Meeting Deadlines Together

Laura Bocchi¹, Julien Lange², and Nobuko Yoshida²

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 (TAs) 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-zenoess 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

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 timeframes. 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-zenoess [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]).
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. $WA_{data}(y < 1 \land y' < 10, y := 0)$.

**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 \( \pi \)-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 \( p, q, r, a \), etc.). Let \( \mathcal{A} \) be a finite alphabet of messages ranged over by \( a, b, c \), etc. The set of finite words on \( \mathcal{A} \) is denoted by \( \mathcal{A}^* \), \( w w' \) is the concatenation of \( w \) and \( w' \), and \( \varepsilon \) is the empty word (overloaded on any alphabet). The set of channels is \( \mathcal{C} = \{ pq \mid p, q \in \mathcal{P} \text{ and } p \neq q \} \). Given a (finite) set of clocks \( \mathcal{X} \) (ranged over by \( x, y, \), etc.), the set of actions (ranged over by \( \ell \) ) is \( \text{Act}_\mathcal{X} \defeq \mathcal{C} \times \{ !, ? \} \times \mathcal{A} \times \Phi(\mathcal{X}) \times 2^\mathcal{X} \), and the set of guards (ranged over by \( g \) ) \( \Phi(\mathcal{X}) \) is

\[
g := \text{true} \mid x \leq c \mid c \leq x \mid \neg g \mid g_1 \land g_2 \]

where \( c \) ranges over constants in \( \mathbb{Q}_{\geq 0} \), and from which we derive the usual abbreviations. We write \( \text{fc}(g) \) for the set of clocks in \( g \) and \( \text{sr} a(g, \lambda) \) or \( \text{sr} ? a(g, \lambda) \) for an element of \( \text{Act}_\mathcal{X} \).

Action \( \text{sr} a(g, \lambda) \) says that \( a \) sends a message \( a \) to \( r \), provided that guard \( g \) is satisfied, and resets the clocks in \( \lambda \subseteq \mathcal{X} \); the dual receiving action is \( \text{sr} ? a(g, \lambda) \) for an element of \( \text{Act}_\mathcal{X} \). Action \( \text{sr} ? a(g, \lambda) \) says that \( a \) sends a message \( a \) to \( r \), provided that guard \( g \) is satisfied, and resets the clocks in \( \lambda \subseteq \mathcal{X} \); the dual receiving action is \( \text{sr} ? a(g, \lambda) \) for an element of \( \text{Act}_\mathcal{X} \). Action \( \text{sr} ? a(g, \lambda) \) says that \( a \) sends a message \( a \) to \( r \), provided that guard \( g \) is satisfied, and resets the clocks in \( \lambda \subseteq \mathcal{X} \); the dual receiving action is \( \text{sr} ? a(g, \lambda) \) for an element of \( \text{Act}_\mathcal{X} \).

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 \( \delta \subseteq Q \times \text{Act}_\mathcal{X} \times Q \) 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 \( \ell, \ell' \in \text{Act}_\mathcal{X} \), if \( (q, \ell, q'), (q, \ell', q'') \in \delta \) and \( \text{msg}(\ell) = \text{msg}(\ell') \), 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 \( \text{sr} a(g, \lambda) \) (resp. \( \text{sr} ? a(g, \lambda) \)); 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 $\nu : X \to \mathbb{R}_{\geq 0}$ of the clocks in $X$, $\nu \models g$ denotes that the guard $g$ is satisfied by $\nu$ and $\lambda(\nu)$ denotes a valuation where all clocks in $\lambda$ are set to 0 (reset) and clocks not in $\lambda$ keep their values in $\nu$.

**Definition 1** (Timed communicating system). A timed communicating system (or *system*), is a tuple $S = (M_p)_{p \in P}$ where each $M_p = (Q_p, q_0, X_p, \delta_p)$ is a CTA and for all $p \neq q \in P : X_p \cap X_q = \emptyset$. A configuration of $S$ is a triple $s = (q; \bar{w}; \nu)$ where: $q = (q_p)_{p \in P}$ is the control state and $q_p \in Q_p$ is the local state of machine $M_p$; $\bar{w} = (w_{pq})_{p \in P} \in \mathbb{R}^*$ is a vector of queues; $\nu : \bigcup_{p \in P} X_p \to \mathbb{R}_{\geq 0}$ is a clock valuation. The initial configuration of $S$ is $s_0 = (q_0; \bar{0}; \nu_0)$ with $q_0 = (q_p)_{p \in P}$, $\bar{0}$ being the vector of empty queues, and $\nu_0(x) = 0$ for each clock $x \in \bigcup_{p \in P} X_p$.

