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 on timed asynchronous systems, in the formal setting of Communicating Timed Automata. In particular, we introduce three notions of refinement for systems of CTAs, which are compositional (i.e., the refinement of a system is obtained by refining its components) and decidable. We establish conditions under which our refinements preserve good behavioural properties of systems, like e.g. their global and local progress. Our theory can be used to refine abstract models into concrete ones: our preservation results guarantee that, under some decidable conditions, the properties of the abstract model will still hold in the concrete one. All our results build upon new semantics of CTAs which faithfully model the behaviour of real-time primitives available in mainstream programming languages.

1 Introduction

Formal reasoning and verification of real-time computing systems are supported by well-established theories and frameworks based, for instance, on timed automata [1,31, 42]. In the standard theory of timed automata, communication between components is *synchronous*, i.e. a component can send a message only when its counterpart is ready to receive it. However, in many concrete distributed systems, e.g. web-based systems, communication is *asynchronous*, and it is often implemented through middlewares that support for FIFO messaging [2,39]. These systems can be modelled as Communicating Timed Automata (CTA) [29], an extension of timed automata featuring 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 [21] and in the untimed [13] setting. Several works propose restrictions of the general model, or sound approximate techniques, for the verification of systems of CTAs [10, 21]. However, all 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 (and ultimately, between models and implementations) is usually expressed as a *refinement* relation. This notion has been used, for instance, to create abstract models which enhance effectiveness of verification techniques (e.g., abstraction refinement [24, 40], time-wise refinement [38]), or to concretize abstract models into implementations [20, 22].

Existing notions of refinement between timed models are based on *synchronous* communications [5, 17, 25, 32]. Asynchronous refinement has been investigated only in the *untimed* setting, under the name of subtyping between session types [6, 19, 23, 33–35]. Remarkably, it was recently shown that asynchronous subtyping is undecidable [14, 30] in the untimed setting. To our knowledge, a notion of refinement in an asynchronous and timed setting has only been studied in [11], as a relationship between timed models (timed multiparty session types) and their implementations (processes in

an extended π-calculus). However, the work in [11] has two main limitations. First, the model considered in [11] is not as general as Communicating Timed Automata. In particular it does not allow to express *mixed states*, namely states with both sending and receiving outgoing transitions. Mixed states are, in fact, crucial to capture common programming patterns like *timeouts* [36], used e.g. by a server which waits for a message, and, if nothing is received within a certain deadline, sends a timeout notification. This pattern is supported, for example, by the receive/after construct of Erlang [3], and by some (untimed) session typed calculi [6, 16, 28]. The second limitation of [11] is that in its calculus, the timing of an action is scheduled to be a precise point in time, which is defined statically. Hence, the calculus cannot model e.g., real-world blocking receive primitives.

To be usable in practice, a theory of refinements for timed asynchronous systems should support for real-world programming patterns (e.g., timeouts à la Erlang), and feature *decidable* notions of refinement. Further, refinement should be *compositional* (namely, 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 refinement on timed asynchronous systems, in the formal setting of Communicating Timed Automata. In particular, we introduce three notions of refinement for systems of CTAs, which are compositional (i.e., the refinement of a system is obtained by refining its components) and decidable. We establish conditions under which our refinements preserve good behavioural properties of systems, like e.g. their global and local progress. Our theory can be used to refine abstract models into concrete ones: our preservation results guarantee that, under some decidable conditions, the properties of the abstract model will still hold in the concrete one. All our results build upon new semantics of CTAs which faithfully model the behaviour of real-time primitives available in mainstream programming languages.

In the rest of this section we give a detailed outline of our contributions.

New semantics of CTAs. Our first contribution is a new semantics of systems of CTAs, both in the non-urgent and urgent versions (Definitions 4 and 6, respectively). These semantics depart from the original semantics in [29], which always allows time to elapse, even when this prevents the system from performing any available action. For instance, consider the following two CTAs, which we call $\mathsf{A}_{\mathtt{s}}$ (for *sender*) and $\mathsf{A}_{\mathtt{r}}$ (for *receiver*):

The CTA $\mathsf{A}_{\mathtt{s}}$ models a participant $\mathtt{s}$ who wants to send a message a to another participant $\mathtt{r}$. The guard $x \leq 2$ on the edge between q_0 and q_1 is a time constraint, stating that the message must be sent within 2 time units. Dually, $\mathsf{A}_{\mathtt{r}}$ models a participant $\mathtt{r}$ who wants to receive the message a from $\mathtt{s}$ within 3 time units.

In the semantics of [29], a possible (partial) computation of the system composed of $\mathsf{A}_{\mathtt{s}}$ and $\mathsf{A}_{\mathtt{r}}$ would be the following:

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

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

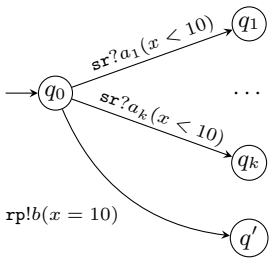


Fig. 1: The `receive`/`after` pattern of Erlang (left), and the corresponding CTA (right).

The tuple 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 `r` and `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 transition of the system, which corresponds to a delay of 5 time units.

