Meeting Deadlines Together

Laura Bocchi[1], Julien Lange[2], and Nobuko Yoshida[2]

1 **University of Kent**
2 **Imperial College London**

Abstract

This paper studies safety, progress, and non-zeno properties of Communicating Timed Automata (CTAs), which are timed automata (TA) extended with unbounded communication channels, and presents a procedure to *build* timed global specifications from systems of CTAs. We define safety and progress properties for CTAs by extending properties studied in communicating finite-state machines to the timed setting. We then study non-zenoness for CTAs; our aim is to prevent scenarios in which the participants have to execute an infinite number of actions in a finite amount of time. We propose sound and decidable conditions for these properties, and demonstrate the practicality of our approach with an implementation and experimental evaluations of our theory.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Program

Keywords and phrases timed automata, multiparty session types, global specification

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

Meeting deadlines is part of our everyday life; this is also the case for distributed software systems that have real-time constraints, such as e-business and financial systems, where exchanges of agreements and data transmissions need to be completed within specified time-frames. Guaranteeing that a single entity will finish its assigned task within an upcoming deadline is a crucial requirement that is generally difficult to attain. It is even harder to ensure that several, distributed, and interdependent entities will work *together* in a timely fashion to meet each other's deadlines. To model such real-time distributed behaviours, communicating timed automata (CTAs) [17] have been introduced as an extension of communicating finite-state machines (CFSMs) [9] with time constraints. A system of CTAs consists of several automata that exchange messages through unbounded FIFO channels and must comply with time constraints on emission/reception of messages. These two features (unbounded channels and time) make CTAs difficult to verify, e.g., reachability is undecidable in general [12].

This paper tackles the following two shortcomings of the current state-of-the-art of CTAs. First, to the best of our knowledge, safety and progress properties, such as absence of deadlocks and unspecified reception (type) errors, which are standard in the literature on CFSMs [10], and essential for distributed systems, have not been studied in the context of CTAs. Moreover, customary properties for TAs such as time-divergence [2] and non-zenoness [7, 21] (preventing that some participant's only possible way forward is by firing actions at increasingly short intervals of time) have not been investigated for CTAs.

Second, while global specifications such as message sequent charts (MSC) and choreographies [8, 16] are useful to model protocols from a global viewpoint, there has not been any work to *build* global specifications from CTAs. The top-down approach [6] alone, which requires a preexisting global specification, is not satisfactory in agile development life-cycles [23], in refinement and reverse-engineering of existing systems, or to compose real-time distributed components, possibly dynamically (see [14, 18, 19]).

Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 1–14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

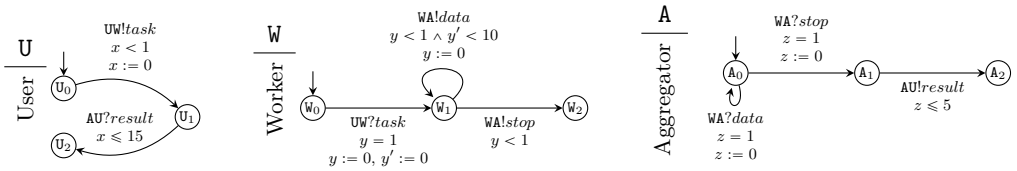


Figure 1 Scheduled Task Protocol (System S_{ST})

This work introduces classical properties of CFSMs and TAs to the world of CTAs, and investigates the interplay between asynchronous communications through unbounded channels and time constraints. We define the classes of CTAs that enjoy four properties – *safety*, *progress*, *non-zenoness*, and *eventual reception* – and give a sound decision procedure for checking whether a system of CTAs belongs to these classes. This procedure does not rely on any other information than the CTAs themselves. Interestingly, a property of CFSMs called multiparty compatibility (MC) [14], which characterises a sound and complete correspondence with multiparty session types in the untimed setting [16], soundly characterises safe CTAs and offers a basis for decidable decision procedures for progress and non-zenoness in the timed setting. We give: (i) a sound characterisation for progress by checking the satisfiability of first order logic formulae (thus verifiable by generic SMT solvers), and (ii) a sound characterisation of non-zenoness by using a synchronous execution of CTAs. Eventual reception follows from (i) and (ii). In addition, we present an algorithm to build a timed global type [6] from CTAs, whose traces are equivalent to the original system. Thus, if a system validates some of the properties discussed above, then the CTAs obtained by projecting its timed global type onto its participants will preserve these properties.

The system S_{ST} in Fig. 1 (Scheduled Task Protocol) will be used to illustrate our approach throughout the paper. S_{ST} consists of three participants (or machines): a user `U`, a worker `W`, and an aggregator `A`, who exchange messages through *unbounded* FIFO buffers. Each machine is equipped with one or more clocks, initially set to 0 and possibly reset during the protocol. Time elapses at the same pace for all clocks, which is a standard assumption [17]. The protocol is as follows: `U` sends a task to `W`, `W` progressively sends intermediary data to `A`, and finally `A` sends the aggregated result to `U`. The time constraints are:

- `U` must send a task to `W` within one time unit, reset its clock x, and expects to receive the result within 15 time units.
- `W` must consume `U`'s *task* message at time 1, reset its clocks y and y', and repeatedly send *data* to `A`, waiting less than 1 time unit between each emission (modelled by the constraint and reset on y). The overall iteration cannot last more than 10 time units (modelled by the constraint on y', which is not reset in the loop). When `W` has finished, it must send a notification *stop* to `A`.
- `A` must read intermediary data every 1 time unit, reset each time its clock z, and send the overall result to `U` within 5 time units after receiving *stop*.

This example, albeit small, models a complex interaction where each machine has its own, interdependent, deadlines; e.g., `U` relies on the other machines' deadlines to receive the final result within 15 time units. Note that the channel between `W` and `A` is *unbounded*: `W` can send to `A` an arbitrary number of messages before `A` receives them, cf. $\texttt{WA!}data(y < 1 \wedge y' < 10, y := 0)$.

Contribution and synopsis In the rest of the paper, we give several conditions that guarantee that no participant misses its deadlines, that every message sent is eventually received *on*

time, and that no participant is forced to perform actions infinitely fast, i.e., forced into a zeno behaviour. In § 2 we recall basic definitions on CTAs. In § 3 we extend the standard safety properties of CFSMs to the timed setting, and show that multiparty compatibility (MC) is a sound condition for safety (Theorem 6). MC CTAs still allow undesirable scenarios when, e.g., (1) the system gets stuck because of unmeetable deadlines, (2) the system's only possibility to meet its deadlines is through zeno behaviours, or (3) sent messages are never received. We give sound and decidable conditions to rule out (1) in § 4 (Theorem 13) and (2-3) in § 5 (Theorem 17 and Theorem 19). In § 6, we discuss the applications of our theory and its implementation. The work in [6] studies a correspondence between timed local types (projected from timed global types) and CTAs, focusing on type-checking timed π-calculus processes. The present work studies CTAs directly, i.e., without relying on a priori global knowledge of the system, and gives more general conditions for safety, progress, and non-zenoness. None of the previous works [14, 18, 19] on building global specifications from local ones caters for time constraints. Unlike existing work on the properties of CTAs (e.g., reachability) our results do not set limitations to channel size or to network topologies [12, 17]. We discuss related work further in § 7. The proofs, additional material, and the implementation are available online [3].

