Decentralised LTL monitoring

Andreas Bauer[1] and Yliès Falcone[2] [*]

[1] NICTA [**] Software Systems Group and Australian National University
[2] Laboratoire d'Informatique de Grenoble, UJF Université Grenoble I, France

Abstract. Users wanting to monitor distributed or component-based systems often perceive them as monolithic systems which, seen from the outside, exhibit a uniform behaviour as opposed to many components displaying many local behaviours that together constitute the system's global behaviour. This level of abstraction is often reasonable, hiding implementation details from users who may want to specify the system's global behaviour in terms of an LTL formula. However, the problem that arises then is how such a specification can actually be monitored in a distributed system that has no central data collection point, where all the components' local behaviours are observable. In this case, the LTL specification needs to be decomposed into sub-formulae which, in turn, need to be distributed amongst the components' locally attached monitors, each of which sees only a distinct part of the global behaviour.

The main contribution of this paper is an algorithm for distributing and monitoring LTL formulae, such that satisfaction or violation of specifications can be detected by local monitors alone. We present an implementation and show that our algorithm introduces only a minimum delay in detecting satisfaction/violation of a specification. Moreover, our practical results show that the communication overhead introduced by the local monitors is considerably lower than the number of messages that would need to be sent to a central data collection point.

1 Introduction

Much work has been done on monitoring systems w.r.t. formal specifications such as linear-time temporal logic (LTL [1]) formulae. For this purpose, a system is thought of more or less as a "black box", and some (automatically generated) monitor observes its outside visible behaviour in order to determine whether or not the runtime behaviour satisfies an LTL formula. Applications include monitoring programs written in Java (cf. [2, 3]) or C (cf. [4]), monitoring of abstract Web services (cf. [5]), or transactions on typical e-commerce sites (cf. [6]).

From a system designer's point of view, who defines the overall behaviour that a system has to adhere to, this "black box" view is perfectly reasonable. For example, most modern cars have the ability to issue a warning if a passenger (including the driver) is not wearing a seat belt after the vehicle has reached a certain speed. One could imagine using a monitor to help issue this warning based on the following LTL formalisation, which captures this abstract requirement:

$$\varphi = \mathbf{G}\big(speed\_low \vee ((pressure\_sensor\_1\_high \Rightarrow seat\_belt\_1\_on) \\ \wedge \ldots \\ \wedge (pressure\_sensor\_n\_high \Rightarrow seat\_belt\_n\_on)))$$

The formula φ asserts that, at all times, when the car has reached a certain speed, and the pressure sensor in a seat $i \in [1, n]$ detects that a person is sitting in it ($pressure\_sensor\_i\_high$), it has to be the case that the corresponding seat belt is fastened ($seat\_belt\_i\_on$). Moreover, one can build a monitor for φ, which receives the respective sensor values and is able to assert whether or not these values constitute a violation—*but*, only if some central component exists in the car's network of components, which collects these sensor values and consecutively sends them to the monitor as input! In many real-world scenarios, such as the automotive one, this is an unrealistic assumption mainly for economic reasons, but also because the communication on a car's bus network has to be kept minimal. Therefore one cannot continuously send unnecessary sensor information on a bus that is shared by potentially critical applications where low latency is paramount (cf. [7]). In other words, in these scenarios, one has to monitor such a requirement not based

on a single behavioural trace, assumed to be collected by some global sensor, but based on the many *partial* behavioural traces of the components which make up the actual system. We refer to this as *decentralised LTL monitoring* when the requirement is given in terms of an LTL formula.

The main constraint that decentralised LTL monitoring needs to address is the lack of a global sensor and a central decision making point asserting whether the system's behaviour has violated or satisfied a specification. We already pointed out that, from a practical point of view, a central decision making point (i.e., global sensor) would require all the individual components to continuously send events over the network, and thereby negatively affecting the response time for other potentially critical applications on the network. Moreover from a theoretical point of view, a central observer (resp. global sensor) basically resembles the classical LTL monitoring problem, where the decentralised nature of the system under scrutiny does not play a role.

Arguably, there exist a number of real-world component-based applications, where the monitoring of an LTL formula can be realised via global sensors and/or central decision making points, e.g., when network latency and criticality do not play an important role. However, here we want to focus on those cases where there exists no global trace, no central decision making point, and where the goal is to keep the communication, required for monitoring the LTL formula, at a minimum.

In the decentralised setting, we assume that the system under scrutiny consists of a set of n components $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$, communicating on a synchronous bus, each of which has a local monitor attached to it. The set of all events is $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \ldots \cup \Sigma_n$, where Σ_i is the set of events visible to the monitor at component C_i. The global LTL formula, on the other hand, is specified over a set of propositions, AP, such that $\Sigma = 2^{AP}$. Moreover, we demand for all $i, j \leq n$ with $i \neq j$ that $\Sigma_i \cap \Sigma_j = \emptyset$ holds, i.e., events are local w.r.t. the components where they are monitored.

At a first glance, the synchronous bus may seem an overly stringent constraint imposed by our setting. However, it is by no means unrealistic, since in many real-world systems, especially critical ones, communication is synchronous. For example, the FlexRay bus protocol (cf. [8]) used for safety-critical systems in the automotive domain, allows synchronous communication. Similar systems are used in avionics, where synchronous implementations of control systems have, arguably, played an even greater role than in the automotive domain due to their deterministic notion of concurrency and the strong guarantees one can give concerning their correctness.

Brief overview of the approach. Let as before φ be an LTL formula formalising a requirement over the system's global behaviour. Then every local monitor, M_i, will at any time, t, monitor its own LTL formula, φ_i^t, w.r.t. a partial behavioural trace, u_i. Let us use $u_i(m)$ to denote the $(m+1)$-th event in a trace u_i, and $\mathbf{u} = (u_1, u_2, \ldots, u_n)$ for the *global trace*, obtained by pair-wise parallel composition of the partial traces, each of which at time t is of length $t+1$ (i.e., $\mathbf{u} = u_1(0) \cup u_2(0) \cup \ldots \cup u_n(0) \cdots u_1(t) \cup u_2(t) \cup \ldots \cup u_n(t)$). Note that from this point forward we will use $\mathbf{u}$ only when, in a given context, it is important to consider a global trace. However, when the particular type of trace (i.e., partial or global) is irrelevant, we will simply use u, u_i, etc. We also shall refer to partial traces as local traces due to their locality to a particular monitor in the system.

The decentralised monitoring algorithm evaluates the global trace $\mathbf{u}$ by considering the locally observed traces $u_i, i \in [1, n]$ in separation. In particular, it exhibits the following properties.

- If a local monitor yields $\varphi_i^t = \bot$ (resp. $\varphi_i^t = \top$) on some component C_i by observing u_i, it implies that $\mathbf{u}\Sigma^\omega \subseteq \Sigma^\omega \setminus \mathcal{L}(\varphi)$ (resp. $\mathbf{u}\Sigma^\omega \subseteq \mathcal{L}(\varphi)$) holds where $\mathcal{L}(\varphi)$ is the set of infinite sequences in Σ^ω described by φ. That is, a locally observed violation (resp. satisfaction) is, in fact, a global violation (resp. satisfaction). Or, in other words, $\mathbf{u}$ is a bad (resp. good) prefix for φ.
- If the monitored trace $\mathbf{u}$ is such that $\mathbf{u}\Sigma^\omega \subseteq \Sigma^\omega \setminus \mathcal{L}(\varphi)$ (resp. $\mathbf{u}\Sigma^\omega \subseteq \mathcal{L}(\varphi)$), one of the local monitors on some component C_i yields $\varphi_i^{t'} = \bot$ (resp. $\varphi_i^{t'} = \top$), $t' \geq t$, for an observation u_i', an extension of u_i, the local observation of $\mathbf{u}$ on C_i, because of some latency induced by decentralised monitoring, as we shall see.

However, in order to allow for the local detection of global violations (and satisfactions), monitors must be able to communicate, since their traces are only partial w.r.t. the global behaviour of the system. Therefore, our second important objective is to also monitor with *minimal communication overhead* (in comparison with a centralised solution where at any time, t, all n monitors send the observed events to a central decision making point).

Outline. Section 2 introduces basic notions and notation. LTL monitoring by means of formula rewriting (progression), a central concept to our paper, is discussed in Sec. 3. In Sec. 4, we lift this concept to the decentralised setting. The

Table 1: LTL semantics over infinite traces

$$w^i \models p \qquad \Leftrightarrow p \in w(i), \text{ for any } p \in AP$$
$$w^i \models \neg\varphi \qquad \Leftrightarrow w^i \not\models \varphi$$
$$w^i \models \varphi_1 \vee \varphi_2 \Leftrightarrow w^i \models \varphi_1 \vee w^i \models \varphi_2$$
$$w^i \models \mathbf{X}\varphi \qquad \Leftrightarrow w^{i+1} \models \varphi$$
$$w^i \models \varphi_1\mathbf{U}\varphi_2 \Leftrightarrow \exists k \in [i, \infty[.\ w^k \models \varphi_2 \wedge \forall l \in [i, k[.\ w^l \models \varphi_1$$

semantics induced by decentralised LTL monitoring is outlined in Sec. 5, whereas Sec. 6 details on how the local monitors operate in this setting and gives a concrete algorithm for this purpose. Experimental results, showing the feasibility of our approach, are presented in Sec. 7. Section 8 concludes and gives pointers to some related approaches. The proofs for all results claimed in this paper are in Appendix A.

2 Preliminaries

The considered architecture. Each component of the system emits events at discrete time instances. An event σ is a set of *actions* denoted by some atomic propositions from the set AP, i.e., $\sigma \in 2^{AP}$. We denote 2^{AP} by Σ and call it the *alphabet* (of system events).

As our system operates under the *perfect synchrony hypothesis* (cf. [9]), we assume that its components communicate with each other in terms of sending and receiving messages (which, for the purpose of easier presentation, can also be encoded by actions) at *discrete* instances of time, which are represented using identifier $t \in \mathbb{N}^{\geq 0}$. Under this hypothesis, it is assumed that neither computation nor communication take time. In other words, at each time t, a component may receive up to $n - 1$ messages and dispatch up to 1 message, which in the latter case will always be available at the respective recipient of the messages at time $t + 1$. Note that these assumptions extend to the components' monitors, which operate and communicate on the same synchronous bus. The hypothesis of perfect synchrony essentially abstracts away implementation details of how long it takes for components or monitors to generate, send, or receive messages. As indicated in the introduction, this is a common hypothesis for certain types of systems, which can be designed and configured (e.g., by choosing an appropriate duration between time t and $t + 1$) to not violate this hypothesis (cf. [9]).

We use a projection function Π_i to restrict atomic propositions or events to the local view of monitor M_i, which can only observe those of component C_i. For atomic propositions, $\Pi_i : 2^{AP} \to 2^{AP}$ and we note $AP_i = \Pi_i(AP)$ for $i \in [1, n]$. For events, $\Pi_i : 2^{\Sigma} \to 2^{\Sigma}$ and we note $\Sigma_i = \Pi_i(\Sigma)$, for $i \in [1, n]$. We also assume $\forall i, j \leq n.\ i \neq j \Rightarrow AP_i \cap AP_j = \emptyset$ and consequently $\forall i, j \leq n.\ i \neq j \Rightarrow \Sigma_i \cap \Sigma_j = \emptyset$. Seen over time, each component C_i produces a *trace* of events, also called its *behaviour*, which for t time steps is encoded as $u_i = u_i(0) \cdot u_i(1) \cdots u_i(t - 1)$ with $\forall t' < t.\ u_i(t') \in \Sigma_i$. Finite traces over an alphabet Σ are elements of the set Σ^* and are typically encoded by $u, u', \ldots$, whereas infinite traces over Σ are elements of the set Σ^ω and are typically encoded by $w, w', \ldots$ The set of all traces is given by the set $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. The set $\Sigma^* \setminus \{\epsilon\}$ is noted Σ^+. The finite or infinite sequence w^t is the *suffix* of the trace $w \in \Sigma^\infty$, starting at time t, i.e., $w^t = w(t) \cdot w(t + 1) \cdots$. The system's global behaviour, $\mathbf{u} = (u_1, u_2, \ldots, u_n)$ can now be described as a sequence of pair-wise union of the local events in component's traces, each of which at time t is of length $t + 1$ i.e., $\mathbf{u} = u(0) \cdots u(t)$.

Linear Temporal Logic (LTL). We monitor a system w.r.t. a global specification, expressed as an LTL [1] formula, that does not state anything about its distribution or the system's architecture. Formulae of LTL can be described using the following grammar: $\varphi ::= p \mid (\varphi) \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \varphi\mathbf{U}\varphi$, where $p \in AP$. Additionally, we allow the following operators, each of which is defined in terms of the above ones: $\top = p \vee \neg p$, $\bot = \neg\top$, $\varphi_1 \wedge \varphi_2 = \neg(\neg\varphi_1 \vee \neg\varphi_2)$, $\mathbf{F}\varphi = \top\mathbf{U}\varphi$, and $\mathbf{G}\varphi = \neg\mathbf{F}(\neg\varphi)$. The operators typeset in bold are the temporal operators. Formulae without temporal operators are called *state formulae*. We describe the set of all LTL formulae over AP by the set $\mathrm{LTL}(AP)$, or just LTL when the set of atomic propositions is clear from the context or does not matter. The semantics of LTL [1] is defined w.r.t. infinite traces:

Definition 1. *Let $w \in \Sigma^\omega$ and $i \in \mathbb{N}^{\geq 0}$. Satisfaction of an* LTL *formula by w at time i is inductively defined as given in Table 1.*

When $w^0 \models \varphi$ holds, we also write $w \models \varphi$ to denote the fact that w *is a model for* φ. As such, every formula $\varphi \in \mathrm{LTL}(AP)$ describes a set of infinite traces, called its *language*, and is denoted by $\mathcal{L}(\varphi) \subseteq \Sigma^\omega$. In this paper, a language describes desired or undesired system behaviours, formalised by an LTL formula.

3 Monitoring LTL formulae by progression

Central to our monitoring algorithm is the notion of *good and bad prefixes* for an LTL formula or, to be more precise, for the language it describes:

Definition 2. *Let $L \subseteq \Sigma^\omega$ be a language. The set of all* good prefixes *(resp.* bad prefixes*) of L is given by* $\mathrm{good}(L)$ *(resp.* $\mathrm{bad}(L)$*) and defined as follows:*

$$\mathrm{good}(L) = \{u \in \Sigma^* \mid u \cdot \Sigma^\omega \subseteq L\}, \qquad \mathrm{bad}(L) = \{u \in \Sigma^* \mid u \cdot \Sigma^\omega \subseteq \Sigma^\omega \setminus L\}.$$

To further ease presentation, we will shorten $\mathrm{good}(\mathcal{L}(\varphi))$ (resp. $\mathrm{bad}(\mathcal{L}(\varphi))$) to $\mathrm{good}(\varphi)$ (resp. $\mathrm{bad}(\varphi)$).

Although there exist a myriad of different approaches to monitoring LTL formulae, based on various finite-trace semantics (cf. [10]), one valid way of looking at the monitoring problem for some formula $\varphi \in \mathrm{LTL}$ is the following: The monitoring problem of $\varphi \in \mathrm{LTL}$ is to devise an efficient monitoring algorithm which, in a stepwise manner, receives events from a system under scrutiny and states whether or not the trace observed so far constitutes a good or a bad prefix of $\mathcal{L}(\varphi)$. One monitoring approach along those lines is described in [11]. We do not want to reiterate how in [11] a monitor is constructed for some LTL formula, but rather review an alternative monitoring procedure based on formula rewriting, which is also known as formula progression, or just *progression* in the domain of planning with temporally extended goals (cf. [12]).

Progression splits a formula into a formula expressing what needs to be satisfied by the current observation and a new formula (referred to as a *future goal* or *obligation*), which has to be satisfied by the trace in the future. As progression plays a crucial role in decentralised LTL monitoring, we recall its definition for the full set of LTL operators.

Definition 3. *Let $\varphi, \varphi_1, \varphi_2 \in \mathrm{LTL}$, and $\sigma \in \Sigma$ be an event. Then, the* progression function $P : LTL \times \Sigma \to LTL$ *is inductively defined as follows:*

$$
\begin{aligned}
P(p \in AP, \sigma) &= \top, \; \textit{if } p \in \sigma, \bot \textit{ otherwise} \\
P(\varphi_1 \vee \varphi_2, \sigma) &= P(\varphi_1, \sigma) \vee P(\varphi_2, \sigma) \\
P(\varphi_1 \mathbf{U} \varphi_2, \sigma) &= P(\varphi_2, \sigma) \vee P(\varphi_1, \sigma) \wedge \varphi_1 \mathbf{U} \varphi_2 \\
P(\mathbf{G}\varphi, \sigma) &= P(\varphi, \sigma) \wedge \mathbf{G}(\varphi) \\
P(\mathbf{F}\varphi, \sigma) &= P(\varphi, \sigma) \vee \mathbf{F}(\varphi)
\end{aligned}
\qquad
\begin{aligned}
P(\top, \sigma) &= \top \\
P(\bot, \sigma) &= \bot \\
P(\neg\varphi, \sigma) &= \neg P(\varphi, \sigma) \\
P(\mathbf{X}\varphi, \sigma) &= \varphi
\end{aligned}
$$

Note that monitoring using rewriting with similar rules as above has been described, for example, in [13, 14], although not necessarily with the same finite-trace semantics in mind that we are discussing in this paper. Informally, the progression function "mimics" the LTL semantics on an event σ, as it is stated by the following lemma.

Lemma 1. *Let φ be an* LTL *formula, σ an event and w an infinite trace, we have $\sigma \cdot w \models \varphi \Leftrightarrow w \models P(\varphi, \sigma)$.*

Lemma 2. *If $P(\varphi, \sigma) = \top$, then $\sigma \in \mathrm{good}(\varphi)$, whereas if $P(\varphi, \sigma) = \bot$, then $\sigma \in \mathrm{bad}(\varphi)$.*

Moreover, from Corollary 2 and Definition 2 it follows that if $P(\varphi, \sigma) \notin \{\top, \bot\}$, then there exist traces $w, w' \in \Sigma^\omega$, such that $\sigma \cdot w \models \varphi$ and $\sigma \cdot w' \not\models \varphi$ hold. Let us now get back to [11], which introduces a finite-trace semantics for LTL monitoring called LTL_3. It is captured by the following definition.

Definition 4. *Let $u \in \Sigma^*$, the satisfaction relation of LTL_3, $\models_3 \colon \Sigma^* \times \mathrm{LTL} \to \mathbb{B}_3$, with $\mathbb{B}_3 = \{\top, \bot, ?\}$, is defined as*

$$
u \models_3 \varphi = \begin{cases} \top & \textit{if } u \in \mathrm{good}(\varphi), \\ \bot & \textit{if } u \in \mathrm{bad}(\varphi), \\ ? & \textit{otherwise.} \end{cases}
$$

Based on this definition, it now becomes obvious how progression *could* serve as a monitoring algorithm for LTL_3.

Theorem 1. *Let $u = u(0) \cdots u(t) \in \Sigma^+$ be a trace, and $v \in \text{LTL}$ be the verdict, obtained by $t + 1$ consecutive applications of the progression function of φ on u, i.e., $v = P(\ldots (P(\varphi, u(0)), \ldots, u(t))))$. The following cases arise: If $v = \top$, then $u \models_3 \varphi = \top$ holds. If $v = \bot$, then $u \models_3 \varphi = \bot$ holds. Otherwise, $u \models_3 \varphi = ?$ holds.*

Note that in comparison with the monitoring procedure for LTL_3, described in [11], our algorithm, implied by this theorem, has the disadvantage that the formula, which is being progressed, may grow in size relative to the number of events. However, in practice, the addition of some practical simplification rules to the progression function usually prevents this problem from occurring.

4   Decentralised progression

Conceptually, a monitor, M_i, attached to component C_i, which observes events over $\Sigma_i \subseteq \Sigma$, is a rewriting engine that accepts as input an event $\sigma \in \Sigma_i$, and an LTL formula φ, and then applies LTL progression rules. Additionally at each time t, in our n-component architecture, a monitor can send a message and receive up to $n - 1$ messages in order to communicate with the other monitors in the system, using the same synchronous bus that the system's components communicate on. The purpose of these messages is to send future or even past obligations to other monitors, encoded as LTL formulae. In a nutshell, a formula is sent by some monitor M_i, whenever the most urgent outstanding obligation imposed by M_i's current formula at time t, φ_i^t, cannot be checked using events from Σ_i alone. Intuitively, the urgency of an obligation is defined by the occurrences (or lack of) certain temporal operators in it. For example, in order to satisfy $p \wedge \mathbf{X}q$, a trace needs to start with p, followed by a q. Hence, the obligation imposed by the subformula p can be thought of as "more urgent" than the one imposed by $\mathbf{X}q$. A more formal definition is given later in this section.

When progressing an LTL formula, e.g., in the domain of planning to rewrite a temporally extended LTL goal during plan search, the rewriting engine, which implements the progression rules, will progress a state formula $p \in AP$, with an event σ such that $p \notin \sigma$, to $\bot$, i.e., $P(p, \emptyset) = \bot$ (see Definition 3). However, doing this in the decentralised setting, could lead to wrong results. In other words, we need to make a distinction as to why $p \notin \sigma$ holds locally, and then to progress accordingly. Consequently, the progression rule for atomic propositions is simply adapted by parameterising it by a local set of atomic propositions AP_i:

$$P(p, \sigma, AP_i) = \begin{cases} \top & \text{if } p \in \sigma, \\ \bot & \text{if } p \notin \sigma \wedge p \in AP_i, \\ \overline{\mathbf{X}}p & \text{otherwise,} \end{cases} \tag{1}$$

where for every $w \in \Sigma^\omega$ and $j > 0$, we have $w^j \models \overline{\mathbf{X}}\varphi$ if and only if $w^{j-1} \models \varphi$. In other words, $\overline{\mathbf{X}}$ is the dual to the $\mathbf{X}$-operator, sometimes referred to as the "previously-operator" in past-time LTL (cf. [15]). To ease presentation, the formula $\overline{\mathbf{X}}^m \varphi$ is a short for $\overbrace{\overline{\mathbf{X}}\,\overline{\mathbf{X}} \ldots \overline{\mathbf{X}}}^{m} \varphi$. Our operator is somewhat different to the standard use of $\overline{\mathbf{X}}$: it can only precede an atomic proposition or an atomic proposition which is preceded by further $\overline{\mathbf{X}}$-operators. Hence, the restricted use of the $\overline{\mathbf{X}}$-operator does not give us the full flexibility (or succinctness gains [16]) of past-time LTL. Using the $\overline{\mathbf{X}}$-operator, let us now formally define the *urgency* of an LTL formula φ using a pattern matching on φ as follows:

Definition 5. *Let φ be an LTL formula, and $\Upsilon : \text{LTL} \to \mathbb{N}^{\geq 0}$ be an inductively defined function assigning a level of urgency to an LTL formula as follows.*

$$\begin{aligned} \Upsilon(\varphi) = \ &match\ \varphi\ with \\ &\varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \to \max(\Upsilon(\varphi_1), \Upsilon(\varphi_2)) \\ &\mid \overline{\mathbf{X}}\varphi' \qquad\qquad\quad \to 1 + \Upsilon(\varphi') \\ &\mid \_ \qquad\qquad\qquad\quad \to 0 \end{aligned}$$

A formula φ is said to be more urgent than formula ψ, if and only if $\Upsilon(\varphi) > \Upsilon(\psi)$ holds. A formula φ where $\Upsilon(\varphi) = 0$ holds is said to be not urgent.