Note that this computation does not correspond to any reasonable behaviour of the modelled protocol.[3] What we would expect, is that the send action is performed *before* the deadline expires. Our semantics faithfully models this intuition, by requiring that the elapsing of time does not prevent the send action in $\mathsf{A}_\mathtt{s}$ to be performed. This means that we can procrastinate the send for 2 time units, but then time cannot pass any longer, and the only possible action is the send:

$$\begin{aligned}((q_0, q_0'), (\varepsilon, \varepsilon), \{x, y \mapsto 0\}) &\xrightarrow{2} ((q_0, q_0'), (\varepsilon, \varepsilon), \{x, y \mapsto 2\}) \\ &\xrightarrow{\mathtt{sr}!a} ((q_1, q_0'), (a, \varepsilon), \{x, y \mapsto 2\})\end{aligned}$$

Formally, in Theorem 1 we prove that our semantics enjoys a form of *persistency*, which is asymmetric in the way it deals with send and receive actions. More specifically, if there is at least one receive action that 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 receive actions. Instead, time passing can disable all send actions, but *only if* it preserves at least one guaranteed receive action.

Our semantics extends the one proposed in [10] to the general case of CTAs with *mixed states*, that is states which have both sending and receiving outgoing edges. Mixed states are a practically relevant and non-trivial feature of CTAs. Consider, for instance, the snippet of code in Figure 1 (left), which shows a typical usage of the `receive`/`after` construct in Erlang. The code snippet attempts to receive a message matching one of the patterns $\mathtt{\{s,a_1\}},\ldots,\mathtt{\{s,a_k\}}$, where `s` represents the identifier of the sender, and $\mathtt{a_1},\ldots,\mathtt{a_k}$ are the message labels. If no such message arrives within 10 milliseconds, then the process in the `after` branch is executed. This process sends immediately a message `b` to process `p`.

This behaviour can be modelled by the CTA in Figure 1 (right). Note that the state q_0 is mixed, because it has both receiving outgoing edges (labelled $\mathtt{sr}?a_i(x < 10)$),

[3] The formal framework in [29] allows to rule out these executions at a later stage, using final states and languages. See Section 6 for further discussion.

and one sending edge (labelled $\texttt{rp}!b(x = 10)$). When embedded in a system, this CTA will attempt to receive one of the messages a_i within 10 time units; if no message is received by that deadline, it will send message b to participant p at time 10.

This behaviour can not be modelled either by the semantics in [10], because it only deals with CTAs without mixed states, nor by timed multiparty session types of [11]. Actually, in [11] the interactions of each participant are described in terms of two constructs: *selection*, which corresponds to an internal choice of send actions, and *branching*, which corresponds to 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 [11].

Besides mixed states, there is also another practical aspect that is not well captured by the existing semantics of CTAs. We illustrate the issue through the Erlang-like process `receive a -> end`, which waits for a message a (potentially forever), and terminates after receiving it. Intuitively, this process could be modelled as the CTA $\mathsf{A}_\texttt{r}$ below on the left (the CTA $\mathsf{A}_\texttt{s}$ on the right is its communication partner):

A possible (partial) computation of the system $(\mathsf{A}_\texttt{r}, \mathsf{A}_\texttt{s})$ is:

$$((q_0, q'_0), (\varepsilon, \varepsilon), \{x \mapsto 0\}) \xrightarrow{\texttt{sr}!a} ((q_0, q'_1), (\varepsilon, a), \{x \mapsto 0\}) \xrightarrow{\text{one billion years}}$$

The semantics of CTAs in [10, 29] would allow the above computation, whereas receive primitives commonly used in mainstream programming languages have an *urgent* behaviour, i.e. they unblock as soon as the expected message is available. These include non-blocking input primitives (e.g., `WaitFreeReadQueue.read()` of Real-Time Java [15], `receive...after 0` of Erlang), blocking primitives (e.g., `WaitFree ReadQueue.waitForData()` of Real-Time Java and `receive...after infinity` of Erlang), and blocking primitives with deadline (e.g., `receive...after d` of Erlang).

To properly model the behaviour of these primitives, we also provide CTAs with an *urgent* semantics (Definition 6), that forces receive actions as soon as the expected message is available. The urgent semantics refines the non-urgent semantics, in the sense that the former allows a subset of the executions of the latter (Theorem 9). Interestingly enough, if a system of CTAs enjoys progress in the non-urgent semantics, then it will also enjoy progress in the urgent one (Theorem 10) under a minor and common assumption on the syntax of the time constraints (Definition 20).

Refinements. We propose three refinements of systems of CTAs. These refinements are *compositional*, in the sense that they result from the *point-wise* refinements of the single CTAs in the system. Compositionality is important as it enables modular development. It allows, for example, different organisations to contribute to a system by implementing only a part, independently. Our point-wise refinements (Definition 8) focus on the timing of each action: basically, the refinement of a CTA A is a CTA A' which has the same structure of A, but different constraints for the timing of send and receive actions. More specifically:

– In the first refinement, which we call *send/receive restriction*, the time constraints of both send and receive actions can be restricted[4]. This reflects practical scenarios in which the programmer sets narrower time windows than the specification.
– The second refinement, which we call *send restriction/receive procrastination* is the special case of the first one where the constraint of receive actions is required to respect the final deadline. Taken to the extreme, this refinement corresponds to implementing receive actions with blocking primitives, in the very last moment when the receive can be performed.
– In the *asymmetric restriction* refinement, the constraints for send actions can be restricted, while those for receive actions can be relaxed. This refinement seems the natural (timed) extension of the one used for subtyping on session types [23][5].