2 Communicating Timed Automata

We introduce communicating timed automata (CTA) following definitions from [14, 17]. Fix a finite set $\mathcal{P}$ of *participants* (ranged over by $\mathtt{p}$, $\mathtt{q}$, $\mathtt{r}$, $\mathtt{s}$, etc.). Let $\mathbb{A}$ be a finite alphabet of messages ranged over by a, b, etc. The set of finite words on $\mathbb{A}$ is denoted by $\mathbb{A}^*$, ww' is the concatenation of w and w', and ε is the empty word (overloaded on any alphabet). The set of *channels* is $C \stackrel{\text{def}}{=} \{\mathtt{pq} \mid \mathtt{p}, \mathtt{q} \in \mathcal{P} \text{ and } \mathtt{p} \neq \mathtt{q}\}$. Given a (finite) set of *clocks* $\mathcal{X}$ (ranged over by x, y, etc.), the set of *actions* (ranged over by ℓ) is $Act_{\mathcal{X}} \stackrel{\text{def}}{=} C \times \{!, ?\} \times \mathbb{A} \times \Phi(\mathcal{X}) \times 2^{\mathcal{X}}$, and the set of *guards* (ranged over by g) $\Phi(\mathcal{X})$ is

$$g ::= \mathsf{true} \mid x \leqslant c \mid c \leqslant x \mid \neg g \mid g_1 \wedge g_2$$

where c ranges over constants in $\mathbb{Q}_{\geqslant 0}$, and from which we derive the usual abbreviations. We write $\mathsf{fc}(g)$ for the set of clocks in g and $\mathtt{sr}!a(g,\lambda)$ or $\mathtt{sr}?a(g,\lambda)$ for an element of $Act_{\mathcal{X}}$. Action $\mathtt{sr}!a(g,\lambda)$ says that $\mathtt{s}$ sends a message a to $\mathtt{r}$, provided that guard g is satisfied, and resets the clocks in $\lambda \subseteq \mathcal{X}$; the dual receiving action is $\mathtt{sr}?a(g,\lambda)$. Given $\ell = \mathtt{sr}!a(g,\lambda)$ or $\ell = \mathtt{sr}?a(g,\lambda)$, we define: $\mathsf{msg}(\ell) = a$, $\mathsf{guard}(\ell) = g$, and $\mathsf{reset}(\ell) = \lambda$. We define the subject of an action: $\mathsf{subj}(\mathtt{pr}!a(g,\lambda)) = \mathsf{subj}(\mathtt{sp}?a(g,\lambda)) \stackrel{\text{def}}{=} \mathtt{p}$.

A *communicating timed automaton*, or *machine*, is a finite transition system given by a tuple $M = (Q, q_0, \mathcal{X}, \delta)$ where Q is a finite set of *states*, $q_0 \in Q$ is the initial state, $\mathcal{X}$ is a set of clocks, and $\delta \subseteq Q \times Act_{\mathcal{X}} \times Q$ is a set of *transitions*. We write $q \xrightarrow{\ell} q'$ when $(q, \ell, q') \in \delta$.

A machine $M = (Q, q_0, \mathcal{X}, \delta)$ is *deterministic* if for all states $q \in Q$ and all actions $\ell, \ell' \in Act_{\mathcal{X}}$, if $(q, \ell, q'), (q, \ell', q'') \in \delta$ and $\mathsf{msg}(\ell) = \mathsf{msg}(\ell')$, then $q' = q''$ and $\ell = \ell'$. A state $q \in Q$ is: *final* if it has no outgoing transitions; *sending* (resp. *receiving*) if it is not final and each of its outgoing transitions is of the form $\mathtt{sr}!a(g,\lambda)$ (resp. $\mathtt{sr}?a(g,\lambda)$); and *mixed* if it is neither final, sending, nor receiving. We say that q is *directed* if it contains only sending/receiving actions to/from the same participant. Hereafter, we only consider deterministic machines, whose states are directed and not mixed. These assumptions, adapted from [14], ensure that a machine corresponds to a syntactic local session type [16]. We discuss how to lift some of these restrictions in § 7.

A *timed communicating system* consists of a finite set of machines and a set of queues (one

for each channel) used for asynchronous message passing. Given a valuation $\nu : \mathcal{X} \to \mathbb{R}_{\geqslant 0}$ of the clocks in $\mathcal{X}$, $\nu \models g$ denotes that the guard g is satisfied by ν and $\lambda(\nu)$ denotes a valuation where all clocks in λ are set to 0 (reset) and clocks not in λ keep their values in ν.

▸ **Definition 1** (Timed communicating system)**.** A timed communicating system (or *system*), is a tuple $S = (M_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$ where each $M_{\mathtt{p}} = (Q_{\mathtt{p}}, q_{0\mathtt{p}}, \mathcal{X}_{\mathtt{p}}, \delta_{\mathtt{p}})$ is a CTA and for all $\mathtt{p} \neq \mathtt{q} \in \mathcal{P}$: $\mathcal{X}_{\mathtt{p}} \cap \mathcal{X}_{\mathtt{q}} = \varnothing$. A *configuration* of S is a triple $s = (\vec{q}; \vec{w}; \nu)$ where: $\vec{q} = (q_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$ is the *control state* and $q_{\mathtt{p}} \in Q_{\mathtt{p}}$ is the *local state* of machine $M_{\mathtt{p}}$; $\vec{w} = (w_{\mathtt{pq}})_{\mathtt{pq}\in C}$ with $w_{\mathtt{pq}} \in \mathbb{A}^*$ is a vector of queues; $\nu : \bigcup_{\mathtt{p}\in\mathcal{P}} \mathcal{X}_{\mathtt{p}} \to \mathbb{R}_{\geqslant 0}$ is a clock valuation. The *initial configuration* of S is $s_0 = (\vec{q_0}; \vec{\varepsilon}; \nu_0)$ with $\vec{q_0} = (q_{0\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$, $\vec{\varepsilon}$ being the vector of empty queues, and $\nu_0(x) = 0$ for each clock $x \in \bigcup_{\mathtt{p}\in\mathcal{P}} \mathcal{X}_{\mathtt{p}}$. ◇

Hereafter, we fix a machine $M_{\mathtt{p}} = (Q_{\mathtt{p}}, q_{0\mathtt{p}}, \mathcal{X}_{\mathtt{p}}, \delta_{\mathtt{p}})$ for each participant $\mathtt{p} \in \mathcal{P}$ (assuming that $\forall \mathtt{p} \in \mathcal{P} : \ (q, \ell, q') \in \delta_{\mathtt{p}} \implies \mathsf{subj}(\ell) = \mathtt{p}$), and let $S = (M_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$ be the corresponding system. We write $\mathcal{X}$ for $\bigcup_{\mathtt{p}\in\mathcal{P}} \mathcal{X}_{\mathtt{p}}$ and $\nu + t$ for the valuation mapping each $x \in \mathcal{X}$ to $\nu(x) + t$. The definition below is from [17, Definition 1], omitting internal transitions.

▸ **Definition 2** (Reachable configuration)**.** Configuration $s' = (\vec{q'}; \vec{w'}; \nu')$ is *reachable* from configuration $s = (\vec{q}; \vec{w}; \nu)$ by *firing the transition* α, written $s \xrightarrow{\alpha} s'$ (or $s \to s'$ when the label is immaterial), if either:

1. $(q_{\mathtt{s}}, \mathtt{sr}!a(g, \lambda), q'_{\mathtt{s}}) \in \delta_{\mathtt{s}}$ and (a) $q'_{\mathtt{p}} = q_{\mathtt{p}}$ for all $\mathtt{p} \neq \mathtt{s}$; (b) $w'_{\mathtt{sr}} = w_{\mathtt{sr}}a$ and $w'_{\mathtt{pq}} = w_{\mathtt{pq}}$ for all $\mathtt{pq} \neq \mathtt{sr}$; (c) $\nu' = \lambda(\nu)$; (d) $\alpha = \mathtt{sr}!a(g, \lambda)$, and $\nu \models g$;
2. $(q_{\mathtt{r}}, \mathtt{sr}?a(g, \lambda), q'_{\mathtt{r}}) \in \delta_{\mathtt{r}}$ and (a) $q'_{\mathtt{p}} = q_{\mathtt{p}}$ for all $\mathtt{p} \neq \mathtt{r}$; (b) $w_{\mathtt{sr}} = aw'_{\mathtt{sr}}$ and $w'_{\mathtt{pq}} = w_{\mathtt{pq}}$ for all $\mathtt{pq} \neq \mathtt{sr}$; (c) $\nu' = \lambda(\nu)$; (d) $\alpha = \mathtt{sr}?a(g, \lambda)$ and $\nu \models g$; or
3. $\alpha = t \in \mathbb{R}_{\geqslant 0}$, $\nu' = \nu + t$, $w'_{\mathtt{pq}} = w_{\mathtt{pq}}$ for all $\mathtt{pq} \in C$, and $q'_{\mathtt{p}} = q_{\mathtt{p}}$ for all $\mathtt{p} \in \mathcal{P}$.

We let ρ range over sequences of labels $\alpha_1 \cdots \alpha_k$ and write $\to^*$ for the reflexive transitive closure of $\to$. The *reachability set* of S is $RS(S) \stackrel{\text{def}}{=} \{s \mid s_0 \to^* s\}$. ◇

Condition (1) allows a machine $\mathtt{s}$ to put a message a on queue $\mathtt{sr}$, if the time constraints in g are satisfied by ν; dually, (2) allows $\mathtt{r}$ to consume a message from the queue, if g is satisfied; and (3) models the elapsing of time (or a delay).

