Distributed monitoring of timed properties

Léo Henry[1], Thierry Jéron[2], Nicolas Markey[2], and Victor Roussanaly[3]

[1] University College London, UK
[2] IRISA, CNRS & Inria & Univ. Rennes, France
[3] Université de Lorraine, France

Abstract. In formal verification, monitoring monitoring consists of observing the execution of a system in order to decide as quickly as possible whether or not it satisfies a given property. We consider monitoring in a distributed setting, for properties given as reachability timed automata. In such a setting, the system is made of several components, each equipped with its own local clock and monitor. The monitors observe events occurring on their associated component, and receive timestamped events from other monitors through FIFO channels. Since clocks are local, they cannot be perfectly synchronized, resulting in imprecise timestamps. Consequently, they must be seen as intervals, leading monitors to consider possible reorderings of events. In this context, each monitor aims to provide, as early as possible, a verdict on the property it is monitoring, based on its potentially incomplete and imprecise knowledge of the current execution. In this paper, we propose an on-line monitoring algorithm for timed properties, robust to time imprecision and partial information from distant components. We first identify the date at which a monitor can safely compute a verdict based on received events. We then propose a monitoring algorithm that updates this date when new information arrives, maintains the current set of states in which the property can reside, and updates its verdict accordingly.

1 Introduction

Runtime verification. Formal verification is a branch of computer science that aims to check whether computer systems satisfy their requirements. This includes in particular model checking [10,3] and deductive verification [20,15], which work offline and check properties of all behaviours of the system under study. In contrast, runtime verification [23,4] is a set of efficient techniques to monitor the behaviour of a *running* system, and to detect, as early as possible during the execution, whether some properties are satisfied or violated. This domain of formal verification has been extensively studied over the last 25 years, and is now mature enough to be applied in various application domains.

As real-life systems typically comprise several connected components, runtime verification techniques must be adapted to handle situations where the behaviours of the components are only partially known and may be imprecise: each component is equipped with a monitor, and monitors exchange information asynchronously with each other.

Because the system is distributed, we assume that each monitor only has access to a local clock, which may slightly drift w.r.t a reference clock. Hence the dates of the events are only known with a limited precision, and the exact order of the events occurring on different subsystems may be unknown. We assume that communications between local monitors are FIFO; this way, when a monitor receives the information that some event took place on some subsystem, it also knows all the events that have occurred earlier on that subsystem.

Thanks to these hypotheses, each monitor can determine a time in the past for which it currently has enough information to decide whether the property it is checking already held, or failed to hold, at that time. We consider real-time properties modelled as *timed automata* (TA) [1]; the expressiveness of these models allows to account for the precise timing of events, which magnifies the impact of the lack of an observable global clock.

Example 1. Consider a private (*e.g.*, enterprise, university) network with a set of terminals and servers containing user account data logged in and a router connected to them. Properties of interest can be that (1) all machines should remain connected to the network and (2) no account should be logged in more than one terminal at a time. Even if two terminals signal that they are connected to a given account, a monitor has to wait to ensure that no "log out" message is pending from one of the two terminals. Plus, once such signals are received, the approximate timings may lead to situations where two terminals may or may not have been connected to the same user session at the same time, with no ability to conclude.

Our contributions. We present a monitoring algorithm for properties expressed as (deterministic) TAs in a distributed setting without a global clock. Each monitor keeps track of the most recent date at which it has collected the full history (relying on the assumption that communications with other monitors are FIFO). The prefix of the collected trace at this time already contains uncertainty, due to the absence of a global clock, which entails that reordering events from distinct components must be considered. The computation of a verdict is based on the incremental update of a structure that encodes the set of states compatible with the prefix of the collected trace up to the time of interest. We show that the monitoring algorithm is sound and complete.

Related works. This paper studies distributed runtime verification for timed properties described as timed automata [1]. Several related approaches have been developed in the literature. Extensions of Linear-time Temporal Logic (LTL) integrating dense time have been explored for runtime verification and monitor synthesis [6]. Among them, Metric Temporal Logic (MTL) is of particular interest as it is directly related to TAs [25] and is equipped with a progression function allowing to evaluate formulas at runtime [28]. Similarly, Timed Regular Expressions are as expressive as TAs and can be translated to TAs [2]. The tool Montre [29] monitors them using timed pattern matching. Monitoring TA models has been realized in the case of one-clock non-deterministic TAs [19]. Pinisetty *et al.* [26] introduce a predictive setting for runtime verification of timed properties, leveraging reachability analysis to anticipate the detection of verdicts.

The aforementioned approaches consider that the monitored system is *centralized* and the decision procedure is fed with a unique trace containing complete observations. *Decentralized* runtime verification [7] (see also a recent overview in [13]) handles separate traces corresponding to each monitor. Decentralized (also called synchronous) methods however assume the existence of a global clock shared by all components and monitors. We relax this assumption here, and consider *asynchronous distributed monitoring* (or *distributed monitoring* for short). For a discussion on the links and differences between synchronous and asynchronous methods, see [16].

Most approaches in decentralized runtime verification take as input Linear-time Temporal Logic formulas [7,11,17] or finite-state automata [14,12]. These approaches monitor specifications of discrete time, which does not account for the physical time that impacts the evaluation of the specifications nor the moment at which monitors perform their evaluation and deliver their verdicts. An approach close to ours is [5], in which properties are specified in an extension of Metric Temporal Logic to tackle both timing and data values. Similar to us, the authors also deal with out-of-order messages, but also failures and lost messages. However, they consider that local clocks are accurate.

Distributed runtime verification exhibits similarities with diagnosis [30,31] ([9] for TAs), which aims to identify the occurrence of a fault and the component(s) responsible for it after a finite number of discrete steps, and has to cope with partial observation. Our approach differs from diagnosis, as we assume that monitors'(combined) local information suffices to detect violations; diagnosis does not aim to check membership to an arbitrary timed regular expression. The approach of [21] for robust diagnosis shares several common aspects with ours. While centralized, it considers diagnosis where communication between the system and the diagnoser is subject to varying latency, clock drift and out-of-order observation. The problem is different but is similar in spirit: incrementally building a verdict based on approximate and partial timed observations. Moreover some constructions have clear similarities, which is not surprising: in both cases the language of runs compatible with the current partial and approximate observation has to be considered. Other approaches comparable to ours perform monitoring on timed properties in a decentralized [27] or distributed [18] fashion. Our own approach discusses distributed monitoring directly on timed automata models.

2 Preliminaries

We present the basic hypotheses about our formalization of the distributed monitoring problem in Section 2.1, the notions of words and languages at play in Sections 2.2 and 2.3, and timed automata in Section 2.4.

2.1 Distributed timed systems

We consider systems made of n independent components $(C_i)_{1 \leq i \leq n}$. Each C_i is observed by a local monitor M_i. The components being independent, we assume that each C_i has its own finite alphabet of actions Act_i, disjoint from the alphabets of the other components. We write $\mathsf{Act} = \bigsqcup_{1 \leq i \leq n} \mathsf{Act}_i$ for the alphabet of all actions. For an action $a \in \mathsf{Act}_i$, we write $\mathsf{Comp}(a) = i$. An action a fired by component C_i is observed and timestamped by its monitor M_i, giving rise to an *event* (a, t) in $\mathsf{Act}_i \times \mathbb{R}_{\geq 0}$. We assume the following about the system:

Respective knowledge: monitors know each others' alphabets of actions;

Communication: monitors send messages to their peers, carrying the timestamped events they observed (and events they learnt about from other monitors) in the order in which they observed them. We assume that local events can be strictly ordered.

Communication channels: communications between monitors obey a FIFO policy with no message loss: messages are received in the order they were sent, and any sent message is eventually received, although no upper bound on communication delays is assumed.

Communication topology: the connectivity is such that a monitor M_i can receive messages (either directly or indirectly) from any monitor M_j managing some action appearing in M_i's property.

Local liveness: at any given time, each monitor will eventually have a local observation in the future, and will eventually send it to the other monitors.

Time approximation: monitors do not share a global clock, but one can assume that each local clock has a maximal skew ε with respect to a global *reference clock*[4]. We suppose that clocks are non-decreasing.

Most of these assumptions are easy to satisfy. A FIFO policy can be achieved by numbering the events exchanged between one monitor and another, ensuring that a monitor can handle events in the order they were sent. Local liveness can be ensured by adding empty events that are sent when no events have been observed for a long duration. The main practical constraints are the absence of message losses, and the bound on clock skews. However, even these assumptions can be mitigated. Message losses can be detected using the message numbering, while a mechanism such as Network Time Protocol (NTP) can be used to limit the skew to ε.

We consider that each monitor is in charge of verifying some property (given as a timed automaton, see Sect. 2.4). As the monitoring algorithms and properties are symmetric for each such monitor, we restrict our dissertation to a fixed monitor M_i and assume that the entire set Act of actions is necessary for its property (some of these actions are still observed by other monitors).

2.2 Timed words and languages

We consider intervals in $\mathbb{R}$, denoted $\mathcal{I}(\mathbb{R})$. For an interval $I = \langle l, u \rangle$, with $\langle \in \{(, [\}$ and $\rangle \in \{),]\}$, we write $\mathsf{lb}(I) = l$, $\mathsf{ub}(I) = u$ for its lower and upper bounds. For two intervals I_1, I_2 of $\mathbb{R}$, we write $I_1 \prec I_2$ if $\mathsf{ub}(I_1) \leq \mathsf{lb}(I_2)$; when this

[4] Our approach can be easily generalized to different ε for each monitor, with our theorems depending on the greatest one.

condition is met, the intervals intersect in at most one point. We also consider intervals of natural numbers and write $[\![1;m]\!]$ for $\{k \in \mathbb{N} \mid 1 \leq k \leq m\}$.

Timed Words and Languages. We consider *timed words* built on the alphabet of actions Act as (finite or infinite) sequences $\sigma = (a_k, t_k)_{k \in [\![1;m]\!]}$[5] of *events* in $\mathsf{Act} \times \mathbb{R}_{\geq 0}$ whose sequence of *dates* $(t_k)_{k \in [\![1;m]\!]}$ is non-decreasing. We write $\sigma[k] = (a_k, t_k)$ for its k-th event. For any interval I of $\mathbb{R}_{\geq 0}$, we write $\sigma_{|I}$ for the subword of σ restricted to dates in I, and $\sigma_{|t}$ as a shorthand for $\sigma_{|[0,t]}$ with $t \in \mathbb{R}_{\geq 0}$. For a finite timed word $\sigma = (a_k, t_k)_{k \in [\![1;m]\!]}$, we write $|\sigma| = m$ for its length, $\mathsf{firstt}(\sigma) = t_1$ and $\mathsf{lastt}(\sigma) = t_m$ respectively for the dates of its first and last events; for the empty word γ, we let $\mathsf{firstt}(\gamma) = \mathsf{lastt}(\gamma) = 0$. Two timed words σ_1 and σ_2 can be concatenated into $\sigma_1 \cdot \sigma_2$ if, and only if, $\mathsf{lastt}(\sigma_1) \leq \mathsf{firstt}(\sigma_2)$. For any interval $I \subseteq \mathbb{R}_{\geq 0}$, we write $\mathsf{TT}_I(\mathsf{Act})$ for the set of timed words in $(\mathsf{Act} \times I)^*$, and $\mathsf{TT}(\mathsf{Act}) = \mathsf{TT}_{\mathbb{R}_{\geq 0}}(\mathsf{Act})$. A language of timed words is any subset of $\mathsf{TT}(\mathsf{Act})$.