Preservation upon refinements. When refining a specification, one would like the concrete model to enjoy the same behavioural properties as the abstract one. For instance, in the realm of session types it is usually required that refining a type into a process preserves *progress* (i.e., execution never deadlocks), and enjoys *session fidelity* (i.e., the process does not add actions that are not planned by the type [9]). By adapting these properties to our timed and asynchronous setting, we obtain three properties: timed behaviour preservation (Definition 12), and preservation of *global* and *local* progress (Definitions 13 and 14, respectively). Behaviour preservation is defined in terms of timed similarity [18], and it ensures that the observable behaviour of the abstract system simulates that of the concrete one. Progress preservation requires that refinement will not introduce deadlocks, either globally (i.e., the whole system gets stuck), or locally (i.e., a single CTA gets stuck).

Our first result is negative: the considered refinements do *not* preserve behaviour nor progress, even in the simple case where mixed states are ruled out (Theorem 3). This witnesses the difficulty of implementing a system which, at the same time, enjoys progress and is coherent with the specification. The problem is particularly serious for *asymmetric* refinement. Indeed, while in the untimed setting the contravariant refinement of inputs (i.e. adding branches to external choices) corresponds to offering more options to the context, in the timed setting relaxing guards on receive actions corresponds to deliberately taking more time for performing actions. This 'selfish' behaviour, intuitively, causes the protocol being changed, instead of being refined, and the new protocol may no longer enjoy progress (as we will show in Example 14).

One of our main results is that the other refinements preserve behaviour and progress under two quite general conditions on sending and receiving edges (Definitions 16 and 17, respectively). Although these conditions are undecidable in the general case (Theorems 4 and 5), *send restriction/receive procrastination* enjoys the condition on receiving edges by construction, and the condition on sending edges can be approximated by a decidable condition on point-wise refinements (Theorem 8).

[4] This kind of refinement was the one used in [11] to relate session types and process calculi.

[5] In the line of work that follows [23], subtyping is seen as co-variant with respect to output behaviour and counter-variant w.r.t. input behaviour. Namely, a refining model is able to receive/handle a larger (or equal) set of inputs, and to send a smaller (or equal) set of outputs. This asymmetry still allows one to substitute a part of a system with its refinement while preserving the system's behaviour and properties.

1.2 Structure of the paper

In Section 2 we introduce our semantics of systems of CTAs, both in the non-urgent and in the urgent version. In Section 3 we define our CTA refinements. In Section 4 we define the properties of refinements we are interested in: behavioural preservation, and global/local progress preservation; then, in Section 5 we state the preservation results. Section 6 concludes and discusses related work. The proofs of our statements and additional examples are in the appendix.

2 Communicating Timed Automata

In Section 2.1 we recap the basic definitions related to CTAs. In Sections 2.2 and 2.3 we introduce two semantics for CTAs, in the non-urgent and urgent versions.

2.1 Background

We assume a finite set $\mathcal{P}$ of *participants*, ranged over by $\mathtt{p}$, $\mathtt{q}$, $\mathtt{r}$, $\mathtt{s}$, ..., and a finite set $\mathbb{A}$ of *messages*, ranged over by a, b, ... 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', ...), 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, ...), we define the set Δ_X of *guards* over X (ranged over by δ, δ', ...) 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:

- $\nu + t$ as the valuation mapping each $x \in X$ to $\nu(x) + t$.
- $\lambda(\nu)$ 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 ℓ, ℓ', ...). 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.

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 $\gamma_0 = (\boldsymbol{q_0}, \varepsilon, \nu_0)$ where $\boldsymbol{q_0} = (q_{0\mathtt{p}})_{\mathtt{p} \in \mathcal{P}}$, ε is the sequence of empty queues, and $\nu_0(x) = 0$ for each clock $x \in \bigcup_{\mathtt{p} \in \mathcal{P}} X_\mathtt{p}$.

2.2 A new semantics for systems of CTAs

We introduce a new semantics of systems of CTAs that generalizes Definition 9 in [10] 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 some clock valuation ν.

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

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

Example 1. In the initial clock valuation ν_0, we have that:

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

The relation $\leq_\nu$ is reflexive, transitive and total, but it 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 $c \neq c'$.

Lemma 1. *The relation $\leq_\nu$ is a total preorder, for all clock valuations ν.*

Proof. See page 28.

The following lemma states some basic properties of later satisfiability.

Lemma 2. *For all guards δ, δ', for all $t \in \mathbb{R}_{\geq 0}$, and $c, d \in \mathbb{Q}_{\geq 0}$:*

1. $(x \leq c) \leq_\nu (x \leq c + d) \iff d \geq 0$
2. $\delta \wedge \delta' \leq_\nu \delta'$
3. $\delta \leq_\nu \delta' \implies \delta \leq_{\nu + t} \delta'$

Proof. See page 28.

