Compositional Asynchronous Timed Refinement

Massimo Bartoletti[1], Laura Bocchi[2], and Maurizio Murgia[2]

[1] Università degli Studi di Cagliari, Cagliari, Italy
[2] University of Kent, Canterbury, UK

Abstract. We develop a theory of refinement for timed asynchronous systems, in the setting of Communicating Timed Automata. Our notion of refinement is compositional (i.e., the refinement of a system is obtained by refining its components independently) and decidable. We establish a decidable condition under which our refinement preserves behavioural properties of systems, such as their global and local progress. Our theory provides guidelines on how to implement timed protocols using the real-time primitives of programming languages. We validate our theory though a series of experiments, supported by an open-source tool which implements our verification techniques.

1 Introduction

Formal reasoning of real-time computing systems is supported by well-established theories and frameworks based, for instance, on timed automata [1, 28, 40]. In the standard theory of timed automata, communication between components is *synchronous*: a component can send a message only when its counterpart is ready to receive it. However, in many concrete scenarios, such as web-based systems, communications are *asynchronous* and often implemented through middlewares supporting FIFO messaging [2, 38]. These systems can be modelled as Communicating Timed Automata (CTA) [26], an extension of timed automata with asynchronous communication. Asynchrony comes at the price of an increased complexity: interesting behavioural properties, starting from reachability, become undecidable in the general case, both in the timed [19] and in the untimed [11] setting. Several works propose restrictions of the general model, or sound approximate techniques for the verification of CTAs [8, 19]. These works leave one important problem unexplored: *the link between asynchronous timed models and their implementations*.

The relationship between models at different levels of abstraction is usually expressed as a *refinement* relation. Refinements have been used, for instance, to create abstract models which enhance effectiveness of verification techniques (e.g., abstraction refinement [22, 39], time-wise refinement [36]), or to concretize abstract models into implementations [18, 20]. Existing notions of refinement between timed models are based on *synchronous* communications [4, 14, 23, 29]. Asynchronous refinement has been investigated in the *untimed* setting, under the name of subtyping between session types [5, 17, 21, 30–32]. To our knowledge, no notion of refinement has been yet investigated in the *asynchronous timed* setting. The only work that studies a notion close to that of refinement is [9], which focusses on the relation between timed multiparty session types and their implementations (processes in an extended π-calculus). The work in [9] has two main limitations. First, their model is not as general as CTAs: in particular, it does not allow states with both sending and receiving outgoing transitions (so-called *mixed states*). Mixed states are crucial to capture common programming patterns like *timeouts* [34] (e.g. a server waiting for a message that sends a timeout notification after a deadline). Some programming languages provide specific primitives to express timeouts, e.g. the receive/after construct of Erlang [3]. The second limitation of [9] is that its calculus is very simple (actions are statically set to happen at precise points in time), and cannot express common real-world blocking receive primitives (with or without timeout) that listen on a channel until a message is available.

To be usable in practice, a theory of refinements for timed asynchronous systems should support real-world programming patterns (e.g., timeouts à la Erlang) and primitives, and feature *decidable* notions of refinement. Further, refinement should be *compositional* (i.e. a system can be refined by refining its single components, independently), and preserve desirable properties (e.g., progress) of the abstract system being refined.

1.1 Contributions

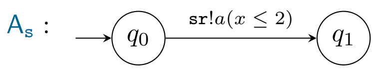
We develop a theory of asynchronous timed refinement for CTAs. Our main purpose is to study preservation of behavioural properties under refinement, focussing on two aspects: *timed behaviour* and *progress*. The former kind of preservation, akin timed similarity [15], ensures that the observable behaviour of the concrete system can be simulated by the abstract system. The latter requires that refinement does not introduce deadlocks, either *globally* (i.e., the whole system gets stuck), or *locally* (i.e., a single CTA gets stuck, although the whole system may still proceed).

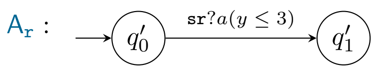
Refinement We introduce a new refinement relation. This relation is decidable, and compositional (i.e., the refinement of a system of CTAs is obtained by refining its components, independently) so to enable modular development of systems of CTA. Our refinement operates on time constraints: it may restrict time constraints of any (send or receive) action. Further, for receive actions, the refined constraint must include the deadline of the original constraint (i.e., the receiving component must be ready to receive until the very last moment allowed of the original constraint). We anticipate that this way of refining receive actions is key to obtain positive results.

Positive results Our main positive result (Theorem 3) is a decidable condition called *Locally Latest-Enabled Send Preservation* (LLESP) ensuring preservation of timed behaviour, global and local progress under our refinement. To assess the practicality of our theory we develop a tool and apply, with the help of the tool, instances of our refinement to a portfolio of case studies from literature. Our refinement and the LLESP condition naturally apply to most of the case studies in our portfolio. In Section 6 we show how our tool and results can be used to guide the implementation of timed protocols with the Go programming language.

Negative results We also considered other refinement strategies: (i) arbitrary restriction of constraints of send and receive actions (similarly to [9]), and (ii) asymmetric restriction where constraints of send actions may be restricted, while those of receive actions may be relaxed (this is the natural timed extension of the subtyping relation in [21]). Besides being relevant in literature, (i) and (ii) ultimately reflect common programming practices: (i) caters for e.g. non-blocking receive with constraint reduced to an arbitrary point in the model's guard, and (ii) caters e.g. for blocking receive without timeouts. For (i) and (ii) we only have negative results, even when LLESP holds, and even if mixed states are forbidden (Fact 4). Our negative results have a practical relevance on their own: they establish that if you implement a CTA as described above, you may have no guarantees of behaviour/progress preservation.

A new semantics for CTAs The original semantics for CTAs [26] was introduced for studying decidability issues for timed languages. To achieve such goals, [26] adopts the usual language-based approach of computability theory: (1) it always allows time to elapse, even when this prevents the system from performing any available action, and (2) it rules out 'bad' executions *a posteriori*, e.g. only keeping executions that terminate in final states. Consider, for example, the following two CTAs:





The CTA $\mathsf{A_s}$ models a sender $\mathtt{s}$ who wants to deliver a message a to a receiver $\mathtt{r}$. The guard $x \leq 2$ is a time constraint, stating that the message must be sent within 2 time units. The receiver wants to read the message a from $\mathtt{s}$ within 3 time units. In [26], a possible (partial) computation of the system $(\mathsf{A_s}, \mathsf{A_r})$ would be the following:

$$\gamma_0 = ((q_0, q'_0), (\varepsilon, \varepsilon), \{x, y \mapsto 0\}) \xrightarrow{5} ((q_0, q'_0), (\varepsilon, \varepsilon), \{x, y \mapsto 5\})$$

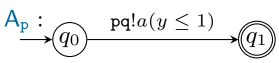
The tuple γ_0 at the LHS of the arrow is the initial *configuration* of the system, where both CTAs are in their initial states; the pair $(\varepsilon, \varepsilon)$ means that the communication queues between $\mathtt{r}$ and $\mathtt{s}$ are empty (in both directions); the last component means that the clocks x and y are set to 0. The label on the arrow represents a delay of 5 time units. This computation does *not* correspond to any reasonable behaviour of the given protocol. What we would expect, is that the send action is performed *before* the deadline expires.

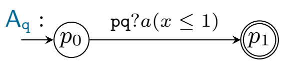
To capture this intuition, we introduce a semantics of CTAs, requiring that the elapsing of time does not disable the send action in $\mathsf{A_s}$. Namely, we can procrastinate the send for 2 time units; then, time cannot delay further, and the only possible action is the send:

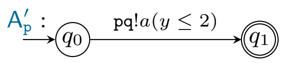
$$\gamma_0 \xrightarrow{2} ((q_0, q'_0), (\varepsilon, \varepsilon), \{x, y \mapsto 2\}) \xrightarrow{\mathtt{sr}!a} ((q_1, q'_0), (a, \varepsilon), \{x, y \mapsto 2\})$$

In Theorem 1 we prove that our semantics enjoys a form of *persistency*: if at least one receive action is guaranteed to be enabled in the future (i.e. a message is ready in its queue and its time constraint is still satisfiable now or at some point in the future) then time passing has to preserve at least one these guaranteed actions. Instead, time passing can disable all send actions, but *only if* it preserves at least one guaranteed receive.

It is well known that language-based approaches are not well suited to deal with concurrency issues like those addressed in this paper. To see this, consider the following CTAs, where the states with a double circle are accepting:







The systems $S = (\mathsf{A_p}, \mathsf{A_q})$ and $S' = (\mathsf{A'_p}, \mathsf{A_q})$ accept the same language, namely $t_0\ \mathtt{pq}!a\ t_1\ \mathtt{pq}?a\ t_2$ with $t_0 + t_1 \leq 1$ and $t_2 \in \mathbb{R}_{\geq 0}$. So, the language-based approach does not capture a fundamental difference between S and S': S enjoys progress, while

```
receive  {s,a1}  -> Body1
                   ...
         {s,ak}  -> Bodyk
after    10      -> p!b
```

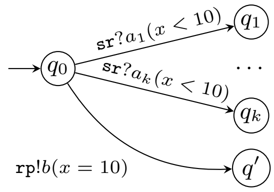


Fig. 1: The `receive`/`after` pattern of Erlang (left), and the corresponding CTA (right).

S' does not. Our approach to defining CTA semantics provides us with a natural way to reason on standard properties of protocols like progress, and to compare behaviours using e.g., (bi)simulation.

Our semantics allows for CTAs with mixed states, by suitably extending the one in [8] (where, instead, mixed states are forbidden). As said above, mixed states enable useful programming patterns. Consider e.g. the code snippet in Figure 1 (left), showing a typical use of the `receive`/`after` construct in Erlang. The snippet attempts to receive a message matching one of the patterns `{s,a1}`,...,`{s,ak}`, where `s` represents the identifier of the sender, and $\mathtt{a}_1$,...,$\mathtt{a}_k$ are the message labels. If no such message arrives within 10 ms, then the process in the `after` branch is executed, sending immediately a message `b` to process `p`. This behaviour can be modelled by the CTA in Figure 1 (right), where q_0 is mixed. Our semantics properly models the intended behaviour of timeouts.

Urgency. Another practical aspect that is not well captured by the existing semantics of CTAs [8, 26] is *urgency*. Indeed, while in known semantics [8, 26] receive actions can be deferred, the receive primitives of mainstream programming languages unblock as soon as the expected message is available. These primitives include e.g. the non-blocking (resp. blocking) `WaitFreeReadQueue.read()` (resp. `WaitFreeReadQueue.waitForData()`) of Real-Time Java [13], and `receive`...`after` in Erlang, just to mention some. The consequence of analysing a system only on the basis of a non-urgent semantics may be a (risky) inconsistence between the behaviour of the model and that of the final implementation.

To correctly characterise urgent behaviour, we introduce a second semantics for CTAs (Definition 13), that is *urgent* in what it forces receive actions as soon as the expected message is available. Theorem 5 shows that the urgent semantics preserves the behaviour of the non-urgent. However, the urgent semantics does *not* enjoy the preservation results of Theorem 3. Still, it is possible to obtain preservation under refinement by combining Theorem 3 with Theorem 6. More specifically, the latter ensures that, if a system of CTAs enjoys progress in the non-urgent semantics, then it will also enjoy progress in the urgent one, under a minor and common assumption on the syntax of time constraints. So, one can use Theorem 3 to obtain a refinement which guarantees progress (in the non-urgent semantics), and then lift the preservation result to the urgent semantics through Theorem 6.

Overall, our theory suggests that, despite the differences between semantics of CTAs and programming languages, verification techniques based on CTAs can be helpful for implementing distributed timed programs.

Additional material The proofs of our statements and additional examples are in the appendix. We make available a tool (the link is omitted to preserve anonymity) which implements our verification techniques, and our suite of experiments.

2 Communicating Timed Automata

We assume a finite set $\mathcal{P}$ of *participants*, ranged over by $\mathtt{p}$, $\mathtt{q}$, $\mathtt{r}$, $\mathtt{s}$, $\ldots$, and a finite set $\mathbb{A}$ of *messages*, ranged over by a, b, $\ldots$ We define the set $\mathcal{C}$ of *channels* as $\mathcal{C} = \{\mathtt{pq} \mid \mathtt{p}, \mathtt{q} \in \mathcal{P} \text{ and } \mathtt{p} \neq \mathtt{q}\}$. We denote with $\mathbb{A}^*$ the set of finite words on $\mathbb{A}$ (ranged over by w, w', $\ldots$), with ww' the concatenation of w and w', and with ε the empty word.

Clocks and guards. Given a (finite) set of *clocks* X (ranged over by $x, y, \ldots$), we define the set Δ_X of *guards* over X (ranged over by $\delta, \delta', \ldots$) as follows:

$$\delta ::= \mathit{true} \mid x \leq c \mid c \leq x \mid \neg\delta \mid \delta_1 \wedge \delta_2 \qquad (c \in \mathbb{Q}_{\geq 0})$$

Clock valuations. We denote with $\mathbb{V} = X \to \mathbb{R}_{\geq 0}$ the set of *clock valuations* on X. Given $t \in \mathbb{R}_{\geq 0}$, $\lambda \subseteq C$, and a clock valuation ν, we define the clock valuations: (i) $\nu + t$ as the valuation mapping each $x \in X$ to $\nu(x) + t$; (ii) $\lambda(\nu)$ as the valuation which resets to 0 all the clocks in $\lambda \subseteq X$, and preserves to $\nu(x)$ the values of the other clocks $x \notin \lambda$. Furthermore, given a set K of clock valuations, we define the *past* of K as the set of clock valuations $\downarrow K = \{\nu \mid \exists \delta \geq 0 : \nu + \delta \in K\}$.

Semantics of guards. We define the function $[\![\cdot]\!] : \Delta_X \to \wp(\mathbb{V})$ as follows:

$$[\![\mathit{true}]\!] = \mathbb{V} \qquad [\![x \leq c]\!] = \{\nu \mid \nu(x) \leq c\} \qquad [\![\delta_1 \wedge \delta_2]\!] = [\![\delta_1]\!] \cap [\![\delta_2]\!]$$
$$[\![\neg\delta]\!] = \mathbb{V} \setminus [\![\delta]\!] \qquad [\![c \leq x]\!] = \{\nu \mid c \leq \nu(x)\}$$

Actions. We denote with $\mathsf{Act} = \mathcal{C} \times \{!, ?\} \times \mathbb{A}$ the set of *untimed actions*, and with $\mathsf{TAct}_X = \mathsf{Act} \times \Delta_X \times 2^X$ the set of *timed actions* (ranged over by $\ell, \ell', \ldots$). A (timed) action of the form $\mathtt{sr}!a(\delta, \lambda)$ is a *sending action*: it models a participant $\mathtt{s}$ who sends to $\mathtt{r}$ a message a, provided that the guard δ is satisfied. After the message is sent, the clocks in $\lambda \subseteq X$ are reset. An action of the form $\mathtt{sr}?a(\delta, \lambda)$ is a *receiving action*: if the guard δ is satisfied, $\mathtt{r}$ receives a message a sent by $\mathtt{s}$, and resets the clocks in $\lambda \subseteq X$ afterwards. Given $\ell = \mathtt{pr}!a(\delta, \lambda)$ or $\ell = \mathtt{qp}?a(\delta, \lambda)$, we define: (i) $\mathsf{msg}(\ell) = a$, (ii) $\mathsf{guard}(\ell) = \delta$, (iii) $\mathsf{reset}(\ell) = \lambda$, (iv) $\mathsf{subj}(\ell) = \mathtt{p}$, and (v) $\mathsf{act}(\ell)$ is $\mathtt{pr}!$ (in the first case) or $\mathtt{qp}?$ (in the second case). We omit δ if true, and λ if empty.

CTAs. A *communicating timed automaton* A is a tuple of the form (Q, q_0, X, E), where Q is a finite set of *states*, $q_0 \in Q$ is the initial state, X is a set of clocks, and $E \subseteq Q \times \mathsf{TAct}_X \times Q$ is a set of *edges*, such that the set $\bigcup \{\mathsf{subj}(e) \mid e \in E\}$ is a singleton, that we denote as $\mathsf{subj}(\mathsf{A})$. We write $q \xrightarrow{\ell} q'$ when $(q, \ell, q') \in E$. We say that a state is *sending* (resp. *receiving*) if it has some outgoing sending (resp. receiving) edge. We say that A has *mixed states* if it has some state which is both sending and receiving. We say that a state q is *final* if there exist no ℓ and q' such that $(q, \ell, q') \in E$.

Systems of CTAs. *Systems* of CTAs (ranged over by $S, S', \ldots$) are sequences $(\mathsf{A}_\mathtt{p})_{\mathtt{p} \in \mathcal{P}}$, where each $\mathsf{A}_\mathtt{p} = (Q_\mathtt{p}, q_{0\mathtt{p}}, X_\mathtt{p}, E_\mathtt{p})$ is a CTA, and (i) for all $\mathtt{p} \in \mathcal{P}$, $\mathsf{subj}(\mathsf{A}_\mathtt{p}) = \mathtt{p}$; (ii) for all $\mathtt{p} \neq \mathtt{q} \in \mathcal{P}$, $X_\mathtt{p} \cap X_\mathtt{q} = \emptyset = Q_\mathtt{p} \cap Q_\mathtt{q}$.