Moreover, the above modification to the progression rules has obviously the desired effect: If $p \in \sigma$, then nothing changes, otherwise if $p \notin \sigma$, we return $\overline{\mathbf{X}}p$ in case that the monitor M_i cannot observe p at all, i.e., in case that $p \notin AP_i$ holds. This effectively means, that M_i cannot decide whether or not p occurred, and will therefore turn the state formula p into an obligation for some other monitor to evaluate rather than produce a truth-value. Of course, the downside of rewriting future goals into past goals that have to be processed further, is that violations or satisfactions of a global goal will usually be detected *after* they have occurred. However, since there is no central observer which records all events at the same time, the monitors *need* to communicate their respective results to other monitors, which, on a synchronous bus, occupies one or more time steps, depending on how often a result needs to be passed on until it reaches a monitor which is able to actually state a verdict. We shall later give an upper bound on these communication times, and show that our decentralised monitoring framework is, in fact, optimal under the given assumptions (see Theorem 2).

Example 1. Let us assume we have a decentralised system consisting of three components, A, B, C, such that $AP_A = \{a\}$, $AP_B = \{b\}$, and $AP_C = \{c\}$, and that a global formula $\varphi = \mathbf{F}(a \wedge b \wedge c)$ needs to be monitored in a decentralised manner. Let us further assume that, initially, $\varphi_A^0 = \varphi_B^0 = \varphi_C^0 = \varphi$. Let $\sigma = \{a, b\}$ be the system event at time 0; that is, M_A (resp. M_B, M_C) observes $\Pi_A(\sigma) = \{a\}$ (resp. $\Pi_B(\sigma) = \{b\}$, $\Pi_C(\sigma) = \emptyset$) when σ occurs. The rewriting that takes place in all three monitors to generate the next local goal formula, using the modified set of rules, and triggered by σ, is as follows:

$$
\begin{aligned}
\varphi_A^1 = P(\varphi, \{a\}, \{a\}) &= P(a, \{a\}, \{a\}) \wedge P(b, \{a\}, \{a\}) \wedge P(c, \{a\}, \{a\}) \vee \varphi \\
&= \overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi \\
\varphi_B^1 = P(\varphi, \{b\}, \{b\}) &= P(a, \{b\}, \{b\}) \wedge P(b, \{b\}, \{b\}) \wedge P(c, \{b\}, \{b\}) \vee \varphi \\
&= \overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}c \vee \varphi \\
\varphi_C^1 = P(\varphi, \emptyset, \{c\}) &= P(a, \emptyset, \{c\}) \wedge P(b, \emptyset, \{c\}) \wedge P(c, \emptyset, \{c\}) \vee \varphi \\
&= \overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}b \wedge \bot \vee \varphi = \varphi
\end{aligned}
$$

But we have yet to define progression for past goals: For this purpose, each monitor has local storage to keep a *bounded* number of past events. The event that occurred at time $t - k$ is referred as $\sigma(-k)$. On a monitor observing Σ_i, the progression of a past goal $\overline{\mathbf{X}}^m \varphi$, at time $t \geq m$, is defined as follows:

$$
P(\overline{\mathbf{X}}^m \varphi, \sigma, AP_i) = \begin{cases} \top & \text{if } \varphi = p \text{ for some } p \in AP_i \cap \Pi_i(\sigma(-m)), \\ \bot & \text{if } \varphi = p \text{ for some } p \in AP_i \setminus \Pi_i(\sigma(-m)), \\ \overline{\mathbf{X}}^{m+1} \varphi & \text{otherwise}, \end{cases} \tag{2}
$$

where, for $i \in [1, n]$, Π_i is the projection function associated to each monitor M_i, respectively. Note that since we do not allow $\overline{\mathbf{X}}$ for the specification of a global system monitoring property, our definitions will ensure that the local monitoring goals, φ_i^t, will never be of the form $\overline{\mathbf{X}}\mathbf{X}\mathbf{X}p$, which is equivalent to a future obligation, despite the initial $\overline{\mathbf{X}}$. In fact, our rules ensure that a formula preceded by the $\overline{\mathbf{X}}$-operator is either an atomic proposition, or an atomic proposition which is preceded by one or many $\overline{\mathbf{X}}$-operators. Hence, in rule (2), we do not need to consider any other cases for φ.

5   Semantics

In the previous example, we can clearly see that monitors M_A and M_B cannot determine whether or not σ, if interpreted as a trace of length 1, is a good prefix for the global goal formula φ.[3] Monitor M_C on the other hand did not observe an action c, and therefore, is the only monitor after time 0, which knows that σ is not a good prefix, and that, as before, after time 1, φ is the goal that needs to be satisfied by the system under scrutiny. Intuitively, the other two monitors know that if their respective past goals were satisfied, then σ would be a good prefix, but in order to determine this information, they need to send and receive messages to and from each other, containing obligations, i.e., LTL formulae.

---

[3] Note that $\mathcal{L}(\varphi)$, being a *liveness* language [17], does not have any bad prefixes.

Before we outline how this is done in our setting, let us discuss the semantics, we obtain from this decentralised application of progression. We already said that monitors detect good and bad prefixes for a global formula. In other words, if a monitor's progression evaluates to $\top$ (resp. $\bot$), then the trace seen so far is a good (resp. bad) prefix, and if neither monitor comes to a Boolean truth-value as verdict, we keep monitoring. This latter case indicates that, so far, the trace is neither a good nor a bad prefix for the global formula.

Definition 6. *Let* $\mathcal{C} = \{C_1, \ldots, C_n\}$ *be the set of system components,* $\varphi \in$ LTL *be a global goal, and* $\mathcal{M} = \{M_1, \ldots, M_n\}$ *be the set of component monitors. Further, let* $\mathbf{u} = u_1(0) \cup \ldots \cup u_n(0) \cdots u_1(t) \cup \ldots \cup u_n(t) \in \Sigma^*$ *be the global behavioural trace of the system, obtained by composition of all local component traces, at time* $t \in \mathbb{N}^{\geq 0}$. *If for some component* C_i, *with* $i \leq n$, *containing a local obligation* φ_i^t, M_i *reports* $P(\varphi_i^t, u_i(t), AP_i) = \top$ *(resp.* $\bot$*), then* $\mathbf{u} \models_D \varphi = \top$ *(resp.* $\bot$*). Otherwise, we have* $\mathbf{u} \models_D \varphi = ?$.

By $\models_D$ we denote the satisfaction relation on finite traces in the decentralised setting to differentiate it from LTL_3 as well as standard LTL which is defined on infinite traces. Obviously, $\models_3$ and $\models_D$ both yield values from the same truth-domain. However, the semantics are not equivalent, since the modified progression function used in the above definition sometimes rewrites a state formula into an obligation concerning the past rather than returning a verdict. On the other hand, in the case of a one-component system (i.e., all propositions of a formula can be observed by a single monitor), the definition of $\models_D$ matches Theorem 1, in particular because our progression rule (1) is then equivalent to the standard case. Monitoring LTL_3 with progression becomes a special case of decentralised monitoring, in the following sense:

Corollary 1. *If* $|\mathcal{M}| = 1$, *then* $\forall u \in \Sigma^*. \forall \varphi \in$ LTL. $u \models_3 \varphi = u \models_D \varphi$.

6 Communication and decision making

Let us now describe the communication mechanism that enables local monitors to determine whether a trace is a good or a bad prefix. Recall that each monitor only sees a projection of an event to its locally observable set of actions, encoded as a set of atomic propositions, respectively.

Generally, at time t, when receiving an event σ, a monitor, M_i, will progress its current obligation, φ_i^t, into $P(\varphi_i^t, \sigma, AP_i)$, and send the result to another monitor, $M_{j \neq i}$, whenever the most urgent obligation, $\psi \in \mathrm{sus}(P(\varphi_i^t, \sigma, AP_i))$, is such that $\mathrm{Prop}(\psi) \subseteq (AP_j)$ holds, where $\mathrm{sus}(\varphi)$ is the *set of urgent subformulae* of φ and $\mathrm{Prop} : \mathrm{LTL} \to 2^{AP}$ is the function which yields the set of occurring propositions of an LTL formula.

Definition 7. *The function* $\mathrm{sus} : \mathrm{LTL} \to 2^{\mathrm{LTL}}$ *is inductively defined as follows:*

$$\mathrm{sus}(\varphi) = \textit{match } \varphi \textit{ with} \quad \begin{array}{ll} \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \to \mathrm{sus}(\varphi_1) \cup \mathrm{sus}(\varphi_2) \\ \mid \neg\varphi' & \to \mathrm{sus}(\varphi') \\ \mid \overline{\mathbf{X}}\varphi' & \to \{\overline{\mathbf{X}}\varphi'\} \\ \mid \_ & \to \emptyset \end{array}$$

The set $\mathrm{sus}(\varphi)$ contains the past sub-formulae of φ, i.e., sub-formulae starting with a future temporal operator are discarded. It uses the fact that, in decentralised progression, $\overline{\mathbf{X}}$-operators are only introduced in front of atomic propositions. Thus, only the cases mentioned explicitly in the pattern matching need to be considered. Moreover, for formulae of the form $\overline{\mathbf{X}}\varphi'$, i.e., starting with an $\overline{\mathbf{X}}$-operator, it is not needed to apply sus to φ' because φ' is necessarily of the form $\overline{\mathbf{X}}^d p$ with $d \geq 0$ and $p \in AP$, and does not contain more urgent formulae than $\overline{\mathbf{X}}\varphi'$.

Note that, if there are several equally urgent obligations for distinct monitors, then M_i sends the formula to only one of the corresponding monitors according to a priority order between monitors. Using this order ensures that the delay induced by evaluating the global system specification in a decentralised fashion is bounded, as we shall see in Theorem 2. For simplicity, in the following, for a set of component monitors $\mathcal{M} = \{M_1, \ldots, M_n\}$ the sending order is the natural order on the interval $[1, n]$. This choice of the local monitor to send the obligation is encoded through the function $\mathrm{Mon} : \mathcal{M} \times 2^{AP} \to \mathcal{M}$. For a monitor $M_i \in \mathcal{M}$ and a set of atomic propositions $AP' \in 2^{AP}$, $\mathrm{Mon}(M_i, AP')$ is the monitor $M_{j_{\min}}$ s.t. $j_{\min}$ is the smallest integer in $[1, n]$ s.t. there is a monitor for an atomic proposition in AP'. Formally: $\mathrm{Mon}(M_i, AP') = j_{\min} = \min\{j \in [1, n] \setminus \{i\} \mid AP' \cap AP_j \neq \emptyset\}$.

Table 2: Decentralised progression of $\varphi = \mathbf{F}(a \wedge b \wedge c)$ in a 3-component system.

$t:$	0	1	2	3
$\sigma:$	$\{a,b\}$	$\{a,b,c\}$	$\emptyset$	$\emptyset$
$M_A:$	$\varphi_A^1 := P(\varphi, \sigma, AP_A)$ $= \overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi$	$\varphi_A^2 := P(\varphi_B^1 \wedge \#, \sigma, AP_A)$ $= \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi)$	$\varphi_A^3 := P(\varphi_C^2 \wedge \#, \sigma, AP_A)$ $= \overline{\mathbf{X}}^2 b \vee (\overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi)$	$\varphi_A^4 := P(\varphi_C^3 \wedge \#, \sigma, AP_A)$ $= \overline{\mathbf{X}}^3 b \vee (\overline{\mathbf{X}}b \wedge \overline{\mathbf{X}}c \vee \varphi)$
$M_B:$	$\varphi_B^1 := P(\varphi, \sigma, AP_B)$ $= \overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}c \vee \varphi$	$\varphi_B^2 := P(\varphi_A^1 \wedge \#, \sigma, AP_B)$ $= \overline{\mathbf{X}}^2 c \vee (\overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}c \vee \varphi)$	$\varphi_B^3 := P(\#, \sigma, AP_B)$ $= \#$	$\varphi_B^4 := P(\varphi_A^3 \wedge \#, \sigma, AP_B)$ $= \top$
$M_C:$	$\varphi_C^1 := P(\varphi, \sigma, AP_C)$ $= \varphi$	$\varphi_C^2 := P(\varphi, \sigma, AP_C)$ $= \overline{\mathbf{X}}a \wedge \overline{\mathbf{X}}b \vee \varphi$	$\varphi_C^3 := P(\varphi_A^2 \wedge \varphi_B^2 \wedge \#, \sigma, AP_C)$ $= \overline{\mathbf{X}}^2 a \wedge \overline{\mathbf{X}}^2 b \vee \varphi$	$\varphi_C^4 := P(\#, \sigma, AP_C)$ $= \#$

Once M_i has sent $P(\varphi_i^t, \sigma, AP_i)$, it sets $\varphi_i^{t+1} = \#$, where $\# \notin AP$ is a special symbol for which we define progression by

$$P(\#, \sigma, AP_i) = \#. \tag{3}$$

and $\forall \varphi \in \text{LTL}. \ \varphi \wedge \# = \varphi$. On the other hand, whenever M_i receives a formula, $\varphi_{j \neq i}$, sent from a monitor M_j, it will add the new formula to its existing obligation, i.e., its current obligation φ_i^t will be replaced by the conjunction $\varphi_i^t \wedge \varphi_{j \neq i}$. Should M_i receive further obligations from other monitors but j, it will add each new obligation as an additional conjunct in the same manner.

Let us now summarise the above steps in the form of an explicit algorithm that describes how the local monitors operate and make decisions.

Algorithm L (*Local Monitor*). Let φ be a global system specification, and $\mathcal{M} = \{M_1, \ldots, M_n\}$ be the set of component monitors. The algorithm Local Monitor, executed on each M_i, returns $\top$ (resp. $\bot$), if $\sigma \models_D \varphi_i^t$ (resp. $\sigma \not\models_D \varphi_i^t$) holds, where $\sigma \in \Sigma_i$ is the projection of an event to the observable set of actions of the respective monitor, and φ_i^t the monitor's current local obligation.

L1. [Next goal.] Let $t \in \mathbb{N}^{\geq 0}$ denote the current time step and φ_i^t be the monitor's current local obligation. If $t = 0$, then set $\varphi_i^t := \varphi$.

L2. [Receive event.] Read next σ.

L3. [Receive messages.] Let $\{\varphi_j\}_{j \in [1,n], j \neq i}$ be the set of received obligations at time t from other monitors. Set $\varphi_i^t := \varphi_i^t \wedge \bigwedge_{j \in [1,n], j \neq i} \varphi_j$.

L4. [Progress.] Determine $P(\varphi_i^t, \sigma, AP_i)$ and store the result in φ_i^{t+1}.

L5. [Evaluate and return.] If $\varphi_i^{t+1} = \top$ return $\top$, if $\varphi_i^{t+1} = \bot$ return $\bot$.

L6. [Communicate.] Set $\psi \in \text{sus}(\varphi_i^{t+1})$ to be the most urgent obligation of φ_i^{t+1}. Send φ_i^{t+1} to monitor $\text{Mon}(M_i, \text{Prop}(\psi))$.

L7. [Replace goal.] If in step L6 a message was sent at all, set $\varphi_i^{t+1} := \#$. Then go back to step L1. $\qquad \square$

The input to the algorithm, σ, will usually resemble the latest observation in a consecutively growing trace, $u_i = u_i(0) \cdots u_i(t)$, i.e., $\sigma = u_i(t)$. We then have that $\sigma \models_D \varphi_i^t$ (i.e., the algorithm returns $\top$) implies that $u \models_D \varphi$ holds (resp. for $\sigma \not\models_D \varphi_i^t$).

Example 2. To see how this algorithm works, let us continue the decentralised monitoring process initiated in Example 1. Table 2 shows how the situation evolves for all three monitors, when the global LTL specification in question is $\mathbf{F}(a \wedge b \wedge c)$ and the ordering between components is $A < B < C$. An evolution of M_A's local obligation, encoded as $P(\varphi_B^1 \wedge \#, \sigma, AP_A)$ (see cell M_A at $t = 1$) indicates that communication between the monitors has occurred: M_B sent its obligation to M_A, at the end of step 0. Likewise for the other obligations and monitors. The interesting situations are marked in grey: In particular at $t = 0$, M_C is the only monitor who knows for sure that, so far, no good nor bad prefix occurred (see grey cell at $t = 0$). At $t = 1$, we have the desired situation $\sigma = \{a, b, c\}$, but because none of the monitors can see the other monitors' events, it takes another two rounds of communication until both M_A and M_B detect that, indeed, the global obligation had been satisfied at $t = 1$ (see grey cell at $t = 3$).

This example highlights a worst case *delay* between the occurrence and the detection of a good (resp. bad) trace by a good (resp. bad) prefix, caused by the time it takes for the monitors to communicate obligations to each other. This delay directly depends on the number of monitors in the system, and is also the upper bound for the number of past events each monitor needs to store locally in order to be able to progress all occurring past obligations:

Theorem 2. *Let, for any $p \in AP$, $\overline{\mathbf{X}}^m p$ be a local obligation obtained by Algorithm L executed on some monitor $M_i \in \mathcal{M}$. At any time $t \in \mathbb{N}^{\geq 0}$, $m \leq \min(|\mathcal{M}|, t + 1)$.*

Proof. We provide below a sketch of the proof explaining the intuition on the theorem. The formal proof can be found in Appendix A.3.

Recall that $\overline{\mathbf{X}}$-operators are only introduced directly in front of atomic propositions according to rule (1) when M_i rewrites a propositional formula p with $p \notin AP_i$. Further $\overline{\mathbf{X}}$-operators can only be added according to rule (2) when M_i is unable to evaluate an obligation of the form $\overline{\mathbf{X}}^h p$. The interesting situation occurs when a monitor M_i maintains a set of urgent obligations of the form $\{\overline{\mathbf{X}}^h p_1, \ldots, \overline{\mathbf{X}}^j p_l\}$ with $h, j \in \mathbb{N}^{\geq 0}$, then, according to step L6 of Algorithm L, M_i will transmit the obligations to one monitor only thereby adding one additional $\overline{\mathbf{X}}$-operator to the remaining obligations: $\{\overline{\mathbf{X}}^{h+1} p_2, \ldots, \overline{\mathbf{X}}^{j+1} p_l\}$. Obviously, a single monitor cannot have more than $|\mathcal{M}| - 1$ outstanding obligations that need to be sent to the other monitors at any time t. So, the worst case delay is initiated during monitoring, if at some time *all* outstanding obligations of each monitor M_i, $i \in [1, |\mathcal{M}|]$, are of the form $\{\overline{\mathbf{X}} p_1, \ldots, \overline{\mathbf{X}} p_l\}$ with $p_1, \ldots, p_l \notin AP_i$ (i.e., the obligations are all equally urgent), in which case it takes $|\mathcal{M}| - 1$ time steps until the last one has been chosen to be sent to its respective monitor M_j. Using an ordering between components ensures here that each set of obligations will decrease in size after being transmitted once. Finally, a last monitor, M_j will receive an obligation of the form $\overline{\mathbf{X}}^{|\mathcal{M}|} p_k$ with $1 \leq k \leq l$ and $p_k \in AP_j$. $\qquad\square$

Consequently, the monitors only need to memorise a *bounded history* of the trace read so far, i.e., the last $|\mathcal{M}|$ events.

Example 2 also illustrates the relationship to the LTL$_3$ semantics discussed earlier in Sec. 3. This relationship is formalised by the two following theorems stating the "soundness and completeness" of the algorithm.

Theorem 3. *Let $\varphi \in$ LTL and $u \in \Sigma^*$, then $u \models_D \varphi = \top/\bot \Rightarrow u \models_3 \varphi = \top/\bot$, and $u \models_3 \varphi = ? \Rightarrow u \models_D \varphi = ?$.*

In particular, the example shows how the other direction of the theorem does not necessarily hold. Consider the trace $u = \{a, b\} \cdot \{a, b, c\}$: clearly, $u \models_3 \mathbf{F}(a \wedge b \wedge c) = \top$, but we have $u \models_D \mathbf{F}(a \wedge b \wedge c) = ?$ in our example. Again, this is a direct consequence of the delay introduced in our setting.

However, Algorithm L detects all verdicts for a specification as if the system was not distributed.

Theorem 4. *Let $\varphi \in$ LTL and $u \in \Sigma^*$, then $u \models_3 \varphi = \top/\bot \Rightarrow \exists u' \in \Sigma^*. |u'| \leq n \wedge u \cdot u' \models_D \varphi = \top/\bot$, where n is the number of components in the system.*

7   Experimental results

DECENTMON is an implementation, simulating the above distributed LTL monitoring algorithm in 1,800 LLOC, written in the functional programming language OCaml. It can be freely downloaded and run from [18]. The system takes as input multiple traces (that can be automatically generated), corresponding to the behaviour of a distributed system, and an LTL formula. Then the formula is monitored against the traces in two different modes: a) by merging the traces to a single, global trace and then using a "central monitor" for the formula (i.e., all local monitors send their respective events to the central monitor who makes the decisions regarding the trace), and b) by using the decentralised approach introduced in this paper (i.e., each trace is read by a separate monitor). We have evaluated the two different monitoring approaches (i.e., centralised vs. decentralised) using two different set-ups described in the remainder of this section.

