Efficient Decentralized LTL Monitoring Framework Using Tableau Technique

Omar Al-Bataineh
National University of Singapore

David Rosenblum
National University of Singapore

Mark Reynolds
University of Western Australia

ABSTRACT
This paper presents a novel framework for decentralized monitoring of Linear Temporal Logic (LTL) formulas, under the situation where processes are synchronous and the formula is represented as a tableau. The tableau technique allows one to construct a semantic tree for the input LTL formula, which can be used to optimize the decentralized monitoring of LTL in various ways. Given a system \( P \) and an LTL formula \( \phi \), we construct a tableau \( T_{\phi} \). The tableau \( T_{\phi} \) is used for two purposes: (a) to synthesize an efficient round-robin communication policy for processes, and (b) to find the minimal ways to decompose the formula and communicate observations of processes in an efficient way. In our framework, processes can propagate truth values of both atomic and compound formulas (non-atomic formulas) depending on the syntactic structure of the input LTL formula and the observation power of processes. We demonstrate that this approach of decentralized monitoring based on tableau construction is more straightforward, more flexible, and more likely to yield efficient solutions than alternative approaches.

ACM Reference Format:

1 INTRODUCTION
Run-time verification (RV) has been recognized as one of the integral parts of software and hardware design process. RV is a lightweight formal method that aims to verify (at run-time) the conformance of the executions of the system under analysis with respect to some desired properties. Typically the system is considered as a black box that feeds events to a monitor. An event usually consists of a set of atomic propositions that describe some abstract operations in the system. Based on the received events, the monitor emits verdicts in a truth domain that indicate whether or not the run complies with the specification. The technique has been successfully applied to a number of industrial software systems, providing extra assurance of behavior correctness [8, 9, 18].

Recently, there has been a growing interest in run-time verification of distributed systems, that is, systems with multiple processes and no central observation point. Building a decentralized run-time monitor for a distributed system is a non-trivial task since it involves designing a distributed algorithm that coordinates the monitors in order to reason consistently about the temporal behavior of the system. The key challenge is that the monitors have a partial view of the system and need to account for communication and consensus.

In this work, we consider the decentralized monitoring of systems under the following setting: (a) processes are synchronous, and (b) the formula is expressed in a high-level requirement specification written in LTL and represented as a tableau. The tableau technique [20] has many applications in logic. It is used as a method of verifying whether a given formula is a tautology, as a method of proving semantical consistency of a set of formulas, and even as an algorithm for verifying the validity of arguments. Using the tableau technique in decentralized monitoring has several advantages. First, the tableau technique can be used to detect early whether the formula is a tautological formula or unsatisfiable formula. Second, it allows processes to propagate information about only feasible branches of the tableau (i.e. successful branches), where no information will be propagated about infeasible branches (i.e. failed branches). Third, it helps to find a minimal way to decompose the formula and communicate partial observations of processes in an efficient way.

The presented decentralized framework consists mainly of two parts: (a) a tableau-based algorithm that allows processes to compute at each state of the input execution trace the minimal set of formulas whose truth values need to be propagated, and (b) an LTL inference engine that allows processes to extract the maximal amount of information from a received message in case the message contains truth values of non-atomic formulas. We demonstrate that this approach of decentralized monitoring based on tableau construction is more straightforward, more flexible, and more likely to yield efficient solutions than alternative approaches.

Contributions. We summarize our contributions as follows.
- We present a new decentralized monitoring framework for LTL formulas under the assumption where processes are synchronous and the formula is represented as a tableau. The framework inherits known advantages of the tableau technique, LTL epistemic rules, and static monitoring approaches based on round-robin policies.
- We show how the tableau technique can be used to optimize decentralized monitoring of LTL (a) by finding the minimal ways to represent and decompose the global formula, and (b) by synthesizing efficient round-robin communication policies based on the semantics of the constructed tableau of the monitored formula.
- We develop an LTL inference engine that takes into consideration the observation power of processes and the syntactic
structure of the LTL formula being analyzed. The engine is used by processes to extract the maximal amount of information from the received messages. The inference engine operates with two categories of rules: (a) epistemic rules for propositional logic that are not dependent on the observation power of processes, and (b) epistemic rules for propositional logic that are dependent on the observation power of processes.

2 PRELIMINARIES

In this section, we give a brief review of the syntax and semantics of linear temporal logic LTL and the decentralized LTL monitoring problem. Then we review the basic tableau decomposition/expansion rules for LTL.

2.1 Decentralized LTL Monitoring Problem

A distributed program $P = \{p_1, p_2, ..., p_n\}$ is a set of $n$ processes which cooperate with each other in order to achieve a certain task. Distributed monitoring is less developed and more challenging than local monitoring: they involve designing a distributed algorithm that monitors another distributed algorithm. In this work, we assume that no two processes share a common variable. Each process of the distributed system emits events at discrete time instants. Each event $\sigma$ is a set of actions denoted by some atomic propositions from the set $AP$. We denote $2^{AP}$ by $\Sigma$ and call it the alphabet of the system. We assume that the distributed system operates under the perfect synchrony hypothesis, and that each process sends and receives messages at discrete instances of time, which are represented using identifier $t \in \mathbb{N}^0$.

Since each process sees only a projection of an event to its locally observable set of actions, we use a projection function $\Pi_i$ to restrict atomic propositions to the local view of monitor $M_i$ attached to process $p_i$, which can only observe those of process $p_i$. For atomic propositions (local to process $p_i$), $\Pi_i : 2^{AP} \rightarrow 2^{AP}$, and we denote $AP_i = \Pi_i(AP)$, for all $i = 1, ..., n$. For events, $\Pi_i : 2^\Sigma \rightarrow 2^\Sigma$, and we denote $\Sigma_i = \Pi_i(\Sigma)$ for all $i = 1, ..., n$. We assume that $\forall i,j \leq n, i \neq j \Rightarrow AP_i \cap AP_j = \emptyset$ and consequently $\forall i,j \leq n, i \neq j \Rightarrow \Sigma_i \cap \Sigma_j = \emptyset$. That is, events are local to the processes where they are monitored. The system’s global trace, $g = (g_1, g_2, ..., g_n)$ can now be described as a sequence of pair-wise unions of the local events of each process’s traces. Finite traces over an alphabet $\Sigma$ are denoted by $\Sigma^*$, while infinite traces are denoted by $\Sigma^\omega$.

Definition 1. (LTL formulas [19]). The set of LTL formulas is inductively defined by the grammar

$$\varphi ::= true \mid c \mid \neg \varphi \mid \varphi \lor \varphi \mid X\varphi \mid F\varphi \mid G\varphi \mid \varphi U \varphi$$

where $X$ is read as next, $F$ as eventually (in the future), $G$ as always (globally), $U$ as until, and $c$ is a propositional variable.

Definition 2. (LTL Semantics [19]). Let $w = a_1 a_2 ... \in \Sigma^\omega$ be a infinite word with $i \in N$ being a position. Then we define the semantics of LTL formulas inductively as follows

- $w, i \models true$
- $w, i \models \neg \varphi$ iff $w, j \not\models \varphi$ for some $j \geq i$
- $w, i \models \varphi$ iff $w, j \models \varphi$ for all $j \geq i$
- $w, i \models \varphi_1 U \varphi_2$ iff $\exists k \geq i$ with $w, k \models \varphi_2$ and $\forall 1 \leq l < k \ w, l \models \varphi_1$
- $w, i \models X \varphi$ iff $w, i + 1 \models \varphi$

We now review the definition of three-valued semantics LTL$ \omega$ that is used to interpret common LTL formulas, as defined in [4]. The semantics of LTL$ \omega$ is defined on finite prefixes to obtain a truth value from the set $\mathbb{B}_3 = \{ \top, \neg, ? \}$.

Definition 3. (LTL$ \omega$ semantics). Let $u \in \Sigma^*$ denote a finite word. The truth value of a LTL$ \omega$ formula $\varphi$ with respect to $u$, denoted by $[u \models \varphi]$, is an element of $\mathbb{B}_3$ defined as follows:

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

Note that according to the semantics of LTL$ \omega$ the outcome of the evaluation of $\varphi$ can be inconclusive (?). This happens if the so far observed prefix $u$ itself is insufficient to determine how $\varphi$ evaluates in any possible future continuation of $u$.