Hereafter, we fix a machine $M_p = (Q_p, q_0, X_p, \delta_p)$ for each participant $p \in P$ (assuming that $\forall p \in P : (q, \ell, q') \in \delta_p \implies \text{subj}(\ell) = p$), and let $S = (M_p)_{p \in P}$ be the corresponding system. We write $X$ for $\bigcup_{p \in P} X_p$ and $\nu + t$ for the valuation mapping each $x \in X$ to $\nu(x) + t$.

The definition below is from [17, Definition 1], omitting internal transitions.

**Definition 2** (Reachable configuration). Configuration $s' = (\bar{q}'; \bar{w}'; \nu'; t)$ is reachable from configuration $s = (\bar{q}; \bar{w}; \nu; t)$ by firing the transition $\alpha$, written $s \xrightarrow{\Delta, s} s'$ (or $s \to s'$ when the label is immaterial), if either:

1. $(q_\alpha, \text{sr} a(g, \lambda), q'_\alpha) \in \delta_\alpha$ and (a) $q'_\alpha = q_p$ for all $p \neq \alpha$; (b) $w_{\bar{q}' \alpha} = w_{\bar{q} \alpha} a$ and $w_{q'_\alpha} = w_{q_\alpha}$ for all $p \neq \alpha$; (c) $\nu' = \nu(\alpha)$; (d) $\alpha \to sa(g, \lambda)$, and $\nu \models g$;
2. $(q_\alpha, \text{sr} a(g, \lambda), q'_\alpha) \in \delta_\alpha$ and (a) $q'_\alpha = q_p$ for all $p \neq \alpha$; (b) $w_{\bar{q}' \alpha} = w_{\bar{q} \alpha} a$ and $w_{q'_\alpha} = w_{q_\alpha}$ for all $p \neq \alpha$; (c) $\nu' = \nu(\alpha)$; (d) $\alpha \to sa(g, \lambda)$ and $\nu \models g$; or
3. $\alpha = t \in \mathbb{R}_{\geq 0}$, $\nu' = \nu + t$, $w_{q'_\alpha} = w_{q_\alpha}$ for all $p \in C$, and $q'_\alpha = q_p$ for all $p \notin P$.

We let $\rho$ range over sequences of labels $a_1 \cdots a_k$ and write $\xrightarrow{*}$ for the reflexive transitive closure of $\to$. The reachability set of $S$ is $RS(S) = \{ s_0 \xrightarrow{\Delta, s} s \}$. Condition (1) allows a machine $a$ to put a message $a$ on queue $\text{sr} a$, if the time constraints in $g$ are satisfied by $\nu$; dually, (2) allows $r$ 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-zero properties in § 4.

Let $n$ range over vectors of local states; and $\ell$ range over events, which are elements of the set $C \times \mathbb{N} \times \Phi(X) \times 2^X \times \Phi(X) \times 2^X$, and write $(a \to r : a ; g_\alpha, \lambda_\alpha ; g_r, \lambda_r)$ for the event in which $a$ sends message $a$ to $r$, with $a$ (resp. $r$) having guard $g_\alpha$ (resp. $g_r$) and resets $\lambda_\alpha$ (resp. $\lambda_r$). 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 $(N, n_0, \to, E)$ such that:

- $\to$ is the relation defined as $n \xrightarrow{e} n'$ with $e = (a \to r : a ; g_\alpha, \lambda_\alpha ; g_r, \lambda_r)$ if $n = q; n' = \bar{q}'$ and $q_\alpha = q_\alpha a_\alpha(q, \lambda_\alpha), q_\alpha = q_\alpha a_\alpha(q, \lambda_\alpha), q_{\bar{q}'}$, and $\forall p \in P \setminus \{s, r\} : q_p = q_p'$ (write $\to$ when $e$ is unimportant and $\to^*$ for the reflexive and transitive closure of $\to$);
- $n_0 = q_0$ is the initial node; $N = \{ n \mid n_0 \to^* n \}$ is the (finite) set of nodes; and $E = \{ e \mid \exists n, n' \in N \text{ and } n \xrightarrow{e} n' \}$ is the set of events.
The STS of the Scheduled Task Protocol (Sₜₛₜ) is given in Fig. 2; essentially, it models all the synchronous executions of Sₜₛₜ. In the following, we fix STS(Sₜₛₜ) = (N, n₀, →, E).