Evaluation of randomly generated formulae. DECENTMON randomly generated 1,000 LTL formulae of various sizes in the architecture described in Example 1. How both monitoring approaches compared on these formulae can be seen in Table 3. The first columns show the size of the monitored LTL formulae and the underlying alphabet(s) of the

Table 3: Benchmarks for randomly generated LTL formulae

$|\varphi|$	Σ_c and Σ_d	centralised		decentralised		*diff. ratio*	
		$|$trace$|$	#msg.	$|$trace$|$	#msg.	$|$trace$|$	#msg.
1	$\{a,b,c\}$ $\{a|b|c\}$	1.369	4.107	1.634	0.982	1.1935	0.2391
2	$\{a,b,c\}$ $\{a|b|c\}$	2.095	6.285	2.461	1.647	1.1747	0.262
3	$\{a,b,c\}$ $\{a|b|c\}$	3.518	10.554	4.011	2.749	1.1401	0.2604
4	$\{a,b,c\}$ $\{a|b|c\}$	5.889	17.667	6.4	4.61	1.0867	0.2609
5	$\{a,b,c\}$ $\{a|b|c\}$	9.375	28.125	9.935	7.879	1.0597	0.2801
6	$\{a,b,c\}$ $\{a|b|c\}$	11.808	35.424	12.366	9.912	1.0472	0.2798

Table 4: Benchmarks for LTL specification patterns

pattern	Σ_c and Σ_d	centralised		decentralised		*diff. ratio*	
		$|$trace$|$	#msg.	$|$trace$|$	#msg.	$|$trace$|$	#msg.
absence	$\{a,b,c\}$ $\{a|b|c\}$	156.17	468.51	156.72	37.94	1.0035	0.0809
existence	$\{a,b,c\}$ $\{a|b|c\}$	189.90	569.72	190.42	44.41	1.0027	0.0779
bounded existence	$\{a,b,c\}$ $\{a|b|c\}$	171.72	515.16	172.30	68.72	1.0033	0.1334
universal	$\{a,b,c\}$ $\{a|b|c\}$	97.03	291.09	97.66	11.05	1.0065	0.0379
precedence	$\{a,b,c\}$ $\{a|b|c\}$	224.11	672.33	224.72	53.703	1.0027	0.0798
response	$\{a,b,c\}$ $\{a|b|c\}$	636.28	1,908.86	636.54	360.33	1.0004	0.1887
precedence chain	$\{a,b,c\}$ $\{a|b|c\}$	200.23	600.69	200.76	62.08	1.0026	0.1033
response chain	$\{a,b,c\}$ $\{a|b|c\}$	581.20	1,743.60	581.54	377.64	1.0005	0.2165
constrained chain	$\{a,b,c\}$ $\{a|b|c\}$	409.12	1,227.35	409.62	222.84	1.0012	0.1815

monitor(s). Note that our system measures formula size in terms of the operator entailment[4] inside it (state formulae excluded), e.g., $\mathbf{G}(a \wedge b) \vee \mathbf{F}c$ is of size 2. The entry $|$trace$|$ denotes the average length of the traces needed to reach a verdict. For example, the last line in Table 3 says that we monitored 1,000 randomly generated LTL formulae of size 6. On average, traces were of length 11.808 when the central monitor came to a verdict, and of length 12.366 when one of the local monitors came to a verdict. The difference ratio, given in the second last column then shows the average delay; that is, on average the traces were 1.0472 times longer in the decentralised setting than the traces in the centralised setting. The number of messages, #msg., in the centralised setting, corresponds to the number of events sent by the local monitors to the central monitor (i.e., $|trace| \times |\Sigma_d|$), and in the decentralised setting to the number of obligations transmitted between local monitors. What is striking here is that the amount of communication needed in the decentralised setting is ca. only 25% of the communication overhead induced by central monitoring, where local monitors need to send each event to a central monitor.

Evaluation using specification patterns. In order to evaluate our approach also at the hand of realistic LTL specifications, we conducted benchmarks using LTL formulae following the well-known LTL specification patterns ([19], whereas the actual formulae underlying the patterns are available at this site [20] and recalled in [18]). In this context, to randomly generate formulae, we proceeded as follows. For a given specification pattern, we randomly select one of the formulae associated to it. Such a formulae is "parametrised" by some atomic propositions. To obtain the randomly generated formula, using the distributed alphabet, we randomly instantiate the atomic propositions.

The results of this test are reported in Table 4: for each kind of pattern (absence, existence, bounded existence, universal, precedence, response, precedence chain, response chain, constrained chain), we generated again 1,000 formulae, monitored over the same architecture as used in Example 1.

---

[4] Our practical experiments show that this way of measuring the size of a formula is more representative of how difficult it is to progress it in a decentralised manner.

Summary. Both benchmarks certainly substantiate that the decentralised monitoring of an LTL formula induces a much lower communication overhead compared to a centralised solution. In fact, when considering the more realistic benchmark using the specification patterns, the communication overhead was significantly lower compared to monitoring randomly generated formulae. The same is true for the delay: in case of monitoring LTL formulae corresponding to specification patterns, the delay is almost negligible; that is, the local monitors detect violation/satisfaction of a monitored formula at almost the same time as a global monitor with access to all observations at any time. Note that we have further benchmarks available on [18] (omitted for space reasons), also to highlight the effect of differently sized alphabets and validate the maximal delay (Theorem 2). Note further that in our tests, we have used continuous simplification of the goal formulae in order to avoid a formula explosion problem caused by rewriting. In DECENTMON, advanced syntactic simplification rules[5] were introduced and sufficient for the purpose of our experiments.

8 Related work and conclusions

This work is by no means the first to introduce an approach to monitoring the behaviour of distributed systems. For example, [21] introduced MTTL, a temporal logic for describing properties of asynchronous systems, as well as a monitoring procedure that, given a partially ordered execution of a parallel asynchronous system, establishes whether or not there exist runs in the execution that violate a given MTTL correctness property. While at first this may seem to coincide with the work presented in this paper, there are noteworthy differences: First, many of the problems addressed in [21] stem from the fact that the systems to be monitored operate concurrently; that is, create a partially ordered set of behaviours. Our application domain are distributed but synchronous systems. Second, we take LTL "off-the-shelf"; that is, we do not add modalities to express properties concerning the distributed nature of the system under scrutiny. On the contrary, our motivation is to enable users to conceive a possibly distributed system as a single, monolithic system by enabling them to specify properties over the outside visible behaviour only—independent of implementation specific-details, such as the number of threads or components—and to automatically "distribute the monitoring" process for such properties for them. (Arguably, this also bears the advantage that users do not need to learn another formalism to express system properties.) Finally, we address the fact that in many distributed systems it is not possible to collect a global trace or insert a global decision making point, thereby forcing the automatically distributed monitors to communicate. But at the same time we try and keep communication at a minimum; that is, to not transmit the occurrence of every single observed event, because many such applications would not tolerate this kind of overhead. This aspect, on the other hand, does not play a role in [21] where the implementation was tried on parallel (Java) programs which are not executed on physically separated CPUs as in our case, and where one can collect a set of global behaviours to reason about.

Other recent works like [22] target physically distributed systems, but do not focus on the communication overhead that may be induced by their monitoring. Similarly, this work also mainly addresses the problem of monitoring systems which produce partially ordered traces (à la Diekert and Gastin), and introduces abstractions to deal with the combinational explosion of these traces.

To the best of our knowledge, our work is the first to address the problem of automatically distributing LTL monitors, and to introduce a decentralised monitoring approach that not only avoids a global point of observation or any form of central trace collection, but also tries to keep the number of communicated messages between monitors at a minimum. What is more, our experimental results show that this approach does not only "work on paper", but that it is feasible to be implemented. Indeed, even the expected savings in communication overhead could be observed for the set of chosen LTL formulae and the automatically generated traces, when compared to a centralised solution in which the local monitors transmit all observed events to a global monitor.

References

1. Amir Pnueli. The temporal logic of programs. In *Foundations of Computer Science (FOCS)*, pages 46–57. IEEE, 1977.
2. Eric Bodden. A lightweight LTL runtime verification tool for Java. In *Proc. 19th Conf. Object-Oriented Programming (OOP-SLA)*, pages 306–307. ACM, 2004.

---

[5] Compared to RuleR [14], the state-of-art rule-based runtime verification tool, for LTL specifications, our simplification function produced better results (see [18])

3. Patrick O'Neil Meredith and Grigore Rosu. Runtime verification with the RV System. In Barringer et al. [23], pages 136–152.

4. Justin Seyster, Ketan Dixit, Xiaowan Huang, Radu Grosu, Klaus Havelund, Scott A. Smolka, Scott D. Stoller, and Erez Zadok. Aspect-oriented instrumentation with GCC. In Barringer et al. [23], pages 405–420.

5. Sylvain Hallé and Roger Villemaire. Runtime verification for the web-a tutorial introduction to interface contracts in web applications. In Barringer et al. [23], pages 106–121.

6. Andreas Bauer, Rajeev Gore, and Alwen Tiu. A first-order policy language for history-based transaction monitoring. In *Proc. 6th Intl. Colloq. Theoretical Aspects of Computing (ICTAC)*, volume 5684 of *LNCS*, pages 96–111. Springer, 2009.

7. Manfred Broy. Challenges in automotive software engineering. In *Proc. 28th Intl. Conf. on Software Engineering (ICSE)*, pages 33–42. ACM, 2006.

8. Traian Pop, Paul Pop, Petru Eles, Zebo Peng, and Alexandru Andrei. Timing analysis of the FlexRay communication protocol. *Real-Time Syst.*, 39:205–235, 2008.

9. Axel Jantsch. *Modeling Embedded Systems and SoC's: Concurrency and Time in Models of Computation*. Morgan Kaufmann, 2003.

10. Andreas Bauer, Martin Leucker, and Christian Schallhart. Comparing LTL semantics for runtime verification. *Logic and Computation*, 20(3):651–674, 2010.

11. Andreas Bauer, Martin Leucker, and Christian Schallhart. Monitoring of real-time properties. In *Proc. 26th Conf. on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 4337 of *LNCS*. Springer, 2006.

12. Fahiem Bacchus and Froduald Kabanza. Planning for temporally extended goals. *Annals of Mathematics and Artificial Intelligence*, 22:5–27, 1998.

13. Grigore Roşu and Klaus Havelund. Rewriting-based techniques for runtime verification. *Automated Software Engineering*, 12(2):151–197, 2005.

14. Howard Barringer, David E. Rydeheard, and Klaus Havelund. Rule systems for run-time monitoring: from Eagle to RuleR. *J. Log. Comput.*, 20(3):675–706, 2010.

15. Orna Lichtenstein, Amir Pnueli, and Lenore D. Zuck. The glory of the past. In *Conf. on Logic of Programs*, pages 196–218. Springer, 1985.

16. Nicolas Markey. Temporal logic with past is exponentially more succinct, concurrency column. *Bulletin of the EATCS*, 79:122–128, 2003.

17. Bowen Alpern and Fred B. Schneider. Recognizing safety and liveness. *Distributed Computing*, 2(3):117–126, 1987.

18. DECENTMON Website. http://decentmonitor.forge.imag.fr.

19. Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Patterns in property specifications for finite-state verification. In *Intl. Conf. on Software Engineering (ICSE)*, pages 411–420. ACM, 1999.

20. Specification Patterns Website. http://patterns.projects.cis.ksu.edu/.

21. Koushik Sen, Abhay Vardhan, Gul Agha, and Grigore Rosu. Decentralized runtime analysis of multithreaded applications. In *20th Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2006.

22. Alexandre Genon, Thierry Massart, and Cédric Meuter. Monitoring distributed controllers. In *Formal Methods (FM)*, volume 4085 of *LNCS*, pages 557–572. Springer, 2006.

23. Howard Barringer, Yliès Falcone, Bernd Finkbeiner, Klaus Havelund, Insup Lee, Gordon J. Pace, Grigore Rosu, Oleg Sokolsky, and Nikolai Tillmann, editors. *Proc. Intl. Conf. on Runtime Verification (RV)*, volume 6418 of *LNCS*. Springer, 2010.

A Proofs

This section contains the proofs of the results stated in this paper.

A.1 Proofs for Section 3

Proof of Lemma 1. The following inductive proof follows the argument conveyed by Proposition 3 of [12]. For completeness sake, here we want to give the complete, formal, detailed proof.

The lemma is a direct consequence of the semantics of LTL (Definition 1) and the definition of progression (Definition 2). Recall that this lemma states that the progression function "mimics" the LTL semantics on some event σ.

Proof. We shall prove the following statement:

$$\forall \sigma \in \Sigma. \forall w \in \Sigma^\omega. \forall \varphi \in \text{LTL}. \ \sigma \cdot w \models \varphi \Leftrightarrow w \models P(\varphi, \sigma).$$

Let us consider an event $\sigma \in \Sigma$ and an infinite trace $w \in \Sigma^\omega$, the proof is done by a structural induction on $\varphi \in \text{LTL}$.

Base Case: $\varphi \in \{\top, \bot, p \in AP\}$**.**

- Case $\varphi = \top$. This case is trivial since, according to the definition of the progression function, $\forall \sigma \in \Sigma. \, P(\top, \sigma) = \top$. Moreover, according to the LTL semantics of $\top$, $\forall w \in \Sigma^\omega. \, w \models \top$.
- Case $\varphi = \bot$. This case is symmetrical to the previous one.
- Case $\varphi = p \in AP$. Recall that, according to the progression function for atomic propositions, we have $P(p, \sigma) = \top$ if $p \in \sigma$ and $\bot$ otherwise.
  - Let us suppose that $\sigma \cdot w \models p$. According to the LTL semantics of atomic propositions, it means that $p \in \sigma$, and thus $P(p, \sigma) = \top$. And, due to the LTL semantics of $\top$, we have $\forall w \in \Sigma^\omega. \, w \models \top$.
  - Let us suppose that $w \models P(p, \sigma)$. Since $P(p, \sigma) \in \{\top, \bot\}$, we have necessarily $P(p, \sigma) = \top$. According to the progression function, $P(p, \sigma) = \top$ amounts to $p \in \sigma$. Using the LTL semantics of atomic propositions, we deduce that $\sigma \cdot w \models p$.

Induction Case: $\varphi \in \{\neg \varphi', \varphi_1 \vee \varphi_2, \varphi_1 \wedge \varphi_2, \mathbf{G}\varphi', \mathbf{F}\varphi', \mathbf{X}\varphi', \varphi_1 \mathbf{U} \varphi_2\}$. Our induction hypothesis states that the lemma holds for some formulae $\varphi', \varphi_1, \varphi_2 \in$ LTL.

- Case $\varphi = \neg \varphi'$. On one hand, using the progression function for $\neg$, we have $P(\neg \varphi', \sigma) = \neg P(\varphi', \sigma)$. On the other hand, using the LTL semantics of operator $\neg$, we have $w \models \varphi \Leftrightarrow w \not\models \neg \varphi$. Thus, we have $\sigma \cdot w \models \neg \varphi'$ iff $\sigma \cdot w \not\models \varphi'$ iff (induction hypothesis on φ') $w \not\models P(\varphi', \sigma)$ iff $w \models \neg P(\varphi', \sigma)$ iff $w \models P(\neg \varphi', \sigma)$.
- Case $\varphi = \varphi_1 \vee \varphi_2$. Recall that, according to the progression function for operator $\vee$, we have $P(\varphi_1 \vee \varphi_2, \sigma) = P(\varphi_1, \sigma) \vee P(\varphi_2, \sigma)$.
  - Let us suppose that $\sigma \cdot w \models \varphi_1 \vee \varphi_2$. We distinguish again two sub-cases: $\varphi_1 \vee \varphi_2 = \top$ or $\varphi_1 \vee \varphi_2 \neq \top$. If $\varphi_1 \vee \varphi_2 = \top$, then this case reduces to the case where $\varphi = \top$, already treated. If $\varphi_1 \vee \varphi_2 \neq \top$, it means that either $\sigma \cdot w \models \varphi_1$ or $\sigma \cdot w \models \varphi_2$. Let us treat the case where $\sigma \cdot w \models \varphi_1$ (the other case is similar). From $\sigma \cdot w \models \varphi_1$, we can apply the induction hypothesis on φ_1 to obtain $w \models P(\varphi_1, \sigma)$, then, $w \models P(\varphi_1, \sigma) \vee P(\varphi_1, \sigma) = P(\varphi_1 \vee \varphi_2, \sigma)$.
  - Let us suppose that $w \models P(\varphi_1 \vee \varphi_2, \sigma) = P(\varphi_1, \sigma) \vee P(\varphi_2, \sigma)$. We distinguish again two sub-cases: $P(\varphi_1 \vee \varphi_2, \sigma) = \top$ or $P(\varphi_1 \vee \varphi_2, \sigma) \neq \top$.
    * If $P(\varphi_1 \vee \varphi_2, \sigma) = \top$, then we again distinguish two sub-cases:
      · If $P(\varphi_1, \sigma) = \top$ or $P(\varphi_2, \sigma) = \top$. Let us treat the case where $P(\varphi_1, \sigma) = \top$ (the other case is similar). Applying the induction hypothesis on φ_1, we have $\sigma \cdot w \models \varphi_1 \Leftrightarrow w \models P(\varphi_1, \sigma)$. Then, consider $w \in \Sigma^\omega$, we have $\sigma \cdot w \models \varphi_1$, and consequently $\sigma \cdot w \models \varphi_1 \vee \varphi_2$.
      · If $P(\varphi_1, \sigma) \neq \top$ and $P(\varphi_2, \sigma) \neq \top$, then we have $P(\varphi_1, \sigma) = \neg P(\varphi_2, \sigma)$. Applying the induction hypothesis on φ_1 and φ_2, we obtain $\sigma \cdot w \models \varphi_1 \Leftrightarrow \sigma \cdot w \not\models \varphi_2$. Let us consider $w \in \Sigma^\omega$. If $\sigma \cdot w \models \varphi_1$, then we have $\sigma \cdot w \models \varphi_1 \vee \varphi_2$. Else ($\sigma \cdot w \not\models \varphi_1$), we have $\sigma \models \varphi_2$, and then $\sigma \cdot w \models \varphi_1 \vee \varphi_2$.
    * If $P(\varphi_1 \vee \varphi_2, \sigma) \neq \top$, then we have either $w \models P(\varphi_1, \sigma)$ or $w \models P(\varphi_2, \sigma)$. Let us treat the case where $w \models P(\varphi_1, \sigma)$ (the other case is similar). From $w \models P(\varphi_1, \sigma)$, we can apply the induction hypothesis on φ_1 to obtain $\sigma \cdot w \models \varphi_1$, and thus $\sigma \cdot w \models \varphi_1 \vee \varphi_2$.
- Case $\varphi = \varphi_1 \wedge \varphi_2$. This case is similar to the previous one.
- Case $\varphi = \mathbf{G}\varphi'$. Recall that, according to the progression function for operator $\mathbf{G}$, $P(\mathbf{G}\varphi', \sigma) = P(\varphi', \sigma) \wedge \mathbf{G}\varphi'$.
  - Let us suppose that $\sigma \cdot w \models \mathbf{G}\varphi'$. According to the LTL semantics of operator $\mathbf{G}$, we have $\forall i \in \mathbb{N}^{\geq 0}. \, (\sigma \cdot w)^i \models \varphi'$. In particular, it implies that $(\sigma \cdot w)^0 \models \varphi'$, i.e., $\sigma \cdot w \models \varphi'$ and $\forall i \in \mathbb{N}^{\geq 0}. \, (\sigma \cdot w^1)^i \models \varphi'$, i.e., $(\sigma \cdot w)^1 = w \models \mathbf{G}\varphi'$. Using the induction hypothesis on φ', from $\sigma \cdot w \models \varphi'$, we obtain $w \models P(\varphi', \sigma)$. As expected, according to the LTL semantics of operator $\wedge$, we have $w \models P(\mathbf{G}\varphi', \sigma) \wedge \mathbf{G}\varphi' = P(\mathbf{G}\varphi', \sigma)$.
  - Let us suppose that $w \models P(\mathbf{G}\varphi', \sigma) = P(\varphi', \sigma) \wedge \mathbf{G}\varphi'$. It follows that $w \models P(\varphi', \sigma)$, and thus, using the induction hypothesis on φ', $\sigma \cdot w \models \varphi'$. Using the LTL semantics of operator $\mathbf{G}$, from $\sigma \cdot w \models \varphi'$ and $w \models \mathbf{G}\varphi'$, we deduce $\forall i \in \mathbb{N}^{\geq 0}. \, w^i \models \varphi'$, and then $\forall i \in \mathbb{N}. \, (\sigma \cdot w)^i \models \varphi'$, i.e., $\sigma \cdot w \models \mathbf{G}\varphi'$.
- Case $\varphi = \mathbf{F}\varphi'$. This case is similar to the previous one.
- Case $\varphi = \mathbf{X}\varphi'$. On one hand, using the progression function for $\mathbf{X}$, we have $P(\mathbf{X}\varphi', \sigma) = \varphi'$. On the other hand, using the LTL semantics of operator $\mathbf{X}$, we have $\sigma \cdot w \models \mathbf{X}\varphi'$ iff $w \models \varphi'$. Thus, we have $\sigma \cdot w \models \mathbf{X}\varphi'$ iff $w \models \varphi'$ iff (induction hypothesis on φ') $w \models P(\mathbf{X}\varphi', \sigma)$.
- Case $\varphi = \varphi_1 \mathbf{U} \varphi_2$. Recall that, according to the progression function for operator $\mathbf{U}$, $P(\varphi_1 \mathbf{U} \varphi_2, \sigma) = P(\varphi_2, \sigma) \vee (P(\varphi_1, \sigma) \wedge \varphi_1 \mathbf{U} \varphi_2)$.
  - Let us suppose that $\sigma \cdot w \models \varphi_1 \mathbf{U} \varphi_2$. According to the LTL semantics of operator $\mathbf{U}$, we have $\exists i \in \mathbb{N}^{\geq 0}. \, (\sigma \cdot w)^i \models \varphi_2 \wedge \forall 0 \leq l < i. \, (\sigma \cdot w)^l \models \varphi_1$. Let us distinguish two cases: $i = 0$ and $i > 0$.

13

* If $i = 0$, then we have $\sigma \cdot w \models \varphi_2$. Applying the induction hypothesis on φ_2, we have $w \models P(\varphi_2, \sigma)$, and consequently $w \models P(\varphi_1 \mathbf{U} \varphi_2, \sigma)$.
* Else ($i > 0$), we have $\forall 0 \leq l < i.\ (\sigma \cdot w)^l \models \varphi_1$. Consequently, we have $(\sigma \cdot w)^0 \models \varphi_1$, and thus $\sigma \cdot w \models \varphi_1$. Moreover, from $\forall 0 \leq l < i.\ (\sigma \cdot w)^l \models \varphi_1$, we deduce $\forall 0 \leq l < i - 1.\ w^l \models \varphi_1$. From $(\sigma \cdot w)^i \models \varphi_2$, we deduce $w^{i-1} \models \varphi_2$. From $w^{i-1} \models \varphi_2$ and $\forall 0 \leq l < i.\ (\sigma \cdot w)^l \models \varphi_1$, we deduce $w \models \varphi_1 \mathbf{U} \varphi_2$. Applying, the induction hypothesis on φ_1, from $\sigma \cdot w \models \varphi_1$, we obtain $w \models P(\varphi_1, \sigma)$. Finally, from $w \models \varphi_1 \mathbf{U} \varphi_2$ and $w \models P(\varphi_1, \sigma)$, we obtain $w \models P(\varphi_1 \mathbf{U} \varphi_2, \sigma)$.