Configurations. CTAs in a system communicate via asynchronous message passing on FIFO queues, one for each channel. For each couple of participants $(\mathtt{p}, \mathtt{q})$ there are two channels, $\mathtt{pq}$ and $\mathtt{qp}$, with corresponding queues $w_{\mathtt{pq}}$ (containing the messages from $\mathtt{p}$ to $\mathtt{q}$) and $w_{\mathtt{qp}}$ (messages from $\mathtt{q}$ to $\mathtt{p}$). The state of a system S, or *configuration*, is a triple $\gamma = (\boldsymbol{q}, \boldsymbol{w}, \nu)$ where: (i) $\boldsymbol{q} = (q_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$ is the sequence of the current states of all the CTAs in S; (ii) $\boldsymbol{w} = (w_{\mathtt{pq}})_{\mathtt{pq}\in\mathcal{C}}$ with $w_{\mathtt{pq}} \in \mathbb{A}^*$ is a sequence of queues; (iii) $\nu : \bigcup_{\mathtt{p}\in\mathcal{P}} X_{\mathtt{p}} \to \mathbb{R}_{\geq 0}$ is a *clock valuation*. The initial configuration of S is $\gamma_0 = (\boldsymbol{q_0}, \boldsymbol{\varepsilon}, \nu_0)$ where $\boldsymbol{q_0} = (q_{0\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$, $\boldsymbol{\varepsilon}$ is the sequence of empty queues, and $\nu_0(x) = 0$ for each $x \in \bigcup_{\mathtt{p}\in\mathcal{P}} X_{\mathtt{p}}$. We say that $(\boldsymbol{q}, \boldsymbol{w}, \nu)$ is *final* when all $q \in \boldsymbol{q}$ are final.

We introduce a new semantics of systems of CTAs, that generalises Definition 9 in [8] to account for mixed states. To this aim, we first give a few auxiliary definitions. We start by defining when a guard δ' is satisfiable *later* than δ in a clock valuation.

Definition 1 (Later satisfiability). *For all ν, we define the relation $\leq_\nu$ as:*

$$\delta \leq_\nu \delta' \quad\iff\quad \forall t \in \mathbb{R}_{\geq 0} :\ \nu + t \in [\![\delta]\!] \implies \exists t' \geq t :\ \nu + t' \in [\![\delta']\!]$$

The following lemma states some basic properties of later satisfiability.

Lemma 1. *The relation $\leq_\nu$ is a total preorder, for all clock valuations ν. Further, for all guards δ, δ', for all $t \in \mathbb{R}_{\geq 0}$, and $c, d \in \mathbb{Q}_{\geq 0}$: (a) $(x \leq c) \leq_\nu (x \leq c{+}d) \iff d \geq 0$; (b) $\delta \wedge \delta' \leq_\nu \delta'$; (c) $\delta \leq_\nu \delta' \implies \delta \leq_{\nu+t} \delta'$.*

Definition 2 (FE, LE, ND). *In a configuration $(\boldsymbol{q}, \boldsymbol{w}, \nu)$, we say that an edge $(q, \ell, q') \in E_{\mathtt{p}}$ is future-enabled (FE), latest-enabled (LE), or non-deferrable (ND) iff, respectively:*

- $\exists t \in \mathbb{R}_{\geq 0}.\ \nu + t \in [\![\mathsf{guard}(\ell)]\!]$ *(FE)*
- $\forall \ell', q'' : (q, \ell', q'') \in E_{\mathtt{p}} \implies \mathsf{guard}(\ell') \leq_\nu \mathsf{guard}(\ell)$*, and* (q, ℓ, q') *is FE* *(LE)*
- $\exists \mathtt{s}, w' : \mathsf{act}(\ell) = \mathtt{sp}?$*,* $w_{\mathtt{sp}} = \mathsf{msg}(\ell) w'$ *and* (q, ℓ, q') *is FE* *(ND)*

An edge is FE when its guards can be satisfied at some time in the future; it is LE when no other edge (starting from the same state) can be satisfied later than it. The type of action (send or receive) and the co-party involved are immaterial to determine FE and LE edges. A receiving edge is ND when the expected message is already at the head of the queue, and there is some time in the future when it can be read. Note that an edge $(q, \mathtt{sp}?a(\delta, \lambda), q')$ is deferrable when $w_{\mathtt{sp}} = bw'$ and $a \neq b$ (i.e., the first message in the queue is not the expected one). Non-deferrability is not affected by the presence of send actions in the outgoing edges. It could happen that two receiving edges in a CTA are ND, if both expected messages are in the head of each respective queue.

The semantics of systems of CTAs is defined in terms of a timed transition system (TLTS) between configurations.

Definition 3 (Semantics of systems). *Given a system S, we define the TLTS $[\![S]\!] = (Q, \mathcal{L}, \rightarrow)$, where (i) Q is the set of configurations of S, (ii) $\mathcal{L} = \mathsf{Act} \cup \mathbb{R}_{\geq 0}$, and (iii) $\gamma = (\boldsymbol{q}, \boldsymbol{w}, \nu) \xrightarrow{\alpha} (\boldsymbol{q'}, \boldsymbol{w'}, \nu') = \gamma'$ holds when one of the following rules apply:*

1. $\alpha = \mathtt{sr}!a$*,* $(q_{\mathtt{s}}, \alpha(\delta, \lambda), q'_{\mathtt{s}}) \in E_{\mathtt{s}}$*, 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)* $\nu' = \lambda(\nu)$ *and* $\nu \in [\![\delta]\!]$*;*

2. $\alpha = \mathtt{sr}?a$, $(q_{\mathtt{r}}, \alpha(\delta,\lambda), q'_{\mathtt{r}}) \in E_{\mathtt{r}}$, *and (a)* $q'_{\mathtt{p}} = q_{\mathtt{p}}$ *for all* $\mathtt{p} \neq \mathtt{r}$*; (b)* $w_{\mathtt{sr}} = a w'_{\mathtt{sr}}$ *and* $w'_{\mathtt{pq}} = w_{\mathtt{pq}}$ *for all* $\mathtt{pq} \neq \mathtt{sr}$*; (c)* $\nu' = \lambda(\nu)$ *and* $\nu \in [\![\delta]\!]$*;*
3. $\alpha = t \in \mathbb{R}_{\geq 0}$*, and (a)* $q'_{\mathtt{p}} = q_{\mathtt{p}}$ *for all* $\mathtt{p} \in \mathcal{P}$*; (b)* $w'_{\mathtt{pq}} = w_{\mathtt{pq}}$ *for all* $\mathtt{pq} \in \mathcal{C}$*; (c)* $\nu' = \nu + t$*; (d) for all* $\mathtt{p} \in \mathcal{P}$*, if some* sending *edge starting from* $q_{\mathtt{p}}$ *is LE in* γ*, then such edge is LE also in* γ'*; (e) for all* $\mathtt{p} \in \mathcal{P}$*, if some edge starting from* $q_{\mathtt{p}}$ *is ND in* γ*, then there exists an edge starting from* $q_{\mathtt{p}}$ *that is ND in* γ'*.*

We write $\gamma \to \gamma'$ *when* $\gamma \xrightarrow{\alpha} \gamma'$ *for some label* α*, and* $\gamma \xrightarrow{\alpha}$ *if* $\gamma \xrightarrow{\alpha} \gamma'$ *for some configuration* γ'*. We denote with* $\to^*$ *the reflexive and transitive closure of* $\to$*. We write* $\gamma \not\to_{\mathtt{p}}$ *when, in configuration* γ*, condition* (3) *is violated by the CTA* $\mathtt{p}$*.*

Rules (1), (2) and the first three items of (3) are adapted from [8]. In particular, (1) allows a CTA $\mathtt{s}$ to send a message a on channel $\mathtt{sr}$ if the time constraints in δ are satisfied by ν; dually, (2) allows $\mathtt{r}$ to consume a message from the channel, if δ is satisfied. In both rules, the clocks in λ are reset. Rule (3) models the elapsing of time. Items (a) and (b) require that states and queues are not affected by the passing of time, which is implemented by item (c). Items (d) and (e) put constraints on when time can pass. Condition (d) requires that time passing preserves LE sending edges: this means that if the current state of a CTA has the option to send a message (possibly in the future), time passing cannot prevent it to do so. Instead, condition (e) ensures that, if at least one of the expected messages is already at the head of a queue, time passing must still allow at least one of the messages already at the head of some queue to be received.

Our semantics (Definition 3) enjoys two classic properties [34] of timed systems.

Lemma 2. *The semantics of CTAs enjoys time determinism and time additivity [34].*

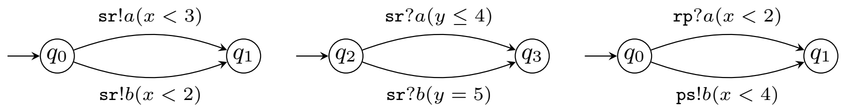
Our semantics does not, instead, enjoy *persistency* [34], because the passing of time can suppress the ability to perform some actions. However, it enjoys a weaker persistency property, stated by Theorem 1. More specifically, if a receive action is ND, then time passing cannot suppress *all* receive actions: at least a ND action (not necessarily the first one) always remains FE after a delay. Instead, time passing can disable all send actions, but *only if* it preserves at least a ND receive action.

Theorem 1 (Weak persistency). *For all configurations* γ, γ'*:*

$$\gamma \xrightarrow{t'} \xrightarrow{\mathtt{rp}?} \wedge \ \gamma \xrightarrow{t} \gamma' \implies \exists \gamma'', \mathtt{s}, t'' : \gamma' \xrightarrow{t''} \gamma'' \ \wedge \ (\gamma'' \xrightarrow{\mathtt{sp}?} \ \vee \ \gamma'' \not\to_{\mathtt{s}})$$
$$\gamma \xrightarrow{t'} \xrightarrow{\mathtt{pr}!} \wedge \ \gamma \xrightarrow{t} \gamma' \implies \exists \gamma'', \mathtt{s}, t'' : \gamma' \xrightarrow{t''} \gamma'' \ \wedge \ (\gamma'' \xrightarrow{\mathtt{ps}!} \ \vee \ \gamma'' \xrightarrow{\mathtt{sp}?} \ \vee \ \gamma'' \not\to_{\mathtt{s}})$$

Definition 4 below will be useful to reason on executions of systems.

Definition 4 (Maximal run). *A* run *of a system* S *starting from* γ *is a (possibly infinite) sequence* $\rho = \gamma_1 \xrightarrow{t_1} \gamma'_1 \xrightarrow{\alpha_1} \gamma_2 \xrightarrow{t_2} \cdots$ *with* $\gamma_1 = \gamma$ *and* $\alpha_i \in \mathsf{Act}$ *for all* i*. We omit the clause "starting from* s*" when* $\gamma = \gamma_0$*. We call* trace *the sequence* $t_1\, \alpha_1\, t_2 \cdots$*. For all* $n > 0$*, we define the partial functions:* $conf_n(\rho) = \gamma_n$*,* $delay_n(\rho) = t_n$*,* $act_n(\rho) = \alpha_n$*. We say that a run is* maximal *when it is infinite, or given its last element* γ_n *it never happens that* $\gamma_n \xrightarrow{t} \xrightarrow{\alpha}$*, for any* $t \in \mathbb{R}_{\geq 0}$ *and* $\alpha \in \mathsf{Act}$*.*



A_1 A_2 A_3

Fig. 2: A collection of CTAs, to illustrate the semantics of systems.

We show the peculiarities of our semantics through the CTAs in Figure 2. First, consider the system composed of A_1 and A_2. A possible maximal run of $(\mathsf{A}_1, \mathsf{A}_2)$ from the initial configuration $\gamma_0 = ((q_0, q_2), \boldsymbol{\varepsilon}, \nu_0)$ is the following:

$$\gamma_0 \xrightarrow{2} \gamma_1 = ((q_0, q_2), (\varepsilon, \varepsilon), \nu_0 + 2) \xrightarrow{\mathtt{sr}!a} \gamma_2 = ((q_1, q_2), (a, \varepsilon), \nu_0 + 2)$$
$$\xrightarrow{1.5} \gamma_3 = ((q_1, q_2), (a, \varepsilon), \nu_0 + 3.5) \xrightarrow{\mathtt{rs}?a} \gamma_4 = ((q_1, q_3), (\varepsilon, \varepsilon), \nu_0 + 3.5)$$

The first delay transition is possible because there are no ND edges in A_1 (both edges are sending), and the LE edge $(q_0, \mathtt{sr}!a(x < 3), q_1)$, continues to be LE in $\nu_0 + 2$; further, in A_2 there are no LE sending edges, and no ND edges (since the queue $\mathtt{sr}$ is empty). Note that condition (d) prevents γ_0 from making transitions with label $t \geq 3$, since $(q_0, \mathtt{sr}!a(x < 3), q_1)$ is LE in γ_0, but it is *not* LE in $\nu_0 + t$ if $t \geq 3$. The transition from γ_1 to γ_2 corresponds to a send action. The delay transition from γ_2 to γ_3 is possible because the state of A_1 is final, while the state q_2 of A_2 has a ND edge, $(q_2, \mathtt{sr}?a(y \leq 4), q_3)$, which is still ND at $\nu_0 + 3.5$. Note instead that condition (e) prevents γ_2 from making a transition with $t > 2$, because no edge is ND in $\nu_0 + 2 + t$ if $t > 2$. Indeed, the last moment when the edge $(q_2, \mathtt{sr}?a(y \leq 4), q_3)$ is FE is $y = 4$. Finally, the transition from γ_3 to γ_4 corresponds to a receive action.

The CTA A_3 has mixed states, with the send action enabled for longer than the receive action. We show the behaviour of A_3 (abstracting from its co-parties that, we assume, always allow delays e.g. have all guards set to *true*). This CTA has a LE sending action $(q_0, \mathtt{ps}!b(x < 4), q_1)$ in the initial configuration γ_0. Hence, condition (d) is satisfied in γ_0 iff the delay t is less than 4. Condition (e) is satisfied in γ_0, as there are no ND edges. When A_3 is at state q_0, with $w_{\mathtt{rp}} = a$ and $\nu(x) = 0$, the CTA allows a delay t iff $t < 2$: later, no edge would be ND, so (e) would be violated. If the message a is in the queue but it is too late to receive it (i.e., $\nu(x) \geq 2$), then the receive action would be deferrable, and so a delay would be allowed — if condition (d) is respected.

3 Compositional asynchronous timed refinement

In this section we introduce a decidable notion of refinement for systems of CTAs. In the general case (e.g. when refinements may arbitrarily alter the interaction structures), establishing if an *asynchronous* (FIFO-based) communication model is a refinement of another is undecidable, even in the untimed scenario [12, 27]. Our system refinement is defined *point-wise* on its CTAs. Point-wise refinement $\mathsf{A}' \sqsubseteq_1 \mathsf{A}$ only alters the guards, in the refined CTA A', while leaving the rest unchanged. The guards of A' — both in send and receive actions — must be narrower than those of A. Further, the guards in receive actions must have *the same past* in both CTAs.

Formally, to define the relation $\mathsf{A}' \sqsubseteq_1 \mathsf{A}$ we use *structure-preserving* functions that map the edges of A into those of A', preserving everything but the guards.

Definition 5 (Structure-preserving). *Let E, E' be sets of edges of CTAs. We say that a function $f : E \to E'$ is* structure-preserving *when, for all $(q, \ell, q') \in E$, $f(q, \ell, q') = (q, \ell', q')$ with $\mathsf{act}(\ell) = \mathsf{act}(\ell')$, $\mathsf{msg}(\ell) = \mathsf{msg}(\ell')$, and $\mathsf{reset}(\ell) = \mathsf{reset}(\ell')$.*

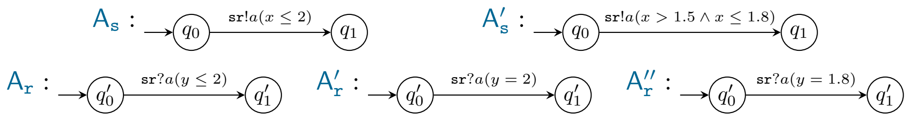
Definition 6 (Refinement). *Let $\mathsf{A} = (Q, q_0, X, E)$ and $\mathsf{A}' = (Q, q_0, X, E')$ be CTAs. The relation $\mathsf{A}' \sqsubseteq_1 \mathsf{A}$ holds whenever there exists a structure-preserving isomorphism $f : E \to E'$ such that, for all edges $(q, \ell, q') \in E$, if $f(q, \ell, q') = \ell'$, then:*
(a) $[\![\mathsf{guard}(\ell')]\!] \subseteq [\![\mathsf{guard}(\ell)]\!]$*;*
(b) *if (q, ℓ, q') is a receiving edge, then* $\downarrow [\![\mathsf{guard}(\ell')]\!] = \downarrow [\![\mathsf{guard}(\ell)]\!]$.

Condition (a) allows the guards of send/receive actions to be restricted. For receive actions, condition (b) requires restriction to preserve the final deadline.

System refinement reflects a modular engineering practice where parts of the system are implemented independently, without knowing how other parts are implemented.