Problem 1. (The decentralized monitoring problem). Given a distributed system $P = \{p_1, p_2, ..., p_n\}$, a finite global trace $a \in \Sigma^*$, an LTL property $\varphi$ with a set of atomic propositions $AP$ formulating a requirement over the system global behaviour, and a set of monitor processes $M = \{M_1, M_2, ..., M_n\}$ such that

- each process $p_1$ has a local set of propositions $AP_1 \subset AP$, and
- each process $p_1$ has a local monitor $M_1$, and
- each process $p_1$ has a partial view of the global trace $a$, and
- monitor $M_1$ can read truth values of $AP_1$, and
- monitor $M_1$ can communicate with the other monitors.

The decentralised monitoring problem aims then to design an algorithm for distributing and monitoring $\varphi$, such that satisfaction or violation of $\varphi$ can be detected by local monitors alone.

The main constraint that decentralised LTL monitoring addresses is the lack of a global sensor or monitor and a central decision making point asserting whether the system’s global trace $a$ has violated or satisfied the property $\varphi$. One key metric when analysing the efficiency of a decentralised monitoring algorithm is the communication complexity of the algorithm: the number and size of messages needed to reach a verdict for the monitored property. To develop a decentralised monitoring solution one needs to address three issues regarding communication: (i) which processes communicate to which (i.e., what is the communication model used by processes), (ii) when they communicate, and (iii) what they communicate (how observations of processes are propagated). We discuss how we address these issues in our framework at Sections 5, 6, and 7.

Note that the decentralise monitoring problem can be studied under different settings and different assumptions. However, in this work, we make a number of assumptions about the class of systems that can be monitored in our framework.

- $A1$: the monitored system is a synchronous timed system with a global clock; and
- $A2$: processes know the “observation power” of each other.

That is, if process $A$ observes the atomic proposition $a$ then process $B$ knows that $A$ observes $a$; and
Declarative LTL Monitoring Framework Using Tableau

- **A3**: processes are reliable (i.e., no process is malicious and that processes do not crash); and
- **A4**: The communication model used by processes is mainly based on a predefined static topology that does not change between the global states of the trace. So that the receiving process of observations of process \( p \) is fixed along the entire monitored trace; and
- **A5**: Each process can send and receive only one message at any communication round.

By using a static communication topology based on a round-robin scheduling policy we ensure that all processes will eventually reach a verdict about the global property without necessarily using a broadcast communication model. This also ensures the global termination of the monitor processes. Furthermore, processes can compute the knowledge states of their neighbour processes and send only update messages with new observations. This is mainly due to the fact that observations will be circulated among all the processes. The broadcast model can solve the problem by reaching a verdict for the monitored property in a faster way but the communication cost can be extremely expensive and it can overwhelm the network specially when dealing with large-scale systems.

### 2.2 Tableau Construction for LTL

Among the various existing tableau systems for LTL [6, 12, 15, 20, 22, 24] we selected Reynolds’s implicit declarative one [20]. The interesting completeness and termination of the tableau, in addition to its efficiency and simplicity are the key reasons for choosing this style of tableau. The tableau is unique in that it is wholly traditional in style (labels are sets of formulas), it is tree shaped tableau construction, and it can handle repetitive loops efficiently.

Given an LTL formula \( \varphi \) we construct a directed graph (tableau) \( T_\varphi \) using the standard expansion rules for LTL. Applying expansion rules to a formula leads to a new formula but with an equivalent semantics. We review here the basic expansion rules of propositional logic that can be used by processes to deduce definite truth values that were not already satisfied between the global states of the trace. So that the receiving process of observations of process \( p \) is fixed along the entire monitored trace; and

**Tableau Expansion Rules for Propositional Logic**

1. **Gp**: add to its leaf the chain of two nodes containing the formulas \( A \) and \( B \).
2. **Fp**: add to its leaf the chain of two nodes containing the formulas \( A \) and \( \neg B \).
3. **Xp**: add to its leaf the chain of two nodes containing the formulas \( A \) and \( Xl \).
4. **p \land (q \lor r)**: add to its leaf the chain of two nodes containing the formulas \( p \), \( q \), and \( r \).
5. **p \lor q**: add to its leaf the chain of two nodes containing the formulas \( p \) and \( q \).
6. **\neg p**: add to its leaf the chain of two nodes containing the formulas \( \neg p \).

#### Figure 1: A tableau for \( (p \land (q \lor r)) \)

- **Figure 2: A tableau for Gp**

Reynolds [20] introduced a new tableau rule (the PRUNE rule) which supports a new simple traditional style tree-shaped tableau for LTL. The PRUNE rule provides a simple way to curtail repetitive branch extension. The PRUNE rule works as follows. If a node at the end of a branch has a label which has appeared already twice above, and between the second and third appearance there are no new eventualities satisfied that were not already satisfied between the first and second appearances then that whole interval of states (second to third appearance) has been useless. In this case we cut the construction and declare that the branch is unsuccessful.

Using the PRUNE rule and the LOOP rule (a rule that cuts construction after a poised label appears two times in the branch) we guarantee completeness and termination of the tableau construction (i.e., it always terminates and returns a semantic graph for the monitored formula including formulas containing nested temporal operators) [20]. For example, the formula \( Gp \) (see Fig. 2) gives rise to a very repetitive infinite tableau without the LOOP rule, but succeeds quickly with it. We first break down the formula into its elementary ones. Note that the atoms and their negations can be satisfied immediately provided there are no contradictions, but to reason about the X formula \( XGp \) we need to move forwards in time. Reasoning switches to the next time point and we carry over only information nested below X. However, as one can see the label \( (p, XGp) \) is repeated two times and hence a fixed-point is reached with no contradictions. In this case we stop the analysis and declare that the tableau is successful.

### 3 DECENTRALIZED LTL INFERENCE ENGINE

In this section we describe an inference engine (a set of IF-THEN epistemic rules) which consists of a number of epistemic rules on propositional logic that can be used by processes to deduce definite truth values of propositions in compound formulas. The engine performs syntactic decomposition of the compound formula using tableau decomposition rules. Before formalizing the epistemic rules, we introduce some notations:

- **obs(\( p \))** be the set of propositions whose definite truth values are known to process \( p \) either by direct observation or remote observation through communication;
- **LOP(\( \phi \))**, the set of logical operators of formula \( \phi \);
- **TOP(\( \phi \))**, the set of temporal operators of formula \( \phi \); and
Atoms(ϕ), the set of positive atomic propositions in ϕ and Abs(ϕ) the set of negative atomic propositions in ϕ. For example, for ϕ = ((a ∧ b) ∨ (¬c ∧ d)) we have Atoms(ϕ) = {a, b, d} and Abs(ϕ) = {c}.

Our decentralized monitoring algorithm is mainly based on a three-valued semantics for the future linear temporal logic LTL, where the interpretation of the third truth value, denoted by ?, follows Kleene logic [16]. For example, a monitor might not know the Boolean value of a proposition at a time point because it is not within its observation power. In this case, the monitor assigns the proposition the truth value ?. Note that no verdict is produced if under the current knowledge the specification evaluates to ?.

We can then summarize the basic cases for three-valued logical operators involving (?) as follows: (T ∧ ?) = ?, (⊥ ∧ ?) = ⊥, (? ∧ ?) = ?, (T ∨ ?) = T, (⊥ ∨ ?) = ?, (? ∨ ?) = ?, (?) = ?. Epistemic rules are written in the following form

Premise1; ...; Premise_n

Conclusion_1 ∧ ... ∧ Conclusion_n

We write (ϕ)^p_l to denote that the formula ϕ has been evaluated at p ∈ B_3 by process p. We classify the rules into two categories: (a) epistemic rules on propositional logic that are not dependent on the observation power of processes, and (b) epistemic rules on propositional logic that are dependent on the observation power.