- Let us suppose that $w \models P(\varphi_1 \mathbf{U} \varphi_2, \sigma)$.
  We distinguish two cases: $P(\varphi_1 \mathbf{U} \varphi_2, \sigma) = \top$ and $P(\varphi_1 \mathbf{U} \varphi_2, \sigma) \neq \top$.
  * If $P(\varphi_1 \mathbf{U} \varphi_2, \sigma) = P(\varphi_2, \sigma) \vee (P(\varphi_1, \sigma) \wedge \varphi_1 \mathbf{U} \varphi_2) = \top$. We distinguish again two sub-cases.
    · If $P(\varphi_2, \sigma) = \top$ or $P(\varphi_1, \sigma) \wedge \varphi_1 \mathbf{U} \varphi_2 = \top$. If $P(\varphi_2, \sigma) = \top$, then applying the induction hypothesis on φ_2, we have $\sigma \cdot w \models \varphi_2 \Leftrightarrow w \models \top$. Then, from $\sigma \cdot w \models \varphi_2$, we obtain, according to the LTL semantics of operator $\mathbf{U}$, $\sigma \cdot w \models \varphi_1 \mathbf{U} \varphi_2$. If $P(\varphi_1, \sigma) \wedge \varphi_1 \mathbf{U} \varphi_2 = \top$, we directly deduce that $\varphi_1 \mathbf{U} \varphi_2 = \top$, and then this case reduces to the case where $\varphi = \top$, already treated.
    · If $P(\varphi_2, \sigma) \neq \top$ and $P(\varphi_1, \sigma) \wedge \varphi_1 \mathbf{U} \varphi_2 \neq \top$, then we have $P(\varphi_2, \sigma) = \neg (P(\varphi_1, \sigma) \wedge \varphi_1 \mathbf{U} \varphi_2) = \neg P(\varphi_1, \sigma) \vee \neg (\varphi_1 \mathbf{U} \varphi_2)$. Applying the induction hypothesis on φ_1 and φ_2, we have $\sigma \cdot w \models \varphi_1 \Leftrightarrow w \models P(\varphi_1, \sigma)$, and $\sigma \cdot w \models \varphi_2 \Leftrightarrow w \models P(\varphi_2, \sigma)$, and thus $\sigma \cdot w \models \varphi_2 \Leftrightarrow (\sigma \cdot w \not\models \varphi_1 \vee w \not\models \varphi_1 \mathbf{U} \varphi_2)$. Let us now follow the LTL semantics of operator $\mathbf{U}$ and consider the two cases: $\sigma \cdot w \models \varphi_2$ or $\sigma \cdot w \not\models \varphi_2$. If $\sigma \cdot w \models \varphi_2$, thus $\sigma \cdot w \models \varphi_1 \mathbf{U} \varphi_2$ (according to the LTL semantics of $\mathbf{U}$). Else $(\sigma \cdot w \not\models \varphi_2)$, then $\sigma \cdot w \models \varphi_1$ and $w \models \varphi_1 \mathbf{U} \varphi_2$, and thus $\sigma \cdot w \models \varphi_1 \mathbf{U} \varphi_2$.
  * If $P(\varphi_1 \mathbf{U} \varphi_2, \sigma) \neq \top$, it means that either $w \models P(\varphi_2, \sigma)$ or $w \models P(\varphi_1, \sigma) \wedge \varphi_1 \mathbf{U} \varphi_2$.
    · If $w \models P(\varphi_2, \sigma)$, then applying the induction hypothesis on φ_2, we have $\sigma \cdot w \models \varphi_2$. Then, following the LTL semantics of operator $\mathbf{U}$, we obtain $\sigma \cdot w \models \varphi_1 \mathbf{U} \varphi_2$.
    · If $w \models P(\varphi_1, \sigma) \wedge \varphi_1 \mathbf{U} \varphi_2$, then we have $w \models P(\varphi_1, \sigma)$ and $w \models \varphi_1 \mathbf{U} \varphi_2$. Applying the induction hypothesis on φ_1, we have $\sigma \cdot w \models \varphi_1$. From $w \models \varphi_1 \mathbf{U} \varphi_2$, we have $\exists i \in \mathbb{N}^{\geq 0}.\ w^i \models \varphi_2 \wedge \forall 0 \leq l < i.\ w^l \models \varphi_1$. It implies that $(\sigma \cdot w)^{i+1} \models \varphi_2$ and $\forall 0 < l < i + 1.\ (\sigma \cdot w)^l \models \varphi_1$. Using, $\sigma \cdot w \models \varphi_1$, i.e., $(\sigma \cdot w)^0 \models \varphi_1$ and the LTL semantics of operator $\mathbf{U}$, we finally obtain $\sigma \cdot w \models \varphi_1 \mathbf{U} \varphi_2$. $\qquad \square$

Proof of Lemma 2. We shall prove the following statement.

$$\forall \varphi \in LTL. \forall \sigma \in \Sigma. \quad P(\varphi, \sigma) = \top \Rightarrow \sigma \in \mathrm{good}(\varphi)$$
$$\wedge\ P(\varphi, \sigma) = \bot \Rightarrow \sigma \in \mathrm{bad}(\varphi).$$

The proof uses the definition of the LTL semantics (Definition 1), the definition of good and bad prefixes (Definition 2), the progression function (Definition 3), and Lemma 1.

Proof. According to Lemma 1, we have $\forall \sigma \in \Sigma. \forall w \in \Sigma^\omega.\ \sigma \cdot w \models \varphi \Leftrightarrow w \models P(\varphi, \sigma)$. Consequently, we have $\forall \sigma \in \Sigma. \forall w \in \Sigma^\omega.\ \sigma \cdot w \models \varphi \Leftrightarrow \forall \sigma \in \Sigma. \forall w \in \Sigma^\omega.\ w \models P(\varphi, \sigma)$ and $\forall \sigma \in \Sigma. \forall w \in \Sigma^\omega.\ \sigma \cdot w \not\models \varphi \Leftrightarrow \forall \sigma \in \Sigma. \forall w \in \Sigma^\omega.\ w \not\models P(\varphi, \sigma)$. Consequently, when $P(\varphi, \sigma) = \top$, we have $\forall \sigma \in \Sigma. \forall w \in \Sigma^\omega.\ \sigma \cdot w \models \varphi$, i.e., $\sigma \in \mathrm{good}(\varphi)$. Similarly, when $P(\varphi, \sigma) = \bot$, we have $\forall \sigma \in \Sigma. \forall w \in \Sigma^\omega.\ \sigma \cdot w \not\models \varphi$, i.e., $\sigma \in \mathrm{bad}(\varphi)$.

The proof can also be obtained in a more detailed manner as shown below. Let us consider $\sigma \in \Sigma$ and $\varphi \in$ LTL. The proof is performed by a structural induction on φ.

Base Case: $\varphi \in \{\top, \bot, p \in AP\}$**.**

- Case $\varphi = \top$. In this case, the proof is trivial since $P(\top, \sigma) = \top$ and, according to the LTL semantics of $\top$ and the definition of good prefixes, $\mathrm{good}(\top) = \Sigma^*$.
- Case $\varphi = \bot$. Similarly, in this case, the proof is trivial since $P(\bot, \sigma) = \bot$ and $\mathrm{bad}(\bot) = \Sigma^*$.
- Case $\varphi = p \in AP$.
  Let us suppose that $P(\varphi, \sigma) = \top$. According to the progression function, it means that $p \in \sigma$. Moreover, since $\varphi = p$, according to the LTL semantics of atomic propositions, for any $w \in \Sigma^\omega$, we have $\sigma \cdot w \models \varphi$. According to the definition of good prefixes, it means that $\sigma \in \mathrm{good}(\varphi)$.
  The proof for $P(\varphi, \sigma) = \bot \Rightarrow \sigma \in \mathrm{bad}(\varphi)$ is similar.

Induction Case: $\varphi \in \{\neg\varphi', \varphi_1 \vee \varphi_2, \varphi_1 \wedge \varphi_2, \mathbf{G}\varphi', \mathbf{F}\varphi', \mathbf{X}\varphi', \varphi_1\mathbf{U}\varphi_2\}$**.** Our induction hypothesis states that the lemma holds for some formulae $\varphi', \varphi_1, \varphi_2 \in$ LTL.

- Case $\varphi = \neg\varphi'$. In this case, the result is obtained by using the induction hypothesis on φ' and the equality's $\bot = \neg\top$ and $\neg(\neg\varphi) = \varphi$.
- Case $\varphi = \varphi_1 \vee \varphi_2$. Recall that, according to the progression function for operator $\vee$, $P(\varphi_1 \vee \varphi_2, \sigma) = P(\varphi_1, \sigma) \vee P(\varphi_2, \sigma)$.
  Let us suppose that $P(\varphi, \sigma) = \top$. We distinguish two cases:
  - If $P(\varphi_1, \sigma) = \top$ or $P(\varphi_2, \sigma) = \top$. Let us treat the case where $P(\varphi_1, \sigma) = \top$. Using the induction hypothesis on φ_1, we have $\sigma \in \mathrm{good}(\varphi_1)$. According to the definition of good prefixes, we have $\forall w \in \Sigma^\omega. \sigma \cdot w \models \varphi_1$. We easily deduce, using the LTL semantics of operator $\vee$, that $\forall w \in \Sigma^\omega. \sigma \cdot w \models \varphi_1 \vee \varphi_2$, that is, $\sigma \in \mathrm{good}(\varphi_1 \vee \varphi_2)$.
  - If $P(\varphi_1, \sigma) \neq \top$ and $P(\varphi_2, \sigma) \neq \top$. Since $P(\varphi, \sigma) = \top$, we have $P(\varphi_1, \sigma) = \neg P(\varphi_2, \sigma)$. Using Lemma 1, we have $\forall w \in \Sigma^\omega. \sigma \cdot w \models \varphi_1 \Leftrightarrow w \models P(\varphi_1, \sigma)$ and $\forall w \in \Sigma^\omega. \sigma \cdot w \models \varphi_2 \Leftrightarrow w \models P(\varphi_2, \sigma)$. We deduce that $\forall w \in \Sigma^\omega. \sigma \cdot w \models \varphi_1 \Leftrightarrow \sigma \cdot w \not\models \varphi_2$. Let us consider $w \in \Sigma^\omega$. If $\sigma \cdot w \models \varphi_1$, we have $\sigma \cdot w \models \varphi_1 \vee \varphi_2$. Else ($\sigma \cdot w \not\models \varphi_1$), we have $\sigma \cdot w \models \varphi_2$, and then $\sigma \cdot w \models \varphi_2 \vee \varphi_1$. That is, $\forall w \in \Sigma^\omega. \sigma \cdot w \models \varphi_1 \vee \varphi_2$, i.e., $\sigma \in \mathrm{good}(\varphi_1 \vee \varphi_2)$.
  Let us suppose that $P(\varphi, \sigma) = \bot$. In this case, we have $P(\varphi_1, \sigma) = \bot$ and $P(\varphi_2, \sigma) = \bot$. Similarly, we can apply the induction hypothesis on φ_1 and φ_2 to find that σ is bad prefix of both φ_1 and φ_2, and is thus a bad prefix of $\varphi_1 \vee \varphi_2$ (using the LTL semantics of operator $\vee$).
- Case $\varphi = \varphi_1 \wedge \varphi_2$. This case is symmetrical to the previous one.
- Case $\varphi = \mathbf{G}\varphi'$. Recall that, according to the progression function for operator $\mathbf{G}$, $P(\mathbf{G}\varphi', \sigma) = P(\varphi', \sigma) \wedge \mathbf{G}\varphi'$.
  Let us suppose that $P(\varphi, \sigma) = \top$. It means that $P(\varphi', \sigma) = \top$ and $\mathbf{G}\varphi' = \top$. This case reduces to the case where $\varphi = \top$.
  Let us suppose that $P(\varphi, \sigma) = \bot$. We distinguish two cases.
  - If $P(\varphi', \sigma) = \bot$ or $\mathbf{G}\varphi' = \bot$. We distinguish again two sub-cases.
    * Sub-case $P(\varphi', \sigma) = \bot$. Using the induction hypothesis on φ', we deduce that $\sigma \in \mathrm{bad}(\varphi')$, i.e., $\forall w \in \Sigma^\omega. \sigma \cdot w \not\models \varphi'$. Following the LTL semantics of operator $\mathbf{G}$, we deduce that $\forall w \in \Sigma^\omega. \sigma \cdot w \not\models \mathbf{G}\varphi'$, i.e., $\sigma \in \mathrm{bad}(\mathbf{G}\varphi')$.
    * Sub-case $\mathbf{G}\varphi' = \bot$. This case reduces to the case where $\varphi = \bot$.
  - If $P(\varphi', \sigma) \neq \bot$ and $\mathbf{G}\varphi' \neq \bot$. From $P(\varphi', \sigma) \wedge \mathbf{G}\varphi' = \bot$, we deduce that $P(\varphi', \sigma) = \neg\mathbf{G}\varphi'$. Using Lemma 1 on φ', we have $\forall w \in \Sigma^\omega. \sigma \cdot w \models \varphi' \Leftrightarrow w \models P(\varphi', \sigma)$. Thus $\forall w \in \Sigma^\omega. \sigma \cdot w \models \varphi' \Leftrightarrow w \not\models \mathbf{G}\varphi'$. Let us consider $w \in \Sigma^\omega$. If $\sigma \cdot w \models \varphi'$, then we have $w \not\models \mathbf{G}\varphi'$. According to the LTL semantics of operator $\mathbf{G}$, it means that $\exists i \in \mathbb{N}^{\geq 0}. w^i \not\models \varphi'$. Thus, still following the LTL semantics of operator $\mathbf{G}$, $(\sigma \cdot w)^{i+1} \not\models \varphi'$, and, consequently $\sigma \cdot w \not\models \mathbf{G}\varphi'$. Else ($\sigma \cdot w \not\models \varphi'$), we have directly $\sigma \cdot w \not\models \mathbf{G}\varphi'$.
- Case $\varphi = \mathbf{F}\varphi'$. Recall that, according to the progression function for operator $\mathbf{F}$, $P(\mathbf{F}\varphi', \sigma) = P(\varphi', \sigma) \vee \mathbf{F}\varphi'$.
  Let us suppose that $P(\varphi, \sigma) = \top$. We distinguish two cases.
  - If $P(\varphi', \sigma) = \top$ or $\mathbf{F}\varphi' = \top$.
    * Sub-case $P(\varphi', \sigma) = \top$. Following the previous reasoning, using the induction hypothesis on φ', the LTL semantics of operator $\mathbf{F}$, and the definition of good prefixes, we obtain the expected result.
    * Sub-case $\mathbf{F}\varphi' = \top$. This case reduces to the case where $\varphi = \top$.
  - If $P(\varphi', \sigma) \neq \top$ and $\mathbf{F}\varphi' \neq \top$. From $P(\varphi', \sigma) \vee \mathbf{G}\varphi' = \bot$, we deduce that $P(\varphi', \sigma) = \neg\mathbf{F}\varphi'$. Using Lemma 1 on φ', we have $\forall w \in \Sigma^\omega. \sigma \cdot w \models \varphi' \Leftrightarrow w \models P(\varphi', \sigma)$. We thus have $\forall w \in \Sigma^\omega. \sigma \cdot w \models \varphi' \Leftrightarrow w \not\models \mathbf{F}\varphi'$. Let us consider $w \in \Sigma^\omega$. If $\sigma \cdot w \models \varphi'$, using the LTL semantics of operator $\mathbf{F}$, we have directly $\sigma \cdot w \models \mathbf{F}\varphi'$. Else ($\sigma \cdot w \not\models \varphi'$), we have $w \models \mathbf{F}\varphi'$. According to the LTL semantics of operator $\mathbf{F}$, it means that $\exists i \in \mathbb{N}^{\geq 0}. w^i \models \varphi'$, and thus $(\sigma \cdot w)^{i+1} \models \varphi'$. Consequently $\sigma \cdot w \models \mathbf{F}\varphi'$. That is, $\sigma \in \mathrm{good}(\mathbf{F}\varphi')$.
  Let us suppose that $P(\varphi, \sigma) = \bot$. It means that $P(\varphi', \sigma) = \bot$ and $\mathbf{F}\varphi' = \bot$. A similar reasoning as the one used for the case $\varphi = \mathbf{G}\varphi'$ and $P(\varphi, \sigma) = \top$ can be applied to obtain the expected result.
- Case $\varphi = \mathbf{X}\varphi'$. Recall that, according to the progression function for operator $\mathbf{X}$, $P(\mathbf{X}\varphi', \sigma) = \varphi'$.
  Let us suppose that $P(\varphi, \sigma) = \top$. It means that $\varphi' = \top$. According to the LTL semantics of $\top$, we have $\forall w \in \Sigma^\omega. w \models \varphi'$. Then, $\forall w \in \Sigma^\omega. \sigma \cdot w \models \mathbf{X}\varphi' = \varphi$. That is, $\sigma \in \mathrm{good}(\mathbf{X}\varphi')$.
  Let us suppose that $P(\varphi, \sigma) = \bot$. It means that $\varphi' = \bot$. According to the LTL semantics of $\bot$, we have $\forall w \in \Sigma^\omega. w \not\models \varphi'$. Then, $\forall w \in \Sigma^\omega. \sigma \cdot w \not\models \mathbf{X}\varphi' = \varphi$. That is, $\sigma \in \mathrm{bad}(\mathbf{X}\varphi')$.

– Case $\varphi = \varphi_1 \mathbf{U} \varphi_2$. Recall that, according to the progression function for operator $\mathbf{U}$, $P(\varphi_1 \mathbf{U} \varphi_2, \sigma) = P(\varphi_2, \sigma) \vee (P(\varphi_1, \sigma) \wedge \varphi_1 \mathbf{U} \varphi_2)$.
Let us suppose that $P(\varphi, \sigma) = \top$. We distinguish two cases.

- If $P(\varphi_2, \sigma) = \top$ or $P(\varphi_1, \sigma) \wedge \varphi_1 \mathbf{U} \varphi_2 = \top$.
  * Sub-case $P(\varphi_2, \sigma) = \top$. Using the induction hypothesis on φ_2, we have $\sigma \in \mathrm{good}(\varphi_2)$. Let us consider $w \in \Sigma^\omega$, we have $\sigma \cdot w \in \mathcal{L}(\varphi_2)$, i.e., $(\sigma \cdot w)^0 \models \varphi_1 \mathbf{U} \varphi_2$. According to the LTL semantics of $\mathbf{U}$, we have $\sigma \cdot w \models \varphi_1 \vee \varphi_2$, i.e., $\sigma \cdot w \in \mathcal{L}(\varphi_1 \mathbf{U} \varphi_2)$. We deduce that $\sigma \in \mathrm{good}(\varphi_1 \mathbf{U} \varphi_2)$.
  * Sub-case $P(\varphi_1, \sigma) \wedge \varphi_1 \mathbf{U} \varphi_2 = \top$. Necessarily, $\varphi_1 \mathbf{U} \varphi_2 = \top$ and this case reduces to the first one already treated.
- If $P(\varphi_2, \sigma) \neq \top$ and $P(\varphi_1, \sigma) \wedge \varphi_1 \mathbf{U} \varphi_2 \neq \top$. From $P(\varphi_1 \mathbf{U} \varphi_2, \sigma) = \top$, we deduce that $P(\varphi_2, \sigma) = \neg(P(\varphi_1, \sigma) \wedge \varphi_1 \mathbf{U} \varphi_2)$. Applying Lemma 1 to φ_2, we obtain $\forall w \in \Sigma^\omega$. $\sigma \cdot w \models \varphi_2 \Leftrightarrow w \models P(\varphi_2, \sigma)$. We thus have $\forall w \in \Sigma^\omega$. $\sigma \cdot w \models \varphi_2 \Leftrightarrow w \not\models P(\varphi_1, \sigma) \wedge \varphi_1 \mathbf{U} \varphi_2$. Let us consider $w \in \Sigma^\omega$. Let us distinguish two cases. If $\sigma \cdot w \models \varphi_2$, according to the LTL semantics of $\mathbf{U}$, we have $\sigma \cdot w \models \varphi_1 \mathbf{U} \varphi_2$. Else ($\sigma \cdot w \not\models \varphi_2$), it implies that $\sigma \cdot w \models P(\varphi_1, \sigma) \wedge \varphi_1 \mathbf{U} \varphi_2$, and, in particular $\sigma \cdot w \models \varphi_1 \mathbf{U} \varphi_2$. That is, in both cases, $\sigma \in \mathrm{good}(\varphi_1 \mathbf{U} \varphi_2)$.

Additional notation. For the remaining proofs, we define $\mathcal{P}$, the extended progression function on traces that consists in applying successively the progression function defined so far to each event in order.

Definition 8. *Given a formula $\varphi \in$ LTL and a trace $u = u(0) \cdots u(t-1) \in \Sigma^+$, the application of extended progression function $\mathcal{P}$ to φ and u is defined as:*

$$\mathcal{P}(\varphi, u(0) \cdots u(t-1)) = \mathcal{P}(\varphi, u) = P(\ldots (P(\varphi, u(0)), \ldots, u(t-1))))$$

For the sake of readability, in the remainder, we overload the notation of the progression function on events to traces, i.e., $\mathcal{P}(\varphi, u)$ is denoted $P(\varphi, u)$.

Some intermediate lemmas. Based on the previous introduced notation and the definition of the progression function (Definition 2), we extend the progression function to traces. The following lemma states some equality's that directly follow from an inductive application of the definition of the progression function on events.

