) may affect the performance. We summarize key characteristics of the policies according to the metrics used in the complexity analysis, including the number of subformulas, the max number of interval-skewed counting quantifiers of a particular counting quantifier, and the lower bound and period of corresponding variables, as listed in column 3-7 in Table 2. The policies cover a wide range of each complexity metrics, e.g., the number of subformulas varies from 10 to 505. We also include a multivariate arithmetic relations as in P7.

The monitor of each policy is implemented in C language according to the algorithm described in Section 5 and is compiled as an Android kernel module. For quantified variables in the tested policies, the calculation of its corresponding *period* and *lower bound* is currently done manually. Since there is no dynamic memory allocation in the monitoring algorithm, the total memory occupied by a monitor is just the size of the monitor after loaded to the kernel, as shown in the second column in Table 2. There is no obvious expansion in the

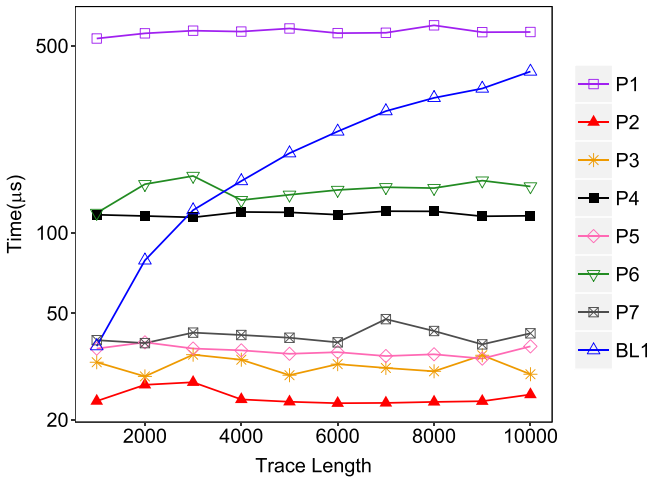

Fig. 1. Time of single round checking on Android.

size of monitor as the number of subformulas increases by dozens of times. The overall memory in Android emulator is around 840 MB, in this case the memory overhead of the monitor is just around 0.5 percent. We run LogicDroid on an emulator and a prepared kernel module is inserted to the system once its initialization is finished. We measure the performance of each policy separately, which means only one kernel module is inserted at each experiment. The monitor module is aware of every occurrence of the instrumented events, and is able to continuously and simultaneously check whether the system behavior so far complies with the policy stipulated. To facilitate the experiment, we also implement a fuzzy app to trigger IPC calls randomly at user-specified throughput.

All the experiments are conducted on a PC with 64-bit Ubuntu 16.04LTS system, 16 GB RAM and a 3.50 GHz Intel Xeon(R) CPU E5-1650 v3. Our implementation and the modules used in this section are all available on [41]. We first measure the time used by each iteration of the for-loop as shown in Algorithm 1, and we call it iteration time. The fuzzy app is set to send an IPC call every 0.1s and we show the results when the event trace length grows to 10,000. For each policy, we record the time used for each iteration and calculate the average time for every 1,000 calls. Note that the time measured here is the pure execution time of the monitor excluding that used to finish the user requested tasks or switch between user mode and kernel mode. To get statistically significant result, we take the average of five times' measurement for all the experiments. From the results shown in Fig. 1, we can see that the iteration time for each policy keeps constant as trace length increasing, which is a strong evidence for the trace-length independence. The y-axis is in logarithm scale. Policies P2, P3 and P4 are with increasing max instants by the magnitude of 10. Policies P2, P5 and P6 are with increasing number of subformulas resulting from the increasing lower bounds, and the magnitude is also around 10. For both cases, the increase in examination time is finally around 5 times as can be seen from the figure. Among these seven policies, P1 requires the longest examination time, approximately 500 microseconds, mainly due to the max number of interval-skewed quantifiers. We also conduct a contrast experiment as baseline, where the event trace is recorded by the monitor for future query. It is not implemented as a specific policy but only to record and query, which is the basic

TABLE 3
Time Overhead under Different Throughput for Android

Event Interval (ms)	Overhead(%)						
	P1	P2	P3	P4	P5	P6	P7
0	8.77	0.72	1.53	4.78	4.21	1.48	1.69
1	3.59	1.28	1.69	3.15	2.54	1.23	1.43
10	0.07	-0.14	-0.24	-0.42	-0.27	0.15	0.04

functionality in a naive monitoring algorithm. The UIDs of the caller and callee are stored as integers and the timestamps are stored as a long integer in the memory. Whenever the trace grows, we measure the time used to traverse the whole history. The result is shown as the *BL1* line in Fig. 1. We can see that when the trace length grows from 0 to 10,000, the time used for each iteration roughly increases by 8 times. This is a significant defect for realtime runtime monitoring.

We also measure how much time overhead is introduced after adding the monitoring mechanism to Android. Since different IPC calls may take different time to finish, instead of triggering several IPC calls randomly, here we just trigger Internet socket requests. The fuzzy app is set to open Internet socket every 0, 1, and 10 milliseconds respectively for 10,000 times. We measure the time used to finish this task by the emulator with and without the monitoring module and the overhead results are shown in Table 3. For the largest throughput, the worst performance is observed for P1, 8.77 percent, which is rarely spotted by users. When the interval between events drops to 10 milliseconds, the time overhead becomes less than 0.5 percent. Here are several negative overhead due to some minor deviation in time measurement. It also reflects that the overhead is extremely low and maybe affected by measurement error. In normal usage of mobiles, the events throughput is even smaller, and we can claim that the overhead of our monitoring mechanism is negligible.