Definition 7 (System Refinement). *Let $S = (\mathsf{A}_1, \ldots, \mathsf{A}_n)$, and let $S' = (\mathsf{A}'_1, \ldots, \mathsf{A}'_n)$. We write $S \sqsubseteq S'$ iff $\mathsf{A}_i \sqsubseteq_1 \mathsf{A}'_i$ for all $i \in 1 \ldots n$.*

Example 1. With the CTAs below, we have: $\mathsf{A}'_{\mathtt{s}} \sqsubseteq_1 \mathsf{A}_{\mathtt{s}}$, $\mathsf{A}'_{\mathtt{r}} \sqsubseteq_1, \mathsf{A}_{\mathtt{r}}$, and $\mathsf{A}''_{\mathtt{r}} \not\sqsubseteq_1 \mathsf{A}_{\mathtt{r}}$.



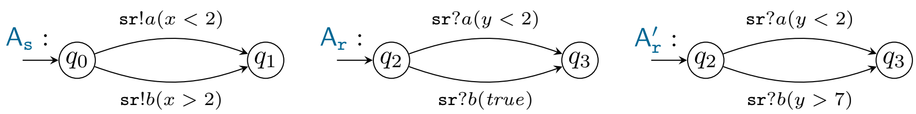
Theorem 2 establishes decidability of $\sqsubseteq_1$. This follows by the fact that CTAs have a finite number of states and that: (i) the function $\downarrow [\![\delta]\!]$ is computable, and the result can be represented as a guard [7, 24]; (ii) the relation $\subseteq$ between guards is computable.

Theorem 2. *Establishing whether $\mathsf{A}' \sqsubseteq_1 \mathsf{A}$ is decidable.*

We now formalise properties of systems of CTAs that one would like to be preserved upon refinement. *Behaviour preservation*, which is based on the notion of timed similarity [15], requires that an implementation (refining system) at any point of a run allows *only* actions that are allowed by its specification (refined system). Below, we use $\uplus$ to denote the disjoint union of TLTSs, i.e. $(Q_1, \Sigma_1, \to_1) \uplus (Q_2, \Sigma_2, \to_2) = (Q_1 \uplus Q_2, \Sigma_1 \cup \Sigma_2, \{((i, q), a, (i, q')) \mid (q, a, q') \in \to_i\})$, where $Q_1 \uplus Q_2 = \{(i, q) \mid q \in Q_i\}$.

Definition 8 (Behaviour preservation). *Let $\mathcal{R}$ be a binary relation between systems. We say that $\mathcal{R}$* preserves behaviour *iff, whenever $S_1 \mathrel{\mathcal{R}} S_2$, we have $(\gamma_0^1, 1) \lesssim (\gamma_0^2, 2)$ in the TLTS $[\![S_1]\!] \uplus [\![S_2]\!]$, where γ_0^1 and γ_0^2 are the initial configurations of S_1 and S_2.*

Example 2 (Behaviour preservation). Let $\mathcal{R}$ be the inclusion of runs, let $S_1 = (\mathsf{A}_{\mathtt{s}}, \mathsf{A}'_{\mathtt{r}})$ and $S_2 = (\mathsf{A}_{\mathtt{s}}, \mathsf{A}_{\mathtt{r}})$, where:



We have that $S_2 \ \mathcal{R} \ S_1$, while $S_1 \ \mathcal{R} \ S_2$ does not hold, since the traces with b in S_1 strictly include those of S_2. The relation $\mathcal{R}$ preserves timed behaviour in $\{S_1, S_2\}$: indeed, $(\gamma_0^2, 1) \lesssim (\gamma_0^1, 2)$ follows by trace inclusion and by the fact that S_1, S_2 have deterministic TLTS. Now, let S_3 be as S_2, but for the guard of $\mathtt{sr}?b(\mathit{true})$, which is replaced by $y < 2$. We have that $S_3 \ \mathcal{R} \ S_1$, and $\mathcal{R}$ preserves timed behaviour in $\{S_2, S_3\}$. However, S_3 does *not* allow to continue with the message exchange: b is sent too late to be received by $\mathtt{r}$, who keeps waiting while b remains in the queue forever. □

As shown by Example 2, behaviour preservation may allow a system (e.g., S_3) to remove "too much" from the runs of the original system (e.g., S_2): while ensuring that no new actions are introduced, it may introduce deadlocks. So, besides behaviour preservation we consider below two other properties of systems: *global* progress of the overall system, and *local* progress of each single participant.

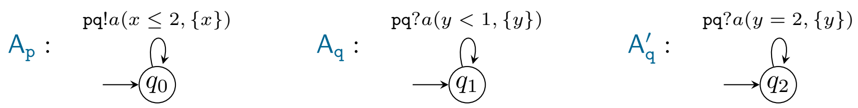
Definition 9 (Global/local progress). *We say that a system S enjoys:*

global progress *when:* $\gamma_0 \rightarrow^* \gamma$ *not final* $\implies \exists t \in \mathbb{R}_{\geq 0}, \alpha \in \mathsf{Act} : \gamma \xrightarrow{t} \xrightarrow{\alpha}$
local progress *when:* $\gamma_0 \rightarrow^* \gamma = (\boldsymbol{q}, \boldsymbol{w}, \nu)$ *and* $q_{\mathtt{p}}$ *not final* $\implies$
$\forall$ *maximal runs* ρ *starting from* $\gamma : \exists n : \mathsf{subj}(act_n(\rho)) = \mathtt{p}$

Lemma 3. *If a system enjoys local progress, then it also enjoys global progress.*

The converse of Lemma 3 does not hold, as witnessed by Example 3.

Example 3 (Global vs. local progress). Consider the following CTAs:



The system $(\mathsf{A}_{\mathtt{p}}, \mathsf{A}_{\mathtt{q}})$ enjoys global progress, since, in each reachable configuration, $\mathsf{A}_{\mathtt{p}}$ can always send a message (hence the system makes an action in Act). However, if $\mathsf{A}_{\mathtt{p}}$ sends a after time 1, then $\mathsf{A}_{\mathtt{q}}$ cannot receive it, since its guard $y < 1$ is not satisfied. Formally, in any maximal run starting from $((q_0, q_1), (a, \varepsilon), \{x, y \mapsto 1\})$, there will be no actions with subject $\mathtt{q}$, so $(\mathsf{A}_{\mathtt{p}}, \mathsf{A}_{\mathtt{q}})$ does *not* enjoy local progress. The system $(\mathsf{A}_{\mathtt{p}}, \mathsf{A}'_{\mathtt{q}})$, instead, enjoys both global and local progress. □

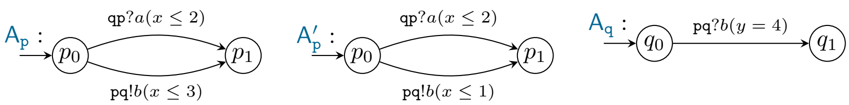
Definition 10 (Progress preservation). *Let $\mathcal{R}$ be a binary relation between systems. We say that $\mathcal{R}$* preserves global (resp. local) progress *iff, whenever $S_1 \ \mathcal{R} \ S_2$ and S_2 enjoys global (resp. local) progress, then S_1 enjoys global (resp. local) progress.*

Example 4. Let S_1, S_2, S_3 as in Example 2. Observe that S_1 and S_2 enjoy progress (both local and global), while S_3 does not enjoy progress (either local or global). Hence, $\mathcal{R} = \{(S_2, S_1), (S_3, S_1), (S_3, S_2)\}$ (i.e., trace inclusion restricted to the three given systems), does not preserve progress (either local or global). □

4 Verification of properties of refinements

We now study preservation of behaviour/progress upon refinements. Our first result is negative: *in general*, refinement does not preserve behaviour nor (local/global) progress, even for CTAs without mixed states. This is shown by following example.

Example 5. Let $S = (\mathsf{A}_{\mathsf{p}}, \mathsf{A}_{\mathsf{q}})$, and let $S' = (\mathsf{A}'_{\mathsf{p}}, \mathsf{A}_{\mathsf{q}})$, where:



We have that $\mathsf{A}'_{\mathsf{p}} \sqsubseteq_1 \mathsf{A}_{\mathsf{p}}$, and so $S' \sqsubseteq S$. Behaviour is not preserved, because S' allows the run $\gamma_0 \xrightarrow{4}$, while S does not. Progress, which is enjoyed by S (both local and global) does not hold in S'. Indeed, S' allows $\gamma_0 \xrightarrow{2} \gamma = ((p_0, q_0), \boldsymbol{\varepsilon}, \nu_0 + 2)$, but there are no t and $\alpha \in \mathsf{Act}$ such that $\gamma \xrightarrow{t} \xrightarrow{\alpha}$. $\square$

The issue in Example 5 is that a LE sending edge, which was crucial for making execution progress, is lost after the refinement. In Definition 12 we devise a decidable condition — which we call LLESP after *locally LE send preservation* — that excludes scenarios like the above. In Theorem 3 we show that, with the additional LLESP condition, $\sqsubseteq_1$ guarantees preservation of behaviour and progress. Unlike Definition 6, which is defined "edge by edge", LLESP is defined "state by state". This is because LLESP preserves the existence of LE sending edges (outgoing from the given state), and not necessarily the LE sending edge himself, making the analysis more precise.

Definition 11. *Let $\mathsf{A} = (Q, q_0, X, E)$, let $q \in Q$, and let K be a set of clock valuations. We define the following sets of clock valuations:*

$$Pre^{\mathsf{A}}_q = \{\nu_0 \mid q_0 = q\} \cup \{\nu \mid \exists q', \ell, \nu' : (q', \ell, q) \in E,\ \nu' \in [\![\mathsf{guard}(\ell)]\!],\ \nu = \mathsf{reset}(\ell)(\nu')\}$$

$$Les^{\mathsf{A}}_q = \{\nu \mid q \textit{ has a LE sending edge in } \nu\}$$

$$Post^{\mathsf{A}}_q(K) = \{\nu + t \,\big|\, \nu \in K \ \wedge\ (\nu \in Les^{\mathsf{A}}_q \implies \nu + t \in Les^{\mathsf{A}}_q)\}$$

We briefly comment the auxiliary definition above. The set Les^{A}_q is self-explanatory, and its use is auxiliary to the definition of $Post$. The sets Pre^{A}_q and $Post^{\mathsf{A}}_q(K)$ are useful for obtaining an over-approximation, denoted $Post^{\mathsf{A}}_q(Pre^{\mathsf{A}}_q)$, of the set of clock valuations ν such that there is a configuration $(\boldsymbol{q}, \boldsymbol{w}, \nu)$, where q is in $\boldsymbol{q}$, that can be reached by the initial configuration of some system S containing A. The set Pre^{A}_q contains all (but not only) the clock valuations under which a configuration like the one above can be reached with a label $\alpha \in \mathsf{Act}$ fired by A. Instead, $Post^{\mathsf{A}}_q(K)$ computes a symbolic step of timed execution, in the following sense: if $\nu \in K$ and $\gamma \xrightarrow{t} (\boldsymbol{q}, \boldsymbol{w}, \nu')$, where q is in $\boldsymbol{q}$, then $\nu' \in Post^{\mathsf{A}}_q(K)$. This is obtained by defining $Post^{\mathsf{A}}_q(K)$ as the set of clock valuations that would satisfy item (d) of Definition 3 for A at runtime, when starting from a configuration whose clock valuation is in K. Since every configuration reachable with a finite run and with an action in Act as last label can also be reached by a run ending with a delay (the original run followed by a null delay), the set $Post^{\mathsf{A}}_q(Pre^{\mathsf{A}}_q)$ contains the clock valuations we were looking for.

Definition 12 (LLESP). *We say that a relation $\mathcal{R}$ is* locally LE send preserving *(in short, LLESP) iff, for all* $\mathsf{A} = (Q, q_0, X, E)$ *and* $\mathsf{A}' = (Q, q_0, X, E')$ *such that* $\mathsf{A}' \mathcal{R} \mathsf{A}$*, and for all* $q \in Q$*:* $Post_q^{\mathsf{A}'}(Pre_q^{\mathsf{A}'}) \cap Les_q^{\mathsf{A}} \subseteq Post_q^{\mathsf{A}'}(Pre_q^{\mathsf{A}'}) \cap Les_q^{\mathsf{A}'}$*. We define* $\sqsubseteq_1^L$ *as the largest LLESP relation contained in* $\sqsubseteq_1$*.*

Basically, LLESP requires that, whenever $\mathsf{A}' \mathcal{R} \mathsf{A}$, if q has a LE sending edge in ν with respect to A, then q has a LE sending edge in ν with respect to A', where ν ranges over elements of $Post_q^{\mathsf{A}'}(Pre_q^{\mathsf{A}'})$.

It follows our main result: $\sqsubseteq_1^L$ preserves behaviour and progress (both global and local). Further, LLESP is decidable, so paving the way towards automatic verification.

Theorem 3 (Preservation under LLESP). $\sqsubseteq_1^L$ *preserves behaviour, and global and local progress. Furthermore, establishing whether* $\mathsf{A}' \sqsubseteq_1^L \mathsf{A}$ *is decidable.*

Negative results on alternative refinement strategies As mentioned in Section 1, besides introducing a new refinement we have investigated behavioural and progress preservation using two refinement strategies known in literature. They are both variants of our definition of refinement that alter conditions (a) and (b) in Definition 6. The first strategy (e.g., [9]) is a naïve variant of Definition 6 where (b) is dropped. The second strategy (e.g., [21]) is an asymmetric variant of Definition 6 that allows to *relax* guards of the receive actions: (a) is substituted by $[\![\mathsf{guard}(\ell')]\!] \supseteq [\![\mathsf{guard}(\ell)]\!]$ and (b) is dropped.

Fact 4 *LLESP restrictions of 'naïve' and 'asymmetric' refinements do not preserve behaviour, global progress, nor local progress, not even if mixed states are ruled out.*

Counter-examples of behaviour and progress preservation for LLESP restrictions of 'naïve' and 'asymmetric' refinements without mixed states are relegated in Appendix C.4 (Examples 14 and 15, respectively). Note that Example 5 is also a counter-example for 'naïve' and 'asymmetric' refinements, in the general case.

Experiments We evaluate our theory against a suite of timed protocols from literature. To support the evaluation we built a tool that determines, given A and A', if $\mathsf{A}' \sqsubseteq_1 \mathsf{A}$ and if $\mathsf{A}' \sqsubseteq_1^L \mathsf{A}$. For each role of each protocol we construct three refinements as follows: (i) for receiving edges, if the guard has a strict upper bound (e.g. $x \leq 10$) then we restrict the guard *as* its upper bound (e.g. $x = 10$); if the upper bound is not strict (e.g. $x < 10$) we 'procrastinate' the guard, but making it fully left-closed (Definition 14) (e.g. $10 - \varepsilon \leq x < 10$, where we set ε as a unit of time); if there is no upper bound (e.g. $x > 10$) the guard is left unchanged; (ii) for sending edges, if the guard has an upper bound (e.g. $x \leq 10$) then we refine it with, respectively, the lower bound (e.g. $x = 0$), the average value (e.g. $x = 5$), and the upper bound (if any) (e.g. $x = 10$); Our tool correctly classifies the pairs of CTAs defined above as refinements. In Table 1 we show the output of the tool when checking LLESP. We can see that samples #2 and #3 never break the LLESP property. While this should always hold for sample #3 (procrastinating sending edges guarantees that LE sending edges are preserved), the case for sample #2 is incidental. Among the case studies, Ford Credit web portal and SMTP contain mixed states (used to implement timeouts). The fact that, for each protocol, there is always some sample refinement that satisfies LLESP (hence a provably safe way to implement that protocol) witnesses the practicality of our theory. Quite surprisingly, the states that falsify LLESP are not mixed.

Case study	Sample #1	Sample #2	Sample #3
Ford Credit web portal [35]	✗Server	✓ Server	
Scheduled Task Protocol [8]	✓User ✓Worker ✓Aggregator	✓User ✓Worker ✓Aggregator	✓User ✓Worker ✓Aggregator
OOI word counting [33]	✓Master ✓Worker ✓Aggregator	✓Master	✓Master
ATM [16]	✗Bank, ✓User ✗Machine	✓Bank ✓User ✓Machine	✓Bank ✓User
Fisher Mutual Exclusion [6]	✓Producer ✓Consumer	✓Producer	✓Producer
SMTP [37]	✓Client	✓Client	

Table 1: Benchmarks. Roles satisfying LLESP are marked with ✓, the others with ✗. We omitted roles for which the refinement strategy was not meaningful or gave identical results to strategies in other columns.

5 Preservation under an urgent semantics

The semantics in Definition 3 does not force the receive actions to happen, (unless time passing prevents the CTA from receiving in the future, by condition (e)). This behaviour, also present in [8, 26], contrasts with the actual behaviour of the `receive` primitives of mainstream programming languages which return *as soon as* a message is available.

We now introduce a variant of the semantics in Definition 3 which faithfully models this behaviour. We make receive actions *urgent* [10, 34] by forbidding delays when a receiving edge is enabled and the corresponding message is at the head of the queue. Below, $\mathsf{Act}^?$ denotes the set of input labels.