3 Safety in CTAs

This section defines safe CTAs and gives a sufficient condition for safety, called multiparty compatibility (MC) [14], in the timed setting. Here, we present a new approach based on *synchronous transition systems* (*STS*); the *STS* is also useful for defining progress and non-zeno properties in § 4.

Let n range over vectors of local states; and e range over events, which are elements of the set $C \times \mathbb{A} \times \Phi(\mathcal{X}) \times 2^{\mathcal{X}} \times \Phi(\mathcal{X}) \times 2^{\mathcal{X}}$, and write $(\mathtt{s} \to \mathtt{r} : a;\ g_{\mathtt{s}}, \lambda_{\mathtt{s}};\ g_{\mathtt{r}}, \lambda_{\mathtt{r}})$ for the event in which $\mathtt{s}$ sends message a to $\mathtt{r}$, with $\mathtt{s}$ (resp. $\mathtt{r}$) having guard $g_{\mathtt{s}}$ (resp. $g_{\mathtt{r}}$) and resets $\lambda_{\mathtt{s}}$ (resp. $\lambda_{\mathtt{r}}$). We introduce the synchronous transition system of S, following [19].

▸ **Definition 3** (Synchronous transition system)**.** The *synchronous transition system* of S, written $STS(S)$, is a tuple $(N, n_0, \hookrightarrow, E)$ such that:

- $\hookrightarrow$ is the relation defined as $n \stackrel{e}{\hookrightarrow} n'$ with $e = (\mathtt{s} \to \mathtt{r} : a;\ g_{\mathtt{s}}, \lambda_{\mathtt{s}};\ g_{\mathtt{r}}, \lambda_{\mathtt{r}})$ iff $n = \vec{q}, n' = \vec{q'}, q_{\mathtt{s}} \xrightarrow{\mathtt{sr}!a(g_{\mathtt{s}}, \lambda_{\mathtt{s}})} q'_{\mathtt{s}}, q_{\mathtt{r}} \xrightarrow{\mathtt{sr}?a(g_{\mathtt{r}}, \lambda_{\mathtt{r}})} q'_{\mathtt{r}}$, and $\forall \mathtt{p} \in \mathcal{P} \setminus \{\mathtt{s}, \mathtt{r}\} : \ q_{\mathtt{p}} = q'_{\mathtt{p}}$ (write $\hookrightarrow$ when e is unimportant and $\hookrightarrow^*$ for the reflexive and transitive closure of $\hookrightarrow$);
- $n_0 = \vec{q_0}$ is the initial node; $N = \{n \mid n_0 \hookrightarrow^* n\}$ is the (finite) set of nodes; and $E = \{e \mid \exists n, n' \in N \text{ and } n \stackrel{e}{\hookrightarrow} n'\}$ is the set of events.

We write $n_1 \xrightarrow{e_1 \cdots e_k} n_{k+1}$, when, for some $n_2, \ldots, n_k \in N$, $n_1 \overset{e_1}{\hookrightarrow} n_2 \cdots n_k \overset{e_k}{\hookrightarrow} n_{k+1}$. Let φ range over (possibly empty) sequences of events $e_1 \cdots e_k$, and ε denote the empty sequence. ◇

The *STS* of the Scheduled Task Protocol (S_{ST}) is given in Fig. 2; essentially, it models all the synchronous executions of S_{ST}. In the following, we fix $STS(S) = (N, n_0, \hookrightarrow, E)$.

Given $e = (\mathtt{s} \to \mathtt{r} : a; g_{\mathtt{s}}, \lambda_{\mathtt{s}}; g_{\mathtt{r}}, \lambda_{\mathtt{r}})$, we define $\mathsf{sid}(e) \overset{\text{def}}{=} \mathtt{s}$, $\mathsf{rid}(e) \overset{\text{def}}{=} \mathtt{r}$, and $\mathsf{id}(e) \overset{\text{def}}{=} \{\mathtt{s}, \mathtt{r}\}$. The projection of e on $\mathtt{p}$ (written $e\!\upharpoonright_{\mathtt{p}}$) is given by: $(\mathtt{s} \to \mathtt{r} : a; g_{\mathtt{s}}, \lambda_{\mathtt{s}}; g_{\mathtt{r}}, \lambda_{\mathtt{r}})\!\upharpoonright_{\mathtt{s}} = \mathtt{sr}!a(g_{\mathtt{s}}, \lambda_{\mathtt{s}})$; $(\mathtt{s} \to \mathtt{r} : a; g_{\mathtt{s}}, \lambda_{\mathtt{s}}; g_{\mathtt{r}}, \lambda_{\mathtt{r}})\!\upharpoonright_{\mathtt{r}} = \mathtt{sr}?a(g_{\mathtt{r}}, \lambda_{\mathtt{r}})$; and $(\mathtt{s} \to \mathtt{r} : a; g_{\mathtt{s}}, \lambda_{\mathtt{s}}; g_{\mathtt{r}}, \lambda_{\mathtt{r}})\!\upharpoonright_{\mathtt{p}} = \varepsilon$, if $\mathtt{p} \notin \{\mathtt{s}, \mathtt{r}\}$. We extend $\varphi\!\upharpoonright_{\mathtt{p}}$ to sequences of events and, given $n \in N$, define $\mathsf{ids}(n) \overset{\text{def}}{=} \bigcup \{\mathsf{id}(e) \mid n \hookrightarrow^* \overset{e}{\hookrightarrow}\}$.

▸ **Definition 4** (Multiparty compatibility (MC)). System S is *multiparty compatible* if for all $\mathtt{p} \in \mathcal{P}$, for all $q \in Q_{\mathtt{p}}$, and for all $n = \vec{q} \in N$, if $q_{\mathtt{p}} = q$, then

1. if q is a sending state, then $\forall (q, \ell, q') \in \delta_{\mathtt{p}} : \exists \varphi, \exists e \in E : n \overset{\varphi \cdot e}{\hookrightarrow} \wedge\, e\!\upharpoonright_{\mathtt{p}} = \ell \wedge \varphi\!\upharpoonright_{\mathtt{p}} = \varepsilon$;
2. if q is a receiving state, then $\exists (q, \ell, q') \in \delta_{\mathtt{p}} : \exists \varphi, \exists e \in E : n \overset{\varphi \cdot e}{\hookrightarrow} \wedge\, e\!\upharpoonright_{\mathtt{p}} = \ell \wedge \varphi\!\upharpoonright_{\mathtt{p}} = \varepsilon$. ◇

Intuitively, condition (1) ensures that for every sending state, all messages that can be sent can also be received, while (2) guarantees that, for every receiving state, at least one transition will be eventually fireable, i.e., an *expected* message will eventually be received. System S_{ST}, in Fig. 1, is multiparty compatible.

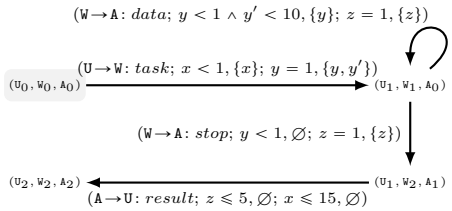


Figure 2 *STS* for Scheduled Task, cf. Fig. 1

Observe that $STS(S)$ and MC do not address time constraints. In fact, $STS(S)$ might include interactions forbidden by time constraints. These can be ruled out at a later stage when analysing time properties in § 4. We deliberately kept communication and time properties separated, so that we can provide simpler and modular definitions in § 7. Crucially, MC guarantees that any *asynchronous* execution can be mapped to a path in $STS(S)$, i.e., it can be simulated by $STS(S)$.

We recall two types of errors from the CFSM model, which are ruled out by MC also in the timed setting. Let $s = (\vec{q}; \vec{w}; \nu)$ be a configuration of a system S; s is a ***deadlock configuration*** [10, Def. 12] if $\vec{w} = \vec{\varepsilon}$, there is $\mathtt{r} \in \mathcal{P}$ such that $q_{\mathtt{r}}$ is a receiving state, and for every $\mathtt{p} \in \mathcal{P}$, $q_{\mathtt{p}}$ is a receiving or final state, i.e., all machines are blocked waiting for messages; and s is an ***orphan message configuration*** if all $q_{\mathtt{p}} \in \vec{q}$ are final but $\vec{w} \neq \vec{\varepsilon}$, i.e., there is at least a non-empty buffer and all the machines are in a final state.