7.2 Application on Autonomous Vehicle

Our intrusion detection system for autonomous vehicle on its CAN bus has been built on Robot Operating System with Ubuntu 16.04. We use USB-CAN adapters to build a real CAN bus system for evaluation. In our setup, the CAN bus system connects a computer running the autonomous driving suite and issues control commands over the CAN bus to the corresponding electronic control units (ECUs), and ECUs could send feedback to the driving system via the bus. In the meantime, our monitor links to the CAN bus to detect any abnormal traffic. It sniffs CAN frames with libpcap and checks communication features of each frame ID. To detect any DoS attacks, the monitor counts and compares the frequency of each CAN frame traffic with a threshold periodically for any sudden increasing. F_i is used to denote the CAN frame with ID i and the policies used in the evaluation are as follows:

P8: $\boxdot\neg[\mathcal{C}_{[0,10)}x : \langle\bot, F_i\rangle.(x > 20)]$
P9: $\boxdot\neg[\mathcal{C}_{[0,50)}x : \langle\bot, F_i\rangle.(x > 20)]$
P10: $\boxdot\neg[\mathcal{C}_{[0,250)}x : \langle\bot, F_i\rangle.(x > 20)]$
P11: $\boxdot\neg[\mathcal{C}_{[0,10)}x : \langle\bot, F_i\rangle.(x > 100)]$
P12: $\boxdot\neg[\mathcal{C}_{[0,10)}x : \langle\bot, F_i\rangle.(x > 500)]$

The monitor is implemented as an independent process and just "listening" to the CAN frames on the CAN bus, which means that it has no effect on the timing of CAN communication. Hence, it makes no sense to investigate the delay

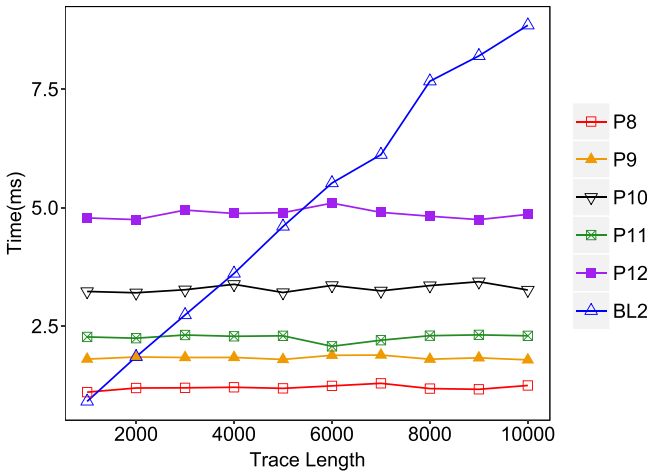

Fig. 2. Time of single round checking on autonomous vehicle.

caused by the monitor under different throughput, and we only look into its performance with different policies. The time intervals are in millisecond for high precision measurement. Policies P8, P9 and P10 are with increasing time interval (by 5 times), and policies P8, P11 and P12 are with increasing lower bound (by around 5 times). Detailed characteristics of the policies and the memory usage are available in Table 4. This is designed to measure the performance of policies under different complexity. We calculate the average time used for every 1000 CAN frames, and the results are shown in Fig. 2. Basically, the monitor takes 1 to 5 milliseconds to do a single round checking, which is low enough for monitoring. When the size of the time interval increases by 25 times, the time used increases by around 3 times. When the number of the subformulas increases by 25 times, the time used increase by around 5 times. The overall increment is slow, which means the monitor can potentially provide good performance for even more complicated policies. Similar to the previous evaluation on Android, we implement a baseline monitor which records each frame with its ID and timestamp, and simply traverses through the entire trace for each new world, i.e., not implementing any specific policy. As shown in the *BL2* line in Fig. 2, the time increases rapidly as the trace length grows. With more and more delays, the monitor fails to react in a real-time manner.

8 Related Work

Instead of listing all the related works, we discuss the most related ones in this section, which will cover a brief history of trace-length independent RM and also a bunch of specification languages with counting semantics but unfortunately fail to have a trace-length independent monitor.

TLI Monitors. The concept of trace-length independence is proposed in 2013 [9], where a TLI monitoring algorithm based on spawning automaton for a first-order temporal logic, LTL^{FO}, is devised. Actually, there have been some works provided with TLI property before that. In 2002, [10] gives a TLI solution for the monitoring of past-time LTL with dynamic programming. Later, the same problem is discussed in terms of MTL, the metric temporal extension of LTL. Thati and Roşu [11] present a TLI monitoring algorithm for MTL with both past and future fragments, but restraining the

TABLE 4
Details of Policies for Autonomous Vehicle

Policy	Module Size(B)	$\lvert\mathcal{SF}^*(\phi)\rvert$	$\max(\phi_C)$	Quantifiers		
				variable	*lb*	*pd*
P8	336,472	25	10	x	21	1
P9	336,472	25	50	x	21	1
P10	336,472	25	250	x	21	1
P11	336,476	105	10	x	101	1
P12	336,500	505	10	x	501	1

interpretation of future to truncated finite traces. Their algorithm is also designed with dynamic programming based on the derivatives of formulas. The time and space complexity is linear to the size of the formula for MTL with only past time operators, and exponential for full fragment MTL. TLI monitoring algorithms for ptMTL with pointed-based and interval-based time models are discussed in [8]. For the point-based ptMTL semantics, their algorithm provides a better performance on time complexity compared to that in [11]. However, the list of timestamps used in their algorithms dealing with $\mathcal{S}$ operator can grow boundlessly large when the time sequence is not strictly increasing, that is, it is dependent on the event rate. This problem also exists in the monitor of full-fragment MTL proposed in [42]. This restriction on time sequence is released in [3], where a TLI monitoring algorithm for ptMTL extended with recursive definition is designed while restricting all the time intervals bounded with the temporal operators to starting from 0. Recently, there also arises some discussion on event-rate independent monitoring MTL [18] and metric dynamic logic [43].