Definition 13 (Urgent semantics of systems). *Given a system S, we define the TLTS $[\![S]\!]_u = (Q, \mathcal{L}, \rightarrow_u)$, where Q is the set of configurations of S, $\mathcal{L} = \mathsf{Act} \cup \mathbb{R}_{\geq 0}$, and:*

$$\gamma \xrightarrow{\alpha}_u \gamma' \iff \begin{cases} \gamma \xrightarrow{\alpha} \gamma' & \text{if } \alpha \in \mathsf{Act} \\ \gamma \xrightarrow{t} \gamma' & \text{if } \alpha = t \text{ and } \forall t' < t, \gamma'', \alpha' \in \mathsf{Act}^? : \gamma \xrightarrow{t'} \gamma'' \implies \gamma'' \not\xrightarrow{\alpha'} \end{cases}$$

The non-urgent and the urgent semantics are very similar: they only differ in time actions. In the urgent semantics, a system can make a time action t only if no receive action is possible earlier than t (hence no message is waiting in a queue with 'enabled' guard). Theorem 5 formally relates the two semantics. Since the urgent semantics restricts the behaviour of systems (by dropping some timed transitions), the urgent semantics preserves the behaviour of the non-urgent one.

Theorem 5. *For all systems S, the relation $\{((\gamma, 1), (\gamma, 2)) \mid \gamma$ is a configuration of $S\}$ between states of $[\![S]\!]_u \uplus [\![S]\!]$ is a timed simulation.*

In general, however, a system that enjoys progress with the non-urgent semantics may not enjoy progress with the urgent one. This is illustrated by Example 6.

Example 6. Consider the system $S = (\mathsf{A}_{\mathtt{s}}, \mathsf{A}_{\mathtt{r}})$, where:

$$\mathsf{A}_{\mathtt{s}} : \;\rightarrow q_0 \xrightarrow{\mathtt{sr}!a(y=0)} q_1 \qquad\qquad \mathsf{A}_{\mathtt{r}} : \;\rightarrow q'_0 \xrightarrow{\mathtt{sr}?a(x>3)} q'_1$$

With the non-urgent semantics, $\gamma_0 \xrightarrow{\mathtt{sr}!a} \xrightarrow{3} \gamma = ((q_1, q'_0), (a, \varepsilon), \nu_0 + 3) \xrightarrow{t} \xrightarrow{\mathtt{sr}?a}$, for all $t \in \mathbb{R}_{\geq 0}$. With the urgent semantics, $\gamma_0 \xrightarrow{\mathtt{sr}!a}_u \xrightarrow{3}_u \gamma \not\xrightarrow{\alpha}_u$, for all $\alpha \neq 0$. Hence, the non-urgent semantics leads to a final state, whereas the urgent semantics does not. □

The issue highlighted by Example 6 is subtle (but known in literature [10]): if there is no precise point in time in which a guard becomes enabled (e.g. in $x > 3$), then the run may get stuck. In Definition 14 we deal with this issue through a restriction on guards, which guarantees that urgent semantics preserves progress. Our restriction, generalising the notion of *right-open time progress* [10] (to deal with non-convex guards), corresponds to forbidding guards defined as the conjunction of sub-guards of the form $x > c$ (but we allow subguards of the form $x \geq c$). To keep our results independent from the syntax of guards, our definition is based on sets of clock valuations.

Definition 14 (Fully left closed). *For all ν, and for all sets of clock valuations K, let $D_\nu(K) = \{t \mid \nu + t \in K\}$ and let $\inf Z$ denote the infimum of Z. We say that a guard δ is* fully left closed *iff: $\forall \nu : \forall K \subseteq [\![\delta]\!] : \big(D_\nu(K) \neq \emptyset \implies \nu + \inf D_\nu(K) \in [\![\delta]\!]\big)$. We say that a CTA is* input fully left closed *when all guards in its receiving edges are fully left closed. A system is input fully left closed when all its components are such.*

Fully left closed guards ensure that there is an exact time instant in which a guard of an urgent action becomes enabled. The requirement that left closedness must hold for any subset K of the semantics of the guards is needed to cater for non-convex guards (i.e. guards with disjunctions). Consider e.g. $\delta = 1 \leq x \leq 3 \vee x \geq 4$. While δ is left closed, it is *not* fully left closed: indeed, for $K = [\![x \geq 4]\!] \subseteq [\![\delta]\!]$, it holds that $\inf D_{\nu_0}(K) = 4$, but $\nu + 4 \notin [\![\delta]\!]$.

Example 7. The guard $x > 3$ in Example 6 is not fully left closed, as $\inf D_{\nu_0}([\![x > 3]\!]) = \inf \{t \mid t > 3\} = 3$, but $\nu_0 + 3 \notin [\![x > 3]\!]$. Instead, guard $x \geq 3$ is fully left closed. Consider now a variant of the system of Example 6 where guard $x > 3$ is replaced by $x \geq 3$. The run $\gamma_0 \xrightarrow{\texttt{sr!}a}_u \xrightarrow{3}_u \gamma$ would not get stuck and allow $\gamma \xrightarrow{\texttt{sr?}a}$. □

The following theorem states that urgent semantics preserves progress with respect to non-urgent semantics, when considering fully left closed systems.

Theorem 6 (Preservation of progress vs. urgency). *Let S be input fully left closed. If S enjoys global (resp. local) progress under the non-urgent semantics, then S enjoys global (resp. local) progress under the urgent semantics.*

6 Implementing protocols via refinement

In this section we show how our theory can be exploited when implementing timed protocols. We consider a CTA inspired to a case study from [33], modelling a (fragment of) a protocol for distributed computation of a word count over a set of logs. The system has two nodes: a master $\mathsf{A}_{\mathtt{M}}$ and a worker $\mathsf{A}_{\mathtt{W}}$. We focus on $\mathsf{A}_{\mathtt{M}}$ in Figure 3 (left). $\mathsf{A}_{\mathtt{M}}$ repeatedly: sends a log to $\mathsf{A}_{\mathtt{W}}$, either receives data from $\mathsf{A}_{\mathtt{W}}$ (if it arrives within timeout $x < 8$) or sends a notice and terminates. We implement the CTA in Go, a popular programming language with concurrency features.

A naïve implementation in Go We first attempt to implement $\mathsf{A}_{\mathtt{M}}$ obliviously of our results. We start from the edge from q_0 to q_1, assuming that the preparation of the log to send takes 1 ms (with negligible jitter). This could result in the snippet below:

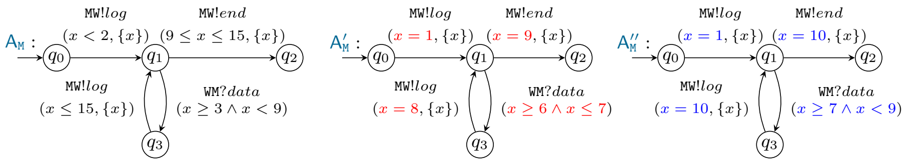


Fig. 3: A_M (left); $A'_M \not\sqsubseteq_1 A_M$ (centre); $A''_M \sqsubseteq_1 A_M$ (right).

```
1  x := time.Now() // initial setting of clock x
2  time.Sleep(x.Add(time.Millisecond * 1).Sub(time.Now())) // sleep for 1ms
3  x = time.Now() // reset x
4  MW <- "log" // send string "log" on FIFO channel MW
```

The statement in line 2 represents the invocation of a time-consuming function that prepares the log to be sent in line 4 (here we send the string `"log"`). In general, implementations may be informed by estimated durations of code instructions. Providing such information is made possible by orthogonal research on cost analysis, e.g. [25].

Next, we want to (i) implement the receive action from q_1 to q_3 as a *blocking* primitive with timeout, (ii) minimise the waiting time of the master listening on the channel, and restrict the interval to $x \geq 6 \wedge x \leq 7$. This could result in the following:

```
1  time.Sleep(x.Add(time.Millisecond * 6).Sub(time.Now()))
2  select {
3    case res := <- WM:
4    // here goes the implementation of edge q3 ---> q1
5    case <- time.After(x.Add(time.Millisecond * 7)
6                .Add(time.Nanosecond).Sub(time.Now())):
7    // here goes the implementation of edge q1 ---> q2
```

Next, we implement the edge from q_1 to q_2 by substituting line 6 above with:

```
1  time.Sleep(x.Add(time.Millisecond * 9).Sub(time.Now()))
2  x = time.Now() // reset x
3  MW <- "end" // send string "end"
```

The edge from q_3 to q_1 can be implemented in a similar way, where the sleep statement represents a time-consuming log preparation of 1 ms, as before.

Assessing the implementation via our tool The sketch of implementation described in the previous paragraphs corresponds to the CTA A'_M in Figure 3. The analysis of A'_M with our tool points out that $A'_M \not\sqsubseteq_1 A_M$. In fact, the constraint of receiving edges of A_M have been restricted not respecting the final deadlines.From Section 4 we know that A'_M may *not* preserve behaviour and progress. Suppose that the worker node is set to send the data to A_M when $x = 8.5$: according to the original specification A_M, this message is in time hence the worker will expect to receive a log message back from the master. However, in the implementation reflected in A'_M, the master will reply with an end message, potentially causing a deadlock. Thanks to Theorem 3 we know that we can, instead, safely restrict the constraints in A'_M using $\sqsubseteq_1$: guard $x \geq 6 \wedge x \leq 7$ can be amended as $x \geq 7 \wedge x < 9$. After this amendment, however, the tool detects a violation of LLESP: the deadlines set by guards of sending edges from q_3 and q_1 are after the deadline of the receive action. A correct refinement $A''_M \sqsubseteq_1^L A_M$ is shown in Figure 3 (right). A''_M can be used to produce the following implementation in Go:

```go
MW := make(chan string, 100)
WM := make(chan string, 100)
go func(){
  // q0 ---> q1
  x := time.Now()
  time.Sleep(x.Add(time.Millisecond*1)
        .Sub(time.Now()))
  x = time.Now()
  MW <- "log"
  // q1 ---> q3
  time.Sleep(x.Add(time.Millisecond*7)
        .Sub(time.Now()))
  select {
```

```go
case res := <- WM:
// q3 ---> q2
x = time.Now()
time.Sleep(x.Add(time.Millisecond*10)
    .Sub(time.Now()))
MW <- "log"
case <- time.After(x.Add(time.
    Millisecond * 8).Sub(time.Now())):
// q1 ---> q2
time.Sleep(x.Add(time.Millisecond*10)
    .Sub(time.Now()))
x = time.Now()
MW <- "end"  }}()}
```

Strategy of implementation The Go implementation could be simpler (e.g., specifying absolute delays instead of using 'clocks'). We chose this strategy of implementation to highlight correspondences between patterns in CTA and Go programs, which we are using, in ongoing work, as guidelines for semi-automated code derivation from CTA.

Strategy of refinement We implemented the receive action using a blocking primitive. In some scenarios, one may want to use *non-blocking* receive primitives. These can be modelled as CTA refinements where a constraint (e.g., $x \leq 9$) is restricted to a point in time (e.g., $x = 9$), although this may not be possible (e.g., if the original constraint was $x < 9$). In some other cases, it may be desirable to *not* restrict the constraint of receive actions, for example to be able to receive a message as soon as possible, and continue.

Accounting for other delays Delays of the communication medium can be represented in the model e.g., by adding them at the receiver side. This is common when using separated semantics (i.e., actions are timeless and delays are modelled separately), as they can be encoded into semantics where actions have an associated duration.

7 Conclusions

The purpose of our work was to provide formal basis to support implementation of well-behaved systems from well-behaved models. We have given a decidable refinement and a condition (LLESP) that guarantee behaviour and progress preservation.

In general, establishing if an asynchronous communication model is a refinement of another is undecidable [12,27]. We obtained decidability by focusing on "purely timed" refinements (structure preserving mappings that only affect guards). While not fully general (e.g., it does not affect the branching structures of CTAs), our refinement captures the practical relationship between models, and implementations obtained by following those models (as shown in Section 6). Moreover, our refinement and the LLESP condition apply well to realistic protocols expressed as CTA (Section 4): for each role of each protocol in our portfolio, there exist one or more non-trivial (i.e. not the identity) LLESP refinements, from which one can derive behaviour- and progress-preserving implementations of that protocol. Other "purely timed" refinements strategies inspired by literature gave only negative results (Fact 4) when applied to the asynchronous timed scenario, hence e.g., even if an implementation preserves the interactions structure of the initial CTA, and even if the timings of actions chosen for the implementation are within the range of the guards of the initial CTA, still that implementation may not preserve behaviour or progress.

Technically, we focused on interaction-based rather than language-based semantics. In this context, our semantics extend the state of the art in two ways: mixed choices and urgency. Mixed choices cannot be expressed in models related to session types of [8, 9]. There, the interactions are described in terms of two constructs: *selection*, which corresponds to an internal choice of send actions, and *branching*, an external choice of receive actions. The behaviour of mixed states captured by our semantics falls somewhere in between internal and external choices, so it is not expressible in the setting of [8, 9]. Besides, the known semantics [8, 9, 26] do not account for urgency. The preservation results from non-urgent to urgent semantics in Section 5 pave the way to *implementations* of refinements (e.g. derived incrementally using the non-urgent semantics, and relying on the results in Section 4) that preserve behaviour and progress.

Evaluation of the theory was facilitated by a tool that can also be used to guide implementations (shown in Section 6). The tool is the first necessary step towards a framework for assisted implementation of CTA, which is an ongoing work.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. 126(2), 183–235 (1994)
2. Advanced Message Queuing protocols (AMQP) homepage. https://www.amqp.org/
3. Armstrong, J.: Programming Erlang: Software for a Concurrent World. Pragmatic Bookshelf (2007)
4. Bartoletti, M., Cimoli, T., Murgia, M.: Timed session types. Logical Methods in Computer Science 13(4) (2017)
5. Bartoletti, M., Scalas, A., Zunino, R.: A semantic deconstruction of session types. In: CONCUR. Lecture Notes in Computer Science, vol. 8704, pp. 402–418. Springer (2014)
6. Behrmann, G., David, A., Larsen, K.G.: A tutorial on uppaal. In: SFM. Lecture Notes in Computer Science, vol. 3185, pp. 200–236. Springer (2004)
7. Bengtsson, J., Yi, W.: Timed automata: Semantics, algorithms and tools. In: Lectures on Concurrency and Petri Nets. Lecture Notes in Computer Science, vol. 3098, pp. 87–124. Springer (2003)
8. Bocchi, L., Lange, J., Yoshida, N.: Meeting deadlines together. In: CONCUR. LIPIcs, vol. 42, pp. 283–296. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2015)
9. Bocchi, L., Yang, W., Yoshida, N.: Timed multiparty session types. In: CONCUR. Lecture Notes in Computer Science, vol. 8704, pp. 419–434. Springer (2014)
10. Bornot, S., Sifakis, J., Tripakis, S.: Modeling urgency in timed systems. In: COMPOS. Lecture Notes in Computer Science, vol. 1536, pp. 103–129. Springer (1997)
11. Brand, D., Zafiropulo, P.: On communicating finite-state machines. J. ACM 30(2), 323–342 (1983)
12. Bravetti, M., Carbone, M., Zavattaro, G.: Undecidability of asynchronous session subtyping. Inf. Comput. 256, 300–320 (2017)
13. Bruno, E.J., Bollella, G.: Real-Time Java Programming: With Java RTS. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edn. (2009)
14. Cattani, S., Kwiatkowska, M.Z.: A refinement-based process algebra for timed automata. Formal Asp. Comput. 17(2), 138–159 (2005)
15. Cerans, K.: Decidability of bisimulation equivalences for parallel timer processes. In: CAV. Lecture Notes in Computer Science, vol. 663, pp. 302–315. Springer (1992)
16. Chandrasekaran, P., Mukund, M.: Matching scenarios with timing constraints. In: FORMATS. Lecture Notes in Computer Science, vol. 4202, pp. 98–112. Springer (2006)
17. Chen, T., Dezani-Ciancaglini, M., Scalas, A., Yoshida, N.: On the preciseness of subtyping in session types. Logical Methods in Computer Science 13(2) (2017)
18. Chilton, C., Kwiatkowska, M.Z., Wang, X.: Revisiting timed specification theories: A linear-time perspective. In: FORMATS. Lecture Notes in Computer Science, vol. 7595, pp. 75–90. Springer (2012)
19. Clemente, L., Herbreteau, F., Stainer, A., Sutre, G.: Reachability of communicating timed processes. In: FoSSaCS, Lecture Notes in Computer Science, vol. 7794, pp. 81–96. Springer (2013)
20. David, A., Larsen, K.G., Legay, A., Nyman, U., Traonouez, L., Wasowski, A.: Real-time specifications. STTT 17(1), 17–45 (2015)
21. Demangeon, R., Honda, K.: Full abstraction in a subtyped pi-calculus with linear types. In: CONCUR. Lecture Notes in Computer Science, vol. 6901, pp. 280–296. Springer (2011)
22. Dierks, H., Kupferschmid, S., Larsen, K.G.: Automatic abstraction refinement for timed automata. In: FORMATS. Lecture Notes in Computer Science, vol. 4763, pp. 114–129. Springer (2007)
23. Fecher, H., Majster-Cederbaum, M.E., Wu, J.: Refinement of actions in a real-time process algebra with a true concurrency model. Electr. Notes Theor. Comput. Sci. 70(3), 260–280 (2002)

