Decentralized Asynchronous Crash-Resilient Runtime Verification

Borzoo Bonakdarpour¹, Pierre Fraigniaud², Sergio Rajsbaum³, David A. Rosenblueth³, and Corentin Travers⁴

1 McMaster University, Canada, borzoo@mcmaster.ca
2 CNRS and University Paris Diderot, France, pierref@irif.fr
3 UNAM, México, {rajsbaum,drosenbl}@unam.mx
4 University of Bordeaux, France, travers@labri.fr

Abstract

Runtime Verification (RV) is a lightweight method for monitoring the formal specification of a system during its execution. It has recently been shown that a given state predicate can be monitored consistently by a set of crash-prone asynchronous distributed monitors, only if sufficiently many different verdicts can be emitted by each monitor. We revisit this impossibility result in the context of LTL semantics for RV. We show that employing the four-valued logic RV-LTL will result in inconsistent distributed monitoring for some formulas. Our first main contribution is a family of logics, called LTL₂⁴⁺, that refines RV-LTL incorporating ₂⁴ truth values, for each  ≥ 0. The truth values of LTL₂⁴⁺ can be effectively used by each monitor to reach a consistent global set of verdicts for each given formula, provided  is sufficiently large. Our second main contribution is an algorithm for monitor construction enabling fault-tolerant distributed monitoring based on the aggregation of the individual verdicts by each monitor.

Introduction

Runtime Verification (RV) is a technique where a monitor process determines whether or not the current execution of a system under inspection complies with its formal specification. The state-of-the-art RV methods for distributed systems exhibit the following shortcomings. They (1) employ a central monitor, (2) employ several monitors but lack a systematic way to monitor formally specified properties of a system (e.g., [10–12]), or (3) assume a fault-free setting, where each individual monitor is resilient to failures [5, 7, 8, 15–17, 19]. Relaxing the latter assumption, that is, handling monitors subject to failures, poses significant challenges as individual monitors would become unable to agree on the same perspective of the execution, due to the impossibility of consensus [9]. Thus, it is unavoidable that individual monitors emit different local verdicts about the current execution, so that a consistent global verdict with respect to a correctness property can be constructed from these verdicts.

The necessity of using more than just the two truth values of Boolean logic is a known fact in the context of RV with a single monitor. For instance, RV-LTL [3] has four truth values  = {⊥, ⊤, ⊤p, ⊥p}. These values identify cases where a finite execution (1) permanently satisfies, (2) permanently violates, (3) presumably satisfies, or (4) presumably violates an LTL formula. For example, consider a request/acknowledge property, where a
Decentralized Asynchronous Crash-Resilient Runtime Verification

request $r_1$ is eventually responded by acknowledgement $a_1$, and $a_1$ should not occur before $r_1$; i.e., LTL formula $\varphi = G(\neg a_1 \land \neg r_1) \lor [(\neg a_1 U r_1) \land F a_1]$. In RV-LTL, a finite execution containing $r_1$ and ending in $a_1$ (i.e., the request has been acknowledged) yields the truth value ‘permanently satisfied’, whereas an execution containing only $r_1$ (i.e., the request has not yet been acknowledged) yields ‘presumably violated’.

Although RV-LTL can monitor $\varphi$ (see Fig. 1 for its monitor automaton) in a centralized setting, we show $B_4$ is not sufficient to consistently monitor a conjunction of two such formulas in a framework of several asynchronous unreliable monitors. Namely, the set of verdicts emitted by the monitors may not be sufficient to distinguish executions that satisfy the formula from those that violate it. Intuitively, this is because each monitor has only a partial view of the system under scrutiny, and after a finite number of rounds of communication among monitors, still too many different perspectives about the global system state remain.

In fact, it was proved in [12] using algebraic topology techniques [13] that fault-tolerant distributed monitoring requires that the individual verdicts are taken from a set whose size depends on the formula being monitored.

**Our results.** In this paper, we propose a framework for distributed fault-tolerant RV. To this end, we make a novel connection between RV and consensus in a failure-prone distributed environment by proposing a multi-valued temporal logic. This new logic is a refinement of RV-LTL. More specifically, we propose a family of $(2k + 4)$-valued logics, denoted $\text{LTL}_{2k+4}$, for $k \geq 0$. In particular, $\text{LTL}_{2k+4}$ coincides with RV-LTL when $k = 0$. The syntax of $\text{LTL}_{2k+4}$ is identical to that of LTL. Its semantics is based on $\text{F}LTL$ [14] and LTL$_3$ [4], two LTL-based finite trace semantics for RV. For each $k \geq 0$, the $k$th instance of the family has $2k + 4$ truth values, that intuitively represent a degree of certainty that the formula is satisfied. We characterize the formulas that when verified at run time with $\text{LTL}_{2k+4}$, no additional information is gained if they are verified with $\text{LTL}_{2k'+4}$, for a larger value $k'$. We present a monitor construction algorithm that generates a finite-state Moore machine for any given LTL formula and $k \geq 0$.

For example, for formula $\varphi = \varphi_1 \land \ldots \land \varphi_t$, where each $\varphi_i$ is an independent request/acknowledgement formula, $\text{LTL}_{2k+4}$ can be used to consistently monitor $\varphi_i$ whenever $k \geq t$. In particular, when $t = 2$, the set of truth values is $B = \{ \top_0, \bot_0, \top_1, \bot_1, \top_2, \bot_2, \top, \bot \}$. Moreover, formula $\varphi$ evaluates to: $\top_0$ (presumably true with the lowest degree of certainty) in a finite execution that does not contain neither $r_1$ nor $a_1$, then to $\bot_1$ in an extension where $r_1$ appears (presumably true with a higher degree of certainty), to $\top_1$ in an extension that includes both $r_1$ and $a_1$, to $\bot_2$ if $r_2$ appears, and finally to $\top$ (permanently true) in an execution that contains $r_1, a_1, r_2$, and $a_2$.