Given e = (s → r: a; gₛ, λₛ; gᵣ, λᵣ), we define sid(e) = s, rid(e) = r, and id(e) = {s, r}. The projection of e on p (written eₚ) is given by: (s → r: a; gₛ, λₛ; gᵣ, λᵣ)ₚ = srลา(gₛ, λₛ); (s → r: a; gₛ, λₛ; gᵣ, λᵣ)₁ = srα(gₛ, λₛ); and (s → r: a; gₛ, λₛ; gᵣ, λᵣ)₂ₚ = ε, if p ∉ {s, r}. We extend ϕₚ to sequences of events and, given n ∈ N, define ids(n) = ⋃{id(e) | n →* ε}.

> Definition 4 (Multiparty compatibility (MC)). System S is multiparty compatible if for all p ∈ P, for all q ∈ Qₚ, and for all n = q ∈ N, if qₚ = q, then
1. if q is a sending state, then ∀(q, ℓ, q′) ∈ δₚ : ∃ϕ, ∃e ∈ E : n ↾ₚ e = ℓ ∧ ϕₚ = ε;
2. if q is a receiving state, then ∀(q, ℓ, q′) ∈ δₚ : ∃ϕ, ∃e ∈ E : n ↾ₚ e = ℓ ∧ ϕₚ = ε.

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 Sₜₛₜ, in Fig. 1, is multiparty compatible.

Observe that STS(Sₜₛₜ) and MC do not address time constraints. In fact, STS(Sₜₛₜ) 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 asynchronou execution can be mapped to a path in STS(Sₜₛₜ), i.e., it can be simulated by STS(Sₜₛₜ).

We recall two types of errors from the CFSM model, which are ruled out by MC also in the timed setting. Let s = (qₛ; w; ν) be a configuration of a system S; s is a deadlock configuration [10, Def. 12] if w = ε, there is r ∈ P such that qᵣ is a receiving state, and for every p ∈ P, qₚ is a receiving or final state, i.e., all machines are blocked waiting for messages; and s is an orphan message configuration if all qₚ ∈ q are final but w ≠ ε, 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 ∈ 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(Sₜₛₜ)₁ₚ of a synchronous transition system STS(Sₜₛₜ) on a machine p is given by substituting each event e ∈ E with its projection eₚ, then minimising the automaton w.r.t. language equivalence. For example, the projections of STS(Sₜₛₜ) onto U, W, and A are isomorphic to the system Sₜₛₜ in Fig. 1. Below ~ denotes the standard timed bisimulation [15].
Theorem 7 (Equivalence). If $S = (M_p)_{p \in P}$ is MC then $S \sim (STS(S))_{p \in P}$.

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 $s = (\tilde{q}, \tilde{w}; \nu) \in RS(S)$; $s$ is an unsuccessful reception configuration if there exists $r \in P$ such that $q_r$ is a receiving state, and for all $(q_x, \mathbf{s}r?a(y, \lambda), q'_x) \in \delta_r$ either (i) $w_{aq} \neq \varepsilon$ and $w_{aq} \neq a\tilde{w}$ or (ii) $\forall t \in \mathbb{R}_{>0} : \nu + t \not\models g$ (i.e., $r$ 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 $s \in P$ such that $q_s$ is a sending state, and $(q_x, \mathbf{s}r!a(g, \lambda), q'_x) \in \delta_x$ implies that $\forall t \in \mathbb{R}_{>0} : \nu + t \not\models g$ (i.e., $s$ 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. $\alpha = t \in \mathbb{R}_{>0}$, $\nu' = \nu + t$, $\forall p q \in C : w'_{pq} = w_{pq}$, and $\forall p \in P : q'_p = q_p$ and
   a. $q_p$ sending $\implies \exists (q_f, t, q'_f) \in \delta_p : \exists t' \in \mathbb{R}_{>0} : \nu' + t' \models guard(t)$
   b. $\forall (q_p, \mathbf{s}p?a(y, \lambda), q'_p) \in \delta_p : (w_{ap} \in a\tilde{w}) \implies \exists t' \in \mathbb{R}_{>0} : \nu' + t' \models g$

