Decentralized Asynchronous Crash-resilient Runtime Verification

BORZOO BONAKDARPOUR, Michigan State University, U.S.A.
PIERRE FRAIGNIAUD, Université de Paris and CNRS, France
SERGIO RAJSBAUM and DAVID ROSENBLUETH, Universidad Nacional Autónoma de México, México
CORENTIN TRAVERS, University of Bordeaux and CNRS, France

Runtime verification 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 observing the system, only if each monitor can emit verdicts taken from a *large enough* finite set. We revisit this impossibility result in the concrete context of linear-time logic (LTL) semantics for runtime verification, that is, when the correctness of the system is specified by an LTL formula on its execution traces. First, we show that monitors synthesized based on the 4-valued semantics of LTL (RV-LTL) may result in inconsistent distributed monitoring, even for some simple LTL formulas. More generally, given any LTL formula φ, we relate the number of different verdicts required by the monitors for consistently monitoring φ, with a specific structural characteristic of φ called its *alternation number*. Specifically, we show that, for every $k \geq 0$, there is an LTL formula φ with alternation number k that cannot be verified at runtime by distributed monitors emitting verdicts from a set of cardinality smaller than $k + 1$. On the positive side, we define a family of logics, called *distributed* LTL (abbreviated as DLTL), parameterized by $k \geq 0$, which refines RV-LTL by incorporating $2k + 4$ truth values. Our main contribution is to show that, for every $k \geq 0$, every LTL formula φ with alternation number k can be consistently monitored by distributed monitors, each running an automaton based on a $(2\lceil k/2 \rceil + 4)$-valued logic taken from the DLTL family.

CCS Concepts: • **Theory of computation → Verification by model checking**; **Distributed computing models**; • **Computing methodologies → Distributed algorithms;**

Additional Key Words and Phrases: Runtime verification, distributed computing, fault-tolerant verification, wait-free tasks, distributed monitoring, model checking, temporal logic, linear-time logic

An extended abstract of a preliminary version of this article appeared in the proceedings of the 27th International Conference on Concurrency Theory (CONCUR), August 23–26, 2016, Québec City, Canada.
Borzoo Bonakdarpour Supported by NSF FMitF Award 1917979 and SaTC Award 1813388.
Pierre Fraigniaud Supported by the ANR projects DESCARTES and FREDDA, and by the INRIA project GANG.
Sergio Rajsbaum Supported by the UNAM-PAPIIT IN106520 grant.
Corentin Travers Supported by the ANR projects DESCARTES and FREDDA.

Authors' addresses: B. Bonakdarpour, Michigan State University, 428 S. Shaw Lane, Engineering Building, East Lansing, MI 48824-1226; P. Fraigniaud, Université de Paris and CNRS, IRIF, Université Paris Cité, Case 7014, 75205 Paris CEDEX 13, France; S. Rajsbaum, Instituto de Matematicas, Universidad Nacional Autonoma de Mexico (UNAM), Ciudad Universitaria, D.F. 04510, Mexico; D. Rosenblueth, Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas, Universidad Nacional Autónoma de México, Apdo 20-126, 01000 México, D.F., México; C. Travers, University of Bordeaux and CNRS, 351 cours de la Libération 33400 Talence, France.

0004-5411/2022/10-ART34 $15.00
https://doi.org/10.1145/3550483

ACM Reference format:
Borzoo Bonakdarpour, Pierre Fraigniaud, Sergio Rajsbaum, David Rosenblueth, and Corentin Travers. 2022. Decentralized Asynchronous Crash-resilient Runtime Verification. *J. ACM* 69, 5, Article 34 (October 2022), 31 pages.
https://doi.org/10.1145/3550483

1 INTRODUCTION

1.1 Context

Runtime verification 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 runtime verification methods exhibit the following shortcomings: Either they classically employ a central monitor or they employ several monitors but assume a *fault-free* setting, where each individual monitor is resilient to failures [9, 13, 14, 21, 25–27, 31]. Relaxing the latter assumption, that is, handling several monitors subject to failures, poses significant challenges, as these monitors would become unable to agree on the same perspective of the execution, due to the impossibility of consensus [17]. Thus, it is unavoidable that these monitors emit different *individual* verdicts about the current execution, so a consistent *global* verdict with respect to a correctness property can be constructed from these verdicts. Concretely, the two truth values of Boolean logic may be insufficient for allowing each monitor to express a wide spectrum of individual verdicts.

The necessity of using more than just the two truth values of Boolean logic is actually a known fact in the context of runtime verification, even with a single monitor. For instance, the **linear temporal logic (LTL)** [28] has been one of the most widely used specification languages to express the requirements of computing systems.[1] While LTL is a widely accepted language to reason about infinite execution traces, its three-valued semantics (denoted by LTL_3) [8] is a logic on finite execution traces with three truth values in:

$$\mathbb{B}_3 = \{\top, \bot, ?\}.$$

These truth values respectively express whether, given the finite trace observed so far, an LTL formula is permanently satisfied, or permanently violated, or whether the observation is inconclusive. Likewise, RV-LTL [7] has four truth values in

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

These values respectively identify cases where a finite execution permanently satisfies, permanently violates, presumably satisfies, or presumably violates a given LTL formula. For example, consider a request/acknowledge property, where a request r should be eventually responded to by acknowledgment a, and a should not occur before r. Formally, an LTL formula for the request/acknowledge property is

$$\varphi_{ra} = \Box(\neg a \wedge \neg r) \ \vee \ [(\neg a \ \mathcal{U} \ r) \ \wedge \ \Diamond a]. \tag{1}$$

This formula holds if either $\Box(\neg a \wedge \neg r)$ holds (i.e., there is no request and no acknowledgment), or $(\neg a \, \mathcal{U} \, r) \wedge (\Diamond a)$ holds (i.e., a request is made at present or some future state and an acknowledgment is made after this request in the future). In RV-LTL, a finite execution containing r and ending in a (i.e., the request has been acknowledged) yields the truth value "permanently satisfied," whereas an execution containing only r (i.e., the request has not yet been acknowledged) yields

[1] We refer the reader to Reference [16], where the author formalized 54 commonly used requirements as LTL formulas. We also note that the area of runtime verification mainly focuses on specification languages that are *trace-based*. This is due to the fact that at runtime, monitors can realistically observe only a finite execution trace. The semantics of temporal logics such as CTL is based on computation trees and is not suitable for runtime monitoring.

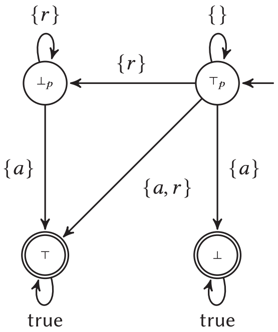


Fig. 1. RV-LTL monitor of φ_{ra}.

"presumably violated." Although RV-LTL can monitor φ_{ra} in a centralized setting (see Figure 1 for its monitor automaton), it is not powerful enough to monitor a conjunction of two such formulas in a framework of two asynchronous unreliable monitors:

$$\varphi_{ra2} = (\Box(\neg a_1 \wedge \neg r_1) \vee [(\neg a_1 \ \mathcal{U}\ r_1) \wedge \Diamond a_1]) \wedge (\Box(\neg a_2 \wedge \neg r_2) \vee [(\neg a_2 \ \mathcal{U}\ r_2) \wedge \Diamond a_2]).$$

Indeed, the set of verdicts emitted by the monitors is not sufficient to distinguish executions that satisfy the formula from those that violate it. Intuitively (we will formally establish this result further in the text), 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 of the global system state remain. For instance, the case where a monitor M_1 has observed a partial trace containing only r_1 (for which it should output $\bot_p \in \mathbb{B}_4$) is distinct from the case where M_1 has observed a partial trace containing only a_1. However, M_1 should not output $\bot$ in this latter case (of course, it should not output either $\top$ or $\top_p$), because it may well be the case that another monitor M_2 has observed r_1, yet M_1 is not aware of this observation, because of asynchrony and unreliability.

In fact, it was recently proved in Reference [20] that even deciding whether a *single* system state satisfies some given Boolean predicate, using a distributed set of asynchronous crash-prone monitors, requires that the individual verdicts be taken from a set whose size depends on the predicate under scrutiny. Although this size cannot exceed the number n of monitors, it is proved that, for any $k \in [0, n]$, there are Boolean predicates on system states that require verdicts taken from a set of size at least $k + 1$. A matching upper bound is also presented in Reference [20]. In this article, we extend the preliminary results in Reference [20] to the setting of distributed monitoring execution traces whose correctness is expressed by LTL formulas, and we provide distributed monitors defined in terms of finite automata corresponding to multi-valued logics.

1.2 Our Results

In this article, we propose a framework for distributed fault-tolerant runtime verification, where the monitors are asynchronous and subject to crash. A monitor that crashes stops executing its code and does nothing afterwards. To this end, we introduce 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 by DLTL, for *distributed* LTL. In particular, DLTL with $k = 0$ coincides with RV-LTL. The syntax of DLTL is identical to the one of LTL, and its semantics is based, as RV-LTL, on both FLTL [24] and LTL$_3$ [8], which are two LTL-based finite trace semantics for runtime monitoring. For each $k \geq 0$, the kth instance of the family DLTL has $2k + 4$ truth values

$$\mathbb{B}_{2k+4} = \{\top, \bot, \top_0, \bot_0, \top_1, \bot_1, \ldots, \top_k, \bot_k\}.$$

The index i of a logical value intuitively represents a *degree of certainty* that the formula is satisfied ($\top_i$) or not ($\bot_i$). In a nutshell, we characterize the formulas that can be monitored at runtime by a in DLTL$_k$, but cannot be distributedly monitored in DLTL$_{k-1}$.

More specifically, our first contribution (Theorem 5.2) is a *lower bound* on the cardinality of the set of values used by each monitor for expressing its local verdict. We revisit the result in Reference [20] and show that this lower bound can be expressed in terms of a particular characteristic of the LTL formula under consideration, called its *alternation number*. Roughly, the alternation number of an LTL formula φ is the maximum, taken over all finite traces $\alpha = \alpha_0\alpha_1 \cdots \alpha_n$, that the valuation of φ can alternate in the finite semantics of LTL. In other words, the alternation number of φ is the maximum number of times φ can change its truth value in FLTL by gaining more and more information about the truth values of the atomic propositions characterizing the current system's global state α_n. As opposed to Reference [20], this number of changes depends not only on the current state α_n of the system, but also on the sequence of preceding states (i.e., those in α). We show that, for every $k \geq 0$, there is an LTL formula φ with alternation number k that cannot be distributedly monitored by monitors emitting verdicts from a set of cardinality smaller than $k + 1$.

Our second contribution (Theorem 6.5) is a concrete mechanism for fault-tolerant distributed runtime verification. Each monitor gets a partial view of the system's global state, communicates with the other monitors, and then emits a verdict in DLTL using $2\lceil k/2 \rceil + 4$ truth values, where k is the alternation number of the LTL formula under scrutiny. The sets of verdicts collectively provided by the monitors are in one-to-one correspondence with the RV-LTL verdicts that would be computed by a centralized monitor with a full view of the system. In view of our lower bound, our algorithm is essentially optimal in terms of the number of verdicts emitted by the distributed monitors (up to a small additive constant). Our mechanism is concrete in the sense that we present a monitor construction algorithm that generates a finite-state Moore machine that, for any LTL formula φ, computes the alternation number k of φ, and constructs the DLTL automaton enabling to distributedly monitor φ using $2\lceil k/2 \rceil + 4$ logical values.

We emphasize that we do not make an assumption on whether the system under scrutiny is centralized or distributed. In fact, this has no impact on our results and, hence, the type of the system is abstracted away.

We note that there is long literature on what is *monitorable*. The classic definition Reference [29] is that an LTL formula is monitorable if any prefix can be extended to some other finite prefix that evaluates to a permanently false or true verdict. In this sense, all *safety* and *co-safety* formulas are monitorable. However, not all monitorable formulas are either safety or co-safety. A liveness formula such as $\Box\Diamond p$ is not monitorable, intuitively because one cannot observe p infinitely often within a finite prefix at run time. Having said this, the above notion of monitorability is not relevant to our results in this article. First, observe that the request/acknowledgment formula is neither safety no co-safety but is monitorable. The issue here is that even for such a formula RV-LTL is not sufficient to consistently monitor the formula due to the partial observability of the monitors.

1.3 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 References [13, 25]. This line of work does not address monitoring properties with temporal requirements. This shortcoming is partially addressed in References [27, 30], but for offline monitoring. In Reference [31], 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. These techniques, however, assume perfect monitors that are not subject to faults.

Runtime monitoring of LTL formulas for synchronous distributed systems where processes share a single global clock has been studied in References [9, 14]. In Reference [10], the authors introduce parallel algorithms for runtime verification of sequential programs. Our work is inspired by the research line initiated in References [18–20]. Reference [19] pioneered the investigation of distributed decision in the context of asynchronous fault-tolerant distributed computing and characterized the Boolean predicates on system states that can be distributedly monitored with verdicts chosen from sets of two or three values. The follow-up contribution [20] extended this characterization to verdicts chosen from a set of k values, for any $k \geq 2$, and Reference [18] analyzed the specific case of monitoring the Boolean predicates on system states corresponding to checking the correctness of k-set agreement tasks.

1.4 Organization

The rest of the article is organized as follows: Section 2 presents the preliminary concepts. We introduce our model of computation for distributed monitoring in Section 3. Then, in Section 4, we show why the power of RV-LTL is insufficient to deal with fault-tolerant distributed monitoring. The notion of alternation number is presented in Section 5, while its impact on the design of DLTL is discussed in Section 6. Finally, we make concluding remarks and discuss future work in Section 7.

2 BACKGROUND

We recall basic concepts related to LTL and its finite semantics for runtime verification.

2.1 Linear Temporal Logic (LTL)

Let AP be a set of *atomic propositions* and $\Sigma = 2^{\mathrm{AP}}$ be the set of all possible *states*. A *trace* is a sequence $s_0 s_1 \cdots$, where $s_i \in \Sigma$ for every $i \geq 0$. We denote by Σ^* (respectively, Σ^ω) the set of all finite (respectively, infinite) traces. We denote the empty trace by ϵ. For a finite trace $\alpha = s_0 s_1 \cdots s_k$, $|\alpha|$ denotes its *length*, that is, its number of states, i.e., $k + 1$. Also, for $\alpha = s_0 s_1 \cdots s_k$, by α^i, we mean trace $s_i s_{i+1} \cdots s_k$ of α.

The syntax and semantics of ***linear temporal logic*** **(LTL)** [28] are defined for infinite traces. The syntax is defined by the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc\varphi \mid \varphi \,\mathcal{U}\, \varphi,$$

where $p \in \mathrm{AP}$, and where $\bigcirc$ and $\mathcal{U}$ are the "next" and "until" temporal operators. We view other propositional and temporal operators as abbreviations, that is, $\mathrm{true} = p \vee \neg p$, $\mathrm{false} = \neg\mathrm{true}$, $\varphi \rightarrow \psi = \neg\varphi \vee \psi$, $\varphi \wedge \psi = \neg(\neg\varphi \vee \neg\psi)$, $\Diamond\varphi = \mathrm{true} \,\mathcal{U}\, \varphi$ (*finally* φ), and $\Box\varphi = \neg\Diamond\neg\varphi$ (*globally* φ).

The infinite-trace semantics of LTL is defined as follows: Let $\sigma = s_0 s_1 s_2 \cdots \in \Sigma^\omega$, let $i \geq 0$, and let $\models$ denote the *satisfaction*.

$$\begin{array}{lll}
\sigma, i \models p & \Longleftrightarrow & p \in s_i \\
\sigma, i \models \neg\varphi & \Longleftrightarrow & \sigma, i \not\models \varphi \\
\sigma, i \models \varphi_1 \vee \varphi_2 & \Longleftrightarrow & \sigma, i \models \varphi_1 \text{ or } \sigma, i \models \varphi_2 \\
\sigma, i \models \bigcirc\varphi & \Longleftrightarrow & \sigma, i+1 \models \varphi \\
\sigma, i \models \varphi_1 \,\mathcal{U}\, \varphi_2 & \Longleftrightarrow & \exists k \geq i : \sigma, k \models \varphi_2 \text{ and } \forall j \in [i, k) : \sigma, j \models \varphi_1
\end{array}$$

Also, $\sigma \models \varphi$ holds if and only if $\sigma, 0 \models \varphi$ holds. For instance, the *request/acknowledgment* LTL formula in Equation (1) specifies that, first, if a request r is emitted, then such a request should eventually be acknowledged by a, and, second, an acknowledgment happens only in response to a request.

2.2 Logics for Runtime Verification

In the context of runtime verification, the semantics of LTL is not fully appropriate, as it is defined over infinite traces. Before we delve into the details, we note that many distributed programs are not-terminating (e.g., databases, internet services, blockchains, web servers, content delivery). However, the goal of runtime monitoring is to evaluate the health of a system by only observing *finite* behaviors of the system. In some cases, the monitor is able to issue a verdict that generalizes to any infinite extension (e.g., permanently false and true verdicts). In this sense, the monitor can inspect the health of a program regardless of whether it is terminating or non-terminating.