For two languages of timed words L_1 and L_2, their concatenation $L_1 \cdot L_2$ is defined and equal to $\{\sigma_1 \cdot \sigma_2 \mid \sigma_1 \in L_1, \sigma_2 \in L_2\}$ if, and only if, $\mathsf{supp}\{\mathsf{lastt}(\sigma_1) \mid \sigma_1 \in L_1\} \leq \inf\{\mathsf{firstt}(\sigma_2) \mid \sigma_2 \in L_2\}$. The restriction to an interval naturally generalizes to languages: for a language $L \subseteq \mathsf{TT}(\mathsf{Act})$ and an interval I of $\mathbb{R}$, $L_{|I} = \{\sigma_{|I} \mid \sigma \in L\}$.

Projections on monitors. In our setting, Act is the disjoint union of alphabets Act_i. For a timed word σ, we write $\mathsf{p}_i(\sigma)$ for the projection on the actions monitored by M_i, defined by induction as $\mathsf{p}_i(\gamma) = \gamma$ for the empty word, and $\mathsf{p}_i(\sigma \cdot (a,t)) = \mathsf{p}_i(\sigma) \cdot (a,t)$ if $a \in \mathsf{Act}_i$ and $\mathsf{p}_i(\sigma \cdot (a,t)) = \mathsf{p}_i(\sigma)$ otherwise.

Conversely, we define a tensor operation on timed words $\sigma_1 \otimes \sigma_2$ that merges the events while re-ordering them by ascending timestamps. This operation is such that $\sigma = \otimes_{i \in [\![1;n]\!]} \mathsf{p}_i(\sigma)$.

2.3 Approximate Timed Words

If we had perfect clocks, timed words as defined above would be the model of choice for representing the knowledge of the monitors; restriction to intervals would be used to identify the part of the knowledge that each monitor knows is complete (as opposed to the part of the knowledge where informations from some of the components did not arrive yet).

In the context of distributed monitoring, we consider that the clocks of the monitors may be imprecise, resulting in a potential drift of up to a uniform bound ε w.r.t. a reference clock. Because of this, we have to rely on a notion of *approximate timed words*, and to define restriction to intervals for this new model.

Because of timing imprecisions, an event (a,t), made of action a timestamped with t by the monitor that observed it, may have happened anywhere in the interval $[t-\varepsilon, t+\varepsilon] \cap \mathbb{R}_{\geq 0}$ with respect to the reference clock[6]. Thus, while the information collected by a monitor has the form of a timed word $\sigma =$

[5] We abusively use such a notation for both finite and infinite sequences.
[6] Intersection with $\mathbb{R}_{\geq 0}$ is used to rule out events with negative dates.

$(a_k, t_k)_{k \in [\![1;m]\!]}$, the real date of each event (a_k, t_k) lies in the interval $I_k = [t_k - \varepsilon, t_k + \varepsilon] \cap \mathbb{R}_{\geq 0}$. We call *approximate timed word of* σ, the sequence $\nu(\sigma) = (a_k, I_k)_{k \in [\![1;m]\!]}$.

Approximate timed words. An *approximate timed word* (ATW for short) is a sequence of pairs $\nu = (a_k, I_k)_{k \in [\![1;m]\!]}$ such that, for all $k \in [\![1;m]\!]$, $a_k \in \mathsf{Act}$ and $I_k \in \mathcal{I}(\mathbb{R})$ is an interval (open or closed for generality). We denote by $\mathsf{ATW}(\mathsf{Act})$ the set of approximate timed words on Act.

With any approximate timed word, we associate two languages: its *ordered* language and its *non-ordered* language. Intuitively, the *ordered language* of ν is the language of timed words that respect the order of the events and the intervals given by ν, while the *non-ordered language* will be the union of the ordered languages for all possible reoderings of the events.

The ordered semantics defines a "tube" of timed words with the same untimed projection. It is defined as follows:

$$[\![(a_k, I_k)_{k \in [\![1;m]\!]}]\!]_{\mathsf{ord}} = \left\{(a_k, t_k)_{k \in [\![1;m]\!]} \in \mathsf{TT}(\mathsf{Act}) \mid \forall k \in [\![1;m]\!].\ t_k \in I_k\right\}$$

By definition of $\mathsf{TT}(\mathit{Act})$, the sequence $(t_k)_{k \in [\![1;m]\!]}$ is non-decreasing, which induces constraints on subsequent t_k's. Notice that $[\![\nu]\!]_{\mathsf{ord}} = \emptyset$ when $I_k = \emptyset$ for some k, or if no increasing sequences of dates can be found in the sequence $(I_k)_{k \in [\![1;m]\!]}$.

In order to define the *non-ordered language* of ν, we introduce the subset $\mathcal{F}(\nu)$ of permutations of events in ν that respect the strict order of events sharing the same component (each monitor knows the order of events occurring in the component it supervises). We define it as follows, with $\mathsf{Perms}([\![1;m]\!])$ being the set of permutations of $[\![1;m]\!]$:

$$\begin{aligned}\mathcal{F}((a_k, I_k)_{k \in [\![1;m]\!]}) = \big\{f \in \mathsf{Perms}([\![1;m]\!]) \mid \forall k, l \in [\![1;m]\!].\\ (\mathsf{Comp}(a_k) = \mathsf{Comp}(a_l) \wedge k < l) \Rightarrow f(k) < f(l)\big\}.\end{aligned}$$

Then for $f \in \mathcal{F}(\nu)$, we abuse the notation and write $f(\nu)$ for the ATW $(a_{f(k)}, I_{f(k)})_{k \in [\![1;m]\!]}$. Finally, we define the *(non-ordered) language* of ν as the set of timed words that respect both the intervals given by ν, and the strict local order on each component:

$$[\![\nu]\!] = \bigcup_{f \in \mathcal{F}(\nu)} [\![f(\nu)]\!]_{\mathsf{ord}}.$$

Intuitively, this language includes commutations of events that occurred at sufficiently close dates on different components: indeed, if $\nu = \nu_1 \cdot (b_k, t) \cdot (b_l, t') \cdot \nu_2$ with $\mathsf{Comp}(b_k) \neq \mathsf{Comp}(b_l)$ and $|t - t'| \leq 2\varepsilon$, then $[\![\nu]\!]$ contains timed words with form $\sigma_1 \cdot (b_k, t) \cdot (b_l, t') \cdot \sigma_2$ and $\sigma_1 \cdot (b_l, t' - \varepsilon) \cdot (b_k, t + \varepsilon) \cdot \sigma_2$.

Back to the monitoring problem, given some monitor and an observation prefix σ that is sufficient (in some sense clarified later), considering the imprecision due to the skew ε in the approximate timed word $\nu(\sigma)$, the non-ordered language $[\![\nu(\sigma)]\!]$ is the set of executions that could produce this observation prefix, hence that has to be considered for monitoring.

Operations on approximate timed words. We now focus on defining a restriction of approximate timed words to an interval of dates. This will be useful for the incremental update of the monitor's knowledge (see Sect. 3.3). Semantically, the restriction of an ATW ν to an interval I is the set of restrictions to I of all the timed words contained in $[\![\nu]\!]$. In this section, we present a syntactic definition, which will be the basis of an effective computation.

To this aim, we first define *intersection*: for $\nu = (a_k, I_k)_{k \in [\![1;m]\!]}$ and an interval I, the intersection of ν with I is the ATW $\nu_{\cap I} = (a_k, I_k \cap I)_{k \in [\![1;m]\!]}$. Notice that $[\![\nu_{\cap I}]\!]_{\mathsf{ord}} = [\![\nu_{\cap I}]\!] = \emptyset$ if $I_k \cap I = \emptyset$ for some $k \in [\![1;m]\!]$.

Our syntactic definition of restriction adapts the notion of subword. For any interval I of $\mathbb{R}_{\geq 0}$, and two ATW ν' and ν'', we say that $\nu' = \nu'_1 \cdots \nu'_n$ is a *subword of* $\nu'' = \nu''_1 \cdot \nu'_1 \cdots \nu''_n \cdot \nu'_n \cdot \nu''_{n+1}$ *conditioned by* I, written $\nu' \preceq_I \nu''$, if, and only if,

- for all $l \in [\![1; n+1]\!]$, for any (a_k, I_k) in ν''_l, $\neg(I_k \subseteq I)$: all events in ν''_l (which are dropped) *may occur outside of* I;
- for all $l \in [\![1; n]\!]$, for any $(a'_{k'}, I'_{k'})$ in ν'_l, $I'_{k'} \cap I \neq \emptyset$: all events in ν'_l (which are not dropped) *may occur in* I;
- there is $f \in \mathcal{F}(\nu'')$ s.t. $f(\nu'') = \nu_1 \cdot \nu' \cdot \nu_3$ for some ν_1 and ν_3: this encodes the fact that two events in the same component can not be permuted.

We can now define the (syntactic) *restriction* of an approximate timed word ν to an interval I as $\nu_{|I} = \{\nu'_{\cap I} \mid \nu' \preceq_I \nu\}$. As a shorthand, for a timestamp T, we write $\nu_{|[0,T]} = \nu_{|T}$. We overload the term *restriction* of ν to I because of the characterization of Lemma 1 below: the syntactic restriction corresponds to the semantic approach of taking the language of timed words associated with ν, and restricting each of its words. This provides us with a way of representing, manipulating and computing restriction of ATW to intervals:

Lemma 1. *For any approximate timed word ν and any timestamp T, it holds $\cup_{\nu' \in \nu_{|T}} [\![\nu']\!] = [\![\nu]\!]_{|T}$.*

Following this, we write $[\![\nu_{|T}]\!]$ for $\cup_{\nu' \in \nu_{|T}} [\![\nu']\!]$.

Example 2. Consider the approximate timed word $\nu = (a, [1, 3])(b, [2, 4])(c, [3, 5])$, and $I = [0, 3]$ with the components of the actions being pair-wise different. Then:

- $\nu_{\cap I}$ is the approximate timed word $(a, [1, 3])(b, [2, 3])(c, \{3\})$.
- assuming that all three events occur on different components (and can then be freely swapped), the set $\{\nu' \mid \nu' \preceq_I \nu\}$ is

$$\{\epsilon, (a, [1, 3]), (a, [1, 3])(b, [2, 4]), (a, [1, 3])(c, [3, 5]),$$
$$(a, [1, 3])(b, [2, 4])(c, [3, 5]), (b, [2, 4]), (b, [2, 4])(c, [3, 5]), (c, [3, 5])\}$$

  (in other terms, all subwords are allowed, since all intervals intersect I and none of them are included in I). It follows that $\nu_{|I}$ is the union of ϵ, $(a, [1, 3])$, $(a, [1, 3])(b, [2, 3])$, $(a, [1, 3])(c, [3, 3])$, $(a, [1, 3])(b, [2, 3])(c, [3, 3])$, $(b, [2, 3])$, $(b, [2, 3])(c, [3, 3])$ and $(c, [3, 3])$.

  Now, assume that b and c relate to the same component, so that they cannot be swapped. In this case, $(a, [1, 3])(c, [3, 5])$ is no longer a subword of ν for I, because the third condition is no longer fulfilled.

2.4 Formalism for timed systems

We monitor properties given as TAs, which are standard formalism for expressing properties of time-constrained systems: the aim of our monitoring procedure is to decide if the execution we are (partially and imprecisely) observing is (or will be) accepted by a given timed automaton. We introduce the formalism of timed automata in thie section.

Let X be a finite set of clocks. A clock valuation is a function $v\colon X \to \mathbb{R}_{\geq 0}$. We write $\mathcal{V}(X)$ for the set of valuations. The initial valuation is $\mathbf{0}\colon x \in X \mapsto 0$; a time elapse for a delay $t \in \mathbb{R}_{\geq 0}$ maps valuation v to $v + t\colon x \mapsto v(x) + t$, and a clock reset for a subset $X' \subseteq X$ maps v to $v_{[X']}$ such that $v_{[X']}(x) = v(x)$ if $x \notin X'$, and $v_{[X']}(x) = 0$ otherwise.