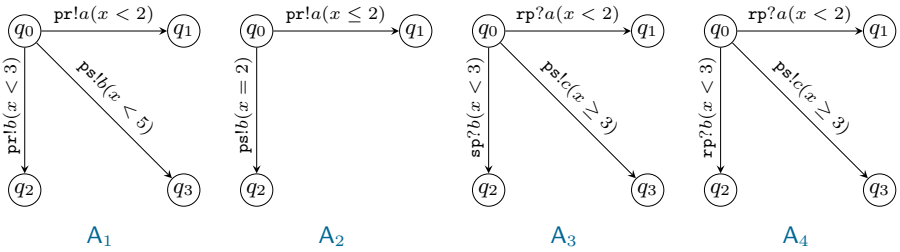


Fig. 2: CTAs for Example 2.

Definition 2 (Future-enabled, latest-enabled, and deferrable edge). *In a configuration* $(\boldsymbol{q}, \boldsymbol{w}, \nu)$*, we say that an edge* $(q, \ell, q') \in E_{\mathtt{p}}$ *is:*

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

An edge is future-enabled when its guards can be satisfied at some time in the future; it is latest-enabled when not other edge (starting from the same state) can be satisfied later than it. Note that the type of action (send or receive) and the co-party involved are immaterial to determine future-enabled and latest-enabled edges. A receiving edge is non-deferrable 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 non-deferrable, if both expected messages are in the head of each respective queue.

Example 2. Fix a configuration $(\boldsymbol{q}, \boldsymbol{w}, \nu)$ of some system that includes one of the CTAs in Figure 2. The edge $(q_0, \mathtt{pr}!b(x < 3), q_2)$ of A_1 is future-enabled iff $\nu(x) < 3$; the latest-enabled edge is $(q_0, \mathtt{ps}!b(x < 5), q_3)$ 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 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.

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

Definition 3 (Timed LTS). *A timed labelled transition system (in short, TLTS) is a triple* $(Q, \mathcal{L}, \to)$*, where* Q *is the set of* states*,* $\mathcal{L} \supseteq \mathbb{R}_{\geq 0}$ *is the set of* labels*, and* $\to \subseteq Q \times \mathcal{L} \times Q$ *is the* transition relation*. We use* $\alpha, \beta, \ldots$ *to range over* $\mathcal{L}$*.*

Definition 4 (Semantics of systems). *Given a system S, we define the TLTS $[\![S]\!] = (Q, \mathcal{L}, \to)$, 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 latest-enabled in* γ, *then such edge is latest-enabled also in* γ';
   (e) *for all* $\mathtt{p} \in \mathcal{P}$, *if some edge starting from* $q_{\mathtt{p}}$ *is non-deferrable in* γ, *then there exists an edge starting from* $q_{\mathtt{p}}$ *that is non-deferrable 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 the adapted from [10]. 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 (3a) and (3b) require that states and queues are not affected by the passing of time, which is implemented by item (3c). Items (3d) and (3e) put constraints on when time can pass. In particular, condition (3d) requires that time passing preserves latest-enabled 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 (3e) ensures that, if some expected message is already at the head of the queue, time passing cannot prevent that message to be received.

The following lemma states that our semantics enjoys two basic properties of timed systems, namely *time determinism* and *time additivity* [36].

Lemma 3. *For all configurations $\gamma, \gamma', \gamma''$, and for all $t, t' \in \mathbb{R}_{\geq 0}$, we have that:*

$$\gamma \xrightarrow{t} \gamma' \ \wedge\ \gamma \xrightarrow{t} \gamma'' \implies \gamma' = \gamma'' \qquad \textit{(Time determinism)}$$

$$\gamma \xrightarrow{t+t'} \gamma' \iff \exists \tilde{\gamma} : \gamma \xrightarrow{t} \tilde{\gamma} \ \wedge\ \tilde{\gamma} \xrightarrow{t'} \gamma' \qquad \textit{(Time additivity)}$$

Proof. See page 28.

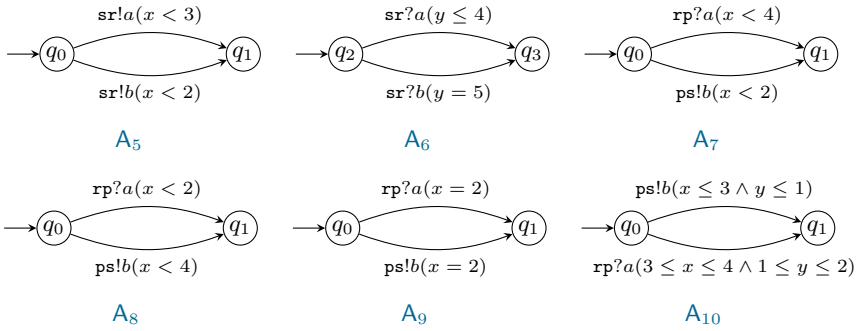


Fig. 3: A collection of CTAs, to illustrate the semantics of systems.