▸ **Definition 5** (Safe system). S is *safe* if for all $s \in RS(S)$, s is not a deadlock, nor an orphan message configuration. ◇

▸ **Theorem 6** (Safety). *If S is multiparty compatible, then it is safe.*

The proof follows from the fact that (*i*) MC guarantees safety in CFSMs [14] and (*ii*) time constraints imply that a subset of the configurations reachable in the untimed setting are reachable in the timed setting (modulo clock valuations). Thus, if there is a deadlock or an orphan message configuration in the timed setting, there is one in the untimed setting, which contradicts the results in [14].

The projection $STS(S)\!\upharpoonright_{\mathtt{p}}$ of a synchronous transition system $STS(S)$ on a machine $\mathtt{p}$ is given by substituting each event $e \in E$ with its projection $e\!\upharpoonright_{\mathtt{p}}$, then minimising the automaton w.r.t. language equivalence. For example, the projections of $STS(S)$ onto $\mathtt{U}$, $\mathtt{W}$, and $\mathtt{A}$ are isomorphic to the system S_{ST} in Fig. 1. Below $\sim$ denotes the standard timed bisimulation [15].

▸ **Theorem 7** (Equivalence). *If* $S = (M_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$ *is MC then* $S \sim (STS(S)\!\downarrow_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$.

Theorem 7 says that the behaviour of the original system is preserved by $STS(S)$, this result is crucial to be able to construct a global specification that is equivalent to a system of CTAs. It follows from the fact that, (i) if the system is MC, then all the machine's behaviour is preserved except for the receive actions that are never executed; and (ii) since we assume that the machines are deterministic w.r.t. messages, the projections of $STS(S)$ also preserve all required transitions.

4 Progress with Time Constraints

This section introduces a *progress* property for CTAs, ensuring that no communication mismatch prevents the progress of the overall system (cf. § 4.1). In § 4.2, we give a sufficient condition to guarantee progress in CTAs (cf. Theorem 13).

4.1 Progress Properties

We identify several types of errors, inspired by their counterparts in the (untimed) CFSM model, which may arise in timed communicating systems. Let $s = (\vec{q}; \vec{w}; \nu) \in RS(S)$; s is an ***unsuccessful reception configuration*** if there exists $\mathtt{r} \in \mathcal{P}$ such that $q_{\mathtt{r}}$ is a receiving state, and for all $(q_{\mathtt{r}}, \mathtt{sr}?a(g,\lambda), q'_{\mathtt{r}}) \in \delta_{\mathtt{r}}$ either (i) $w_{\mathtt{sr}} \neq \varepsilon$ and $w_{\mathtt{sr}} \notin a\mathbb{A}^*$ or (ii) $\forall t \in \mathbb{R}_{\geqslant 0} : \nu + t \not\models g$ (i.e., $\mathtt{r}$ cannot receive messages from any of its queues, as they either contain an unexpected message or none of the transition guards will ever be satisfied); and s is an ***unfeasible configuration*** if there exists $\mathtt{s} \in \mathcal{P}$ such that $q_{\mathtt{s}}$ is a sending state, and $(q_{\mathtt{s}}, \mathtt{sr}!a(g,\lambda), q'_{\mathtt{s}}) \in \delta_{\mathtt{s}}$ implies that $\forall t \in \mathbb{R}_{\geqslant 0} : \nu + t \not\models g$ (i.e., $\mathtt{s}$ is unable to send a message because none of its guards will ever be satisfied).

▸ **Definition 8** (Progress). S satisfies the *progress* property if for all $s \in RS(S)$, s is not a deadlock, an orphan message, an unsuccessful reception, nor an unfeasible configuration. ◇

Observe that the original semantics of CTAs in [17] and in Def. 2 do not allow us to identify unsuccessful reception or unfeasible configurations. From Def. 2, a system may take a time transition which *permanently* prevents a machine from firing further actions. Below, we adjust the semantics of CTAs and give examples of "undesirable" scenarios it prevents.

▸ **Definition 9** (Reachable configuration (2)). $s \xrightarrow{\alpha} s'$ is defined as Def. 2, replacing (3) with:

3. $\alpha = t \in \mathbb{R}_{>0}$, $\nu' = \nu + t$, $\forall\, \mathtt{pq} \in C : w'_{\mathtt{pq}} = w_{\mathtt{pq}}$, and $\forall\, \mathtt{p} \in \mathcal{P} : q'_{\mathtt{p}} = q_{\mathtt{p}}$ and
   a. $q_{\mathtt{p}}$ sending $\implies \exists (q_{\mathtt{p}}, \ell, q''_{\mathtt{p}}) \in \delta_{\mathtt{p}} : \exists t' \in \mathbb{R}_{\geqslant 0} : \nu' + t' \models \mathsf{guard}(\ell)$
   b. $\forall (q_{\mathtt{p}}, \mathtt{sp}?a(g,\lambda), q''_{\mathtt{p}}) \in \delta_{\mathtt{p}} : (w_{\mathtt{sp}} \in a\mathbb{A}^* \implies \exists t' \in \mathbb{R}_{\geqslant 0} : \nu' + t' \models g)$

Unless stated otherwise, we only consider this semantics hereafter. ◇

Condition (3a) handles the case of machines waiting to perform send actions, and (3b) handles receive transitions, as illustrated by the examples below:

Consider configuration $((q_0, q_2); \vec{\varepsilon}; \nu_0)$ in which $\mathtt{s}$ must send a message within 3 time units. Condition (3a) prevents a time transition with delay $t = 3$. Indeed, with a clock valuation $\nu_0 + 3$, none of the action of $\mathtt{s}$ from q_0 can be fired. Consider now configuration $((q_1, q_2); \vec{w}; \nu)$ with $w_{\mathtt{sr}} = a$ and $\nu(x) = \nu(y) = 3.5$. Condition (3b) rules out a time transition with $t = 1$. Indeed, even if $\mathtt{r}$ has a transition whose guard will be enabled after time $\nu(y) + 1 = 4.5$, i.e.,

$(q_2, \mathtt{sr}?b(y = 5), q_3)$, this transition cannot be fired due to the content of queue $w_{\mathtt{sr}} \notin b\mathbb{A}^*$; on the other hand transition $(q_2, \mathtt{sr}?a(y = 4), q_3)$ is no longer fireable, due to its time constraint.

4.2 A Sound Characterisation of Progress

Roadmap We give a sound condition that guarantees progress in the presence of time constraints. The main property, *interaction-enabling* (IE) in Def. 12, essentially checks that future actions are possible. IE guarantees that: (1) whatever the past, each machine that is in a sending state is eventually able to fire one of its transitions and (2) for every message that is sent, there exists a (future) time where this message can be received. IE relies on checking whether an action ℓ is *progress enabling* (Def. 11) which ensures that, for all possible past clock valuations, there exists a future time where the guard of ℓ is satisfied.

In the rest of this section, we give (i) a procedure for understanding the past of a configuration, based on a graph modelling the causal dependencies between previously executed actions; and (ii) a procedure to check that, for any reachable configuration, there is always a future time where an available action can be fired.

Understanding the past We check that S has progress by analysing paths, i.e., sequences of events, in $STS(S)$. Since $STS(S)$ gives an over-approximation of the causal dependencies between actions, we will construct a graph of the actual dependencies of the underlying actions of a path. We compute the underlying actions of a path via the function:

$$\mathsf{nodes}(e_1 \cdots e_k) \stackrel{\text{def}}{=} e_1\!\upharpoonright_{\mathsf{sid}(e_1)} \cdot e_1\!\upharpoonright_{\mathsf{rid}(e_1)} \cdots e_k\!\upharpoonright_{\mathsf{sid}(e_k)} \cdot e_k\!\upharpoonright_{\mathsf{rid}(e_k)} \qquad (k \geqslant 0)$$

Remarkably, given a path φ and two actions ℓ_i and ℓ_j in $\mathsf{nodes}(\varphi)$, $i < j$ does not imply that there is a causal dependency between ℓ_i and ℓ_j. For instance, in

$$\mathsf{nodes}(\varphi) = \mathtt{sr}!a(x < 10, \varnothing) \cdot \mathtt{sr}?a(10 \leqslant y, \varnothing) \cdot \mathtt{sp}!a(x < 10, \varnothing) \cdot \mathtt{sp}?a(10 \leqslant z, \varnothing)$$

the two receive actions $\mathtt{sr}?a(10 \leqslant y, \varnothing)$ and $\mathtt{sp}?a(10 \leqslant z, \varnothing)$ may not always be executed in that order, since they are executed by two different participants.

The graph of dependencies of an action ℓ_k in a sequence of actions $\ell_1 \cdots \ell_k$ (Def. 10 below) gives an abstraction of all actions on which ℓ_k depends. This is done by taking into account two kinds of dependencies: output/input dependencies between matching send and receive actions, and local dependencies within a single machine.