### Epistemic rules on propositional logic not dependent on the observation power of processes

Rules R1-R2 are straightforward rules which take advantage of the semantics of logical operators (∧, ∨) to deduce definite truth values of propositions in a formula.

\[(a_1 ∧ a_2 ∧ \ldots ∧ a_n)^T \land (V_{i=1...n}(a_i = T))\]

\[(a_1 ∨ a_2 ∨ \ldots ∨ a_n)^T \land (V_{i=1...n}(a_i = ⊤))\]

**R1:**

\[
(V_{i=1...n}(a_i = T)) \\
(V_{i=1...n}(a_i = ⊤))
\]

**R2:**

### Epistemic rules on propositional logic dependent on the observation power of processes

An interesting set of epistemic rules on propositional logic can be developed if we allow processes to take into consideration the observation power of the sending processes when extracting information from the received compound formulas, as described in rules R3-R10. The rules given in this category are straightforward rules except rules R9 and R10, which we will explain later by an example. The rules work as follows: depending on the syntactic structure of the compound formula ϕ and its truth value as evaluated by a sending process p, a receiving process q can deduce definite truth values of some propositions in ϕ. Therefore, these rules should be viewed as epistemic rules used by process q after receiving a message from process p containing truth values of compound formulas.

\[(a_1 ∧ a_2 ∧ \ldots ∧ a_n)^T \land ((a_1, \ldots, a_n) \cap obs(p)) \neq ∅ \land (V_{i=1...n}(a_i = T)) \land (V_{i=1...n}(a_i = ⊤))\]

\[
(V_{i=1...n}(a_i = ⊤))q \land (V_{i=1...n}(a_i = T))q
\]

\[(a_1 ∨ a_2 ∨ \ldots ∨ a_n)^T \land ((a_1, \ldots, a_n) \cap obs(p)) \neq ∅ \land (V_{i=1...n}(a_i = ⊤)) \land (V_{i=1...n}(a_i = T)) \land (V_{i=1...n}(a_i = ⊤))q \land (V_{i=1...n}(a_i = T))q\]

**R3:**

**R4:**

R5:

\[(a ∨ (b ∧ c))^T \land (a ∈ obs(p)) \land c \notin obs(p) \land (V_{i=1...n}(a_i = T))q \land (V_{i=1...n}(a_i = ⊤))q\]

**R6:**

\[(a ∨ (b ∧ c))^T \land (a ∈ obs(p)) \land (V_{i=1...n}(a_i = T))q \land (V_{i=1...n}(a_i = ⊤))q \land (V_{i=1...n}(a_i = ⊤))q\]

**R7:**

\[(a ∨ (b ∧ c))^T \land (a ∈ obs(p)) \land (V_{i=1...n}(a_i = T))q \land (V_{i=1...n}(a_i = ⊤))q \land (V_{i=1...n}(a_i = ⊤))q\]

**R8:**

\[(a ∨ (b ∧ c))^T \land (a ∈ obs(p)) \land (V_{i=1...n}(a_i = T))q \land (V_{i=1...n}(a_i = ⊤))q \land (V_{i=1...n}(a_i = ⊤))q\]

**R9:**

\[(a ∨ (b ∧ c))^T \land (a ∈ obs(p)) \land (V_{i=1...n}(a_i = T))q \land (V_{i=1...n}(a_i = ⊤))q \land (V_{i=1...n}(a_i = ⊤))q\]

**R10:**

We now turn to discuss rules R9 and R10. Note that rule R10 is the dual of rule R9 and we therefore explain only R9. The rule can be used if the syntactic structure of the complex compound formula Φ = (ϕ_1 ∨ ... ∨ ϕ_n) satisfies the following conditions: (a) the only logical operator appearing in subformula ϕ_i is (∧), (b) ϕ_i has no temporal operator (c) process p observes at most one proposition in each subformula ϕ_i, and (d) Φ is evaluated to ⊤ by p. Then once a process q receives a message from p containing the truth value of Φ it can deduce that all positive propositions in Φ observed by p have truth values ⊤ and all negative propositions in Φ observed by p have truth values ⊥. For example, consider Φ = ((a ∧ b) ∨ (¬c ∧ d)), where process p observes {a, c} and evaluates Φ to ⊤. Then once q receives a message from p containing the truth value of Φ it deduces that a = ⊤ ∧ c = ⊤.

Thanks to above inference engine (the epistemic rules R1-R10), the processes can send the truth value of an atomic proposition without explicitly sending it. However, in some cases processes may need to apply multiple epistemic rules on the same compound formula depending on the syntactic structure of the formula. So that if the compound formula can be decomposed into multiple compound formulas of the forms given in rules R1-R10, then multiple epistemic rules will be applied to derive definite truth values of the propositions in the original compound formula.

**Example 1.** Suppose we have a compound formula Φ = ((a ∧ b) ∨ (c ∧ (d ∨ e))) and that a process p has AP_p = {a, d, e}. Suppose that at time step t the process p observes that a = d = e = ⊤ and hence evaluates the formula Φ to ⊤. Suppose further that p propagates the truth value of Φ to some other process q which knows that p observes the atoms {a, d, e}. Note that Φ can be decomposed into two compound subformulas ϕ_1 = (a ∧ b) and ϕ_2 = (c ∧ (d ∨ e)). Since p told q that
\[\Phi \equiv \bot \text{ then } q \text{ can deduce that } \phi_1 = \bot \text{ and } \phi_2 = \bot. \text{ In this case process } q \text{ can deduce that } a = \bot \land d = \bot \land e = \bot.\]

4 \ THE OBSERVATION POWER OF PROCESSES

In this section, we discuss the problem of computing the observation power of processes in a decentralized fashion, the set \(ob\(s(p_i)\)) (i.e. the set of atomic propositions whose truth value are known to process \(p_i\) up to the current time step \(t\)). The first intuitive way of computing the observation power of a process \(p_i\) is to assume that \(p_i\) knows the truth values of its local atomic propositions from the initial step up to the current step. We formalize this as follows:

\[
\text{obs}(p_i) = AP^{(0)}_{p_i} \cup ... \cup AP^{(t)}_{p_i}
\]

The above formula does not take into consideration the fact that processes communicate with each other and that the observation power of processes can be enhanced through communication. It is thus not limited to their local atomic propositions. However, computing the set \(ob\(s(p_i)\)) while considering remote observations made by \(p_i\) is crucial in our decentralized monitoring framework as it makes the epistemic rules R3-R10 more powerful in the sense that more information may be extracted when applying these rules.

Since processes use static communication scheme in which the order at which observations of processes are propagated is fixed between states, they can then compute precisely the observation power of each other. Let \(|\pi_{i,j}|\) be the number of communication rounds needed for process \(p_i\) to influence the information state of process \(p_j\), where \(|\pi_{i,j}|\) represents the length of the communication path between nodes \((p_i, p_j)\). Note that in our framework we assume that each process can send at most one message at any one communication round. Therefore, when process \(p_i\) makes a new observation \(O_t\) at time step \(t\), then process \(p_j\) knows about \(O_t\) at time step \((t + |\pi_{i,j}|)\). So given a static communication scheme in which the length of communication path \(|\pi_{i,j}|\) for any pair \((p_i, p_j)\) is prior knowledge, the observation power of a process \(p_i\) at time step \(t \geq 0\) can then be computed as follows:

\[
\text{obs}(p_i) = (\{AP^{(0)}_{p_i}\} \cup \ldots \cup AP^{(t)}_{p_i}) \cup (\forall \tau = 0, \ldots, t \land 0 \land \ldots \land \exists \pi_{i,j}\{\bigcup_{\tau = 0}^{t} |\pi_{i,j}|\} \{AP^{(\tau)}_{p_i}\}))
\]

That is, at time step \(t\), process \(p_i\) knows the truth values of its local propositions from time 0 to time \(t\) and the true values of all propositions of process \(p_j\) from time 0 to time \(t\), given that \((t - \tau) \geq |\pi_{i,j}|\). To demonstrate how processes can compute the observation power of each other, let us consider Example 2.

Example 2. Suppose that processes \(\{A, B, C\}\) communicate with each other using the fixed communication scheme \(\text{comm} = (A \rightarrow B \rightarrow C \rightarrow A)\) and that \(\text{comm}\) is common knowledge among processes, where the direction of the arrow represents the direction of communication. We can then compute for instance the observation power of processes \(A, B\) and \(C\) up to step \(t = 2\) as follows:

\[
\text{obs}(A) = \{AP^{(0)}_A, AP^{(1)}_A, AP^{(2)}_A, AP^{(0)}_B, AP^{(1)}_B, AP^{(2)}_B, AP^{(0)}_C, AP^{(1)}_C, AP^{(2)}_C\}
\]

\[
\text{obs}(B) = \{AP^{(0)}_A, AP^{(1)}_A, AP^{(2)}_A, AP^{(0)}_B, AP^{(1)}_B, AP^{(2)}_B, AP^{(0)}_B, AP^{(1)}_C, AP^{(2)}_C\}
\]

\[
\text{obs}(C) = \{AP^{(0)}_A, AP^{(1)}_A, AP^{(2)}_A, AP^{(0)}_B, AP^{(1)}_B, AP^{(2)}_B, AP^{(0)}_C, AP^{(1)}_C, AP^{(2)}_A\}
\]

Note that when the set \(\text{obs}(B)\) contains the set \(AP^{(0)}_A\), it means that \(B\) knows the truth values of all propositions in the set \(AP^{(0)}_A\) at time step 0. To demonstrate how the epistemic rules can be used in decentralized monitoring, let us consider the following example.