24. Henzinger, T.A., Nicollin, X., Sifakis, J., Yovine, S.: Symbolic model checking for real-time systems. Inf. Comput. 111(2), 193–244 (1994)
25. Hoffmann, J., Shao, Z.: Automatic static cost analysis for parallel programs. In: ESOP. Lecture Notes in Computer Science, vol. 9032, pp. 132–157. Springer (2015)
26. Krcál, P., Yi, W.: Communicating timed automata: The more synchronous, the more difficult to verify. In: CAV. Lecture Notes in Computer Science, vol. 4144, pp. 249–262. Springer (2006)
27. Lange, J., Yoshida, N.: On the undecidability of asynchronous session subtyping. In: FoSSaCS. Lecture Notes in Computer Science, vol. 10203, pp. 441–457 (2017)
28. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a nutshell. STTT 1(1-2), 134–152 (1997)
29. Lynch, N.A., Segala, R., Vaandrager, F.W.: Hybrid I/O automata. Inf. Comput. 185(1), 105–157 (2003)
30. Mostrous, D.: Session Types in Concurrent Calculi: Higher-Order Processes and Objects. Ph.D. thesis, Imperial College London (November 2009)
31. Mostrous, D., Yoshida, N.: Session-based communication optimisation for higher-order mobile processes. In: TLCA. Lecture Notes in Computer Science, vol. 5608, pp. 203–218. Springer (2009)
32. Mostrous, D., Yoshida, N., Honda, K.: Global principal typing in partially commutative asynchronous sessions. In: ESOP. Lecture Notes in Computer Science, vol. 5502, pp. 316–332. Springer (2009)
33. Neykova, R., Bocchi, L., Yoshida, N.: Timed runtime monitoring for multiparty conversations. Formal Asp. Comput. 29(5), 877–910 (2017)
34. Nicollin, X., Sifakis, J.: An overview and synthesis on timed process algebras. In: CAV. Lecture Notes in Computer Science, vol. 575, pp. 376–398. Springer (1991)
35. Ponge, J., Benatallah, B., Casati, F., Toumani, F.: Analysis and applications of timed service protocols. ACM Trans. Softw. Eng. Methodol. 19(4), 11:1–11:38 (2010)
36. Schneider, S.: Concurrent and Real Time Systems: The CSP Approach. John Wiley & Sons, Inc., New York, NY, USA, 1st edn. (1999)
37. The Simple Mail Transfer Protocol. http://tools.ietf.org/html/rfc5321
38. Vinoski, S.: Advanced message queuing protocol. IEEE Internet Computing 10(6), 87–89 (2006)
39. Wang, W., Jiao, L.: Trace abstraction refinement for timed automata. In: ATVA. Lecture Notes in Computer Science, vol. 8837, pp. 396–410. Springer (2014)
40. Yovine, S.: Kronos: A verification tool for real-time systems. (kronos user's manual release 2.2). STTT 1(1-2), 123–133 (1997)

A Proofs and additional material for Section 2

Definition 15 (Timed LTS). *A timed labelled transition system (in short, TLTS) is a triple $(Q, \mathcal{L}, \rightarrow)$, where Q is the set of* states*, $\mathcal{L} \supseteq \mathbb{R}_{\geq 0}$ is the set of* labels*, and $\rightarrow \subseteq Q \times \mathcal{L} \times Q$ is the* transition relation*. We use $\alpha, \beta, \ldots$ to range over $\mathcal{L}$.*

Example 8. In the initial clock valuation ν_0, we have that:

$$(x \leq 2) \ \leq_{\nu_0} \ (x \leq 3) \qquad (x \leq 2) \ \leq_{\nu_0} \ (x = 3) \qquad (x \leq 3) \ \not\leq_{\nu_0} \ (x \leq 2)$$

Note that the relation $\leq_\nu$ is *not* antisymmetric: e.g., for all $c, c' \in \mathbb{R}_{\geq 0}$, we have that $(x > c) \leq_\nu (x > c') \leq_\nu (x > c)$, even if $c \neq c'$.

Proof of Lemma 1

Proving reflexivity and transitivity is straightforward. To prove that $\leq_\nu$ is total, we show that if $\delta \not\leq_\nu \delta'$, then it must be $\delta' \leq_\nu \delta$. Since $\delta \not\leq_\nu \delta'$, then there exists some $t_0 \in \mathbb{R}_{\geq 0}$ such that $\nu + t_0 \in [\![\delta]\!]$, but:

$$\nexists t' \geq t_0 : \nu + t' \in [\![\delta']\!] \tag{1}$$

To prove $\delta' \leq_\nu \delta$, let $t' \in \mathbb{R}_{\geq 0}$ such that $\nu + t' \in [\![\delta']\!]$ (if no such t' exists, we trivially obtain the thesis). By (1), it must be $t' < t_0$. Hence, we have found a $t_0 > t'$ such that $\nu + t_0 \in [\![\delta]\!]$, from which we conclude that $\delta' \leq_\nu \delta$.

For item (a), assume that $d \geq 0$. Let $t \in \mathbb{R}_{\geq 0}$ be such that $\nu + t \in [\![x \leq c]\!]$. Then, $\nu(x) + t \leq c$, and so $\nu(x) + t \leq c + d$. Choosing $t' = t$, we have $\nu + t' \in [\![x \leq c + d]\!]$ from which we obtain the thesis $(x \leq c) \leq_\nu (x \leq c + d)$. To prove the converse, assume that $d < 0$. Let $t = c - \nu(x)$. Then, $\nu(x) + t \leq c$, but there exists no $t' \geq t$ such that $\nu(x) + t' \leq c + d$. Hence, $(x \leq c) \not\leq_\nu (x \leq c + d)$.

For item (b), let $t \in \mathbb{R}_{\geq 0}$ be such that $\nu + t \in [\![\delta \wedge \delta']\!] = [\![\delta]\!] \cap [\![\delta']\!]$. Choose $t' = t$. Then, $\nu + t \in [\![\delta']\!]$, from which we obtain the thesis $\delta \wedge \delta' \leq_\nu \delta'$.

For item (c), assume that $\delta \leq_\nu \delta'$. Let $\tilde{t} \in \mathbb{R}_{\geq 0}$ be such that $(\nu + t) + \tilde{t} \in [\![\delta]\!]$. Since $\delta \leq_\nu \delta'$, there exists $\tilde{t}' \geq t + \tilde{t}$ such that $\nu + \tilde{t}' \in [\![\delta']\!]$. Choose $t' = \tilde{t}' - t$. Then, $t' \geq \tilde{t}$, and $(\nu + t) + t' = (\nu + t) + (\tilde{t}' - t) = \nu + \tilde{t}' \in [\![\delta']\!]$. Therefore, $\delta \leq_{\nu + t} \delta'$. □

Example 9. Fix a configuration $(\boldsymbol{q}, \boldsymbol{w}, \nu)$ of some system that includes one of the CTAs in Figure 4. The edge $(q_0, \texttt{pr!}b(x < 3), q_2)$ of A_1 is future-enabled iff $\nu(x) < 3$; the latest-enabled edge is $(q_0, \texttt{ps!}b(x < 5), q_3)$ if $\nu(x) < 5$, otherwise there are no latest-enabled edges. Both outgoing edges of A_2 are latest-enabled if $\nu(x) \leq 2$. In A_3 we can

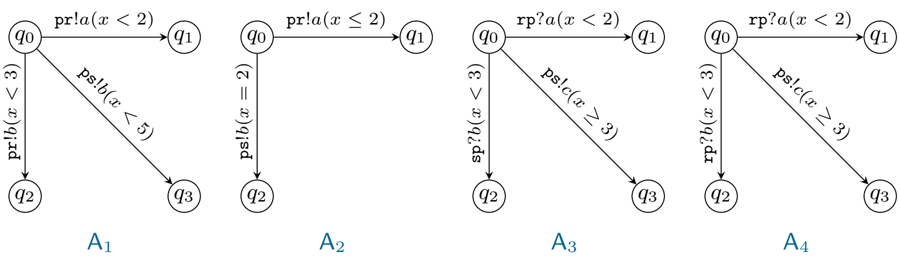


Fig. 4: CTAs for Example 9.

have two non-deferrable edges: $(q_0, \mathtt{rp}?a(x < 2), q_1)$ if $w_{\mathtt{rp}} = aw'_{\mathtt{rp}}$ and $\nu(x) < 2$, and $(q_0, \mathtt{sp}?b(x < 3), q_2)$ if $w_{\mathtt{sp}} = bw'_{\mathtt{sp}}$ and $\nu(x) < 3$. In A_4, if $w_{\mathtt{rp}} = aw'_{\mathtt{rp}}$ and $\nu(x) < 2$, then the edge $(q_0, \mathtt{rp}?a(x < 2), q_1)$ is non-deferrable. Otherwise, if $w_{\mathtt{rp}} = bw'_{\mathtt{rp}}$ and $\nu(x) < 3$, then $(q_0, \mathtt{rp}?b(x < 3), q_2))$ is non-deferrable. In this CTA, only one of the two edges can be non-deferrable, as they both receive from $\mathtt{r}$. If the head of the queue $w_{\mathtt{rp}}$ is neither a nor b, then A_4 does not have non-deferrable edges.

Lemma 4. *Let $e = (q, \ell, q') \in E_{\mathtt{p}}$. Then:*

$$e \textit{ future-enabled in } (\boldsymbol{q}, \boldsymbol{w}, \nu + t) \implies e \textit{ future-enabled in } (\boldsymbol{q}, \boldsymbol{w}, \nu)$$

Proof. Trivial. □

Proof of Lemma 2

We have to prove that, for all configurations $\gamma, \gamma', \gamma''$, and for all $t, t' \in \mathbb{R}_{\geq 0}$, we have that:

$$\gamma \xrightarrow{t} \gamma' \ \wedge\ \gamma \xrightarrow{t} \gamma'' \implies \gamma' = \gamma'' \qquad \text{(Time determinism)}$$
$$\gamma \xrightarrow{t+t'} \gamma' \iff \exists \tilde{\gamma} : \gamma \xrightarrow{t} \tilde{\gamma} \ \wedge\ \tilde{\gamma} \xrightarrow{t'} \gamma' \qquad \text{(Time additivity)}$$

Time determinism follows immediately by definition 3. Time additivity follows by lemma 4. □

Proof of Theorem 1

For receive persistency, since $\gamma \xrightarrow{t'} \xrightarrow{\mathtt{rp}?}$ then some message expected from $\mathtt{p}$ is already at the head of the queue $\mathtt{rp}$ in configuration γ. Hence, there exists some edge in $E_{\mathtt{p}}$ which is non-deferrable in γ. Since $\gamma \xrightarrow{t} \gamma'$, condition ((e)) of Definition 3 ensures that there exists some non-deferrable edge also in γ'. Since $\leq_\nu$ is total, then there exists a non-deferrable edge e which is enabled after all the other non-deferrable edges. Assume that $\mathsf{act}(e) = \mathtt{sp}?$. Since non-deferrable edges are also future-enabled, there exists t'' such that the guard of e is true in $\nu' + t''$, where ν' is the clock valuation of γ'. We show that the transition $\gamma' \xrightarrow{t''}$ is admitted by our semantics. Condition ((d)) of Definition 3 is satisfied, because if the latest-enabled edge is a sending edge, then the latest time when it can be fired falls after t'' (otherwise, the edge e would be the latest enabled one). Condition ((e)) of Definition 3 holds as well, because the edge e itself remains non-deferrable in $\nu' + t''$. Therefore, by condition (2) of Definition 3, we obtain the thesis $\gamma' \xrightarrow{t''} \xrightarrow{\mathtt{sp}?}$.

For send persistency, since $\gamma \xrightarrow{t'} \xrightarrow{\mathtt{pr}!}$ then there exists a sending edge in $E_{\mathtt{p}}$ which is future-enabled. Since $\leq_\nu$ is total, then there exists a sending edge e_s which is enabled after all the other sending edges. Note that e_s is not necessarily latest-enabled, because the latest-enabled edge could be a receiving one. There are two cases:

1. If there exist no non-deferrable receiving edges, let t'' be such that the guard of e_s is true in $\nu' + t''$, where ν' is the clock valuation of γ' (such t'' always exists, because e_s is future-enabled). Assume that $\mathsf{act}(e) = \mathtt{ps}!$. We show that the transition $\gamma' \xrightarrow{t''} \gamma''$ is admitted by our semantics. Condition ((d)) of Definition 3 holds: indeed, by the choice of e_s and of t'' it follows that if e_s was latest-enabled in γ',

then it is latest-enabled also in γ''. Condition ((e)) holds trivially, because in this case we are assuming all receiving edges to be deferrable. Further, by condition (1) of Definition 3, $\gamma'' \xrightarrow{\texttt{ps!}}$. Therefore we conclude that $\gamma' \xrightarrow{t''} \xrightarrow{\texttt{ps!}}$.

2. If there exists some non-deferrable receiving edges, let e_r be the latest-enabled among them. Let t_r be the latest delay where the guard of e_r is satisfied from ν', and let t_s be the latest delay where the guard of e_s is satisfied from ν'. There are two further subcases:
   (a) if $t_r \geq t_s$, we show that the latest sending action is preserved. Let $t'' = t_s$, and let $\mathsf{act}(e_s) = \texttt{ps!}$. We show that the transition $\gamma' \xrightarrow{t''} \gamma''$ is admitted by our semantics. Condition ((d)) of Definition 3 holds: indeed, by the choice of e_s and of t'' it follows that if e_s was latest-enabled in γ', then it is latest-enabled also in γ''. Condition ((e)) holds, because $t_r \geq t_s$, and so the edge e_r is still non-deferrable in γ''. Since the guard of e_s is satisfied in $\nu' + t''$, by condition (1) of Definition 3, we obtain the thesis $\gamma' \xrightarrow{t''} \xrightarrow{\texttt{ps!}}$.
   (b) otherwise, if $t_r < t_s$, we show that the latest receiving action is preserved. Let $t'' = t_r$, and let $\mathsf{act}(e_r) = \texttt{sp?}$. We show that the transition $\gamma' \xrightarrow{t''} \gamma''$ is admitted by our semantics. Condition ((d)) of Definition 3 holds: indeed, by the choice of e_s and of t'' it follows that e_s is latest-enabled in γ''. Condition ((e)) holds, because t_r is the longest delay such that e_r is still non-deferrable in γ''. Since the guard of e_r is satisfied in $\nu' + t''$, by condition (2) of Definition 3, we obtain the thesis $\gamma' \xrightarrow{t''} \xrightarrow{\texttt{sp?}}$. □

B Proofs and additional material for Section 3

Definition 16 (Timed similarity). *Let $(Q, \mathcal{L}, \rightarrow)$ be a TLTS. A timed simulation is a relation $\mathcal{R} \subseteq Q \times Q$ such that, whenever $\gamma_1 \; \mathcal{R} \; \gamma_2$:*

$$\forall \alpha \in \mathcal{L} \; : \; \gamma_1 \xrightarrow{\alpha} \gamma_1' \implies \exists \gamma_2' : \gamma_2 \xrightarrow{\alpha} \gamma_2' \text{ and } \gamma_1' \; \mathcal{R} \; \gamma_2'$$

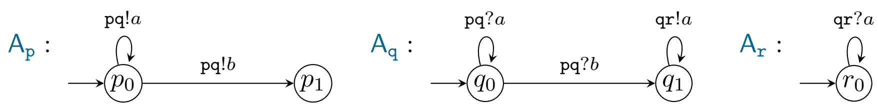
We call timed similarity *(in symbols, $\lesssim$) the largest timed simulation relation.*

Proof of Lemma 3

We prove the contrapositive. Assume that S does *not* enjoy global progress, i.e. by Definition 9 there exists $\gamma = (\boldsymbol{q}, \boldsymbol{w}, \nu)$ not final such that $\gamma_0 \rightarrow^* \gamma$, but for no t and $\alpha \in \mathsf{Act}$ it holds that $\gamma \xrightarrow{t} \xrightarrow{\alpha}$. So, by Definition 4, $\rho = \gamma$ is a maximal run of γ. Since γ is not final, there exists a participant $\texttt{p}$ such that $q_{\texttt{p}}$ is not final. Clearly, since the run γ has no transitions, there is no n such that $\mathsf{subj}(act_n(\rho)) = \texttt{p}$. Therefore, by Definition 9, we conclude that S does not enjoy local progress. □

The following example shows a situation where all runs admit a continuation containing send/receive actions of *all* CTAs (if not in a final state). Yet, local progress does *not* hold, because there also exist (maximal) runs where a CTA is stuck.