Our second contribution is an algorithm for fault-tolerant distributed RV, where the monitors are asynchronous wait-free processes that communicate with each other via a read/write shared-memory, and any of them can fail by crashing. (For simplicity we use this abstract model, which is well-understood [2, 13], and is known to be equivalent, with respect to task computability, to a message-passing model where less than half the processes can crash.) Each monitor gets a partial view of the system’s global state, communicates with the other monitors a fixed number of rounds, and then emits a verdict from $B_{2k+4}$. We show how, given any LTL formula and a large enough $k$, the truth values of $\text{LTL}_{2k+4}$ can be effectively used such that a set of verdicts collectively provided by the monitors can be mapped to the verdict computed by a centralized monitor that has full view of the system under inspection.

It follows from the general lower bound result in [12] that our algorithm is optimal, meaning that for any $k \geq 0$, there exists an LTL formula that cannot be monitored consistently in
LTL 2k+4, if k is not sufficiently large. Finally, we prove that the value of k is solely a function of the structure of the LTL formula.

**Related Work.** While there has been significant progress in sequential monitoring in the past decade, there has been less work devoted to distributed monitoring. Lattice-theoretic centralized and decentralized online predicate detection in distributed systems has been studied in [7, 15]. This line of work does not address monitoring properties with temporal requirements. This shortcoming is partially addressed in [17], but for offline monitoring. In [19], the authors design a method for monitoring safety properties in distributed systems using the past-time linear temporal logic (PLTL). In such a work, however, the valuation of some predicates and properties may be overlooked. This is because monitors gain knowledge about the state of the system by piggybacking on the existing communication among processes. That is, if processes rarely communicate, then monitors exchange little information and, hence, some violations of properties may remain undetected. Runtime verification of LTL for synchronous distributed systems where processes share a single global clock has been studied in [5, 8]. In [6], the authors introduce parallel algorithms for runtime verification of sequential programs. As already mentioned, our work is inspired by the research line of [10–12], the first one to study the effects of monitor failures in distributed RV. Distributed applications that can be runtime monitored with three opinions were studied in [11], and the number of opinions needed to runtime monitor set agreement was analyzed in [10]. More generally, [12] proves a tight lower bound on the number of opinions needed to monitor a property based on its alternation number. The goal of this paper is to give a formal semantics to the opinions studied in [10–12], and derive a framework in the actual formal context of runtime verification.

2 Background: Linear Temporal Logics for RV

Let AP be a set of atomic propositions and $\Sigma = 2^{AP}$ be the set of all possible states. A trace is a sequence $s_0s_1 \cdots$, where $s_i \in \Sigma$ for every $i \geq 0$. We denote by $\Sigma^*$ (resp., $\Sigma^\omega$) the set of all finite (resp., infinite) traces. Throughout the paper, we denote infinite traces by the letter $\alpha$, and finite traces by the letter $\alpha$. We denote the empty trace by $\epsilon$. For a finite trace $\alpha = s_0s_1 \cdots s_n$, $|\alpha|$ denotes its length, i.e., its number of states $n + 1$. Finally, by $\alpha^+$, we mean trace $s_is_{i+1} \cdots s_n$ of $\alpha$. We assume that the syntax and semantics of standard LTL is common knowledge.

**Example.** We use the following request/acknowledgement LTL formula throughout the paper to explain the concepts:

$$\phi_{ra} = G(\neg a \wedge \neg r) \lor [(\neg a U r) \wedge Fa]$$

That is (1) if a request is emitted (i.e., $r = true$), then it should eventually be acknowledged (i.e., $a = true$), and (2) an acknowledgement happens only in response to a request.

**Finite LTL (FLTL).** In the context of runtime verification, the semantics of LTL is not fully appropriate as it is defined over infinite traces. Finite LTL (FLTL, see [14]) allows us to reason about finite traces for verifying properties at run time. The syntax of FLTL is identical to that of LTL and the semantics is based on the truth values $B_2 = \{\top, \bot\}$. The semantics of FLTL for atomic propositions and Boolean operators are identical to those of LTL. We now recall the semantics of FLTL for the temporal operators. Let $\phi$, $\phi_1$, and $\phi_2$
be LTL formulas, \( \alpha = s_0 s_1 \cdots s_n \) be a non-empty finite trace, and \( \models_F \) denote satisfaction in \( \text{FLTL} \). We have
\[
[\alpha \models_F X \varphi] = \begin{cases} 
[\alpha^1 \models_F \varphi] & \text{if } \alpha^1 \neq \epsilon \\
\bot & \text{otherwise}
\end{cases}
\]
and
\[
[\alpha \models_F \varphi_1 U \varphi_2] = \begin{cases} 
\top & \text{if } \exists k \in [0,n]: ([\alpha^k \models_F \varphi_2]) \land (\forall \ell \in [0,k], [\alpha^\ell \models_F \varphi_1] = \top) \\
\bot & \text{otherwise}
\end{cases}
\]

To illustrate the difference between LTL and FLTL, let \( \varphi = \text{F}_p \) and \( \alpha = s_0 s_1 \cdots s_n \). If \( p \in s_i \) for some \( i \in [0,n] \), then we have \([\alpha \models_F \varphi] = \top\). Otherwise, \([\alpha \models_F \varphi] = \bot\), and this holds even if the program under inspection extends \( \alpha \) in the future to a state where \( p \) becomes true.