Counting Semantics. There are some works focusing particularly on the counting semantics, and also some others on general aggregation semantics. The most related counting quantifier is proposed in [6] but their algorithm is PSPACE-complete in the size of history length. Laroussinie et al. [44] present a quantitative extension of LTL, called CLTL, allowing to specify the number of states satisfying certain sub-formulas. The semantics is the same with ours on the non-metric part. According to their investigation, even though CLTL formulas can be translated into classical LTL, an exponential blow-up in formula size is inevitable. As for the satisfiability and model-checking problem of CLTL, it turns out to be EXPSPACE-complete, and PSPACE-complete when restricting to a fragment. Actually this fragment belongs to set of the TLI-formulas stipulated in our work, for which the valuation of the term will grow monotonously with the count increasing.

Here are some works extending specification languages with general aggregation semantics. Basin et al. [7] extend metric first-order temporal logic (MFOTL) with aggregation operators, like SUM, CNT, MAX and AVG, and propose a monitoring algorithm where policies are translated to an extended relational algebra and checked via SQL-like processing. SOLOIST [14] is a many-sorted first-order metric temporal logic extended with new temporal modalities that support aggregate operators for events occurring in a certain time window. On its monitoring problem, Bianculli et al. [45] and Bersani et al. [46] give some solutions, but all are dependent on SMT-solver, and evaluations in these two works show that increasing time and memory are needed when trace length

grows. Other specification languages with support for different kinds of aggregations are LarvaSat [47], LOLA [48], as well as rule-based EAGLE [49], RULER [50] and LOGFIRE [51], and also one based on algebraic alternating automata [52]. However, monitoring algorithms for the above languages all need to record the specific count values, even though most of them avoided storing the entire trace history. In principle, these counts can increase indefinitely, so the space required is not constant.

To the best of our knowledge, this is the first work on TLI monitoring algorithms for logics equipped with metric temporal counting quantifier.

9 Conclusion

We presented a formal policy specification language MTL_{cnt} that allows expressions of temporal quantitative policies. We considered the questions of when a formula is trace-length independent monitorable. For univariate relations, we obtained sufficient and necessary conditions for the relations to be TLI-monitorable. We then discussed an extension to the multivariate relations. Assuming the relations are all TLI-monitorable, we constructed a TLI monitoring algorithm for MTL_{cnt}. We implemented and tested our monitoring algorithm, and the experimental results confirmed our theoretical results. Currently, we have not yet addressed the integration of the counting quantifier with metric operators and recursive predicates of [3]. This is a subject of immediate future work. We also plan to look to incorporate other, more expressive aggregate operators from [7].

Appendix A
Example of TLI-Monitorable Relation

Example 2. Let $t(x) = x^2 - 8x + 15, x \in \mathbb{N}$. In this case, the characteristic function of t is

$$f_t(x) = \begin{cases} 0 & \text{if } 3 \le x \le 5, \\ 1 & \text{otherwise.} \end{cases}$$

Since $t(x) \le 0$ holds only if $3 \le x \le 5$, function f_t is lb-periodic on $[6, +\infty)$, with lower bound 6 and period 1. So $t > 0$ is TLI-monitorable according to Lemma 2.

Example of Interval-Skewed Subformula

Example 3. For $\phi = p_1 \mathcal{S}_{[2,4)} p_2$, since $\{[2,4) - n | n \in [1,3]\} = \{[1,3), [0,2), [0,1)\}$, we get

$$\mathcal{SF}^k(\phi) = \{p_1 \mathcal{S}_{[1,3)} p_2 \ , \ p_1 \mathcal{S}_{[0,2)} p_2 \ , \ p_1 \mathcal{S}_{[0,1)} p_2\}.$$

Specially, if the *since* operator is bounded with an infinite time interval, e.g., $I = [3, +\infty)$, then $(I - n) \in \{[0, +\infty), [1, +\infty), [2, +\infty)\}$.

Proof of Proposition 2

Proof. ($\Leftarrow$) Supposing $e \ge b_i + T_i$, accordingly we have

$$[e] = \{a \,|\, (a - b_i) \bmod T_i + b_i = e, a \ge b_i, a \in \mathbb{N}\}.$$

According to $(a - b_i) \bmod T_i + b_i = e$ and $a \ge b_i$ we can get

$$\begin{aligned} &(a - b_i) \bmod T_i \in [0, T_i) \\ \Leftrightarrow &(a - b_i) \bmod T_i + b_i \in [b_i, b_i + T_i) \\ \Leftrightarrow &e \in [b_i, b_i + T_i). \end{aligned}$$

This is contradictory to $e \ge b_i + T_i$, so $\nexists\, a \ge b_i . (a - b_i) \bmod T_i + b_i = e$, i.e., $[e]$ is empty.