A *zone* is a finite conjunction of clock constraints of the forms $x_1 \bowtie n$ and $x_1 - x_2 \bowtie n$, with $x_1, x_2 \in X$, $\bowtie \in \{<, \leq, =, \geq, >\}$ and $n \in \mathbb{N}$. We write $\mathcal{Z}(X)$ for the set of zones of X. We write $v \models z$ and say that v *satisfies* z when the values of the clocks in v satisfy the constraints in z.

Definition 2. *A* timed automaton *(TA) is a tuple* $\mathcal{A} = (\mathcal{L}, \ell_{init}, \mathsf{Act}, X, E)$ *where* $\mathcal{L}$ *is a finite set of* locations *containing the initial location* ℓ_{init}*,* Act *and* X *are finite sets of actions and clocks respectively, and* $E \subseteq \mathcal{L} \times \mathcal{Z}(X) \times \mathsf{Act} \times 2^X \times \mathcal{L}$ *is the set of* transitions*. For a transition* $(\ell, g, a, z, \ell') \in E$*, we call* ℓ *its* source*,* ℓ' *its* target*,* g *and* a *its* guard *and* action *and* z *its* reset set*. We call* configurations of $\mathcal{A}$ *the triples* $(\ell, v, t) \in \mathcal{L} \times \mathcal{V} \times \mathbb{R}_{\geq 0}$*, and* initial configuration *the configuration* $(\ell_{init}, \mathbf{0}, 0)$*.*

Remark 1. We add to the usual notion of configuration a date representing the instant at which the system is in this configuration (after a given behaviour). We do this because our reasoning is based on timestamps, not delays, following the definition of timed words based on observations. This can be readily implemented by giving an additional clock (which is never reset) to the TA.

A timed word $\sigma = (a_i, t_i)_{1 \leq i}$ is *a trace* of a timed automaton if, starting from the initial configuration $(\ell_{init}, \mathbf{0}, 0)$, one can find a sequence $\rho = ((\ell_{i-1}, v_{i-1}, t_{i-1}) \xrightarrow{\delta_i} (\ell_{i-1}, v_{i-1} + \delta_i, t_{i-1} + \delta_i) \xrightarrow{e_i} (\ell_i, v_i, t_{i-1} + \delta_i))_{1 \leq i}$ with $e_i = (\ell_{i-1}, g_i, a_i, z_i, \ell_i) \in E$, $\delta_i = t_i - t_{i-1}$ (with $t_0 = 0$), and at each step $v_{i-1} + \delta_i \models g_i$ and $v_i = (v_{i-1} + \delta_i)_{[z_i]}$. Such a sequence ρ is called a (finite or infinite) *run*, and we write $trace(\rho) = \sigma$. A timed word σ is a *partial* trace if there exists a timed word σ' such that $\sigma' \cdot \sigma$ is a trace. A partial trace thus corresponds to a *partial run* that does not (necessarily) start from $(\ell_{init}, \mathbf{0}, 0)$. For a (partial) trace σ with $\mathsf{firstt}(\sigma) \geq t$, we write $(\ell, v, t) \overset{\sigma}{\underset{t}{\rightsquigarrow}} (\ell', v', t')$ when σ leads from configuration (ℓ, v, t) to configuration (ℓ', v', t').

A timed automaton is said *deterministic* if for any two transitions (ℓ, g, a, z, ℓ') and (ℓ, g', a, z', ℓ') such that $g \cap g' \neq \emptyset$, it holds $g = g'$ and $z = z'$. It is said *complete* when for any configuration (ℓ, v, t) and action $a \in \mathsf{Act}$, there is at least a transition (ℓ, g, a, z, ℓ') such that $v \models g$. When considering complete deterministic automata, the *trace* function defined above is a bijection, and we identify traces with their associated runs. In this context, we add the possibility of having a final delay after a trace: for a configuration (ℓ, v, t), a date t'' and a (partial) trace σ such that $\mathsf{firstt}(\sigma) \geq t$ and $\mathsf{lastt}(\sigma) = t' \leq t''$ we write $(\ell, v, t)\ \mathsf{after}_t^{t''}\ \sigma$ for the unique configuration (ℓ', v'', t'') such that $(\ell, v, t) \overset{\sigma}{\underset{t}{\rightsquigarrow}} (\ell', v', t') \xrightarrow{t''-t'} (\ell', v'', t'')$, *i.e.*, reached from (ℓ, v, t) after the trace σ followed by the delay $t'' - t'$. This can be generalized to sets of configurations and languages: for a set of configurations S and a language of timed words L, such that $\inf\{\mathsf{firstt}(\sigma) \mid \sigma \in L\} \geq t$ and $\mathsf{supp}\{\mathsf{lastt}(\sigma) \mid \sigma \in L\} \leq t''$, $S\ \mathsf{after}_t^{t''}\ L = \{(\ell', v', t'') \mid \exists (\ell, v, t) \in S, \exists \sigma \in L, (\ell', v', t'') = (\ell, v, t)\ \mathsf{after}_t^{t''}\ \sigma\}$. Finally, we define the corresponding notion when no date is given: $S\ \mathsf{after}\ L = \{(\ell', v', t') \mid \exists (\ell, v, t) \in S, \exists \sigma \in L, \exists t', (\ell', v', t') = (\ell, v, t)\ \mathsf{after}_t^{t'}\ \sigma\}$. Notice that in this last definition, t and t' are not bound and can range on all values such that $\mathsf{after}_t^{t'}$ is defined.

Modelling properties. The property associated with monitor M_i is defined by a deterministic and complete TA $\mathcal{A}_i$ and a subset $\mathcal{F}_i$ of locations specifying a reachability property[7]: we write $[\![\mathcal{A}_i, \mathcal{F}_i]\!]$ for the set of (runs of) finite traces σ that end in some location of $\mathcal{F}_i$ when applied to $\mathcal{A}_i$ from its initial configuration $(\ell_{init}, \mathbf{0}, 0)$; we extend $[\![\mathcal{A}_i, \mathcal{F}_i]\!]$ (abusively keeping the same notation) to include runs of infinite traces σ for which there is a length k such that all prefixes of σ of length larger than k are in $[\![\mathcal{A}_i, \mathcal{F}_i]\!]$.

Given a property specified by $\mathcal{A}_i$ and $\mathcal{F}_i$, a finite trace σ is a *good prefix* (resp. *bad prefix*) if for all *infinite* continuations $\sigma \cdot \sigma' \in (\mathsf{Act} \times \mathbb{R}_{\geq 0})^\omega$ of σ, $\sigma \cdot \sigma' \in [\![\mathcal{A}_i, \mathcal{F}_i]\!]$ (resp. $\sigma \cdot \sigma' \notin [\![\mathcal{A}_i, \mathcal{F}_i]\!]$). In terms of automata, this means that the prefix reached some configuration in $\mathcal{L}_i \times \mathcal{V} \times \mathbb{R}_{\geq 0}$ from which it will always eventually stay in $\mathcal{F}_i \times \mathcal{V} \times \mathbb{R}_{\geq 0}$ (resp. it never visited and will never visit $\mathcal{F}_i$). We note this set of configurations $\mathit{Inev}(\mathcal{F}_i)$ (resp. $\mathit{Never}(\mathcal{F}_i)$). Good prefixes (resp. bad prefixes) are then traces of runs in $[\![\mathcal{A}_i, \mathit{Inev}(\mathcal{F}_i)]\!]$ (resp. $[\![\mathcal{A}_i, \mathit{Never}(\mathcal{F}_i)]\!]$). Starting from $\mathcal{F}_i$, the state sets $\mathit{Inev}(\mathcal{F}_i)$ and $\mathit{Never}(\mathcal{F}_i)$ can be computed off-line by a zone-based co-reachability analysis [8]. Thanks to this, we restrict our focus to the reachability of pairs of locations and zones without loss of generality. These notions can be extended to languages, thus to approximate timed words. A language $L \in \mathsf{TT}_{\mathsf{Act}}$ is a *good* (resp. *bad*) *language prefix* if $L \subseteq [\![\mathcal{A}_i, \mathit{Inev}(\mathcal{F}_i)]\!]$ (resp. $L \subseteq [\![\mathcal{A}_i, \mathit{Never}(\mathcal{F}_i)]\!]$). These can also be computed using $\mathit{Inev}(\mathcal{F}_i)$ and $\mathit{Never}(\mathcal{F}_i)$.

3 Monitoring with complete information

The role of monitoring algorithms is to provide us with verdicts when analyzing executions of the system. Since we want this to be performed online, verdicts should be given as soon as possible, based on the observation of a finite execution prefix. However, in the context of distributed systems, the observation collected by a monitor at a given date may be imperfect, with missing events and approximate

[7] We here restrict to deterministic and complete TAs for simplicity, but generalization to non-deterministic and incomplete TAs is easy.

dates. We first identify the points in time where we have enough information to decide a verdict in Sect. 3.1, and then define our verdicts of interest in Sect. 3.2. Using this, we explain the data structure we use and its related operations in Sect. 3.3, and explain how to compute a verdict in Sect. 3.4.

3.1 Point of Certainty

When the components of the system perform actions, their corresponding monitors $(M_j)_{j \in [\![1;n]\!]}$ instantly *observe* these actions and timestamp them with the value of their (local) clock. However, they need to wait for the communication of other monitors in order to *collect* the information about the other components' events.

Communication policy. We consider the simple policy in which each monitor M_j instantly sends its observations (action performed and timestamp) to every other monitor M_i that needs it for checking its property, grouping in the same message all the events that occurred at the same instant[8]. As there are no bounds in communication delays, monitors still have to deal with partial and out-of-order information. Moreover, local time approximation induces imprecision in event dates. To make monitoring sound, we first determine the time point at which we can safely monitor with no missing event.

Formally, consider the *global observation* σ^o such that at a given global time t^g, the observation collected by all monitors is hence $\sigma^o_{|t^g}$. We know that the *global trace* σ^g of the run of the system is such that $\sigma^g \in [\![\nu(\sigma^o)]\!]$. This trace can not be observed, yet it is the one we want to monitor, hence we will start our reasoning from (prefixes of) the language $[\![\nu(\sigma^o)]\!]$.

Moreover, no single monitor has access to $\sigma^o_{|t^g}$ at time t^g due to the need for synchronization. Let the *collected trace* at t by M_i, written $\sigma_i(t)$, be the monitoring information gathered by a monitor M_i at the M_i-local time t. It is composed of a subset of the global observation $\sigma^o_{|t+\varepsilon}$, containing at least its own local observation $\mathsf{p}_i(\sigma^o)_{|t}$ and events received from other monitors, forming for each monitor M_j a timed word $\mathsf{p}_j(\sigma^o)_{|t_j}$ with $t_j \leq t + 2\varepsilon$. Indeed, communication being FIFO, M_i receives the information from each individual M_j in order, but potentially with M_j-local timestamps up to $t + 2\varepsilon$, as both the M_i-local time t and the M_j-local time can skew by a maximum of ε from the global time.

For M_i at M_i-local date t, consider the set

$$\{(j, t_j) \in [\![1;n]\!] \times \mathbb{R}_{\geq 0} \mid j \neq i \wedge \big(\mathsf{lastt}(\mathsf{p}_j(\sigma_i(t))) = t_j\big)\} \cup \{(i, t)\}$$

of pairs made of the index j of each monitor coupled with the timestamp of the last observation received from M_j by M_i. Let $(\mathsf{jtmin}_k)_{k \in [\![1;n]\!]} = (\mathsf{jmin}_k, \mathsf{tmin}_k)_{k \in [\![1;n]\!]}$ be the sequence obtained by ordering this set of pairs by ascending timestamp [9].

[8] This technical detail is useful for Prop. 3 to ensure that all events of same date issued from the same component are collected simultaneously. It can be implemented by waiting any non null delay before sending a message aggregating the observations.