Note that our semantics does not enjoy *persistency* [36], because the passing of time can suppress the ability to perform some actions. For instance, in the system composed of the CTAs A_5 and A_6 in Figure 3, from the initial configuration we can delay for 2.5 time units, so disabling the action $\mathtt{sr}!b$ in A_5 (this example is elaborated further later in this section). However, our semantics enjoys a weaker persistency property, stated by Theorem 1. More specifically, if a receive action is non-deferrable, then time passing cannot suppress *all* receive actions: at least a non-deferrable action (not necessarily the first one) always remains future-enabled after a time delay. Instead, time passing can disable all send actions, but *only if* it preserves at least a non-deferrable receive action. All this is possible unless some *other* CTA $\mathtt{s}$ blocks time.

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

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

Proof. See page 28.

Definition 5 (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:*

$$conf_n(\rho) = \gamma_n \qquad delay_n(\rho) = t_n \qquad act_n(\rho) = \alpha_n$$

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 $\alpha \in \mathsf{Act}$.*

Examples. We show the peculiarities of our semantics through the CTAs in Figure 3.

First, consider the system composed of A_5 and A_6. A possible maximal run of $(\mathsf{A}_5, \mathsf{A}_6)$ from the initial configuration $\gamma_0 = ((q_0, q_2), \boldsymbol{\varepsilon}, \nu_0)$ is the following:

$$\begin{aligned}
\gamma_0 &\xrightarrow{2} \gamma_1 = ((q_0, q_2), (\varepsilon, \varepsilon), \nu_0 + 2) && \text{by (3)}\\
&\xrightarrow{\mathtt{sr}!a} \gamma_2 = ((q_1, q_2), (a, \varepsilon), \nu_0 + 2) && \text{by (1)}\\
&\xrightarrow{1.5} \gamma_3 = ((q_1, q_2), (a, \varepsilon), \nu_0 + 3.5) && \text{by (3)}\\
&\xrightarrow{\mathtt{rs}?a} \gamma_4 = ((q_1, q_3), (\varepsilon, \varepsilon), \nu_0 + 3.5) && \text{by (2)}
\end{aligned}$$

The first delay transition is possible because there are no non-deferrable edges in A_5 (as both edges are sending), and the latest enabled edge $(q_0, \mathtt{sr}!a(x < 3), q_1)$, continues to be such in $\nu_0 + 2$; further, in A_6 there are no latest-enabled sending edges, and no non-deferrable edges (since the queue $\mathtt{sr}$ is empty). Note that condition (3d) prevents γ_0 from making transitions with label $t \geq 3$, since $(q_0, \mathtt{sr}!a(x < 3), q_1)$ is latest-enabled in γ_0, but it is *not* latest enabled 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_5 is final, while the state q_2 of A_6 has a non-deferrable edge, $(q_2, \mathtt{sr}?a(y \leq 4), q_3)$, which is still non-deferrable at $\nu_0 + 3.5$. Note instead that condition (3e) prevents γ_2 from making a transition with $t > 2$, because no edge is non-deferrable 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 future-enabled is $y = 4$. Finally, the transition from γ_3 to γ_4 corresponds to a receive action.

In the following examples we only consider a single CTA at a time (the other CTAs in the system are immaterial to our illustration).

Receive/send mixed states. The CTA A_7 has mixed states, because the state q_0 is both sending and receiving. Also, the receive action is enabled for longer than the send action. In the initial configuration γ_0, the only latest-enabled edge is the receiving one, and since the queue $w_{\mathtt{rp}}$ is empty in γ_0, it is also deferrable. Therefore, relatively to A_7, condition (3d) is satisfied in γ_0 because the only sending edge is not latest-enabled, and condition (3e) is satisfied because there are no non-deferrable edges.

Send/receive mixed states. The CTA A_8 has also mixed states, but now it is the send action to be enabled for longer than the receive action. This CTA has a latest-enabled sending action in the initial configuration, i.e. $(q_0, \mathtt{ps}!b(x < 4), q_1)$. Hence, condition (3d) is satisfied in γ_0 if and only if the delay t is less than 4. Condition (3e) is satisfied in γ_0 because there are no non-deferrable edges. In the configuration where A_8 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 non-deferrable, so condition (3e) would be violated. Note that, 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 (3d) is respected. The behaviour of this CTA can be seen as a *timeout*: first it tries to receive a message, and after the deadline it does something else (in this case, sends a message).

Mixed states with equal constraints. In A_9, the deadlines on the two edges are equal: hence, both edges are latest-enabled in the initial configuration γ_0. Only condition (3d) (on the sending edge) is taken into account in γ_0, because the receiving edge is deferrable in γ_0. The condition requires the sending edge to remain latest-enabled, hence time can elapse only up to 2 time units. Instead, in the configuration where A_9 is in q_0, $\nu(x) = 0$, and the message a is the head of $w_{\mathtt{rp}}$, then the receive edge is non-deferrable. Also in this case, condition (3e) allows time to elapse only up to 2 time units.

Mixed states and multiple clocks. Finally, consider the CTA A_{10}, which uses two clocks, x and y. The guard on the receiving edge is not future-enabled from the initial configuration. Hence, it is not latest-enabled as well, and so the only latest-enabled edge is the sending one. To remain latest-enabled, its guard requires $y \leq 1$: hence, condition (3d) only allows time to pass up to 1 time unit. Consider now a configuration γ_1 where A_{10} is in state q_0, queues are empty, $\nu(x) = 2$, and $\nu(y) = 0$. From γ_1,

the guard on the receiving edge is future-enabled (i.e., after a delay 1), hence this edge becomes the latest-enabled one. Condition (3d) is satisfied for all delays (since there are no latest-enabled sending edges), and condition (3e) is satisfied as well (no edge is non-deferrable, since queues are empty).