2.2.1 Finite LTL. **Finite LTL (FLTL)** [24] allows us to reason about finite traces for verifying properties at runtime. The syntax of FLTL is identical to that of LTL. The semantics of FLTL for both atomic propositions and Boolean operators are identical to those of LTL. FLTL employs two truth values to evaluate a formula with respect to a finite trace, denoted by $\mathbb{B}_2 = \{\bot, \top\}$. We now recall the semantics of FLTL for the temporal operators. Let φ, φ_1, and φ_2 be LTL formulas, let $\alpha = s_0 s_1 \cdots s_n$ be a non-empty finite trace, and let $\models_F$ denote satisfaction in FLTL. We have:

$$[\alpha \models_F \bigcirc\, \varphi] = \begin{cases} [\alpha^1 \models_F \varphi] & \text{if } \alpha^1 \neq \epsilon \\ \bot & \text{otherwise,} \end{cases}$$

and

$$[\alpha \models_F \varphi_1 \ \mathcal{U} \ \varphi_2] = \begin{cases} \top & \text{if } \exists k \in [0, n] : ([\alpha^k \models_F \varphi_2] = \top) \ \wedge \ (\forall \ell \in [0, k), [\alpha^\ell \models_F \varphi_1] = \top) \\ \bot & \text{otherwise.} \end{cases}$$

To illustrate the difference between LTL and FLTL, consider formula $\varphi = \Diamond p$ and finite trace $\alpha = s_0 s_1 \cdots s_n$. If $p \in s_i$ for some $i \in [0, n]$, then we have $[\alpha \models_F \varphi] = \top$. However, if $p \notin s_i$ for every $i \in [0, n]$, then $[\alpha \models_F \varphi] = \bot$, and this holds even if α is extended to another finite sequence including a state where p holds.

2.2.2 Three-valued Semantics for LTL. As illustrated in the previous subsection, FLTL ignores the possible future extensions of finite traces when evaluating a formula. Three-valued LTL (LTL$_3$) [8] also evaluates LTL formulas for finite traces, but with an eye on possible extensions. In LTL$_3$, the set of truth values is $\mathbb{B}_3 = \{\top, \bot, ?\}$, where $\top$ (respectively, $\bot$) denotes that the formula is *permanently* satisfied (respectively, violated), no matter how the current trace extends, and "?" denotes an unknown verdict—i.e., there exists an extension that can falsify the formula, and another extension that can truthify the formula. Let $\alpha \in \Sigma^*$ be a non-empty finite trace. The truth value of an LTL$_3$ formula φ with respect to α, denoted by $[\alpha \models_3 \varphi]$, is defined as follows:

$$[\alpha \models_3 \varphi] = \begin{cases} \top & \text{if } \forall \sigma \in \Sigma^\omega : \alpha\sigma \models \varphi \\ \bot & \text{if } \forall \sigma \in \Sigma^\omega : \alpha\sigma \not\models \varphi \\ ? & \text{otherwise.} \end{cases}$$

For example, consider formula $\varphi = \Box p$ and a finite trace $\alpha = s_0 s_1 \cdots s_n$. If $p \notin s_i$ for some $i \in [0, n]$, then $[\alpha \models_3 \varphi] = \bot$. That is, the formula is permanently violated. Now, consider formula $\varphi = \Diamond p$ and a finite trace $\alpha = s_0 s_1 \cdots s_n$. If $p \notin s_i$ for all $i \in [0, n]$, then $[\alpha \models_3 \varphi] =?$. This is because there exist infinite extensions to α that can satisfy or violate φ in the infinite semantics of LTL.

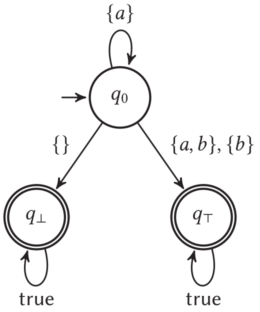


Fig. 2. Ltl$_3$ monitor for $\varphi = a \ \mathcal{U} \ b$.

Definition 2.1. The Ltl$_3$ *monitor* for a formula φ is the unique deterministic finite-state machine

$$\mathcal{M} = (\Sigma, Q, q_0, \delta, \lambda),$$

where Q is the set of states, q_0 is the initial state, $\delta : Q \times \Sigma \to Q$ is the transition function, and $\lambda : Q \to \mathbb{B}_3$ is a function such that

$$\lambda\Big(\delta(q_0, \alpha)\Big) = [\alpha \models_3 \varphi]$$

for every finite trace $\alpha \in \Sigma^*$.

For example, Figure 2 shows the monitor automaton for formula $\varphi = a \ \mathcal{U} \ b$. The function λ for this automaton is as follows: $\lambda(q_0) = ?, \lambda(q_\bot) = \bot$, and $\lambda(q_\top) = \top$.

2.2.3 Four-valued Semantics for LTL (RV-LTL). The four-valued logic rv-ltl [7] refines the truth value "?" into $\bot_p$ and $\top_p$. That is, its set of verdicts is $\mathbb{B}_4 = \{\top, \top_p, \bot_p, \bot\}$. More specifically, evaluation of a formula in rv-ltl agrees with ltl$_3$ if the verdict is $\bot$ or $\top$. Otherwise, (i.e., when the verdict in ltl$_3$ is ?), rv-ltl utilizes fltl to compute a more refined truth value. Let $\alpha \in \Sigma^*$ be a finite trace. The truth value of an rv-ltl formula φ with respect to α, denoted by $[\alpha \models_4 \varphi]$, is defined as follows:

$$[\alpha \models_4 \varphi] = \begin{cases} \top & \text{if } [\alpha \models_3 \varphi] = \top \\ \bot & \text{if } [\alpha \models_3 \varphi] = \bot \\ \top_p & \text{if } [\alpha \models_3 \varphi] = ? \ \wedge \ [\alpha \models_F \varphi] = \top \\ \bot_p & \text{if } [\alpha \models_3 \varphi] = ? \ \wedge \ [\alpha \models_F \varphi] = \bot \end{cases}.$$

Definition 2.2. The rv-ltl *monitor* of a formula φ is the unique deterministic finite-state machine

$$\mathcal{M} = (\Sigma, Q, q_0, \delta, \lambda),$$

where Q is the 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 $\alpha \in \Sigma^*$.

An algorithm that takes as input an ltl formula and constructs as output the rv-ltl monitor is described in Reference [8]. For example, Figure 1 shows the rv-ltl monitor for the request/acknowledgment formula in Equation (1).

Remark. We note that the sizes of RV-LTL and LTL$_3$ monitors are exponential in the size of the input LTL formula. However, since the size of formulas is typically small, the size of corresponding monitors after determinization and minimization is not expected to be large (usually a handful of states).

3 DISTRIBUTED FAULT-TOLERANT MONITORING

In this section, we present a general computation model for asynchronous distributed fault-tolerant monitoring.

3.1 General Objective

Throughout the rest of the article, 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 φ by a set $\mathcal{M} = \{M_1, M_2, \ldots, M_n\}$ of monitors. The monitors run *asynchronously* and are subject to *crash* failures. When a monitor crashes, it stops functioning, i.e., does not perform any computation step, and will never recover. For the sake of simplifying the presentation, we assume that the monitors exchange information by atomic read/write accesses to a shared memory. Indeed, our focus is to measure the impact of distributed monitoring, not to deal with the subtleties of complex communication media, and, hence, we choose the *wait-free* distributed computing model, which is well understood [6]. Moreover, this model is known to be equivalent, with respect to task computability, to the message-passing model, under the weak assumption that fewer than half the monitors can crash [3].

To compare the power and limitations of distributed monitoring with those of centralized monitoring, we assume that the monitors perform their observation of the system, their computation, and their emission of verdicts reflecting their vision of the current trace $\alpha = s_0 s_1 \cdots s_k$ w.r.t. some LTL formula φ, before the trace is extended to αs_{k+1}. In other words, the distributed monitors have time to observe, compute, and output *in between any two global steps* of the system execution. This allows us to compare the behavior of the distributed monitor with the behavior of a centralized event-triggered monitor observing the global execution of the system.

Informally, we aim at designing distributed monitors whose outputs enable to infer the verdicts that would be produced by a centralized monitor on the same execution trace. Specifically, we will compare our distributed monitors with a centralized monitor producing verdicts in RV-LTL. That is, assuming that the distributed monitors choose their verdicts from a set V, they must be able to map the sets of verdicts produced by the monitors to the truth values in $\mathbb{B}_4 = \{\top, \top_p, \bot_p, \bot\}$ produced by a (centralized) RV-LTL automaton monitoring the system, and this mapping

$$\mu : 2^V \rightarrow \mathbb{B}_4$$

must guarantee the *soundness* condition that, for every finite trace α, if the distributed monitors produce a set $m \in 2^V$ of verdicts for α, then

$$\mu(m) = [\alpha \models_4 \varphi]. \tag{2}$$

Note that m is a set of verdicts. Indeed, each monitor observes and maintains only a partial view of the system, and so two monitors may have different perspectives on the correctness of the system. Moreover, since the monitors run asynchronously, different read/write interleavings are possible, where each interleaving may lead to a different collective set m of verdicts emitted by the monitors for the same system state.

In the remaining of the section, we formally specify distributed fault-tolerant monitoring.

3.2 LTL on Partial Traces

In the centralized setting, recall from Section 2 that a state of the system is an element of 2^{AP}. We will use the notation $\{\mathsf{true}, \mathsf{false}\}^{|\mathsf{AP}|}$, specifying which atomic propositions are satisfied, and which ones are not satisfied in a given state. However, in a distributed setting, each monitor in $\mathcal{M}$ has only a *partial view* of the system under inspection, and it may be able to observe the truthfulness of only a subset of atomic propositions, so the value of the remaining propositions are unknown to the monitor. This leads us to the definition of *partial* states, and partial traces (see also References [11, 12]). We fix the notation $s[p]$ to denote the "value" of proposition p in state s (i.e., from the set $\{\mathsf{true}, \mathsf{false}\}$). We use the same notation for partial states and propositions.

Definition 3.1. Let $\widehat{\Sigma} = \{\mathsf{true}, \mathsf{false}, \natural\}^{|\mathsf{AP}|}$ where $\natural$ denotes an *unknown* value. A *partial state* is an element of $\widehat{\Sigma}$, and a *partial trace* is an element of $\widehat{\Sigma}^* \cup \widehat{\Sigma}^\omega$. Given a partial state $\hat{s}$, a state s is a *completion* of $\hat{s}$ if, for every $p \in \mathsf{AP}$, $s[p] \in \{\mathsf{true}, \mathsf{false}\}$, and

$$(\hat{s}[p] \neq \natural) \;\Rightarrow\; (s[p] = \hat{s}[p]).$$

A trace α is a completion of a partial trace $\hat{\alpha}$ if $|\alpha| = |\hat{\alpha}|$ and, for every $i \geq 0$, the ith state of α is a completion of the ith partial state of $\hat{\alpha}$.

We denote by $\mathsf{cmpl}(\hat{\alpha})$ the set of all traces α completing the partial trace $\hat{\alpha}$. Then, for every finite partial trace $\hat{\alpha}$, we set

$$[\hat{\alpha} \models_3 \varphi] = \begin{cases} \top & \text{if } \forall \alpha \in \mathsf{cmpl}(\hat{\alpha}), \forall \sigma \in \Sigma^\omega : \alpha\sigma \models \varphi \\ \bot & \text{if } \forall \alpha \in \mathsf{cmpl}(\hat{\alpha}), \forall \sigma \in \Sigma^\omega : \alpha\sigma \not\models \varphi \\ ? & \text{otherwise.} \end{cases} \tag{3}$$

When a state s is reached in a finite trace, each monitor in $\mathcal{M}$ takes a *sample* from s, which results in obtaining a partial state. In a sample, if the value of an atomic proposition is known, then the sampled value is consistent with state s, so the actual state is a completion of any of its samples.

Definition 3.2. A *sample* of a state $s \in \Sigma$ is a partial state $\hat{s} \in \widehat{\Sigma}$ such that, for every $p \in \mathsf{AP}$,

$$(\hat{s}[p] \neq \natural) \;\Rightarrow\; (\hat{s}[p] = s[p]).$$

We assume that two monitors M and M' cannot take inconsistent samples. That is, if $\hat{s}$ and $\hat{s}'$ are two samples of a state s by monitors M and M', respectively, then we assume that, for every $p \in \mathsf{AP}$,

$$(\hat{s}[p] \neq \hat{s}'[p]) \;\Rightarrow\; (\hat{s}[p] = \natural \,\vee\, \hat{s}'[p] = \natural).$$

We say that a set of monitors *covers* a state if the collection of partial views of these monitors covers the value of the all atomic propositions in s. A set $\mathcal{M}$ of monitors satisfies *state coverage* for a state s if, for every $p \in \mathsf{AP}$, there exists $M \in \mathcal{M}$ whose sample $\hat{s}$ satisfies $\hat{s}[p] \neq \natural$. Unfortunately, distributed monitoring with monitors subject to crash failures is subject to an important limitation: State coverage cannot be guaranteed. Indeed, even if it is guaranteed that $\mathcal{M}$ initially satisfies state coverage, the presence of crashes may result in this property no longer being true during the course of execution of the system. This follows from the fact that $\mathcal{M}' = \{M_i \mid i \in I\}$ may not satisfy state coverage for $I \subset [1, n]$, even if $\mathcal{M} = \{M_i \mid i \in [1, n]\}$ satisfies state coverage, because the monitors M_i, where $i \in [1, n] \setminus I$, have crashed.

Since state coverage cannot be guaranteed, one must also specify the correctness of partial traces in FLTL so monitors can emit non-trivial verdicts even on partial traces. In this article, we do so via an *extrapolation function* allowing to associate a Boolean value with each atomic proposition, even if its truth value is unknown.

Definition 3.3. An *extrapolation function* is a function $\mathbf{x} = (\mathbf{x}_p)_{p \in \mathsf{AP}}$, where

$$\mathbf{x}_p : \{\mathsf{true}, \mathsf{false}, \natural\} \to \{\mathsf{true}, \mathsf{false}\}$$

satisfies $\mathbf{x}_p(\mathsf{true}) = \mathsf{true}$ and $\mathbf{x}_p(\mathsf{false}) = \mathsf{false}$.

Given an extrapolation function $\mathbf{x}$, for every finite (partial) trace $\hat{\alpha} = \hat{s}_0\hat{s}_1 \cdots \hat{s}_k$, we define

$$[\hat{\alpha} \models_{F,\mathbf{x}} \varphi] := [\mathbf{x}(\hat{s}_0)\mathbf{x}(\hat{s}_1) \cdots \mathbf{x}(\hat{s}_k) \models_F \varphi]. \tag{4}$$

In the following, we assume that all the monitors in $\mathcal{M}$ are using the *same* extrapolation function $\mathbf{x}$. Note that, once LTL$_3$ and FLTL have been both extended to partial traces, the extension of RV-LTL to partial traces directly follows:

$$[\hat{\alpha} \models_4 \varphi] = \begin{cases} \top & \text{if } [\hat{\alpha} \models_3 \varphi] = \top \\ \bot & \text{if } [\hat{\alpha} \models_3 \varphi] = \bot \\ \top_p & \text{if } [\hat{\alpha} \models_3 \varphi] = \,?\, \wedge\ [\hat{\alpha} \models_{F,\mathbf{x}} \varphi] = \top \\ \bot_p & \text{if } [\hat{\alpha} \models_3 \varphi] = \,?\, \wedge\ [\hat{\alpha} \models_{F,\mathbf{x}} \varphi] = \bot \end{cases}.$$

Having extended LTL$_3$ and FLTL to partial traces, we can therefore refine our objective by revisiting Equation (2), rephrased as

$$\mu(m) = [\hat{\alpha} \models_4 \varphi],$$

where $\hat{\alpha}$ is the partial trace of an actual trace $\alpha = s_0s_1s_2 \cdots s_k$, defined as the sequence of partial states $\hat{s}_i$ of s_i resulting from the unions of all the samples of s_i taken by the monitors, $\mathcal{M} = \{M_1, M_2, \ldots, M_n\}$, and m is the set of verdicts returned by the monitors after having observed s_k.

Remark. The choice of the extrapolation function $\mathbf{x}$ used to extend FLTL to partial traces has no impact on our setting. Therefore, in the following, for simplifying the notations, and for the sake of improving readability, we shall no longer use the "ˆ" symbol for distinguishing traces from partial traces, and we shall no longer specify extrapolation using $\mathbf{x}$. The reader must solely remember that, from this point on, any mention of LTL$_3$ refers to the semantics of Equation (3), and any mention of FLTL refers to the semantics of Equation (4).