Lemma 3. *Given some formulae $\varphi, \varphi_1, \varphi_2 \in$ LTL, and a trace $u \in \Sigma^+$, the progression function can be extended to the trace u by successively applying the previously defined progression function to each event of u in order. Moreover, we have: $\forall \varphi, \varphi_1, \varphi_2 \in$ LTL. $\forall u \in \Sigma^+$.*

$$
\begin{aligned}
P(\top, u) &= \top, \\
P(\bot, u) &= \bot, \\
P(p \in AP, u) &= \top \ \textit{if } p \in u(0), \bot \ \textit{otherwise}, \\
P(\neg \varphi, u) &= \neg P(\varphi, u), \\
P(\varphi_1 \vee \varphi_2, u) &= P(\varphi_1, u) \vee P(\varphi_2, u), \\
P(\varphi_1 \wedge \varphi_2, u) &= P(\varphi_1, u) \wedge P(\varphi_2, u), \\
P(\mathbf{G}\varphi, u) &= \textstyle\bigwedge_{i=0}^{|u|-1} P(\varphi, u^i) \wedge \mathbf{G}\varphi, \\
P(\mathbf{F}\varphi, u) &= \textstyle\bigvee_{i=0}^{|u|-1} P(\varphi, u^i) \vee \mathbf{F}\varphi, \\
P(\mathbf{X}\varphi, u) &= \begin{cases} \varphi & \textit{if } |u| = 1 \\ P(\varphi, u^1) & \textit{otherwise} \end{cases} \\
P(\varphi_1 \mathbf{U} \varphi_2, u) &= \begin{cases} P(\varphi_2, u) \vee P(\varphi_1, u) \wedge \varphi_1 \mathbf{U} \varphi_2 & \textit{if } |u| = 1 \\ \bigvee_{i=0}^{|u|-1} \left(P(\varphi_2, u^i) \wedge \bigwedge_{j=0}^{i-1} P(\varphi_1, u^j) \right) \vee \bigwedge_{i=0}^{|u|-1} P(\varphi_1, u^i) \wedge \varphi_1 \mathbf{U} \varphi_2 & \textit{otherwise} \end{cases}
\end{aligned}
$$

Proof. The proof is done by two inductions: an induction on the length of the trace u (which is also the number of times the progression function is applied) and a structural induction on $\varphi \in$ LTL.
Base Case: $u = \sigma \in \Sigma, |u| = 1$.
In this case, the result holds thanks to the definition of the progression function.
Induction case:

Let us suppose that the lemma holds for any trace $u \in \Sigma^+$ of some length $t \in \mathbb{N}$ and let us consider the trace $u \cdot \sigma \in \Sigma^+$, we perform a structural induction on $\varphi \in \text{LTL}$.

Structural Base case: $\varphi \in \{\top, \bot, p \in AP\}$.

- Case $\varphi = \top$. In this case the result is trivial since we have:

$$
\begin{aligned}
P(\top, u \cdot \sigma) &= P(P(\top, u), \sigma) \text{ (extended progression)} \\
&= P(\top, \sigma) \qquad \text{(induction hypothesis on } u) \\
&= \top \qquad\qquad \text{(progression on events)}
\end{aligned}
$$

- Case $\varphi = \bot$. This case is symmetrical to the previous one.
- Case $\varphi = p \in AP$. Let us distinguish two cases: $p \in u(0)$ or $p \notin u(0)$.
  - If $p \in u(0)$, we have:

$$
\begin{aligned}
P(p, u \cdot \sigma) &= P(P(p, u), \sigma) \text{ (extended progression)} \\
&= P(\top, \sigma) \qquad \text{(induction hypothesis on } u) \\
&= \top \qquad\qquad \text{(progression on events)}
\end{aligned}
$$

  - If $p \notin u(0)$, we have:

$$
\begin{aligned}
P(p, u \cdot \sigma) &= P(P(p, u), \sigma) \text{ (extended progression)} \\
&= P(\bot, \sigma) \qquad \text{(induction hypothesis on } u) \\
&= \bot \qquad\qquad \text{(progression on events)}
\end{aligned}
$$

Induction Case: $\varphi \in \{\neg\varphi', \varphi_1 \vee \varphi_2, \varphi_1 \wedge \varphi_2, \mathbf{G}\varphi', \mathbf{F}\varphi', \mathbf{X}\varphi', \varphi_1 \mathbf{U} \varphi_2\}$. Our induction hypothesis states that the lemma holds for some formulae $\varphi', \varphi_1, \varphi_2 \in \text{LTL}$.

- Case $\varphi = \neg\varphi'$. We have:

$$
\begin{aligned}
P(\neg\varphi', u \cdot \sigma) &= P(P(\neg\varphi', u), \sigma) \text{ (extended progression)} \\
&= P(\neg P(\varphi', u), \sigma) \text{ (induction hypothesis on } u \text{ and } \varphi') \\
&= \neg P(P(\varphi', u), \sigma) \text{ (progression on events)} \\
&= \neg P(\varphi', u \cdot \sigma) \qquad \text{(extended progression)}
\end{aligned}
$$

- Case $\varphi = \mathbf{X}\varphi'$. We have:

$$
\begin{aligned}
P(\mathbf{X}\varphi', u \cdot \sigma) &= P(P(\mathbf{X}\varphi', u), \sigma) \text{ (extended progression)} \\
&= P(P(\varphi', u^1), \sigma) \quad \text{(induction hypothesis on } u \text{ and } \varphi') \\
&= P(\varphi', u^1\sigma) \qquad\quad \text{(extended progression)} \\
&= P(\varphi', (u \cdot \sigma)^1)
\end{aligned}
$$

- Case $\varphi = \varphi_1 \vee \varphi_2$. We have:

$$
\begin{aligned}
P(\varphi_1 \vee \varphi_2, u \cdot \sigma) &= P(P(\varphi_1 \vee \varphi_2, u), \sigma) & \text{(extended progression)} \\
&= P(P(\varphi_1, u) \vee P(\varphi_2, u), \sigma) & \text{(induction hypothesis on } u \text{ and } \varphi_1, \varphi_2) \\
&= P(P(\varphi_1, u), \sigma) \vee P(P(\varphi_2, u), \sigma) & \text{(progression on events)} \\
&= P(\varphi_1, u \cdot \sigma) \vee P(\varphi_2, u \cdot \sigma) & \text{(extended progression)}
\end{aligned}
$$

- Case $\varphi = \varphi_1 \wedge \varphi_2$. This case is similar to the previous one.
- Case $\varphi = \mathbf{G}\varphi'$. We have:

$$
\begin{aligned}
&P(\mathbf{G}\varphi', u \cdot \sigma) \\
&= P(P(\mathbf{G}\varphi', u), \sigma) & \text{(extended progression)} \\
&= P(\textstyle\bigwedge_{i=0}^{|u|-1} P(\varphi', u^i) \wedge \mathbf{G}\varphi', \sigma) & \text{(induction hypothesis on } u \text{ and } \varphi') \\
&= P(\textstyle\bigwedge_{i=0}^{|u|-1} P(\varphi', u^i), \sigma) \wedge P(\mathbf{G}\varphi', \sigma) & \text{(progression on events for } \wedge) \\
&= \textstyle\bigwedge_{i=0}^{|u|-1} P(P(\varphi', u^i), \sigma) \wedge P(\mathbf{G}\varphi', \sigma) & \text{(extended progression for } \wedge) \\
&= \textstyle\bigwedge_{i=0}^{|u|-1} P(\varphi', u^i \cdot \sigma) \wedge P(\mathbf{G}\varphi', \sigma) & \text{(extended progression)} \\
&= \textstyle\bigwedge_{i=0}^{|u|-1} P(\varphi', u^i \cdot \sigma) \wedge P(\varphi', \sigma) \wedge \mathbf{G}\varphi' & \text{(progression on events for } \mathbf{G}) \\
&= \textstyle\bigwedge_{i=0}^{|u \cdot \sigma|-2} P(\varphi', (u \cdot \sigma)^i) \wedge P(\varphi', (u \cdot \sigma)^{|u \cdot \sigma|-1}) \wedge \mathbf{G}\varphi' & (u^i \cdot \sigma = (u \cdot \sigma)^i \text{ and } \sigma = (u \cdot \sigma)^{|u \cdot \sigma|-1}) \\
&= \textstyle\bigwedge_{i=0}^{|u \cdot \sigma|-1} P(\varphi', (u \cdot \sigma)^i) \wedge \mathbf{G}\varphi'
\end{aligned}
$$

17

– Case $\varphi = \mathbf{F}\varphi'$. We have:

$$P(\mathbf{F}\varphi', u \cdot \sigma)$$
$$= P(P(\mathbf{F}\varphi', u), \sigma) \qquad\qquad\qquad \text{(extended progression)}$$
$$= P(\bigvee_{i=0}^{|u|-1} P(\varphi', u^i) \vee \mathbf{F}\varphi', \sigma) \qquad\qquad \text{(induction hypothesis on } u \text{ and } \varphi')$$
$$= P(\bigvee_{i=0}^{|u|-1} P(\varphi', u^i), \sigma) \vee P(\mathbf{F}\varphi', \sigma) \qquad\qquad \text{(progression on events)}$$
$$= \bigvee_{i=0}^{|u|-1} P(\varphi', u^i \cdot \sigma) \vee P(\mathbf{F}\varphi', \sigma) \qquad\qquad \text{(extended progression for } \vee)$$
$$= \bigvee_{i=0}^{|u|-1} P(\varphi', u^i \cdot \sigma) \vee P(\varphi', \sigma) \vee \mathbf{F}\varphi' \qquad\qquad \text{(progression on events for } \mathbf{F})$$
$$= \bigvee_{i=0}^{|u\cdot\sigma|-2} P(\varphi', (u \cdot \sigma)^i) \vee P(\varphi', (u \cdot \sigma)^{|u\cdot\sigma|-1}) \vee \mathbf{F}\varphi' \;\; (u^i \cdot \sigma = (u \cdot \sigma)^i \text{ and } \sigma = (u \cdot \sigma)^{|u\cdot\sigma|-1})$$
$$= \bigvee_{i=0}^{|u\cdot\sigma|-1} P(\varphi', (u \cdot \sigma)^i) \vee \mathbf{F}\varphi'$$

– Case $\varphi = \varphi_1 \mathbf{U} \varphi_2$. We have:

$P(\varphi_1 \mathbf{U} \varphi_2, u \cdot \sigma)$
(extended progression)
$$= P(P(\varphi_1 \mathbf{U} \varphi_2, u), \sigma)$$
(induction hypothesis on u, and structural induction hypothesis on φ_1 and φ_2)
$$= P\Big(\bigvee_{i=0}^{|u|-1} \big(P(\varphi_2, u^i) \wedge \bigwedge_{j=0}^{i-1} P(\varphi_1, u^j) \big) \vee \bigwedge_{i=0}^{|u|-1} P(\varphi_1, u^i) \wedge \varphi_1 \mathbf{U} \varphi_2, \sigma \Big)$$
(progression on events for $\vee$)
$$= P\Big(\bigvee_{i=0}^{|u|-1} \big(P(\varphi_2, u^i) \wedge \bigwedge_{j=0}^{i-1} P(\varphi_1, u^j) \big), \sigma \Big) \vee P\big(\bigwedge_{i=0}^{|u|-1} P(\varphi_1, u^i) \wedge \varphi_1 \mathbf{U} \varphi_2, \sigma \big)$$
(progression on events for $\wedge$ and $\vee$)
$$= \bigvee_{i=0}^{|u|-1} \big(P(P(\varphi_2, u^i), \sigma) \wedge \bigwedge_{j=0}^{i-1} P(P(\varphi_1, u^j), \sigma) \big) \vee \bigwedge_{i=0}^{|u|-1} P(P(\varphi_1, u^i), \sigma) \wedge P(\varphi_1 \mathbf{U} \varphi_2, \sigma)$$
(extended progression)
$$= \bigvee_{i=0}^{|u|-1} \big(P(\varphi_2, u^i \cdot \sigma) \wedge \bigwedge_{j=0}^{i-1} P(\varphi_1, u^j \cdot \sigma) \big) \vee \bigwedge_{i=0}^{|u|-1} P(\varphi_1, u^i \cdot \sigma) \wedge P(\varphi_1 \mathbf{U} \varphi_2, \sigma)$$

Moreover:

$$\bigwedge_{i=0}^{|u|-1} P(\varphi_1, u^i \cdot \sigma) \wedge P(\varphi_1 \mathbf{U} \varphi_2, \sigma)$$
(progression on events for $\mathbf{U}$)
$$= \bigwedge_{i=0}^{|u|-1} P(\varphi_1, u^i \cdot \sigma) \wedge \big(P(\varphi_2, \sigma) \vee P(\varphi_1, \sigma) \wedge \varphi_1 \mathbf{U} \varphi_2 \big)$$
(distribution of $\wedge$ over $\vee$)
$$= \big(\bigwedge_{i=0}^{|u|-1} P(\varphi_1, u^i \cdot \sigma) \wedge P(\varphi_2, \sigma) \big) \vee \big(\bigwedge_{i=0}^{|u|-1} P(\varphi_1, u^i \cdot \sigma) \wedge P(\varphi_1, \sigma) \wedge \varphi_1 \mathbf{U} \varphi_2 \big)$$
($\sigma = (u \cdot \sigma)^{|u\cdot\sigma|-1}$ and elimination of $P(\varphi_1, \sigma)$)
$$= \big(\bigwedge_{i=0}^{|u|-1} P(\varphi_1, u^i \cdot \sigma) \wedge P(\varphi_2, \sigma) \big) \vee \big(\bigwedge_{i=0}^{|u\cdot\sigma|-1} P(\varphi_1, u^i \cdot \sigma) \wedge \varphi_1 \mathbf{U} \varphi_2 \big)$$

Furthermore:

$$\bigvee_{i=0}^{|u|-1} \big(P(\varphi_2, u^i \cdot \sigma) \wedge \bigwedge_{j=0}^{i-1} P(\varphi_1, u^j \cdot \sigma) \big) \vee \big(\bigwedge_{i=0}^{|u|-1} P(\varphi_1, u^i \cdot \sigma) \wedge P(\varphi_2, \sigma) \big)$$
(variable renaming)
$$= \bigvee_{i=0}^{|u|-1} \big(P(\varphi_2, u^i \cdot \sigma) \wedge \bigwedge_{j=0}^{i-1} P(\varphi_1, u^j \cdot \sigma) \big) \vee \big(P(\varphi_2, \sigma) \wedge \bigwedge_{j=0}^{|u|-1} P(\varphi_1, u^j \cdot \sigma) \big)$$
($\sigma = (u \cdot \sigma)^{|u\cdot\sigma|-1}$)
$$= \bigvee_{i=0}^{|u\cdot\sigma|-2} \big(P(\varphi_2, (u \cdot \sigma)^i) \wedge \bigwedge_{j=0}^{i-1} P(\varphi_1, u^j \cdot \sigma) \big) \vee \big(P(\varphi_2, (u \cdot \sigma)^{|u\cdot\sigma|-1}) \wedge \bigwedge_{j=0}^{|u\cdot\sigma|-2} P(\varphi_1, u^j \cdot \sigma) \big)$$
$$= \bigvee_{i=0}^{|u\cdot\sigma|-1} \big(P(\varphi_2, (u \cdot \sigma)^i) \wedge \bigwedge_{j=0}^{i-1} P(\varphi_1, (u \cdot \sigma)^j) \big)$$

Finally:

$$P(\varphi_1 \mathbf{U} \varphi_2, u \cdot \sigma)$$
$$= \bigvee_{i=0}^{|u|-1} \big(P(\varphi_2, u^i \cdot \sigma) \wedge \bigwedge_{j=0}^{i-1} P(\varphi_1, u^j \cdot \sigma) \big) \vee \big(\bigwedge_{i=0}^{|u|-1} P(\varphi_1, u^i \cdot \sigma) \wedge P(\varphi_2, \sigma) \big)$$
$$\vee \big(\bigwedge_{i=0}^{|u\cdot\sigma|-1} P(\varphi_1, u^i \cdot \sigma) \wedge \varphi_1 \mathbf{U} \varphi_2 \big)$$
$$= \bigvee_{i=0}^{|u\cdot\sigma|-1} \big(P(\varphi_2, u^i \cdot \sigma) \wedge \bigwedge_{j=0}^{i-1} P(\varphi_1, u^j \cdot \sigma) \big) \vee \big(\bigwedge_{i=0}^{|u\cdot\sigma|-1} P(\varphi_1, u^i \cdot \sigma) \wedge \varphi_1 \mathbf{U} \varphi_2 \big)$$
$$= \bigvee_{i=0}^{|u\cdot\sigma|-1} \big(P(\varphi_2, (u \cdot \sigma)^i) \wedge \bigwedge_{j=0}^{i-1} P(\varphi_1, (u \cdot \sigma)^j) \big)$$
$$\vee \big(\bigwedge_{i=0}^{|u\cdot\sigma|-1} P(\varphi_1, (u \cdot \sigma)^i) \wedge \varphi_1 \mathbf{U} \varphi_2 \big)$$

$\square$

We introduce another intermediate lemma, which is a consequence of the definition of the LTL semantics (Definition 1) and the definition of the progression function (Definition 8). This lemma will be useful in the remaining proofs. This lemma states that the progression function "mimics" the semantics of LTL on a trace $u \in \Sigma^+$.

Lemma 4. *Let φ be an* LTL *formula, $u \in \Sigma^+$ a non-empty trace and $w \in \Sigma^\omega$ an infinite trace, we have $u \cdot w \models \varphi \Leftrightarrow w \models P(\varphi, u)$.*

Proof. We shall prove the following statement:

$$\forall u \in \Sigma^+. \forall w \in \Sigma^\omega. \forall \varphi \in \text{LTL. } u \cdot w \models \varphi \Leftrightarrow w \models P(\varphi, u).$$

Let us consider $u \in \Sigma^+$, the proof is done by a structural induction on $\varphi \in \text{LTL}$.
Base case: $\varphi \in \{\top, \bot, p \in AP\}$.

- Case $\varphi = \top$. This case is trivial since, using Lemma 3 on $\top$ and u, we have $P(\top, u) = \top$. Moreover, according to the LTL semantics of $\top$, $\forall w \in \Sigma^\omega. u \cdot w \models \top$.
- Case $\varphi = \bot$. This case is symmetrical to the previous one.
- Case $\varphi = p \in AP$.
  - Let us suppose that $u \cdot w \models p$. By applying Lemma 3 on $\top$ and u, we have $P(u, p) = \top$. Moreover, due to the LTL semantics of $\top$, we have $\forall w \in \Sigma^\omega. w \models \top = P(u, p)$.
  - Let us suppose that $w \models P(p, u)$. Since $P(p, u) \in \{\top, \bot\}$, we have necessarily $P(p, u) = \top$. According to the progression function, $P(p, u) = \top$ necessitates that $p \in u(0)$. Using the LTL semantics of atomic propositions, we deduce that $(u \cdot w)^0 \models p$, i.e., $u \cdot w \models p$.

Induction Case: $\varphi \in \{\neg\varphi', \varphi_1 \vee \varphi_2, \varphi_1 \wedge \varphi_2, \mathbf{G}\varphi', \mathbf{F}\varphi', \mathbf{X}\varphi', \varphi_1 \mathbf{U} \varphi_2\}$. Our induction hypothesis states that the lemma holds for some formulae $\varphi', \varphi_1, \varphi_2 \in \text{LTL}$.

- Case $\varphi = \varphi_1 \vee \varphi_2$. Recall that, by applying Lemma 3 on $\varphi_1 \vee \varphi_2$ and u, we have $P(\varphi_1 \vee \varphi_2, u) = P(\varphi_1, u) \vee P(\varphi_2, u)$.
  - Let us suppose that $u \cdot w \models \varphi_1 \vee \varphi_2$. Let us distinguish two cases: $\varphi_1 \vee \varphi_2 = \top$ and $\varphi_1 \vee \varphi_2 \neq \top$. If $\varphi_1 \vee \varphi_2 = \top$, then this case reduces to the case where $\varphi = \top$ already treated. If $\varphi_1 \vee \varphi_2 \neq \top$, it means that either $u \cdot w \models \varphi_1$ or $u \cdot w \models \varphi_2$. Let us treat the case where $u \cdot w \models \varphi_1$ (the other case is similar). From $u \cdot w \models \varphi_1$, we can apply the structural induction hypothesis on φ_1 to obtain $w \models P(\varphi_1, u)$, and then, $w \models P(\varphi_1, u) \vee P(\varphi_2, u) = P(\varphi_1 \vee \varphi_2, u)$.
  - Let us suppose that $w \models P(\varphi_1 \vee \varphi_2, u)$. Let us again distinguish two cases. If $P(\varphi_1, u) \vee P(\varphi_2, u) = \top$, then it reduces to the case where $\varphi = \top$ already treated. If $P(\varphi_1, u) \vee P(\varphi_2, u) \neq \top$, then we have either $w \models P(\varphi_1, u)$ or $w \models P(\varphi_2, u)$. Let us treat the case where $w \models P(\varphi_1, u)$ (the other case is similar). From $w \models P(\varphi_1, u)$, we can apply the structural induction hypothesis on φ_1 to obtain $u \cdot w \models \varphi_1$, and thus, using the LTL semantics of $\vee$, $u \cdot w \models \varphi_1 \vee \varphi_2$.
- Case $\varphi = \varphi_1 \wedge \varphi_2$. This case is similar to the previous one.
- Case $\varphi = \mathbf{G}\varphi'$. Recall that, by applying Lemma 3 on $\mathbf{G}\varphi'$ and u, we have $P(\mathbf{G}\varphi', u) = \bigwedge_{i=0}^{|u|-1} P(\varphi', u^i) \wedge \mathbf{G}\varphi'$.
  - Let us suppose that $u \cdot w \models \mathbf{G}\varphi'$. From the LTL semantics of operator $\mathbf{G}$, we have $\forall i \in \mathbb{N}^{\geq 0}. (u \cdot w)^i \models \varphi'$. In particular, it implies that $\forall 0 \leq i \leq |u| - 1. u^i \cdot w \models \varphi'$ and $\forall i \geq 0. ((u \cdot w)^{|u|-1})^i \models \varphi'$. Using, $\forall 0 \leq i \leq |u| - 1. u^i \cdot w \models \varphi'$ and applying the structural induction hypothesis on φ' and the u_i's, we obtain $\forall 0 \leq i \leq |u| - 1. w \models P(\varphi', u^i)$, and thus $w \models \bigwedge_{i=0}^{|u|-1} P(\varphi', u^i)$. Using $\forall i \geq 0. w^i = ((u \cdot w)^{|u|-1})^i \models \varphi'$, we obtain $w \models \mathbf{G}\varphi'$. As expected, according to the LTL semantics of $\wedge$, we have $w \models \bigwedge_{i=0}^{|u|-1} P(\varphi', u^i) \wedge \mathbf{G}\varphi' = P(\mathbf{G}\varphi', u)$.
  - Let us suppose that $w \models P(\mathbf{G}\varphi', u)$. We have $\forall 0 \leq i \leq |u| - 1. w \models P(\varphi', u^i)$ and $w \models \mathbf{G}\varphi'$. Using the structural induction hypothesis on φ' and the u^i's, it follows that $\forall 0 \leq i \leq |u| - 1. u^i \cdot w = (u \cdot w)^i \models \varphi'$. Using the semantics of operator $\mathbf{G}$, from $w \models \mathbf{G}\varphi'$ and $\forall 0 \leq i \leq |u| - 1. u^i \cdot w = (u \cdot w)^i \models \varphi'$, we deduce $u \cdot w \models \mathbf{G}\varphi'$.
- Case $\varphi = \mathbf{F}\varphi'$. This case is similar to the previous one.