▸ **Definition 10** (Graph of Dependencies). Let $\mathsf{dep}(\varepsilon; \ell) \stackrel{\text{def}}{=} \varnothing$ and

$$\mathsf{dep}(\rho \cdot \ell_1; \ell_2) \stackrel{\text{def}}{=} \begin{cases} \{(\ell_1, \ell_2)\} \cup \mathsf{dep}(\rho; \ell_i)_{i=1,2} & \text{if } \ell_1 = \mathtt{sr}!a(g_1, \lambda_1),\ \ell_2 = \mathtt{sr}?a(g_2, \lambda_2) \\ \{(\ell_1, \ell_2)\} \cup \mathsf{dep}(\rho; \ell_1) & \text{if } \mathsf{subj}(\ell_1) = \mathsf{subj}(\ell_2) \\ \mathsf{dep}(\rho; \ell_2) & \text{otherwise} \end{cases}$$

The graph of dependencies of $\rho = \ell_1 \cdots \ell_k$ $(k > 0)$, written $\mathsf{DG}(\rho)$, is the graph (D, A) s.t. $A = \mathsf{dep}(\ell_1 \cdots \ell_{k-1}; \ell_k) \backslash \{(\ell_i, \ell_k) \,|\, 1 \leqslant i < k\}$ and $D = \{\ell_r \neq \ell_k \mid \exists (\ell_i, \ell_j) \in A \wedge r \in \{i, j\}\}$.[1] ◇

$\mathsf{DG}(\ell_1 \cdots \ell_k)$ is a graph whose nodes form a subset of $\{\ell_1, \ldots, \ell_{k-1}\}$ and whose edges model causal dependencies between actions (computed backwards starting from ℓ_k). In Fig. 3 (in solid black), we give the graph of dependencies of $\mathtt{WA}?data(z = 1, \{z\})$ in the sequence ρ_{ST}, corresponding to an execution of the Scheduled Task Protocol.

[1] For the sake of presentation, we write ℓ_i for the node (i, ℓ_i) in D where ℓ_i is an action in ρ and i is its position in ρ. This guarantees that each element in ρ is assigned a unique node in D.

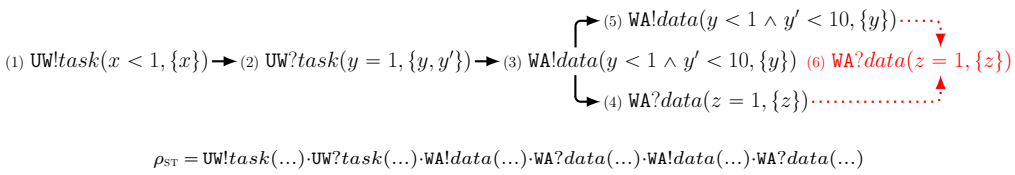


Figure 3 Graph of dependencies $\mathsf{DG}(\rho_{\text{ST}})$, in solid black, cf. Scheduled Task Protocol (Fig. 1)

$$\mathsf{idx}(\rho) \stackrel{\text{def}}{=} \{i \mid \ell_i \in D\} \qquad W_x^i(\rho) \stackrel{\text{def}}{=} \begin{cases} v_i - v_j & \text{if } 0 \leqslant j = max\{j < i \mid \ell_j \in D \wedge x \in \mathsf{reset}(\ell_j)\} \\ v_i & \text{otherwise} \end{cases}$$

$$\mathsf{allpast}(\rho) \stackrel{\text{def}}{=} \bigwedge_{i \in \mathsf{idx}(\rho)} \mathsf{absolute}_\rho(\ell_i)$$

$$\mathsf{elapse}(\rho) \stackrel{\text{def}}{=} \bigwedge_{(\ell_i,\ell_j)\in A} v_i \leqslant v_j \qquad \mathsf{absolute}_\rho(\ell_i) \stackrel{\text{def}}{=} \mathsf{guard}(\ell_i)\ \{x \mapsto W_x^i(\rho)\}_{x\in\mathcal{X}}$$

Figure 4 Functions on graphs of dependencies, where $\mathsf{DG}(\rho) = (D, A)$

Given a graph of dependencies $\mathsf{DG}(\rho)$, we define several functions that allow us to construct predicates modelling the past. The definitions of these functions are given in Fig. 4, where we fix $\mathsf{DG}(\rho) = (D, A)$. Below, we illustrate how they behave using $\mathsf{DG}(\rho_{\text{ST}})$ in Fig. 3. First, we transform the guard of an action ℓ_i such that its solutions are the possible *absolute* times (i.e., from the initial configuration of the system) in which one may execute ℓ_i (taking into account the last reset of each clock in ρ). In our example, we have:

$$\mathsf{absolute}_{\rho_{\text{ST}}}(\ell_5) = v_5 - v_3 < 1 \wedge v_5 - v_2 < 10 \qquad \text{with } \ell_5 = \texttt{WA!}data(y < 1 \wedge y' < 10, \{y\})$$

Observe that clock y (resp. y') is replaced by the difference between variable v_5 and variable v_3 (resp. v_2) corresponding to the *latest* step where y (resp. y') was reset. Unifying, e.g., y and y' into v_5 models the fact that time elapses at the same pace for all clocks. Next, we aggregate the information in $\mathsf{DG}(\rho)$, by (i) recording the indices of all the actions on which ℓ_k depends ($\mathsf{idx}(\rho)$); (ii) taking the conjunction of all constraints in absolute time ($\mathsf{allpast}(\rho)$); and (iii) recording the fact that time never decreases between two causally dependent actions ($\mathsf{elapse}(\rho)$). Taking the dependencies for ρ_{ST} in Fig. 3, we have:

$$\mathsf{allpast}(\rho_{\text{ST}}) = v_1 < 1 \wedge v_2 = 1 \wedge (v_3 - v_2 < 1 \wedge v_3 - v_2 < 10) \wedge v_4 = 1 \wedge (v_5 - v_3 < 1 \wedge v_5 - v_2 < 10)$$
$$\mathsf{elapse}(\rho_{\text{ST}}) = v_1 \leqslant v_2 \ \wedge\ v_2 \leqslant v_3 \ \wedge\ v_3 \leqslant v_4 \ \wedge\ v_3 \leqslant v_5 \qquad \mathsf{idx}(\rho_{\text{ST}}) = \{1, 2, 3, 4, 5\}$$

Predicting the future We now give the main definition of this section, allowing to check whether the past implies that there exists a satisfiable future. We use the functions defined above to check whether a given event in $STS(S)$ can indeed meet its time constraints.

▸ **Definition 11** (Progress enabling (PE)). A pair (n, e) is *progress enabling* (PE) for $\mathsf{p} \in \mathsf{id}(e)$ if for *all* paths φ such that $n_0 \stackrel{\varphi}{\hookrightarrow} n$, letting:

$$\rho = \begin{cases} \mathsf{nodes}(\varphi \cdot e) & \text{if } \mathsf{p} = \mathsf{rid}(e) \\ \mathsf{nodes}(\varphi) \cdot e{\downarrow_{\mathsf{sid}(e)}} & \text{otherwise} \end{cases}$$

and $k = |\rho|$, $\ell_k = e{\downarrow_{\mathsf{p}}}$, $\vec{v} = \{v_i \mid i \in \mathsf{idx}(\rho)\}$; the following holds

$$\forall \vec{v}\ \exists v_k : \quad \mathsf{allpast}(\rho) \ \wedge\ \mathsf{elapse}(\rho) \implies \mathsf{absolute}_\rho(\ell_k) \ \wedge\ \textstyle\bigwedge_{v_i \in \vec{v}} v_i \leqslant v_k$$

A pair (n, φ) is *recursively progress enabling* (RPE) for $P \subseteq \mathcal{P}$ if $\varphi = \varepsilon$ and $P = \varnothing$; or if (n, e) is PE for $\mathsf{sid}(e)$ and for $\mathsf{rid}(e)$ and (n', φ') is RPE for $P \backslash \mathsf{id}(e)$ with $\varphi = e \cdot \varphi'$ and $n \xhookrightarrow{e} n'$. ◇

Given a node n and an event e in $STS(S)$, and a participant $\mathtt{p}$, the above definition ensures that for all possible past clock valuations, there exists a *future* time where participant $\mathtt{p}$ has the possibility to execute action $e\lfloor_{\mathtt{p}}$. For instance, the pair $((\mathtt{U}_1, \mathtt{W}_1, \mathtt{A}_0), (\mathtt{W} \to \mathtt{A} : \mathit{data};\ y < 1 \wedge y' < 10, \{y\};\ z = 1, \{z\}))$ is PE for $\mathtt{A}$, notably because the following holds:

$$\forall v_1 \dots v_5\ \exists v_6 :\ \mathsf{allpast}(\rho_{\mathrm{ST}})\ \wedge\ \mathsf{elapse}(\rho_{\mathrm{ST}})\ \Longrightarrow\ (v_6 - v_4) = 1 \wedge v_1 \leqslant v_6 \dots v_5 \leqslant v_6$$

Below, Def. 11 is used in $STS(S)$ to ensure progress of the overall system.