Example 10. Consider the following CTAs (guards are $true$, and clocks are immaterial):



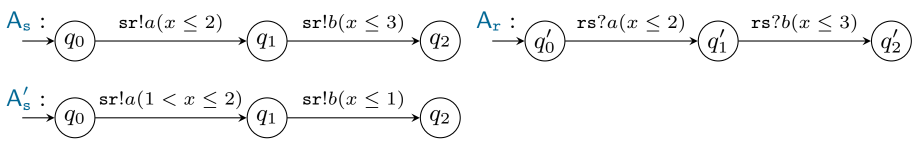
In any reachable configuration of $(\mathsf{A_p}, \mathsf{A_q}, \mathsf{A_r})$ there is a continuation where any CTA can make an action in Act. However, in runs where $\mathsf{A_p}$ always sends a to $\mathsf{A_q}$, the CTA $\mathsf{A_r}$ is stuck. Hence, the system enjoys global progress, but it does not enjoy local progress.

C Proofs and additional material for Section 4

This section contains the proof of the main result Theorem 3. The proof relies on properties of some subclasses of the "naïve" refinement strategy mentioned in the main text (and formally defined in Definition 20), which we will refer to with symbol $\sqsubseteq_{sr}$. In particular, we introduce two properties LESP and NDP (Definitions 18 and 19) on system refinements that guarantees preservation of behaviour and progress of $\sqsubseteq_{sr}$ (Theorem 9). Note that LESP and NDP are not direcly usable in practice due to their undecidability (Theorems 7 and 8). We then show that $\sqsubseteq_1$ is a NDP restriction of $\sqsubseteq_{sr}$ (Lemma 15), and that if a point-wise refinement is LLESP, then the system refinement associated with it is LESP (Theorem 11). From the above it follows that $\sqsubseteq_1^L$ preserves behaviour and progress.

Additional counter-example of preservation in the general case. Note that all CTAs are not mixed.

Example 11. Let $S = (\mathsf{A_s}, \mathsf{A_r})$, and and let $S' = (\mathsf{A'_s}, \mathsf{A_r})$, where:



Note that $\mathsf{A'_s} \sqsubseteq_1 \mathsf{A_s}$, and so $S' \sqsubseteq S$. Consider the run $\gamma_0 \xrightarrow{2} \gamma_1 \xrightarrow{\mathtt{sr}!a} \gamma_2 \xrightarrow{\mathtt{sr}?a} \gamma_3 \xrightarrow{3}$ of S', where γ_i are uniquely determined by the labels. The last delay is possible since S' has no FE actions in γ_3 (so, it is stuck). Instead, in S the last delay is *not* possible, since $\mathsf{A_s}$ has a FE action in γ_3, but no FE actions after a delay of 3 time units. Since S' has a trace not allowed by S, behaviour is *not* preserved. Similarly, S enjoys progress (both global and local), while S' gets stuck in $((q_1, q'_1), (\varepsilon, \varepsilon), \nu)$, with $1 < \nu(x) \leq 2$. So, global and local progress are *not* preserved.

C.1 General properties

We start with some auxuliary definitions and lemmas about CTAs and refinements.

Definition 17. *For all configurations $\gamma = (\boldsymbol{q}, \boldsymbol{w}, \nu)$ of a given system S, and for all $t \in \mathbb{R}_{\geq 0}$, we define:*

$$BSE_\gamma(t) = \left\{ e \;\middle|\; \begin{array}{r} \exists \mathtt{p}, q'_\mathtt{p}, \ell : e = (q_\mathtt{p}, \ell, q'_\mathtt{p}) \text{ latest enabled sending edge in } \gamma \\ \text{and not future enabled in } (\boldsymbol{q}, \boldsymbol{w}, \nu + t) \end{array} \right\}$$

$$BRE_\gamma(t) = \left\{ e \;\middle|\; \begin{array}{r} \exists \mathtt{p}, q'_\mathtt{p}, \ell : (q_\mathtt{p}, \ell, q'_\mathtt{p}) \text{ non-deferrable edge in } \gamma \text{ and} \\ \forall \ell', q''_\mathtt{p} : (q_\mathtt{p}, \ell', q''_\mathtt{p}) \text{ not non-deferrable in } \nu + t \end{array} \right\}$$

$$BE_\gamma(t) = BSE_\gamma(t) \cup BRE_\gamma(t)$$

We then define the following family of sets (indexed over $n \in \mathbb{N}$):

$$R^n(\gamma) = \{\gamma' \mid \exists t \in \mathbb{R}_{\geq 0}, \alpha \in \mathsf{Act} : \gamma' = (\boldsymbol{q}, \boldsymbol{w}, \nu + t) \xrightarrow{\alpha} \text{ and } |BE_\gamma(t)| = n\}$$

Intuitively, the set $BE_\gamma(t)$ contains the actions that prevent the timed transition $\gamma \xrightarrow{t}$. It is composed by the sets $BSE_\gamma(t)$ and $BRE_\gamma(t)$, that contains the actions violating, respectively, conditions ((d)) and ((e)) of Definition 3 in an attempt to derive $\gamma \xrightarrow{t}$. Finally, the set $R^n(\gamma)$ is composed of those configurations γ' that can perform a discrete transition immediately, and in a derivation of $\gamma \xrightarrow{t} \gamma'$ there are exactly n edges that breaks one among conditions ((e)), ((d)), while the other conditions hold. Such an apparently contorted definition will be useful in proofs, as it enables us to perform inductions on the index n of $R^n(\gamma)$.

Lemma 5. *Let $\gamma = (\boldsymbol{q}, \boldsymbol{w}, \nu)$ be such that $\gamma_0 \rightarrow^* \gamma$. For all $t \in \mathbb{R}_{\geq 0}$ and $\alpha \in \mathsf{Act}$:*

$$\gamma \xrightarrow{t} \gamma' \xrightarrow{\alpha} \iff \gamma' \in R^0(\gamma)$$

Proof. Direct consequence of the fact that $BSE_\gamma(t)$ and $BRE_\gamma(t)$ are composed by those edges that, respectively, break conditions ((e)) and ((d)) of Definition 3. □

Lemma 6. *Let S be a system of machines, and let γ be any configuration such that $\gamma_0 \rightarrow^* \gamma$. Then, for all $t, t', f \in \{BSE_\gamma, BRE_\gamma, BE_\gamma\}$:*

$$t \leq t' \implies f(t) \subseteq f(t')$$

Proof. Let S and γ be as in the statement, and suppose $t \leq t'$. We proceed by cases on f.

- $f = BSE_\gamma$. Suppose $e = (q_\mathtt{p}, \ell, q'_\mathtt{p}) \in BSE_\gamma(t)$. It must be ℓ latest enabled sending in γ and not future enabled in $\nu + t$. It remains to show ℓ not future enabled in $\nu + t'$: an easy inspection of definition 2, using the assumption $t < t'$.
- $f = BRE_\gamma$. Similar to the above.
- $f = BE_\gamma$. Immediate consequence of the above cases.

□

Lemma 7. *Let S be a system of machines, and let $\gamma = (\boldsymbol{q}, \boldsymbol{w}, \nu)$ be a configuration such that $\gamma_0 \rightarrow^* \gamma$. Then, for all t and for all $\alpha \notin \mathbb{R}_{\geq 0}$:*

$$(\boldsymbol{q}, \boldsymbol{w}, \nu + t) \xrightarrow{\alpha} \wedge BRE_\gamma(t) = \emptyset \implies \exists t', \ell' : \gamma \xrightarrow{t'} \xrightarrow{\ell'}$$

Proof. Let S and γ be as in the statement. Since, for all t, any γ' in the form $(\boldsymbol{q}, \boldsymbol{w}, \nu + t)$ and such that $\gamma' \xrightarrow{\alpha}$ is a member of $R^n(\gamma)$, for some n, we proceed by induction on n. The base case follows by lemma 5.
For the inductive step, let $n > 0$ and let $\gamma' \in R^n(\gamma)$ be such that $BRE_\gamma(t) = \emptyset$. Pick any member e of $BE_\gamma(t)$. Since $BRE_\gamma(t)$ is empty, it must be $e \in BSE_\gamma(t)$, and thus $e = (q_\mathtt{p}, \ell, q'_\mathtt{p})$ for some p, with ℓ latest enabled (and hence future-enabled) sending in γ, but not future enabled in $\nu + t$. Then, there is $t' < t$ such that $\nu + t' \in [\![\mathsf{guard}(\ell)]\!]$. Therefore, ℓ is future enabled in $\nu + t'$ and hence $e \notin BE_\gamma(t')$. Thanks to this, together with lemma 6, we can conclude $BE_\gamma(t') \subset BE_\gamma(t)$ and therefore $(\boldsymbol{q}, \boldsymbol{w}, \nu + t') \in R^{n'}(\gamma)$ for some $n' < n$. By the induction hypothesis, it follows $\gamma \xrightarrow{t''} \xrightarrow{\ell'}$ for some $t'', \alpha' \notin \mathbb{R}_{\geq 0}$. □

Lemma 8. *Let S_1 and S_2 be systems such that every maximal run ρ of S_1 is a maximal run of S_2. Then, S_2 has local progress $\implies$ S_1 has local progress.*

Proof. Let S_1 and S_2 be as in the statement, and suppose S_2 has strong local progress. We have to show that S_1 has strong local progress as well. So, suppose $\gamma_0 \rightarrow^*_{S_1} \gamma = (\boldsymbol{q}, \boldsymbol{w}, \nu)$, with run, say, $\rho = \gamma_0 \xrightarrow{t_1}_{S_1} \gamma_0' \xrightarrow{\alpha_1}_{S_1} \gamma_1 \ldots \gamma$. According to definition 9, we have to show that, for all $\mathtt{p}$ such that $q_{\mathtt{p}}$ is not final, and for all maximal runs ρ' of S_1 starting from γ, there is n such that $\mathsf{subj}(act_n(\rho')) = \mathtt{p}$. So, suppose $q_{\mathtt{p}}$ is not final, and let ρ' be a maximal run of S_1 starting from γ. Clearly, $\rho\rho'$ is a maximal run of S_1 and hence of S_2. Thus, γ is reachable by S_2 and ρ' is a maximal run of S_2 starting from γ. Note that $q_{\mathtt{p}}$ is not final with respect to S_2 as well, and hence, since S_2 has strong local progress by assumption, there is n such that $\mathsf{subj}(act_n(\rho')) = \mathtt{p}$. $\square$

Lemma 9. *Let S be a system that progress. Then, for all $\gamma = (\boldsymbol{q}, \boldsymbol{w}, \nu)$ not final such that $\gamma_0 \rightarrow^* \gamma$, there is $\mathtt{p}$ such that $q_{\mathtt{p}}$ has an edge ℓ such that ℓ is latest-enabled sending in γ or ℓ is non-deferrable in γ.*

Proof. Let S and γ be as in the statement. Suppose, by contradiction, that, for every $\mathtt{p}$, every edge of $q_{\mathtt{p}}$ is neither latest-enabled sending nor non-deferrable in γ. As a consequence, for all t and for all input labels α, $(\boldsymbol{q}, \boldsymbol{w}, \nu + t) \not\xrightarrow{\alpha}$. It is still possible that $(\boldsymbol{q}, \boldsymbol{w}, \nu + t) \xrightarrow{\alpha}$ for some t and some output label α. If this is not the case, system S does not enjoy progress, and we are done by contradiction. Otherwise, there must be some $\mathtt{p}$ such that $q_{\mathtt{p}}$ has a future-enabled sending edge ℓ in γ. Since $q_{\mathtt{p}}$ has no latest-enabled sending edges, $q_{\mathtt{p}}$ must have a reading edge ℓ' such that $\mathsf{guard}(\ell') \not\leq_\nu \mathsf{guard}(\ell)$. Then, there is t such that $\nu + t \in [\![\mathsf{guard}(\ell')]\!]$ and for all $t' \geq t$, $\nu + t' \notin [\![\mathsf{guard}(\ell)]\!]$. Then, $(\boldsymbol{q}, \boldsymbol{w}, \nu + t') \not\xrightarrow{\mathsf{act}(\ell)\mathsf{msg}(\ell)}$ for all $t' \geq t$. Since every machine has finitely many edges, and S is composed by finitely many machines, iterating the above argument we can find a t such that, for every $t' \geq t$, $(\boldsymbol{q}, \boldsymbol{w}, \nu + t') \not\rightarrow$. Since, for every machine of γ, conditions (d) and (e) of Definition 3 hold trivially for any delay by assumption, $\gamma \xrightarrow{t} (\boldsymbol{q}, \boldsymbol{w}, \nu + t)$, which is stuck. Hence S does not progress: contradiction. $\square$

C.2 LESP and NDP

Definition 18 (Latest-enabled send preservation). *We say that a relation $\mathcal{R}$ between systems is* latest-enabled send preserving *(in short, LESP) iff, whenever $S_1 \; \mathcal{R} \; S_2$, for all $\gamma = (\boldsymbol{q}, \boldsymbol{w}, \nu)$ such that $\gamma_0 \rightarrow^*_{S_1} \gamma$, and for all $\mathtt{p}$, if $q_{\mathtt{p}}$ has a latest-enabled sending edge in γ for S_2, then $q_{\mathtt{p}}$ has a latest-enabled sending edge in γ for S_1.*

Example 12. Recall S and S' from Example 5. The relation $\mathcal{R} = \{(S', S)\}$ is *not* LESP. In S, the sending edge $(p_0, \mathtt{pq}!b(x \leq 3), p_1)$ is latest-enabled in γ_0, but the only sending edge in S', i.e. $(p_0, \mathtt{pq}!b(x \leq 1), p_1)$, is *not* latest-enabled in γ_0. Indeed, from state p_0 of $\mathsf{A}'_{\mathtt{p}}$ there is a receiving edge with guard $x \leq 2$, and $(x \leq 2) \not\leq_{\nu_0} (x \leq 1)$. Now, let $\mathsf{A}''_{\mathtt{p}}$ be equal to $\mathsf{A}'_{\mathtt{p}}$ but for the guard on the send action, which is replaced by $x \leq 2$, and let $S'' = (\mathsf{A}''_{\mathtt{p}}, \mathsf{A}_{\mathtt{q}})$. We have that $\mathcal{R}' = \{(S'', S)\}$ is LESP.

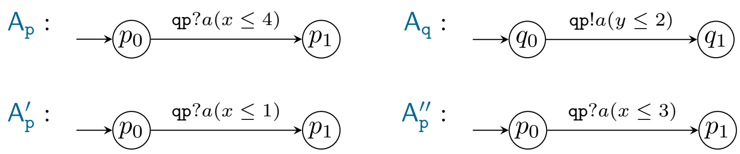
The LESP property is complex to check, in the general case, as it concerns the behaviour of the whole system: in fact, we prove it is undecidable (Theorem 7).

Theorem 7 (Undecidability of LESP). *Establishing whether restrictions of the system refinement $\sqsubseteq_1$ are LESP is undecidable.*

Proof (Sketch). The proof consist in showing that a solution to the problem in the statement would solve (a variation of) the reachability problem for CTAs, known to be undecidable. Let S be a system including a machine A owned by $\mathtt{p}$, and let q be a state of A. The reachability problem asks whether a configuration $(\boldsymbol{q}, \boldsymbol{w}, \nu)$, with $q_{\mathtt{p}} = q$, is reachable from the initial configuration γ_0. If the answer is positive we say that q is reachable in S. Let A' and A'' be two slighly modified copies of A: they differ in the fact that q has only one exiting edge. This edge is sending, and its guard is false for A' and true for A''. Note that $\mathsf{A}' \sqsubseteq \mathsf{A}''$ for $\sqsubseteq \in \{\sqsubseteq_{sr}, \sqsubseteq_a, \sqsubseteq_1\}$ by construction. Let S' and S'' be as S except for A, that is substituted with A' and A'' respectively. Note that q is reachable in S is equivalent to q is reachable in S' and to q is reachable in S''. This follows by the fact that S' and S'' are equal to S, except for the edges exiting from q. We claim that the relation $\{(S', S'')\}$ is not LESP iff q is reachable in S. This follows by the above and by the fact that the only configurations that breaks the LESP property of $\{(S', S'')\}$ are those in the form $(\boldsymbol{q}, \boldsymbol{w}, \nu)$, with $q_{\mathtt{p}} = q$. Since a set is recursive if and only if its complement is, LESP checking is undecidable. $\square$

Definition 19 (Non-deferrable preserving). *We say that a relation $\mathcal{R}$ between systems is* non-deferrable preserving *(in short, NDP) iff, whenever $S_1 \;\mathcal{R}\; S_2$, for all $\gamma = (\boldsymbol{q}, \boldsymbol{w}, \nu)$ such that $\gamma_0 \rightarrow^*_{S_1} \gamma$, and for all $\mathtt{p}$, if $q_{\mathtt{p}}$ has a non-deferrable future-enabled edge in γ for S_2, then $q_{\mathtt{p}}$ has a non-deferrable future-enabled edge in γ for S_1.*

Example 13 (Non-deferrable preserving). Consider the following CTAs:



Let $S = (\mathsf{A}_{\mathtt{p}}, \mathsf{A}_{\mathtt{q}})$, $S' = (\mathsf{A}'_{\mathtt{p}}, \mathsf{A}_{\mathtt{q}})$, $S'' = (\mathsf{A}''_{\mathtt{p}}, \mathsf{A}_{\mathtt{q}})$, let $\mathcal{R}' = \{(S', S)\}$, and $\mathcal{R}'' = \{(S'', S)\}$. Clearly, both $\mathcal{R}'$ and $\mathcal{R}''$ are LESP, because the configurations of S do not have latest-enabled sending edges. We have that $\mathcal{R}'$ is *not* NDP. To show that, let $\gamma = ((p_0, q_1), (\varepsilon, a), \nu_0 + 2)$, which is reachable both in S and S' since $\gamma_0 \xrightarrow{2} \xrightarrow{\mathtt{qp}!a} \gamma$. The only edge of $\mathsf{A}_{\mathtt{p}}$ is non-deferrable in γ, while the edge of $\mathsf{A}'_{\mathtt{p}}$ is deferrable, as it is *not* future-enabled in γ. Instead, $\mathcal{R}''$ is NDP, because the edge of $\mathsf{A}''_{\mathtt{p}}$ is non-deferrable in γ.