Example 3. Suppose that processes \(\{A, B, C\}\) communicate with each other using the static communication scheme \(\text{comm} = (A \rightarrow B \rightarrow C \rightarrow A)\) and that \(\text{AP}_A = \{a_1, a_2\}, AP_B = \{b\}, \text{and } AP_C = \{c\}\). The property to be monitored by processes is \(\phi = F(a_1 \land \neg a_2 \land b \land \neg c)\).

Let us denote the formula \(\phi = F(a_1 \land \neg a_2 \land b \land \neg c)\) by \(\phi\). We first construct a tableau for \(\phi\) using the tableau system of Section 2.2 (see Fig. 3).

Suppose that at state \(g_0\) (the initial state) the propositions of the formula have the following truth values: \(a_1 = \top, a_2 = \bot, b = \top\), and \(c = \bot\). For brevity, we describe only the information propagated from process \(A\) to process \(B\) concerning state \(g_0\).

\[
\begin{align*}
&F\phi \\
&\quad \rightarrow \\
&\quad \phi \lor XF\phi \\
&\quad a_1, \neg a_2, b, \neg c \\
&F\phi \\
&\quad \rightarrow \\
&\quad \phi \lor XF\phi
\end{align*}
\]

Figure 3: A tableau for \(F\phi\), where \(\phi = (a_1 \land \neg a_2 \land b \land \neg c)\)

- At time step \(t = 0\) process \(A\) sends \((\phi(g_0) = \top)\) to \(B\). Then using rule R3 process \(B\) deduces that \(a_1 = \top\) and \(a_2 = \bot\) at state \(g_0\) as \(A\) observes both \(a_1, a_2\).
- At time step \(t = 1\) process \(A\) sends \((\phi(g_0) = \top)\) to \(B\). Then using rule R3 again process \(B\) deduces that \(a_1 = \top\) and \(a_2 = \bot\) at state \(g_0\) as \(A\) observes \(a_1, a_2\) and has already received a message from \(C\) at time step \(t = 0\) about the truth value of variable \(c\) at state \(g_0\). At this step, process \(B\) knows truth values of all variables at state \(g_0\) as \(B\) is locally observed by \(B\).

5 \ PROPAGATING OBSERVATIONS IN AN EFFICIENT WAY

In this section we describe an algorithm that allows processes to propagate their observations in an efficient way. Suppose that we use a round-robin communication policy in which process \(p\) sends its observations to process \(q\). Then instead of allowing \(p\) to propagate its entire knowledge to \(q\), it propagates only the set of observations that are not known to \(q\). We measure by observations the set of atomic propositions whose definite truth values are known to \(p\) but not to \(q\). Note that process \(p\) knows the set of propositions whose definite truth values are known to \(q\) from time 0 to time \(t\). This is due to the assumption that processes use a static communication scheme in which the order at which observations are propagated is fixed between states. Let \(\text{MinList}\) be the minimal set of formulas whose truth values need to be propagated from step \(t\) (the earliest step at which a new observation has been made by \(p\)) to step \(t\) (the
current step of the trace). However, to compute the list MinList process \( p \) needs to follow the following steps.

1. For each \( k \in [\tau, t] \) process \( p \) computes the set \( M_k = (\text{obs}^{[k]}(p)) \setminus \text{obs}^{[k]}(q) \).
2. For each \( M_k \) process \( p \) computes the set MinList\(_k\) using Algorithm 1.
3. It then combines the sets MinList\(_k\) for all \( k \in [\tau, t] \) to obtain MinList.

In our setting we assume that each process maintains a truth table at each state of the monitored trace, which consists of the set of formulas in the resulting tableau of the monitored formula. The truth tables maintain only the set of non-atomic formulas that match the syntactic structure of formulas in the epistemic rules R1-R10, in addition to the set of atomic propositions of the main formula. At each time step \( \tau \), process \( p \) examines the non-atomic formulas in its truth tables to compute the minimal set of formulas whose truth values need to be propagated (see Algorithm 1). The truth table consists of three columns: the set of formulas, their truth values, and their unique index values. We view the table \( T_{\tau} \) as a set of entries of the form \((f, val, index)\). The set CompoundsFormulas used in the algorithm represents the set of compound formulas in the epistemic rules R1-R10. The operation MatchSynt\((\phi_i, \phi_j)\) is a Boolean operation that checks whether the two formulas \( \phi_i \) and \( \phi_j \) have the same syntactic structure. The operation DeductionSet\((\phi)\) returns the set of atomic propositions in the compound formula \( \phi \) whose definite truth values can be deduced. As one can see Algorithm 1 consists of two phases: (a) the exploration phase in which non-atomic formulas in the truth table \( T_{\tau} \) are examined, and (b) the refinement phase in which redundant formulas in the list MinList\(_{\tau}\) (if any) are removed. Note that it is possible to have a formula whose deduction list is subset of the deduction list of another formula or they have some atomic propositions in common. The goal of the refinement phase is then to detect and remove redundant information so that we ensure that observations are propagated in an efficient way.

Algorithm 1 aims to propagate observations of processes compactly by allowing processes to send truth values of compound formulas from which definite truth values of atomic formulas can be deduced. However, in some cases it may not be possible to deduce truth values of atomic formulas from the given compound formulas in the epistemic rules, as this depends on the outcome of decomposition of the main formula and the truth values of compound formulas at each state of the trace. In such cases the process needs to add the truth value of these atomic variables to the messages propagated to the receiving process (see lines 22-23 of algorithm 1).

**Theorem 1. (Soundness of Algorithm 1).** Let \( P = \{p_1, ..., p_n\} \) be a distributed system and \( \varphi \) be an LTL formula formalising a requirement over the system’s global behaviour. Let \( M_\tau \) be the set of variables whose truth values need to be propagated from process \( p \) to its neighbour process at state \( \tau \) and MinList\(_{\tau}\) be the set of formulas that \( p \) needs to communicate at state \( \tau \) as computed by Algorithm 1. Then the set MinList\(_{\tau}\) satisfies the following properties:

1. **Correctness.** Truth values of all variables in \( M_\tau \) can be deduced from the set MinList\(_\tau\).
2. **Minimality.** MinList\(_{\tau}\) is the minimal set of formulas from which truth values of all variables in \( M_\tau \) can be deduced.

The first property of the theorem concerns correctness of Algorithm 1 so that information needed to evaluate the global formula will be shared among processes. The minimality property (the second property) ensures efficiency rather than correctness so that messages propagated by processes contain no redundant information or information that is already known to the receiving process from previous communication rounds. One of the key advantages of the presented algorithm is that processes may not need to communicate if communication does not change their states. That is, when \( \text{obs}(p) \setminus \text{obs}(q) = \emptyset \) at some step \( t \) (i.e., no new observation has been made by \( p \)) then \( p \) needs not to communicate with \( q \) at \( t \).

### 6 STATIC MONITORING APPROACHES

The knowledge state of processes in decentralized monitoring increases monotonically over time due to local and remote observations. It is therefore necessary to have an efficient communication strategy that allows processes to propagate only necessary observations. The propagation of monitoring information from one peer to the other in our framework follows a static communication strategy based on a round-robin scheduling policy. The advantage of using a static communication strategy in decentralized monitoring is that processes can compute the knowledge state of each other and communicate only new observations to their neighbour processes. This helps to reduce significantly the size of propagated messages, which is crucial when monitoring large-scale distributed systems (systems with huge number of processes).

To develop efficient static round-robin policies for decentralized monitoring we use ranking functions to rank processes (i.e. assign a unique value to each process) using some interesting criteria that take into consideration the observation power of processes and the syntactic structure of the formula. The values assigned to the processes using ranking functions will be used then to specify the order of communication in the round-robin policy. We consider here two different ranking strategies.

1. For the first strategy, a process that contributes to the truth value of the formula via a larger number of propositions receives higher priority in the round-robin order; if multiple processes have the same number of propositions, then the order is fixed by an (externally provided) process ID (PID). We therefore guarantee that the order of communication that results from the strategy is unique. The intuition behind choosing such a strategy is that processes that observe more propositions of the formula own more information about the global trace of the system, and hence specifying the order of communication in such a way may help to detect violations somewhat faster.