3.3 A Generic Algorithm for Distributed Monitoring

3.3.1 Wait-free Computing. Each monitor is a process, and the monitors run in the standard *asynchronous read/write shared memory* model [6]. Each monitor runs at its own speed, which may vary along with time and may fail by *crashing* (i.e., halt and never recover). We assume no bound on the number of monitors that can crash, and thus a monitor never "waits" for another monitor, since this may cause a livelock (a process waiting for an event that will never occur). This model of computation is thus often referred to as *wait-free* shared memory computing. Every monitor that does not crash is required to output, i.e., in the context of this article, to emit a monitoring verdict. A distributed algorithm in this setting consists, for each process, of a bounded sequence of read/write accesses to the shared memory, at the end of which an output is produced, i.e., a verdict is emitted. If the number of possible inputs is bounded (which is the case in the setting of monitoring an LTL formula as every state is of bounded size), then the lengths of such read/write sequences are bounded. We thus assume, without loss of generality, that each monitor accesses the shared memory a *fixed* arbitrarily large number of times before emitting a verdict (see Reference [22] for more details).

3.3.2 Wait-free Snapshots. Consider an array SM of single-writer/multi-reader registers, where process (monitor) M_i can write to SM[i], and can read the register SM[j] of any other processes M_j. Programming using such an array can be significantly simplified, using instead *snapshot* operations. A process M_i can still write only to SM[i], but it can read all the array SM in a single atomic snapshot operation. If it would be possible to stop all other processes temporarily, to allow

M_i to read one-by-one all registers, then M_i could obtain a snapshot SM. However, in a wait-free system, this is not allowed.

Remarkably, it is possible to implement a snapshot operation wait-free, allowing all other processes to continue executing their operations, possibly even writing and reading concurrently. Many wait-free atomic snapshot implementations have been proposed, on top of read/write registers, e.g., References [1, 2, 5, 23]. Furthermore, implementations of snapshots on top of a message passing system have also been proposed [4, 15]. Such implementations have high computation cost, but the main purpose of this article is to study feasibility, not efficiency. Our algorithms could be implemented by simply replacing the snapshot operation by the read/write algorithm implementing the snapshots (or potentially even the implementation on top of a message passing system as mentioned above), without compromising the correctness of the results in the rest of the article.

Thus, for the sake of simplifying the presentation, all our algorithms use atomic *snapshot* operations. That is, we assume that a monitor can acquire the entire memory SM in a single atomic "global read" instruction. A *view* of the shared memory SM is merely the result of a snapshot.

Using snapshots does not artificially strengthen the power of distributed monitoring, but considerably simplifies the presentation of the algorithms and their analysis. Indeed, snapshots are ordered by inclusion, because they return the contents of the shared memory that existed at some point in time, between the invocation of the snapshot operation, and the moment the operation returns. Thus, two snapshot operations may return the same view if they took effect simultaneously. Otherwise, one returns a view at some point in time, and the other a view of the contents of the shared memory at a later time. In this sense, we have the following statement:

Lemma 3.4 (Attiya et al. [5]). *The snapshots are ordered by inclusion, i.e., for any two monitors M_i and M_j, and any two snapshots of these monitors returning two views w_i and w_j of the shared memory, we have either $w_i \subseteq w_j$ or $w_i \supseteq w_j$.*

3.3.3 A Generic RV Algorithm. As mentioned earlier, RV is concerned with verifying finite traces. Distributed monitoring works as follows: Let $s_0 s_1 s_2 \cdots s_k$ be a finite trace under scrutiny. We perform a sequence of phases, where each phase $j \in [0, k]$ consists in evaluating the correctness of the trace $s_0 s_1 \ldots s_j$. That is, at phase j, each monitor receives a sample from state s_j, which forms its input, then performs a fixed number R of access to the shared memory, after which it produces its verdict regarding the trace $s_0 s_1 \cdots s_j$. We now describe this process in more detail.

Each monitor $M_i \in \mathcal{M}$, where $i \in [1, n]$, is provided with a local memory, lm_i. The shared memory is denoted by SM. For the sake of establishing a strong lower bound, we consider protocols that are not subject to any constraints in terms of how much data can be stored, and how much data can be transferred at once during a read (snapshot) or a write. In other words, we consider *full knowledge* protocols [22]. (Note, however, that our upper bound will be shown efficient in terms of both memory storage and bandwidth utilization).

Both the shared memory and the local memories are organized in levels, where, for every $j \in [0, k]$, both the jth level $\mathsf{SM}[j]$ and $\mathsf{lm}_i[j]$, $i \in [1, n]$ store data used when considering state s_j of the monitored trace. Moreover, the jth level of the shared memory is organized in $R \cdot n$ registers, where n is the number of monitors, and R denotes the number of rounds of read/write instructions. Specifically, $\mathsf{SM}[j][r, i]$ stores data written by M_i during its rth write. Similarly, the local memory of M_i is organized in $R+2$ registers, where $\mathsf{lm}[j][0]$ stores the sample of s_j by M_i, and, for $1 \leq r \leq R$, $\mathsf{lm}[j][r]$ stores data extracted by M_i from the shared memory during its rth read. (An extra level $\mathsf{lm}[j][R+1]$ is used for synchronization, as explained below.) We assume that all variables are initialized to $\natural$.

Each monitor $M_i \in \mathcal{M}$, $i \in [1, n]$, runs Algorithm 1 that we detail next. First, before sampling s_j, each monitor takes a snapshot of the shared memory. This is to make sure that all the

ALGORITHM 1: Generic behavior of Monitor M_i, for $i \in [1, n]$.

```
  Data: LTL formula $\varphi$ and state $s_j$, $j \geq 0$
  Result: a verdict from some fixed set $V$
1 if $j > 0$ then
2     $\mathsf{lm}_i[j-1][R+1] \leftarrow \mathsf{SM}[j-1]$;    /* $M_i$ snapshots the $(j-1)$th level of shared memory */
3 $\mathsf{lm}_i[j][0] \leftarrow$ sample$_i(s_j)$;    /* $M_i$ takes sample from state $s_j$ */
4 for $r = 1$ to $R$ do
5     $\mathsf{SM}[j][r, i] \leftarrow \mathsf{lm}_i[j][r-1]$;    /* $M_i$ writes its current knowledge in shared memory */
6     $\mathsf{lm}_i[j][r] \leftarrow \mathsf{SM}[j]$;    /* $M_i$ takes a snapshot of the shared memory */
7 emit a verdict in $V$;    /* $M_i$ decides based on the knowledge accumulated in $\mathsf{lm}_i$ */
```

monitors share the same information about the partial trace resulting from the global observation of $s_0 s_1 \cdots s_{j-1}$. Indeed, recall that it is assumed that all non-faulty monitors sample, compute, and emit their verdict in between every two consecutive steps of the system. Thus, when M_i starts considering s_j, all non-faulty monitors have emitted their verdict about $s_0 s_1 \ldots s_{j-1}$. In particular, the values of all the atomic propositions of s_{j-1} that are covered by the set of non-faulty monitors have been written in shared memory when M_i samples s_j. The instructions performed in Lines 1 and 2 allow M_i to get all such values. As a consequence, for any two monitors M_i and $M_{i'}$ monitoring $s_0 s_1 \ldots s_j$, it holds

$$\forall p \in [0, j-1], \quad \mathsf{lm}_i[p][R+1] = \mathsf{lm}_{i'}[p][R+1].$$

That is, they agree on $s_0 s_1 \cdots s_{j-1}$.

For any given new state s_j, monitor M_i takes a sample from state s_j (cf. Line 3), which is stored in local memory $\mathsf{lm}_i[j][0]$, at the 0th level. (Recall that the value of an atomic proposition in a sample is either true, false, or $\natural$.) After sampling, each monitor M_i executes a sequence of write/snapshot actions (cf. Lines 5 and 6) for some *a priori* known number of times R. More precisely, in Line 5, at the rth iteration, M_i atomically *writes* all its knowledge accumulated so far, i.e., during the $r-1$ previous rounds of read/write instructions. This knowledge is stored at the rth level of the shared memory, in the register dedicated to data from monitor M_i. In Line 6, M_i reads all the registers in $\mathsf{SM}[j]$, and copies them into $\mathsf{lm}_i[j][r]$, in a single atomic step.

The R iterations of the for-loop allow M_i to collect information about the current state s_j. After R iterations, the for-loop ends, and M_i emits a verdict based all the knowledge accumulated in its local memory. For our lower bound, we impose no restriction on the way this verdict is computed. However, for our upper bound, this verdict will be computed solely based on evaluating φ on the partial trace accumulated by M_i. Note that, even for a large R, M_i may still not be aware of all the atomic propositions of s_j, simply because the monitors that were covering these atomic propositions may be slow and may have not yet reported their samples in the shared memory. Also note that there is no point in waiting for the slow monitors, since it may well be the case that they have actually crashed, and waiting for them would yield a livelock.

A distributed-monitoring algorithm is an instantiation of the generic algorithm depicted in Algorithm 1. A concrete example of such an instantiation is provided in Section 4. Note that the generic Algorithm 1 takes full advantage of the total power of distributed wait-free computing.

3.4 Statement of the Problem

For any state s_j, when a set of monitors execute Algorithm 1, different interleavings, and hence different sets of verdicts, are possible. Global consistency is the property enabling to map the set of verdicts of the distributed monitors to *the* verdict of a centralized monitor that has the view of states *identical to the cumulated views of the monitors*. More specifically, given a state s_j, the *cover* of s_j is the partial state $\hat{s}_j$ such that, for every $p \in \mathsf{AP}$, $\hat{s}[p] \neq \natural$ if and only if p is in the sample of s_j by some non-faulty monitor M_i. From this point on, any reference to an execution trace $\alpha = s_0 s_1 \cdots s_j$ actually refers to the sequence of states covered by the monitors.

A *monitor trace* for an execution trace $\alpha = s_0 s_1 \cdots s_k$ is a sequence $m = m_0 m_1 \cdots m_k$, where, for every $j \in [0, k]$, $m_j \subseteq V$ for some verdict set V, and each element of each m_j is the verdict of some monitor $M_i \in \mathcal{M}$ emitted when considering state s_j. Let φ be an LTL formula, and let $\alpha = s_0 s_1 \cdots s_k$ be a finite (partial) trace corresponding to the sequence of (partial) states covered by the monitors.

Definition 3.5. A monitor trace $m = m_0 m_1 \ldots m_k$ with verdict set V satisfies *global consistency* for α with interpretation

$$\mu : 2^V \to \mathbb{B}_4$$

if, for every $0 \leq j \leq k$, if no monitors crash between the time when the system enters state s_j and the time when the system leaves state s_j, then

$$\mu(m_j) = [s_0 s_1 \cdots s_j \models_4 \varphi].$$

Note that $p \in \mathsf{AP}$ might be in the sample of a monitor observing the system in state s_j, but this monitor may crash before reporting this sample to the shared memory or may report this sample in the shared memory before crashing, but does it so late that no other monitors can see this sample (because asynchrony and failures prevent any monitor from waiting for any other monitor). This is why global consistency is required to hold only if no monitors crash when monitoring state s_j.

Definition 3.6. Let $\mathcal{A}$ be an instantiation of Algorithm 1 for an LTL formula φ with verdict set V. Algorithm $\mathcal{A}$ is *sound* for RV-LTLif there exists a function $\mu : 2^V \to \mathbb{B}_4$ such that, for every finite (partial) trace $\alpha \in \Sigma^*$ covered by the monitors and for every monitor trace m produced by $\mathcal{A}$ for α, m satisfies global consistency for α with interpretation μ.

The problem: Given an LTL formula φ, design an instantiation $\mathcal{A}$ of Algorithm 1 that correctly monitors φ, with monitors emitting verdicts picked from a small set V of values.

In particular, is any LTL formula φ correctly distributedly monitorable using $\mathbb{B}_4$ as verdict set for the monitors? The next section shows that the answer to this question is negative. However, further ahead in the text, it will be shown that, for every LTL formula φ, there is a distributed algorithm that correctly monitors φ with verdicts picked from the set of logical values of a multi-valued logic extending RV-LTL, whose cardinality is related neither to $|\mathsf{AP}|$ nor to $|\mathcal{M}|$, but to a specific characteristic of the formula φ.

4 DISTRIBUTED MONITORING USING RV-LTL

In this section, we pursue two goals. First, in Section 4.1, we modify Algorithm 1, so each monitor emits a verdict in $\mathbb{B}_4$, that is, truth values of RV-LTL. This constructs Algorithm 2, which we describe in detail. Then, in Section 4.2, we provide a concrete example of how distributed monitors can successfully monitor an LTL formula using Algorithm 2. In Section 4.3, we discuss our second goal and show that Algorithm 2 cannot monitor any LTL formula while ensuring soundness. In Section 5, we generalize this negative result to an impossibility result for fault-tolerant monitoring.

4.1 Distributed Monitoring with Verdicts in RV-LTL

As in the generic case, the local memory lm_i of monitor M_i is organized in levels, one for each state of the monitored trace. The same holds for the shared memory. For every $k \geq 0$, $\mathsf{lm}_i[k]$ stores a partial state, i.e., an $|\mathsf{AP}|$-dimensional vector with values in $\{\mathsf{true}, \mathsf{false}, \natural\}$. For every $k \geq 0$, and every $i \in [1, n]$, $\mathsf{SM}[k][i]$ stores a partial state, i.e., $\mathsf{SM}[k][i][p] \in \{\mathsf{true}, \mathsf{false}, \natural\}$ stores the value in s_k of the atomic proposition $p \in \mathsf{AP}$, as written by monitor M_i. Every monitor M_i also uses an

ALGORITHM 2: Behavior of monitor M_i, $i \in [1, n]$, using RV-LTL.

```
   Data: LTL formula $\varphi$ and state $s_k$, $k \geq 0$
   Result: a verdict from $\mathbb{B}_4$
1  if $k > 0$ then
2      $\mathsf{lm}'_i \leftarrow \mathsf{SM}[k-1]$;                /* $M_i$ snapshots the $(k-1)$th level of shared memory */
3      for every $p \in \mathsf{AP}$ do
4          if $(\mathsf{lm}_i[k-1][p] = \natural) \wedge (\exists j \in [1, n] : \mathsf{lm}'_i[j][p] \neq \natural)$ then
5              $\mathsf{lm}_i[k-1][p] := \mathsf{lm}'_i[j][p]$;          /* $M_i$ completes its view of $s_{k-1}$ */
6  $\mathsf{lm}_i[k] \leftarrow$ sample$_i(s_k)$;                /* $M_i$ takes sample, and gets some $p \in \mathsf{AP}$ for $s_k$ */
7  $\mathsf{SM}[k][i] \leftarrow \mathsf{lm}_i[k]$;               /* $M_i$ writes its current view of $s_k$ in shared memory */
8  $\mathsf{lm}'_i \leftarrow \mathsf{SM}[k]$;                   /* $M_i$ takes a snapshot of the shared memory */
9  for every $p \in \mathsf{AP}$ do
10     if $(\mathsf{lm}_i[k][p] = \natural) \wedge (\exists j \in [1, n] : \mathsf{lm}'_i[j][p] \neq \natural)$ then
11         $\mathsf{lm}_i[k][p] := \mathsf{lm}'_i[j][p]$;          /* $M_i$ gets propositions that were not in its sample */
12 emit $[\mathsf{lm}_i[0]\mathsf{lm}_i[1]\cdots\mathsf{lm}_i[k] \models_4 \varphi]$;   /* $M_i$ evaluates trace $\mathsf{lm}_i[0]\cdots\mathsf{lm}_i[k]$ in RV-LTL */
```

auxiliary storage variable lm'_i for local computation, which has the same format as one level of the shared memory, i.e., lm'_i stores one partial state for each monitor M_i. Again, we assume that all variables are initialized to $\natural$.

Algorithm 2 proceeds as follows: As in Algorithm 1, Lines 1–5 allow all non-faulty monitors observing s_k to share the same information about the partial trace resulting from the global observation of $s_0 s_1 \cdots s_{k-1}$. That is, for any monitor M_i and sampling s_k in Line 6, it holds

$$\mathsf{lm}_i[0]\mathsf{lm}_i[1]\cdots\mathsf{lm}_i[k-1] = s_0 s_1 \cdots s_{k-1}.$$

Let us now focus on the core of the algorithm. In Line 6, the monitor takes a sample of the current state s_k. This sample gives M_i the value of some atomic propositions $p \in \mathsf{AP}$, in which case $\mathsf{lm}_i[k][p] \in \{\mathsf{true}, \mathsf{false}\}$, but M_i may not become aware of some other atomic propositions $p' \in \mathsf{AP}$, in which case $\mathsf{lm}_i[k][p'] = \natural$. Then, only one round of the generic algorithm is run. That is, M_i writes its partial view of s_k (Line 7), and takes a snapshot of the shared memory (Line 8) with the objective of getting the values of atomic propositions of s_k that it is missing in its view. If there is indeed such a proposition p in its snapshot, then M_i adds this value in its partial view of s_k, in Line 11.