2.3 An urgent variant of the semantics

The semantics in Definition 4 does not force the receive actions to happen, unless time passing prevents the CTA from receiving in the future (as shown in the third example in Section 1.1). This behaviour, shared also with the semantics in [10,29], contrasts with the actual behaviour of the `receive` primitive of Erlang, as well as the one of similar primitives in mainstream programming languages: there, the primitive returns *as soon as* a message is available.

We now introduce a variant of the semantics in Definition 4 which faithfully models this behaviour. Technically, we make receive transitions *urgent* [12, 36] by forbidding delays when a receiving edge is enabled and the corresponding message is at the head of the queue. With this semantics, receiving edges model *blocking* primitives, that wait for input in exactly the time window prescribed by the guard.

Below, we let $\mathsf{Act}^? \subseteq \mathsf{Act}$ be the set of input labels.

Definition 6 (Urgent semantics of systems). *Given a system S, we define the TLTS $[\![S]\!]_u = (Q, \mathcal{L}, \to_u)$, where Q is the set of configurations of S, $\mathcal{L} = \mathsf{Act} \cup \mathbb{R}_{\geq 0}$, and $\to_u$ is defined as follows:*

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

The non-urgent and the urgent semantics are actually very similar: they differ only in that in the urgent one we forbid time to pass when in the current configuration there is a CTA that has some outgoing reading edges with enabled guard, and the corresponding message waiting in the queue.

Example 3. Consider again the system $(\mathsf{A}_5, \mathsf{A}_6)$ with the CTAs in Figure 3. According to the non-urgent semantics, a possible run would be (recalling from Section 2.2):

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

Note that $\gamma_2 \xrightarrow{\mathtt{rs}?a}$. Hence, the second clause of Definition 6 states that $\gamma_2 \not\xrightarrow{t}$ for all $t > 0$. Then, a maximal run of $(\mathsf{A}_5, \mathsf{A}_6)$ under the urgent semantics would be:

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

We will relate the non-urgent and the urgent semantics in Section 5.5. Hereafter, where not specified otherwise, we will refer to the non-urgent semantics.

	Send/receive restriction $\mathsf{A}' \sqsubseteq_{sr} \mathsf{A}$	**Send restriction/receive procrastination** $\mathsf{A}' \sqsubseteq_{srp} \mathsf{A}$	**Asymmetric restriction** $\mathsf{A}' \sqsubseteq_{a} \mathsf{A}$
send	$[\![\delta']\!] \subseteq [\![\delta]\!]$	$[\![\delta']\!] \subseteq [\![\delta]\!]$	$[\![\delta']\!] \subseteq [\![\delta]\!]$
receive	$[\![\delta']\!] \subseteq [\![\delta]\!]$	$[\![\delta']\!] \subseteq [\![\delta]\!] \qquad \downarrow [\![\delta']\!] = \downarrow [\![\delta]\!]$	$[\![\delta']\!] \supseteq [\![\delta]\!]$

Table 1: Three point-wise refinements. The conditions apply to all edges $(q, \ell, q') \in E$, where $f(q, \ell, q') = \ell'$, $\mathsf{guard}(\ell) = \delta$ and $\mathsf{guard}(\ell') = \delta'$.

3 Refinements

In this section we introduce our refinements for systems of CTAs. Our refinement relations (i) are decidable[6], (ii) produce refinements which model implementations with concrete programming primitives, and (iii) reflect a modular engineering practice where parts of the system are implemented independently, without knowing how other parts are implemented. This last goal is attained by defining *system refinement* (Definition 9) as the independent, or *point-wise*, refinement of each single CTA (Definition 8).

3.1 Point-wise and system refinements

Our refinements operate on time constraints: more specifically, they only alter the guards, in the refined CTA, while leaving all the rest (actions and resets) unchanged. To relate two CTAs A and A', we use *structure preserving* functions that map the edges of A into those of A', preserving everything but the guards.

Definition 7 (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 $\mathsf{reset}(\ell) = \mathsf{reset}(\ell')$.*

We introduce three point-wise refinements in Definition 8. We have a *send/receive restriction* (written $\mathsf{A}' \sqsubseteq_{sr} \mathsf{A}$) when the guards of A' — both in send and receive actions — are narrower than those of A. A *send restriction/receive procrastination* ($\mathsf{A}' \sqsubseteq_{srp} \mathsf{A}$) additionally requires that the guards in receive actions have *the same past* in both CTAs. In an *asymmetric restriction* $\mathsf{A}' \sqsubseteq_{a} \mathsf{A}$, the guards of A' restrict those in A for send actions, while they relax those in A for receive actions.

Definition 8 (Point-wise refinement). *A point-wise refinement $\sqsubseteq$ is a preorder relation between CTAs, say $\mathsf{A} = (Q, q_0, X, E)$ and $\mathsf{A}' = (Q, q_0, X, E')$, for which there exists a structure-preserving isomorphism $f : E \to E'$. In Table 1 we define three specific point-wise refinements, by specifying conditions on the guards of A and A'.*

A point-wise refinement induces a refinement relation between systems of CTAs.