2. For the second strategy the tableau is considered, where we rank processes based on their observation power while taking into consideration the structure of the formula \( \varphi \). Let \( T_\varphi \) be a tableau of \( \varphi \). Then the process that contributes to a larger number of branches receives higher priority in the order of communication. Note that the tableau technique allows one to construct a semantic tree for the monitored formula, where each branch of the tree represents a way to satisfy the formula. We believe that this strategy would help to speed up the RV process.
Decentralized LTL Monitoring Framework Using Tableau

| 1: Inputs : \((M_r, T_r, \text{CompoundFormulas})\) |
| 2: Output : \(\text{MinList}_r = \emptyset\) |
| 3: for each entry \(\in T_r\) do \(\rightarrow\) Exploration phase |
| 4: if \(M_r \cap \text{Atoms(entry.f)} \neq \emptyset\) then |
| 5: if MatchSynt(entry.f, \(\phi\)) for any \(\phi \in \text{CompoundFormulas}\) then |
| 6: if entry.val = \(\phi\).val then |
| 7: DeductionSet(entry.f) \(\subseteq\) DeductionSet(\(\psi\)) for any \(\psi \in \text{MinList}_r\) then |
| 8: add (entry.index, entry.val) to \(\text{MinList}_r\) |
| 9: end if |
| 10: end if |
| 11: end if |
| 12: end if |
| 13: end for |
| 14: for each \(\phi \in \text{MinList}_r\) do \(\rightarrow\) Refinement phase |
| 15: if DeductionSet(\(\phi\)) \(\subseteq\) DeductionSet(\(\psi\)) for any \(\psi \in \text{MinList}_r\) then |
| 16: remove \(\phi\) from \(\text{MinList}_r\) |
| 17: end if |
| 18: if DeductionSet(\(\phi\)) \(\subseteq\) \((\bigcup \psi \in \text{MinList}_r )\text{DeductionSet}(\(\psi\))\)) then |
| 19: remove \(\phi\) from \(\text{MinList}_r\) |
| 20: end if |
| 21: end for |
| 22: if there exist \(a \in M_r\) such that \(a \notin \bigcup \phi \in \text{MinList}_r\text{DeductionSet}(\(\phi\))\)) then |
| 23: add (index\(a\), val\(a\)) to \(\text{MinList}_r\) |
| 24: end if |
| 25: return \(\text{MinList}_r\) |

Algorithm 1: Computing minimal set of formulas for process \(p\) at step \(r\)

**Example 4.** Suppose we have a distributed system \(S = \{A, B, C\}\) and a property \(\varphi = G(a_1 \land \sim a_2 \land b_1) \lor F(b_2 \land c)\) of \(S\) that we would like to monitor in a decentralized fashion, where \(AP_A = \{a_1, a_2\}\), \(AP_B = \{b_1, b_2\}\), and \(AP_C = \{c\}\). From the syntactic structure of \(\varphi\), it is easy to see that the tableau \(T_p\) constructed consists of three branches, where one branch corresponds to formula \(G(a_1 \land \sim a_2 \land b_1)\) and two branches correspond to formula \(F(b_2 \land c)\). Hence the two presented strategies yield different RR policies as follows

\[
\text{Comm}_1 = A \rightarrow B \rightarrow C \rightarrow A
\]

\[
\text{Comm}_2 = B \rightarrow C \rightarrow A \rightarrow B
\]

It is interesting to note that the communication topology used by processes in our framework can be constructed by each process independently and hence each process knows its predecessor and its successor. This is possible since processes share the same tableau graph and that the observation power of processes are common knowledge.

Furthermore, each process will be assigned a unique ranking value which will be used to specify the order of communication and hence the communication topology will be unique. These ranking values are computed based on the constructed tableau and the observation power of processes as described above. However, arbitrary topologies other than those described above are also possible as long as all relevant information is transmitted and the topology is complete (i.e., all processes that contribute to the truth value of the formula are part of the topology).

7 DECENTRALIZED MONITORING ALGORITHM

Our monitoring algorithm consists of two phases: setup and monitor. The setup phase creates the monitors and defines their communication topology. The monitor phase allows the monitors to begin monitoring and propagating information to reach a verdict when possible. We first describe the steps of the setup phase.

- Each process constructs a tableau \(T_p\) for \(\varphi\) using the method of Section 2.2.
- Each process then refines the constructed tree \(T_p\) by removing redundant formulas and infeasible branches from the tree. Note that in some cases the constructed tableau of an LTL formula may contain redundant formulas due to repetitive loops (see Fig. 2).
- Each process constructs a truth table that consists of compound formulas in \(T_p\) that match the syntactic structure of the formulas in the epistemic rules \(R1-R10\), in addition to the atomic formulas of \(\varphi\).
- Each process assigns a unique index value to each formula in the constructed truth table. We assume here that all processes use the same enumerating procedure when assigning index values to the formulas.

The advantage of assigning unique index values to the formulas in the constructed truth table is that it allows processes to propagate truth values of formulas as pairs of the form \((idx(\phi), val)\), where \(idx(\phi)\) is the index value of the formula \(\phi\) and \(val \in B_3\), which helps to reduce the size of propagated messages. Recall also that processes

### Example 4

Suppose we have a distributed system \(S = \{A, B, C\}\) and a property \(\varphi = G(a_1 \land \neg a_2 \land b_1) \lor F(b_2 \land c)\) of \(S\) that we would like to monitor in a decentralized fashion, where \(AP_A = \{a_1, a_2\}\), \(AP_B = \{b_1, b_2\}\), and \(AP_C = \{c\}\). From the syntactic structure of \(\varphi\), it is easy to see that the tableau \(T_p\) constructed consists of three branches, where one branch corresponds to formula \(G(a_1 \land \neg a_2 \land b_1)\) and two branches correspond to formula \(F(b_2 \land c)\). Hence the two presented strategies yield different RR policies as follows:

\[
\text{Comm}_1 = A \rightarrow B \rightarrow C \rightarrow A
\]

\[
\text{Comm}_2 = B \rightarrow C \rightarrow A \rightarrow B
\]

It is interesting to note that the communication topology used by processes in our framework can be constructed by each process independently and hence each process knows its predecessor and its successor. This is possible since processes share the same tableau graph and that the observation power of processes are common knowledge. Furthermore, each process will be assigned a unique ranking value which will be used to specify the order of communication and hence the communication topology will be unique. These ranking values are computed based on the constructed tableau and the observation power of processes as described above. However, arbitrary topologies other than those described above are also possible as long as all relevant information is transmitted and the topology is complete (i.e., all processes that contribute to the truth value of the formula are part of the topology).
in our framework can propagate truth values of compound formulas (using the epistemic rules R1-R10), which helps to reduce further the size of propagated messages. We now summarize the actual monitoring steps in the form of an explicit algorithm that describes how local monitors operate and make decisions:

1. [Read next event]. Read next $\sigma_i \in \Sigma_i$ (initially each process reads $\sigma_0$).
2. [Compute minimal set of formulas to be transmitted]. Use Algorithm 1 to derive $\text{MinList}_p^\varphi_i$.
3. [Send truth value of formulas in $\text{MinList}_p^\varphi_i$]. Propagate the truth value of formulas in $\text{MinList}_p^\varphi_i$ as pairs of the form $(\text{id}(\varphi), \text{val})$ to the receiving process.
4. [Receive new observations and update monitoring table]. Receive new observations from your neighbour process and update the truth table according to the new observations.
5. [Evaluate the formula $\varphi$ and return]. If a definite verdict of $\varphi$ is found return it: That is, if $\varphi = \top$ return $\top$, if $\varphi = \bot$ return $\bot$.
6. [Go to step 1]. If the trace has not been finished or a decision has not been made then go to step 1.

We now turn to discuss the basic properties of our decentralized monitoring framework. Let $\models_D$ be the satisfaction relation on finite traces in the decentralized setting and $\models_C$ be the satisfaction relation on finite traces in the centralized setting, where both $\models_D$ and $\models_C$ yield values from the same truth domain. Note that in a centralized monitoring algorithm we assume that there is a central process that observes the entire global trace of the system being monitored, while in our decentralized monitoring algorithm processes observe part of the trace, perform remote observation, and use some deduction rules in order to evaluate the property. The following theorems state the soundness and completeness of our decentralized monitoring algorithm.

**Theorem 2. (Soundness).** Let $\varphi \in \text{LTL}$ and $\alpha \in \Sigma^*$. Then $\alpha \models_D \varphi : B \Rightarrow \alpha \models_C \varphi : B$, where $B \in \{\top, \bot\}$.

Soundness means that all verdicts (truth values taken from a truth-domain) found by the decentralized monitoring algorithm for a global trace $\alpha$ with respect to the property $\varphi$ are actual verdicts that would be found by a centralized monitoring algorithm that have access to the trace $\alpha$.