[9] We should write $(\mathsf{jtmin}_k(t))_{k \in [\![1;n]\!]} = (\mathsf{jmin}_k(t), \mathsf{tmin}_k(t))_{k \in [\![1;n]\!]}$, *i.e.*, parametrize by t, but we will often forget t when clear from the context.

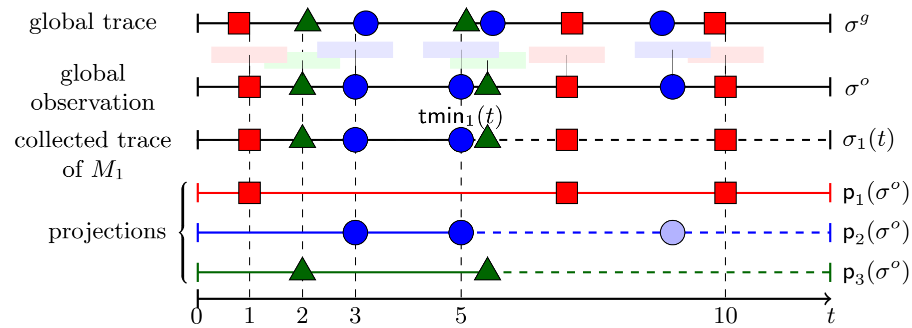

Fig. 1: A global finite trace σ^g (top), its corresponding global observation σ^o at local M_i-time t (below) with rectangles figuring global time approximation ($\varepsilon = 0.7$), the collected trace $\sigma_1(t) = (a, 1)(c, 2)(b, 3)(b, 5)(c, 5.5)(a, 7)(a, 10)$ (middle), *i.e.*, observation of M_1 completed with some events received from M_2 and M_3, the projections observed locally by three monitors M_1, M_2, M_3 at t (bottom). Dashed lines represent information uncertainty, *e.g.*, M_1 ignores what happened after the last event received from each of the other monitors.

Initially, those timestamps are all 0, and any order may be chosen. Then, this sequence with its ordering can be easily maintained on-the-fly when new events are observed or received by M_i. Clearly, $(\mathsf{jmin}_1, \mathsf{tmin}_1)$ identifies the monitor M_j for which M_i is aware of the earliest timestamp (in its local time). In the absence of a skew, M_i would be sure to have complete information from all other monitors at time tmin_1, with jmin_1 being the monitor for which the last event known by M_i (if any) is the oldest. However, as time is approximated, and since verdicts should be given on global traces, knowing all events at local times up to tmin_1 only certifies that all events have been recorded for global time up to $\mathsf{tmin}_1 - \varepsilon$, as seen in Ex. 3. Notice that this also entails that the verdict can be given at t only if $t \geq \varepsilon$.

Example 3. In Fig. 1, for monitor M_1, $\mathsf{jmin}_1 = 2$ at time t. Yet, if the verdict was given with respect to observations after the last event from M_2, the last event from M_3, which happened before it but was marked with a later timestamp, would be missed. The verdict should thus restrict to the earliest possible global date for the last event of M_2, namely $\mathsf{tmin}_1 - \varepsilon$.

Conversely, we are sure that for all monitors, we collected some event with timestamp at least tmin_1. As local times are non-decreasing and communications are FIFO, no monitor can send a new observation with global time below $\mathsf{tmin}_1 - \varepsilon$. Thus, we know that all events of global time below $\mathsf{tmin}_1 - \varepsilon$ have already been collected by M_i. The set of possible traces of the system corresponding to that observation is $\nu(\sigma_i(t))_{|\mathsf{tmin}_1 - \varepsilon}$. This is the purpose of the first part of the following proposition. The second part claims that, indeed, the observation at $\mathsf{tmin}_1(t) - \varepsilon$ is in the set of "tubes" of possible observations of σ^g restricted to $\mathsf{tmin}_1(t) - \varepsilon$.

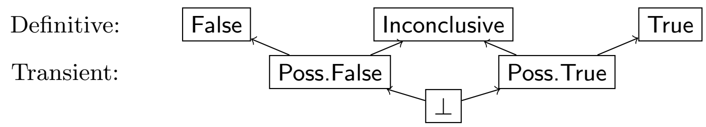

Fig. 2: Hasse diagram of the verdict preorder

Proposition 3. *For any monitor M_i, at any local time $t \geq \varepsilon$, $\sigma^g_{|\mathsf{tmin}_1(t)-\varepsilon}$ belongs to $[\![\nu(\sigma_i(t))_{|\mathsf{tmin}_1(t)-\varepsilon}]\!]$. Similarly, $\sigma_i(t)_{|\mathsf{tmin}_1(t)-\varepsilon}$ belongs to $[\![\nu(\sigma^g)_{|\mathsf{tmin}_1(t)-\varepsilon}]\!]$.*

3.2 Verdicts at $\mathsf{tmin}_1 - \varepsilon$

As demonstrated by Prop. 3, at global time $\mathsf{tmin}_1 - \varepsilon$ the global trace $\sigma^g_{|\mathsf{tmin}_1-\varepsilon}$ necessarily belongs to the language of traces compatible with the collected trace for M_i at local time t, restricted to $\mathsf{tmin}_1 - \varepsilon$. Consequently, this is a sufficient language to ensure safe monitoring, in the sense that a *definitive* verdict (True, False or Inconclusive) for this language built from a current observed trace cannot be changed by future observations.

To go a step further, we add *transient* verdicts to the definitive verdicts: Poss.True (resp. Poss.False) means that the property can not be falsified (resp. verified), although it is still possible to reach an Inconclusive verdict instead of a True (resp. False) one. The induced pre-order of verdicts $\lesssim$ is displayed in Fig. 2. The *verdict function* $V(t)$ then links at each date t the compatible traces $[\![\nu(\sigma_i(t))_{|\mathsf{tmin}_1(t)-\varepsilon}]\!]$ with the property specified by $\mathcal{A}_i$. Formally $V(t)$ is

- True when $[\![\nu(\sigma_i(t))_{|\mathsf{tmin}_1(t)-\varepsilon}]\!]$ is a good language prefix;
- False when $[\![\nu(\sigma_i(t))_{|\mathsf{tmin}_1(t)-\varepsilon}]\!]$ is a bad language prefix;
- Inconclusive when $[\![\nu(\sigma_i(t))_{|\mathsf{tmin}_1(t)-\varepsilon}]\!]$ intersects both languages $[\![\mathcal{A}_i, \mathit{Inev}(\mathcal{F}_i)]\!]$ and $[\![\mathcal{A}_i, \mathit{Never}(\mathcal{F}_i)]\!]$;
- Poss.True when $[\![\nu(\sigma_i(t))_{|\mathsf{tmin}_1(t)-\varepsilon}]\!]$ intersects $[\![\mathcal{A}_i, \mathit{Inev}(\mathcal{F}_i)]\!]$ but no higher verdict (True or Inconclusive) applies;
- Poss.False when $[\![\nu(\sigma_i(t))_{|\mathsf{tmin}_1(t)-\varepsilon}]\!]$ intersects $[\![\mathcal{A}_i, \mathit{Never}(\mathcal{F}_i)]\!]$ but no higher verdict (False or Inconclusive) applies;
- $\perp$ by default, if none of these conditions holds.

Because properties are specified by complete TAs, all traces and their continuations are traces of the TA, and since good/bad prefixes are closed under continuation, verdicts can only progress to higher verdicts in the above pre-order:

Lemma 4. *For a fixed monitor M_i with a property $(\mathcal{A}_i, \mathcal{F}_i)$, for any $t' > t$, we have $V(t) \lesssim V(t')$.*

In particular, *definitive* verdicts remain true eternally as soon as they hold. We say that σ is ε*-conclusive* when $\nu(\sigma)$ yields a definitive non-Inconclusive verdict (recall that $\nu(\sigma)$ depends on ε for the size of its intervals). Equivalently:

Definition 5. *For a fixed monitor M_i with a property $(\mathcal{A}_i, \mathcal{F}_i)$ and a skew ε, we say that a timed word σ is* ε-conclusive *when $[\![\nu(\sigma)]\!]$ is either a good or a bad language prefix.*

3.3 Data structure

So far, we have defined verdicts with respect to the language $[\![\nu(\sigma_i(t))_{|\mathsf{tmin}_1(t)-\varepsilon}]\!]$. However, in order to incrementally compute verdicts when new observations arrive, it will be more adequate to manipulate sets of configurations reached after this language. The next proposition justifies the adequacy of this approach.

Proposition 6. *The verdict $V(t)$ at M_i-local date t can be computed by checking whether the set of configurations $(\ell_{init}, \mathbf{0}, 0)$ $\mathsf{after}_0^{\mathsf{tmin}_1(t)-\varepsilon}$ $[\![\nu(\sigma_i(t))_{|\mathsf{tmin}_1(t)-\varepsilon}]\!]$ intersects or is included in $Inev(\mathcal{F}_i)$ or $Never(\mathcal{F}_i)$.*

In the following, we proceed in steps to define the computation of the above set of configurations: first, we decompose the computation on *unordered* languages into computations on *ordered ones*, where permutations of events are fixed (Prop. 8). We encode them in a data structure $\mathsf{R}(t)$ that represents both the set of configurations reached after some trace in $[\![\nu(\sigma_i(t))_{|\mathsf{tmin}_1(t)-\varepsilon}]\!]$, and the remaining events necessary for incremental computation. Then, we intersect the sets of configurations with the precise time of interest ($\mathsf{tmin}_1 - \varepsilon$) to obtain the equality with the above set (Prop. 10). Finally, we show how to incrementally update the proposed data structure when new observations are collected.

Decomposition of $\{(\ell_{init}, \mathbf{0}, 0)\}$ $\mathsf{after}[\![\nu(\sigma_i(t))_{|\mathsf{tmin}_1(t)-\varepsilon}]\!]$. The separation of an unordered language into a union of ordered ones requires considering permutations of the restriction $\nu(\sigma_i(t))_{|\mathsf{tmin}_1(t)-\varepsilon}$. For this, we define the *decomposition* of an ATW ν at some date T, as the set of all possible prefixes of $\nu_{|T}$ paired with the set of events that are yet to be accounted for, as this set will be necessary to incrementally update the data structure.

Definition 7. *The* decomposition of $\nu \in \mathsf{ATW}(\mathsf{Act})$ at global time T *is*

$$\mathcal{D}(\nu, T) = \{(f(\nu_1); \nu_2) \mid \nu = \nu_1 \otimes \nu_2 \wedge \nu_1 \preceq_{[0,T]} \nu \wedge f \in \mathcal{F}(\nu_1)\}.$$

For a pair $(\nu_1; \nu_2)$ in $\mathcal{D}(\nu, T)$, ν_1 corresponds to a permutation of an element in $\nu_{|T}$, while ν_2 lists the remaining events that are not taken into account in ν_1. We can then use the decomposition to express the unordered language of a restriction as a union of ordered ones as follows.