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:

<table>
<thead>
<tr>
<th>s:</th>
<th>$\mathbf{s}r!a(x &lt; 3)$</th>
<th>$\mathbf{s}r?b(x \leq 2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>r:</td>
<td>$\mathbf{s}r?a(y = 4)$</td>
<td>$\mathbf{s}r!b(y = 5)$</td>
</tr>
</tbody>
</table>

Consider configuration $((q_0, q_2); \varepsilon; \nu_0)$ in which $s$ 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 $s$ from $q_0$ can be fired. Consider now configuration $((q_1, q_2); \tilde{w}; \nu)$ with $w_{aq} = a$ and $\nu(x) = \nu(y) = 3.5$. Condition (3b) rules out a time transition with $t = 1$. Indeed, even if $r$ has a transition whose guard will be enabled after time $\nu(y) + 1 = 4.5$, i.e.,
\((q_2, \text{sr} \equiv (y = 5), q_3)\), this transition cannot be fired due to the content of queue \(w_{\text{sr}} \neq b \cdot a^*\); on the other hand transition \((q_2, \text{sr} \equiv a(y = 4), q_3)\) 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 \(\ell\) is progress enabling (Def. 11) which ensures that, for all possible past clock valuations, there exists a future time where the guard of \(\ell\) 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:

\[
\text{nodes}(e_1 \cdots e_k) = e_1[e_{\text{sid}(e_1)}] \cdots e_k[e_{\text{sid}(e_k)}], \quad (k \geq 0)
\]

Remarkably, given a path \(\varphi\) and two actions \(\ell_i\) and \(\ell_j\) in \(\text{nodes}(\varphi)\), \(i < j\) does not imply that there is a causal dependency between \(\ell_i\) and \(\ell_j\). For instance, in

\[
\text{nodes}(\varphi) = \text{sr}!a(x < 10, \emptyset) \cdot \text{sr}?a(10 \leq y, \emptyset) \cdot \text{sp}!a(x < 10, \emptyset) \cdot \text{sp}?a(10 \leq z, \emptyset)
\]

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

The graph of dependencies of an action \(\ell_k\) in a sequence of actions \(\ell_1 \cdots \ell_k\) (Def. 10 below) gives an abstraction of all actions on which \(\ell_k\) 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 \(\text{dep}(\varepsilon; \ell) \overset{\text{def}}{=} \emptyset\) and

\[
\text{dep}(\rho \cdot \ell_1; \ell_2) \overset{\text{def}}{=} \begin{cases} 
\{(\ell_1, \ell_2)\} & \text{if } \ell_1 = \text{sr}!a(g_1, \lambda_1) \land \ell_2 = \text{sr}^?a(g_2, \lambda_2) \\
\{(\ell_1, \ell_2)\} \cup \text{dep}(\rho; \ell_1) & \text{if } \text{subj}(\ell_1) = \text{subj}(\ell_2) \\
\text{dep}(\rho; \ell_2) & \text{otherwise}
\end{cases}
\]

The graph of dependencies of \(\rho = \ell_1 \cdots \ell_k\) \((k > 0)\), written \(DG(\rho)\), is the graph \((D, A)\) s.t. \(A = \text{dep}(\ell_1 \cdots \ell_{k-1}; \ell_k) \cup \{(\ell_i, \ell_k) | 1 \leq i < k\}\) and \(D = \{\ell_r \neq \ell_k \mid \exists (\ell_i, \ell_j) \in A \land r \in \{i, j\}\}\).\(^1\)\(^1\)

\(DG(\ell_1 \cdots \ell_k)\) 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 \(\ell_k\)). In Fig. 3 (in solid black), we give the graph of dependencies of \(W\cdot \text{data}(z = 1, \{z\})\) in the sequence \(\rho_{st}\), corresponding to an execution of the Scheduled Task Protocol.