**Multi-valued LTLs.** As illustrated above, for a finite trace \( \alpha \), FLTL ignores the possible future extensions of \( \alpha \), when evaluating a formula. 3-valued LTL (\( \text{LTL}_3 \), see [4]) evaluates LTL formulas for finite traces with an eye on possible future extensions. In \( \text{LTL}_3 \), the set of truth values is \( \mathbb{B}_3 = \{ \top, \bot, ? \} \), where ‘\( \top \)’ (resp., ‘\( \bot \)’) denotes that the formula is permanently satisfied (resp., violated), no matter how the current execution extends, and ‘\( ? \)’ denotes an unknown verdict; i.e., there exist an extension that can falsify the formula, and another extension that can truthify the formula.

Now, let \( \alpha \in \Sigma^* \) be a non-empty finite trace. The truth value of an \( \text{LTL}_3 \) formula \( \varphi \) with respect to \( \alpha \), denoted by \([\alpha \models_3 \varphi]\), is defined as follows:
\[
[\alpha \models_3 \varphi] = \begin{cases} 
\top & \text{if } \forall \sigma \in \Sigma^\omega : \alpha \sigma \models \varphi \\
\bot & \text{if } \forall \sigma \in \Sigma^\omega : \alpha \sigma \not\models \varphi \\
? & \text{otherwise.}
\end{cases}
\]

\( \text{RV-LTL} \) [3], which we will denote in this paper \( \text{LTL}_4 \), refines the truth value \( ? \) into \( \bot_p \) and \( \top_p \). That is, \( \mathbb{B}_4 = \{ \top, \top_p, \bot_p, \bot \} \). More specifically, evaluation of a formula in \( \text{LTL}_4 \) agrees with \( \text{LTL}_3 \) if the verdict is \( \bot \) or \( \top \). Otherwise, (i.e., when the verdict in \( \text{LTL}_3 \) is \( ? \)), \( \text{LTL}_4 \) utilizes \( \text{FLTL} \) to compute a more refined truth value.

Now, let \( \alpha \in \Sigma^* \) be a finite trace. The truth value of an \( \text{LTL}_4 \) formula \( \varphi \) with respect to \( \alpha \), denoted by \([\alpha \models_4 \varphi]\), is defined as follows:
\[
[\alpha \models_4 \varphi] = \begin{cases} 
\top & \text{if } [\alpha \models_3 \varphi] = \top \\
\bot & \text{if } [\alpha \models_3 \varphi] = \bot \\
\top_p & \text{if } [\alpha \models_3 \varphi] = ? \land [\alpha \models_F \varphi] = \top \\
\bot_p & \text{if } [\alpha \models_3 \varphi] = ? \land [\alpha \models_F \varphi] = \bot
\end{cases}
\]

The \( \text{LTL}_4 \) monitor of a formula \( \varphi \) is the unique deterministic finite state machine \( \mathcal{M}_\varphi^4 = (\Sigma, Q, q_0, \delta, \lambda) \), where \( Q \) is a set of states, \( q_0 \) is the initial state, \( \delta : Q \times \Sigma \to Q \) is the transition function, and \( \lambda : Q \to \mathbb{B}_4 \), is a function such that:
\[
\lambda(\delta(q_0, \alpha)) = [\alpha \models_4 \varphi]
\]
for every finite trace \( \alpha \in \Sigma^* \). In [4], the authors introduce an algorithm that takes as input an LTL formula and constructs as output an \( \text{LTL}_4 \) monitor. For example, Fig. 1 shows the \( \text{LTL}_4 \) monitor for the request/acknowledgement formula \( \varphi_{ra} = \text{G}(\neg a \land \neg r) \lor ([\neg a \lor r] \land \text{Fa}) \).
In this section, we present a general computation model for asynchronous distributed wait-free monitoring. Throughout the rest of the paper, the system under inspection produces a finite trace $\alpha = s_0 s_1 \cdots s_k$, and is inspected with respect to an LTL formula $\varphi$ by a set $\mathcal{M} = \{M_1, M_2, \ldots, M_n\}$ of asynchronous distributed wait-free monitors.

Algorithm sketch: For every $j \in [0, k - 1]$, between each $s_j$ and $s_{j+1}$, each monitor, in a wait-free manner:

1. reads the value of propositions in $s_j$, which may result in a partial observation of $s_j$;
2. repeatedly communicates its partial observation with other monitors through a single-writer/multi-reader shared memory;
3. updates its knowledge resulting from the aforementioned communication, and
4. evaluates $\varphi$ and emits a verdict from $B_4$.

Since each monitor observes and maintains only a partial view of $s_j$, and since the monitors run asynchronously, different read/write interleavings are possible, where each interleaving may lead to a different collective set of verdicts emitted by the monitors in $\mathcal{M}$ for $s_j$. In Subsection 3.1, we formally introduce our notion of \textit{wait-free distributed monitoring}.