($\Rightarrow$) Given $[e]$ is empty, according to the definition of $[e]$,

$$[e] = \begin{cases} \{e\} & \text{if } e < b_i, \\ \{a \,|\, (a - b_i) \bmod T_i + b_i = e, a \ge b_i\} & \text{otherwise.} \end{cases}$$

since $e \notin [e]$, we can get $e \ge b_i$, such that

$$\begin{aligned} &[e] = \{a | (a - b_i) \bmod T_i + b_i = e, a \ge b_i, a \in \mathbb{N}\} = \emptyset \\ \Leftrightarrow &\forall a \ge b_i . (a - b_i) \bmod T_i + b_i \ne e, \\ \Leftrightarrow &\forall a \ge b_i . (a - b_i) \bmod T_i \ne e - b_i \\ \Rightarrow &e - b_i \ge T_i, (e > b_i) \\ \Rightarrow &e \ge b_i + T_i. \end{aligned}$$

Now the proposition follows. □

Proof of Theorem 5

Proof. In this proof, the valuation of x determined by $\mathcal{C}_I x : \langle \alpha, \beta \rangle)$ is analyzed under the two cases of whether $0 \in I$. Since we assume the counting formula is TLI-monitorable, x is used either in a TLI-monitorable univariate relation whose characteristic function is lb-periodic on $[b, +\infty)$ and with period T, or a TLI-monitorable multivariate relation for which the characteristic functions of slices on x are lb-periodic on $[b, +\infty)$ and with period T. According to the semantics of metric temporal counting quantifier defined in Section 2, for $\mathcal{C}_I x : \langle \alpha, \beta \rangle)$ we denote

$$\begin{aligned} \mathbb{B} &= \{h \,|\, h \models \beta, \text{ for } \tau_1 - \tau_h \in I, m < h \le 1\}, \\ \mathbb{A} &= \{h \,|\, h \models \alpha, \text{ for } \tau_1 - \tau_h \in I, 1 \le h \le 1\}, \\ l &= |\mathbb{B}|, \\ m &= max(\mathbb{A} \cup \{0\}). \end{aligned}$$

Thus we only need to prove l always belongs to the equivalence class $[\mathfrak{r}(1, \mathcal{C}_I x : \langle \alpha, \beta \rangle)]$.

Case(1): If $0 \notin I$, then

$$0 \notin I \Rightarrow \tau_1 - \tau_1 \notin I \Rightarrow 1 \notin \mathbb{B} \Rightarrow \mathbb{B} = \emptyset.$$

We can conclude that

$$\mathbb{B} = \emptyset \Rightarrow l = 0.$$

Hence, $l \in [\mathfrak{r}(1, \mathcal{C}_I x : \langle \alpha, \beta \rangle)]$.

Case(2): If $0 \in I$, cases are analyzed based on the satisfaction of $1 \models \beta$ and $1 \models \alpha$.

For case(a),

$$1 \models \alpha \Rightarrow 1 \in \mathbb{A} \Rightarrow m = 1 (0 \in I) \Rightarrow \mathbb{B} = \emptyset \Rightarrow l = 0.$$

Thus, $l \in [\mathfrak{r}(1, \mathcal{C}_I x : \langle \alpha, \beta \rangle)]$.

For case(b),

$$1 \not\models \alpha \Rightarrow \mathbb{A} = \emptyset \Rightarrow m = 0,$$

and

$$1 \not\models \beta \Rightarrow 1 \notin \mathbb{B} \Rightarrow \mathbb{B} = \emptyset \Rightarrow l = 0.$$

Hence, $l \in [\mathfrak{r}(1, \mathcal{C}_I x : \langle \alpha, \beta \rangle)]$.

For case(c) and case(d), it is easy to get

$$1 \not\models \alpha \Rightarrow m = 0,$$

and

$$1 \models \beta \Rightarrow 1 \in \mathbb{B}(0 \in I) \Rightarrow \mathbb{B} = \{1\} \Rightarrow l = 1.$$

Following the definition of equivalence class in Definition 11, we can easily know $l \in [\mathfrak{r}(1, \mathcal{C}_I x : \langle \alpha, \beta \rangle)]$.

In conclusion, the theorem follows for the cases above. □

Proof of Lemma 4

Proof. We assume all subformulas in $\mathcal{SF}^+(\phi)$ are well-ordered. The procedure *Init* will assign verdict to the subformulas respecting the partial order, i.e., for any $\phi_i \prec \phi_j$, ϕ_i will be valuated before ϕ_j. For each subformula, we conduct inductive analysis based on its structure. For ϕ_k :

(Base cases)

Case $\bot$: Obviously $1 \not\models \bot$ and with $cur[k] = false$, we have

$$1 \models \bot \text{ iff } cur[k] \equiv true.$$

Case $p : p$ is an atomic proposition, and for state ρ_1, $cur[k] = p \in \rho_1$, obviously

$$1 \models p \text{ iff } cur[k] \equiv true.$$

Case $t > 0$: This is an arithmetic relation. Since the valuations of all variables are determined initially, the truth value of this relation can be got following normal calculation rules, that is $cur[k] = t^v > 0$, so

$$1 \models t > 0 \text{ iff } cur[k] \equiv true.$$

Case $\odot_I \psi$: Since there is no previous states for the first state, $1 \not\models \odot_I \psi$. With $cur[k] = false$, naturally

$$1 \models \odot_I \psi \text{ iff } cur[k] \equiv true.$$

(Induction cases)