\(^1\) For the sake of presentation, we write \(\ell_i\) for the node \((i, \ell_i)\) in \(D\) where \(\ell_i\) is an action in \(\rho\) and \(i\) is its position in \(\rho\). This guarantees that each element in \(\rho\) is assigned a unique node in \(D\).
Meeting Deadlines Together

\[ (1) \forall \text{task}(x < 1, \{x\}) \rightarrow (2) \forall \text{task}(y = 1, \{y, y'\}) \rightarrow (3) \forall \text{data}(y < 1 \land y' < 10, \{y\}) \rightarrow (4) \forall \text{data}(z = 1, \{z\}) \]

\[ \rho_{ST} = \forall \text{task}(\ldots) \forall \text{task}(\ldots) \forall \text{data}(\ldots) \forall \text{data}(\ldots) \forall \text{data}(\ldots) \]

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

\[ \text{id}(\rho) \overset{\text{def}}{=} \{ i \mid \ell_i \in D \} \quad W^2_x(\rho) \overset{\text{def}}{=} \left\{ v_i - v_j \quad \text{if } 0 \leq j = \max \{ j < i \mid \ell_j \in D \land x \in \text{reset}(\ell_j) \} \right\} \]

\[ \text{allpast}(\rho) \overset{\text{def}}{=} \bigwedge_{i \in \text{id}(\rho)} \text{absolute}_p(\ell_i) \quad \text{elapse}(\rho) \overset{\text{def}}{=} \bigwedge_{(\ell_i, \ell_j) \in A} v_i \leq v_j \quad \text{absolute}_p(\ell_i) \overset{\text{def}}{=} \text{guard}(\ell_i) \{ x \mapsto W^i_x(\rho) \}_{x \in X} \]

\[ \text{Figure 4} \quad \text{Functions on graphs of dependencies, where } DG(\rho) = (D, A) \]

Given a graph of dependencies \( DG(\rho) \), 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 \( DG(\rho) = (D, A) \). Below, we illustrate how they behave using \( DG(\rho_{ST}) \) in Fig. 3. First, we transform the guard of an action \( \ell_i \) such that its solutions are the possible absolute times (i.e., from the initial configuration of the system) in which one may execute \( \ell_i \) (taking into account the last reset of each clock in \( \rho \)). In our example, we have:

\[ \text{absolute}_{\rho_{ST}}(\ell_5) = v_5 - v_3 < 1 \land v_5 - v_2 < 10 \quad \text{with } \ell_5 = \text{data}(y < 1 \land y' < 10, \{y\}) \]

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 \( DG(\rho) \), by (i) recording the indices of all the actions on which \( \ell_k \) depends (\text{id}(\rho)); (ii) taking the conjunction of all constraints in absolute time (\text{allpast}(\rho)); and (iii) recording the fact that time never decreases between two causally dependent actions (\text{elapse}(\rho)). Taking the dependencies for \( \rho_{ST} \) in Fig. 3, we have:

\[ \text{allpast}(\rho_{ST}) = v_1 < 1 \land v_2 = 1 \land (v_3 - v_2 < 1 \land v_3 - v_2 < 10) \land v_4 = 1 \land (v_5 - v_3 < 1 \land v_5 - v_2 < 10) \]

\[ \text{elapse}(\rho_{ST}) = v_1 \leq v_2 \land v_2 \leq v_3 \land v_3 \leq v_4 \land v_3 \leq v_5 \quad \text{id}(\rho_{ST}) = \{1, 2, 3, 4, 5\} \]

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.

\[ \text{Definition 11} \quad \text{(Progress enabling (PE)).} \quad \text{A pair } (n, \varepsilon) \text{ is progress enabling (PE) for } p \in \text{id}(\varepsilon) \text{ if for all paths } \varphi \text{ such that } n_0 \xrightarrow{\varphi} n, \text{ letting:} \]

\[ \rho = \begin{cases} \text{nodes}(\varphi \cdot e) & \text{if } p = \text{id}(\varepsilon) \\ \text{nodes}(\varphi) \cdot e_{|\text{id}(\varepsilon)} & \text{otherwise} \end{cases} \]

and \( k = |\rho|, \ell_k = e_{|\rho}, \bar{v} = \{ v_i \mid i \in \text{id}(\rho) \}; \) the following holds

\[ \forall \bar{v} \exists v_k : \text{allpast}(\rho) \land \text{elapse}(\rho) \implies \text{absolute}_p(\ell_k) \land \bigwedge_{v_i \in \bar{v}} v_i \leq v_k \]
A pair \((n, \varphi)\) is recursively progress enabling (RPE) for \(P \subseteq P'\) if \(\varphi = \varepsilon\) and \(P = \emptyset\); or if \((n, e)\) is PE for \(\text{id}(e)\) and for \(\text{rid}(e)\) and \((n', \varphi')\) is RPE for \(P' \setminus \text{id}(e)\) with \(\varphi = e \cdot \varphi'\) and \(n \nleq n'\).