- Case $\varphi = \mathbf{X}\varphi'$. Recall that, by applying Lemma 3 on u and $\mathbf{X}\varphi'$, we have $P(\mathbf{X}\varphi', u) = P(\varphi', u^1 \cdot \sigma)$. Using the LTL semantics of $\mathbf{X}$, we have $u \cdot w \models \mathbf{X}\varphi'$ iff $u^1 \cdot w \models \varphi'$. Thus we have $u \cdot w \models \mathbf{X}\varphi'$ iff $u^1 \cdot \sigma \cdot w \models \varphi'$ iff (induction hypothesis on φ') $w \models P(\varphi', u^1 \cdot \sigma) = P(\mathbf{X}\varphi', u)$.
- Case $\varphi = \neg\varphi'$. Recall that, by applying Lemma 3 on u and $\neg\varphi'$, we have $P(\neg\varphi', u) = \neg P(\varphi', u)$. Using the LTL semantics of operator $\neg$, we have $\forall \varphi \in \mathrm{LTL}.\forall w \in \Sigma^\omega.\ w \models \varphi \Leftrightarrow w \not\models \neg\varphi$. Thus, we have $u \cdot w \models \neg\varphi'$ iff $u \cdot w \not\models \varphi'$ iff (induction hypothesis on φ') $w \not\models P(\varphi', u)$ iff $w \models \neg P(\varphi', u)$ iff $w \models P(\neg\varphi', u)$.
- Case $\varphi = \varphi_1 \mathbf{U} \varphi_2$. Recall that, by applying Lemma 3 on u and $\varphi_1 \mathbf{U} \varphi_2'$, we have

$$P(\varphi_1 \mathbf{U} \varphi_2', u) = \bigvee_{i=0}^{|u|-1} \left(P(\varphi_2, u^i) \wedge \bigwedge_{j=0}^{i-1} P(\varphi_1, u^j) \right) \vee \bigwedge_{i=0}^{|u|-1} P(\varphi_1, u^i) \wedge \varphi_1 \mathbf{U} \varphi_2.$$

- Let us suppose that $u \cdot w \models \varphi_1 \mathbf{U} \varphi_2$. According to the LTL semantics of operator $\mathbf{U}$, $\exists k \in \mathbb{N}^{\geq 0}.\ (u \cdot w)^k \models \varphi_2 \wedge \forall 0 \leq l < k.\ (u \cdot w)^l \models \varphi_1$. Let us distinguish two cases: $k > |u|$ and $k \leq |u|$.
  * If $k > |u|$, then we have in particular $\forall 0 \leq l \leq |u| - 1.\ u^l \cdot w \models \varphi_1$. Applying the structural induction hypothesis on φ_1 and the u^l's, we find $\forall 0 \leq l \leq |u|.\ w \models P(\varphi_1, u^l)$, i.e., $w \models \bigwedge_{l=0}^{|u|-1} P(\varphi_1, u^l)$. From $(\sigma \cdot w)^k \models \varphi_2$ and $k > |u| - 1$, we deduce that $\exists k' \geq 0.\ w^{k'} \models \varphi_2$ and $k' = k - |u| + 1$. Furthermore, we have $\forall 0 \leq i \leq k'.\ ((u \cdot w)^{|u|-1})^{k'} = w \models P(\varphi_1, u)$, i.e., $w \models \bigwedge_{i=0}^{k'} P(\varphi_1, u^i)$. Finally, $w \models P(\varphi_1 \mathbf{U} \varphi_2, u)$.
  * If $k \leq |u| - 1$, then from $(u \cdot w)^k \models \varphi_2$, we have $u^k \cdot w \models \varphi_2$. Using the induction hypothesis on φ_2 and u^k, we have $w \models P(\varphi_2, u^k)$. Moreover, using $\forall l \leq |k|.\ (u \cdot w)^l = u^l \cdot w \models \varphi_1$ and the induction hypothesis on φ_1 and the u^l's, we obtain $\forall l \leq |k|.\ (u \cdot w)^l = w \models P(\varphi_1, u^l)$. Finally, we have $w \models \bigwedge_{l=0}^{k} \models P(\varphi_1, u^l) \wedge P(\varphi_2, u^k)$, and thus $w \models P(\varphi_1 \mathbf{U} \varphi_2, u)$.
- Let us suppose that $w \models P(\varphi_1 \mathbf{U} \varphi_2, u)$. We distinguish two sub-cases: $P(\varphi_1 \mathbf{U} \varphi_2, u) = \top$ and $P(\varphi_1 \mathbf{U} \varphi_2, u) \neq \top$.
  * Sub-case $P(\varphi_1 \mathbf{U} \varphi_2, u) = \top$. We distinguish again three sub-cases:
    · Sub-case $\bigvee_{i=0}^{|u|-1} \left(P(\varphi_2, u^i) \wedge \bigwedge_{j=0}^{i-1} P(\varphi_1, u^j) \right) = \top$. Necessarily, we have $\exists 0 \leq i \leq |u| - 1.\ P(\varphi_2, u^i) \wedge \bigwedge_{j=0}^{i-1} P(\varphi_1, u^j) = \top$. Otherwise, that would mean that $\exists i_1, i_2 \in [0, |u|-1].\ P(\varphi_2, u^{i_1}) \wedge \bigwedge_{j=0}^{i_1-1} P(\varphi_1, u^j) = \neg P(\varphi_2, u^{i_2}) \wedge \bigwedge_{j=0}^{i_2-1} P(\varphi_1, u^j)$ and we would obtain a contradiction. From $P(\varphi_2, u^i) \wedge \bigwedge_{j=0}^{i-1} P(\varphi_1, u^j) = \top$, we have $P(\varphi_2, u^i) = \top$ and $\bigwedge_{j=0}^{i-1} P(\varphi_1, u^j) = \top$. Using the induction hypothesis on φ_1 and φ_2, we obtain $u^i \cdot w \models \varphi_2$ and $\forall 0 \leq j < i.\ u^j \cdot w \models \varphi_1$. According to the LTL semantics of operator $\mathbf{U}$, it means $u \cdot w \models \varphi_1 \mathbf{U} \varphi_2$.
    · Sub-case $\bigwedge_{i=0}^{|u|-1} P(\varphi_1, u^i) \wedge \varphi_1 \mathbf{U} \varphi_2 = \top$. In this case, we have necessarily $\varphi_1 \mathbf{U} \varphi_2 = \top$, and this case reduces to the case where $\varphi = \top$.
    · Sub-case $\bigvee_{i=0}^{|u|-1} \left(P(\varphi_2, u^i) \wedge \bigwedge_{j=0}^{i-1} P(\varphi_1, u^j) \right) \neq \top$ and $\bigwedge_{i=0}^{|u|-1} P(\varphi_1, u^i) \wedge \varphi_1 \mathbf{U} \varphi_2 \neq \top$. We have then

$$\bigvee_{i=0}^{|u|-1} \left(P(\varphi_2, u^i) \wedge \bigwedge_{j=0}^{i-1} P(\varphi_1, u^j) \right) = \neg\left(\bigwedge_{i=0}^{|u|-1} P(\varphi_1, u^i) \wedge \varphi_1 \mathbf{U} \varphi_2 \right).$$

Let us suppose that $\forall i \in \mathbb{N}^{\geq 0}.\ (u \cdot \sigma) \not\models \varphi_2$. Following the induction hypothesis on φ_2, it means in particular that $\forall 0 \leq i \leq |u| - 1.\ w \not\models P(\varphi_2, u^i)$. Then, since $w \models P(\varphi_2 \mathbf{U} \varphi_2)$, it would imply that $w \models \bigwedge_{i=0}^{|u|-1} P(\varphi_1, u^i) \wedge \varphi_1 \mathbf{U} \varphi_2$. But, from $w \models \varphi_1 \mathbf{U} \varphi_2$, we would obtain a contradiction according to the LTL semantics. Hence, let us consider i the minimal $k \in \mathbb{N}^{\geq 0}$ s.t. $(u \cdot w)^k \models \varphi_2$. If $i > |u| - 1$, then similarly we have $w \models \bigwedge_{i=0}^{|u|-1} P(\varphi_1, u^i) \wedge \varphi_1 \mathbf{U} \varphi_2$. It follows that $\forall 0 \leq l \leq |u| - 1.\ u^l \cdot w \models \varphi_1$ and $\forall |u| - 1 \leq l < i.\ (u \cdot w)^l \models \varphi_1$, and thus $u \cdot w \models \varphi_1 \mathbf{U} \varphi_2$. Else ($i \leq |u| - 1$), we can follow a similar reasoning to obtain the expected result.
  * Sub-case $P(\varphi_1 \mathbf{U} \varphi_2, u) = \top$. Similarly, in this case, we can show that $\exists k \in \mathbb{N}^{\geq 0}.\ (u \cdot w)^k \models \varphi_2$. Then we consider k_{min} the minimal k s.t. $(u \cdot w)^k \models \varphi_2$. Then, we can show that $\forall k' < k_{min}.\ (u \cdot w)^{k'} \models \varphi_1$. And then $u \cdot w \models \varphi_1 \mathbf{U} \varphi_2$.

$\square$

Proof for Theorem 1. We shall prove the following statement:

$$\forall u \in \Sigma^+.\forall \varphi \in \text{LTL}. \quad v = P(\varphi, u)$$
$$\Rightarrow (v = \top \Rightarrow u \models_3 \varphi = \top) \wedge (v = \bot \Rightarrow u \models_3 \varphi = \bot).$$

The proof uses the definition of the LTL semantics (Definition 1), the definition of good and bad prefixes (Definition 8), the progression function (Definition 3), and Lemma 1.

Proof. According to Lemma 4, we have $\forall u \in \Sigma^+.\forall w \in \Sigma^\omega. u \cdot w \models \varphi \Leftrightarrow w \models P(\varphi, u)$. Consequently, we have $\forall u \in \Sigma^+.\forall w \in \Sigma^\omega. u \cdot w \models \varphi \Leftrightarrow \forall u \in \Sigma^+.\forall w \in \Sigma^\omega. w \models P(\varphi, u)$ and $\forall u \in \Sigma^+.\forall w \in \Sigma^\omega. u \cdot w \not\models \varphi \Leftrightarrow \forall u \in \Sigma^+.\forall w \in \Sigma^\omega. w \not\models P(\varphi, u)$. Consequently, when $P(\varphi, u) = \top$, we have $\forall u \in \Sigma^+.\forall w \in \Sigma^\omega. u \cdot w \models \varphi$, i.e., $u \in \text{good}(\varphi)$. Also, when $P(\varphi, u) = \bot$, we have $\forall u \in \Sigma^+.\forall w \in \Sigma^\omega. u \cdot w \not\models \varphi$, i.e., $u \in \text{bad}(\varphi)$. $\square$

A.2 Proofs for Section 5

Proof of Corrolary 1. We shall prove the following statement:

$$|\mathcal{M}| = 1 \Rightarrow \forall u \in \Sigma^*.\forall \varphi \in \text{LTL}. u \models_3 \varphi = u \models_D \varphi$$

Proof. The proof is trivial, since in case of one component in the system, the extended progression rule (1) is reduced to its initial definition in the centralised case, i.e., $\forall p \in AP.\forall \sigma \in \Sigma. P(p, \sigma, AP_1) = P(p, \sigma)$. Moreover, no past goal is generated, i.e., the extended progression rule (2) is never applied. $\square$

A.3 Proofs for Section 6

Let us first formalize a bit more Algorithm L by introducing some additional notation.

- $\text{send}(i, t, j) \in \{\text{true}, \text{false}\}$ is a predicate indicating whether or not the monitor i sends a formula to monitor j at time t with $i \neq j$.
- $\text{send}(i, t) \in \{\text{true}, \text{false}\}$ is a predicate indicating whether or not the monitor i sends a formula to some monitor at time t.
- $\text{kept}(i, t) \in \text{LTL}$ is the local obligation kept by monitor i at time t for the next round (time $t + 1$).
- $\text{received}(i, t, j) \in \text{LTL}$ is the obligation received by monitor i at time t by monitor j with $i \neq j$.
- $\text{received}(i, t) \in \text{LTL}$ is the obligation received by monitor i at time t from all monitors.
- $\text{inlo}(i, t, \varphi) \in \text{LTL}$ is the local obligation of monitor i at time t when monitoring the global specification formula φ, before applying the progression functioni.e, after applying step L3 of Algorithm L.
- $\text{lo}(i, t, \varphi) \in \text{LTL}$ is the local obligation of monitor i at time t when monitoring the global specification formula φ after applying the progression function, i.e, after applying step L4 of Algorithm L.
- $\text{mou}(\varphi) \in \text{sus}(\varphi)$ is the most urgent formula belonging to the set of urgent subformulae of φ.
- $\text{ulo}(i, t, \varphi) = \text{sus}\big(\text{lo}(i, t, \varphi)\big)$ is the set of urgent local obligation of monitor i at time t when monitoring the global specification formula φ.

Based on the previous notation and Algorithm L, we have the following relations:

- $\text{send}(i, t, j)$ is true if monitor M_j is the first monitor containing the most urgent obligation contained in the local obligation of M_i, according to the order in $[1, m]$. Formally:

$$\text{send}(i, t, j) = \begin{cases} \text{true} & \text{if } M_j = \text{Mon}\big(M_i, \text{Prop}(\text{ulo}(i, t, \varphi))\big) \wedge \text{ulo}(i, t, \varphi) \neq \emptyset \\ \text{false} & \text{otherwise} \end{cases}$$

- $\text{send}(i, t)$ is true if monitor M_i sends his local obligation to some monitor. Formally: $\text{send}(i, t) = \exists j \in [1, n] \setminus \{i\}. \text{send}(i, t, j)$.
- $\text{kept}(i, t) \in \text{LTL}$ is either # if M_i sends its local obligation to some monitor at time $t - 1$ or its local obligation at time $t - 1$ otherwise. Formally:

$$\text{kept}(i, t) = \begin{cases} \# & \text{if } \exists j \in [1, n] \setminus \{i\}. \text{send}(i, t - 1, j) \\ \text{lo}(i, t - 1, \varphi) & \text{else} \end{cases}$$

- received(i, t, j) is the local obligation of M_j received by M_i at time t if $t \geq 1$ and M_j sends actually something to M_i. Formally:

$$\text{received}(i, t, j) = \begin{cases} \text{lo}(j, t-1, \varphi) \text{ if } \exists j \in [1, n] \setminus \{i\}. \text{ send}(j, t-1, i) \wedge t \geq 1 \\ \# \quad\quad\quad\quad\quad\quad \text{else} \end{cases}$$

- received(i, t) is the conjunction of all obligations received by monitor i from all other monitors at time t. Formally:

$$\text{received}(i, t) = \bigwedge_{j=1, j \neq i}^{|\mathcal{M}|} \text{received}(i, t, j)$$

- inlo(i, t, φ) is
  - at time $t \geq 1$ what was kept by M_i at time $t - 1$ and the received obligation at time t;
  - at time $t = 0$ the initial obligation, i.e., the global specification φ.

  Formally:

$$\text{inlo}(i, t, \varphi) = \begin{cases} \varphi \quad\quad\quad\quad\quad\quad\quad\quad\quad \text{if } t = 0 \\ \text{kept}(i, t-1) \wedge \text{received}(i, t) \text{ else} \end{cases}$$

- lo(i, t, φ) is
  - at time $t \geq 1$ the result of progressing what was kept by M_i at time $t - 1$ and the received obligation at time t with the current local event $u_i(t)$;
  - at time $t = 0$ the result of progressing the initial obligation, i.e., the global specification with the current local event $u_i(0)$.

  Formally:

$$\text{lo}(i, t, \varphi) = \begin{cases} P(\varphi, u_i(0), AP_i) \quad\quad\quad\quad\quad\quad\quad\quad \text{if } t = 0 \\ P(\text{kept}(i, t-1) \wedge \text{received}(i, t), u_i(t), AP_i) \text{ else} \end{cases}$$

Now, we can clearly state the theorem:

$$\forall t \in \mathbb{N}^{\geq 0}.\forall \varphi \in \text{LTL}.\forall i \in [1, n].\forall \overline{\mathbf{X}}^d p \in \text{ulo}(i, t, \varphi). \; d \leq \min(n, t+1)$$

Preliminaries to the proof. Let us first start with some remarks. At step L3 in Algorithm L, the local obligation of a monitor M_i is defined to be $\varphi_i^t \wedge \bigwedge_{j \in [1, m], j \neq i} \varphi_j$ where φ_j is an obligation received from monitor M_j and φ_i^t is the local obligation kept from time $t - 1$ (if $t = 0$, $\varphi_i^t = \varphi$). Let us note that the local obligation kept by the monitor from time $t - 1$ to time t, with $t \geq 1$, are not urgent. The result should thus be established on the *urgent* local obligations transmitted and rewritten by local monitors. More formally, this is stated by the following lemma.

Lemma 5. *According to Algorithm L, we have:*

$$\text{ulo}(i, t, \varphi) = \bigcup_{j=1, j \neq i}^{|\mathcal{M}|} \text{sus}\big(P(\text{received}(i, t), u_i(t), AP_i)\big)$$

Proof. First let us notice that the formulae kept by any monitor M_i at any time t are not urgent. Indeed, we have: $\forall i \in [1, n].\forall t \in \mathbb{N}^{\geq 0}$.

$$\text{sus}(\text{kept}(i, t)) = \begin{cases} \text{sus}(\#) \quad\quad\quad \text{if } \exists j \in [1, n] \setminus \{i\}. \text{ send}(i, t, j) \\ \text{sus}(\text{lo}(i, t-1, \varphi)) \text{ if } \text{sus}(\text{lo}(i, t-1, \varphi)) = \emptyset \end{cases}$$

That is $\forall i \in [1, n].\forall t \geq 0. \; \text{sus}(\text{kept}(i, t)) = \emptyset$. Thus, $\forall i \in [1, n].\forall t \in \mathbb{N}^{\geq 0}.\forall \varphi \in \text{LTL}$.

$$\begin{aligned}
&\text{ulo}(i, t, \varphi) \\
&= \text{sus}\big(P(\text{received}(i, t), u_i(t), AP_i)\big) \\
&= \text{sus}\big(P(\bigwedge_{j=1, j \neq i}^{|\mathcal{M}|} \text{received}(i, t, j), u_i(t), AP_i)\big) \text{ (definition of received}(i, t, j)) \\
&= \text{sus}\big((\bigwedge_{j=1, j \neq i}^{|\mathcal{M}|} P(\text{received}(i, t), u_i(t), AP_i))\big) \quad \text{(progression on events)} \\
&= \bigcup_{j=1, j \neq i}^{|\mathcal{M}|} \text{sus}\big(P(\text{received}(i, t), u_i(t), AP_i)\big) \quad \text{(definition of sus)}
\end{aligned}$$

$\square$

Another last lemma will be needed before entering specifically into the proof. This lemma states that if a past obligation $\overline{\mathbf{X}}^d p$ is part of a progressed formula, then the past obligation $\overline{\mathbf{X}}^{d-1} p$ is part of its un-progressed form. More formally, this is stated by the following lemma.

Lemma 6. *Let us consider $\mathcal{M} = \{M_1, \ldots, M_n\}$ where each monitor M_i has a set of local atomic propositions $AP_i = \Pi_i(AP)$ and observes the set of events Σ_i, we have:*

$$\forall i \in [1, n].\forall \sigma \in \Sigma_i.\forall \varphi \in \text{LTL}.\forall \overline{\mathbf{X}}^d \in \text{sus}\left(P(\varphi, \sigma, AP_i)\right). d > 1 \Rightarrow \overline{\mathbf{X}}^{d-1} p \in \text{sus}(\varphi)$$

Proof. Let us consider $\sigma \in \Sigma, \Sigma_i \subseteq \Sigma$. The proof is done by a structural induction on $\varphi \in \text{LTL}$.
  Base Case: $\varphi \in \{\top, \bot, p' \in AP\}$

- Case $\varphi = \top$. In this case, the proof is trivial since $P(\top, \sigma, AP_i) = \top$ and $\text{sus}(\top) = \emptyset$.
- Case $\varphi = \bot$. This case is similar to the previous one.
- Case $\varphi = p' \in AP$. If $p' \in AP_i$, then $P(p', \sigma, AP_i) \in \{\top, \bot\}$ and $\text{sus}(P(p', \sigma, AP_i)) = \emptyset$. Else $(p' \notin AP_i)$, $P(p', \sigma, AP_i) = \overline{\mathbf{X}}p'$ and $\text{sus}\left(P(p', \sigma, AP_i)\right) = \emptyset$.

Induction Case: $\varphi \in \{\neg\varphi', \varphi_1 \vee \varphi_2, \varphi_1 \wedge \varphi_2, \overline{\mathbf{X}}^{d'} p', \mathbf{G}\varphi', \mathbf{F}\varphi', \mathbf{X}\varphi', \varphi_1 \mathbf{U}\varphi_2\}$. Our induction hypothesis states that the result holds for some formulae $\varphi', \varphi_1, \varphi_2 \in \text{LTL}$.