Case $\neg\psi$: Since $\psi \prec \neg\psi$ and it has been proved

$$1 \models \psi \text{ iff } cur[idx(\psi)] \equiv true,$$

then with $cur[k] = \neg cur[idx(\psi)]$ we have

$$1 \models \neg\psi \text{ iff } cur[k] \equiv true.$$

Case $\psi_1 \vee \psi_2$: Since $\psi_1 \prec \phi_k$, $\psi_2 \prec \phi_k$ and

$$1 \models \psi_1 \text{ iff } cur[idx(\psi_1)] \equiv true,$$
$$1 \models \psi_2 \text{ iff } cur[idx(\psi_2)] \equiv true,$$

then with $cur[k] = cur[idx(\psi_1)] \vee cur[idx(\psi_2)]$, we have

$$1 \models \psi_1 \vee \psi_2 \text{ iff } cur[k] \equiv true.$$

Case $\psi_1 \mathcal{S}_I \psi_2$: At state ρ_1, according to the semantics defined in Section 2.2, the only possible value of j is 1, in which this case there exists no valid k. The model relation holds if and only if $\tau_i - \tau_i = 0 \in I$ and $1 \models \psi_2$. Since $\psi_2 \prec \phi_k$, we have

$$1 \models \psi_2 \text{ iff } cur[idx(\psi_2)] \equiv true.$$

With $cur[k] = 0 \in I \wedge cur[idx(\psi_2)]$ we have

$$1 \models \psi_1 \mathcal{S}_I \psi_2 \text{ iff } cur[k] \equiv true.$$

Case $\mathcal{C}_I x : \langle \alpha, \beta \rangle . \varphi(x)$: For the valuation of x, the correctness has been proved in Theorem 5. We mention here since $\alpha \prec \phi_k$, $\beta \prec \phi_k$ and $\varphi^{v[x \mapsto c\_cnt[idx_c(\mathcal{C}_I x:\langle \alpha,\beta \rangle)]} \prec \phi_k$, it follows that

$$1 \models \alpha \text{ iff } cur[idx(\alpha)] \equiv true,$$
$$1 \models \beta \text{ iff } cur[idx(\beta)] \equiv true,$$
$$1 \models \varphi' \text{ iff } cur[idx(\varphi')] \equiv true,$$

with $\varphi' = \varphi^{v[x \mapsto c\_cnt[idx_c(\mathcal{C}_I x:\langle \alpha,\beta \rangle)]}$.

With the above inductive analysis, the lemma follows. □

Proof of Lemma 5

Proof. In *Iter* procedure, first *prev* array is updated with *cur* array from previous iteration. Accordingly, the assumption becomes,

$$\text{for all } \phi_k \in \mathcal{SF}^+(\phi), i-1 \models \phi_k \text{ iff } prev[idx(\phi_k)].$$

Formulas in $\mathcal{SF}^+(\phi)$ are well-ordered and each formula will be judged after its subformulas. Since the formulas are constructed inductively, we will conduct inductive analysis on the structure of each formula to obtain the truth value. Then for ϕ_k :

(Base cases)

Case $\bot$: Obviously $i \not\models \bot$ and with $cur[k] = false$, we have

$$i \models \bot \text{ iff } cur[k] \equiv true.$$

Case $p : p$ is an atomic proposition, and for state ρ_1, $cur[k] = p \in \rho_1$, obviously

$$i \models p \text{ iff } cur[k] \equiv true.$$

Case $t > 0$: This is an arithmetic relation. Since the valuations of all variables are determined initially, the truth value of this relation can be got following normal calculation rules, that is $cur[k] = t^v > 0$, so

$$i \models t > 0 \text{ iff } cur[k] \equiv true.$$

(Induction cases)

Case $\neg\psi$: According to the semantics,

$$i \models \neg\psi \text{ iff } i \not\models \psi.$$

Since $\psi \prec \neg\psi$ and

$$i \models \psi \text{ iff } cur[idx(\psi)] \equiv true,$$

then with $cur[k] = \neg cur[idx(\psi)]$ we can get

$$i \models \neg\psi \text{ iff } cur[k] \equiv true.$$

Case $\psi_1 \vee \psi_2$: According to the semantics,

$$i \models \psi_1 \vee \psi_2 \text{ iff } i \models \psi_1 \text{ or } i \models \psi_1.$$

Since $\psi_1 \prec \psi_1 \vee \psi_2$ and $\psi_2 \prec \psi_1 \vee \psi_2$, and also

$$i \models \psi_1 \text{ iff } cur[idx(\psi_1)] \equiv true,$$
$$i \models \psi_2 \text{ iff } cur[idx(\psi_2)] \equiv true,$$

then with $cur[k] = cur[idx(\psi_1)] \vee cur[idx(\psi_2)]$ we can get

$$i \models \psi_1 \vee \psi_2 \text{ iff } cur[k] \equiv true.$$

Case $\odot_I\psi$: According to the semantics,

$$i \models \odot_I\psi \text{ iff } i-1 \models \psi \text{ and } \tau_i - \tau_{i-1} \in I.$$

Since

$$i-1 \models \psi \text{ iff } prev[idx(\psi)] \equiv true,$$

then with $cur[k] = prev[idx(\psi)] \wedge (\tau_i - \tau_{i-1}) \in I$ we can get

$$i \models \odot_I\psi \text{ iff } cur[k] \equiv true.$$

Case $\psi_1 \mathcal{S}_I \psi_2$: According to the semantics,

$$\begin{aligned} \nu, i \models \phi_1 \mathcal{S}_I \phi_2 \text{ iff } & 0 \in I \text{ and } \nu, i \models \phi_2, \text{ or} \\ & i > 1, \text{ and } \nu, i \models \varphi_1 \text{ and } \nu, i \models \phi_1 \mathcal{S}_{I'} \phi_2 \\ & \text{where } I' = I - (\tau_i - \tau_{i-1}). \end{aligned}$$