For emitting its verdict, monitor M_i evaluates trace $\mathsf{lm}_i[0]\cdots\mathsf{lm}_i[k]$ in RV-LTL, that is, its verdict is the truth value in $\mathbb{B}_4$ equal to:

$$[\mathsf{lm}_i[0]\mathsf{lm}_i[1]\cdots\mathsf{lm}_i[k] \models_4 \varphi].$$

Algorithm 2 is probably the most natural way of providing fault-tolerant distributed monitoring. However, as we show in the next subsection, RV-LTL is far from being sufficient, and even simple LTL formulas cannot be evaluated using distributed monitors using RV-LTL.

4.2 A Positive Example for Distributed Monitoring Using RV-LTL

Let $\mathcal{M} = \{M_1, M_2\}$, and let us consider monitoring the aforementioned request-acknowledgment formula

$$\varphi_{ra} = \Box(\neg a \wedge \neg r) \ \vee \ ((\neg a \ \mathcal{U} \ r) \wedge \Diamond a).$$

We represent a (partial) state in a finite trace for φ_{ra} as a vector

$$s = \begin{pmatrix} r \\ a \end{pmatrix},$$

where the propositions range over {true, false, $\natural$}. Let us assume that atomic proposition $\natural$ is extrapolated to false (we will show that the choice of extrapolation does not matter). Using a central monitor, evaluation in RV-LTL should return the following verdicts:

$\begin{pmatrix} r \\ a \end{pmatrix}$	$\begin{pmatrix} \text{false} \\ \text{false} \end{pmatrix}$	$\begin{pmatrix} \text{true} \\ \text{false} \end{pmatrix}$	$\begin{pmatrix} \text{false} \\ \text{true} \end{pmatrix}$	$\begin{pmatrix} \text{true} \\ \text{true} \end{pmatrix}$
verdict	$\top_p$	$\bot_p$	$\bot$	$\top$

,

where each column represents a trace of length one (i.e., a single state). In a distributed setting, a monitor may observe the following corresponding partial states and return verdicts in RV-LTL:

$\begin{pmatrix} r \\ a \end{pmatrix}$	$\begin{pmatrix} \natural \\ \natural \end{pmatrix}$	$\begin{pmatrix} \natural \\ \text{false} \end{pmatrix}$	$\begin{pmatrix} \natural \\ \text{true} \end{pmatrix}$	$\begin{pmatrix} \text{false} \\ \natural \end{pmatrix}$	$\begin{pmatrix} \text{false} \\ \text{false} \end{pmatrix}$	$\begin{pmatrix} \text{false} \\ \text{true} \end{pmatrix}$	$\begin{pmatrix} \text{true} \\ \natural \end{pmatrix}$	$\begin{pmatrix} \text{true} \\ \text{false} \end{pmatrix}$	$\begin{pmatrix} \text{true} \\ \text{true} \end{pmatrix}$
verdict	$\top_p$	$\top_p$	$\bot$	$\top_p$	$\top_p$	$\bot$	$\bot_p$	$\bot_p$	$\top$

.

Thanks to Lemma 3.4, the sets of possible verdicts returned by a collection of distributed monitors observing the system are, for the four possible scenarios:

$\begin{pmatrix} r \\ a \end{pmatrix}$	$\begin{pmatrix} \text{false} \\ \text{false} \end{pmatrix}$	$\begin{pmatrix} \text{true} \\ \text{false} \end{pmatrix}$	$\begin{pmatrix} \text{false} \\ \text{true} \end{pmatrix}$	$\begin{pmatrix} \text{true} \\ \text{true} \end{pmatrix}$
verdict sets	$\{\top_p\}$	$\{\bot_p\}$ or $\{\top_p, \bot_p\}$	$\{\bot\}$ or $\{\top_p, \bot\}$	$\{\top\}$ or $\{\top, \top_p\}$ or $\{\top, \bot_p\}$ or $\{\top, \bot\}$

.

Let us define the following interpretation function: For every non-empty $m \subseteq \mathbb{B}_4 = \{\top, \bot, \top_p, \bot_p\}$,

$$\mu(m) = \begin{cases} \top & \text{if } \top \in m \\ \bot & \text{if } \top \notin m \text{ and } \bot \in m \\ \bot_p & \text{if } m \cap \{\top, \bot\} = \emptyset \text{ and } \bot_p \in m \\ \top_p & \text{otherwise.} \end{cases}$$

With such an interpretation function, we do have

$$\mu(m) = [s \models_4 \varphi_{ra}],$$

as desired. This analysis can be extended to traces, and to monitor traces, establishing that Algorithm 2 correctly monitors φ_{ra} in RV-LTL.

4.3 A Counterexample to Distributed Monitoring Using RV-LTL

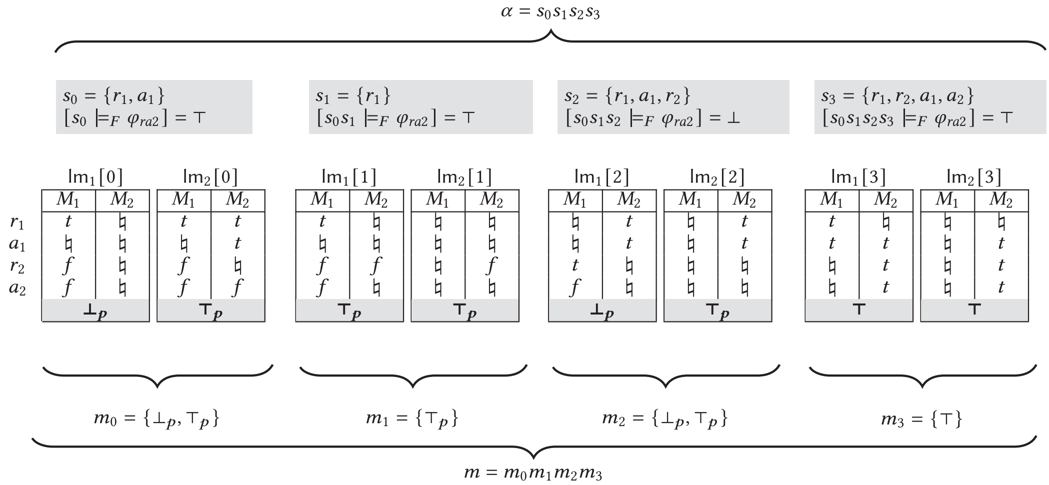
Let $\mathcal{M} = \{M_1, M_2\}$ and let us consider the LTL formula for two requests and two acknowledgments:

$$\varphi_{ra2} = (\Box(\neg a_1 \wedge \neg r_1) \vee [(\neg a_1 \ \mathcal{U}\ r_1) \wedge \Diamond a_1]) \wedge (\Box(\neg a_2 \wedge \neg r_2) \vee [(\neg a_2 \ \mathcal{U}\ r_2) \wedge \Diamond a_2]).$$

4.3.1 Negative Example of Monitoring φ_{ra2}. Figure 3 shows a concrete finite trace α and its corresponding monitor trace resulting from running Algorithm 2, where f stands for false, and t stands for true (in this example, too, $\natural$ is extrapolated to false). It also shows the content of the local memories of two monitors M_1 and M_2 monitoring α, as well as their individual evaluations of φ_{ra2} with respect to the observed trace. For instance, for s_0, let:

$$\text{sample}_1(s_0) = \begin{pmatrix} \text{true} \\ \natural \\ \text{false} \\ \text{false} \end{pmatrix} \quad \text{sample}_2(s_0) = \begin{pmatrix} \text{true} \\ \text{true} \\ \natural \\ \text{false} \end{pmatrix},$$

where each vector shows the value of propositions r_1, a_1, r_2, and a_2. Then, when M_1 and M_2 perform the write-snapshot instructions of Lines 7 and 8 of Algorithm 2, Figure 3 illustrates an execution


$\alpha = s_0 s_1 s_2 s_3$
$s_0 = \{r_1, a_1\}$
$[s_0 \models_F \varphi_{ra2}] = \top$
$s_1 = \{r_1\}$
$[s_0 s_1 \models_F \varphi_{ra2}] = \top$
$s_2 = \{r_1, a_1, r_2\}$
$[s_0 s_1 s_2 \models_F \varphi_{ra2}] = \bot$
$s_3 = \{r_1, r_2, a_1, a_2\}$
$[s_0 s_1 s_2 s_3 \models_F \varphi_{ra2}] = \top$
$m_0 = \{\bot_p, \top_p\}$
$m_1 = \{\top_p\}$
$m_2 = \{\bot_p, \top_p\}$
$m_3 = \{\top\}$
$m = m_0 m_1 m_2 m_3$

	$\mathsf{lm}_1[0]$		$\mathsf{lm}_2[0]$		$\mathsf{lm}_1[1]$		$\mathsf{lm}_2[1]$		$\mathsf{lm}_1[2]$		$\mathsf{lm}_2[2]$		$\mathsf{lm}_1[3]$		$\mathsf{lm}_2[3]$	
	M_1	M_2	M_1	M_2	M_1	M_2	M_1	M_2	M_1	M_2	M_1	M_2	M_1	M_2	M_1	M_2
r_1	t	$\natural$	t	t	t	$\natural$	$\natural$	$\natural$	$\natural$	t	$\natural$	t	t	$\natural$	$\natural$	$\natural$
a_1	$\natural$	$\natural$	$\natural$	t	$\natural$	$\natural$	$\natural$	$\natural$	$\natural$	t	$\natural$	t	t	t	$\natural$	t
r_2	f	$\natural$	f	$\natural$	f	f	$\natural$	f	t	$\natural$	$\natural$	$\natural$	$\natural$	t	$\natural$	t
a_2	f	$\natural$	f	f	f	$\natural$	$\natural$	$\natural$	f	$\natural$	$\natural$	$\natural$	$\natural$	t	$\natural$	t
	$\bot_p$		$\top_p$		$\top_p$		$\top_p$		$\bot_p$		$\top_p$		$\top$		$\top$	

Fig. 3. A monitor trace as computed by Algorithm 2.

in which M_1 does not get any new information (M_1 took the snapshot before M_2 wrote), while M_2 gets the partial trace sampled by M_1. As a result,

$$\mathsf{lm}_1[0] = \begin{pmatrix} \mathsf{true} \\ \natural \\ \mathsf{false} \\ \mathsf{false} \end{pmatrix} \quad \mathsf{lm}_2[0] = \begin{pmatrix} \mathsf{true} \\ \mathsf{true} \\ \mathsf{false} \\ \mathsf{false} \end{pmatrix}.$$

It follows that M_1 emits

$$\bot_p = \left[\mathsf{lm}_1[0] \models_4 \varphi_{ra2}\right],$$

while M_2 emits

$$\top_p = \left[\mathsf{lm}_2[0] \models_4 \varphi_{ra2}\right].$$

Since $[s_0 \models_4 \varphi_{ra2}] = \top_p$, it must be case that the set of verdicts $m_0 = \{\top_p, \bot_p\}$ is interpreted as $\top_p$, i.e.,

$$\mu(m_0) = \top_p.$$

A contradiction can be observed when considering M_1 and M_2 observing $s_0 s_1 s_2$. Indeed, in this case, too, the set of verdicts emitted by the monitors can be $m_2 = m_0 = \{\top_p, \bot_p\}$ for some interleaving of the write-snapshot instruction. However, $[s_0 s_1 s_2 \models_4 \varphi_{ra2}] = \bot_p$. Therefore, we get

$$\mu(m_2) \neq [s_0 s_1 s_2 \models_4 \varphi_{ra2}].$$

That is, Algorithm 2 does not correctly monitor φ_{ra2}.

4.3.2 *Negative Result on Monitoring a Single State for φ_{ra2}.*

We show that Algorithm 2 does not even correctly monitor φ_{ra2} on a single state. Figure 4 shows different execution interleavings of monitors M_1 and M_2 when running Algorithm 2 from two different states

$$s_0 = \{r_1, a_1\},$$

and

$$s'_0 = \{r_1, a_1, r_2\}.$$

Again, let us represent a state in a partial trace for φ_{ra2} as a vector

$$s = \begin{pmatrix} r_1 \\ a_1 \\ r_2 \\ a_2 \end{pmatrix}$$

with entries in {true, false, $\natural$}. In case of s_0, after executing Line 6 of Algorithm 2, monitors' samples consist of

$$\mathsf{lm}_1[0] = \begin{pmatrix} \mathsf{true} \\ \natural \\ \mathsf{false} \\ \mathsf{false} \end{pmatrix}, \text{ and } \mathsf{lm}_2[0] = \begin{pmatrix} \mathsf{true} \\ \mathsf{true} \\ \natural \\ \mathsf{false} \end{pmatrix}.$$

Likewise, for state s'_0, Figure 4 shows different local snapshots by M_1 and M_2. The verdict depends on the different interleavings of write/snapshot. In Figure 4, M_1, M_2 (respectively, M_2, M_1) denotes the case where monitor M_1 (respectively, M_2) executes write-snapshot instructions (Lines 7–8 of Algorithm 2) before monitor M_2 (respectively, M_1) does, and $M_1||M_2$ denotes the case where monitors M_1 and M_2 execute their write-snapshot actions concurrently.

Figure 4 shows that RV-LTL is unable to consistently monitor φ_{ra2}. More precisely, observe that, in the figure, the shaded collective verdicts m_0 and m'_0, for trace s_0 and trace s'_0, respectively, are identical, both equal to $\{\bot_p, \top_p\}$, while $[s_0 \models_4 \varphi_{ra2}] \neq [s'_0 \models_4 \varphi_{ra2}]$. Specifically, let us consider the following scenarios:

Scenario 1: Starting from state s_0 with M_1, M_2 interleaving, we have $[\mathsf{lm}_1[0] \models_4 \varphi_{ra2}] = \bot_p$ and $[\mathsf{lm}_2[0] \models_4 \varphi_{ra2}] = \top_p$. That is, the collective set of local verdicts is $m_0 = \{\bot_p, \top_p\}$.

Scenario 2: Starting from state s'_0, with M_2, M_1 interleaving, we have $[\mathsf{lm}'_1[0] \models_4 \varphi_{ra_2}] = \bot_p$ and $[\mathsf{lm}'_2[1] \models_4 \varphi_{ra_2}] = \top_p$. That is, the collective set of local verdicts is $m'_0 = \{\bot_p, \top_p\}$.

Therefore, although the valuations of φ_{ra_2} for two finite traces s_0 and s'_0 are different in RV-LTL (i.e., $\top_p$ and $\bot_p$, respectively), the collective set of verdicts emitted by monitors M_1 and M_2 in the above two scenarios are identical (i.e., $\{\bot_p, \top_p\}$). That is,

$$[s_0 \models_4 \varphi_{ra_2}] \neq [s'_0 \models_4 \varphi_{ra_2}],$$

but $\mu(m_0) = \mu(m'_0)$ for any μ, and, thus, φ_{ra2} is not correctly monitored, even on traces consisting in a single state.