**Theorem 3. (Completeness).** Let $\varphi \in \text{LTL}$ and $\alpha \in \Sigma^*$. Then $\alpha \models_C \varphi : B \Rightarrow \exists \alpha' \in \Sigma^*. |\alpha'| \leq n \wedge \alpha.\alpha' \models_D \varphi : B$, where $n$ is the number of processes in the monitored system and $B \in \{\top, \bot\}$.

Completeness means that all verdicts found by the centralized monitoring algorithm will be found by the decentralized monitoring algorithm but not necessarily at the same time (i.e., after consuming the same number of events). That is, the decentralized algorithm reaches the same verdict as the centralized algorithm but with some bounded delay $\tau \leq n$. This is mainly due to the distribution of information and communications. The soundness and completeness of the presented framework can be shown using Theorem 1 and the fact that observations of processes are propagated in a circular fashion using an RR communication policy.

**Monitoring formulas with nested operators.** Our framework can also handle formulas with nested operators (e.g., $G(\varphi \Rightarrow F\psi)$), where the operators are expanded repeatedly using usual tableau expansion rules and the new LOOP checking rule and PRUNE rules introduced by Reynolds which can be used to halt the expansion of infinite branches. The process stops at trivial cases or when a certain loop condition is met (i.e. a fixed point has been reached). Processes can then exchange truth values of atomic formulas or compound formulas depending on the tableau shape of the main formula.

**Message Complexity of The Algorithm.** To analyse the message complexity of the algorithm we need to analyse both the communication topology used by processes and the size of propagated messages at each global state of the monitored trace. Let us assume that we have $n$ monitor processes and a global property $\varphi$ with a set of atomic propositions $\text{AP}$ and a trace $\alpha$ of a distributed system $\mathcal{P}$. Before discussing the complexity of the algorithm let us first recall the following properties of the algorithm:

- processes propagate their observations in a circular way using a round-robin scheduling policy. The topology consists of $n$ nodes representing the distributed monitor processes.
- each process sends one message at each state of the trace.
- each process receives one message at each state of the trace.

This makes it a total of $|\alpha| \times n$ messages throughout the algorithm. Recall also that processes can send truth values of both atomic formulas and compound formulas as described in the inference engine. Note that the inference engine consists of 10 epistemic rules for compound formulas, which maybe used during RV of the monitored property. Hence, the message complexity is

$$O(\log(|\alpha|) + 10) \times |\alpha| \times n).$$

It is interesting to note that the number of messages propagated by each process at each global state of the monitored trace will not be increased when the number of processes increases since in our framework each process will send only one message at each state. However, the overall communication complexity of the algorithm will increase when the number of processes increases as the complexity depends directly on the number of processes.

**The Algorithm in terms of Memory Consumption and CPU Usage.** Our algorithm is efficient in terms of both CPU usage and memory consumption. We give here some intuitive theoretical explanation to show that. First, note that the tableau representation (graph) of the monitored formula will not be maintained in the memory during the actual RV process, we rather construct a truth table based on the tableau representation of the formula which contains only the information that is useful for the RV process of the formula. This information includes the set of atomic variables in the formula and the compound formulas that appear in the tableau graph that matches those in the epistemic rules R1-R10. Second, processes maintain the invariant values of propositions instead of propositions themselves. This is possible because all of the sent propositions are pre-known thanks to the tableau decomposition. The algorithm checks first whether there are some new observations that need to be propagated by process $p$ at certain state $\tau$ and if so it examines efficiently the table to determine the set of formulas that need to be propagated at that state. The implementation given at Section 8 shows the feasibility of the presented algorithm in terms of both CPU usage and memory consumption.
8 EXPERIMENTAL RESULTS

We have evaluated our static monitoring strategies against the LTL decentralized monitoring approach of Bauer and Falcone [5], in which the authors developed a monitoring algorithm for LTL based on the formula-progression technique [2]. The formula progression technique takes a temporal formula $\phi$ and a current assignment $I$ over the literals of $\phi$ as inputs and returns a new formula after acting $I$ on $\phi$. The idea is to rewrite a temporal formula when an event $e$ is observed or received to a formula which represents the new requirement that the monitored system should fulfill for the remaining part of the trace. To compare our monitoring approach with their decentralized algorithm, we use the tableau system of Section 2.2, which allows one to efficiently construct a semantic graph (tableau) for the input formula. We also use the tool DECENTMON3 (http://decentmon3.forge.imag.fr) in our evaluation, which is a tool dedicated to decentralized monitoring and it is mainly based on an implementation of the algorithm given in [7]. The tool takes as input multiple traces, corresponding to the behavior of a distributed system, and an LTL formula. The main reason behind choosing DECENTMON3 in our evaluation is that it makes similar assumptions to our presented approach. Furthermore, DECENTMON3 improves the original DECENTMON tool developed in [5] by limiting the growth of the size of local obligations and hence it may reduce the size of propagated messages in decentralized monitoring. We believe that by choosing the tool DECENTMON3 as baseline for comparison we make the evaluation much fairer.

We denote by BF the monitoring approach of Bauer and Falcone, DM1 the first RR strategy of this paper in which processes are ordered according to the number of propositions they contribute to in the formula, and DM2 the second RR strategy of this paper in which processes are ordered according to the number of branches they contribute to in the tableau of the monitored formula. We compare the approaches against two benchmarks of formulas: randomly generated formulas (see Table 1) and benchmark for patterns of formulas [1] (see Table 2). In Tables 1 and 2, the following metrics are used: $\#msg$, the total number of exchanged messages; $|msg|$, the total size of exchanged messages (in bytes); $|trace|$, the average length of the traces needed to reach a verdict; $|mem|$, the memory in bytes needed for the structures (i.e., formulas plus state for our algorithm); and CPU processing time measured in milliseconds. For example, the last line in Table 1 says that we monitored 1,000 randomly generated LTL formulas of size 6. On average, traces were of length 72.43 (i.e., average number of global states that have been monitored) when one of the local monitors in approach BF came to a verdict, and of length 72.9 and 72.8 when one of the monitors in DM1 and DM2 respectively came to a verdict.

8.1 Evaluation of randomly generated formulas

Following the evaluation scheme of Falcone et al. [5, 13], we evaluate the performance of each approach against a set of random LTL formulas of various sizes. For each size of formula (from 1 to 6), DECENTMON3 randomly generated 1,000 formulas. The result of comparing the three monitoring approaches can be seen in Table 1. The first column of these tables shows the size of the monitored LTL formulas. Note that we measure the formula size in terms of operator entailment inside it; for instance, $G(a \land b) \lor G(c \land d) \lor F(e)$ is of size 3. Experiments show that operator entailment is more representative of how difficult it is to progress it in a decentralized manner [5, 13]. As shown in Table 1 our decentralized monitoring approaches lead to significant reduction on the size of propagated messages, the memory consumption, and the CPU processing time compared to the BF approach (i.e. the formula progression technique). This demonstrates the effectiveness of the presented round-robin approaches and the inference LTL engine in decentralized monitoring.

8.2 Benchmarks for Patterns of formulas

We also compared the three approaches with more realistic specifications obtained from specification patterns [10]. Table 2 reports the verification results for different kinds of patterns (absence, existence, bounded existence, universal, precedence, response, precedence chain, response chain, constrained chain). The actual specification formulas are available at [1]. We generated also 1000 formulas monitored over the same setting (processes are synchronous and reliable). For this benchmark we generated formulas as follows. For each pattern, we randomly select one of its associated formulas. Such a formula is “parametrized” by some atomic propositions from the alphabet of the distributed system which are randomly instantiated. For this benchmark (see Table 2), the presented approaches lead also to significant reduction on both the size of messages and the amount of memory consumption compared to DECENTMON3.

8.3 Discussion Based on Evaluation Results

Comparing the decentralized monitoring algorithms, the number of messages when using BF is always lower but the size of messages and the memory consumption is much bigger and by several orders of magnitude than our approaches. However, the approach DM2 showed better performance (on most of the cases) than DM1 in terms of the number and size of propagated messages. This demonstrates the advantage of using tableau in synthesizing an RR communication policy for decentralized monitoring. A key drawback of using progression in decentralized monitoring (BF) is the continuous growth of the size of local obligations with the length of the trace, which imposes heavy overhead after a certain number of events. While progression minimizes communication in terms of number of messages, it has the risk of saturating the communication device as processes send their obligations as rewritten temporal formulas. On the other hand, processes in our static monitoring approaches (DM1 and DM2) propagate their observations as pairs of the form $(idx(\phi), val)$ (rather that rewritten LTL formulas), where observations may be propagated as truth values of temporal and compound formulas. Furthermore, in our approaches, processes propagate only new observations to their neighbour processes, which helps to reduce significantly the size of messages.