Proposition 8. $[\![\nu_{|T}]\!] = \bigcup_{(\nu_1;\nu_2)\in\mathcal{D}(\nu,T)} [\![\nu_{1\cap[0,T]}]\!]_{ord}$

We are now ready to define our data structure $\mathsf{R}(t)$ (we should write $\mathsf{R}_i(t)$ but the monitor is clear from the context), which encodes the configurations reached through each ν_1 in the decomposition, associated with its remainder ν_2:

Definition 9. *For a monitor M_i and a local time t, we define $\mathsf{R}(t)$, as*

$$\mathsf{R}(t) = \Big\{(\{(\ell_{init}, \mathbf{0}, 0)\}\ \mathsf{after}[\![\nu_1]\!]_{ord}; \nu_2) \;\Big|\; (\nu_1; \nu_2) \in \mathcal{D}(\nu(\sigma_i(t)), \mathsf{tmin}_1(t) - \varepsilon)\Big\}.$$

The set $\mathsf{R}(t)$ represents a set of configurations in which the system can be after some sequence of events in $\nu(\sigma_i(t))_{|\mathsf{tmin}_1(t)-\varepsilon}$. A restriction to dates before $\mathsf{tmin}_1(t) - \varepsilon$ is still necessary (see the intersection with $[0, T]$ in Prop. 8). In order to add this constraint, we call *state of* $\mathsf{R}(t)$ the set of configurations $\mathsf{state}(\mathsf{R}(t)) = \bigcup_{(S,\nu_2)\in\mathsf{R}(t)} \{(\ell, v, t) \in S \mid t = \mathsf{tmin}_1(t) - \varepsilon\}$ and get the desired equality:

Proposition 10. $\mathit{state}(R(t)) = \{(\ell_{init}, \mathbf{0}, 0)\}\ \mathit{after}_0^{\mathit{tmin}_1(t)-\varepsilon}\ [\![\nu(\sigma_i(t))_{|\mathit{tmin}_1(t)-\varepsilon}]\!]$

Example 4. Let us consider the example from Fig. 1 with the collected trace at time $t = 10$ being $\sigma_i(t) = (a, 1)(c, 2)(b, 3)(b, 5)(c, 5.5)(a, 7)(a, 10)$ and the skew $\varepsilon = 0.7$. We have $\mathsf{tmin}_1 - \varepsilon = 4.8$. We first have to consider the set C of all possible configurations reached by an interleaving of the first three observations $(a, 1)(c, 2)(b, 3)$. Then, we have to consider every case for the events $(b, 5)(c, 5.5)$ that can occur before or after $\mathsf{tmin}_1 - \varepsilon$. $\mathsf{R}(10)$ is then composed of the following elements, for each $c \in C$:

- $(c; \nu((b, 5)(c, 5.5)(a, 7)(a, 10)))$, meaning that all events occur after time 4.8;
- $(c\ \mathsf{after}\ [\![(b, [4.3, 5.7])]\!]_{\mathsf{ord}}; \nu((c, 5.5)(a, 7)(a, 10)))$, considering that the event b happened before time 4.8;
- $(c\ \mathsf{after}\ [\![(c, [4.8, 6.2])]\!]_{\mathsf{ord}}; \nu((b, 5)(a, 7)(a, 10)))$, considering that c occured before time 4.8.
- the two elements $(c\ \mathsf{after}\ [\![(b, [4.3, 5.7])(c, [4.8, 6.2])]\!]_{\mathsf{ord}}; \nu((a, 7)(a, 10)))$ and $(c\ \mathsf{after}\ [\![(c, [4.8, 6.2])(b, [4.3, 5.7])]\!]_{\mathsf{ord}}; \nu((a, 7)(a, 10)))$ considering that both b and c occurred before time 4.8, thus both orderings should be considered.

Note that $\mathsf{R}(10)$ contains configurations that can only be reached if the collected events occur after $\mathsf{tmin}_1 - \varepsilon$, but they do not appear in $\mathsf{state}(\mathsf{R}(10))$, since we only consider configurations that are reached at time $\mathsf{tmin}_1 - \varepsilon$, meaning that the events leading to these configurations must have occurred before this time.

Updates of R. When time passes, new events may be collected. If they do not change tmin_1, the only updates to R is their addition to ν_2. If tmin_1 changes at $t' > t$ because of newly collected events ν', then the combination of $\mathsf{R}(t)$ and ν' contains all the necessary information to update R, as each element of $\mathsf{R}(t)$ encodes all of $\nu(\sigma_i(t))_{|\mathsf{tmin}_1(t)-\varepsilon}$. Thus, updating the structure is only a matter of selecting, for each $(S, \nu_2) \in \mathsf{R}(t)$, the possible sub-words of $\nu_2 \cdot \nu'$ to apply after S.

Proposition 11. *Let* $t' \geq t$ *and* ν' *be the sequence of events received in the interval* $(t, t']$ *(*i.e.*, such that* $\sigma_i(t') = \sigma_i(t) \otimes \nu'$*), then*

$$R(t') = \left\{(S\ \mathit{after}\ [\![\nu'_1]\!]_{\mathit{ord}}; \nu'_2) \;\middle|\; (S; \nu_2) \in R(t),\ (\nu'_1, \nu'_2) \in \mathcal{D}(\nu_2 \otimes \nu', \mathit{tmin}_1(t') - \varepsilon)\right\}.$$

Intuitively, for each pair (S, ν_2) in $\mathsf{R}(t)$, the extension of ν_2 with the newly collected events ν' is decomposed at $\mathsf{tmin}_1(t') - \varepsilon$. For each possible element (ν'_1, ν'_2) in this decomposition, $\mathsf{R}(t')$ builds the pair made of the set S after $[\![\nu'_1]\!]_{\mathsf{ord}}$ associated with the remainder ν'_2. We call $\mathit{next}(\mathsf{R}(t), t')$ the function that computes $\mathsf{R}(t')$ at the time t' when $\mathsf{tmin}_1(.)$ changes according to Prop. 11. We now have a data structure that can be used to compute the set of configurations needed to infer verdicts (Prop. 10) and can be updated incrementally based on the new collected observations and $\mathsf{tmin}_1(\cdot) - \varepsilon$ (Prop. 11).

3.4 Monitoring at $\mathsf{tmin}_1(.) - \varepsilon$

The previous discussions lead to the monitoring algorithm presented in Algorithm 1. It uses a triple of boolean values (I, N, C) encoding the intersection of $\mathsf{state}(\mathsf{R})$ respectively with $\mathit{Inev}(\mathcal{F}_i)$, $\mathit{Never}(\mathcal{F}_i)$ and the complement of their union. The algorithm starts with R being the initial state with no remaining events, the minimal time for monitoring $\mathsf{tmin} = \varepsilon$, jtmin initially set to $(k, 0)_{k\in[\![1;n]\!]}$. Each new collected sequence of observations from a monitor M_j (recall that monitors group all events with same date in a unique message) is added to all continuations ν in R, and jtmin is updated. If tmin_1 has changed, tmin and R are updated (as discussed above). The update of (I, N, C) determines the verdict which is returned if definitive (a lazy evaluation of (I, N, C) optimizes the update, I and N being non-decreasing, while C is non-increasing). We do not detail here the communication of verdicts between monitors which could help anticipate their termination.

Algorithm 1: The monitor M_i's algorithm to monitor at $\mathsf{tmin}_1(.) - \varepsilon$.

Init: $\mathsf{R} = \{((\ell_{init}, \mathbf{0}, 0); [])\}$; $\mathsf{tmin} = \varepsilon$; $\mathsf{jtmin} = (k, 0)_{k\in[\![1;n]\!]}$; $\mathsf{verdict}(i) := \bot$;
$(I, N, C) := ((\ell_{init}, \mathbf{0}, 0) \in \mathit{Inev}(\mathcal{F}_i), (\ell_{init}, \mathbf{0}, 0) \in \mathit{Never}(\mathcal{F}_i), \neg(I \vee N))$;

while *True* **do**
  Receive *sequence* $(a_1, t_a) \dots (a_n, t_a)$ *from monitor* $M_j, j \in [\![1; n]\!]$;
  $\mathsf{R} := \{(S, \nu \otimes \nu((a_1, t_a) \dots (a_n, t_a))) \mid (S, \nu) \in \mathsf{R}\}$;
  update jtmin ;
  if $\mathit{tmin}_1 > \mathit{tmin}$ **then**
    $\mathsf{tmin} := \mathsf{tmin}_1$;
    $\mathsf{R} := \mathit{next}(\mathsf{R}, \mathsf{tmin}_1 - \varepsilon)$;
    $I := I \vee (\mathsf{state}(\mathsf{R}) \cap \mathit{Inev}(\mathcal{F}_i) \neq \emptyset)$;
    $N := N \vee (\mathsf{state}(\mathsf{R}) \cap \mathit{Never}(\mathcal{F}_i) \neq \emptyset)$;
    $C := C \wedge (\mathsf{state}(\mathsf{R}) \cap \overline{\mathit{Inev}(\mathcal{F}_i) \cup \mathit{Never}(\mathcal{F}_i)}) \neq \emptyset$;
    switch (I, N, C) **do**
      case $(1, 1, *)$ **do** return (verdict (i):=Inconclusive);
      case $(1, 0, 0)$ **do** return (verdict (i):=True) ;
      case $(0, 1, 0)$ **do** return (verdict (i):=False);
      case $(1, 0, 1)$ **do** verdict (i):=Poss.True;
      case $(0, 1, 1)$ **do** verdict (i):=Poss.False;

Proposition 12. *Algorithm 1 sets the verdict to* $V(t)$.

We can prove the following soundness and completeness of the monitoring algorithm. Notice that completeness is limited to 2ε-conclusive executions.

Theorem 13. *Monitoring at* $\mathsf{tmin}_1(.) - \varepsilon$ *is sound and complete where, for any local monitor* M_i *and its property* $(\mathcal{A}_i, \mathcal{F}_i)$*:*

soundness *means that for any global trace* $\sigma^g \in \mathsf{TT}(\mathsf{Act})$ *produced at date* $T \in \mathbb{R}_{\geq 0}$*, if* $\mathsf{verdict}(i) = \mathsf{True}$ *at time* T*, then* σ^g *is a good prefix of* $(\mathcal{A}_i, \mathcal{F}_i)$

(respectively, if $\mathsf{verdict}(i) = \mathsf{False}$ *at time* T*, then* σ^g *is a bad prefix of* $(\mathcal{A}_i, \mathcal{F}_i)$*). Furthermore, if* $\mathsf{verdict}(i) = \mathsf{Inconclusive}$*, then neither* σ^g *nor its possible continuations are* 2ε*-conclusive on* $(\mathcal{A}_i, \mathcal{F}_i)$*.*

completeness *means that for any global trace* $\sigma^g \in \mathsf{TT}(\mathsf{Act})$*, if a prefix of* σ^g *is* 2ε*-conclusive and good (respectively, bad) on* $(\mathcal{A}_i, \mathcal{F}_i)$ *then there exists some date* $T \in \mathbb{R}_{\geq 0}$ *such that* $\mathsf{verdict}(i) = \mathsf{True}$ *(respectively,* $\mathsf{verdict}(i) = \mathsf{False}$*).*

4 Conclusion

This paper presents a distributed approach to monitor properties specified as deterministic timed automata when faced with approximation on events dates. The approach relies on the identification of the point in time at which sufficient information has been gathered by the local monitor to compute a verdict and the incremental computation of the set of states of the property compatible with the collected observation at this point in time This requires the careful account of potential permutations of events emanating from distant components.

This method allows to apply monitoring on complex systems that are *distributed* in space and whose behaviours *depend strongly on time*, further increasing the reach of this popular runtime verification method. It is interesting to notice the timely nature of this contribution, as the interest for distributed systems is developing not only in verification (see related works) but also in model learning ([24,22] discrete time), which could soon allow to automatically generate models of systems and specifications, allowing—at longer term—for fully black-box distributed tools requiring no expert knowledge.

While we ensured the soundness and completeness of our algorithm, its efficiency still needs to be experimented. Future works should include the implementation and test of this algorithm against realistic models and properties. This implementation could be then compared to the one presented in [27] that allows decentralized monitoring of regular timed expressions (but without clock skew), which can be generated from a timed automaton. Additionally, our algorithm could be extended in several ways. First, with additional hypotheses (*e.g.*, maximal throughput of components), we could issue verdicts earlier by anticipating the occurrence of events. This would require an extension of the structure R, adding a level of uncertainty. The balance between the gain of anticipation and the cost of updating this structure would certainly be an issue and requires experimental tuning. Using similar techniques, we believe we can handle properties defined by non-deterministic timed automata, at least for one-clock automata [19]. Finally, we can also try to reduce the communication overhead. Indeed, we assumed that all the local observations are forwarded to every other monitor, which can be improved in several ways, depending on the system topology.