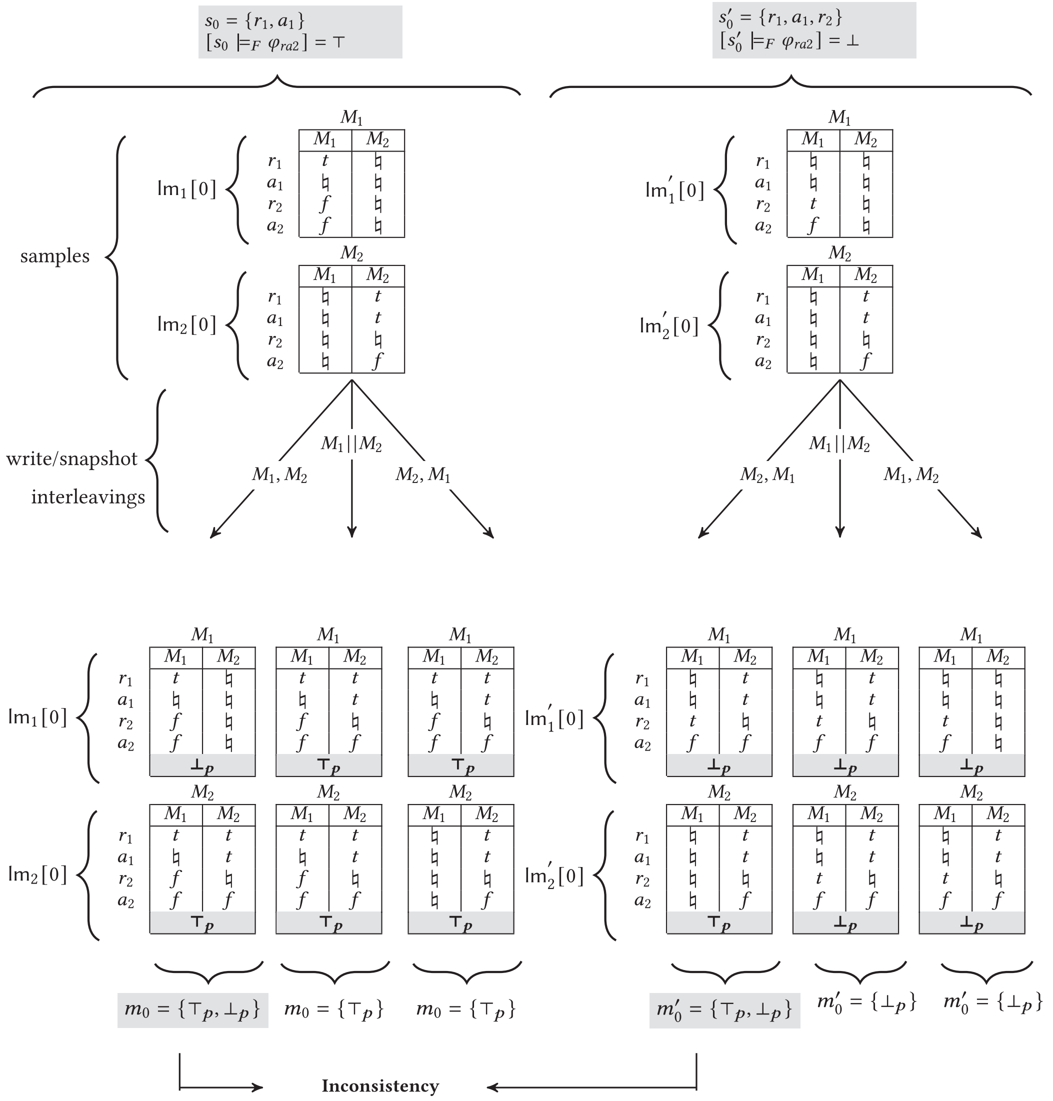
We summarize the discussions in this section by the following:

PROPERTY 4.1. *Not all LTL formulas can be consistently monitored by a 1-round distributed monitor with traces in RV-LTL. In particular, the LTL formula φ_{ra_2} cannot be monitored by a 1-round distributed monitor with traces in RV-LTL, even on traces consisting of a single state, even if monitors satisfy state coverage, and even if no monitors crash during the execution.*

The above results yield several questions. Do they hold only because Algorithm 2 does not perform sufficiently many communication rounds? Do they hold because the monitors exchange only partial states? Do they hold because the four possible individual verdicts are interpreted as logical values in $\mathbb{B}_4$? In the next section, we answer all these questions negatively: even the full-information Algorithm 1 cannot distributedly monitor LTL formula φ_{ra_2} with a verdict set of cardinality 4, independently from its number of rounds $R \geq 1$.

5 DISTRIBUTED MONITORING REQUIRES LARGE VERDICT SETS

In this section, we introduce a parameter that will be shown to have a strong impact on distributed monitoring, namely, the *alternation number* of an LTL formula. In particular, in this section, we



$\mathsf{lm}_1[0]$ (M_1), from s_0:

	M_1	M_2
r_1	t	♮
a_1	♮	♮
r_2	f	♮
a_2	f	♮

$\mathsf{lm}_2[0]$ (M_2), from s_0:

	M_1	M_2
r_1	♮	t
a_1	♮	t
r_2	♮	♮
a_2	♮	f

$\mathsf{lm}'_1[0]$ (M_1), from s'_0:

	M_1	M_2
r_1	♮	♮
a_1	♮	♮
r_2	t	♮
a_2	f	♮

$\mathsf{lm}'_2[0]$ (M_2), from s'_0:

	M_1	M_2
r_1	♮	t
a_1	♮	t
r_2	♮	♮
a_2	♮	f

$\mathsf{lm}_1[0]$ after M_1, M_2 / $M_1 || M_2$ / M_2, M_1:

	M_1	M_2	M_1	M_2	M_1	M_2
r_1	t	♮	t	t	t	t
a_1	♮	♮	♮	t	♮	t
r_2	f	♮	f	♮	f	♮
a_2	f	♮	f	f	f	f
	$\bot_p$		$\top_p$		$\top_p$	

$\mathsf{lm}_2[0]$ after M_1, M_2 / $M_1 || M_2$ / M_2, M_1:

	M_1	M_2	M_1	M_2	M_1	M_2
r_1	t	t	t	t	♮	t
a_1	♮	t	♮	t	♮	t
r_2	f	♮	f	♮	♮	♮
a_2	f	f	f	f	♮	f
	$\top_p$		$\top_p$		$\top_p$	

$\mathsf{lm}'_1[0]$ after M_2, M_1 / $M_1 || M_2$ / M_1, M_2:

	M_1	M_2	M_1	M_2	M_1	M_2
r_1	♮	t	♮	t	♮	♮
a_1	♮	t	♮	t	♮	♮
r_2	t	♮	t	♮	t	♮
a_2	f	f	f	f	f	♮
	$\bot_p$		$\bot_p$		$\bot_p$	

$\mathsf{lm}'_2[0]$ after M_2, M_1 / $M_1 || M_2$ / M_1, M_2:

	M_1	M_2	M_1	M_2	M_1	M_2
r_1	♮	t	♮	t	♮	t
a_1	♮	t	♮	t	♮	t
r_2	♮	♮	t	♮	t	♮
a_2	♮	f	f	f	f	f
	$\top_p$		$\bot_p$		$\bot_p$	

Fig. 4. Monitors M_1 and M_2 monitoring formula φ_{ra2} from two different states s_0 and s'_0.

show that, for every $k \geq 0$, there is an LTL formula φ with alternation number k that cannot be distributedly monitored by monitors emitting verdicts from a set of cardinality smaller than $k + 1$. This lower bound is an adaption of the lower bound in Reference [20], which deals with states whose correctness is specified by Boolean logic, to execution traces whose correctness is specified by linear temporal logic. In the next section, we shall show that the alternation number also essentially determines an upper bound on the number of truth values needed to ensure consistency in distributed monitoring, using truth values from a properly defined *multi-valued* logic.

5.1 Alternation Number

Let $\alpha \in \Sigma^*$ be a finite trace, and let α' be the longest proper prefix of α, i.e., $\alpha = \alpha' s$, where $\alpha' \in \Sigma^*$ and $s \in \Sigma$. Let φ be an LTL formula. We set the *alternation number* of φ with respect to α, denoted by altern(φ, α), as follows: First, for full generality, we do not define the alternation number of φ solely for traces, but also for partial traces. That is, in state s, proposition $p \in \mathsf{AP}$ can be true, false, or unknown ($\natural$). Given two partial states s and s', we set

$$s' \prec s$$

if the following two conditions hold:

- $\forall p \in \mathsf{AP} : (s'[p] \in \{\mathsf{true}, \mathsf{false}\} \ \Rightarrow\ s[p] = s'[p])$;
- $\exists p \in \mathsf{AP} : (s'[p] = \natural \ \wedge\ s[p] \in \{\mathsf{true}, \mathsf{false}\})$.

We denote by $s^{\natural}$ the partial state in which all atomic propositions are unknown.

Definition 5.1. The *alternation number* of an LTL formula φ with respect to a finite partial trace $\alpha = \alpha' s$ with $\alpha' \in \Sigma^*$ and $s \in \Sigma$, denoted by altern(φ, α), is the maximum integer $\ell \geq 0$, such that there exists a sequence of partial states $s_0 s_1 \cdots s_\ell$ with $s_0 = s^{\natural}$, $s_\ell = s$, and, for every $i \in \{0, 1, \ldots, \ell - 1\}$,

$$(s_i \prec s_{i+1}) \ \ \wedge\ \ ([\alpha' s_i \models_F \varphi] \neq [\alpha' s_{i+1} \models_F \varphi]).$$

The *alternation number* of an LTL formula φ is $\mathsf{altern}(\varphi) = \max\{\mathsf{altern}(\varphi, \alpha) \mid \alpha \in \Sigma^*\}$.

It directly follows from this definition that, for any LTL formula φ, its alternation number is bounded by its number of atomic propositions, i.e.,

$$\mathsf{altern}(\varphi) \leq |\mathsf{AP}|.$$

However, the alternation number can be much smaller than the number of atomic propositions. For instance,

$$\varphi = x_1 \wedge x_2 \wedge \cdots \wedge x_t$$

satisfies $|\mathsf{AP}| = t$ and $\mathsf{altern}(\varphi) = 1$ (assuming that the evaluation of a partial trace is performed by replacing all $\natural$ by false). Let us consider a few examples.

- $\mathsf{altern}(\Box p) = 1$, since once p is false, the formula can never evaluate to $\top$.
- $\mathsf{altern}(\Box(r \rightarrow \Diamond a)) = 2$, as witnessed by the partial states

$$\begin{pmatrix} r \\ a \end{pmatrix} = \begin{pmatrix} \natural \\ \natural \end{pmatrix} \begin{pmatrix} \mathsf{true} \\ \natural \end{pmatrix} \begin{pmatrix} \mathsf{true} \\ \mathsf{true} \end{pmatrix},$$

  which evaluate to $\top, \bot, \top$, respectively, in FLTL, when we extrapolate all $\natural$ to false.
- $\mathsf{altern}(\varphi_{ra}) = \mathsf{altern}\Big(\Box(\neg a \wedge \neg r) \ \vee\ [(\neg a \ \mathcal{U}\ r) \ \wedge\ \Diamond a]\Big) = 2$ with

$$\begin{pmatrix} r \\ a \end{pmatrix} = \begin{pmatrix} \natural \\ \natural \end{pmatrix} \begin{pmatrix} \natural \\ \mathsf{true} \end{pmatrix} \begin{pmatrix} \mathsf{true} \\ \mathsf{true} \end{pmatrix},$$

  which evaluate to $\top, \bot, \top$, respectively, in FLTL, when we extrapolate all $\natural$ to false.
- $\mathsf{altern}(\varphi_{ra2}) = 4$ with

$$\begin{pmatrix} r_1 \\ a_1 \\ r_2 \\ a_2 \end{pmatrix} = \begin{pmatrix} \natural \\ \natural \\ \natural \\ \natural \end{pmatrix} \begin{pmatrix} \mathsf{true} \\ \natural \\ \natural \\ \natural \end{pmatrix} \begin{pmatrix} \mathsf{true} \\ \mathsf{true} \\ \natural \\ \natural \end{pmatrix} \begin{pmatrix} \mathsf{true} \\ \mathsf{true} \\ \mathsf{true} \\ \natural \end{pmatrix} \begin{pmatrix} \mathsf{true} \\ \mathsf{true} \\ \mathsf{true} \\ \mathsf{true} \end{pmatrix},$$

  which evaluates to $\top, \bot, \top, \bot, \top$, respectively, in FLTL, when we extrapolate all $\natural$ to false.

5.2 The Impact of Alternation Number on Distributed Monitoring

The following result extends Property 4.1 to any distributed monitoring algorithm. It also extends the lower bound in Reference [20] to execution traces whose correctness is specified by means of linear temporal logic.

Theorem 5.2. *For every $k \geq 0$, there is an* LTL *formula φ with $\mathsf{altern}(\varphi) = 2k$ that cannot be correctly monitored by $n > 2k$ distributed monitors using verdict set V if $|V| \leq \mathsf{altern}(\varphi)$.*

Proof. For the purpose of proving this lower bound, we concentrate on the following variant of the request/acknowledge property: For every integer $k \geq 1$, let ψ_k be defined over the set of atomic propositions $\{r_1, \ldots, r_{k+1}, a_1, \ldots, a_{k+1}\}$. As in φ_{ra}, an acknowledgment must not appear before the corresponding request. However, it is no longer required that every request be acknowledged, but instead that at least one and at most k requests be acknowledged. That is,

$$\psi_k = \bigvee_{S \subsetneq [1,k+1], S \neq \emptyset} \left(\bigwedge_{i \in S} \Big((\neg a_i \,\mathcal{U}\, r_i) \wedge \Diamond a_i \Big) \wedge \bigwedge_{i \in [k+1] \setminus S} \Box \neg a_i \right).$$

Lemma 5.3. $\mathsf{altern}(\psi_k) = 2k$.

For establishing the lemma, let R_j, A_j be the following sequences of vectors in $\{\mathsf{true}, \mathsf{false}, \natural\}^{k+1}$, with $0 \leq j \leq 2k+2$. For every $j \in [0, 2k+2]$ and $i \in [1, k+1]$, we set

$$R_j[i] = \begin{cases} \mathsf{true} & \text{if } i \leq \lfloor j/2 \rfloor; \\ \natural & \text{otherwise.} \end{cases} \qquad A_j[i] = \begin{cases} \mathsf{true} & \text{if } i \leq \lceil j/2 \rceil; \\ \natural & \text{otherwise.} \end{cases}$$

That is, $A_0 = (\natural, \ldots, \natural) = R_0 = R_1$, and for $1 \leq j \leq k+1$,

$$A_{2j-1} = A_{2j} = (\underbrace{\mathsf{true}, \ldots, \mathsf{true}}_{j}, \natural, \ldots, \natural)$$

and

$$R_{2j} = R_{2j+1} = A_{2j-1}.$$

A pair $s_j = (R_j, A_j)$ defines a partial states as follows: For each $i \in [1, k+1]$, the value of the atomic proposition r_i is $R_j[i]$, and the value of the atomic proposition a_i is $A_j[i]$. Observe that $s_0 \prec s_1 \prec \ldots \prec s_{2j+2}$. For every $j \in [1, 2k+2]$, the following holds:

$$[s_j \models_F \psi_k] = \begin{cases} \bot & \text{if } j = 0 \\ \bot & \text{if } j \text{ is odd and } 1 \leq j \leq 2k+1 \\ \top & \text{if } j \text{ is even and } 2 \leq j \leq 2k \\ \bot & \text{if } j \text{ is even and } j = 2k+2 \end{cases}.$$

This is because:

- If $j = 0$, then no request is acknowledged in s_0.
- For $j = 2j' + 1, 0 \leq j' \leq k$, $a_1, \ldots, a_{j'}$ are true but $r_{j'}$ is false in s_j. Hence, there is an acknowledgment without the matching request. Hence, $[s_j \models_F \psi_k] = \bot$.
- For $j = 2j', 1 \leq j' \leq k$, every request $r_1, \ldots, r_{j'}$ is acknowledged, and there is no acknowledgment missing its matching request. Hence, $[s_j \models_F \psi_k] = \top$.
- Finally, in s_{2k+2} (as in s_{2k+1}), there are $k+1$ acknowledgments, and thus $[s_{2k+1} \models_F \psi_k] = [s_{2k+2} \models_F \psi_k] = \bot$.

It follows that the alternation number $\mathsf{altern}(\psi_k, s_0 s_2 \cdots s_{2k+1}) \geq 2k$. Therefore, $\mathsf{altern}(\psi_k) \geq 2k$.

Now, we prove the second part of Lemma 5.3, that is, $\mathrm{altern}(\psi_k) \leq 2k$. Let $\alpha s'$ be a partial trace, such that

$$\mathrm{altern}(\psi_k, \alpha s') = x.$$

That is, there exist partial states $s'_0 = s^{\natural} \prec s'_1 \prec \ldots \prec s'_x$ such that, for every $j = 0, \ldots, 2k$,

$$[\alpha s'_j \models \psi_k] \neq [\alpha s'_{j+1} \models \psi_k].$$

As above, each partial state s'_j can be represented by a pair of vectors

$$(A'_j, R'_j) \in \{\mathrm{true}, \mathrm{false}, \natural\}^{k+1} \times \{\mathrm{true}, \mathrm{false}, \natural\}^{k+1}.$$

Let $ack(j)$ denote the number of atomic propositions a_i whose value is true in the partial trace $\alpha s'_j$, i.e.,

$$ack(j) = |\{i : \exists s \in \alpha s'_j \text{ such that } a_i = \mathrm{true} \text{ in } s\}|.$$

Denote by ℓ and m the smallest (respectively, the largest) $j, 0 \leq j \leq x$, such that ψ_k is satisfied in $\alpha s'_j$. That is,

$$\ell = \min_{0 \leq j \leq x} [\alpha s'_j \models \psi_k] = \top,$$

$$m = \max_{0 \leq j \leq x} [\alpha s'_j \models \psi_k] = \top.$$

Note that $\ell \in \{0, 1\}$ and $m \in \{x - 1, x\}$.

Since, for satisfying ψ_k, it is required that the number of acknowledged requests is at least one, and at most k, we have $1 \leq ack(\ell)$ and $ack(m) \leq k$. Now, observe that if $[\alpha s'_j \models \psi_k] = \top$ and $[\alpha s'_{j+1} \models \psi_k] = \bot$, then $ack(j) < ack(j+1)$. Indeed, in $\alpha s'_j$, for each acknowledgment, there is a matching request, and the number of acknowledgments is at most k. Hence, to have $[\alpha s'_{j+1} \models \psi_k] = \bot$, it must be the case that $A'_j[i] = \natural$ and $A'_{j+1}[i] = \mathrm{true}$ for some $i, 1 \leq i \leq k+1$. It follows that $ack(s_\ell) + \lceil \frac{m-1}{2} \rceil \leq ack(m-1)$. Since $ack(m-1) \leq ack(m) \leq k$, and $s(\ell) \neq 0$, we derive $\lceil \frac{m-1}{2} \rceil \leq k - 1$, from which it follows that $m \leq 2k - 1$. As $m \in \{x-1, x\}$, we have $x \leq 2k$, and thus $\mathrm{altern}(\psi_k, \alpha(s') \leq 2k$.