Our experimental results show that this approach based on tableau does not only “work on theory”, but that it is feasible to be implemented. Indeed, even the expected savings in message sizes, memory consumption, and CPU processing time could be observed for the set of chosen LTL formulae and the automatically generated traces, when compared to the BF approach in which the
local monitors transmit rewritten LTL formulas. The size of propagated messages needed in the decentralized tableau framework is only (15-17)% of the size of propagated messages induced by BF monitoring algorithm. The saving in memory consumption that can be achieved using our algorithm compared to their algorithm can be significant, where our algorithm consumes on average 0.045 of the total memory consumed by their algorithm. Our algorithm requires also much lower CPU processing time on average than the BF algorithm. Note that we use a static communication topology so that the receiving process of process \( p \)'s observations is fixed between states and hence no processing time will be consumed on computing the communication topology. This helps to reduce significantly the computation overhead. The processes need only to run Algorithm 1 at each state to compute the set of new observations that need to be propagated. In Bauer and Falcone algorithm the communication topology may vary between states depending on the resulting LTL obligations. Therefore it may take longer processing time at each global state. While their algorithm may reach a verdict faster than us as the topology is computed in a dynamic way, the size of propagated messages, the amount of memory consumption, and the required processing time can be significantly large which makes it infeasible for embedded systems.

### 9 RELATED WORK

Several algorithms have been developed for verifying distributed systems at run-time [5, 13, 17, 21, 23]. They make different assumptions on the system model and target different kinds of distributed systems and they handle different specification languages.

### Table 1: Benchmarks for 1000 randomly generated LTL formulas of size |\( \varphi \)| (Averages)

| | \#msg. | | msg. | | mem. | | CPU |
|---|---|---|---|---|---|---|
| | BF | DM1 | DM2 | BF | DM1 | DM2 | BF | DM1 | DM2 |
| 1 | 1.56 | 2.42 | 2.34 | 4.107 | 4.51 | 4.32 | 86.5 | 19.3 | 18.7 |
| 2 | 2.77 | 3.35 | 3.85 | 6.285 | 6.95 | 6.49 | 318 | 43.2 | 42.4 |
| 3 | 6.79 | 7.16 | 8.56 | 10.554 | 11.5 | 11.9 | 3,540 | 166 | 153 |
| 4 | 17.8 | 19.57 | 18.23 | 17.667 | 19.6 | 19.9 | 5,300 | 320 | 317 |
| 5 | 27.3 | 33.24 | 31.3 | 28.125 | 32.7 | 32.04 | 4,650 | 777 | 762 |
| 6 | 72.43 | 72.9 | 72.8 | 35.424 | 37.7 | 37.8 | 8,000 | 1,160 | 1,090 |

| | | | | | | | | | |
| | BF | DM1 | DM2 | BF | DM1 | DM2 | BF | DM1 | DM2 |
| | 5.86 | 6.68 | 6.29 | 5.92 | 5.95 | 5.82 | 2,758 | 135 | 132 |
| | 54.8 | 56.6 | 54.5 | 25.4 | 26.3 | 26.2 | 8,625 | 685 | 655 |
| | 622 | 645 | 622 | 425 | 525 | 515 | 22,000 | 1,125 | 1,111 |
| | 4.45 | 4.45 | 4.27 | 481 | 555 | 525 | 5,184 | 368 | 356 |
| | 427 | 442 | 440 | 381 | 395 | 392 | 9,000 | 2,875 | 2,799 |
| | 325 | 335 | 324 | 201 | 233 | 229 | 7,200 | 1,430 | 1,423 |

| | | | | | | | | | |
| | BF | DM1 | DM2 | BF | DM1 | DM2 | BF | DM1 | DM2 |
| | 6.79 | 7.16 | 8.56 | 10.554 | 11.5 | 11.9 | 3,540 | 166 | 153 |
| | 17.8 | 19.57 | 18.23 | 17.667 | 19.6 | 19.9 | 5,300 | 320 | 317 |
| | 27.3 | 33.24 | 31.3 | 28.125 | 32.7 | 32.04 | 4,650 | 777 | 762 |
| | 72.43 | 72.9 | 72.8 | 35.424 | 37.7 | 37.8 | 8,000 | 1,160 | 1,090 |

Table 2: Benchmarks for 1000 generated LTL pattern formulas (Averages)

| | | | | | | | | | |
| | BF | DM1 | DM2 | BF | DM1 | DM2 | BF | DM1 | DM2 |
| | 5.86 | 6.68 | 6.29 | 5.92 | 5.95 | 5.82 | 2,758 | 135 | 132 |
| | 54.8 | 56.6 | 54.5 | 25.4 | 26.3 | 26.2 | 8,625 | 685 | 655 |
| | 622 | 645 | 622 | 425 | 525 | 515 | 22,000 | 1,125 | 1,111 |
| | 4.45 | 4.45 | 4.27 | 481 | 555 | 525 | 5,184 | 368 | 356 |
| | 427 | 442 | 440 | 381 | 395 | 392 | 9,000 | 2,875 | 2,799 |
| | 325 | 335 | 324 | 201 | 233 | 229 | 7,200 | 1,430 | 1,423 |

| | | | | | | | | | |
| | BF | DM1 | DM2 | BF | DM1 | DM2 | BF | DM1 | DM2 |
| | 6.79 | 7.16 | 8.56 | 10.554 | 11.5 | 11.9 | 3,540 | 166 | 153 |
| | 17.8 | 19.57 | 18.23 | 17.667 | 19.6 | 19.9 | 5,300 | 320 | 317 |
| | 27.3 | 33.24 | 31.3 | 28.125 | 32.7 | 32.04 | 4,650 | 777 | 762 |
| | 72.43 | 72.9 | 72.8 | 35.424 | 37.7 | 37.8 | 8,000 | 1,160 | 1,090 |

Sen et al. [23] propose a monitoring framework for safety properties of distributed systems using the past-time linear temporal logic. However, the algorithm is unsound. The evaluation of some properties may be overlooked in their framework. This is because monitors gain knowledge about the state of the system by piggy-backing on the existing communication among processes. That is, if processes rarely communicate, then monitors exchange very little information, and hence, some violations of the properties may remain undetected.

Bauer and Falcone [5] propose a decentralized framework for run-time monitoring of LTL. The framework is constructed from local monitors which can only observe the truth value of a pre-defined subset of propositional variables. The local monitors can communicate their observations in the form of a (rewritten) LTL formula towards its neighbors. The approach has the risk of saturating the communication devices as processes send their obligations as rewritten temporal formulas. Mostafa and Bonakdarpour [17] propose a decentralized LTL monitoring framework under the asynchronous setting, where the LTL propositions are transmitted in the form of conjunctive predicates rather than monitor states or individual atomic propositions.

The work of Falcone et al. [13] proposes a general decentralized monitoring algorithm in which the input specification is given as a deterministic finite-state automaton rather than an LTL formula. Their algorithm takes advantage of the semantics of finite-word automata, and hence they avoid the monitorability issues induced by the infinite-words semantics of LTL. They show that their implementation outperforms the Bauer and Falcone decentralized LTL
Decentralized LTL Monitoring Framework Using Tableau

algorithm [5] using several monitoring metrics. While LTL formulas can be modeled as finite state automata (FSA) whose transitions are labeled with atomic propositions, we believe that presenting the input LTL formula as a tableau can yield greater advantages during decentralized monitoring, as it can help to detect efficiently tautological/unsatisfiable formulas and to identify infeasible parts in the input formula and hence monitoring can be performed in an effective way. The tableau technique should not be seen only as an automatic decomposition procedure that can be used to find the minimal way to decompose the formula by identifying feasible branches in the constructed tree, but also as a semantic graph that can help to synthesize an efficient communication strategy for processes based on the syntactic structure of the formula and the observation power of processes as demonstrated at Section 6.

Colombo and Falcone [7] propose a new way of organizing monitors called choreography, where monitors are organized as a tree across the distributed system, and each child feeds intermediate results to its parent. The proposed approach tries to minimize the communication induced by the distributed nature of the system and focuses on how to automatically split an LTL formula according to the architecture of the system.

El-Hokayem and Falcone [11] propose a new framework for decentralized monitoring with new data structure for symbolic representation and manipulation of monitoring information in decentralized monitoring. In their framework, the formula is modeled as an automaton where transitions of the monitored automaton are labeled with Boolean expressions over atomic propositions.

A closer work to our work is the one of Basin et al. [3], in which the authors use the 3-valued logic and an AND-OR graph to verify the system behavior at run-time with respect to specifications written in the real-time logic MTL. However, in our work we use a different construction to decompose and analyze the formula where we use the tableau graph, and we study the optimality problem in distributed monitoring with respect to the size of messages.