Since $\psi_1 \prec \psi_1 \mathcal{S}_I \psi_2, \psi_2 \prec \psi_1 \mathcal{S}_I \psi_2$ and $\psi_1 \mathcal{S}'_I \psi_2 \prec \psi_1 \mathcal{S}_I \psi_2$,

$$i \models \psi_1 \text{ iff } cur[idx(\psi_1)] \equiv true,$$
$$i \models \psi_1 \text{ iff } cur[idx(\psi_2)] \equiv true,$$
$$i-1 \models \psi_1 \mathcal{S}'_I \psi_2 \text{ iff } prev[idx(\psi_1 \mathcal{S}'_I \psi_2)] \equiv true,$$

then we know it follows that

$$i \models \odot_I\psi \text{ iff } cur[k] \equiv true.$$

Case $\mathcal{C}_I x : \langle \alpha, \beta \rangle . \varphi(x)$: Basically, the algorithm deals with the metric counting quantifier following the recursive semantics given in Theorem 6 and the correctness has been proved. We mention here since $\alpha \prec \phi_k$, $\beta \prec \phi_k$ and $\varphi^{v[x \mapsto c\_cnt[idx_c(\mathcal{C}_I x:\langle\alpha,\beta\rangle)]}} \prec \phi_k$, it follows that

$$i \models \alpha \text{ iff } cur[idx(\alpha_j)] \equiv true,$$
$$i \models \beta \text{ iff } cur[idx(\beta_j)] \equiv true,$$
$$i \models \varphi' \text{ iff } cur[idx(\varphi')] \equiv true,$$

with $\varphi' = \varphi^{v[x \mapsto c\_cnt[idx_c(\mathcal{C}_I x:\langle\alpha,\beta\rangle)]}}$.

With the above induction analysis, the lemma follows. □

Proof of Lemma 6

Proof. The procedure *Monitor* first calls *Init*, according to Lemma 4, we have, for all $\phi_k \in \mathcal{SF}^+(\phi)$,

$$1 \models \phi_k \text{ iff } cur[k] \equiv true.$$

Then for $i \geq 2$, we assume, for all $\phi_k \in \mathcal{SF}^+(\phi)$,

$$i-1 \models \phi_k \text{ iff } cur[k] \equiv true.$$

According to Lemma 5, after executing $Iter(\rho, \tau, \nu, i, \phi, prev, cur, p\_cnt, c\_cnt)$, we have, for all $\phi_k \in \mathcal{SF}^+(\phi)$,

$$i \models \phi_k \text{ iff } cur[k] \equiv true.$$

This lemma follows. □

Proof of Theorem 7

Proof. According to Lemma 6, after executing $Monitor(\rho, \tau, \nu, i, \phi)$, we have, for all $\phi_k \in \mathcal{SF}^+(\phi)$,

$$i \models \phi_k \text{ iff } cur[idx(\phi_k)] \equiv true.$$

Since $\phi \in \mathcal{SF}^+(\phi)$, naturally we can get

$$i \models \phi \text{ iff } cur[idx(\phi)] \equiv true.$$

This theorem follows. □

Summary of Policies for Autonomous Vehicle

Since the monitor is implemented as an independent process, we measured the memory it takes right after the process is launched. From the results we can see, the size of memory taken is largely related to the number of subformulas, but the increment is slow comparing to the increment of subformulas. As there is no allocation of extra memory, the memory usage by the monitor itself would remain at this amount, which is negligibly small.

Acknowledgments

This research was supported (in part) by the National Research Foundation, Prime Ministers Office, Singapore under its National Cybersecurity R&D Program (Award No. NRF2014NCR-NCR001-30) and National Satellite of Excellence in Trustworthy Software System (Award No. NRF2018NCR-NSOE003-0001) and administered by the National Cybersecurity R&D Directorate.

References

[1] R. Alur and T. A. Henzinger, "Real-time logics: Complexity and expressiveness," *Inf. Comput.*, vol. 104, no. 1, pp. 35–77, 1993.
[2] A. Pnueli, "The temporal logic of programs," in *Proc. 18th Annu. Symp. Found. Comput. Sci.*, 1977, pp. 46–57.
[3] H. Gunadi and A. Tiu, "Efficient runtime monitoring with metric temporal logic: A case study in the android operating system," in *Proc. Int. Symp. Formal Methods*, 2014, pp. 296–311.

[4] H. Pieterse and M. S. Olivier, "Android botnets on the rise: Trends and characteristics," in *Proc. Inf. Secur. South Africa*, 2012, pp. 1–5.

[5] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *Proc. IEEE Symp. Secur. Privacy*, 2012, pp. 95–109.

[6] A. Bauer, R. Goré, and A. Tiu, "A first-order policy language for history-based transaction monitoring," in *Proc. Int. Colloquium Theoretical Aspects Comput.*, 2009, pp. 96–111.

[7] D. Basin, F. Klaedtke, S. Marinovic, and E. Zălinescu, "Monitoring of temporal first-order properties with aggregations," in *Proc. Int. Conf. Runtime Verification*, 2013, pp. 40–58.

[8] D. Basin, F. Klaedtke, and E. Zălinescu, "Algorithms for monitoring real-time properties," in *Proc. Int. Conf. Runtime Verification*, 2012, pp. 260–275.