Acknowledgements. This work was partially supported by the EPSRC Standard Grant CLeVer (EP/S028641/1).

References

1. Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
2. Eugene Asarin, Paul Caspi, and Oded Maler. Timed regular expressions. *J. ACM*, 49(2):172–206, 2002.
3. Christel Baier and Joost-Pieter Katoen. *Principles of Model-Checking.* MIT Press, May 2008.
4. Ezio Bartocci and Yliès Falcone, editors. *Lectures on Runtime Verification*, volume 10457 of *Lecture Notes in Computer Science*. Springer-Verlag, 2018.
5. David Basin, Felix Klaedtke, and Eugen Zălinescu. Runtime verification over out-of-order streams. *ACM Transactions on Computational Logic*, 21(1):1–43, October 2019.
6. Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for LTL and TLTL. *ACM Trans. Softw. Eng. Methodol.*, 20(4):14:1–14:64, September 2011.
7. Andreas Klaus Bauer and Yliès Falcone. Decentralised LTL monitoring. In Dimitra Giannakopoulou and Dominique Méry, editors, *FM 2012: Formal Methods - 18th International Symposium, Paris, France, August 27-31, 2012. Proceedings*, volume 7436 of *Lecture Notes in Computer Science*, pages 85–100. Springer, 2012.
8. Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Lectures on Concurrency and Petri Nets*, volume 2098 of *Lecture Notes in Computer Science*, pages 87–124. Springer-Verlag, 2004.
9. Patricia Bouyer, Léo Henry, Samy Jaziri, Thierry Jéron, and Nicolas Markey. Diagnosing timed automata using timed markings. *Int. J. Softw. Tools Technol. Transf.*, 23(2):229–253, 2021.
10. Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem. *Handbook of Model Checking.* Springer-Verlag, April 2018.
11. Christian Colombo and Yliès Falcone. Organising LTL monitors over distributed systems with a global clock. *Formal Methods Syst. Des.*, 49(1-2):109–158, 2016.
12. Antoine El-Hokayem and Yliès Falcone. On the monitoring of decentralized specifications: Semantics, properties, analysis, and simulation. *ACM Trans. Softw. Eng. Methodol.*, 29(1):1:1–1:57, 2020.
13. Yliès Falcone. On decentralized monitoring. In Ayoub Nouri, Weimin Wu, Kamel Barkaoui, and ZhiWu Li, editors, *Verification and Evaluation of Computer and Communication Systems - 15th International Conference, VECoS 2021, Virtual Event, November 22-23, 2021, Revised Selected Papers*, volume 13187 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2021.
14. Yliès Falcone, Tom Cornebize, and Jean-Claude Fernandez. Efficient and generalized decentralized monitoring of regular languages. In Erika Ábrahám and Catuscia Palamidessi, editors, *Formal Techniques for Distributed Objects, Components, and Systems - 34th IFIP WG 6.1 International Conference, FORTE 2014, Held as Part of the 9th International Federated Conference on Distributed Computing Techniques, DisCoTec 2014, Berlin, Germany, June 3-5, 2014. Proceedings*, volume 8461 of *Lecture Notes in Computer Science*, pages 66–83. Springer, 2014.
15. Jean-Christophe Filliâtre. Deductive software verification. *International Journal on Software Tools for Technology Transfer*, 13(5):397–403, October 2011.
16. Adrian Francalanza, Jorge A. Pérez, and César Sánchez. Runtime verification for decentralised and distributed systems. In Ezio Bartocci and Yliès Falcone, editors,

Lectures on Runtime Verification - Introductory and Advanced Topics, volume 10457 of *Lecture Notes in Computer Science*, pages 176–210. Springer, 2018.
17. Florian Gallay and Yliès Falcone. Decentralized LTL enforcement. In Pierre Ganty and Davide Bresolin, editors, *Proceedings 12th International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2021, Padua, Italy, 20-22 September 2021*, volume 346 of *EPTCS*, pages 135–151, 2021.
18. Ritam Ganguly, Yingjie Xue, Aaron Jonckheere, Parker Ljung, Benjamin Schornstein, Borzoo Bonakdarpour, and Maurice Herlihy. Distributed runtime verification of metric temporal properties. *Journal of Parallel and Distributed Computing*, 185:104801, 2024.
19. Alejandro Grez, Filip Mazowiecki, Michal Pilipczuk, Gabriele Puppis, and Cristian Riveros. The monitoring problem for timed automata. *CoRR*, abs/2002.07049, 2020.
20. Charles Antony Richard Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, October 1969.
21. Maximilian A. Köhl and Holger Hermanns. Model-based diagnosis of real-time systems: Robustness against varying latency, clock drift, and out-of-order observations. *ACM Transactions on Embedded Computing Systems*, 22(4):68:1–68:48, 2023.
22. Faezeh Labbaf, Jan Friso Groote, Hossein Hojjat, and Mohammad Reza Mousavi. Compositional learning for interleaving parallel automata. In Orna Kupferman and Pawel Sobocinski, editors, *Foundations of Software Science and Computation Structures - 26th International Conference, FoSSaCS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings*, volume 13992 of *Lecture Notes in Computer Science*, pages 413–435. Springer, 2023.
23. Martin Leucker and Christian Schallart. A brief account of runtime verification. *Journal of Logic and Algebraic Programming*, 78(5):293–303, May 2009.
24. Thomas Neele and Matteo Sammartino. Compositional automata learning of synchronous systems. In *International Conference on Fundamental Approaches to Software Engineering*, pages 47–66. Springer Nature Switzerland, 2023.
25. Dejan Nickovic and Oded Maler. AMT: a property-based monitoring tool for analog systems. In Jean-François Raskin and P. S. Thiagarajan, editors, *Proceedings of the 5th International Conference on Formal modeling and analysis of timed systems (FORMATS 2007)*, volume 4763 of *Lecture Notes in Computer Science*, pages 304–319. Springer-Verlag, 2007.
26. Srinivas Pinisetty, Thierry Jéron, Stavros Tripakis, Yliès Falcone, Hervé Marchand, and Viorel Preoteasa. Predictive runtime verification of timed properties. *J. Syst. Softw.*, 132:353–365, 2017.
27. Victor Roussanaly and Yliès Falcone. Decentralised runtime verification of timed regular expressions. In Alexander Artikis, Roberto Posenato, and Stefano Tonetta, editors, *29th International Symposium on Temporal Representation and Reasoning, TIME 2022, November 7-9, 2022, Virtual Conference*, volume 247 of *LIPIcs*, pages 6:1–6:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
28. Prasanna Thati and Grigore Rosu. Monitoring algorithms for metric temporal logic specifications. *Electronic Notes in Theoretical Computer Science*, 113:145–162, 2005.
29. Dogan Ulus. Montre: A tool for monitoring timed regular expressions. In Rupak Majumdar and Viktor Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, volume 10426 of *Lecture Notes in Computer Science*, pages 329–335. Springer, 2017.

30. Yin Wang, Tae-Sic Yoo, and Stéphane Lafortune. Diagnosis of discrete event systems using decentralized architectures. *Discret. Event Dyn. Syst.*, 17(2):233–263, 2007.
31. Xiang Yin and Stéphane Lafortune. Codiagnosability and coobservability under dynamic observations: Transformation and verification. *Autom.*, 61:241–252, 2015.

A Proof of Section 2

Lemma 1. *For any approximate timed word ν and any timestamp T, it holds* $\cup_{\nu' \in \nu_{|T}} [\![\nu']\!] = [\![\nu]\!]_{|T}$.

Proof. We write $I = [0, T]$ Consider $\sigma_2 \in [\![\nu]\!]_{|I}$. By definition σ_2 is a sub-word of a word σ in $[\![\nu]\!]$ with all events dated outside of I cut out. Formally, there is a permutation $f \in \mathcal{F}(\nu)$ and $\sigma = \sigma_2 \cdot \sigma_3 \in [\![f(\nu)]\!]_{\mathsf{ord}}$. We write $\nu_1 \cdot \nu_2 \cdot \nu_3$ for the corresponding decomposition of $f(\nu)$.

As all dates in σ_2 are in I, $I_k \cap I \neq \emptyset$ for all (a_k, I_k) in ν_2. Similarly, as all dates in σ_3 are outside of I, $\neg(I_k \subseteq I)$ for all (a_k, I_k) in ν_3. Furthermore, if you consider $f^{-1}_{|\nu_2}$ the restriction of f^{-1} to the indices of ν_2, $f^{-1}_{|\nu_2} \in \mathcal{F}(\nu_2)$ (because the constraints arising from components in ν_2 are the same in as a subword of ν and as a word in itself). Thus $f^{-1}(\nu_2)$ is a subword of ν such that $f^{-1}(\nu_2) \preceq_I \nu$. Moreover, clearly $\sigma_2 \in [\![(f^{-1}(\nu_2))_{\cap I}]\!]$. This proves that $[\![\nu]\!]_{|I} \subseteq \cup_{\nu' \in \nu_{|I}} [\![\nu']\!]$.

Conversely, consider $\nu' \in \nu_{|I}$ and $\sigma_2 \in [\![\nu']\!]$. By definition of $[\![.]\!]$, there is a permutation $f' \in \mathcal{F}(\nu')$ s.t. $\sigma_2 \in [\![f'(\nu')]\!]_{\mathsf{ord}}$. By definition of $\nu_{|I}$, there is $f \in \mathcal{F}(\nu)$ such that $f(\nu) = \nu_2 \cdot \nu_3$ with $\nu' = \nu_{2 \cap I}$. The constraints linked to the components are the same in ν' and ν_2, as they share the same untimed projection. Thus $f' \in \mathcal{F}(\nu_2)$ and we can construct $f'' \in \mathcal{F}(\nu)$ such that $f''(\nu) = f(\nu_2) \cdot \nu_3$.

We then extend σ_2 into $\sigma = \sigma_2 \cdot \sigma_3 \in [\![f''(\nu)]\!]_{\mathsf{ord}}$ such that $\sigma_{|I} = \sigma_2$. This is possible because all dates in σ_2 are by definition in I and for all intervals I_k in ν_3, $\neg(I_k \subseteq I)$. Thanks to this σ we know that $\sigma_2 \in [\![\nu]\!]_{|I}$. This proves that $\cup_{\nu' \in \nu_{|I}} [\![\nu']\!] \subseteq [\![\nu]\!]_{|I}$. □

B Proof of Section 3

B.1 Update of jtmin

Algorithm 2: Macro for updating jtmin when receiving a sequence of events with same date $(a_1, t_a) \dots (a_n, t_a)$ from M_j

Let k be such that $\mathsf{jmin}_k = j$;
foreach (a, t_a) *in* $(a_1, t_a) \dots (a_n, t_a)$ **do**
    while $k < n - 1 \wedge \mathit{tmin}_{k+1} < t_a$ **do**
        $\mathsf{jtmin}_k := \mathsf{jtmin}_{k+1}$;
        $k := k + 1$
    $\mathsf{jtmin}_k := (j, t_a)$

B.2 Proof of Section 3.1

Proposition 3. *For any monitor M_i, at any local time $t \geq \varepsilon$, $\sigma^g_{|tmin_1(t)-\varepsilon}$ belongs to $[\![\nu(\sigma_i(t))_{|tmin_1(t)-\varepsilon}]\!]$. Similarly, $\sigma_i(t)_{|tmin_1(t)-\varepsilon}$ belongs to $[\![\nu(\sigma^g)_{|tmin_1(t)-\varepsilon}]\!]$.*

Proof. At any M_i-local time $t > \varepsilon$, by definition of $\mathsf{tmin}_1(t)$ and because all simultaneous observations from a given monitor are sent simultaneously, M_i can only collect or observe events with a date $t' > \mathsf{tmin}_1(t)$. Because of this, for any $T \leq \mathsf{tmin}_1(t)$, $\sigma_i(t)_{|T} = \sigma^o_{|T}$ (recall that σ^o is the global observation, *i.e.*, all events equipped with their local times). As the skew is bounded to ε at any time, we know that any event with a *global timing* lesser than or equal to $\mathsf{tmin}_1(t) - \varepsilon$ appears in $\sigma^o_{|\mathsf{tmin}_1(t)}$, together with (potentially) some events with a larger global timing. Hence, $\sigma^g_{|\mathsf{tmin}_1(t)-\varepsilon} \in [\![\nu(\sigma^o)]\!]_{|\mathsf{tmin}_1(t)-\varepsilon} = [\![\nu(\sigma_i(t))]\!]_{|\mathsf{tmin}_1(t)-\varepsilon}$. By Lemma 1, we then have that $\sigma^g_{|\mathsf{tmin}_1(t)-\varepsilon} \in [\![\nu(\sigma_i(t))_{|\mathsf{tmin}_1(t)-\varepsilon}]\!]$.