Similarly to LESP, also NDP is undecidable (Theorem 8).

Theorem 8 (NDP undecidability). *Establishing whether restrictions of the system refinements $\sqsubseteq_{sr}$, $\sqsubseteq_a$, $\sqsubseteq_1$ are NDP is undecidable.*

We now formally define naïve ($\sqsubseteq_{sr}$) and asymmetric ($\sqsubseteq_a$) refinements. $\sqsubseteq_{sr}$ will be useful for proving properties of $\sqsubseteq_1$.

Definition 20. *Let $\mathsf{A} = (Q, q_0, X, E)$ and $\mathsf{A}' = (Q, q_0, X, E')$ be CTAs. The relation $\mathsf{A}' \sqsubseteq_{sr} \mathsf{A}$ holds whenever there exists a structure-preserving isomorphism $f : E \rightarrow E'$*

such that, for all edges $(q, \ell, q') \in E$*, if* $f(q, \ell, q') = \ell'$*, then* $[\![\mathsf{guard}(\ell')]\!] \subseteq [\![\mathsf{guard}(\ell)]\!]$*. The relation* $\mathsf{A}' \sqsubseteq_a \mathsf{A}$ *holds whenever there exists a structure-preserving isomorphism* $f : E \to E'$ *such that, for all edges* $(q, \ell, q') \in E$*, if* $f(q, \ell, q') = \ell'$ *and* ℓ *is sending, then* $[\![\mathsf{guard}(\ell')]\!] \subseteq [\![\mathsf{guard}(\ell)]\!]$*; if* $f(q, \ell, q') = \ell'$ *and* ℓ *is reading, then* $[\![\mathsf{guard}(\ell)]\!] \subseteq [\![\mathsf{guard}(\ell')]\!]$

Lemma 10. *Let* $\sqsubseteq$ *be a LESP & NDP restriction of* $\sqsubseteq_{sr}$*. Then, for all systems* S_1 *and* S_2 *such that* $S_1 \sqsubseteq S_2$ *and for all* γ, γ'*:*

$$\gamma_0 \rightarrow^*_{S_1} \gamma \xrightarrow{\alpha}_{S_1} \gamma' \implies \gamma \xrightarrow{\alpha}_{S_2} \gamma'$$

Proof. Suppose $S_1 \sqsubseteq S_2$, with isomorphism f, and $\gamma_0 \rightarrow^*_{S_1} \gamma = (\boldsymbol{q}, \boldsymbol{w}, \nu) \xrightarrow{\alpha}_{S_1} \gamma' = (\boldsymbol{q'}, \boldsymbol{w'}, \nu')$. We proceed by cases on the rule of Definition 3 used in the derivation $\gamma \xrightarrow{\alpha}_{S_1} \gamma'$.

For rule item 1, it must be $\alpha = \mathtt{pr}!a$ and $q_\mathtt{p} \xrightarrow{\ell = \mathtt{pr}!a(\delta, \lambda)}_{S_1} q'_\mathtt{p}$, for some $\mathtt{p}, \mathtt{r}, a, \delta, \lambda$ such that $\nu \in [\![\delta]\!]$. By definition 6:

$$q_\mathtt{p} \xrightarrow{f(\ell) = \mathtt{pr}!a(\delta', \lambda)}_{S_2} q'_\mathtt{p}$$

for some δ' such that $[\![\delta]\!] \subseteq [\![\delta']\!]$. Then, $\nu \in [\![\delta']\!]$ and hence, by rule item 1, $\gamma \xrightarrow{\alpha}_{S_2} \gamma'$.

The case for rule item 2 is similar.

For rule item 3, it must be $\alpha = t$, for some t, and $\gamma' = (\boldsymbol{q}, \boldsymbol{w}, \nu + t)$. Hence, we have to show:

$$\gamma \xrightarrow{t}_{S_2} \gamma'$$

The only possible rule for the above transition is item 3. Items (a),(b) and (c) clearly hold for S_2 as well. It remains to show it is the case also for items (d) and (e).

For item (d), suppose $e_{S_2} = (q_\mathtt{p}, \ell_{S_2}, q'_\mathtt{p})$ is a non-deferrable edge of $q_\mathtt{p}$ with respect to S_2 in γ, for some $\mathtt{p}$. We have to show there is an edge $e'_{S_2} = (q_\mathtt{p}, \ell'_{S_2}, q''_\mathtt{p})$ in S_2 that is non-deferrable in γ'. Since $\sqsubseteq$ is NDP, $q_\mathtt{p}$ must have an edge ℓ_{S_1} non-deferrable in γ for S_1, and hence, since condition (d) holds for S_1 by the assumption $\gamma \xrightarrow{t}_{S_1} \gamma'$, it holds that $q_\mathtt{p} \xrightarrow{\ell'_{S_1}}_{S_1} q''_\mathtt{p}$ for some $q''_\mathtt{p}, \ell'_{S_1}$ such that ℓ'_{S_1} is non-deferrable in γ'. Now, let $(q_\mathtt{p}, \ell'_{S_2}, q''_\mathtt{p}) = e'_{S_2}$ be the unique edge of S_2 such that $e'_{S_1} = f(e'_{S_2})$. Since, by definition 6, $[\![\mathsf{guard}(\ell'_{S_1})]\!] \subseteq [\![\mathsf{guard}(\ell'_{S_2})]\!]$, ℓ'_{S_2} is future-enabled in $\nu + t$, and hence item (d) holds for S_2.

For item (e), suppose that, for some p, $q_\mathtt{p}$ has a latest enabled (with respect to S_2) sending action ℓ in γ. We have to show ℓ is future enabled in ν'. Since $\sqsubseteq$ is latest-enabled send preserving, $q_\mathtt{p}$ has a latest-enabled (with respect to S_1) sending edge $(q_\mathtt{p}, \ell', q''_\mathtt{p})$ in ν'. By definition 6, it follows that $(q_\mathtt{p}, f(\ell'), q''_\mathtt{p}) \in E_\mathtt{p}$ for S_2, and $\mathsf{guard}(\ell') \subseteq \mathsf{guard}(f(\ell'))$, and hence $(q_\mathtt{p}, f(\ell'), q''_\mathtt{p})$ is future-enabled in ν'. Now, since ℓ is latest-enabled with respect to S_2, by definition 2 it follows $\mathsf{guard}(f(\ell')) \leq_{nu} \mathsf{guard}(\ell)$. Hence ℓ is future-enabled in ν' as well. $\square$

Lemma 11. *LESP+NDP restrictions of* $\sqsubseteq_{sr}$ *preserve behaviour.*

Proof. Let $\sqsubseteq$ be a LESP & NDP restriction of $\sqsubseteq_{sr}$, and let S_1 and S_2 be systems such that $S_1 \sqsubseteq_{sr} S_2$. We have to show there is a timed simulation r between states of

$S_1 \uplus S_2$ that relates the initial configuration of S_1 with the initial configuration of S_2, i.e. $((1, s_0), (2, s_0)) \in r$. Define:

$$r \stackrel{\text{def}}{=} \left\{ ((1, \gamma), (2, \gamma)) \,\middle|\, \gamma_0 \rightarrow^*_{S_1} \gamma \right\}$$

Clearly, $((1, \gamma_0), (2, \gamma_0))$ is a member of r. The fact that r is a timed simulation is an immediate consequence of lemma 10. $\square$

Lemma 12. *LESP+NDP restrictions of $\sqsubseteq_{sr}$ preserve global progress.*

Proof. Let S_1, S_2 and $\sqsubseteq$ be as in the statement, and suppose that S_2 has global progress. We have to show that S_1 progress. Suppose $\gamma_0 \rightarrow^*_{S_1} \gamma = (\boldsymbol{q}, \boldsymbol{w}, \nu)$. If γ is final we are done. If not, by lemma 11, it follows $\gamma_0 \rightarrow^*_{S_2} \gamma$ as well. Since γ is not final also with respect to S_2, $\gamma \xrightarrow{t}_{S_2} \xrightarrow{\alpha}_{S_2}$, for some t, α. By lemma 5, $BRE^{S_2}_{\gamma}(t) = \emptyset$, and, by lemma 6, $BRE^{S_2}_{\gamma}(0) = \emptyset$ as well. We wish to prove there is a t and a $\alpha \notin \mathbb{R}_{\geq 0}$ such that $(\boldsymbol{q}, \boldsymbol{w}, \nu + t) \xrightarrow{\alpha}_{S_1}$ and $BRE^{S_1}_{\gamma}(t) = \emptyset$. Since S_2 progress, there is a machine q_{p} of S_2 that has a latest-enabled sending or a non-deferrable edge in γ. Then, by the LESP & NDP assumption, q_{p} enjoys the same property with respect to S_1. Now, among the set of non-deferrable edges in γ with respect to S_1, pick a minimal element $e_{S_1} = (q_{\mathrm{p}}, \ell_{S_1}, q'_{\mathrm{p}}$ with respect to the preorder $\leq_\nu$. Such an element exists because $\leq_\nu$ is total and the set is finite and not empty. Then, since ℓ_{S_1} is future-enabled in γ, there is some t such that $(\boldsymbol{q}, \boldsymbol{w}, \nu + t) \xrightarrow{\alpha}_{S_1}$, where α is the action associated to ℓ_{S_1}. Since e_{S_1} is minimal with respect to $\leq_\nu$, every non-deferrable edge in γ of S_1 is non-deferrable in $(\boldsymbol{q}, \boldsymbol{w}, \nu + t)$. Therefore, by lemma 7, $\gamma \xrightarrow{t'}_{S_1} \xrightarrow{\alpha'}_{S_1}$ for some t' and $\alpha' \notin \mathbb{R}_{\geq 0}$. $\square$

Lemma 13. *Let S_1 and S_2 be systems of machines such that S_2 progress $S_1 \sqsubseteq S_2$ for some LESP & NDP restriction $\sqsubseteq$ of $\sqsubseteq_{sr}$. Then:*

$$\rho \text{ is a maximal run of } S_1 \implies \rho \text{ is a maximal run of } S_2$$

Proof. Let S_1 and S_2 be as in the statement, and suppose ρ is a maximal run of S_1. We first show that ρ is a run of S_2, and then we show it is maximal for S_2. For the first part, it suffice to show that, for all i such that $\gamma_i \xrightarrow{t_i}_{S_1} {\gamma'}_i \xrightarrow{\alpha_i}_{S_1} \gamma_{i+1}$ appairs in ρ, it holds that $\gamma_i \xrightarrow{t_i}_{S_2} \gamma'_i \xrightarrow{\alpha_i}_{S_2} \gamma_{i+1}$. But this follows by lemma 10. For the second part, i.e. ρ is maximal with respect to S_2, first note that if ρ is infinite the thesis is trivial. So, suppose ρ is finite, with last state γ_n. Since ρ is maximal with respect to S_1, $\neg \gamma_n \xrightarrow{t}_{S_1} \xrightarrow{\alpha}_{S_1}$ for all t, α. But then, since S_2 progress, by lemma 12 S_1 progress as well, and thus γ_n is final for S_1. Therefore, γ_n is final for S_2 too, and ρ is maximal for S_2. $\square$

Lemma 14. *LESP+NDP restrictions of $\sqsubseteq_{sr}$ preserve local progress.*

Proof. Suppose S_2 has local progress. By lemma 3, S_2 has global progress as well. Then, by lemma 13, maximal runs of S_1 are maximal runs of S_2. Therefore, by lemma 8, S_1 has local progress. $\square$

Theorem 9. *LESP+NDP restrictions of $\sqsubseteq_{sr}$ preserve behaviour, global and local progress.*

Proof. Composition of lemma 11, lemma 12 and lemma 14. $\square$

Lemma 15. *$\sqsubseteq_1$ is a NDP restriction of $\sqsubseteq_{sr}$.*

Proof. The fact that $\sqsubseteq_1$ is a restriction of $\sqsubseteq_{sr}$ follows by an easy inspection of definition 6. It remains to show $\sqsubseteq_1$ is NDP. Let A_1 and A_2 be systems of machines such that $A_1 \sqsubseteq_1 A_2$, with isomorphism f, and let $\gamma = (\boldsymbol{q}, \boldsymbol{w}, \nu)$ be such that $\gamma_0 \rightarrow^*_{S_1} \gamma = (\boldsymbol{q}, \boldsymbol{w}, \nu)$. Suppose $q_{\mathtt{p}}$ has a non-deferrable future-enabled edge $e = (q_{\mathtt{p}}, \ell, q'_{\mathtt{p}})$ in γ for S_2. Then, $f(e) = (q_{\mathtt{p}}, \ell', q'_{\mathtt{p}})$, for some ℓ'. Note that ℓ' is non-deferrable in γ for S_1. It remains to show ℓ' is future-enabled in γ. Since ℓ is future-enabled in γ and, by definition 6, $\downarrow [\![\mathsf{guard}(\ell)]\!] \subseteq \downarrow [\![\mathsf{guard}(\ell)]\!]$, ℓ' is future-enabled in γ. □

Theorem 10. *LESP restrictions of $\sqsubseteq_1$ preserve behaviour, global and local progress.*

Proof. Composition of theorem 9 and lemma 15. □

C.3 Local LESP

Lemma 16. *For all $S = (\mathsf{A}_{\mathtt{p}})_{\mathtt{p} \in \mathcal{P}}$, for all $\gamma = (\boldsymbol{q}, \boldsymbol{w}, \nu)$ such that $\gamma_0 \rightarrow^* \gamma$, and for all $\mathtt{p}$: $\nu \in Post^{\mathsf{A}_{\mathtt{p}}}_{q_{\mathtt{p}}}(Pre^{\mathsf{A}_{\mathtt{p}}}_{q_{\mathtt{p}}})$.*

Proof. Let S be as in the statement. We show the thesis holds for all $\gamma = (\boldsymbol{q}, \boldsymbol{w}, \nu)$ and for all n such that $\gamma_0 \rightarrow^* \gamma$. By induction on n. For the base case, it must be $\gamma = \gamma_0$, and since all $q_{\mathtt{p}}$ are initial in the respective machines, $\nu_0 \in Pre^{\mathsf{A}_{\mathtt{p}}}_{q_{\mathtt{s}}} \subseteq Post^{\mathsf{A}_{\mathtt{p}}}_{q_{\mathtt{s}}}(Pre^{\mathsf{A}_{\mathtt{p}}}_{q})$ for all $\mathtt{p}$. For the inductive case, let $\gamma = (\boldsymbol{q}, \boldsymbol{w}, \nu)$ be such that $\gamma_0 \rightarrow^n \gamma' \rightarrow \gamma$ for some $\gamma' = (\boldsymbol{q'}, \boldsymbol{w'}, \nu')$. We proceed by cases on the rule of definition 3 used for deriving $\gamma' \rightarrow \gamma$.

- Rule item 1. It must be $\alpha = \mathtt{pr}!a$, $(q'_{\mathtt{p}}, \alpha(\delta, \lambda), q_{\mathtt{p}}) \in E_{\mathtt{p}}$, $\nu = \lambda(\nu')$ and $\nu' \in [\![\delta]\!]$. Since machines do not share cloks, by the induction hypothesis it follows $\nu \in Post^{\mathsf{A}_{\mathtt{s}}}_{q_{\mathtt{s}}}(Pre^{\mathsf{A}_{\mathtt{s}}}_{q_{\mathtt{s}}})$ for all participant $\mathtt{s} \neq \mathtt{p}$. For $\mathtt{p}$, note that $\nu \in Pre^{\mathsf{A}_{\mathtt{p}}}_{q_{\mathtt{p}}}$. Therefore $\nu \in Post^{\mathsf{A}_{\mathtt{p}}}_{q_{\mathtt{p}}}(Pre^{\mathsf{A}_{\mathtt{p}}}_{q_{\mathtt{p}}})$.
- Rule item 2. Similar to the above.
- Rule item 3. It must be $\alpha = t$, $\boldsymbol{q} = \boldsymbol{q'}$, $\boldsymbol{w} = \boldsymbol{w'}$, $\nu = \nu' + t$ and conditions (d) and (e) hold. By the induction hypothesis $\nu' \in Post^{\mathsf{A}_{\mathtt{p}}}_{q_{\mathtt{p}}}(Pre^{\mathsf{A}_{\mathtt{p}}}_{q_{\mathtt{p}}})$. The thesis follows by condition (d).

□

We recall some operations on sets of clock valuations from [7], that can be lifted to guards. They are instrumental in the proof of the decidability of LLESP.