To ensure consistent distributed monitoring, one has to be able to map a collective set of verdicts of monitors (for any execution interleaving) to one and only one verdict of a centralized monitor that has the full view $s_j$. A necessary condition for this mapping is that, for every two finite traces $\alpha, \alpha' \in \Sigma^*$, if $[\alphawersym{=}F \varphi] \not\equiv [\alpha'wersym{=}F \varphi']$, then the monitors in $\mathcal{M}$ should compute different collective sets of verdicts for $\alpha$ and $\alpha'$, no matter what their initial partial observation and subsequent read/write interleavings are. We call this condition global consistency, described in detail in Subsection 3.2.

### 3.1 Wait-Free Distributed Monitoring

We consider a set $\mathcal{M} = \{M_1, M_2, \ldots, M_n\}$ of monitors, each observing a system under inspection. We assume that each monitor in $\mathcal{M}$ has only a partial view of the system under inspection.

- **Definition 1.** A \textit{partial state} is a mapping $S$ from the set $AP$ of atomic propositions to the set $\{\text{true, false, } \sharp\}$, where $\sharp$ denotes an unknown value.

When a state $s$ is reached in a finite trace, each monitor $M_i \in \mathcal{M}$, for $1 \leq i \leq n$, takes a sample from $s$, which results in obtaining a partial state. More formally:

- **Definition 2.** A \textit{sample} of a state $s \in \Sigma$ by monitor $M_i$ is a partial state $S^*_i$ such that, for all $ap \in AP$, we have: $\left(S^*_i(ap) = \text{true} \rightarrow ap \in s\right) \land \left(S^*_i(ap) = \text{false} \rightarrow ap \not\in s\right)$.

Definition 2 entails that, in a sample, if the value of an atomic proposition is not unknown, then the sampled value is consistent with state $s$. Thus, two monitors $M_i$ and $M_j$ cannot take inconsistent samples. That is, for any state $s$ and samples $S^*_i, S^*_j$, and for every $ap \in AP$, we have: $\left(S^*_i(ap) \not\equiv S^*_j(ap)\right) \rightarrow \left(S^*_i(ap) = \sharp \lor S^*_j(ap) = \sharp\right)$.

We say that a set of monitors \textit{cover} a state if the collection of partial views of these monitors covers the value of the all atomic propositions. Formally:

- **Definition 3.** A set $\mathcal{M} = \{M_1, M_2, \ldots, M_n\}$ satisfies state coverage for a state $s$ if and only if for every $ap \in AP$, there exists $M_i \in \mathcal{M}$ such that $S^*_i(ap) \not\equiv \sharp$.
Decentralized Asynchronous Crash-Resilient Runtime Verification

Algorithm 1: Behavior of Monitor $M_i$, for $i \in [1, n]$

<table>
<thead>
<tr>
<th>Data: LTL formula $\varphi$ and state $s_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result: a verdict from $\mathbb{B}_4$</td>
</tr>
<tr>
<td>1 initialize all elements of $LS_i[j]$ with $\bot$</td>
</tr>
<tr>
<td>2 $LS_i[j] \leftarrow S^i_0$; /* take sample from state $s_j$ */</td>
</tr>
<tr>
<td>3 for some fixed number of rounds do</td>
</tr>
<tr>
<td>4 $SM_i[j] \leftarrow p(LS_i[j])$; /* write (i.e., project) current knowledge in shared memory */</td>
</tr>
<tr>
<td>5 $LS_i[j] \leftarrow SM_i[j]$; /* take a snapshot of the shared memory */</td>
</tr>
<tr>
<td>6 emit $[x(LS_i[0]) \ldots x(LS_i[j])] \models \varphi$; /* evaluate $\varphi$ using extrapolation function */</td>
</tr>
</tbody>
</table>

Each monitor $M_i$ in $\mathcal{M}$ is a process, and the monitors run in the standard asynchronous wait-free read/write shared memory model [2]. Each monitor (1) runs at its own speed, that may vary along with time and (2) may fail by crashing (i.e., halt and never recover). We assume that up to $n-1$ monitors can crash, and thus a monitor never “waits” for another monitor (since this may cause a livelock). Every monitor that does not fail is required to output; i.e., to emit a verdict. Hence, a distributed algorithm in this settings consists for each monitor in a bounded sequence of read/write accesses to the shared memory at the end of which a verdict is emitted. If the number of possible inputs is bounded, the lengths of such sequences are globally bounded. We thus assume without loss of generality that each monitor accesses the shared memory a fixed number of times before emitting a verdict [13].

More specifically, for every state $s_j$ in $\alpha = s_0 s_1 \cdots s_k$, each monitor $M_i$ maintains a so-called local snapshot $LS_i[j]$ consisting of $n$ registers, one per monitor in $\mathcal{M}$ (i.e., the local snapshot is organized as an array of registers). We denote by $LS_i[j]$ the local register of monitor $M_i$ associated with monitor $M_i$ for state $s_j$. Each register has $|AP|$ elements, one for each atomic proposition in $AP$. The monitors in $\mathcal{M}$ communicate by means of shared memory. The structure of the shared memory $SM$ is similar to monitor local snapshots: for each state $s_j$, $SM[j]$ consists of $n$ atomic registers, one per monitor, and each register has $|AP|$ elements one for each atomic proposition (i.e., single-writer/multiple-reader (SWMR) registers). Thus, for state $s_j$, each monitor $M_i$ can read the entire content of $SM[j]$, but can only write into register $SM_i[j]^1$.