[9] A. Bauer, J.-C. Küster, and G. Vegliach, "From propositional to first-order monitoring," in *Proc. Int. Conf. Runtime Verification*, 2013, pp. 59–75.

[10] K. Havelund and G. Rosu, "Synthesizing monitors for safety properties," in *Proc. Int. Conf. Tools Algorithms Construction Anal. Syst.*, 2002, pp. 342–356.

[11] P. Thati and G. Roşu, "Monitoring algorithms for metric temporal logic specifications," *Electron. Notes Theoretical Comput. Sci.*, vol. 113, pp. 145–162, 2005.

[12] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: An open-source robot operating system," in *Proc. ICRA Workshop Open Source Softw.*, 2009, vol. 3, Art. no. 5.

[13] X. Du, Y. Liu, and A. Tiu, "Trace-length independent runtime monitoring of quantitative policies in LTL," in *Proc. Int. Symp. Formal Methods*, 2015, pp. 231–247.

[14] D. Bianculli, C. Ghezzi, and P. San Pietro, "The tale of SOLOIST: A specification language for service compositions interactions," in *Proc. Int. Workshop Formal Aspects Component Softw.*, 2013, pp. 55–72.

[15] M. Fitting, *First-Order Logic and Automated Theorem Proving*. Berlin, Germany: Springer, 1996.

[16] A. P. Sistla and O. Wolfson, "Temporal conditions and integrity constraints in active database systems," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 1995, pp. 269–280.

[17] N. Markey and P. Schnoebelen, "Model checking a path," in *Proc. Int. Conf. Concurrency Theory*, 2003, pp. 251–265.

[18] D. Basin, B. N. Bhatt, and D. Traytel, "Almost event-rate independent monitoring of metric temporal logic," in *Proc. Int. Conf. Tools Algorithms Construction Anal. Syst.*, 2017, pp. 94–112.

[19] D. Yang, A. Usynin, and J. W. Hines, "Anomaly-based intrusion detection for scada systems," in *Proc. Nuclear Plant Instrumentation Control Human-Mach. Interface Technol.*, 2006, pp. 12–16.

[20] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *Comput. Secur.*, vol. 28, no. 1, pp. 18–28, 2009.

[21] E. Evenchick, "Hopping on the CAN bus," in *Proc. Black Hat Asia*, 2015, pp. 1–31.

[22] C. Miller and C. Valasek, "Adventures in automotive networks and control units," *Def. Con.*, vol. 21, pp. 260–264, 2013.

[23] V. L. L. Thing and J. Wu, "Autonomous vehicle security: A taxonomy of attacks and defences," in *Proc. IEEE Int. Conf. Internet Things and IEEE Green Comput. Commun. and IEEE Cyber Phys. Social Comput. and IEEE Smart Data*, 2016, pp. 164–170.

[24] C. Miller and C. Valasek, "A survey of remote automotive attack surfaces," *Black Hat USA*, vol. 2014, p. 94, 2014.

[25] A. Taylor, N. Japkowicz, and S. Leblanc, "Frequency-based anomaly detection for the automotive CAN bus," in *Proc. World Congress Ind. Control Syst. Secur.*, 2015, pp. 45–49.

[26] H. M. Song, H. R. Kim, and H. K. Kim, "Intrusion detection system based on the analysis of time intervals of can messages for in-vehicle network," in *Proc. Int. Conf. Inf. Netw.*, 2016, pp. 63–68.

[27] G. Jin, L. Song, X. Shi, J. Scherpelz, and S. Lu, "Understanding and detecting real-world performance bugs," in *Proc. 33rd ACM SIGPLAN Conf. Program. Lang. Des. Implementation*, 2012, pp. 77–88.

[28] Y. Liu, C. Xu, and S.-C. Cheung, "Characterizing and detecting performance bugs for smartphone applications," in *Proc. 36th Int. Conf. Softw. Eng.*, 2014, pp. 1013–1024.

[29] Firefox frame rate meter tool webpage, 2011. [Online]. Available: http://blog.mozilla.org/devtools/tag/framerate-monitor/

[30] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," in *Proc. 41st Annu. Int. Symp. Comput. Archit.*, 2014, pp. 361–372.

[31] M. Seaborn, "Exploiting the DRAM rowhammer bug to gain kernel privileges," 2015. [Online]. Available: http://googleprojectzero.blogspot.sg/2015/03/exploiting-dram-rowhammer-bug-to-gain.html

[32] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer.js: A remote software-induced fault attack in JavaScript," *Int. Conf. Detection of Intrusions and Malware, and Vulnerability Assessment*, 2016, pp. 300–321.

[33] N. Herath and A. Fogh, "CPU hardware performance counters for security. BlackHat USA 2015 briefing," 2015.

[34] G. Meng, M. Patrick, Y. Xue, Y. Liu, and J. Zhang, "Securing android app markets via modelling and predicting malware spread between markets," *IEEE Trans. Inf. Forensics Secur.*, vol. 14, no. 7, pp. 1944–1959, Jul. 2019.

[35] G. Meng, Y. Xue, M. Chandramohan, A. Narayanan, Y. Liu, J. Zhang, and T. Chen, "Mystique: Evolving android malware for auditing anti-malware tools," in *Proc. 11th ACM Asia Conf. Comput. Commun. Secur.*, 2016, pp. 365–376.