Recently Kazemlou and Bonakdarpour [14] propose a crash-resilient decentralized synchronous runtime verification framework for LTL formulas. While our framework and their framework are both designed under the synchronous setting, there are several key differences between the two frameworks. First, their framework is mainly based on the automata-based theories where the monitored LTL property is modeled as a finite state automaton. In our framework, we use the tableau technique to model and decompose the monitored property. Hence, in our framework, processes may not monitor the original property but a semantically equivalent property depending on the constructed tableau where all infeasible branches are removed. Second, the communication model used in their framework is a broadcast model where at each state each process broadcasts a message to all other processes carrying its observation power of processes as demonstrated at Section 6.

Recently Kazemlou and Bonakdarpour [14] propose a new framework called choreography, where monitors are organized as a tree across the distributed system, and each child feeds intermediate results to its parent. The proposed approach tries to minimize the communication induced by the distributed nature of the system and focuses on how to automatically split an LTL formula according to the architecture of the system.

El-Hokayem and Falcone [11] propose a new framework for decentralized monitoring with new data structure for symbolic representation and manipulation of monitoring information in decentralized monitoring. In their framework, the formula is modeled as an automaton where transitions of the monitored automaton are labeled with Boolean expressions over atomic propositions.

A closer work to our work is the one of Basin et al. [3], in which the authors use the 3-valued logic and an AND-OR graph to verify the system behavior at run-time with respect to specifications written in the real-time logic MTL. However, in our work we use a different construction to decompose and analyze the formula where we use the tableau graph, and we study the optimality problem in distributed monitoring with respect to the size of messages.

Recently Kazemlou and Bonakdarpour [14] propose a crash-resilient decentralized synchronous runtime verification framework for LTL formulas. While our framework and their framework are both designed under the synchronous setting, there are several key differences between the two frameworks. First, their framework is mainly based on the automata-based theories where the monitored LTL property is modeled as a finite state automaton. In our framework, we use the tableau technique to model and decompose the monitored property. Hence, in our framework, processes may not monitor the original property but a semantically equivalent property depending on the constructed tableau where all infeasible branches are removed. Second, the communication model used in their framework is a broadcast model where at each state each process broadcasts a message to all other processes carrying its observation power of processes as demonstrated at Section 6.

The authors also did not provide any experimental or evaluation results to demonstrate the efficiency of their algorithm.

10 CONCLUSION AND FUTURE WORK

We have presented a novel decentralized monitoring framework for LTL formulas using the tableau technique. The framework consists of two parts: (a) an algorithm that allows processes to compute at each state of the input execution trace the minimal set of formulas whose truth values need to be propagated, and (b) an LTL inference engine that allows processes to extract the maximal amount of information from the received messages. The propagation of monitoring information from one peer to the other in our framework follows a static communication strategy based on a round-robin scheduling policy. The advantage of using a static communication strategy in decentralized monitoring is that processes can compute precisely the knowledge state of each other and hence communicate only necessary observations. In future work, we aim to improve the static approaches by organizing processes into groups and then assign to each group a unique sub-formula of the main formula based on the semantics of the constructed tableau of the formula. This should help to reduce further the size of propagated messages.

REFERENCES

Appendices

A PROOF OF THEOREM 2

Let $P = \{p_0, ..., p_n\}$ be a distributed system and $\varphi$ be a property of $P$ that we seek to monitor in a decentralized manner with a set of propositional variables $PROP$. Suppose that $R$ is a round-robin policy synthesized over set of processes in $P$ that contribute to the truth value of $\varphi$. The soundness of the framework can be proved using Theorem 1 and the fact that observations of processes are propagated in a circular way using a round-robin policy. To prove the theorem it is sufficient to show that at least one process can construct the global trace of the system (knowing the truth values of all variables in $PROP$ at each step of the trace). As this would be sufficient to allow the process to evaluate the global property $\varphi$. Since processes use static communication scheme in which the order at which observations of processes are propagated is fixed between states, then the knowledge state of process $p_i$ will be influenced by knowledge states of all other processes and hence it will be able to construct the global trace of the system and evaluate the property. Let $|\pi_{i,j}|$ be the number of communication rounds needed for process $p_i$ to influence the information state of process $p_j$, where $|\pi_{i,j}|$ represents the length of the communication path between nodes $(p_i, p_j)$. Note that in our framework we assume that each process can send at most one message at any communication round. Therefore, when process $p_k$ makes a new observation $O_i$ at step $t$, then process $p_j$ knows about $O_i$ at time step $(t + |\pi_{i,j}|)$. So given a static communication scheme in which the length of communication path $|\pi_{i,j}|$ for any pair $(p_i, p_j)$ is prior knowledge, the knowledge state of a process $p_i$ at time step $t \geq 0$ can then be computed as follows:

$$K^t_{p_i} = (K^{0}_{p_i} \cup K^{t}_{p_i}) \cup \forall \gamma_0, ..., \gamma_n. i \in \gamma \Rightarrow (\bigcup_{t \geq 0} |\pi_{i,j}| > 0)$$

That is, at time step $t$, process $p_i$ knows the truth values of its local propositions from time 0 to time $t$ and the truth values of all propositions of process $p_j$ from time 0 to time $r$, given that $(t - r) \geq |\pi_{i,j}|$. We know from Theorem 1 that when process $p_{k}$ communicates with process $p_j$ it sends all new observations that are not known to $p_k$ (using Algorithm 1) and hence it is not possible for $p_j$ to miss any observation occurred in the current or past states. From this it is easy to see that a verdict reached by some process in the decentralized framework (if any) will be identical to the one reached by the centralized node in the centralized framework but with some bounded delay $d$, where $0 \leq d \leq |R|$, where $|R|$ represents the number of nodes in the round-robin communication policy $R$.

B PROOF OF THEOREM 3

Let $\alpha \in \Sigma^*$ be a finite trace of the system $P = \{p_1, ..., p_n\}$, $\varphi$ be a global property of $P$, and $Sub(\varphi) = \{\varphi_1, ..., \varphi_m\}$ be a decomposition procedure that decomposes an LTL formula into a set of subformulae using tableau decomposition rules. Let us assume also that processes of $P$ are communicating in a round-robin fashion using the connected graph $G$. Finally, let $T_{k}$ be the corresponding truth table at state $a_k$ of the input trace. Recall that in the truth table approach, processes maintain a truth table at each state $a_k$. Let $T_{k}$ be the truth table of $Sub(\varphi)$ at state $a_k$ maintained by a process $c$ that observes the entire global trace of the system and $T^c_{k}$ be the truth table of $Sub(\varphi)$ at state $a_k$ maintained by process $p_i$ that observes part of the state of the system. Note that when the truth table $T^c_{k}$ is fully constructed then process $p_i$ will be able to evaluate the property $\varphi$ at state $a_k$. So for a finite trace $\alpha$ of length $m$ the process $c$ maintains the truth tables $(T^c_0, T^c_1, ..., T^c_{m-1})$ and process $p_i$ maintains the tables $(T^c_0, T^c_i, ..., T^c_{m-1})$. To prove the theorem we need to show that the trace constructed by a central process $c$ in the centralized framework will be identical to the one constructed through communication by some process $p_i$ in the decentralized framework. As this will be sufficient to show that when the centralized framework reaches a verdict about the monitored property then the decentralized framework will reach the same verdict. Therefore, Theorem 3 can be proved by induction on the length of $\alpha$. We show that for any arbitrary state $a_k$, where $0 \leq k \leq m - 1$, then within a bounded time delay $\delta$, where $0 \leq \delta \leq |G|$, we have $T^c_k = T^c_{k+1}$. That is, process $p_i$ will be able to reconstruct the global trace. First, from the assumption that $G$ is a connected graph then we know that $a_{i,j} < \infty$ for all $i, j = 1, \ldots, n$. Second, from the assumption that processes are communicating in an RR fashion then every $\pi_{i,j}$ time units the information state of process $p_i$ will influence the information state of process $p_j$ (i.e., the nested information structure assumption). Hence, at time step $k + n$, where $n$ is the number of processes, some process $p_i$ will be able to fully construct the truth table $T^c_{k+1}$ which will be identical to $T^c_{k+1}$ and evaluate the formula at state $a_k$. Therefore, when process $c$ reaches a verdict w.r.t. $\varphi$ then some process $p_i$ in the decentralized framework will reach the same verdict but with some bounded delay $\delta$. 

O. Al-Bataineh, D. Rosenblum, and M. Reynolds