The distributed monitoring algorithm. Each monitor $M_i \in \mathcal{M}$, $i \in [1, n]$, runs Algorithm 1 that we shall now describe in detail. For any given new state $s_j$, Monitor $M_i$ first initializes all registers of its local snapshot to $\bot$ (cf. Line 1). Then, $M_i$ takes a sample from state $s_j$ (cf. Line 2). Recall from Def. 2 that the value of an atomic proposition in a sample is either true, false, or $\bot$. The set of values in the sample is copied in local register $LS_i[j]$. After sampling, each monitor $M_i$ executes a sequence of write/snapshot actions (cf. Lines 4 and 5) for some a priori known number of times, that we detail next$^2$.

In Line 4, $M_i$ computes its knowledge about each proposition $ap$, given its content of $LS_i[j]$, and atomically writes it into its associated register $SM_i[j]$ in the shared memory. Function $p = (p_{ap})_{ap \in AP}$ where $p_{ap} : \{true, false, \bot\}^n \rightarrow \{true, false, \bot\}$ is the projection

---

1 We assume that each monitor is aware of the change of state of the system under inspection. Thus, for a state $s_j$, a monitor $M_i$ reads and writes in the associated local and shared memory locations, i.e., $LS_i[j]$ and $SM_i[j]$.

2 Algorithm 1 uses snapshot operations for the sake of simplifying the presentation. We emphasize that atomic snapshots can be implemented using atomic read/write operations in a wait-free manner [1].
function defined by

$$p_{ap}(v_1, \ldots, v_n) = \begin{cases} 
\text{true} & \text{if } \exists i \in [1, n] : v_i = \text{true} \\
\text{false} & \text{if } \exists i \in [1, n] : v_i = \text{false} \\
\top & \text{otherwise}
\end{cases}$$

Given a local snapshot $LS_i$, $p(LS_i)$ denotes the partial state obtained by applying $p_{ap}$ to $n$ values of each atomic proposition $ap$ in $LS_i$. Notice that, based on Definition 2, $p$ cannot receive contradicting values for an atomic proposition.

In Line 5, $M_i$ reads all the registers in $SM[j]$, and copies them into $LS_i[j]$, in a single atomic step. Finally, after a certain number of iterations, the for-loop ends, and $M_i$ evaluates $\varphi$ and emits a verdict based on the content of its local snapshots $LS_i[0] \cdots LS_i[j]$ (cf. Line 6). To evaluate $\varphi$ on $s_0s_1 \cdots s_j$, monitor $M_i$ needs to compute one and only one Boolean value for each atomic proposition. To this end, we assume that for each atomic proposition $ap \in AP$, all monitors are provided with the same extrapolation function $x_{ap}$ allowing them to associate a Boolean value to each atomic proposition, even if its truth value is unknown at some monitors. Such an extrapolation function must satisfy the following consistency condition.

\begin{definition}
Given $ap \in AP$, a function $x_{ap} : \{true, false, \top\}^n \to \{true, false\}$ is an extrapolation function if and only if $p_{ap}(v_1, \ldots, v_n) \neq \top \Rightarrow x_{ap}(v_1, \ldots, v_n) = p_{ap}(v_1, \ldots, v_n)$.
\end{definition}

Given a local snapshot array $LS$, $x(LS)$ denotes the state obtained by applying $x_{ap}$ to $n$ values of each atomic proposition $ap$ in $LS$. Also given a state $s_j$, by $[LS_i[j]]$, we mean the local snapshot of monitor $M_i$ obtained after termination of the for loop in Algorithm 1.

**Example.** Let $\mathcal{M} = \{M_1, M_2\}$ and consider the formula for two requests and acknowledgements:

$$\varphi_{r_{a_2}} = \left( G(\neg a_1 \land \neg r_1) \lor [\neg a_1 U r_1] \land F a_1 \right) \land \left( G(\neg a_2 \land \neg r_2) \lor [\neg a_2 U r_2] \land F a_2 \right)$$

Fig. 2 shows different execution interleavings of monitors $M_1$ and $M_2$ when running Algorithm 1 from states $s_0 = \{r_1, a_1\}$ and $s'_0 = \{r_1, a_1, r_2\}$. Based on the order of monitor write-snapshot actions: $M_1, M_2$ (resp., $M_2, M_1$) denotes the case where monitor $M_1$ (resp., $M_2$) executes a write-snapshot before monitor $M_2$ (resp., $M_1$) does, and $M_1||M_2$ denotes the case where monitors $M_1$ and $M_2$ execute their write-snapshot actions concurrently. In case of $s_0$, after executing Line 2 of Algorithm 1, monitor $M_1$’s sample, i.e., the local snapshot $LS_1[0]$, consists of $S_1^{s_0}(r_1) = true$, $S_1^{s_0}(a_1) = \top$, and $S_1^{s_0}(r_2) = S_1^{s_0}(a_2) = false$. Moreover, initially, $M_1$ has no knowledge of $M_2$’s sample. Monitor $M_2$’s sample from $s_0$, i.e., the local snapshot $LS_2[0]$, consists of $S_2^{s_0}(r_1) = S_2^{s_0}(a_1) = true$, $S_2^{s_0}(r_2) = \top$, and $S_2^{s_0}(a_2) = false$ while it initially has no knowledge of $M_1$’s sample. Likewise, for state $s'_0$, Fig. 2 shows different local snapshots by $M_1$ and $M_2$. Given two values $v_1$ and $v_2$, we define (an arbitrary) extrapolation function as follows:

$$x_{ap}(v_1, v_2) = \begin{cases} 
\text{true} & \text{if } (v_1 = \text{true}) \lor (v_2 = \text{true}) \\
\text{false} & \text{otherwise}
\end{cases}$$

where $ap \in \{a_1, r_1, a_2, r_2\}$. Finally, starting from $s_0$, if (1) the for loop of Algorithm 1 terminates after 1 communication round, and (2) the interleaving is $M_1, M_2$, then $x([LS_2[0]]) = \{r_1, a_1\}$, and evaluation of $\varphi_{r_{a_2}}$ by $M_2$ in LTL4 results in $[x([LS_2[0]])] \models 4 \varphi_{r_{a_2}} = \top$. 

\begin{concur}

B. Bonakdarpour, P. Fraigniaud, S. Rajsbaum, D. A. Rosenblueth, and C. Travers

16:7

CONCUR 2016

\end{concur}
Lemma 7 for details).