We conclude that $\mathrm{altern}(\psi_k) = 2k$, which completes the proof of Lemma 5.3. □

In Reference [18], the authors study a collection of distributed tasks $\mathcal{T}(n, k, \ell)$ defined for n processes, where k, ℓ are integers. In each task in $\mathcal{T}(n, k, \ell)$, the possible inputs for each process p_i are the pairs $(c, d) \in \{1, \ldots, k+1\} \times \{1, \ldots, k+1\}$. The possible outputs form a set U of size ℓ, called the *opinion set*. Any partition $(\mathbf{Y}, \mathbf{N})$ of the multisets of at most n elements of U defines a task $T_{\mathbf{Y},\mathbf{N}} \in \mathcal{T}(n, k, U)$ as follows: In a distributed shared memory execution, we say that a process *participates* if it writes to the shared memory. For any set $P \subseteq \{1, \ldots, n\}$ of participating processes, let μ denote the *multiset* of the output values of these processes. The task $T_{\mathbf{Y},\mathbf{N}}$ is then specified as follows:

- If $1 \leq |\{d_i : i \in P\}| \leq k$ and $\{d_i : i \in P\} \subseteq \{c_i : i \in P\}$, it is required that $\mu \in \mathbf{Y}$;
- Otherwise, it is required that $\mu \in \mathbf{N}$.

A wait-free protocol solves tasks $T_{\mathbf{Y},\mathbf{N}} \in \mathcal{T}(n, k, U)$ whenever there is a constant B such that, for every participating set $P \subseteq \{1, \ldots, n\}$, and for every execution e with participating set P, if every process $i \in P$ has taken at least B steps in e, then every process produces an output value, and the outputs satisfy the requirement above.

Intuitively, the input of each process p_i represents the view of p_i as the outcome of an election: c_i is a candidate ID, and d_i is an elected ID. Election is valid if all elected IDs are candidates (i.e., $\{d_i : i \in P\} \subseteq \{c_i : i \in P\}$), and at least 1 and no more than k IDs are elected (i.e., $|\{d_i : i \in P\}| \leq k$).

By outputting a value $u_i \in U$, process p_i expresses its opinion regarding whether or not the election is globally valid. The processes must collectively be able to distinguish between valid and invalid elections. Indeed, when the inputs of the participating processes represent a valid election, it is required that the multiset of opinions belong to $\mathbf{Y}$, and, otherwise, that multiset must belong to $\mathbf{N}$.

The main result in Reference [18] is a characterization of the wait-free solvability of the tasks $\mathcal{T}(n, k, \ell)$.

Lemma 5.4 ([18]). *For any integers n, k, with $1 \leq k < n$, no task in $\mathcal{T}(n, k, \ell)$ is wait-free solvable if $\ell \leq \min(2k, n)$.*

To complete the lower bound, we show that monitoring ψ_k with a verdict set of size ℓ implies that some tasks in $\mathcal{T}(n, k, \ell)$ are wait-free solvable. Suppose that ψ_k can be monitored with a set of verdicts V of size ℓ. Let M be such a monitor and let $\mu : 2^V \rightarrow \mathbb{B}_4$ be its interpretation (cf. Definition 3.6). We show how M can be used to solve a task $T \in \mathcal{T}(n, k, \ell)$. The opinion set of T is V. The partition $(\mathbf{Y}, \mathbf{N})$ is induced by μ. Given a multiset x, let $\underline{x}$ denote its underlying set. We set:

$$x \in \mathbf{Y} \iff \mu(\underline{x}) \in \{\top_p, \top\}.$$

Algorithm 3 solves wait-free the task $T_{\mathbf{Y},\mathbf{N}} \in \mathcal{T}_{n,k,\ell}$.

ALGORITHM 3: *Solving a task $T \in \mathcal{T}(n, k, \ell)$ using a monitor with verdict set of size ℓ.*

```
Data: $(c_i, d_i) \in \{1, \ldots, k+1\} \times \{1, \ldots, k+1\}$
Result: an opinion from set $V$
1 $A_i \leftarrow \natural^{k+1}; R_i \leftarrow \natural^{k+1}$ ;     /* Construct a partial state according to the input $(c_i, d_i)$ */
2 for $j = 1$ to $k+1$ do
3     if $c_i = j$ then
4         $R_i[j] \leftarrow$ true
5     if $d_i = j$ then
6         $A_i[j] \leftarrow$ true
7 $s_i \leftarrow (A_i, R_i)$; $v_i \leftarrow M(s_i)$;     /* gets a verdict from the monitor algorithm */
8 return $v_i$
```

Algorithm 3 is wait-free, since the underlying monitor M is wait-free. Consider an execution with participating set P in which every participating process produces an output (at line 8). Let x denote the multiset formed by the outputs, and let $\underline{x}$ its underlying set. Let $s = (A, R)$ denote the partial state covered by the partial states computed by the participating processes, that is, for every $1 \leq j \leq k+1$,

$$A[j] = \begin{cases} \text{true} & \text{if } \exists i \in P, A_i[j] = \text{true} \\ \natural & \text{otherwise,} \end{cases} \quad \text{and} \quad R[j] = \begin{cases} \text{true} & \text{if } \exists i \in P, R_i[j] = \text{true} \\ \natural & \text{otherwise.} \end{cases}$$

We consider two cases according to the inputs of the participating processes:

- The inputs of the participating processes represent a valid election. That is,

$$1 \leq |\{d_i : i \in P\}| \leq k, \quad \text{and} \quad \{d_i : i \in P\} \subseteq \{c_i : i \in P\}.$$

  Hence, in $s = (A, R)$, there are at most k acknowledgments, and each of them has a matching request. Recall that the extrapolation function sets the value of each undefined atomic proposition in s to false. Therefore, $[s \models_4 \psi_k] = \top_p$ (as the trace can extend with $k+1$ acknowledgments in total). Hence, $\mu(\underline{x}) = \top_p$, from which we derive $x \in \mathbf{Y}$.

- The inputs of the participating processes represent a invalid election. That is,

$$|\{d_i : i \in P\}| \in \{0, k+1\}, \text{ or there exists } k \in P \text{ s.t. } d_k \notin \{c_i : i \in P\}.$$

  In $s = (A, R)$, there are no acknowledgments, or $k + 1$ acknowledgments, or some acknowledgments without any matching requests. Therefore, $[s \models_4 \psi_k] \in \{\bot_p, \bot\}$, and thus $\mu(\underline{x}) \in \{\bot_p, \bot\}$, from which we derive $x \in \mathbf{N}$.

We conclude that Algorithm 3 solves the task $T_{\mathbf{Y},\mathbf{N}}$ wait-free. As $T_{\mathbf{Y},\mathbf{N}} \in \mathcal{T}(n, k, \ell)$, it follows from Lemma 5.4 that $\ell > \min(n, 2k)$. By Lemma 5.3, $\mathsf{altern}(\psi_k) = 2k$. Therefore, if the number of monitors is larger than $\mathsf{altern}(\psi_k)$, then any correct monitor algorithm for ψ_k will require a verdict set of size larger than $\mathsf{altern}(\psi_k)$. □

6 MULTI-VALUED LTL FOR CONSISTENT DISTRIBUTED MONITORING

In this section, we introduce a novel multi-valued logic, called DLTL for *distributed* LTL, and we relate this logic to the notion of alternation number. We establish our main result in this section. That is, we show that, for every $\ell \geq 0$ and for every LTL formula φ with alternation number ℓ, there are distributed monitors using a verdict set of cardinality $2\lceil \ell/2 \rceil + 4$ that correctly monitor φ, where each monitor uses an automaton for evaluating φ in DLTL, i.e., DLTL with all truth values in

$$\mathbb{B}_{2\lceil \ell/2 \rceil + 4} = \{\top, \bot, \top_0, \bot_0, \dots, \top_{\lceil \ell/2 \rceil}, \bot_{\lceil \ell/2 \rceil}\},$$

which can be automatically synthesized from φ.

6.1 Semantics of DLTL

6.1.1 Definition. DLTL is directly motivated by distributed monitoring. In some sense, DLTL extends RV-LTL to more than four logical values with an eye on the alternation number. However, as opposed to RV-LTL, which is motivated by refining the uncertainty regarding what could occur in the future, DLTL is motivated by refining the uncertainty caused by asynchrony and failures.

For instance, let us consider a monitor M running Algorithm 2, and assume that M eventually collected a partial state s after having sampled a trace α with $|\alpha| = 1$, and after having exchanged information with other monitors. Let us assume that $[s \models_3 \varphi] = ?$ and $[s \models_F \varphi] = \top$. In RV-LTL, such a monitor M would output $\top_p$ as verdict, by Line 12 of Algorithm 2. The objective of DLTL is to refine such a verdict by providing a *level of certainty*. Indeed, it may well be the case that some other monitor M' collected a partial state $s' \prec s$, with $[s' \models_3 \varphi] = ?$ and $[s' \models_F \varphi] = \bot$, yielding a verdict $\bot_p$ from that monitor. With RV-LTL verdicts, i.e., verdicts in $\{\top, \top_p, \bot_p, \bot\}$, the set of verdicts emitted by these two monitors M and M' would be $\{\top_p, \bot_p\}$, while the $\top_p$ verdict emitted by M is somehow more relevant than the verdict $\bot_p$ emitted by M', because M has more information about the system than M'. The objective of DLTL is that M emits a verdict $\top_i$ while M' emits a verdict $\bot_j$, with $i > j$, where i and j are non-negative integers reflecting the degree of certainty of the verdicts. That is, a verdict $\top_i$ is viewed as more certain than a verdict $\bot_j$ whenever $i > j$.

Choosing the right level of certainty at which a verdict must be emitted is at the core of the definition of DLTL below.

Definition 6.1. Let $\alpha = \alpha' s$ be a finite partial trace in Σ^*, i.e., $s \in \Sigma = \{\mathsf{true}, \mathsf{false}, \natural\}$, and $\alpha' \in \Sigma^*$. The truth value in DLTL of an LTL formula φ with respect to α, denoted by $[\alpha \models_D \varphi]$, is defined as follows:

$$[\alpha \models_D \varphi] = \begin{cases} \top & \text{if } [\alpha \models_4 \varphi] = \top \\ \bot & \text{if } [\alpha \models_4 \varphi] = \bot \\ \top_0 & \text{if } [\alpha \models_4 \varphi] = \top_p \ \wedge \ (\forall s' \prec s : [\alpha' s' \models_D \varphi] = \top_0) \\ \bot_0 & \text{if } [\alpha \models_4 \varphi] = \bot_p \ \wedge \ (\forall s' \prec s : [\alpha' s' \models_D \varphi] = \bot_0) \\ \top_i \quad i > 0 & \text{if } [\alpha \models_4 \varphi] = \top_p \ \wedge \ (\exists s' \prec s : [\alpha' s' \models_D \varphi] = \bot_{i-1}) \\ & \quad \wedge \ (\forall s' \prec s, \exists j < i : [\alpha' s' \models_D \varphi] \in \{\top_j, \bot_j\} \cup \{\top_i\}) \\ \bot_i \quad i > 0 & \text{if } [\alpha \models_4 \varphi] = \bot_p \ \wedge \ (\exists s' \prec s : [\alpha' s' \models_D \varphi] = \top_{i-1}) \\ & \quad \wedge \ (\forall s' \prec s, \exists j < i : [\alpha' s' \models_D \varphi] \in \{\top_j, \bot_j\} \cup \{\bot_i\}) \end{cases}.$$

For $\ell \geq 0$, DLTL$_\ell$ is the restriction of DLTL, with all truth values in $\mathbb{B}_\ell = \{\top, \bot, \top_0, \bot_0, \dots, \top_\ell, \bot_\ell\}$.

Hence, in the case discussed above of two monitors M and M' having collected the partial states s and s', respectively, with $s' \prec s$, M can evaluate s in DLTL instead of RV-LTL, leading it to output a verdict $\top_i$, while evaluating s' in DLTL leads M' to output a verdict $\bot_j$, with $i > j$. Indeed, the existence of s' demonstrates that there exists a partial state $s' \prec s$ such that $[s' \models_F \varphi] \neq [s \models_F \varphi]$, so M emits a verdict with more certainty than M'. The level i is actually the length of the longest sequence $s_0 \prec s_1 \prec \cdots \prec s_i$ where $s_i = s$, such that, for every $j \in \{0, \dots, i-1\}$, we have $[s_j \models_F \varphi] \neq [s_{j+1} \models_F \varphi]$. Formally, we have the following:

Lemma 6.2. *Let $\alpha \neq \epsilon$ be a finite partial trace. The alternation number of an LTL formula φ with respect to α satisfies*

$$\text{altern}(\varphi, \alpha) = \begin{cases} 0 & \textit{if } [\alpha \models_D \varphi] \in \{\top, \bot\} \\ \ell & \textit{if } [\alpha \models_D \varphi] \in \{\bot_\ell, \top_\ell\} \textit{ for some } \ell \geq 0 \end{cases}.$$

Proof. Let φ be an LTL formula, and let $\alpha \neq \epsilon$ be a finite partial trace. Also, let $\alpha = \alpha' s$ with $\alpha' \in \Sigma^*$ and $s \in \Sigma$. If $[\alpha \models_D \varphi] \in \{\top, \bot, \top_0, \bot_0\}$, then altern$(\varphi, \alpha) = 0$, because the value of $[\alpha' s' \models_F \varphi]$ is the same for all $s' \preceq s$, and thus there are no alternances. The rest of the proof is by induction on ℓ. Let $\ell > 0$, assume that the lemma holds for $\ell - 1$, and let us show that it holds for ℓ. If $[\alpha \models_D \varphi] = \top_\ell$, then let $s' \prec s$ such that $[\alpha' s' \models_D \varphi] = \bot_{\ell-1}$. By induction, we get that altern$(\varphi, \alpha' s') = \ell - 1$. Moreover, $[\alpha' s' \models_F \varphi] = \bot$, and $[\alpha' s \models_F \varphi] = \top$, with $s' \prec s$. It follows that altern$(\varphi, \alpha' s) \geq \ell$. Moreover, altern$(\varphi, \alpha' s) \leq \ell$, because for every $s' \prec s$, $[\alpha' s' \models_D \varphi] \in \{\top_j, \bot_j\}$ for some $j < \ell$, which implies by induction that altern$(\varphi, \alpha' s') = j < \ell$. It follows that altern$(\varphi, \alpha' s) = \ell$, as claimed. The proof for the case $[\alpha \models_D \varphi] = \bot_\ell$ is analogous. □

6.1.2 *Reducing the Number of Logical Values in DLTL.*

Lemma 6.2 provides the intuition that, using DLTL, distributed monitoring an LTL formula with alternation number $\ell \geq 0$ could be done using verdicts in $\mathbb{B}_\ell = \{\top, \bot, \top_0, \bot_0, \dots, \top_\ell, \bot_\ell\}$, i.e., using $2\ell + 4$ logical values. While we shall prove in the next section that this is indeed the case, one can reduce the number of logical values by a factor of 2. Indeed, let us revisit the case of request-acknowledgment. As we have seen in Section 5, altern$(\varphi_{ra}) = 2$, and, as we have seen in Section 4.2, monitoring φ_{ra} using RV-LTL can be done using verdicts in $\mathbb{B}_4 = \{\top, \bot, \top_p, \bot_p\}$. Instead, Lemma 6.2 suggests that using DLTL would require eight values. This is because DLTL defines the relative certainty of verdicts $\bot_i$ and $\top_j$ only for $i > j$ or $j < i$. One can halve the number of logical values in DLTL by imposing an arbitrary order also between the certainties of $\bot_i$ and $\top_i$. This yields two variants of DLTL, respectively, called DLTL$^+$ and DLTL$^-$, depending on whether one imposes $\top_i$ more certainty than $\bot_i$, or $\top_i$ less certainty than $\bot_i$, respectively. More formally, these logics are defined as follows:

Definition 6.3. Let $\alpha = \alpha' s$ be a finite partial trace in Σ^*, i.e., $s \in \Sigma = \{\text{true}, \text{false}, \natural\}$, and $\alpha' \in \Sigma^*$. The truth value in DLTL$^+$ of an LTL formula φ with respect to α, denoted by $[\alpha \models_{D^+} \varphi]$, is defined as follows:

$$[\alpha \models_{D^+} \varphi] = \begin{cases} \top & \text{if } [\alpha \models_4 \varphi] = \top \\ \bot & \text{if } [\alpha \models_4 \varphi] = \bot \\ \top_0 & \text{if } [\alpha \models_4 \varphi] = \top_p \ \wedge\ (\forall s' \prec s : [\alpha' s' \models_{D^+} \varphi] \in \{\top_0, \bot_0\}) \\ \bot_0 & \text{if } [\alpha \models_4 \varphi] = \bot_p \ \wedge\ (\forall s' \prec s : [\alpha' s' \models_{D^+} \varphi] = \bot_0) \\ \top_i \quad i > 0 & \text{if } [\alpha \models_4 \varphi] = \top_p \ \wedge\ (\exists s' \prec s : [\alpha' s' \models_{D^+} \varphi] \in \{\top_i, \bot_i\}) \\ & \qquad \wedge\ (\forall s' \prec s, \exists j \leq i : [\alpha' s' \models_{D^+} \varphi] \in \{\top_j, \bot_j\}) \\ \bot_i \quad i > 0 & \text{if } [\alpha \models_4 \varphi] = \bot_p \ \wedge\ (\exists s' \prec s : [\alpha' s' \models_{D^+} \varphi] = \top_{i-1}) \\ & \qquad \wedge\ (\forall s' \prec s, \exists j < i : [\alpha' s' \models_{D^+} \varphi] \in \{\top_j, \bot_j\} \cup \{\bot_i\}) \end{cases}.$$

Similarly, the truth value in DLTL$^-$ of an LTL formula φ with respect to α, denoted by $[\alpha \models_{D^-} \varphi]$, is defined as follows:

$$[\alpha \models_{D^-} \varphi] = \begin{cases} \top & \text{if } [\alpha \models_4 \varphi] = \top \\ \bot & \text{if } [\alpha \models_4 \varphi] = \bot \\ \top_0 & \text{if } [\alpha \models_4 \varphi] = \top_p \ \wedge\ (\forall s' \prec s : [\alpha' s' \models_{D^-} \varphi]) = \top_0 \\ \bot_0 & \text{if } [\alpha \models_4 \varphi] = \bot_p \ \wedge\ (\forall s' \prec s : [\alpha' s' \models_{D^-} \varphi] \in \{\top_0, \bot_0\}) \\ \top_i \quad i > 0 & \text{if } [\alpha \models_4 \varphi] = \top_p \ \wedge\ (\exists s' \prec s : [\alpha' s' \models_{D^-} \varphi] = \bot_{i-1} \\ & \qquad \wedge\ (\forall s' \prec s, \exists j < i : [\alpha' s' \models_{D^-} \varphi] \in \{\top_j, \bot_j\} \cup \{\top_i\}) \\ \bot_i \quad i > 0 & \text{if } [\alpha \models_4 \varphi] = \bot_{\mathrm{p}} \ \wedge\ (\exists s' \prec s : [\alpha' s' \models_{D^-} \varphi] \in \{\top_i, \bot_i\})) \\ & \qquad \wedge\ (\forall s' \prec s, \exists j \leq i : [\alpha' s' \models_{D^-} \varphi] \in \{\top_j, \bot_j\}) \end{cases}.$$

It follows from these definitions that DLTL$^+$ induces the following order between the logical values:

$$\bot_0 < \top_0 < \bot_1 < \top_1 < \cdots < \top_{i-1} < \bot_i < \top_i < \bot_{i+1} < \dots$$

while DLTL$^-$ induces

$$\top_0 < \bot_0 < \top_1 < \bot_1 < \cdots < \bot_{i-1} < \top_i < \bot_i < \top_{i+1} < \dots$$

The following lemma illustrates the gain in terms of the number of logical values with respect to the alternation number, in comparison with Lemma 6.2. Recall that $s^\natural$ denotes the partial state in which none of the atomic propositions is known.

LEMMA 6.4. *Let $\alpha = \alpha' s$, with $\alpha' \in \Sigma^*$ and $s \in \Sigma$, be a finite partial trace. The alternation number of an LTL formula φ with respect to α satisfies the following two equalities:*

$$\mathrm{altern}(\varphi, \alpha) = \begin{cases} 0 & \textit{if } [\alpha \models_{D^+} \varphi] \in \{\top, \bot\} \\ 2\ell + 1 & \textit{if } ([\alpha \models_{D^+} \varphi] = \top_\ell) \ \wedge\ ([\alpha' s^\natural \models_F \varphi] = \bot) \\ 2\ell & \textit{if } \Big(([\alpha \models_{D^+} \varphi] = \top_\ell) \ \wedge\ ([\alpha' s^\natural \models_F \varphi] = \top)\Big) \\ & \quad \vee \Big(([\alpha \models_{D^+} \varphi] = \bot_\ell) \ \wedge\ ([\alpha' s^\natural \models_F \varphi] = \bot)\Big) \\ 2\ell - 1 & \textit{if } ([\alpha \models_{D^+} \varphi] = \bot_\ell) \ \wedge\ ([\alpha' s^\natural \models_F \varphi] = \top) \end{cases},$$

$$
\mathsf{altern}(\varphi,\alpha) = \begin{cases} 0 & \textit{if } [\alpha \models_{D^-} \varphi] \in \{\top,\bot\} \\ 2\ell+1 & \textit{if } ([\alpha \models_{D^-} \varphi] = \bot_\ell) \ \wedge \ ([\alpha' s^\natural \models_F \varphi] = \top) \\ 2\ell & \textit{if } \Big(([\alpha \models_{D^-} \varphi] = \bot_\ell) \ \wedge \ ([\alpha' s^\natural \models_F \varphi] = \bot)\Big) \\ & \quad \vee \Big(([\alpha \models_{D^-} \varphi] = \top_\ell) \ \wedge \ ([\alpha' s^\natural \models_F \varphi] = \top)\Big) \\ 2\ell-1 & \textit{if } ([\alpha \models_{D^-} \varphi] = \top_\ell) \ \wedge \ ([\alpha' s^\natural \models_F \varphi] = \bot) \end{cases} .
$$

Proof. Let φ be an LTL formula, and let $\alpha = \alpha' s$ be a finite partial trace. We first consider the statement for DLTL$^+$. If $[\alpha \models_{D^+} \varphi] \in \{\top,\bot\}$, then $\mathsf{altern}(\varphi,\alpha) = 0$, because the value of $[\alpha' s' \models_F \varphi]$ is the same for all $s' \preceq s$, and thus there are no alternances. From this point on, we assume that $[\alpha \models_{D^+} \varphi] \notin \{\top,\bot\}$. The rest of the proof is by induction on ℓ, where the reasoning below applies both to the base case $\ell = 0$, and to the inductive case for $\ell \geq 1$. Let $\ell \geq 0$.

If $[\alpha \models_{D^+} \varphi] = \top_\ell$, then let $s' \prec s$ such that $[\alpha' s' \models_{D^+} \varphi] \in \{\top_\ell, \bot_\ell\}$, and s' is minimal for this property, i.e., for every $s'' \prec s'$, we have $[\alpha' s'' \models_{D^+} \varphi] \notin \{\top_\ell,\bot_\ell\}$. Minimality implies that $[\alpha' s' \models_D \varphi] = \bot_\ell$. Thus, let $s'' \prec s'$ such that $[\alpha' s'' \models_D \varphi] = \top_{\ell-1}$. By induction, we get that $\mathsf{altern}(\varphi, \alpha' s'') = 2\ell - 1$ or $2\ell - 2$, depending on whether $[\alpha' s^\natural \models_F \varphi] = \bot$ or $\top$, respectively. Moreover, $[\alpha' s'' \models_F \varphi] = \top$, $[\alpha' s' \models_F \varphi] = \bot$, and $[\alpha' s \models_F \varphi] = \top$, with $s'' \prec s' \prec s$. It follows that $\mathsf{altern}(\varphi, \alpha' s) \geq 2\ell + 1$ if $[\alpha' s^\natural \models_F \varphi] = \bot$, and $\mathsf{altern}(\varphi,\alpha' s) \geq 2\ell$ if $[\alpha' s^\natural \models_F \varphi] = \top$. Moreover, $\mathsf{altern}(\varphi,\alpha' s)$ cannot be strictly greater than these respective bounds because, for every $s' \prec s$, there exists $j \leq \ell$ such that $[\alpha' s' \models_{D^+} \varphi] \in \{\top_j, \bot_j\}$, which implies that φ cannot alternate more than $2\ell + 1$ (respectively, 2ℓ) times with respect to α when $[\alpha' s^\natural \models_F \varphi] = \bot$ (respectively, $[\alpha' s^\natural \models_F \varphi] = \top$).

If $[\alpha \models_D \varphi] = \bot_\ell$, then let $s' \prec s$ such that $[\alpha' s' \models_{D^+} \varphi] = \top_{\ell-1}$. By induction, we get that $\mathsf{altern}(\varphi,\alpha' s') = 2\ell - 2$ or $2\ell - 3$, depending on whether $[\alpha' s^\natural \models_F \varphi] = \top$ or $\bot$, respectively. Moreover, $[\alpha' s' \models_F \varphi] = \top$, and $[\alpha' s \models_F \varphi] = \bot$, with $s' \prec s$. It follows that $\mathsf{altern}(\varphi,\alpha' s) \geq 2\ell$ if $[\alpha' s^\natural \models_F \varphi] = \bot$, and $\mathsf{altern}(\varphi, \alpha' s) \geq 2\ell - 1$ if $[\alpha' s^\natural \models_F \varphi] = \top$. Moreover, since, for every $s' \prec s$, there exists $j < \ell$ such that $[\alpha' s' \models_{D^+} \varphi] \in \{\top_j, \bot_j\}$, it follows that φ cannot alternate more than 2ℓ (respectively, $2\ell - 1$) times with respect to α when $[\alpha' s^\natural \models_F \varphi] = \bot$ (respectively, $[\alpha' s^\natural \models_F \varphi] = \top$).

This completes the proof for DLTL$^+$. The proof for DLTL$^-$ is analogous and thus omitted. □

As shown in Section 5.1, we have $\mathsf{altern}(\varphi_{ra}) = 2$ with the sequence

$$
\begin{pmatrix} r \\ a \end{pmatrix} = \begin{pmatrix} \natural \\ \natural \end{pmatrix}, \begin{pmatrix} \natural \\ \mathsf{true} \end{pmatrix}, \begin{pmatrix} \mathsf{true} \\ \mathsf{true} \end{pmatrix},
$$

which evaluate to $\top, \bot, \top$, respectively, in FLTL (assuming every atomic proposition $\natural$ is extrapolated to false). Also, we have seen in Section 4.2 that φ_{ra} can be distributedly monitored using RV-LTL. For this, we used an interpretation function μ that returns $\bot_p$ when applied to the set $\{\top_p, \bot_p\}$. This can be put in correspondence with using DLTL$_0^-$, with an interpretation function μ that simply returns the logical value with highest certainty in DLTL$^-$, i.e., $\bot_0$ for the set $\{\top_0, \bot_0\}$. We use such type of interpretation functions in our main theorem, stated in the next section.

6.2 Monitorability and Monitor Synthesis for DLTL

We have now all the ingredients to present our main result.

Theorem 6.5. *For every $\ell \geq 0$, and for every* LTL *formula φ with* $\mathsf{altern}(\varphi) = \ell$*, there are distributed monitors using verdict set $\mathbb{B}_{2\lceil \ell/2 \rceil + 4} = \{\bot, \top, \bot_0, \top_0, \ldots, \bot_{\lceil \ell/2 \rceil}, \top_{\lceil \ell/2 \rceil}\}$ that correctly monitor φ. Each monitor uses an automaton for evaluating φ in* DLTL$^+_{\lceil \ell/2 \rceil}$*, which can be automatically synthesized from φ.*

Proof. Let $\ell \geq 0$, and let φ be an LTL formula with $\mathsf{altern}(\varphi) = \ell$. We first show that φ can be correctly monitored by a set of monitors using DLTL. Later in the proof, we will show how to reduce the number of logical values by using DLTL$^+$. (Using DLTL$^-$ would also achieve this, and we have chosen DLTL$^+$ arbitrarily—see discussion after the proof.) The algorithm performed by each monitor is given in Algorithm 4. This algorithm performs the same instructions as Algorithm 2, but evaluates the collected partial trace in DLTL instead of RV-LTL.

ALGORITHM 4: Behavior of Monitor M_i, $i \in [1, n]$, using DLTL.

```
Data: LTL formula $\varphi$ with $\mathsf{altern}(\varphi) = \ell$, and state $s_k$, $k \geq 0$
Result: a verdict from $\mathbb{B}_{2\ell+4}$
perform instructions of Lines 1–11 in Algorithm 2;          /* sample, write, read, and update */
emit $[\mathsf{Im}_i[0]\mathsf{Im}_i[1]\cdots\mathsf{Im}_i[k] \models_D \varphi]$;          /* $M_i$ evaluates trace $\mathsf{Im}_i[0]\cdots\mathsf{Im}_i[k]$ in DLTL */
```

Let $\mathbb{B}_\infty = \{\top, \bot\} \cup (\cup_{i\geq 0}\{\top_i, \bot_i\})$. The interpretation function

$$\mu : 2^{\mathbb{B}_\infty} \to \mathbb{B}_4$$

interprets any *finite* set $m \in 2^{\mathbb{B}_\infty}$ of logical values in DLTL returned by the monitors as the truth value of RV-LTL corresponding to the highest index i for which $m \cap \{\top_i, \bot_i\} \neq \emptyset$—we will show that, for every i, $\bot_i$ and $\top_i$ cannot be both in m, and that $\bot$ and $\top$ cannot be both in m. More specifically, for every finite set $m \subseteq 2^{\mathbb{B}_\infty}$, we define

$$\mu(m) = \begin{cases} \top & \text{if } \top \in m; \\ \bot & \text{if } \bot \in m; \\ \top_p & \text{if } m \cap \{\top, \bot\} = \emptyset, \text{ and } \left(\exists i \geq 0 : \top_i \in m, \text{ and } \forall j \geq 0, \bot_j \in m \Rightarrow j < i\right); \\ \bot_p & \text{if } m \cap \{\top, \bot\} = \emptyset, \text{ and } \left(\exists i \geq 0 : \bot_i \in m, \text{ and } \forall j \geq 0, \top_j \in m \Rightarrow j < i\right). \end{cases}$$

Let us show that, for every finite partial trace $\alpha = s_0 s_1 \cdots s_k$ with $k \geq 0$, if m is a set of values returned by the monitors for α, then

$$\mu(m) = [\alpha \models_4 \varphi].$$

Recall that every state s_i in α might be a partial state, defined as the partial state covered by all the non-faulty monitors during the ith execution of Algorithm 4 (i.e., the execution of the algorithm on $s_0 s_1 \cdots s_i$). Also recall that, at the beginning of each execution of Algorithm 4, say, at phase i, every monitor takes a snapshot of the shared memory to get the entire partial tate s_{i-1}. That is, when the monitors start executing Algorithm 4 for state s_k, they all agree on the trace $s_0 s_1 \cdots s_{k-1}$. However, the monitors may get different samples of s_k, and, because of asynchrony, may have to emit a verdict based on different perspectives on the state s_k. To sum up, for every $i \neq j$, we have

$$\mathsf{Im}_i[0]\mathsf{Im}_i[1]\cdots\mathsf{Im}_i[k-1] = \mathsf{Im}_j[0]\mathsf{Im}_j[1]\cdots\mathsf{Im}_j[k-1] = s_0 s_1 \cdots s_{k-1},$$

while it may be the case that

$$\mathsf{Im}_i[k] \neq \mathsf{Im}_j[k] \neq s_k.$$

However, by Lemma 3.4, the monitor M_i that performs the snapshot last (i.e., the snapshot in Line 8 of Algorithm 2) satisfies

$$\mathsf{Im}_i[k] = s_k.$$

The verdict of this monitor is $[\mathsf{Im}_i[0]\mathsf{Im}_i[1]\cdots\mathsf{Im}_i[k] \models_D \varphi]$, that is, precisely

$$[s_0 s_1 \cdots s_k \models_D \varphi].$$

By definition of DLTL, this verdict agrees with RV-LTL, in the following sense:

$$\begin{array}{lcl} [s_0s_1\cdots s_k \models_4 \varphi] = \top & \Longleftrightarrow & [s_0s_1\cdots s_k \models_D \varphi] = \top \\ [s_0s_1\cdots s_k \models_4 \varphi] = \bot & \Longleftrightarrow & [s_0s_1\cdots s_k \models_D \varphi] = \bot \\ [s_0s_1\cdots s_k \models_4 \varphi] = \top_p & \Longleftrightarrow & [s_0s_1\cdots s_k \models_D \varphi] = \top_i \text{ with } i \geq 0 \\ [s_0s_1\cdots s_k \models_4 \varphi] = \bot_p & \Longleftrightarrow & [s_0s_1\cdots s_k \models_D \varphi] = \bot_i \text{ with } i \geq 0. \end{array}$$

Moreover, by the extensions of LTL$_3$ and FLTL to partial traces in Section 3.2, if

$$[s_0s_1\cdots s_k \models_D \varphi] = \top,$$

then there are no $s'_k \prec s_k$ such that $[s_0s_1\cdots s'_k \models_D \varphi] = \bot$. Similarly, if $[s_0s_1\cdots s_k \models_D \varphi] = \bot$, then there are no $s'_k \prec s_k$ such that $[s_0s_1\cdots s'_k \models_D \varphi] = \top$. Also, by definition of DLTL, if

$$[s_0s_1\cdots s_k \models_D \varphi] = \top_i,$$

then, for every $s'_k \prec s_k$, we have either

$$[s_0s_1\cdots s'_k \models_D \varphi] = \top_i \text{ or } [s_0s_1\cdots s'_k \models_D \varphi] \in \{\bot_j, \top_j\}$$

for some $j < i$. Similarly, if $[s_0s_1\cdots s_k \models_D \varphi] = \bot_i$, then, for every $s'_k \prec s_k$, we have either $[s_0s_1\cdots s'_k \models_D \varphi] = \bot_i$ or $[s_0s_1\cdots s'_k \models_D \varphi] \in \{\bot_j, \top_j\}$ for some $j < i$. It follows that $\mu(m) = [\alpha \models_4 \varphi]$, as desired.