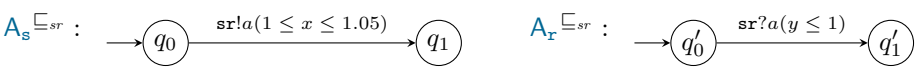
Definition 9 (System Refinement). *Let $\sqsubseteq$ be a point-wise refinement, and let $S = (\mathsf{A}_1, \ldots, \mathsf{A}_n)$ and $S' = (\mathsf{A}'_1, \ldots, \mathsf{A}'_n)$. We write $S \sqsubseteq S'$ iff $\mathsf{A}_i \sqsubseteq \mathsf{A}'_i$ for all $i \in 1 \ldots n$.*

[6] In general, establishing if an *asynchronous* communication model is a refinement of another is undecidable, even in the untimed scenario [14, 30].

In Example 4 we illustrate the restrictions given in Definition 9, starting from a common system to refine.

Example 4. Consider the following system composed of two CTAs:

Refinement by send/receive restriction allows to refine $\mathsf{A_s}$ and $\mathsf{A_r}$ with CTAs having narrower guards in both sending and receiving actions, like e.g.:



Refinement by send restriction/receive procrastination allows to restrict guards, as in the case of send/receive restriction. In this case, however, the guards of receiving actions will have to maintain the "latest" part of the time window, like e.g.:

Finally, asymmetric restriction allows to narrow the guards on send actions, and relax the guards of receive actions, like e.g.:

Theorem 2 below establishes decidability of all the point-wise refinements in Table 1. This follows by the fact that CTAs are finite state systems and that, using guards as defined in Section 2.1:

- the function $\downarrow [\![\delta]\!]$ is computable, and the result can be represented as a guard [8,27]
- the relation $\subseteq$ between guards is computable.

Theorem 2. *Establishing whether* $\mathsf{A}' \sqsubseteq_\circ \mathsf{A}$, *for* $\circ \in \{sr, srp, a\}$, *is decidable.*

4 Characterising essential properties of refinements

In this section we formalise properties of systems of CTAs that we wish to be preserved upon refinement. The first of these properties, which we call *behaviour preservation* (Definition 12) is based on the notion of timed similarity [18].

Definition 10 (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.*

Definition 11 (Disjoint union of TLTSs). *We define the disjoint union of TLTSs as:*

$$(Q_1, \Sigma_1, \rightarrow_1) \uplus (Q_2, \Sigma_2, \rightarrow_2) = (Q_1 \uplus Q_2, \Sigma_1 \cup \Sigma_2, \{((i,q), a, (i,q')) \mid (q,a,q') \in \rightarrow_i\})$$

where $Q_1 \uplus Q_2 = \{(i,q) \mid q \in Q_i\}$.

Behaviour preservation requires that an implementation (refining system) at any point of a run allows *only* actions that are allowed by its specification (refined system).

Definition 12 (Behaviour preservation). *Let $\mathcal{R}$ be a binary relation between systems. We say that $\mathcal{R}$* preserves behaviour *iff, whenever $S_1 \ \mathcal{R} \ S_2$, 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 5 (Behaviour preservation). Let $\mathcal{R}$ be the the inclusion of runs, and let S_2 be the system composed of the following CTAs:

The system S_2 has the following families of (maximal) traces:

$$t_1 \ \texttt{sr}!a \ t_2 \ \texttt{sr}?a \ t_3 \qquad \text{with } t_1 < 2, t_2 < 2 - t_1, \text{ and } t_3 \in \mathbb{R}_{\geq 0}$$
$$t_1 \ \texttt{sr}!b \ t_2 \ \texttt{sr}?b \ t_3 \qquad \text{with } t_1 > 2 \text{ and } t_2, t_3 \in \mathbb{R}_{\geq 0}$$

Let S_1 be as S_2 but for the guard of action $\texttt{sr}?b(true)$, which is replaced by $y > 7$. First, note that $S_2 \ \mathcal{R} \ S_1$, while $S_1 \ \mathcal{R} \ S_2$ does not hold: indeed, the traces with a are unchanged, while the traces with b in S_1 strictly include those of S_2:

$$t_1 \ \texttt{sr}!b \ t_2 \ \texttt{sr}?b \ t_3 \qquad \text{with } t_1 > 2, t_2 > 7 - t_1, \text{ and } t_3 \in \mathbb{R}_{\geq 0}$$

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. Consider now a system S_3, where the guard of $\texttt{sr}?b(true)$ is replaced by $y < 2$. The traces of S_3 containing a are the same as S_1, while those containing b are:

$$t_1 \ \texttt{sr}!b \ t_2 \qquad \text{with } t_1 > 2, t_2 \in \mathbb{R}_{\geq 0}$$

Hence, $S_3 \ \mathcal{R} \ S_1$, and as before, $\mathcal{R}$ preserves timed behaviour in $\{S_2, S_3\}$. However, S_3 does not allow the system to continue with the message exchange: b is sent too late to be received by $\texttt{r}$, who keeps waiting while the message remain in the queue forever.

As shown by Example 5, 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 produce executions that may get stuck. Another desirable property refinements is then the preservation of *progress*, which we conjugate below either as progress of the overall system (global progress, Definition 13), or as progress of each single participant (local progress, Definition 14).