Definition 6. Let $\varphi$ be an LTL formula, $\alpha$ be a finite trace in $\Sigma^*$, and $m$ be any of its monitor traces. We say that $m$ satisfies **global consistency** in LTL$_4$ iff there exists a function $\mu : 2^\Sigma \rightarrow \{\top, \bot\}$ such that $\mu(m|\alpha|_{\sim-1}) = [\alpha \models_\varphi \varphi]$. We now show that LTL$_4$ is unable to consistently monitor all LTL formulas. To see this, observe that in Fig. 2, the shaded collective verdicts $m_0$ and $m'_0$ are both equal to $\{\bot_p, \top_p\}$, but $[s_0 \models_\varphi \varphi] \neq [s'_0 \models_\varphi \varphi]$. This clearly does not meet global consistency (see the proof of Lemma 7 for details).
Theorem 8. Not all LTL formulas can be consistently monitored by a distributed monitor with traces in $LTL_4$, even if monitors satisfy state coverage, even if no monitors crash during the execution of the monitor.

Lemma 7 holds for an arbitrary number of communication rounds as well. Indeed, additional rounds of communication will not result into reaching global consistency. This impossibility result is a direct consequence of the main lower bound in [12], which can be rephrased as follows.

Theorem 8. Not all LTL formulas can be consistently monitored by a distributed monitor with traces in $LTL_4$, even if monitors satisfy state coverage, even if no monitors crash during the execution of the monitor, and even if the monitors perform an arbitrarily large number of communication rounds.

In the next section, we revisit the notion of alternation number introduced in [12] in order to identify formulas that can be monitored by $LTL_4$, and to design a multi-valued logic to monitor LTL formulas that cannot be monitored in $LTL_4$.

### 4 Alternation Number

We now define the notion of alternation number [12] in the context of LTL. In the next section, we shall show that the alternation number essentially determines an upper bound on the number of truth values needed to ensure consistency in distributed monitoring.

Let $\alpha \in \Sigma^*$ be a finite trace, $\alpha'$ be the longest proper prefix of $\alpha$, and $\varphi$ be an LTL formula. We set the alternation number of $\varphi$ with respect to $\alpha$ as follows:

$$AN(\varphi, \alpha) = \begin{cases} 0 & \text{if } |\alpha| = 1 \\ AN(\varphi, \alpha') + 1 & \text{if } (|\alpha| \geq 2) \land (|\alpha'| \models F \varphi) \neq (\alpha \models F \varphi) \\ AN(\varphi, \alpha') & \text{otherwise} \end{cases}$$

The alternation number with respect to infinite traces is defined as follows. Let $\sigma \in \Sigma^\omega$ be an infinite trace. If for any prefix $\alpha$ of $\sigma$, there exists a finite extension $\alpha'$, such that $AN(\varphi, \alpha) < AN(\varphi, \alpha')$, then we set $AN(\varphi, \sigma) = \infty$. Otherwise, we set $AN(\varphi, \sigma) = AN(\varphi, \alpha)$ where $\alpha$ is
such that there does not exist a finite extension \( \alpha' \) of \( \alpha \) such that \( AN(\varphi, \alpha) < AN(\varphi, \alpha') \). Finally, the alternation number of \( \varphi \) with respect to a (possibly infinite) set \( A \) of traces is

\[
AN(\varphi, A) = \max \{ AN(\varphi, \alpha) \mid \alpha \in A \}
\]

**Definition 9.** The alternation number of an LTL formula \( \varphi \) is \( AN(\varphi) = AN(\varphi, \Sigma^*) \).

**Examples.** We have \( AN(Gp) = 1 \) because, in any finite trace \( \alpha \), if the valuation of \( Gp \) in FLTL changes from \( \top \) to \( \bot \), then, in no extension of \( \alpha \) this value can change back to \( \top \). We have \( AN(G(\neg r \rightarrow F a)) = \infty \), because any occurrence of \( r \land \neg a \) evaluates the formula to \( \bot \), and a subsequent occurrence of \( a \) evaluates the formula to \( \top \) in FLTL. We have \( AN(\varphi_{ra}) = AN(G(\neg a \land \neg r) \lor [(\neg a U r) \land Fa]) = 2 \). Indeed, as long as \( \neg r \land \neg a \) is true throughout a trace \( \alpha \), we have \( [\alpha \models_F \varphi_{ra}] = \top \). When \( r \land \neg a \) becomes true, the valuation of \( \varphi_{ra} \) changes to \( \bot \). If \( a \) becomes true subsequently, then \( \varphi_{ra} \) evaluates to \( \top \). By the same type of arguments, we show \( AN(\varphi_{ra}) = 4 \).

Interestingly, the alternation number of an LTL formula \( \varphi \) can be determined from the structure of its LTL\(_4\) monitor automaton \( M^\varphi_4 \).