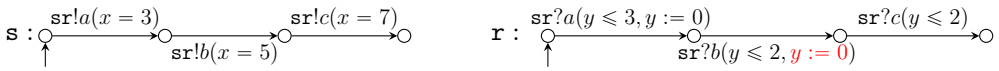
▸ **Definition 12** (Interaction enabling (IE)). A node $n \in N$ is interaction enabling (IE) if either (i) it is final or (ii) the following conditions hold:

1. There is $e \in E$ and φ such that $n \xhookrightarrow{e \cdot \varphi}$ and $(n, e \cdot \varphi)$ is RPE for $\mathsf{ids}(n)$;
2. For all $e \in E$ such that $n \xhookrightarrow{e} n'$, (n, e) is PE for $\mathsf{rid}(e)$, and n' is IE.

A system S is *interaction enabling* (IE) if n_0 is IE. ◇

Def. 12 recursively checks the nodes of $STS(S)$ (starting from n_0) and for each ensures that: (1) there is at least one path, involving all the participants still active at node n, that is RPE, i.e., where each guard along that path is satisfied for any past; (2) each receive action is PE and its successor is IE (note that a send action is always a dependency of its receive action). Condition (1) ensures that no sender will be left in a configuration where it cannot send any message, due to time constraints being unsatisfiable; condition (2) ensures that a receive action is always feasible given that its corresponding send action was executed.

Examples **(1)** The first example shows how resets affect the satisfiability of guards.

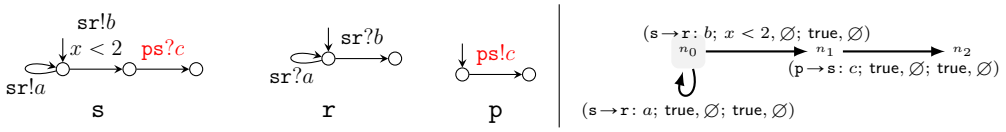


The system above is IE, notably, because the following holds:

$$\forall v_1 v_2 v_3 v_4 v_5\ \exists v_6 :\ v_1 = 3\ \wedge\ v_2 \leqslant 3\ \wedge\ v_3 = 5\ \wedge\ v_4 - v_2 \leqslant 2\ \wedge\ v_5 = 7 \wedge v_1 \leqslant \ldots \leqslant v_5\ \Longrightarrow\ v_6 - v_4 \leqslant 2\ \wedge\ v_1 \leqslant v_6 \dots v_5 \leqslant v_6$$

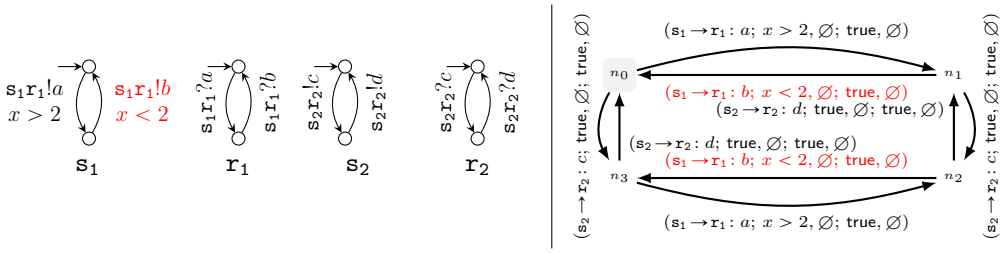
Notice that the resets of clock y (recorded by subtracting v_2 and v_4 in the formula above) allow $\mathtt{r}$ to receive message c before *absolute* time 7. If we modified the example by removing the second reset of y in machine $\mathtt{r}$, then the system would not be IE because message c would be expected before *absolute* time 5, while c can only be sent at time 7. In fact, the RHS of the implication above would become: $v_6 - v_2 \leqslant 2\ \wedge\ v_1 \leqslant v_6 \dots v_5 \leqslant v_6$.

(2) The second example shows a system of three machines, which violates IE (Def. 12).



If participant $\mathtt{s}$ does not send b before time 2, then message c (sent by $\mathtt{p}$), will never be received. This system is *not* IE because there is no path from n_0 that is RPE for $\{\mathtt{s}, \mathtt{r}, \mathtt{p}\}$. The only transition that is PE from n_0 is the loop on n_0 (which does not involve $\mathtt{p}$).

(3) The third example shows that IE captures a "global" notion of progress (i.e., *all* participants must able to proceed). Consider the system of four machines below:



this system is *not* IE. Indeed, although there is one RPE path outgoing node n_1 (machines $\mathtt{s}_2$ and $\mathtt{r}_2$ can continue interacting), there is no path that is RPE for *all* participants $\{\mathtt{s}_1, \mathtt{r}_1, \mathtt{s}_2, \mathtt{r}_2\}$. Observe that $\mathtt{s}_1$ is stuck in n_1, as the transition with label $\mathtt{s}_1\mathtt{r}_1!b(x < 2, \{x\})$ can never be fired by $\mathtt{s}_1$, i.e., $\forall v_0 \exists v_1 :\ v_0 > 2 \implies v_0 \leqslant v_1 < 2$ does not hold.

▸ **Theorem 13** (Progress). *Suppose S is multiparty compatible (Def. 4) and interaction enabling (Def. 12). (1) Then S satisfies the progress property. (2) For all $s = (\vec{q}; \vec{w}; \nu) \in RS(S)$, if there is $\mathtt{p} \in \mathcal{P}$ such that $q_{\mathtt{p}}$ is not final, then there is s' such that $s \to s'$.*

▸ **Theorem 14** (Decidability). *Interaction enabling (Def. 12) is decidable.*

The decidability of Def. 12 relies on the fact that the logic used in Def. 11 forms a subset of the Presburger arithmetic, which is decidable; and that it is enough to check *finite* paths in $STS(S)$. The complexity of the decision procedure is mostly affected by the enumeration of paths in $STS(S)$ (which can be reduced via partial order reduction techniques) and the satisfiability of Presburger formulae (which can be relegated to an SMT solver).

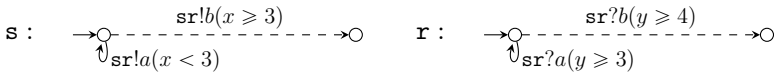
5 Non-Zenoness and Eventual Reception in CTAs

In the presence of time constraints, one needs to make sure that some participant's only possible way forward is not by firing actions at increasingly short intervals of time, i.e., by zeno behaviours. This is a common requirement in real-time systems [2], and it is justified by the assumption that *"any physical process, no matter how fast, cannot be infinitely fast"* [21].

In order to identify zeno behaviours in our systems, we assume without loss of generality that there is a special clock $\hat{x} \in \mathcal{X}$ which is *never* reset, i.e., for all $\mathtt{p} \in \mathcal{P}$ and all $(q, \ell, q') \in \delta_{\mathtt{p}} :\ \hat{x} \notin \mathsf{reset}(\ell)$. Hence, $\hat{x}$ keeps the absolute time since the beginning of the interactions. Let $s = (\vec{q}; \vec{w}; \nu)$ be a configuration of a system S, s is a ***zeno configuration*** if there exists $t \in \mathbb{R}_{\geqslant 0}$ such that for all $s' = (\vec{q'}; \vec{w'}; \nu')$, $s \to^* s'$ implies $\nu'(\hat{x}) < t$ and $s' \to s''$, for some s''.

▸ **Definition 15** (Non-zeno system). S is *non-zeno* (NZ) if $\forall s \in RS(S)$, s is not a zeno configuration. ◇

The following example shows that a zeno configuration may still occur in systems that are multiparty compatible and interaction enabling.