Definition 21. *For all sets of clock valuations K, and for all reset sets λ, we define:*

$$\uparrow K \stackrel{def}{=} \{\nu + t \mid \nu \in K\}$$
$$\lambda(K) \stackrel{def}{=} \{\lambda(\nu) \mid \nu \in K\}$$

Below we define the sets of clock valuations that satisfies, respectively, the guard of a sending edge and the guard of a receiving edge.

Definition 22. *For all* A *and for all* q *state of* A*, we define the following sets of clock valuations:*

$$q^! \stackrel{def}{=} \{\nu \mid \exists \mathtt{p}, \mathtt{r}, a, \delta, \lambda : q \xrightarrow{\mathtt{pr}!a(\delta,\,\lambda)} \wedge \nu \in [\![\delta]\!]\}$$
$$q^? \stackrel{def}{=} \{\nu \mid \exists \mathtt{p}, \mathtt{r}, a, \delta, \lambda : q \xrightarrow{\mathtt{rp}?a(\delta,\,\lambda)} \wedge \nu \in [\![\delta]\!]\}$$

We define the set of guards $\mathsf{RGuards}(q)$ *in the following way (*λ *below is lifted to guards):*

$$\mathsf{RGuards}(q) \stackrel{def}{=} \{\lambda(\delta) \mid \exists \ell : q \xrightarrow{\ell} \wedge\, \delta = \mathsf{guard}(\ell) \wedge \lambda = \mathsf{reset}(\ell)\}$$

And we let δ_0 *be the guard that equals every clock to zero.*

Below, the symbol $\setminus$ denotes set difference.

Lemma 17. *For all* A*, for all* q *state of* A*, and for all* K*, we have that:*

1. $Pre_q^{\mathsf{A}} = \begin{cases} [\![(\bigvee_{\delta \in \mathsf{RGuards}(q)} \delta) \vee \delta_0]\!] & \text{if } q = q_0 \\ [\![\bigvee_{\delta \in \mathsf{RGuards}(q)} \delta]\!] & \text{otherwise} \end{cases}$
2. $Les_q^{\mathsf{A}} = \downarrow (q^! \setminus \downarrow (q^? \setminus \downarrow q^!))$.
3. $Post_q^{\mathsf{A}}(K) = \uparrow (K \setminus Les_q^{\mathsf{A}}) \cup (\uparrow K \cap Les_q^{\mathsf{A}})$.

Proof. Item 1 follows immediately by the semantics of guards in section 2. For item 2, first note that $\downarrow (q^! \setminus \downarrow (q^? \setminus \downarrow q^!)) =$

$$\{\nu \mid \exists t : \nu + t \in q^! \wedge (\forall t' \geq t : \nu + t' \in q^? \implies \exists t'' \geq t' : \nu + t'' \in q^!)\} \tag{2}$$

Indeed:

$$\begin{array}{lr}
\downarrow (q^! \setminus \downarrow (q^? \setminus \downarrow q^!)) & = \\
\downarrow (q^! \setminus \downarrow (\{\nu \mid \nu \in q^?\} \setminus \{\nu \mid \exists t : \nu + t \in q^!\})) & = \\
\downarrow (q^! \setminus \downarrow (\{\nu \mid \nu \in q^? \wedge \forall t : \nu + t \notin q^!\})) & = \\
\downarrow (q^! \setminus \{\nu \mid \exists t : \nu + t \in q^? \wedge \forall t' \geq t : \nu + t' \notin q^!\}) & = \\
\downarrow \{\nu \mid \nu \in q^! \wedge (\forall t : \nu + t \in q^? \implies \exists t' \geq t : \nu + t' \in q^!)\} & = \\
\{\nu \mid \exists t : \nu + t \in q^! \wedge (\forall t' \geq t : \nu + t' \in q^? \implies \exists t'' \geq t' : \nu + t'' \in q^!)\} &
\end{array}$$

Now, suppose that $\nu \in Les_q^{\mathsf{A}}$, i.e. q has a latest-enabled sending edge in ν. Then, $q \xrightarrow{\ell}$ for some sending action ℓ with guard δ such that there is $t : \nu + t \in [\![\delta]\!]$ and, for all ℓ' such that $q \xrightarrow{\ell'}$, it holds that $\mathsf{guard}(\ell') \leq_\nu \delta$. Then, since ℓ is sending, it follows $\nu + t \in q^!$ with t as above. By definition of $\leq_\nu$ (definition 2), it follows that ν satisfies: $(\forall t' \geq t : \nu + t' \in q^? \implies \exists t'' \geq t' : \nu + t'' \in q^!)$. Therefore, by eq. (2), $\nu \in \downarrow (q^! \setminus \downarrow (q^? \setminus \downarrow q^!))$. For the converse, suppose $\nu \in \downarrow (q^! \setminus \downarrow (q^? \setminus \downarrow q^!))$. Then, q has some future-enabled sending edge in ν. Let ℓ be the action associated to the latest-enabled (in ν) among sending edges of q. It must exists because $\leq_\nu$ is total, q has finitely many edges, and a latest-enabled sending edge of q exists. Now, suppose $q \xrightarrow{\ell'}$, for some ℓ'. We have to show $\mathsf{guard}(\ell') \leq_\nu \mathsf{guard}(\ell)$. If ℓ' is sending the thesis follows by the assumption that ℓ is latest-enabled among sending edges. If ℓ' is receiving, suppose $\nu + t' \in \mathsf{guard}(\ell')$, for some t'. By eq. (2), there is some t such that $\nu + t \in q^!$. If $t' \geq t$, there is some $t'' \geq t$ such that $t'' \in q^!$, and hence, since ℓ is latest-enabled among sending edges, ℓ is future-enabled in $\nu + t''$ and we are done. If $t' < t$, it follows ℓ future-enabled in t' with an argument similar to above, and we are done.

For item 3:

$$\begin{aligned}
Post_q^{\mathsf{A}}(K) &= \left\{\nu + t \,\middle|\, \nu \in K \ \wedge \ (\nu \in Les_q^{\mathsf{A}} \implies \nu + t \in Les_q^{\mathsf{A}})\right\} \\
&= \left\{\nu + t \,\middle|\, \nu \in K \ \wedge \ (\nu \notin Les_q^{\mathsf{A}} \vee \nu + t \in Les_q^{\mathsf{A}})\right\} \\
&= \left\{\nu + t \,\middle|\, (\nu \in K \wedge \nu \notin Les_q^{\mathsf{A}}) \vee (\nu \in K \wedge \nu + t \in Les_q^{\mathsf{A}})\right\} \\
&= \left\{\nu + t \,\middle|\, \nu \in K \wedge \nu \notin Les_q^{\mathsf{A}}\right\} \cup \left\{\nu + t \,\middle|\, \nu \in K \wedge \nu + t \in Les_q^{\mathsf{A}}\right\} \\
&= \uparrow \left\{\nu \,\middle|\, \nu \in K \wedge \nu \notin Les_q^{\mathsf{A}}\right\} \cup (\{\nu + t \,|\, \nu \in K\} \cap \left\{\nu \,\middle|\, \nu \in Les_q^{\mathsf{A}}\right\}) \\
&= \uparrow (K \setminus Les_q^{\mathsf{A}}) \cup (\uparrow K \cap Les_q^{\mathsf{A}})
\end{aligned}$$

□

Theorem 11 (From LLESP to LESP). *System refinements induced by LLESP point-wise refinements are LESP.*

Proof. Let $\sqsubseteq$ be a system refinement induced by some locally LESP point-wise refinement, and let $S_1 = (\mathsf{A}^1_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$ and $S_2 = (\mathsf{A}^2_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$ be systems of machines such that $S_1 \sqsubseteq S_2$. Suppose that $\gamma_0 \rightarrow^*_{S_1} \gamma = (\boldsymbol{q}, \boldsymbol{w}, \nu)$, and that $q_{\mathtt{p}}$ has a latest-enabled sending edge e_{S_2} in γ for S_2, i.e. $\nu \in Les_{q_{\mathtt{p}}}^{\mathsf{A}^2_{\mathtt{p}}}$. We have to show that $q_{\mathtt{p}}$ has a latest-enabled sending edge e_{S_2} in γ for S_1, i.e. $\nu \in Les_{q_{\mathtt{p}}}^{\mathsf{A}^1_{\mathtt{p}}}$. By lemma 16, $\nu \in Post_{q_{\mathtt{p}}}^{\mathsf{A}^1_{\mathtt{p}}}(Pre_{q_{\mathtt{p}}}^{\mathsf{A}^1_{\mathtt{p}}})$. Hence, since $\sqsubseteq$ is locally LESP by assumption, $\nu \in Post_{q_{\mathtt{p}}}^{\mathsf{A}^1_{\mathtt{p}}}(Pre_{q_{\mathtt{p}}}^{\mathsf{A}^1_{\mathtt{p}}}) \cap Les_{q_{\mathtt{p}}}^{\mathsf{A}^1_{\mathtt{p}}}$ and hence $\nu \in Les_{q_{\mathtt{p}}}^{\mathsf{A}^1_{\mathtt{p}}}$. □

Proof of Theorem 3

Preservation follows by theorems 10 and 11. Decidability follows by the fact that, by lemma 17, we can effectively construct guards that represents $Post_q^{\mathsf{A}}(Pre_q^{\mathsf{A}}) \cap Les_q^{\mathsf{A}'}$ and $Post_q^{\mathsf{A}}(Pre_q^{\mathsf{A}}) \cap Les_q^{\mathsf{A}}$ respectively, and checking wether $[\![\delta]\!] \subseteq [\![\delta']\!]$ is decidable.

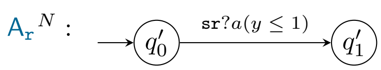
□

C.4 Counter-examples for alternative refinement strategies

Both 'naïve' and 'asymmetric' strategies have been formally defined elsewhere in this appendix (Definition 20) and give two refinements denoted with $\sqsubseteq_{sr}$ and $\sqsubseteq_a$, respectively.

Example 14 (Fact 4: counter-example for naïve strategy ($\sqsubseteq_{sr}$)). Consider the following system composed of two CTAs:

$\mathsf{A_s}$ and $\mathsf{A_r}$ can be refined, using $\sqsubseteq_{sr}$. We let $\mathsf{A_s}$ unchanged and narrow guards in $\mathsf{A_r}$, yielding ${\mathsf{A_r}}^N$ below, with ${\mathsf{A_r}}^N \sqsubseteq_{sr} \mathsf{A_r}$:



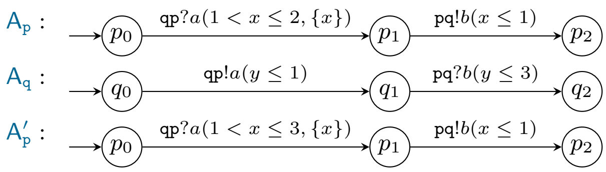
Let $S = (\mathsf{A_s}, \mathsf{A_r})$, and $S' = (\mathsf{A_s}, \mathsf{A_r}^N)$. The relation $\mathcal{R} = \{(S', S)\}$ is LLESP, as the sending edge remains such in S'. Consider the following run (common to S and S'):

$$\gamma_0 \xrightarrow{2} \xrightarrow{\mathtt{sr}!a} \gamma \;=\; ((q_1, q_0'), (a, \varepsilon), \nu_0 + 2)$$

In S' we have $\gamma \xrightarrow{1}$, while in S the only possible timed transition is $\gamma \xrightarrow{0}$. Hence, behaviour is *not* preserved. Since, in S, γ can perform the receive and reach the final configuration, then S enjoys (local/global) progress. Instead, in S' it is too late to receive ($y \leq 1$ is unsatisfiable from $\nu_0 + 2$), hence S' does *not* enjoy (local/global) progress.

Intuitively, the problem with Example 14 is that narrowing the constraints of receiving edges may disable them before the message has been sent.

Example 15 (Fact 4: counter-example for asymmetric strategy ($\sqsubseteq_a$)).
Consider the following CTAs:



Let $S = (\mathsf{A_p}, \mathsf{A_q})$ and $S' = (\mathsf{A_p'}, \mathsf{A_q})$. We have that $\mathsf{A_p'} \sqsubseteq_a \mathsf{A_p}$. The relation $\{(\mathsf{A_p'}, \mathsf{A_p})\}$ is LLESP. However, behaviour is not preserved, because in S' we have the run $\gamma_0 \xrightarrow{\mathtt{qp}!a} \xrightarrow{3}$, while in S we have $\gamma_0 \xrightarrow{\mathtt{qp}!a} \xrightarrow{t}$ only if $t \leq 2$. Progress is not preserved as well. Indeed, S enjoys global/local progress, while in S' we have:

$$\begin{aligned}
\gamma_0 &\xrightarrow{\mathtt{qp}!a} ((p_0, q_1), (\varepsilon, a), \nu_0) \\
&\xrightarrow{3} ((p_0, q_1), (\varepsilon, a), \nu_0 + 3) \\
&\xrightarrow{\mathtt{qp}?a} ((p_1, q_1), (\varepsilon, \varepsilon), \{x \mapsto 0, y \mapsto 3\}) \\
&\xrightarrow{1} ((p_1, q_1), (\varepsilon, \varepsilon), \{x \mapsto 1, y \mapsto 4\}) \\
&\xrightarrow{\mathtt{pq}!b} ((p_2, q_1), (b, \varepsilon), \{x \mapsto 1, y \mapsto 4\})
\end{aligned}$$

Since the last configuration in the run is stuck, S' does not enjoy progress.

C.5 Proofs for Section 5

Example 16. Consider again the system $(\mathsf{A_1}, \mathsf{A_2})$ with the CTAs in Figure 2. According to the non-urgent semantics, a possible run would be (recalling from Section 2):

$$\gamma_0 \xrightarrow{2} \gamma_1 \xrightarrow{\mathtt{sr}!a} \gamma_2 \xrightarrow{1.5} \gamma_3 \xrightarrow{\mathtt{rs}?a} \gamma_4$$

Note that $\gamma_2 \xrightarrow{\mathtt{rs}?a}$. Hence, the second clause of Definition 13 states that $\gamma_2 \not\xrightarrow{t}$ for all $t > 0$. Then, a maximal run of $(\mathsf{A_1}, \mathsf{A_2})$ under the urgent semantics would be:

$$\gamma_0 \xrightarrow{2}_u \gamma_1 \xrightarrow{\mathtt{sr}!a}_u \gamma_2 \xrightarrow{\mathtt{rs}?a}_u \gamma_4' = ((q_1, q_3), (\varepsilon, \varepsilon), \nu_0 + 2)$$

Proof of Theorem 5

A simple inspection of definition 13. □

Proof of Theorem 6

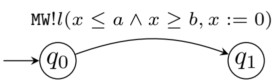
Let S be as in the statement, and suppose S has global progress. We have to show S has global progress also with the urgent semantics. So, suppose $\gamma_0 \rightarrow_u^* \gamma = (\boldsymbol{q}, \boldsymbol{w}, \nu)$. First note that, by theorem 5, $\gamma_0 \rightarrow^* \gamma$ as well. If γ is final, we are done. If not, there are t and α such that $\gamma \xrightarrow{t} \xrightarrow{\alpha}$. If $\gamma \xrightarrow{t}_u$ we are done. If not, there must be some p and $t' < t$ such that q_p has non-deferrable edge in $(\boldsymbol{q}, \boldsymbol{w}, \nu + t')$. Among these edges, pick the minimum element $e = q_\mathrm{p}, \ell, q'_\mathrm{p}$ with respect to $\leq_\nu$. It exists because there are finitely many such edges and $\leq_\nu$ is total. Now, take the least t such that $\nu + t \in [\![\mathsf{guard}(\ell)]\!]$. The existence of such a t follows by the fully left closed assumption. Clearly, $\gamma \xrightarrow{t}_u \xrightarrow{\alpha}$, where α is the label associated to ℓ. $\square$

D Additional material for Section 6

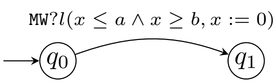
Below we give a (non exhaustive) illustrative set of implementation patterns from CTA to Go programs: sending (resp. receiving) edge implemented with a non-blocking primitive, receiving edge implemented with a blocking primitive with timeout, and mixed choice with timeout implemented with non-blocking send and blocking receive. We assume that variable u contains the time unit used in the application. This can be set to, say, one millisecond, with the following line of code:

```
u := time.Millisecond
```

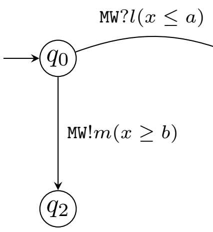
We also assume $a \leq b$.




```
x := time.Now() // x initially 0
time.Sleep(time.Now().Add(n * u).Sub(x)) //a<=n<=b
x = time.Now() // x is reset
MW <- "lab"
```




```
time.Sleep(time.Now().Add(a * u).Sub(x))
select {
 case res := <- WM: //message received in time
  ...
 case <- time.After(time.Now().Add(b * u).
   Sub(x)): //timeout
  ...
```




```
select {
 case res := <- WM:
  //implementation of q1
 case <- time.After(time.Now().Add(a * u).Sub(x)):
  time.Sleep(time.Now().Add(n * u).Sub(x)) //b<=n
  MW <- "m"
```