**Theorem 10.** Let \( \varphi \) be an LTL formula. The alternation number of \( \varphi, AN(\varphi) \), is equal to the length of the longest alternating walk in its LTL\(_4\) monitor \( M^\varphi_4 \).

**Example.** Let \( \varphi_{ra} = G(\neg a \land \neg r) \lor [(\neg a U r) \land Fa] \). We have \( AN(\varphi_{ra}) = 2 \), and one can check on Fig. 1 that indeed the length of the longest alternating walk in \( M^\varphi_{ra} \) is 2.

## 5 Multi-Valued LTL for Consistent Distributed Monitoring

In this section, we introduce a family of multi-valued logics (called LTL\(_{2k+4}\)), for every \( k \geq 0 \), and relate it to the notion of alternation number. For every \( k \geq 0 \), the syntax of LTL\(_{2k+4}\) is identical to that of LTL. We present the semantics, monitor synthesis, and proof of global consistency of LTL\(_{2k+4}\) in Subsections 5.1, 5.2, and 5.3, respectively.

### 5.1 Semantics of LTL\(_{2k+4}\)

**Truth values.** The semantics of LTL\(_{2k+4}\) refines LTL\(_4\). LTL\(_{2k+4}\) employs the following set of \( 2k + 4 \) truth values:

\[
\mathcal{B}_{2k+4} = \{ \bot, \top, \bot_0, \ldots, \bot_k, \top_0, \ldots, \top_k \}
\]

Intuitively, for \( i \in [0, k] \), truth value \( \bot_i \) means *possibly false* with *degree of certainty* \( i \), and truth value \( \top_i \) means *possibly true* with degree of certainty \( i \), while \( \top \) and \( \bot \) have the same meaning as their LTL\(_3\) counterparts. Thus, LTL\(_{2k+4}\) coincides with LTL\(_4\) for \( k = 0 \). Consider a non-empty finite trace \( \alpha = s_0s_1 \cdots s_n \) in \( \Sigma^* \). We denote the valuation of a formula \( \varphi \) with respect to \( \alpha \) in LTL\(_{2k+4}\) by \( [\alpha \models_{2k+4} \varphi] \). Since, for any \( i \in [0, k] \), \( \bot_i \) implies ‘?’ in LTL\(_3\), we require that \( [\alpha \models_{2k+4} \neg \varphi] = \bot_i \rightarrow [\alpha \models_3 \neg \varphi] = ? \land [\alpha \models_F \varphi] = \top \). The latter conjunct is to relate \( \bot_i \) with the valuation of \( \alpha \) in FLTL. Likewise, we require that, for any \( i \in [0, k] \): \( [\alpha \models_{2k+4} \neg \varphi] = \top_i \rightarrow [\alpha \models_3 \neg \varphi] = ? \land [\alpha \models_F \varphi] = \bot \). We determine the degree of certainty of \( [\alpha \models_{2k+4} \varphi] \) inductively according to the judgement rules below, where \( \alpha' = s_0s_1 \cdots s_{n-1} \).

Observe that the degree of certainty does not change if the FLTL valuation does not change in \( \alpha' \) and \( \alpha \), or change from \( \bot \) to \( \top \). On the contrary, the degree of certainty does change if the FLTL valuation changes in \( \alpha' \) and \( \alpha \) from \( \bot \) to \( \top \), respectively.
Then, in each step annotated by monitor transitions (q, a, q'), where q is annotated by T_{i-1} and q' is annotated by T_{i-1}, and adding transitions (q, a, q), where q is annotated by T_{i-1} (Lines 5-10).

**Theorem 12.** Let \( \varphi \) be an LTL formula, and let \( M^{\varphi}_{2k+4} = (\Sigma, Q, q_0, \delta, \lambda) \) be its LTL_{2k+4} monitor such as constructed by Algorithm 2. Then, for any non-empty finite trace \( \alpha \in \Sigma^* \), we have \( \lambda(\bar{q}_0, \alpha) = [\alpha \models_{2k+4} \varphi] \).

5.2 Monitorability and Monitor Synthesis for LTL_{2k+4}

Pnueli and Zaks [18] characterize an LTL formula \( \varphi \) as *monitorable* for a finite trace \( \alpha \), if \( \alpha \) can be extended to one that can be evaluated with respect to \( \varphi \) at run time. That is, an LTL formula \( \varphi \) is *monitorable* in LTL_{3} if and only if: \( \forall \alpha \in \Sigma^* : \exists \alpha' \in \Sigma^* : [\alpha' \models_{=3} \varphi] \neq \).

We stick to the same definition for LTL_{2k+4}.

**Definition 11.** Let \( \varphi \) be an LTL formula. The LTL_{2k+4} monitor of \( \varphi \) is the unique deterministic finite state machine \( M^{\varphi}_{2k+4} = (\Sigma, Q, q_0, \delta, \lambda) \), where \( Q \) is a set of states, \( q_0 \) is the initial state, \( \delta : Q \times \Sigma \rightarrow Q \) is the transition function, and \( \lambda : Q \rightarrow \mathbb{B}_{2k+4} \), such that, for every non-empty finite trace \( \alpha \in \Sigma^* \), we have \([\alpha \models_{2k+4} \varphi] = \lambda(\delta(q_0, \alpha)) \).