By Lemma 6.2, if φ satisfies $\mathsf{altern}(\varphi) = \ell$, then all verdicts are in $\mathbb{B}_{2\ell+4}$. Reducing the number of logical values, from $2\,\mathsf{altern}(\varphi) + 4$ to $2\lceil \mathsf{altern}(\varphi)/2\rceil + 4$ is achieved by replacing the evaluation of the trace in DLTL at each monitor, by an evaluation in DLTL$^+$. By Lemma 6.4, if φ satisfies $\mathsf{altern}(\varphi) = \ell$, then all verdicts are in $\mathbb{B}_{2\lceil \ell/2\rceil+4}$.

To complete the proof, we show how, given any LTL formula, each monitor can evaluate a partial finite trace $\alpha = s_0s_1\cdots, s_k$ in DLTL$^+$. Let

$$\mathcal{M} = (\Sigma, Q, q_0, \delta, \lambda)$$

be the RV-LTL automaton for φ. We have

$$\lambda(\delta(q_0, \alpha)) = [\alpha \models_4 \varphi]$$

for every finite trace $\alpha \in \Sigma^*$ (see Figure 1 for an example of such an automaton). In other words, the prefix $\alpha' = s_0s_1\cdots, s_{k-1}$ of the execution is fully encoded in the state $\delta(q_0, \alpha')$ reached in $\mathcal{M}$ after having executed the k transitions induced by α'. In particular, for any two $\beta_i \in \Sigma^*$, $i \in \{1, 2\}$, if $\delta(q_0, \beta_1) = \delta(q_0, \beta_2)$, then, for any $s \in \Sigma$,

$$\mathsf{altern}(\varphi, \beta_1 s) = \mathsf{altern}(\varphi, \beta_2 s).$$

Therefore, to let monitors evaluate $[\alpha' s_k \models_{D^+} \varphi]$, it is sufficient to provide each monitor with a table Λ containing $|Q| \times |\Sigma|$ entries in $\{0, 1, \ldots, \lceil \mathsf{altern}(\varphi)/2\rceil\}$, where we define:

$$\Lambda[q, s] = \mathsf{altern}(\varphi, \beta s)$$

for any $\beta \in \Sigma^*$ satisfying $\delta(q_0, \beta) = q$. Indeed, for $\alpha = s_0s_1\cdots s_k$ and $\alpha' = s_0s_1\cdots s_{k-1}$, let

$$q = \delta(q_0, \alpha) \text{ and } q' = \delta(q_0, \alpha').$$

Then, we have

$$[\alpha \models_{D^+} \varphi] = \begin{cases} \lambda(q) & \text{if } \lambda(q) \in \{\top, \bot\} \\ \top_{\lceil \ell/2\rceil} & \text{if } \lambda(q) = \top_p, \text{ where } \ell = \Lambda[q', s_k] \\ \bot_{\lceil \ell/2\rceil} & \text{if } \lambda(q) = \bot_p, \text{ where } \ell = \Lambda[q', s_k] \end{cases}.$$

In other words, given the RV-LTL automaton for φ, and given the lookup table Λ, every monitor can evaluate every trace α in DLTL$^+$. □

Remarks. It is worth pointing out that the number of logical values used by the monitors in Theorem 6.5 can be further reduced, but by an additive factor only, under some specific conditions, including the following scenarios. We also note that one significance of Theorem 6.5 is that safety formulas can be efficiently monitored with only four truth values. In general, formulas with only one temporal operator need this many truth values to be consistently monitored. We should also mention that the size of a DLTL monitor is the size of its corresponding RV-LTLmonitor times ℓ (one RV-LTL monitor per alternation).

- Let us consider an LTL formula φ, with $\mathrm{altern}(\varphi) = \ell$ odd. Let us also assume that, for every finite trace α such that there exists a sequence $s_0 \prec s_1 \prec \cdots \prec s_\ell$ of partial states satisfying $[\alpha s_i \models_F \varphi] \neq [\alpha s_{i+1} \models_F \varphi]$ for every $i \in \{0, \ldots, \ell - 1\}$, we have $[\alpha s^\natural \models_F \varphi] = \bot$. Then the number of truth values used by DLTL$^+$ is not $2\lceil \ell/2 \rceil + 4$ but only $2\lfloor \ell/2 \rfloor + 4$. Similarly, if $[\alpha s^\natural \models_F \varphi] = \top$ for every finite trace α such that there exists a sequence $s_0 \prec s_1 \prec \cdots \prec s_\ell$ of partial states satisfying $[\alpha s_i \models_F \varphi] \neq [\alpha s_{i+1} \models_F \varphi]$ for every $i \in \{0, \ldots, \ell - 1\}$, then, using DLTL$^-$ instead of DLTL$^+$ yields using only $2\lfloor \ell/2 \rfloor + 4$ truth values, instead of $2\lceil \ell/2 \rceil + 4$.
- Let us consider an LTL formula φ, with $\mathrm{altern}(\varphi) = \ell$ even. Let us also assume that, for every finite trace α such that there exists a sequence $s_0 \prec s_1 \prec \cdots \prec s_\ell$ of partial states satisfying $[\alpha s_i \models_F \varphi] \neq [\alpha s_{i+1} \models_F \varphi]$ for every $i \in \{0, \ldots, \ell - 1\}$, we have

  $$[\alpha s^\natural \models_F \varphi] = \bot, \text{ and } [\alpha s_\ell \models_3 \varphi] = \top$$

  for all such sequences (note that the evaluation of αs_ℓ is performed in LTL$_3$). An example of such a situation is φ_{ra}. Its alternation number is 2, and every sequence $s_0 \prec s_1 \prec s_2$ alternating twice satisfies $[\alpha s_0 \models_F \varphi_{ra}] = \bot$ with $s_0 = \left(\begin{smallmatrix}\natural\\ \natural\end{smallmatrix}\right)$, and $[\alpha s_2 \models_3 \varphi_{ra}] = \top$ with $s_2 = \left(\begin{smallmatrix}true\\ true\end{smallmatrix}\right)$. In such a scenario, the truth values of highest certainty, $\top_\ell$ and $\bot_\ell$, can be discarded whenever using DLTL$^-$ instead of DLTL$^+$, saving two truth values. That is, one can restrict the truth values to be in $\mathbb{B}_{\ell/2+2} = \{\top, \bot, \top_0, \bot_0, \ldots, \top_{\ell/2-1}, \bot_{\ell/2-1}\}$. In the particular case of φ_{ra}, one can therefore restrict the truth values to be in $\mathbb{B}_4 = \{\top, \bot, \top_0, \bot_0\}$, as it was previously established in Section 4.2.

7 CONCLUSION

We have established a tight (up to a small additive constant) bound on the cardinality of the set of verdicts from which a collection of asynchronous crash-prone monitors pick their individual verdicts for monitoring an LTL formula φ in a distributed manner. This cardinality is related to the alternation number, $\mathrm{altern}(\varphi)$, of the formula. We showed that, for every $\ell \geq 0$, every LTL formula φ with $\mathrm{altern}(\varphi) = \ell$ can be monitored by distributed monitors with verdicts in $\mathbb{B}_{2\lceil \ell/2 \rceil} = \{\bot, \top, \bot_0, \top_0, \ldots, \top_{2\lceil \ell/2 \rceil}, \bot_{2\lceil \ell/2 \rceil}\}$, and each verdict results from evaluating the observed partial trace in the multi-valued logic DLTL$^+$. The bound on the size of the verdict set is (almost) tight, in the sense that, for every $\ell \geq 0$, there exists an LTL formula φ with $\mathrm{altern}(\varphi) = \ell$ such that, for every set V with $|V| \leq \ell$, φ cannot be monitored by distributed monitors with verdicts in V.

For establishing these results, we impose two restrictions. First, we assume that all operations performed by the distributed monitors (sampling the current state, exchanging information with the other monitors, and producing the verdict) can be performed between two changes of states by the monitored system. Second, we specify distributed monitoring by imposing global consistency of the set m_k of verdicts with respect to the centralized evaluation of the actual trace $s_0 s_1 \cdots s_k$ in RV-LTL, by requiring equality $\mu(m_k) = [s_0 s_1 \ldots s_k \models_4 \varphi]$ between the interpretation $\mu(m_k)$ and the evaluation of $s_0 s_1 \cdots s_k$ in RV-LTL, only for verdicts produced in the absence of crashes during the monitoring of s_k. This latter restriction appears natural, and perhaps even unavoidable, because, otherwise, the distributed monitors and the centralized monitor deal with different traces, which are inherently incomparable. However, it might be desirable to relax the former restriction, because such an assumption might not always be satisfied in practice, in particular by rapidly evolving systems. Getting rid of this assumption seems, however, challenging, as one would have to deal not only with issues caused by asynchrony between monitors with different partial views of a *same* state, but also with issues caused by asynchrony between monitors with partial views of *different* states. Reconciliation of such views looks difficult. Nevertheless, this opens a challenging, but rewarding direction for future work.

Another challenging problem in the context of fault-tolerant distributed monitoring is to consider other types of faults, namely, *Byzantine* faults. These faults may arbitrarily change the output of individual monitors, i.e., their verdicts. It is unclear how a collection of faulty monitors that may misrepresent their partial view of the system can be transformed into a sound single verdict that a correct centralized monitor would produce. This problem can also open an entirely new line of research to deal with distributed monitoring in the presence of faults and security attacks.

REFERENCES

[1] Yehuda Afek, Hagit Attiya, Danny Dolev, Eli Gafni, Michael Merritt, and Nir Shavit. 1993. Atomic snapshots of shared memory. *J. ACM* 40, 4 (1993), 873–890.

[2] James Aspnes, Hagit Attiya, Keren Censor-Hillel, and Faith Ellen. 2015. Limited-use atomic snapshots with polylogarithmic step complexity. *J. ACM* 62, 1 (2015), 3:1–3:22. DOI: https://doi.org/10.1145/2732263

[3] Hagit Attiya, Amotz Bar-Noy, and Danny Dolev. 1995. Sharing memory robustly in message-passing systems. *J. ACM* 42, 1 (1995), 124–142. DOI: https://doi.org/10.1145/200836.200869

[4] Hagit Attiya, Sweta Kumari, Archit Somani, and Jennifer L. Welch. 2022. Store-collect in the presence of continuous churn with application to snapshots and lattice agreement. *Inf. Computat.* (2022), 104869. DOI: https://doi.org/10.1016/j.ic.2022.104869

[5] Hagit Attiya and Ophir Rachman. 1998. Atomic snapshots in $O(n \log n)$ operations. *SIAM J. Comput.* 27, 2 (1998), 319–340. DOI: https://doi.org/10.1137/S0097539795279463

[6] Hagit Attiya and Jennifer Welch. 2004. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*. Wiley.

[7] A. Bauer, M. Leucker, and C. Schallhart. 2010. Comparing LTL semantics for runtime verification. *J. Logic Computat.* 20, 3 (2010), 651–674.

[8] A. Bauer, M. Leucker, and C. Schallhart. 2011. Runtime verification for LTL and TLTL. *ACM Trans. Softw. Eng. Methodol.* 20, 4 (2011), 14.

[9] A. K. Bauer and Y. Falcone. 2012. Decentralised LTL monitoring. In *Proceedings of the 18th International Symposium on Formal Methods (FM)*. 85–100.

[10] S. Berkovich, B. Bonakdarpour, and S. Fischmeister. 2013. GPU-based runtime verification. In *Proceedings of the 27th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 1025–1036.

[11] Glenn Bruns and Patrice Godefroid. 1999. Model checking partial state spaces with 3-valued temporal logics. In *Proceedings of the 11th International Conference on Computer Aided Verification (CAV)*. 274–287. DOI: https://doi.org/10.1007/3-540-48683-6_25

[12] Glenn Bruns and Patrice Godefroid. 2000. Generalized model checking: Reasoning about partial state spaces. In *Proceedings of the 11th International Conference on Concurrency Theory (CONCUR)*. 168–182. DOI: https://doi.org/10.1007/3-540-44618-4_14

[13] H. Chauhan, V. K. Garg, A. Natarajan, and N. Mittal. 2013. A distributed abstraction algorithm for online predicate detection. In *Proceedings of the 32nd IEEE Symposium on Reliable Distributed Systems (SRDS)*. 101–110.

[14] C. Colombo and Y. Falcone. 2014. Organising LTL monitors over distributed systems with a global clock. In *Proceedings of the 14th International Conference on Runtime Verification (RV)*. 140–155.

[15] Carole Delporte-Gallet, Hugues Fauconnier, Sergio Rajsbaum, and Michel Raynal. 2018. Implementing snapshot objects on top of crash-prone asynchronous message-passing systems. *IEEE Trans. Parallel Distrib. Syst.* 29, 9 (2018), 2033–2045. DOI: https://doi.org/10.1109/TPDS.2018.2809551

[16] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. 1999. Patterns in property specifications for finite-state verification. In *Proceedings of the International Conference on Software Engineering (ICSE)*. 411 –420.

[17] M. J. Fischer, N. A. Lynch, and M. S. Peterson. 1985. Impossibility of distributed consensus with one faulty processor. *J. ACM* 32, 2 (1985), 373–382.

[18] Pierre Fraigniaud, Sergio Rajsbaum, Matthieu Roy, and Corentin Travers. 2014. The opinion number of set-agreement. In *Proceedings of the 18th International Conference on Principles of Distributed Systems (OPODIS)*. 155–170.

[19] Pierre Fraigniaud, Sergio Rajsbaum, and Corentin Travers. 2013. Locality and checkability in wait-free computing. *Distrib. Comput.* 26, 4 (2013), 223–242. DOI: https://doi.org/10.1007/s00446-013-0188-x

[20] P. Fraigniaud, S. Rajsbaum, and C. Travers. 2014. On the number of opinions needed for fault-tolerant run-time monitoring in distributed systems. In *Proceedings of the 5th International Conference on Runtime Verification (RV)*. 92–107.

[21] R. Ganguly, A. Momtaz, and B. Bonakdarpour. 2020. Distributed runtime verification under partial asynchrony. In *Proceedings of the 24th International Conference on Principles of Distributed Systems (OPODIS)*. 20:1–20:17.

[22] M. H. Herlihy, D. Kozlov, and S. Rajsbaum. 2013. *Distributed Computing Through Combinatorial Topology*. Morgan Kaufmann-Elsevier.

[23] Michiko Inoue and Wei Chen. 1994. Linear-time snapshot using multi-writer multi-reader registers. In *Proceedings of the 8th International Workshop on Distributed Algorithms (WDAG)*. 130–140. DOI: https://doi.org/10.1007/BFb0020429

[24] Z. Manna and A. Pnueli. 1995. *Temporal Verification of Reactive Systems - Safety*. Springer.

[25] N. Mittal and V. K. Garg. 2005. Techniques and applications of computation slicing. *Distrib. Comput.* 17, 3 (2005), 251–277.

[26] M. Mostafa and B. Bonakdarpour. 2015. Decentralized runtime verification of LTL specifications in distributed systems. In *Proceedings of the 29th International Parallel and Distributed Processing Symposium (IPDPS)*. 494–503.

[27] V. A. Ogale and V. K. Garg. 2007. Detecting temporal logic predicates on distributed computations. In *Proceedings of the 21st International Symposium on Distributed Computing (DISC)*. 420–434.

[28] A. Pnueli. 1977. The temporal logic of programs. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*. 46–57.

[29] A. Pnueli and A. Zaks. 2006. PSL model checking and run-time verification via testers. In *Proceedings of the 14th International Symposium on Formal Methods (FM)*. 573–586.

[30] A. Sen and V. K. Garg. 2004. Detecting temporal logic predicates in distributed programs using computation slicing. In *Principles of Distributed Systems*. Springer, 171–183.

[31] K. Sen, A. Vardhan, G. Agha, and G. Rosu. 2004. Efficient decentralized monitoring of safety in distributed systems. In *Proceedings of the 26th International Conference on Software Engineering (ICSE)*. 418–427.

Received 1 September 2020; revised 14 February 2022; accepted 1 July 2022