The system above (*ignoring the dashed transitions*) satisfies MC and IE, e.g., $\forall v_0\, \exists v_1 :\ v_0 < 3 \implies v_1 \geqslant 3 \wedge v_0 \leqslant v_1$, but is *not* NZ. Because of the upper bound $x < 3$ and the fact that x is not reset in the loop, machine $\mathtt{s}$ has to produce an *infinite* number of (send) actions in

a *finite* amount of time (3 time units). A dramatic consequence of this zeno behaviour is that machine `r` will never be able to consume any message a due to the fact that constraint $y \geqslant 3$ will never be satisfied (cf. Def. 9). This system violates *eventual reception*, a property which guarantees that every message that is sent is eventually received. Formally, a system S satisfies ***eventual reception*** (ER) if for all $s = (\vec{q}; \vec{w}; \nu) \in RS(S)$, if $w_{\mathtt{sr}} \in a\mathbb{A}^*$, then $s \longrightarrow^* \xrightarrow{\mathtt{sr}?a(g,\lambda)}$.

The system above (*considering the dashed transitions*) is NZ and satisfies ER: the dashed transitions offer an 'escape' from zeno-only behaviours where time can elapse and thus allow machine `r` to consume any messages that were sent. Observe that in general NZ alone is not sufficient to guarantee ER. However, ER is guaranteed for systems which validate all the condition presented in this paper, see Theorem 19 below.

The example also shows a fundamental difference between CTAs and models with synchronous communications, such as Networks of Timed Automata (NTAs) [2]. The work in [7] shows that it is sufficient that one machine in each loop of an NTA satisfies non-zenoness for the whole system to be non-zeno. This is not generally true for CTAs. In the example above (*ignoring the dashed transitions*), time cannot diverge despite the machine on the right being non-zeno.

Checking non-zenoness Now we give a condition on $STS(S)$ that, together with MC, guarantees non-zenoness. A walk in $STS(S)$ is an alternating sequence $n_1 \cdot e_1 \cdot n_2 \cdots e_{k-1} \cdot n_k$ such that $n_i \overset{e_i}{\hookrightarrow} n_{i+1}$ for all $1 \leqslant i < k$. We let ω range over walks in $STS(S)$. A walk is *elementary* if $(n_i \cdot e_i) \neq (n_j \cdot e_j)$ for all $1 \leqslant i \neq j < k$. A (elementary) cycle in $STS(S)$ is a (elementary) walk $n_1 \cdot e_1 \cdot n_2 \cdots e_{k-1} \cdot n_k$ such that $n_1 = n_k$.

Given guard g and clock x, we say that g *is an upper bound* for x, written g is UB for x, if there is a sub-term $x \leqslant c$ in g (not under a negation) or a sub-term $c \leqslant x$ under a negation. We say that g *is strictly positive*, written g is SP, if for all clocks $x \in \mathsf{fc}(g)$ and for all sub-terms in g of the form $x \leqslant c$ (not under negation) or $c \leqslant x$ (under negation), $c \in \mathbb{Q}_{>0}$.

▸ **Definition 16** (Cycle enabling (CE))**.** System S is cycle enabling (CE) if for each elementary cycle ω in $STS(S)$, and for each clock x such that there is $(\mathtt{s} \to \mathtt{r} : a; g_{\mathtt{s}}, \lambda_{\mathtt{s}}; g_{\mathtt{r}}, \lambda_{\mathtt{r}})$ in ω and $g_{\mathtt{s}}$ is UB for x, the following holds, either

1. there are (i) $(\mathtt{p} \to \mathtt{q} : b; g_{\mathtt{p}}, \lambda_{\mathtt{p}}; g_{\mathtt{q}}, \lambda_{\mathtt{q}})$ in ω s.t. $x \in \lambda_{\mathtt{p}} \cup \lambda_{\mathtt{q}}$, and (ii) $(\mathtt{p}' \to \mathtt{q}' : b'; g_{\mathtt{p}'}, \lambda_{\mathtt{p}'}; g_{\mathtt{q}'}, \lambda_{\mathtt{q}'})$ in ω s.t. $g_{\mathtt{p}'}$ is SP; or
2. for each $(n_i \cdot e \cdot n_{i+1})$ in ω, there is $n' \neq n_i \in N$ and $e' \neq e \in E$ such that $\mathsf{id}(e) = \mathsf{id}(e')$, $n_i \overset{e'}{\hookrightarrow} n'$, and (n_i, e') is PE for $\mathsf{sid}(e')$ ◇

Condition (1) adapts *structural non-zenoness* from [22] to CTAs by requiring that: (i) each x is reset in ω, and (ii) it is possible to let some time elapse at each iteration. Condition (2) requires that the "escape" event e', leading to a different node n', can *always* be taken. Our running example satisfies CE (Def. 16); $STS(S_{\mathrm{ST}})$ has one (elementary) cycle in which two clocks have an upper bound: clock y satisfies (1) since it is reset and the guards have upper bounds strictly greater than 0 in the cycle; clock y' satisfies (2) since there is an escape event, $e' = (\mathtt{W} \to \mathtt{A} : \mathtt{end}; y < 1, \varnothing; z = 1, \{z\})$, which is PE for `W`.

▸ **Theorem 17** (Non-zenoness)**.** *If S is MC and CE, then S is non-zeno.*

▸ **Theorem 18** (Decidability)**.** *Cycle enabling (Def. 16) is decidable.*

▸ **Theorem 19** (Eventual reception)**.** *If S is MC, IE, and CE, then S satisfies ER.*

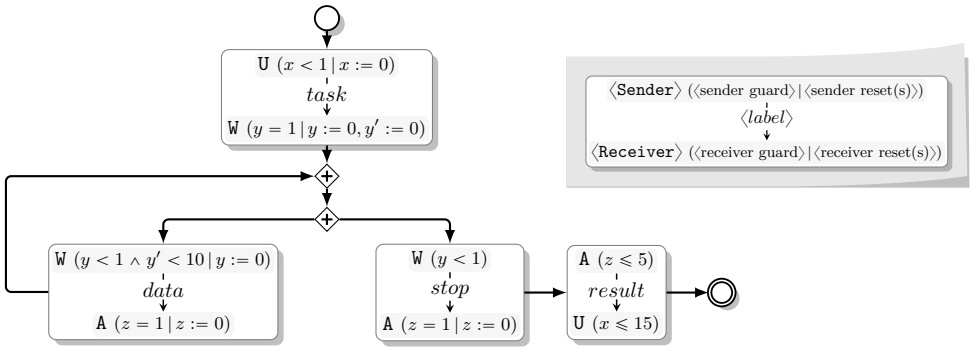


Figure 5 Timed choreography for the Scheduled Task Protocol (S_{ST})

6 Applications and Implementation

Constructing global specifications Our theory can be easily applied and integrated with other works, to construct sound (i.e., satisfying safety, progress, and non-zenoness) timed global specifications, such as (syntactic) multiparty session types [6, 16], or graphical choreographies [8, 13, 19]. Thanks to Theorem 7, we can build on the algorithm in [14] to construct (syntactic) timed global types from CTAs. In Appendix [3], we give the formal definitions of the adaptation of the algorithm in [14]. Given an MC system S our algorithm generates a timed global type [6] equivalent to the original system S (i.e., its projections are timed bisimilar to those of S). This implies that if S is IE (resp. CE) then the constructed timed global type will also enjoy progress (resp. non-zenoness). Similarly, building on the algorithm in [19], we obtain a *graphical* representation reminiscent of BPMN Choreographies, see [8, 19]. When applied to the Scheduled Task Protocol, the algorithm adapted from [19] produces the choreography in Fig. 5; giving a much clearer specification for S_{ST}.

Implementation To assess the applicability and cost of our theory, we have integrated our theory into the tool first introduced in [19], which builds graphical choreographies from CFSMs. Our tool [3] (implemented in Haskell and using Z3) takes as input a textual representation of CTAs on which each condition (MC, IE, and CE) is checked for, and produces an equivalent choreography (such as the one in Fig. 5). The results of our experiments (executed on a Intel i7 computer, with 16GB of RAM) are below; where $|\mathcal{P}|$ is the number of machines, and $|N|$ (resp. $|\hookrightarrow|$) is the number of nodes (resp. transitions) in $STS(S)$.