Similarly, for any local time T, because the skew is bounded to ε, we know that $\sigma^o \in [\![\nu(\sigma^g)]\!]$ and, by Lemma 1, $\sigma^o_{|T} \in [\![\nu(\sigma^g)_{|T}]\!]$. In particular, at any local time $t > \varepsilon$, $\sigma^o_{|\mathsf{tmin}_1(t)-\varepsilon} \in [\![\nu(\sigma^g)_{|\mathsf{tmin}_1(t)-\varepsilon}]\!]$. As we know that $\sigma^o_{|\mathsf{tmin}_1(t)-\varepsilon} = \sigma(i)_{|\mathsf{tmin}_1(t)-\varepsilon}$, we have our result. □

B.3 Proof of Section 3.2

Lemma 4. *For a fixed monitor M_i with a property $(\mathcal{A}_i, \mathcal{F}_i)$, for any $t' > t$, we have $V(t) \lesssim V(t')$.*

Proof. This lemma relies on the underlying fact that $\mathcal{A}_i$ is complete and $Inev(\mathcal{F}_i)$ and $Never(\mathcal{F}_i)$ are attractor sets (as clear per their definition). Hence, all continuations of traces that exist up to a time t exist up to any greater time $t' > t$ (by completeness), and if the trace at time t is in one of these sets, all of its continuations are too.

Keeping this in mind, we make the proof verdict by verdict. For $\bot$, the result is trivial, as any other verdict is greater.

When $V(t) = \mathsf{Poss.True}$ holds, $[\![\nu(\sigma_i(t))_{|\mathsf{tmin}_1(t)-\varepsilon}]\!] \cap [\![\mathcal{A}_i, Inev(\mathcal{F}_i)]\!] \neq \emptyset$. From the above remark, we know that for any $t' > t$, $[\![\nu(\sigma_i(t'))_{|\mathsf{tmin}_1(t')-\varepsilon}]\!] \cap [\![\mathcal{A}_i, Inev(\mathcal{F}_i)]\!] \neq \emptyset$. Thus the three only possible verdicts are Poss.True, True and Inconclusive, which are all greater than Poss.True in $\lesssim$.

The result for $V(t) = \mathsf{Poss.False}$ follows the same ideas for $Never(\mathcal{F}_i)$ instead of $Inev(\mathcal{F}_i)$.

The result for $V(t) = \mathsf{Inconclusive}$ follows the same logic by applying the above idea to *both* $Never(\mathcal{F}_i)$ and $Inev(\mathcal{F}_i)$, as there is no other verdict than Inconclusive that allows both intersections to be non-empty.

For $V(t) = \mathsf{True}$, we know that $[\![\nu(\sigma_i(t))_{|\mathsf{tmin}_1(t)-\varepsilon}]\!] \subseteq [\![\mathcal{A}_i, Inev(\mathcal{F}_i)]\!]$. From the above remark, we know that for any $t' > t$, $[\![\nu(\sigma_i(t'))_{|\mathsf{tmin}_1(t')-\varepsilon}]\!] \subseteq [\![\mathcal{A}_i, Inev(\mathcal{F}_i)]\!]$. It follows that $V(t') = \mathsf{True}$. Finally, $V(t) = \mathsf{False}$ follows the same idea for $Never(\mathcal{F}_i)$ instead of $Inev(\mathcal{F}_i)$.

B.4 Proof of Section 3.3

Proposition 6. *The verdict $V(t)$ at M_i-local date t can be computed by checking whether the set of configurations $(\ell_{init}, \mathbf{0}, 0)$ $\mathit{after}_0^{\mathit{tmin}_1(t)-\varepsilon}$ $[\![\nu(\sigma_i(t))_{|\mathit{tmin}_1(t)-\varepsilon}]\!]$ intersects or is included in $\mathit{Inev}(\mathcal{F}_i)$ or $\mathit{Never}(\mathcal{F}_i)$.*

Proof. By definition, a language is a good (resp. bad) language prefix if and only if all of its words are in $[\![\mathcal{A}_i, \mathit{Inev}(\mathcal{F}_i)]\!]$ (resp. $[\![\mathcal{A}_i, \mathit{Never}(\mathcal{F}_i)]\!]$). It is then enough to notice, for all verdicts, that the language $[\![\mathcal{A}_i, \mathit{Inev}(\mathcal{F}_i)]\!]$ (resp. $[\![\mathcal{A}_i, \mathit{Never}(\mathcal{F}_i)]\!]$) is the set of words whose runs end in $\mathit{Inev}(\mathcal{F}_i)$ (resp. $\mathit{Never}(\mathcal{F}_i)$), which is exactly what is computed by after. Following the definition of $V(t)$ and letting $Q = (\ell_{init}, \mathbf{0}, 0)\ \mathsf{after}_0^{\mathsf{tmin}_1(t)-\varepsilon}\ [\![\nu(\sigma_i(t))_{|\mathsf{tmin}_1(t)-\varepsilon}]\!]$ it is then easy to check the following. $V(t) =$ True (resp. False) if $Q \subseteq \mathit{Inev}(\mathcal{F}_i)$ (resp. $Q \subseteq \mathit{Never}(\mathcal{F}_i)$), $V(t) =$ Inconclusive when Q intersects both $\mathit{Inev}(\mathcal{F}_i)$ and $\mathit{Never}(\mathcal{F}_i)$, $V(t) =$ Poss.True (resp. Poss.False) if Q intersects $\mathit{Inev}(\mathcal{F}_i)$ (resp. $V(t) \neq$ False and Q intersects $\mathit{Never}(\mathcal{F}_i)$) but not greater verdict applies, and $V(t) = \bot$ if no other verdict applies. □

Proposition 8. $[\![\nu_{|T}]\!] = \bigcup_{(\nu_1;\nu_2)\in\mathcal{D}(\nu,T)} [\![\nu_{1\cap[0,T]}]\!]_{\mathit{ord}}$

To prove this property, we first state and prove the following technical lemma rewriting the permutations of a restriction $\nu_{|T}$ as functions of the ν_1 in $\mathcal{D}(\nu_T)$.

Lemma 14. *For an approximate timed word ν and a time T, $\{f(\nu') \mid \nu' \in \nu_{|T},\ f \in \mathcal{F}(\nu')\} = \{\nu_{1\cap[0,T]} \mid (\nu_1, \nu_2) \in \mathcal{D}(\nu, T)\}$*

Proof. This can be shown by expending the definition of $\nu_{|T}$ in both sides. On the left-hand side: $\{f(\nu') \mid \nu' \in \nu_{|T},\ f \in \mathcal{F}(\nu')\} = \{f(\nu') \mid \nu' = \nu''_{\cap[0,T]} \wedge \nu'' \preceq_{[0,T]} \nu\}$. We can see that this corresponds exactly to the definition of the $\nu_{1\cap[0,T]}$ in $\mathcal{D}(\nu, T)$. □

Proof (Proposition 8). By definition of $\nu_{|T}$ and the unordered language: $[\![\nu_{|T}]\!] = \bigcup_{\nu'\in\nu_{|T}} [\![\nu']\!] = \bigcup_{\nu'\in\nu_{|T}} \bigcup_{f\in\mathcal{F}(\nu)} [\![\nu']\!]_{\mathsf{ord}}$. By Lemma 14 (and the linearity of ordered languages), $\bigcup_{\nu'\in\nu_{|T}} \bigcup_{f\in\mathcal{F}(\nu)} [\![\nu']\!]_{\mathsf{ord}} = \bigcup_{(\nu_1,\nu_2)\in\mathcal{D}(\nu,T)} [\![\nu_{1\cap[0,T]}]\!]_{\mathsf{ord}}$. With all of these equalities together we have our result. □

Proposition 10. $\mathit{state}(R(t)) = \{(\ell_{init}, \mathbf{0}, 0)\}\ \mathit{after}_0^{\mathit{tmin}_1(t)-\varepsilon}\ [\![\nu(\sigma_i(t))_{|\mathit{tmin}_1(t)-\varepsilon}]\!]$

Proof. By Proposition 8 and the linearity of $[\![\cdot]\!]_{\mathsf{ord}}$ we have:

$$\{(\ell_{init}, \mathbf{0}, 0)\}\ \mathsf{after}_0^{\mathsf{tmin}_1(t)-\varepsilon}\ [\![\nu(\sigma_i(t))_{|\mathsf{tmin}_1(t)-\varepsilon}]\!] =$$
$$\{(\ell_{init}, \mathbf{0}, 0)\}\ \mathsf{after}_0^{\mathsf{tmin}_1(t)-\varepsilon}\ \Big(\bigcup_{(\nu_1,\nu_2)\in\mathcal{D}(\nu(\sigma_i(t)),\mathsf{tmin}_1(t)-\varepsilon)} [\![\nu_{1\cap[0,\mathsf{tmin}_1(t)-\varepsilon]}]\!]_{\mathsf{ord}}\Big)$$

By definition, writing $S_{\mathsf{tmin}_1(t)-\varepsilon} = \{(\ell, v, t) \in S \mid t = \mathsf{tmin}_1(t) - \varepsilon\}$ for any set of configurations S:

$$\begin{aligned}\mathsf{state}(\mathsf{R}(t)) &= \bigcup_{(S,\nu)\in\mathsf{R}(t)} S_{\mathsf{tmin}_1(t)-\varepsilon}\\ &= \bigcup_{(\nu_1,\nu_2)\in\mathcal{D}(\nu(\sigma_i(t)),\mathsf{tmin}_1(t)-\varepsilon)} (\{(\ell_{init},\mathbf{0},0)\}\ \mathsf{after}[\![\nu_1]\!]_{\mathsf{ord}})_{\mathsf{tmin}_1(t)-\varepsilon}\\ &= \bigcup_{(\nu_1,\nu_2)\in\mathcal{D}(\nu(\sigma_i(t)),\mathsf{tmin}_1(t)-\varepsilon)} \{(\ell_{init},\mathbf{0},0)\}\ \mathsf{after}_0^{\mathsf{tmin}_1(t)-\varepsilon}[\![\nu_1]\!]_{\mathsf{ord}}\ .\end{aligned}$$

The last equation being true by definition of the after operator. Hence, to obtain the desired equality, it suffices to show that

$$\{(\ell_{init},\mathbf{0},0)\}\ \mathsf{after}_0^{\mathsf{tmin}_1(t)-\varepsilon}(L\cup L') = \\ (\{(\ell_{init},\mathbf{0},0)\}\ \mathsf{after}_0^{\mathsf{tmin}_1(t)-\varepsilon}L)\cup(\{(\ell_{init},\mathbf{0},0)\}\ \mathsf{after}_0^{\mathsf{tmin}_1(t)-\varepsilon}L')$$

which is clear by double inclusion. □

Proposition 11. *Let $t'\geq t$ and ν' be the sequence of events received in the interval $(t,t']$ (*i.e.*, such that $\sigma_i(t')=\sigma_i(t)\otimes\nu'$), then*

$$R(t')=\big\{(S\ \mathit{after}\ [\![\nu'_1]\!]_{ord};\nu'_2)\ \big|\ (S;\nu_2)\in R(t),\ (\nu'_1,\nu'_2)\in\mathcal{D}(\nu_2\otimes\nu',\mathit{tmin}_1(t')-\varepsilon)\big\}.$$

We first state a useful lemma that will be re-used in the main proof. It exhibits the linearity of the after operation.