Given a node \(n\) and an event \(e\) in \(\text{STS}(S)\), and a participant \(p\), the above definition ensures that for all possible past clock valuations, there exists a future time where participant \(p\) has the possibility to execute action \(e_p\). For instance, the pair \(((U_1, W_1, A_0), (W \rightarrow k \cdot \text{data}; y < 1 \land y' < 10, \{y\}; z = 1, \{z\}))\) is PE for \(A\), notably because the following holds:

\[
\forall v_1 \ldots v_5 \exists v_6 : \text{allpast}(\rho_{st}) \land \text{elapse}(\rho_{st}) = (v_6 - v_4) = 1 \land v_1 \leq v_6 \ldots v_5 \leq v_6
\]

Below, Def. 11 is used in \(\text{STS}(S)\) 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 \(\varphi\) such that \(n \nleq s\) and \((n, e \cdot \varphi)\) is RPE for \(\text{id}(n)\);
2. For all \(e \in E\) such that \(n \nleq n'\), \((n, e)\) is PE for \(\text{rid}(e)\), and \(n'\) is IE.

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

Def. 12 recursively checks the nodes of \(\text{STS}(S)\) (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 a 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.

\[
\begin{array}{c}
\text{s: } \text{sr}!a(x = 3) \quad \text{sr}!c(x = 7) \\
\text{r: } \text{sr}?a(y \leq 3, y := 0) \quad \text{sr}?c(y \leq 2) \\
\end{array}
\]

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

\[
\forall v_1 v_2 v_3 v_4 v_5 \exists v_6 : v_1 = 3 \land v_2 \leq 3 \land v_3 = 5 \land v_4 - v_2 \leq 2 \land v_5 = 7 \land \\
v_1 \leq \ldots \leq v_5 \implies v_6 - v_4 \leq 2 \land v_1 \leq v_6 \ldots v_5 \leq v_6
\]

Notice that the resets of clock \(y\) (recorded by subtracting \(v_2\) and \(v_4\) in the formula above) allow \(r\) to receive message \(c\) before absolute time 7. If we modified the example by removing the second reset of \(y\) in machine \(r\), 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: \(v_6 - v_2 \leq 2 \land v_1 \leq v_6 \ldots v_5 \leq v_6\).

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

\[
\begin{array}{c}
\text{s: } \text{sr}!a \quad \text{sr}!b \quad \text{ps}!c \\
\text{r: } \text{sr}!b \quad \text{sr}!c \\
\text{p: } (\text{sr} \rightarrow k : x < 2, \{x\}; \text{true}, \{x\}) \quad (\text{ps} \rightarrow a, \{x\}; \text{true}, \{x\}) \quad (\text{ps} \rightarrow b, \{x\}; \text{true}, \{x\})
\end{array}
\]

If participant \(s\) does not send \(b\) before time 2, then message \(c\) (sent by \(p\)), will never be received. This system is not IE because there is no path from \(n_0\) that is RPE for \(\{s, r, p\}\). The only transition that is PE from \(n_0\) is the loop on \(n_0\) (which does not involve \(p\)).
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 $s = (q, \vec{w}; \nu) \in RS(S)$, if there is $p \in \mathcal{P}$ such that $q_p$ is not final, then there is $s'$ such that $s \leadsto s'$.