- Case $\varphi = \neg\varphi'$. On one hand, we have

$$\text{sus}\left(P(\neg\varphi', \sigma, AP_i)\right) = \text{sus}\left(\neg P(\varphi', \sigma, AP_i)\right)$$
$$= \text{sus}\left(P(\varphi', \sigma, AP_i)\right).$$

  On the other hand, we have $\text{sus}(\neg\varphi') = \text{sus}(\varphi')$. Thus, by applying directly the induction hypothesis on φ', we obtain the expected result.
- Case $\varphi = \varphi_1 \vee \varphi_2$. On one hand, we have

$$\text{sus}\left(P(\varphi_1 \vee \varphi_2, \sigma, AP_i)\right) = \text{sus}\left(P(\varphi_1, \sigma, AP_i) \vee P(\varphi_2, \sigma, AP_i)\right)$$
$$= \text{sus}\left(P(\varphi_1, \sigma, AP_i)\right) \cup \text{sus}\left(P(\varphi_2, \sigma, \Sigma_i)\right).$$

  Thus, $\overline{\mathbf{X}}^d \in \text{sus}\left(P(\varphi_1 \wedge \varphi_2, \sigma, AP_i)\right)$ implies that $\overline{\mathbf{X}}^d p \in \text{sus}\left(P(\varphi_1, \sigma, AP_i)\right)$ or $\overline{\mathbf{X}}^d p \in \text{sus}\left(P(\varphi_2, \sigma, AP_i)\right)$. On the other hand, $\text{sus}(\varphi_1 \wedge \varphi_2) = \text{sus}(\varphi_1) \cup \text{sus}(\varphi_2)$. Hence, the result can be obtained by applying the induction hypothesis on either φ_1 or φ_2 depending on whether $\overline{\mathbf{X}}^d p \in \text{sus}\left(P(\varphi_1, \sigma, AP_i)\right)$ or $\overline{\mathbf{X}}^d p \in \text{sus}\left(P(\varphi_2, \sigma, AP_i)\right)$.
- Case $\varphi = \overline{\mathbf{X}}^{d'} p'$ for some $d' \in \mathbb{N}$ and $p' \in AP$. One one hand, if $p' \in AP_i$, then it implies that $P(\overline{\mathbf{X}}^{d'} p', \sigma, AP_i) \in \{\top, \bot\}$. Else $(p' \notin AP_i)$, we have $P(\overline{\mathbf{X}}^{d'} p', \sigma, AP_i) = \overline{\mathbf{X}}^{d'+1} p'$. On the other hand, we have $\text{sus}(\overline{\mathbf{X}}^{d'} p') = \{\overline{\mathbf{X}}^{d'} p'\}$.
- Case $\varphi = \mathbf{G}\varphi'$. By definition of the progression rule for $\mathbf{G}$ and the definition of sus, we have

$$\text{sus}\left(P(\mathbf{G}\varphi', \sigma, AP_i)\right)$$
$$= \text{sus}\left(P(\varphi', \sigma, AP_i) \wedge \mathbf{G}\varphi'\right)$$
$$= \text{sus}\left(P(\varphi', \sigma, AP_i)\right).$$

  Since φ' is behind a future temporal operator, the only case where $\text{sus}\left(P(\varphi', \sigma, AP_i)\right) \neq \emptyset$ is when φ' is a state-formula. In that case, we have $\overline{\mathbf{X}}^d p \in \text{sus}\left(P(\varphi', \sigma, AP_i)\right)$ implies that $d = 1$.
- Cases $\varphi \in \{\mathbf{F}\varphi', \mathbf{X}\varphi', \varphi_1 \mathbf{U}\varphi_2\}$. These cases are similar to the previous one.

$\qquad\square$

Back to the proof of Theorem 2. We have to prove that for any $\overline{\mathbf{X}}^m p \in \text{LTL}$, a local obligation of some monitor $M_i \in \mathcal{M}, m \leq \min(|\mathcal{M}|, t + 1)$ at any time $t \in \mathbb{N}^{\geq 0}$. We will suppose that there are at least two components in the system (otherwise, the proof is trivial), i.e., $|\mathcal{M}| \geq 2$. The proof is done by distinguishing three cases according to the value of $t \in \mathbb{N}^{\geq 0}$.

First case: $t = 0$. In this case, we shall prove that $m \leq 1$. The proof is done by a structural induction on $\varphi \in \text{LTL}$. Recall that for this case, where $t = 0$, we have $\forall i \in [1, |\mathcal{M}|]$. $\text{lo}(i, 0, \varphi) = P(\varphi, u_i(0), AP_i)$.

Base case: $\varphi \in \{\top, \bot, p \in AP\}$.

- Case $\varphi = \top$. In this case we have $\forall i \in [1, |\mathcal{M}|]$. $\text{lo}(i, 0, \top) = P(\top, u_i(0), AP_i) = \top$. Moreover, $\text{sus}(\top) = \emptyset$.
- Case $\varphi = \bot$. This case is symmetrical to the previous one.
- Case $\varphi = p \in AP$. We distinguish two cases: $p \in AP_i$ and $p \notin AP_i$. If $p \in AP_i$, then $\text{lo}(i, 0, p) \in \{\top, \bot\}$ and $\text{sus}\left(\text{lo}(i, 0, p)\right) = \emptyset$. Else ($p \notin AP_i$), we have $\text{lo}(i, 0, p) = \overline{\mathbf{X}}p$, and $\text{sus}\left(\text{lo}(i, 0, p)\right) = \{\overline{\mathbf{X}}p\} = \{\overline{\mathbf{X}}^1 p\}$.

Structural Induction Case: $\varphi \in \{\neg\varphi', \varphi_1 \vee \varphi_2, \varphi_1 \wedge \varphi_2, \mathbf{G}\varphi', \mathbf{F}\varphi', \mathbf{X}\varphi', \varphi_1 \mathbf{U}\varphi_2\}$. Our induction hypothesis states that the result holds for some formulae $\varphi', \varphi_1, \varphi_2 \in \text{LTL}$.

- Case $\varphi = \varphi_1 \vee \varphi_2$. We have:

$$
\begin{aligned}
\text{lo}(i, 0, \varphi_1 \vee \varphi_2) & \\
&= P(\varphi_1 \vee \varphi_2, u_i(0), AP_i) && \text{(lo definition for } t = 0) \\
&= P(\varphi_1, u_i(0), AP_i) \vee P(\varphi_2, u_i(0), AP_i) && \text{(progression on events)} \\
&= \text{lo}(i, 0, \varphi_1) \vee \text{lo}(i, 0, \varphi_2) && \text{(lo definition for } t = 0) \\
\text{sus}\left(\text{lo}(i, 0, \varphi_1 \vee \varphi_2)\right) & \\
&= \text{sus}\left(\text{lo}(i, 0, \varphi_1) \vee \text{lo}(i, 0, \varphi_2)\right) \\
&= \text{sus}\left(\text{lo}(i, 0, \varphi_1)\right) \cup \text{sus}\left(\text{lo}(i, 0, \varphi_2)\right) && \text{(sus definition)}
\end{aligned}
$$

We can apply the induction hypothesis on φ_1 and φ_2 to obtain successively:

$$
\begin{aligned}
&\forall t \geq \mathbb{N}^{\geq 0}.\forall \varphi \in \text{LTL}.\forall \overline{\mathbf{X}}^m p \in \text{sus}\left(\text{lo}(i, t, \varphi_1)\right). m \leq 1 \\
&\forall t \geq \mathbb{N}^{\geq 0}.\forall \varphi \in \text{LTL}.\forall \overline{\mathbf{X}}^m p \in \text{sus}\left(\text{lo}(i, t, \varphi_2)\right). m \leq 1 \\
&\forall t \geq \mathbb{N}^{\geq 0}.\forall \varphi \in \text{LTL}.\forall \overline{\mathbf{X}}^m p \in \text{sus}\left(\text{lo}(i, t, \varphi_1)\right) \cup \text{sus}\left(\text{lo}(i, t, \varphi_2)\right). m \leq 1
\end{aligned}
$$

- Case $\varphi = \neg\varphi'$. We have:

$$
\begin{aligned}
\text{lo}(i, 0, \neg\varphi') &= P(\neg\varphi', u_i(0), AP_i) && \text{(lo definition)} \\
&= \neg P(\varphi', u_i(0), AP_i) && \text{(progression on events)} \\
\text{sus}\left(\text{lo}(i, 0, \neg\varphi')\right) &= \text{sus}\left(\neg P(\varphi', u_i(0), AP_i)\right) \\
&= \text{sus}\left(P(\varphi', u_i(0), AP_i)\right) && \text{(sus definition)} \\
&= \text{sus}\left(\text{lo}(i, 0, \varphi')\right)
\end{aligned}
$$

- Case $\varphi = \mathbf{X}\varphi'$. We have:

$$
\begin{aligned}
\text{lo}(i, 0, \mathbf{X}\varphi') &= P(\mathbf{X}\varphi', u_i(0), AP_i) && \text{(lo definition)} \\
&= \varphi' && \text{(progression on events)} \\
\text{sus}\left(\text{lo}(i, 0, \mathbf{X}\varphi')\right) &= \text{sus}(\varphi')
\end{aligned}
$$

Since φ' is behind a future temporal operator, we have $\text{sus}(\varphi') = \emptyset$.

- Case $\varphi = \mathbf{G}\varphi'$. We have:

$$
\begin{aligned}
\text{lo}(i, 0, \mathbf{G}\varphi') &= P(\mathbf{G}\varphi', u_i(0), AP_i) && \text{(lo definition)} \\
&= P(\varphi', u_i(0), AP_i) \wedge \mathbf{G}\varphi' && \text{(progression on events)} \\
&= \text{lo}(i, 0, \varphi') \wedge \mathbf{G}\varphi' && \text{(lo definition for } \varphi') \\
\text{sus}\left(\text{lo}(i, 0, \mathbf{G}\varphi')\right) &= \text{sus}\left(\text{lo}(i, 0, \varphi') \wedge \mathbf{G}\varphi'\right) \\
&= \text{sus}\left(\text{lo}(i, 0, \varphi')\right) \cup \text{sus}(\mathbf{G}\varphi') && \text{(sus definition)} \\
&= \text{sus}\left(\text{lo}(i, 0, \varphi')\right) && (\text{sus}(\mathbf{G}\varphi') = \emptyset)
\end{aligned}
$$

- Case $\varphi = \mathbf{F}\varphi'$. This case is similar to the previous one.

– Case $\varphi = \varphi_1 \mathbf{U} \varphi_2$. We have:

$$\mathrm{lo}(i, 0, \varphi_1 \mathbf{U} \varphi_2)$$

(lo definition)

$$= P(\varphi_1 \mathbf{U} \varphi_2, u_i(0), AP_i)$$

(progression on events)

$$= P(\varphi_2, u_i(0), AP_i) \vee \big(P(\varphi_1, u_i(0), AP_i) \wedge \varphi_1 \mathbf{U} \varphi_2 \big)$$

(lo definition for φ_1 and φ_2)

$$= \mathrm{lo}(i, 0, \varphi_2) \vee \mathrm{lo}(i, 0, \varphi_1) \wedge \varphi_1 \mathbf{U} \varphi_2$$

$$\mathrm{sus}\big(\mathrm{lo}(i, 0, \varphi_1 \mathbf{U} \varphi_2) \big)$$

$$= \mathrm{sus}\big(\mathrm{lo}(i, 0, \varphi_1) \vee \mathrm{lo}(i, 0, \varphi_2) \wedge \varphi_1 \mathbf{U} \varphi_2 \big)$$

(sus definition)

$$= \mathrm{sus}\big(\mathrm{lo}(i, 0, \varphi_2) \big) \cup \mathrm{sus}\big(\mathrm{lo}(i, 0, \varphi_1) \big) \cup \mathrm{sus}(\varphi_1 \mathbf{U} \varphi_2)$$

(sus($\varphi_1 \mathbf{U} \varphi_2 = \emptyset$)

$$= \mathrm{sus}\big(\mathrm{lo}(i, 0, \varphi_2) \big) \cup \mathrm{sus}\big(\mathrm{lo}(i, 0, \varphi_1) \big)$$

For $t \geq 1$, the proof is done by *reductio ad absurdum*. Let us consider some $t \in \mathbb{N}$ and suppose that the theorem does not hold at time t. It means that:

$$\exists \varphi \in \mathrm{LTL}. \exists i \in [1, |\mathcal{M}|]. \exists \overline{\mathbf{X}}^d p \in \mathrm{ulo}(i, t, \varphi). \ d > \min(|\mathcal{M}|, t+1).$$

According to Lemma 5, since $\mathrm{ulo}(i, t, \varphi) = \bigcup_{j=1, j \neq i}^{|\mathcal{M}|} \mathrm{sus}\big(P(\mathrm{received}(i, t), u_i(t)) \big)$, it means that $\exists j_1 \in [1, |\mathcal{M}|] \setminus \{i\}. \overline{\mathbf{X}}^d p \in \mathrm{sus}\big(P(\mathrm{received}(i, t, j_1), u_i(t), AP_i) \big)$. Using Lemma 6, we have $\overline{\mathbf{X}}^{d-1} p \in \mathrm{sus}(\mathrm{received}(i, t, j_1))$. It implies that $\mathrm{send}(j_1, t-1, i) = \mathrm{true}$ and $M_i = \mathrm{Mon}\big(M_{j_1}, \mathrm{Prop}(\mathrm{ulo}(j_1, t-1, \varphi)) \big)$. We deduce that $i = \min \big\{ j \in [1, |\mathcal{M}|] \setminus \{j_1\} \mid \exists p \in \mathrm{Prop}(\mathrm{ulo}(j, t-1, \varphi)). \ p \in AP_i \big\}$. Moreover, from $\overline{\mathbf{X}}^d p \in \mathrm{ulo}(i, t, \varphi)$, we find $p \notin AP_{i'}$, with $i < i'$.

We can apply the same reasoning on $\overline{\mathbf{X}}^{d-1} p$ to find that $i < j_1 < i'$ and $p \notin \Pi_{j_1}(AP)$. Following the same reasoning and using Lemma 6, we can find a set of indexes $\{j_1, \ldots, j_d\}$ s.t.

$$\{j_1, \ldots, j_d\} \supseteq [1, |\mathcal{M}|]$$
$$\wedge \ \forall j \in \{j_1, \ldots, j_d\}. \ p \notin AP_j \wedge j \in [1, |\mathcal{M}|]$$

Moreover, due to the ordering between components, we know that $\forall k_1, k_2 \in [1, d]. \ k_1 < k_2 \Rightarrow j_{k_1} < j_{k_2}$.

Case $0 < t < |\mathcal{M}|$. In this case we have $d > t + 1$, and thus, we have $\overline{\mathbf{X}}^{d'} p \in \mathrm{sus}\big(\mathrm{lo}(j_t, 0, \varphi) \big)$ with $d' > 1$ which is a contradiction with the result shown for $t = 0$.

Case $t \geq |\mathcal{M}|$. In this case, $\forall k_1, k_2 \in [1, d]. \ k_1 < k_2 \Rightarrow j_{k_1} < j_{k_2}$ implies that $\forall j_{k_1}, j_{k_2} \in \{j_1, \ldots, j_d\}. \ k_1 \neq k_2 \Rightarrow j_{k_1} \neq j_{k_2}$. Hence, we have $p \notin \bigcup_{j=j_1}^{j_d} AP_j \supseteq AP$. This is impossible. $\qquad \square$

Proof of Theorem 3. We shall prove that the decentralised monitoring algorithm is sound, i.e., whenever the decentralised monitoring algorithm yields a verdict for a given trace, then the corresponding centralized algorithm yields the same verdicts.

Some intermediate lemmas. Before proving the main result of this paper, we introduce some intermediate lemmas. The following lemma extends Lemma 1 to the decentralised case, i.e., it states that the progression function mimics LTL semantics in the decentralised case.

Lemma 7. *Let φ be an* LTL *formula, $\sigma \in \Sigma$ an event, σ_i a local event observed by monitor M_i, and w an infinite trace, we have $\sigma \cdot w \models \varphi \Leftrightarrow (\sigma \cdot w)^1 \models P(\varphi, \sigma_i, \Sigma_i)$.*

Proof. We shall prove that:

$$\forall i \in [1, n]. \, \forall \varphi \in \text{LTL}. \forall \sigma \in \Sigma. \forall \sigma_i \in \Sigma_i. \forall w \in \Sigma^\omega.$$
$$\sigma \cdot w \models \varphi \Leftrightarrow (\sigma \cdot w)^1 \models P(\varphi, \sigma_i, AP_i).$$

The proof is done by induction on the formula $\varphi \in \text{LTL}$. Notice that when φ is not an atomic proposition, the lemma reduces to Lemma 1. Thus, we just need to treat the case $\varphi = p \in AP$.

If $\varphi = p \in AP$. We have $\sigma \cdot w \models p \Leftrightarrow p \in \sigma$. Let us consider $i \in [1, n]$, according to the definition of the progression function (1):

$$P(p, \sigma_i, AP_i) = \begin{cases} \top & \text{if } p \in \sigma_i, \\ \bot & \text{if } p \notin \sigma_i \wedge p \in AP_i, \\ \overline{\mathbf{X}}p & \text{otherwise,} \end{cases}$$

Let us distinguish three cases.

- Suppose $p \in \sigma_i$. On one hand, we have $p \in \sigma$ and then $\sigma \cdot w \models p$. On the other hand, we have $P(p, \sigma_i, AP_i) = \top$ and thus $w \models P(p, \sigma_i, AP_i)$.
- Suppose $p \notin \sigma_i$ and $p \in AP_i$. One one hand, we have $p \in \sigma$, and, because $p \in AP_i$ we have $p \notin \sigma$; and thus $\sigma \cdot w \not\models p$. On the other hand, we have $P(p, \sigma_i, AP_i) = \bot$.
- Suppose $p \notin \sigma_i$ and $p \notin AP_i$, we have $(\sigma \cdot w)^1 \models \overline{\mathbf{X}}p \Leftrightarrow \big((\sigma \cdot w)^{-1}\big)^1 \models \overline{\mathbf{X}}p \Leftrightarrow \sigma \cdot w \models p$.

$\square$

The following lemma states that "the satisfaction of an LTL formula" is propagated by the decentralised monitoring algorithm.

Lemma 8.
$$\forall t \in \mathbb{N}^{\geq 0}. \forall i \in [1, n]. \forall \varphi \in \text{LTL}. \forall w \in \Sigma^\omega.$$
$$\text{inlo}(i, t, \varphi) \neq \# \Rightarrow w \models \varphi \Leftrightarrow w^t \models \text{inlo}(i, t, \varphi)$$

Proof. The proof is done by induction on $t \in \mathbb{N}^{\geq 0}$.

- For $t = 0$, the proof is trivial since $\forall i \in [1, n]. \forall \varphi \in \text{LTL}. \text{ inlo}(i, 0, \varphi) = \varphi$ and $w^0 = w$.
- Let us consider some $t \in \mathbb{N}^{\geq 0}$ and suppose that the lemma holds. Let us consider $i \in [1, n]$, we have:

$$\text{inlo}(i, t + 1, \varphi) = \text{kept}(i, t) \wedge \text{received}(1, t + 1).$$

Let us now distinguish four cases according to the communication performed by local monitors at the end of time t, i.e., according to $\text{send}(i, t)$ and $\text{send}(j, t, i)$, for $j \in [1, n] \setminus \{i\}$.

- If $\text{send}(i, t) = \text{false}$ and $\exists j \in [1, n] \setminus \{i\}. \text{ send}(j, t, i) = \text{true}$. Then, we have:

$$\text{inlo}(i, t + 1, \varphi) = P\big(\text{inlo}(i, t, \varphi) \wedge \bigwedge_{j \in J} \text{inlo}(j, t, \varphi), u_i(t + 1), \Sigma_i\big).$$

where $\forall j \in J. \text{ send}(j, t, i) = \text{true}$. Applying the definition of the progression function, we have:

$$\text{inlo}(i, t + 1, \varphi)$$
$$= P\big(\text{inlo}(i, t, \varphi), u_i(t + 1), \Sigma_i\big) \wedge \bigwedge_{j \in J} P\big(\text{inlo}(j, t, \varphi), u_i(t + 1), \Sigma_i\big).$$

Now, we have:
$$w^{t+1} \models \text{inlo}(i, t + 1, \varphi)$$
$$\Leftrightarrow$$
$$\big(w^{t+1} \models P\big(\text{inlo}(i, t, \varphi), u_i(t + 1), \Sigma_i\big)\big)$$
$$\wedge \big(\forall j \in J. \, w^{t+1} \models P\big(\text{inlo}(j, t, \varphi), u_i(t + 1), \Sigma_i\big)\big)$$

26

With:

$$w^{t+1} \models P\big(\mathrm{inlo}(i,t,\varphi), u_i(t+1), \Sigma_i\big)$$
$$\Leftrightarrow (w^t)^1 \models P\big(\mathrm{inlo}(i,t,\varphi), u_i(t+1), \Sigma_i\big) \qquad (w^{t+1} = (w^t)^1)$$
$$\Leftrightarrow (w(t) \cdot w^{t+1})^1 \models P\big(\mathrm{inlo}(i,t,\varphi), u_i(t+1), \Sigma_i\big) \; ((w^t)^1 = (w(t) \cdot w^{t+1})^1)$$
$$\Leftrightarrow w^t \models \mathrm{inlo}(i,t,\varphi) \qquad \text{(Induction Hypothesis)}$$

And similarly:

$$\forall j \in J. \; w^{t+1} \models P\big(\mathrm{inlo}(j,t,\varphi), u_i(t+1), \Sigma_i\big) \Leftrightarrow w^t \models \mathrm{inlo}(j,t,\varphi)$$

It follows that:

$$w^{t+1} \models \mathrm{inlo}(i,t+1,\varphi) \Leftrightarrow w^t \models \mathrm{inlo}(i,t,\varphi) \wedge \bigwedge_{j \in J} w^t \models \mathrm{inlo}(i,t,\varphi).$$

And finally:

$$w^{t+1} \models \mathrm{inlo}(i,t+1,\varphi) \Leftrightarrow w^t \models \mathrm{inlo}(i,t,\varphi).$$

- If $\mathrm{send}(i,t) = \mathrm{true}$ and $\exists j \in [1,n] \setminus \{i\}. \; \mathrm{send}(j,t,i) = \mathrm{true}$. Then, we have:

$$\mathrm{inlo}(i,t+1,\varphi) = P\big(\# \wedge \bigwedge_{j \in J} \mathrm{inlo}(j,t,\varphi), u_i(t+1), \Sigma_i\big)$$
$$= P\big(\bigwedge_{j \in J} \mathrm{inlo}(j,t,\varphi), u_i(t+1), \Sigma_i\big)$$

where $\forall j \in J. \; \mathrm{send}(j,t,i) = \mathrm{true}$. The previous reasoning can be followed in the same manner to obtain the expected result.

- If $\mathrm{send}(i,t) = \mathrm{false}$ and $\forall j \in [1,n] \setminus \{i\}. \; \mathrm{send}(j,t,i) = \mathrm{false}$. Then, we have:

$$\mathrm{inlo}(i,t+1,\varphi) = P\big(\mathrm{inlo}(i,t,\varphi), u_i(t+1), \Sigma_i\big).$$

The previous reasoning can be followed in the exact same manner to obtain the expected result.

- If $\mathrm{send}(i,t) = \mathrm{true}$ and $\forall j \in [1,n] \setminus \{i\}. \; \mathrm{send}(j,t,i) = \mathrm{true}$. Then, we have:

$$\mathrm{inlo}(i,t+1,\varphi) = P(\#, u_i(t+1), \Sigma_i) = \#$$

In this case, the result holds vacuously.

$\square$

Back to the proof of Theorem 3. The soundness of Algorithm L is now a straightforward consequence of the two previous lemmas (Lemmas 7 and 8). Indeed, let us consider $u \in \Sigma^*$ s.t. $|u| = t$. We have $u \models_D \varphi = \top$ implies that $\exists i \in [1,n]. \; \mathrm{lo}(i,t,\varphi) = \top$ and then $\mathrm{inlo}(i,t+1,\varphi) = \top$. It implies that $\forall w \in \Sigma^\omega. \; w \models \mathrm{inlo}(i,t+1,\varphi)$. Since $|u| = t$, it follows that $\forall w \in \Sigma^\omega. \; (u \cdot w)^t \models \mathrm{inlo}(i,t+1,\varphi)$. Applying Lemma 8, we have $\forall w \in \Sigma^\omega. \; u \cdot w \models \varphi$, i.e., $u \models_3 \varphi = \top$.