Algorithm 2 constructs LTL_{2k+4} monitors. Intuitively, our algorithm creates \( k+1 \) copies of LTL_{4} [3] monitors by invoking Function ConstructMonitor, and cascades them in such a way that incrementing the degree of certainty is implemented as prescribed by our definition of LTL_{2k+4}. Observe that for a given value \( i \in [0, k] \), Function ConstructMonitor renames truth value \( T_{i-1} \) (respectively, \( \bot_i \)) in LTL_{4} to \( T_i \) (respectively, \( \bot_i \)) (see Lines 14-18). Cascading the monitors in Algorithm 2 is as follows. Initially, we generate an LTL_{4} monitor for \( k = 0 \) (Line 1). Then, in each step \( i \in [1, k] \) of the for-loop, we generate a new LTL_{4} monitor (cf. Line 3). We ensure incrementing the degree of certainty by removing monitor transitions (q, a, q'), where q is annotated by T_{i-1} and q' is annotated by \( \bot_i \), and adding transitions (q, a, q), where q is annotated by T_{i-1} (Lines 5-10).
5.3 Monitoring Algorithm and Global Consistency in LTL$_{2k+4}$

**Monitoring Algorithm** Let $\alpha = s_0 s_1 \cdots s_k$ be a finite trace in $\Sigma^*$. As discussed in Section 3, for any state $s_j$, where $j \in [0,k]$, each monitor runs Algorithm 1 and emits a verdict. In order to employ LTL$_{2k+4}$ and ensure consistency, each monitor has to compute the highest possible degree of certainty by considering all possible monitor communication interleavings that result in state $s_j$. Formally, the set of all interleavings that reach a state $s \in \Sigma$ is the set of sequences of partial states defined as follows:

$$I_s = \{ S_0 S_1 \cdots S_l \mid (\forall ap \in AP : S_0(ap) = \top) \land (S_l = s) \land \\forall i \in [0,l] : \forall ap \in AP : (S_i(ap) \neq \top) \rightarrow (\forall m \in (i,l) : S_i(ap) = S_m(ap))\}$$

Now, for state $s_j$ in $\alpha$ and formula $\varphi$, a monitor $M_i$ computes $AN(\varphi, I_{S_j(I_{LS,I}))})$. This can be done by running each trace in $I_{S_j(I_{LS,I}))}$ on the LTL$_{2k+4}$ monitor of $\varphi$. This is indeed the key idea to ensure global consistency.

**Observation 13.** For any state $s \in \Sigma$ and LTL formula $\varphi$, we have $AN(\varphi, I_s) \leq AN(\varphi)$.

**Example.** Fig. 4 shows how monitors $M_1$ and $M_2$ evaluate formula $\varphi_{ra2}$ in LTL$_{2k+4}$ with $k = 2$. Observe that the two sets of verdicts that were not distinguishable in Fig. 2 (i.e., $m_0 = m'_0 = \{\top, \bot\}$) are now distinguishable (i.e., $m_0 = \{\bot, \top\}$, while $m'_0 = \{\top, \bot\}$), as we are now using 8 truth values instead of just 4. The ability of monitoring a formula in LTL$_{2k+4}$ for a given $k \geq 0$ is strongly related to the alternation number of the formula.

**Main Results.** The following identifies an upper-bound on the number of truth values needed to monitor any LTL formula.

**Theorem 14.** An LTL formula $\varphi$ can consistently be monitored by a wait-free distributed monitor in LTL$_{2k+4}$, if

$$k \geq \left\lceil \frac{1}{2}(\min(AN(\varphi), n) - 1) \right\rceil$$

where $n$ is the number of monitors.

An immediate consequence of Theorem 14 is for computing $\mu$ (Definition 6) for LTL$_{2k+4}$. For a set $m \in B_{2k+4}$, one can compute $\mu(m)$ by identifying the supremum of $m$, for the total order $\bot_0 < \top_0 < \bot_1 < \top_1 < \ldots < \bot_k < \top_k$. It is straightforward to observe that such a $\mu$ results in global consistency for LTL$_{2k+4}$. Also, notice that Theorem 14 is best possible. It matches the following generalization of Theorem 8. The proof is similar to the lower bound of [12].

**Theorem 15.** For each $k \geq 0$, there is an LTL formula $\varphi$ that cannot be consistently monitored by a wait-free distributed monitor in LTL$_{2k+4}$, if

$$k < \left\lceil \frac{1}{2}(\min(AN(\varphi), n) - 1) \right\rceil$$

where $n$ is the number of monitors.

6 Conclusion and Future Work

In this paper, we proposed a family of multi-valued logics LTL$_{2k+4}$, each one with $2k + 4$ truth values, for fault-tolerant distributed RV, refining existing finite LTL semantics. We
presented an idealized setting where a set of unreliable monitors emit consistent verdicts in LTL_{2k+4} about the correctness of the system under inspection, if \( k \) is sufficiently large.

We note that wait-free computing is a powerful and simple abstraction to model and reason about distributed algorithms. All results in this paper can theoretically be transformed to more practical refinements such as message passing frameworks. Of course, further research is needed to develop such transformations. From a more practical perspective, it would be interesting to relax the timing model enabling monitors to observe, communicate, and emit verdicts between any two global states; to study frameworks for message passing systems, and to address more severe, even Byzantine failures.

### 7 Acknowledgement

This work was partially sponsored by Canada NSERC Discovery Grant 418396-2012 and NSERC Strategic Grants 430575-2012 and 463324-2014, UNAM-PAPIIT Grant IN107714, PASPA-DGAPA-UNAM and Conacyt grants 221341 and 261225, as well as the French State, managed by the French National Research Agency (ANR) in the frame of the "Investments for the future" Programme IdEx Bordeaux - CPU (ANR-10-IDEX-03-02).
References