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 $\hat{x} \in X$ which is never reset, i.e., for all $p \in \mathcal{P}$ and all $(q, \ell, q') \in \delta_p : \hat{x} \notin \text{reset}(\ell)$. Hence, $\hat{x}$ keeps the absolute time since the beginning of the interactions. Let $s = (q; \vec{w}; \nu)$ be a configuration of a system $S$, $s$ is a zeno configuration if there exists $t \in \mathbb{R}_{\geq 0}$ such that for all $s' = (q'; \vec{w}'; \nu')$, $s \leadsto^* s'$ implies $\nu'(\hat{x}) < t$ and $s' \leadsto s''$, 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., $\forall v_0 \exists v_1 : v_0 < 3 \implies v_1 \geq 3$ and $v_0 < 3 \implies v_1 \geq 3$ and $v_0 \leq v_1$, but is not NZ. Because of the upper bound $x < 3$ and the fact that $x$ is not reset in the loop, machine $\mathfrak{s}$ 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 \( y \geq 3 \) 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 \( s = (\bar{q}_i \bar{w}; \nu) \in RS(S) \), if \( w_{\bar{w}} \in aA^* \), then \( s \xrightarrow{e} \exists r(a, \bar{y}, \bar{z}) \).

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 \( n_i \xrightarrow{e_i} n_{i+1} \) for all \( 1 \leq i < k \). We let \( \omega \) range over walks in \( STS(S) \). A walk is elementary if \( (n_i \cdot e_i) \neq (n_j \cdot e_j) \) for all \( 1 \leq i \neq j < k \). 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 \( x \leq c \) in \( g \) (not under a negation) or a sub-term \( c \leq x \) under a negation. We say that \( g \) is strictly positive, written \( g \) is SP, if for all clocks \( x \in fc(g) \) and for all sub-terms in \( g \) of the form \( x \leq c \) (not under negation) or \( c \leq x \) (under negation), \( c \in \mathbb{Q}_{>0} \).

#### Definition 16 (Cycle enabling (CE))

System \( S \) is cycle enabling (CE) if for each elementary cycle \( \omega \) in \( STS(S) \), and for each clock \( x \) such that there is \( (s \rightarrow r: a; g_x, \lambda_x; g_r, \lambda_r) \) in \( \omega \) and \( g_x \) is UB for \( x \), the following holds, either

1. there are \((i) \ (p \rightarrow q : b; g_p, \lambda_p; g_q, \lambda_q) \) in \( \omega \) s.t. \( x \in \lambda_p \cup \lambda_q \), and \((ii) \ (p' \rightarrow q' : b'; g_p', \lambda_p'; g_q', \lambda_q') \) in \( \omega \) s.t. \( g_p' \) is SP; or
2. for each \( (n_i \cdot e \cdot n_{i+1}) \) in \( \omega \), there is \( n' \neq n_i \in N \) and \( e' \neq e \in E \) such that \( \text{id}(e) = \text{id}(e') \), \( n_i \xrightarrow{e} n' \), and \( (n_i, e') \) is PE for \( \text{sid}(e') \).

Condition (1) adapts structural non-zenoness from [22] to CTAs by requiring that: \( (i) \) each \( x \) is reset in \( \omega \), 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); \( STS(S_{ct}) \) 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, \( e' = (\bar{w} \rightarrow A: \text{end}; y < 1, \bar{Q}; z = 1, \{z\}) \), which is PE for \( \bar{w} \).

#### 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.
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 an Intel i7 computer, with 16GB of RAM) are below; where $|P|$ is the number of machines, and $|N|$ (resp. $|\rightarrow|$) is the number of nodes (resp. transitions) in $STS(S)$.

| $S$ | $|P|$ | $|N|$ | $|\rightarrow|$ | MC | IE | CE | $s$ |
|------|------|------|---------|-----|-----|-----|-----|
| Running Example | 3 | 4 | 4 | ✓ | ✓ | ✓ | ✓ | 0.44 | ×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 | ✓ | ✓ | ✓ | 94.04 |
| ATM (Instance) [11] | 3 | 9 | 8 | ✓ | ✓ | ✓ | ✓ | 0.53 | ×3 | 9 | 729 | 1944 | ✓ | ✓ | ✓ | 86.09 |
| 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).

<table>
<thead>
<tr>
<th>Property</th>
<th>$S \sim (STS(S_{i})<em>{i \in P})</em>{P}$</th>
<th>Safety</th>
<th>Progress</th>
<th>Non-Zeno</th>
<th>ER</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC (Def. 4)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MC+IE (Def. 12)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MC+CE (Def. 16)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MC+IE+CE</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Multiparty session types The work in [6] studies a typing system for a timed $\pi$-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] ($w_{M}$ 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