[36] Z. Tang, M. Xue, G. Meng, C. Ying, Y. Liu, J. He, H. Zhu, and Y. Liu, "Securing android applications via edge assistant third-party library detection," *Comput. Secur.*, vol. 80, pp. 257–272, 2019.

[37] G. Meng, Y. Xue, Z. Xu, Y. Liu, J. Zhang, and A. Narayanan, "Semantic modelling of Android malware for effective malware comprehension, detection, and classification," in *Proc. 25th Int. Symp. Softw. Testing Anal.*, 2016, pp. 306–317.

[38] SMS rate limit, 2011. [Online]. Available: http://support.telerivet.com/customer/portal/articles/1205288-customizing-app-setttings

[39] S. Arzt, K. Falzon, A. Follner, S. Rasthofer, E. Bodden, and V. Stolz, "How useful are existing monitoring languages for securing android apps?" in *Proc. Annu. Airline Travel Payments Summit*, 2013, pp. 107–122.

[40] Rage against the cage, 2011. [Online]. Available: http://thesnkchrmr.wordpress.com/2011/03/24/rageagainstthecage

[41] Trace-length independent runtime monitoring of quantitative policies, 2018. [Online]. Available: https://sites.google.com/view/mtlcnt/home

[42] H.-M. Ho, J. Ouaknine, and J. Worrell, "Online monitoring of metric temporal logic," in *Proc. Int. Conf. Runtime Verification*, 2014, pp. 178–192.

[43] D. Basin, S. Krstić, and D. Traytel, "Almost event-rate independent monitoring of metric dynamic logic," in *Proc. Int. Conf. Runtime Verification*, 2017, pp. 85–102.

[44] F. Laroussinie, A. Meyer, and E. Petonnet, "Counting LTL," in *Proc. 17th Int. Symp. Temporal Representation Reasoning*, 2010, pp. 51–58.

[45] D. Bianculli, C. Ghezzi, S. Krstic, and P. San Pietro, "From SOLOIST to CLTLB (3): Checking quantitative properties of service-based applications," Politecnico di Milano-Dipartimento di Elettronica, Informazione e Bioingegneria, Tech. Rep. 2013.26, 2013.

[46] M. M. Bersani, D. Bianculli, C. Ghezzi, S. Krstić, and P. San Pietro, "SMT-based checking of SOLOIST over sparse traces," in *Proc. Int. Conf. Fundam. Approaches Softw. Eng.*, 2014, pp. 276–290.

[47] C. Colombo, A. Gauci, and G. J. Pace, "LarvaStat: Monitoring of statistical properties," in *Proc. Int. Conf. Runtime Verification*, 2010, pp. 480–484.

[48] B. D'Angelo, S. Sankaranarayanan, C. Sanchez, W. Robinson, B. Finkbeiner, H. B. Sipma, S. Mehrotra, and Z. Manna, "LOLA: Runtime monitoring of synchronous systems," in *Proc. 12th Int. Symp. Temporal Representation Reasoning*, 2005, pp. 166–174.

[49] H. Barringer, A. Goldberg, K. Havelund, and K. Sen, "Rule-based runtime verification," in *Proc. Int. Workshop Verification Model Checking Abstract Interpretation*, 2004, pp. 44–57.

[50] H. Barringer, D. Rydeheard, and K. Havelund, "Rule systems for run-time monitoring: From eagle to ruler," *J. Logic Comput.*, vol. 20, no. 3, pp. 675–706, 2010.

[51] K. Havelund, "Rule-based runtime verification revisited," *Int. J. Softw. Tools Technol. Transfer*, vol. 17, no. 2, pp. 1–28, 2012.

[52] B. Finkbeiner, S. Sankaranarayanan, and H. B. Sipma, "Collecting statistics over runtime executions," *Formal Methods Syst. Des.*, vol. 27, no. 3, pp. 253–274, 2005.

Xiaoning Du received the one BS degree in engineering from Fudan University, and the one BS degree in computer science from the University of Dublin. She is working toward the PhD degree in the School of Computer Science and Engineering, Nanyang Technological University. Her research interests are in security, software engineering and formal method, with a focus on the runtime monitoring and program analysis. She has published in top-tier software engineering conferences including FM, ICSE and FSE.

Alwen Tiu received the PhD degree in computer science and engineering from Pennsylvania State University, in 2004. He is a aenior lecturer with the Research School of Computer Science, Australian National University. He worked as a postdoctoral researcher at LORIA/INRIA Lorraine (2004-2005), prior to joining the Australian National University (ANU), in 2006 as a research fellow. In 2013-2017 he worked as an assistant professor at Nanyang Technological University, Singapore. He re-joined ANU in late 2017.

Kun Cheng received the BS degree from the School of Computer Science and Engineering, Beihang University, in 2012, Beijing, China. He is currently working toward the PhD degree at Beihang University. His research interests include system virtualization, cloud computing and real-time systems. He is a student member of both IEEE and the China Computer Federation (CCF).

Yang Liu received the PhD degree from National University Singapre, in 2010. He joined Nanyang Technological University (NTU) as a Nanyang assistant professor, in 2012. He is currently an associate professor and director of the Cybersecurity Lab in NTU. He specializes in software verification, security and software engineering. His research has bridged the gap between the theory and practical usage of formal methods and program analysis to evaluate the design and implementation of software for high assurance and security. By now, he has more than 250 publications in top tier conferences and journals. He has received a number of prestigious awards including MSRA fellowship, TRF fellowship, Nanyang assistant professor, Tan Chin Tuan fellowship, and 8 best paper awards in top conferences like ASE, FSE and ICSE.