The proof for $u \models_D \varphi = \top \Rightarrow u \models_3 \varphi = \top$ is similar. $\square$

Proof of Theorem 4. Let us first define some notations. Consider $\varphi \in \mathrm{LTL}, u \in \Sigma^+, i \in [1, |\mathcal{M}|]$:

- $\mathrm{rp}(\varphi, u)$ is the formula φ where past sub-formulas are removed and replaced by their evaluations using the trace u. Formally:

$$\mathrm{rp}(\varphi, u, i) = \mathtt{match} \; \varphi \; \mathtt{with}$$
$$| \; \overline{\mathbf{X}}^d p \quad \rightarrow \begin{cases} \top & \text{if } p \in u(|u| - d) \\ \bot & \text{otherwise} \end{cases}$$
$$| \; \varphi_1 \wedge \varphi_2 \rightarrow \mathrm{rp}(\varphi_1, u) \wedge \mathrm{rp}(\varphi_2, u)$$
$$| \; \varphi_1 \vee \varphi_2 \rightarrow \mathrm{rp}(\varphi_1, u) \vee \mathrm{rp}(\varphi_2, u)$$
$$| \; \neg \varphi' \quad \rightarrow \neg \mathrm{rp}(\varphi', u)$$
$$| \; \_ \quad \rightarrow \varphi$$

– $\mathrm{rp}(\varphi, u, i)$ is the formula φ where past sub-formulas are removed (if possible) and replaced by their evaluations using only the sub-trace u_i of u.

$$\mathrm{rp}(\varphi, u, i) = \texttt{match } \varphi \texttt{ with}$$
$$\mid \overline{\mathbf{X}}^d p \quad \rightarrow \begin{cases} \top & \text{if } p \in u(|u| - d) \\ \bot & \text{if } p \notin u(|u| - d) \text{ and } p \in AP_i \\ \overline{\mathbf{X}}^d p & \text{otherwise} \end{cases}$$
$$\mid \varphi_1 \wedge \varphi_2 \rightarrow \mathrm{rp}(\varphi_1, u, i) \wedge \mathrm{rp}(\varphi_2, u, i)$$
$$\mid \varphi_1 \vee \varphi_2 \rightarrow \mathrm{rp}(\varphi_1, u, i) \vee \mathrm{rp}(\varphi_2, u, i)$$
$$\mid \neg \varphi' \quad \rightarrow \neg \mathrm{rp}(\varphi', u, i)$$
$$\mid \texttt{-} \quad \rightarrow \varphi$$

The following lemma exhibits some straightforward properties of the function rp.

Lemma 9. *Let φ be an* LTL *formula, $u \in \Sigma^+$ be a trace of length $t + 1$, $i \in [1, |\mathcal{M}|]$ a monitor of one of the component, $u_i(t) \in \Sigma_i$ the last event of u on component i, we have:*

1. $\mathrm{rp}\big(P(\varphi, \sigma_i, AP_i), u\big) = \mathrm{rp}\big(P(\mathrm{rp}(\varphi, u(0) \cdots u(t-1)), \sigma_i, AP_i), u\big)$;
2. $\mathrm{rp}\big(P(\varphi, \sigma_i, AP_i), u\big) = P(\varphi, u(t), AP)$;
3. $P(\varphi, u_i(t), AP_i) = P\big(\mathrm{rp}(\varphi, u(0) \cdots u(t-1), i), u_i(t), AP_i\big)$;
4. $\bigcup_{\varphi' \in \mathrm{sus}(\varphi)} \mathrm{Prop}(\varphi') \subseteq AP_i \Rightarrow \mathrm{rp}(\varphi, u, i) = \mathrm{rp}(\varphi, u)$.
5. *For* $\{i_1, \ldots, i_n\} = [1, |\mathcal{M}|]$: $\mathrm{rp}(\mathrm{rp}(\ldots \mathrm{rp}(\varphi, u, i_1), \ldots), u, i_n) = \mathrm{rp}(\varphi, u)$.

Proof. The proofs of these properties can be done by induction on $\varphi \in$ LTL and follow directly from the definitions of rp and the progression function. $\qquad\square$

Lemma 10. *Any current local obligation where past sub-formulas have been evaluated using the trace read so far is equal to the initial obligation progressed with this same trace read so far. Formally:*

$$\forall u \in \Sigma^+. \forall i \in [1, |\mathcal{M}|]. \forall t \in \mathbb{N}^*.$$
$$|u| = t + 1 \wedge \mathrm{lo}(i, t, \varphi) \neq \# \Rightarrow \mathrm{rp}(\mathrm{lo}(i, t, \varphi), u) = P(\varphi, u).$$

Proof. We shall prove this lemma by induction on $t \in \mathbb{N}^*$. Let us consider some component M_i where $i \in [1, |\mathcal{M}|]$.

– For $t = 0$. In this case, $|u| = 1$ and we have $\mathrm{rp}(\mathrm{lo}(i, 0, \varphi), u) = \mathrm{rp}\big(P(\varphi, \sigma_i, AP_i)\big)$ where $\sigma_i = \Pi(u(0))$. We can obtain the expected result by doing an induction on $\varphi \in$ LTL where the only case interesting case is $\varphi = p \in AP$. According to the definition of the progression function, we have:
$$P(p, \sigma_i, AP_i) = \begin{cases} \top & \text{if } p \in \sigma_i, \\ \bot & \text{if } p \notin \sigma_i \wedge p \in AP_i, \\ \overline{\mathbf{X}} p & \text{otherwise,} \end{cases}$$
Moreover, $p \in \sigma_i$ implies $p \in u(0)$ and $p \notin \sigma_i$ with $p \in AP_i$ implies $\forall j \in [1, |\mathcal{M}|]. p \notin \Pi_j(u(0))$, i.e., $p \notin u(0)$. On one hand, according to the definition of rp, we have:
$$\mathrm{rp}(\overline{\mathbf{X}} p, u(0)) = \begin{cases} \top & \text{if } p \in u(0), \\ \bot & \text{if } p \notin u(0). \end{cases}$$

Thus, we have:
$$\mathrm{rp}\big(P(p, \sigma_i, AP_i)\big) = \begin{cases} \top & \text{if } p \in u(0), \\ \bot & \text{if } p \notin u(0). \end{cases}$$
On the other hand, according to the definition of the progression function, we have:
$$P(\varphi, u(0)) = \begin{cases} \top & \text{if } p \in u(0), \\ \bot & \text{if } p \notin u(0). \end{cases}$$
– Let us consider some $t \in \mathbb{N}^*$ and suppose that the property holds. We have:

$$\mathrm{lo}(i, t + 1, \varphi) = P\big(\mathrm{kept}(i, t) \wedge \mathrm{received}(i, t), u_i(t + 1), AP_i\big).$$

Similarly to the proof of Lemma 8, let us distinguish four cases according to the communication that occurred at the end of time t.

- If $\mathrm{send}(i,t) = \mathrm{false}$ and $\forall j \in [1, |\mathcal{M}|] \setminus \{i\}.\ \mathrm{send}(j,t,i) = \mathrm{false}$. Then, we have:

$$\mathrm{lo}(i, t+1, \varphi) = P(\mathrm{lo}(i, t\varphi), u_i(t+1), AP_i)$$

Let us now compute $\mathrm{rp}(\mathrm{lo}(i, t+1, \varphi), u(0) \cdots u(t+1))$:

$$
\begin{aligned}
\mathrm{rp}(\mathrm{lo}(i, t+1, \varphi), u(0) \cdots u(t+1)) &= \mathrm{rp}(P(\mathrm{lo}(i, t, \varphi), u_i(t+1), AP_i), u(0) \cdots u(t+1)) \\
&\text{(Lemma 9, item 1)} \\
&= \mathrm{rp}(P(\mathrm{rp}(\mathrm{lo}(i, t, \varphi), u(0) \cdots u(t)), u_i(t+1), AP_i), u(0) \cdots u(t+1)) \\
&\text{(induction hypothesis)} \\
&= \mathrm{rp}(P(P(\varphi, u(0) \cdots u(t)), u_i(t+1), AP_i), u(0) \cdots u(t+1)) \\
&\text{(Lemma 9, item 2)} \\
&= P(P(\varphi, u(0) \cdots u(t)), u(t+1), AP) \\
&(P(\varphi, u(0) \cdots u(t)) \text{ is a future formula}) \\
&= P(\varphi, u(0) \cdots u(t+1))
\end{aligned}
$$

- If $\mathrm{send}(i,t) = \mathrm{true}$ and $\exists j \in [1, |\mathcal{M}|] \setminus \{i\}.\ \mathrm{send}(j,t,i) = \mathrm{true}$. Then, we have:

$$\mathrm{lo}(i(i, t+1, \varphi) = P(\bigwedge_{j \in J} \mathrm{lo}(j, t\varphi), u_i(t+1), AP_i)$$

s.t. $\forall j \in J.\ \mathrm{send}(j,t,i) = \mathrm{true}$. Then:

$$
\begin{aligned}
&\mathrm{rp}(\mathrm{lo}(i, t+1, \varphi), u(0) \cdots u(t+1)) \\
&= \mathrm{rp}(P(\bigwedge_{j \in J} \mathrm{lo}(j, t\varphi), u_i(t+1), AP_i), u(0) \cdots u(t+1)) \\
&\text{(definition of the progression function)} \\
&= \mathrm{rp}(\bigwedge_{j \in J} P(\mathrm{lo}(j, t\varphi), u_i(t+1), AP_i), u(0) \cdots u(t+1)) \\
&\text{(definition of rp)} \\
&= \bigwedge_{j \in J} \mathrm{rp}(P(\mathrm{lo}(j, t\varphi), u_i(t+1), AP_i), u(0) \cdots u(t+1)) \\
&\text{(Lemma 9, item 1)} \\
&= \bigwedge_{j \in J} \mathrm{rp}(P(\mathrm{rp}(\mathrm{lo}(j, t\varphi), u(0) \cdots u(t)), u_i(t+1), AP_i), u(0) \cdots u(t+1)) \\
&\text{(induction hypothesis)} \\
&= \bigwedge_{j \in J} \mathrm{rp}(P(P(\varphi, u(0) \cdots u(t)), u_i(t+1), AP_i), u(0) \cdots u(t+1)) \\
&\text{(Lemma 9, item 2)} \\
&= \bigwedge_{j \in J} \mathrm{rp}(P(\varphi, u(0) \cdots u(t) \cdot u(t+1))) \\
&(P(\varphi, u(0) \cdots u(t+1)) \text{ is a future formula}) \\
&= \bigwedge_{j \in J} P(\varphi, u(0) \cdots u(t+1)) \quad = P(\varphi, u(0) \cdots u(t+1))
\end{aligned}
$$

- If $\mathrm{send}(i,t) = \mathrm{false}$ and $\exists j \in [1, |\mathcal{M}|] \setminus \{i\}.\ \mathrm{send}(j,t,i) = \mathrm{true}$. Then, we have:

$$
\begin{aligned}
\mathrm{lo}(i, t+1, \varphi) &= P\big(\mathrm{lo}(i, t, \varphi) \wedge \bigwedge_{i \in J} \mathrm{lo}(j, t, \varphi), u_i(t+1), AP_i\big) \\
&= P\big(\mathrm{lo}(i, t, \varphi), u_i(t+1), AP_i\big) \wedge P\big(\bigwedge_{i \in J} \mathrm{lo}(j, t, \varphi), u_i(t+1), AP_i\big)
\end{aligned}
$$

where $\forall j \in J.\ \mathrm{send}(j,t,i) = \mathrm{true}$. The proof this case is just a combination of the proofs of the two previous cases.
- If $\mathrm{send}(i,t) = \mathrm{true}$ and $\forall j \in [1, |\mathcal{M}|] \setminus \{i\}.\ \mathrm{send}(j,t,i) = \mathrm{false}$. Then, we have: $\mathrm{lo}(i, t+1, \varphi) = \#$. The result holds vacuously.

$\square$

Back to the proof of Theorem 4. The remainder of the proof consists intuitively in showing that in a given architecture, we can take successively two components and merge them to obtain an equivalent architecture in the sense that they produce the same verdicts. The difference is that if in the merged architecture a verdict is emitted, then, in the non-merged architecture the same verdict will be produced with an additional delay.

Lemma 11. *In a two-component architecture, if in the centralised case a verdict is produced for some trace u, then, in the decentralised case, one of the monitor will produce the same verdict. Formally:*

$$\forall \varphi \in \text{LTL}. \forall u \in \Sigma^+. \ P(\varphi, u) = \top/\bot \Rightarrow \forall \sigma \in \Sigma^*. \exists i \in [1, 2]. \ \text{lo}(i, |u \cdot \sigma|, \varphi) = \top/\bot.$$

Proof. Let us consider a formula $\varphi \in \text{LTL}$ and a trace $u \in \Sigma^+$ s.t. $|u| = t$. We shall only consider the case where $P(\varphi, u) = \top$. The other case is symmetrical. Let us suppose that $\text{lo}(1, t, \varphi) \neq \top$ and $\text{lo}(2, t, \varphi) \neq \top$ (otherwise the results holds immediately). Because of the correctness of the algorithm (Theorem 3), we know that $\text{lo}(1, t, \varphi) \neq \bot$ and $\text{lo}(2, t, \varphi) \neq \bot$. Moreover, according to Lemma 10, we have necessarily that $\text{lo}(1, t, \varphi)$ and $\text{lo}(2, t, \varphi)$ are urgent formulas: $\Upsilon(\text{lo}(1, t, \varphi)) > 0$ and $\Upsilon(\text{lo}(2, t, \varphi)) > 0$. Since, there are only two components in the considered architecture, we have $\bigcup_{\varphi' \in \text{sus}(\text{lo}(1,t,\varphi))} \text{Prop}(\varphi') \subseteq AP_2$ and $\bigcup_{\varphi' \in \text{sus}(\text{lo}(2,t,\varphi))} \text{Prop}(\varphi') \subseteq AP_1$. According to Algorithm L, we have then $\text{send}(1, t-1, 2) = \text{true}$ and $\text{send}(2, t-1, \varphi) = \text{true}$. Then $\text{inlo}(1, t, \varphi) = \text{lo}(2, t-1, \varphi) \wedge \# = \text{lo}(2, t-1, \varphi)$. Hence: $\text{lo}(1, t, \varphi) = P(\text{lo}(2, t-1, \varphi), u_1(t), AP_1)$. According to Lemma 9 item 4, we have $\text{lo}(1, t, \varphi) = P(\text{rp}(\text{lo}(2, t-1, \varphi), u(0) \cdots u(t), 1), u_1(t), AP_1)$. Since

$$\bigcup_{\varphi' \in \text{sus}(\text{lo}(2,t,\varphi))} \text{Prop}(\varphi') \subseteq AP_1,$$

we have $\text{rp}\big(\text{lo}(2, t-1, \varphi), u(0) \cdots u(t), 1\big) = \text{rp}\big(\text{lo}(2, t-1, \varphi), u(0) \cdots u(t)\big)$. It follows that:

$$\begin{aligned}
\text{lo}(1, t, \varphi) &= P(\text{rp}(\text{lo}(2, t-1, \varphi), u(0) \cdots u(t)), u_1(t), AP_1) \\
&= P(P(\varphi, u(0) \cdots u(t)), u_1(t), AP_1) \qquad \text{(Lemma 10)} \\
&= P(\top, u_1(t), AP_1) = \top
\end{aligned}$$

Symmetrically, we can find that $\text{lo}(2, t, \varphi) = \top$. $\qquad \square$

Given two components C_1 and C_2 with two monitors attached M_1 and M_2 observing respectively two partial traces u_1 and u_2 of some global trace u. The alphabets of C_1 and C_2 are Σ_1 and Σ_2 respectively. Consider the architecture $\mathcal{C} = \{C_1, C_2\}$ with the set of monitors $\mathcal{M} = \{M_1, M_2\}$. Let us define the new component $\text{merge}(C_1, C_2)$ that produces events in $\Sigma_1 \cup \Sigma_2$. To the component $\text{merge}(C_1, C_2)$ is attached a monitor M observing events in the same alphabet. Now let us consider the architecture $\mathcal{C}' = \{\text{merge}(C_1, C_2)\}$ which is a one-component architecture with the set of monitors $\mathcal{M}' = \{\text{merge}(M_1, M_2)\}$.

We can parameterise the satisfaction relation of LTL formula according to the considered architecture. The relation $\models_D$ becomes $\models_D^{\mathcal{M}}$ where $\mathcal{M}$ is the considered architecture. The definition of $\models_D^{\mathcal{M}}$ is the same as the definition of $\models_D$ (Definition 6).

Lemma 12. *For a monitoring architecture $\mathcal{M} = \{M_1, M_2\}$ and the monitoring architecture $\mathcal{M}' = \{\text{merge}(M_1, M_2)\}$ where monitors of $\mathcal{M}$ have been merged, we have:*

$$\forall u \in \Sigma^+. \forall \varphi \in \text{LTL}. \ u \models_D^{\mathcal{M}} \varphi = \top/\bot \Rightarrow \forall \sigma \in \Sigma^+. \ u \cdot \sigma \models_D^{\mathcal{M}'} \varphi = \top/\bot.$$

Proof. This is a direct consequence of Lemma 11 and Corollary 1. Indeed, $\mathcal{M}'$ is a one-component architecture, thus $u \models_D^{\mathcal{M}'} \varphi = \top/\bot$ implies $u \models_3 \varphi = \top/\bot$, i.e., $P(\varphi, u) = \top/\bot$. Now, since $\mathcal{M}$ is a two-component architecture, using Lemma 11, for all $\sigma \in \Sigma$, there exists $i \in [1, |\mathcal{M}|]$ s.t. $\text{lo}(i, |u \cdot \sigma|, \varphi) = \top/\bot$. That is $u \cdot \sigma \models_D^{\mathcal{M}} \varphi = \bot/\top$. $\qquad \square$

The following lemma relates verdict production in a n-component architecture and in the same architecture where the two components with the lowest priority have been merged.

Lemma 13. *Let $\mathcal{M}$ be a n-component architecture, with $n \geq 2$ such that the priority between components is $M_1 < M_2 < \ldots < M_n$, i.e., M_1 and M_2 are the two components with the lowest priority[6]. Let us consider the architecture $\mathcal{M}' = \{\text{merge}(M_1, M_2), M_3, \ldots, M_n\}$, then we have:*

$$\forall u \in \Sigma^+. \forall \varphi \in \text{LTL}. \ u \models_D^{\mathcal{M}'} \varphi = \top/\bot \Rightarrow \forall \sigma \in \Sigma. \ u \cdot \sigma \models_D^{\mathcal{M}} = \top/\bot.$$

---

[6] Here, without loss of generality, we assume that monitors have been sorted according to their index. If this hypothesis does not hold initially, the indexes of components can be re-arranged so that this hypothesis holds.

Proof. We give a proof for the case where the verdict is $\top$ (the other case is symmetrical). Let us consider $u \in \Sigma^+, \varphi \in$ LTL s.t. $u \models_D^{\mathcal{M}'} \varphi = \top$. Let u' be the smallest prefix of u s.t. $P(\varphi, u') = \top$. From the theorem about the maximal delay (Theorem 2, we have that $|u| - |u'| \leq (n-1)$. Now each of the local obligations in the architecture $\mathcal{M}'$ will transit through at most n monitors following the ordering between components. That is, in the worst case (i.e., if a verdict is not produced before time $|u|$), any obligation will be progressed according to all components. More precisely, each time a local obligation is progressed on some component C_i, past obligations w.r.t. component C_i are removed (Lemma 9 - item 3). Using the compositionality of rp and the progression function on conjunction, in the worst case the local obligation at time $|u'| + n$ will be a conjunction of formulas of the form

$$
P(
$$
$$
\cdots
$$
$$
\cdots P(
$$
$$
P(\mathrm{rp}(\cdots \mathrm{rp}(\mathrm{rp}(\varphi, u', i), u', i_1) \cdots, u', i_n), u_i(|u'|), AP_i)
$$
$$
, u_{i_1}(|u'|+1, AP_{i_1}),
$$
$$
\cdots,
$$
$$
u_{i_n}(|u'|+n), AP_{i_n})
$$

where φ is a local obligation at time $|u'|$ and $\{i_1, \ldots, i_n\} \supseteq [1, |\mathcal{M}'|]$ (because of the ordering between components). Now according to Lemma 9 - item 5:

$$
\mathrm{rp}(\cdots \mathrm{rp}(\mathrm{rp}(\varphi, u', i), u', i_1) \cdots, i_n) = \mathrm{rp}(\varphi, u') = \top.
$$

Following the definition of the progression function for $\top$, we have that necessarily, the resulting local obligation at time $|u'| + n$ is $\top$. □

Lemma 14. *Let $\mathcal{M}$ be a n-component architecture, with $n \geq 2$ such that the priority between components is $M_1 < M_2 < \ldots < M_n$. Let us consider the architecture $\mathcal{M}' = \{\mathrm{merge}(M_n, \mathrm{merge}(\ldots, \mathrm{merge}(M_2, M_1))\}$, then we have:*

$$
\forall u \in \Sigma^+. \forall \varphi \in \mathrm{LTL}. u \models_D^{\mathcal{M}'} \varphi = \top/\bot \Rightarrow \forall u' \in \Sigma^+. |u'| \geq n \Rightarrow u \cdot u' \models_D^{\mathcal{M}} = \top/\bot.
$$

Proof. By an easy induction on the number of components merged using Lemma 13. □

Back to the proof of Theorem 4. Based on the previous results, we can easily show Theorem 4.

Proof. Let us consider an n-component architecture $\mathcal{M} = \{M_1, \ldots, M_n\}$, a trace $u \in \Sigma^+$ and a formula $\varphi \in$ LTL. Let us suppose that $u \models_3 \varphi = \top/\bot$. As the alphabets of monitors are respectively $\Sigma_1, \ldots \Sigma_n$ and each monitor M_i is observing a sub-trace u_i of u where the hypothesis about alphabets partitionning mentioned in Section 2 holds, we can consider the architecture $\mathcal{M}' = \{\mathrm{merge}(M_n, \mathrm{merge}(\ldots, \mathrm{merge}(M_2, M_1))\}$ where there is a unique monitor M observing the same trace u. Now, since $\mathcal{M}'$ is a one-component architecture, from $u \models_3 \varphi = \top/\bot$, by Corollary 1 we get $u \models_D \varphi = \top/\bot$. Using Lemma 13, we obtain that $\forall u' \in \Sigma^+. u \cdot u' \models_D^{\mathcal{M}} = \top/\bot$. □