S	$\lvert\mathcal{P}\rvert$	$\lvert N\rvert$	$\lvert\hookrightarrow\rvert$	MC	IE	CE	s		$\lvert\mathcal{P}\rvert$	$\lvert N\rvert$	$\lvert\hookrightarrow\rvert$	MC	IE	CE	s
Running Example	3	4	4	✓	✓	✓	0.41	×4	12	256	1024	✓	✓	✓	28.49
Bargain	3	5	5	✓	✓	✓	0.44	×2	6	25	50	✓	✓	✓	12.30
Temp. calculation [6]	3	6	6	✓	✓	✓	0.45	×2	6	36	72	✓	✓	✓	9.24
Word Count [20]	3	6	6	✓	✓	✓	0.41	×2	6	36	72	✓	✓	✓	8.63
ATM (Template) [11]	3	9	8	✓	✓	✓	0.36	×3	9	729	1944	✓	✓	✓	94.01
ATM (Instance) [11]	3	9	8	✓	✓	✓	0.53	×3	9	729	1944	✓	✓	✓	96.09
Consumer-Producer [11]	2	1	1	✓	✓	✓	0.16	×5	10	1	5	✓	✓	✓	43.19
Fischers Mutual Excl. [5]	2	4	3	✓	✓	✓	0.21	×4	8	256	768	✓	✓	✓	3.19

Most of the protocols are taken from the literature and all are checked within a minute on average. For the sake of space, we have used small examples throughout the paper, however our benchmarks include bigger protocols (up-to 12 machines), which have comparable size with those we encountered through our collaboration with Cognizant [19, 23]. Since the size of the STS is the most critical parameter for scalability, we have tested systems consisting of the parallel composition of several instances of a protocol. For instance, *Running Example* ×4 is the parallel composition of four instances of S_{ST}, cf. Fig. 1.

7 Conclusions and Related Work

Our results are summarised in the table below. Multiparty compatibility (MC) gives (*i*) an equivalence between an MC system and a system consisting of the projections of its *STS*; and (*ii*) a sufficient condition for *safety*. MC and interaction enabling (IE) form a sufficient condition for *progress*; while MC and cycle enabling (CE) form a sufficient condition for *non-zenoness* (NZ). Together, MC, IE, and CE ensure safety, progress, NZ, and *eventual reception* (ER).

Property	$S \sim (STS(S)\upharpoonright_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$	Safety	Progress	Non-Zeno	ER
MC (Def. 4)	✓	✓	✗	✗	✗
MC+IE (Def. 12)	✓	✓	✓	✗	✗
MC+CE (Def. 16)	✓	✓	✗	✓	✗
MC+IE+CE	✓	✓	✓	✓	✓

Multiparty session types The work in [6] studies a typing system for a timed π-calculus using timed global types. A class of CTAs which are safe and have progress is given in [6] via projection of (well-formed) timed global types onto timed local types (which correspond to deterministic, non-mixed, and directed CTAs). Well-formedness yields conditions on CTAs that are more restrictive than the ones given in this paper. For instance, the system in Fig. 1, which is safe and enjoys progress, is ruled out by the conditions in [6]. In addition, this paper gives sufficient conditions for CTAs to belong to the class of safe CTAs with progress, which was left as an open problem in [6]. The construction of timed global types from either local types or CTAs is not addressed in [6]. Recently, [4] introduced a compliance and sub-typing relation for *binary* timed session types *without queues* (synchronous communication semantics). The existing works for constructing global specifications from local specifications [14, 18, 19] only apply to *untimed* models. Our conditions (IE and CE) are given independently of the definition of MC. The use of a more general notion of MC, as the one given in [19], would allow us to lift the assumptions that the machines are directed and have no mixed states (cf. § 2). Hence, we could capture more general timed choreographies.

Reachability and decidability When extending NTAs [2] with unbounded channels, reachability is no longer decidable in general [17]. Existing work tackles undecidability by restricting the network topologies [12, 17] or the channel size [1]. We give general (w.r.t. topology and channel size) decidable conditions ensuring that a configuration violating safety, progress, or NZ will not be reached. Observe that the scenario in Fig. 1 would be ruled out in [17] (its topology is not a polyforest) and in [1] ($w_{\mathtt{WA}}$ is unbounded). Our conditions are based, instead, on the conversation structures, which also enable the construction of global specifications.

Non-zeno conditions In § 5 we set the conditions for time divergence, by ruling out specifications in which the only way forward is a zeno behaviour. This condition is called time progress in [2] and it is built-in in the definition of runs of a TA. Several conditions have been proposed to ensure absence of non-zeno behaviours in TAs: some, e.g., [21], do not allow any zeno execution, and some, e.g., [7], and this work (cf. Def. 15), ensure that there is always a non-zeno way forward. The condition in [7] can be checked with a simple form of reachability analysis which introduced the notion of 'escape' from a zeno loop, which we also use. [7, 21] consider Networks of TAs (NTAs), which do not feature asynchrony nor unbounded channels.

Acknowledgements We would like to thank the ZDLC team at Cognizant for their stimulating conversations and Dominic Orchard for some (very useful) Haskell tips. This

work is partially supported by UK EPSRC projects EP/K034413/1, EP/K011715/1, and EP/L00058X/1; and by EU 7FP project under grant agreement 612985 (UPSCALE).

References

1 S. Akshay, Paul Gastin, Madhavan Mukund, and K. Narayan Kumar. Model checking time-constrained scenario-based specifications. In *FSTTCS*, volume 8 of *LIPIcs*, pages 204–215, 2010.

2 Rajeev Alur and David L. Dill. A theory of timed automata. *TCS*, 126:183–235, 1994.

3 Webpage of this paper, 2015. `http://www.doc.ic.ac.uk/~jlange/cta/`.

4 Massimo Bartoletti, Tiziana Cimoli, Maurizio Murgia, Alessandro Sebastian Podda, and Livio Pompianu. Compliance and subtyping in timed session types. In *FORTE*, volume 9039 of *LNCS*, pages 161–177. Springer, 2015.

5 Johan Bengtsson et al. Uppaal - a tool suite for automatic verification of real-time systems. In *Hybrid Systems III*, volume 1066 of *LNCS*, pages 232–243. Springer, 1996.

6 Laura Bocchi, Weizhen Yang, and Nobuko Yoshida. Timed multiparty session types. In *CONCUR*, volume 8704 of *LNCS*, pages 419–434. Springer, 2014.

7 Howard Bowman and Rodolfo Gómez. How to stop time stopping. *FAC*, 18(4):459–493, 2006.

8 BPMN 2.0 Choreography, 2012. `http://en.bpmn-community.org/tutorials/34/`.

9 Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *JACM*, 30(2):323–342, 1983.

10 Gérard Cécé and Alain Finkel. Verification of programs with half-duplex communication. *I&C*, 202(2):166–190, 2005.

11 Prakash Chandrasekaran and Madhavan Mukund. Matching scenarios with timing constraints. In *FORMATS*, volume 4202 of *LNCS*, pages 98–112. Springer, 2006.

12 Lorenzo Clemente, Frédéric Herbreteau, Amelie Stainer, and Grégoire Sutre. Reachability of communicating timed processes. In *FOSSACS*, volume 7794 of *LNCS*, pages 81–96. Springer, 2013.

13 Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty session types meet communicating automata. In *ESOP*, volume 7211 of *LNCS*, pages 194–213. Springer, 2012.

14 Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types. In *ICALP (2)*, volume 7966 of *LNCS*, pages 174–186. Springer, 2013.

15 Uno Holmer, Kim Guldstrand Larsen, and Wang Yi. Deciding properties of regular real time processes. In *CAV*, volume 575 of *LNCS*, pages 443–453. Springer, 1991.

16 Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *POPL*, pages 273–284. ACM, 2008.

17 Pavel Krcál and Wang Yi. Communicating timed automata: The more synchronous, the more difficult to verify. In *CAV*, volume 4144 of *LNCS*, pages 249–262, 2006.

18 Julien Lange and Emilio Tuosto. Synthesising Choreographies from Local Session Types. In *CONCUR*, volume 7454 of *LNCS*, pages 225–239. Springer, 2012.

19 Julien Lange, Emilio Tuosto, and Nobuko Yoshida. From communicating machines to graphical choreographies. In *POPL*, pages 221–232, 2015.

20 Rumyana Neykova, Laura Bocchi, and Nobuko Yoshida. Timed runtime monitoring for multiparty conversations. In *BEAT*, volume 162 of *EPTCS*, pages 19–26, 2014.

21 Stavros Tripakis. Verifying progress in timed systems. In *Formal Methods for Real-Time and Probabilistic Systems*, volume 1601 of *LNCS*, pages 299–314. Springer, 1999.

22 Stavros Tripakis, Sergio Yovine, and Ahmed Bouajjani. Checking timed büchi automata emptiness efficiently. *Formal Methods in System Design*, 26(3):267–292, 2005.

23 Zero Deviation Lifecycle. `http://www.zdlc.co`.