Lemma 15. *Consider an approximate timed word $\nu\cdot\nu'$ and the sets of configurations $S=\{(\ell_{init},\mathbf{0},0)\}$ after $[\![\nu]\!]_{ord}$ and $S'=S$ after $[\![\nu']\!]_{ord}$. Then $S'=\{(\ell_{init},\mathbf{0},0)\}$ after $[\![\nu\cdot\nu']\!]_{ord}$*

Proof. By definition of S', a configuration $c'\in S'$ can be reached from a configuration $c\in S$ through a timed word $\sigma'\in[\![\nu']\!]_{\mathsf{ord}}$ starting at the date t associated with c. In the same way, by definition of S, c can be reached from the initial configuration through some timed word $\sigma\in[\![\nu]\!]_{\mathsf{ord}}$. By combining both, we obtain that $\{(\ell_{init},\mathbf{0},0)\}\overset{\sigma}{\underset{0}{\rightsquigarrow}}c\overset{\sigma'}{\underset{t}{\rightsquigarrow}}c'$. Thus $S'\subseteq\{(\ell_{init},\mathbf{0},0)\}\mathsf{after}[\![\nu\cdot\nu']\!]_{\mathsf{ord}}$.

Conversely, $c'\in\{(\ell_{init},\mathbf{0},0)\}\,\mathsf{after}\,[\![\nu\cdot\nu']\!]_{\mathsf{ord}}$ can be reached from the initial configuration by a timed word in $[\![\nu\cdot\nu']\!]_{\mathsf{ord}}$. Because we are considering the ordered language, we can cut this timed word as $\sigma\cdot\sigma'$ with $\sigma\in[\![\nu]\!]_{\mathsf{ord}}$ and $\sigma'\in[\![\nu']\!]_{\mathsf{ord}}$. The configuration c reached after σ is in c and by decomposing the run we obtain the other inequality. □

Proof (Proposition 11). We make the proof by double inclusion.

$\supseteq$: Consider an element of the right-hand side ($S'=S$ after $[\![\nu'_1]\!]_{\mathsf{ord}},\nu'_2$). S is in $\mathsf{state}(\mathsf{R}(t))$, so by definition of R, $S=\{(\ell_{init},\mathbf{0},0)$ after $[\![\nu_1]\!]_{\mathsf{ord}}\}$ for $(\nu_1,\nu_2)\in\mathcal{D}\left(\nu(\sigma_i(t)),\mathsf{tmin}_1(t)-\varepsilon\right)$. By Lemma 15 we have that $S'=(\ell_{init},\mathbf{0},0)$ after $[\![\nu_1\cdot\nu'_1]\!]_{\mathsf{ord}}$. We also have that $\nu(\sigma_i(t'))_{|\mathsf{tmin}_1(t')-\varepsilon}=(\nu_1\cdot\nu'_1)\otimes\nu'_2$. We now show that $\nu_1 \cdot \nu'_1 \preceq_{[0,\mathsf{tmin}_1(t')-\varepsilon]} \nu(\sigma_i(t'))_{|\mathsf{tmin}_1(t')-\varepsilon}$. First, notice that any elements of $\nu(\sigma_i(t'))_{|\mathsf{tmin}_1(t')-\varepsilon}$ not in $\nu_1 \cdot \nu'_1$ is in ν'_2. Hence for any such (a_k, I_k), $\neg(I_k \subseteq [0, \mathsf{tmin}_1(t') - \varepsilon])$ by definition of ν'_2. Second, for any element (a_k, I_k) of $\nu_1 \cdot \nu'_1$, $I_k \cap [0, \mathsf{tmin}_1(t') - \varepsilon] \neq \emptyset$ (we even have a stronger condition for elements of ν_1). Finally, by definition of ν_1, there is a permutation $f \in \mathcal{F}(\nu(\sigma_i(t)))$ such that $f(\nu(\sigma_i(t))) = \nu_1 \cdot \nu_2$. Because no observation can be added before $\mathsf{tmin}_1(t) - \varepsilon$, this permutation can be extended to $\nu(\sigma_i(t'))$ so that $f(\nu(\sigma_i(t'))) = \nu_1 \cdot (\nu_2 \otimes \nu')$. By definition of ν'_1 there is a permutation $f' \in \mathcal{F}(\nu_2 \otimes \nu')$ such that $f'(\nu_2 \otimes \nu') = \nu'_1 \cdot \nu'_2$. By combining the two permutations we can finish the proof that $\nu_1 \cdot \nu'_1 \preceq_{[0,\mathsf{tmin}_1(t')-\varepsilon]} \nu(\sigma_i(t'))_{|\mathsf{tmin}_1(t')-\varepsilon}$. Gathering all evidences together, we have then proven this inclusion.

$\subseteq$ Consider $(S', \nu'_2) \in \mathsf{R}(t')$. By definition, $S' = \{(\ell_{init}, \mathbf{0}, 0)\}\mathsf{after}[\![\nu'_1]\!]_{\mathsf{ord}}$ and $\nu(\sigma_i(t')) = \nu'_1 \otimes \nu'_2$. Now consider a separation $\nu'_1 = \nu_1 \cdot \nu_2$ such that $\nu_1 \preceq_{[0,\mathsf{tmin}_1(t)-\varepsilon]} \nu(\sigma_i(t))$. Such prefix exists since (1) $\nu(\sigma_i(t))$ contains all events that can occur before $\mathsf{tmin}_1(t) - \varepsilon$ so it is enough to construct $\nu_1 \preceq_{[0,\mathsf{tmin}_1(t)-\varepsilon]} \nu(\sigma_i(t'))$ and (2) this construction can be made iteratively by adding an increasingly large prefix of events that must be in ν_1: if it was not possible to construct such a ν_1, in particular $[\![\nu'_1]\!]_{\mathsf{ord}} = \emptyset$ and S' is empty. Now, $\nu_1 \preceq_{[0,\mathsf{tmin}_1(t)-\varepsilon]} \nu(\sigma_i(t))$, so there is $\nu_{2,i}$ such that $\nu(\sigma_i(t)) = \nu_1 \otimes \nu_{2,i}$. Notice that $\nu_{2,i} \otimes \nu' = \nu_2 \otimes \nu'_2$ is the set of events in $\nu(\sigma_i(t'))$ but not in ν_1. From this and the properties of ν'_2 we can deduce that $(\nu_2, \nu'_2) \in \mathcal{D}(\nu_{2,i} \otimes \nu', \mathsf{tmin}_1(t') - \varepsilon)$. By Lemma 15 we also get that $S' = \{(\ell_{init}, \mathbf{0}, 0)\}$ after $[\![\nu_1]\!]_{\mathsf{ord}}$ after $[\![\nu_2]\!]_{\mathsf{ord}}$. Thus (S', ν'_2) is an element of the right-hand side of the equality of Proposition 11 and we get this inclusion. ◻

B.5 Proof of Section 3.4

Proposition 12. *Algorithm 1 sets the verdict to $V(t)$.*

Proof. By Proposition 11, the algorithm maintains the data structure $\mathsf{R}(t)$ at each time t since the beginning of each loop in the algorithm adds the new sequence of observations to all ν_2 in R. Then, by the definition of $\mathsf{state}(\mathsf{R})$, the computation of (I, N, C) and Proposition 6, we have our result. ◻

Theorem 13. *Monitoring at $\mathsf{tmin}_1(.) - \varepsilon$ is sound and complete where, for any local monitor M_i and its property $(\mathcal{A}_i, \mathcal{F}_i)$:*

- **soundness** *means that for any global trace $\sigma^g \in \mathsf{TT}(\mathsf{Act})$ produced at date $T \in \mathbb{R}_{\geq 0}$, if $\mathsf{verdict}(i) = \mathsf{True}$ at time T, then σ^g is a good prefix of $(\mathcal{A}_i, \mathcal{F}_i)$ (respectively, if $\mathsf{verdict}(i) = \mathsf{False}$ at time T, then σ^g is a bad prefix of $(\mathcal{A}_i, \mathcal{F}_i)$). Furthermore, if $\mathsf{verdict}(i) = \mathsf{Inconclusive}$, then neither σ^g nor its possible continuations are 2ε-conclusive on $(\mathcal{A}_i, \mathcal{F}_i)$.*
- **completeness** *means that for any global trace $\sigma^g \in \mathsf{TT}(\mathsf{Act})$, if a prefix of σ^g is 2ε-conclusive and good (respectively, bad) on $(\mathcal{A}_i, \mathcal{F}_i)$ then there exists some date $T \in \mathbb{R}_{\geq 0}$ such that $\mathsf{verdict}(i) = \mathsf{True}$ (respectively, $\mathsf{verdict}(i) = \mathsf{False}$).*

Proof. We prove both properties.

Soundness: Suppose that $\mathsf{verdict}(i) = \mathsf{True}$ (resp. $\mathsf{verdict}(i) = \mathsf{False}$) at time T. Then, by Proposition 12, $\mathsf{True}(T)$ (resp. $\mathsf{False}(T)$) holds and $[\![\nu(\sigma_i(T))_{|\mathsf{tmin}_1(T)-\varepsilon}]\!]$ is a good (resp. bad) language prefix. Then, by Proposition 3 $\sigma^g_{|\mathsf{tmin}_1(T)-\varepsilon}$ is a good (resp. bad) prefix, and so is σ^g.

In the case where $\mathsf{verdict}(i) = \mathsf{Inconclusive}$, as before, by Proposition 12, $\mathsf{Inconclusive}(T)$ holds and $[\![\nu(\sigma_i(T))_{|\mathsf{tmin}_1(T)-\varepsilon}]\!]$ intersects both good and bad prefixes. Hence, $\sigma_i(T)$ is not ε-conclusive (because good and bad language prefixes are suffix closed) and by Proposition 3, $[\![\nu(\sigma_i(T))]\!] \subseteq [\![\nu(\sigma^g, 2\varepsilon)]\!]$, meaning that σ^g is not 2ε-conclusive.

Completeness: Suppose $\sigma^{g\prime}$ is 2ε-conclusive. Then, any $\sigma \in \nu(\sigma^{g\prime})$ is ε-conclusive. By Proposition 3, as soon as $\sigma^{g\prime}$ is contained in $\sigma^g_{|\mathsf{tmin}_1(t)-\varepsilon}$, $\sigma_{i|\mathsf{tmin}_1(t)-\varepsilon}$ is ε-conclusive. Then, by Proposition 12, we have our result. It then suffices to show that there is a date T such that $\sigma^{g\prime}$ is contained in $\sigma^g_{|\mathsf{tmin}_1(t)-\varepsilon}$, *i.e.*, $\mathsf{tmin}_1(t)-\varepsilon > \mathsf{lastt}(\sigma^{g\prime})$. This is ensured by our setting hypotheses: by local liveness each component will always continue to see new events that will be sent to M_i by messages by the communication topology hypothesis. As we make the hypothesis that our communication channels do not lose messages (communication channels hypothesis) we have our result. □