Below, we say that a state q is *final* if there exist no ℓ and q' such that $(q, \ell, q') \in E$, and we say that a configuration $(\boldsymbol{q}, \boldsymbol{w}, \nu)$ is *final* when all $q \in \boldsymbol{q}$ are final.

Definition 13 (Global progress). *We say that a system S enjoys* global progress *when:*

$$\gamma_0 \to^* \gamma \text{ not final} \implies \exists t \in \mathbb{R}_{\geq 0}, \alpha \in \mathsf{Act} : \gamma \xrightarrow{t} \xrightarrow{\alpha}$$

Definition 14 (Local progress). *We say that a system S enjoys* local progress *when:*

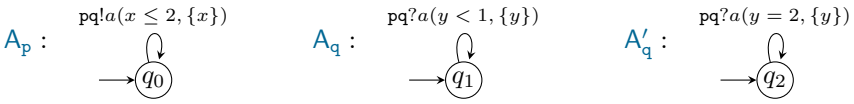
$$\gamma_0 \to^* \gamma = (\boldsymbol{q}, \boldsymbol{w}, \nu) \text{ and } q_{\mathsf{p}} \text{ not final} \implies \\ \forall \text{ maximal runs } \rho \text{ starting from } \gamma : \exists n : \mathsf{subj}(act_n(\rho)) = \mathsf{p}$$

The following lemma states that local progress is stronger then global progress. The converse does not hold, as witnessed by Example 6.

Lemma 4. *If a system enjoys local progress, then it also enjoys global progress.*

Proof. See page 29.

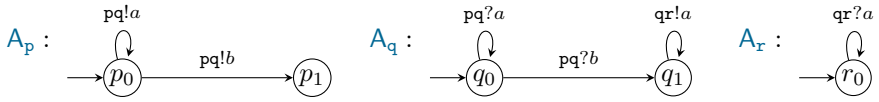
Example 6 (Global vs. local progress). Consider the following CTAs:



The system $(\mathsf{A_p}, \mathsf{A_q})$ enjoys global progress, since, in each reachable state, it is always possible for $\mathsf{A_p}$ to send a message (hence for the system to make an action in Act). However, if $\mathsf{A_p}$ sends a after time 1, then $\mathsf{A_q}$ can not 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 q, so $(\mathsf{A_p}, \mathsf{A_q})$ does *not* enjoy local progress. The system $(\mathsf{A_p}, \mathsf{A'_q})$, instead, enjoys both global and 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 7. 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.

Definition 15 (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 8. Let S_1, S_2, S_3 as in Example 5. 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).

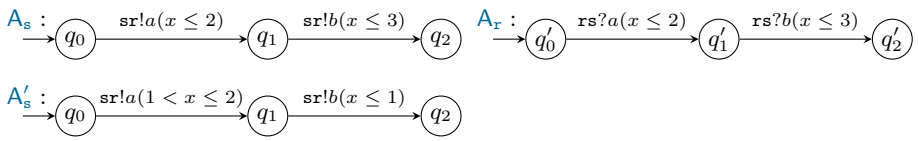
5 Preservation upon refinements

In this section we show results about preservation of behaviour/progress upon refinements. Theorem 3 states that, *in general*, these properties are not preserved.

Theorem 3 (Negative preservation results). *Send/receive restriction, send restriction/receive procrastination, and asymmetric restriction do not preserve behaviour nor (local/global) progress, even in the case of CTAs without mixed states.*

We prove Theorem 3 through a counter-example (Example 9) that applies to all given refinements, since we only restrict the guards of send actions.

Example 9. Let $S = (\mathsf{A_s}, \mathsf{A_r})$, and and let $S' = (\mathsf{A'_s}, \mathsf{A_r})$, where:



Note that $\mathsf{A'_s} \sqsubseteq \mathsf{A_s}$ for every point-wise refinement in Definition 8, and so $S' \sqsubseteq S$ for the corresponding system refinements. Neither behaviour nor progress are preserved. To see why, consider the following run of S':

$$\gamma_0 \xrightarrow{2} \gamma_1 \xrightarrow{\mathtt{sr}!a} \gamma_2 \xrightarrow{\mathtt{sr}?a} \gamma_3 \xrightarrow{3} \qquad (\gamma_i \text{ uniquely determined by the labels})$$

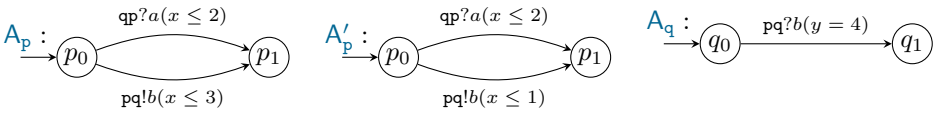
In particular, the last delay is possible since S' has no future-enabled actions in γ_3 (so, it is stuck). Instead, in S the last delay is *not* possible, since $\mathsf{A_s}$ has a future-enabled action in γ_3, but no future-enabled actions after a delay of 3 time units. Since S' has a trace not allowed by S, behaviour is not preserved. For similar reasons, S enjoys progress (both global and local), while S' gets stuck in $((q_1, q'_1), (\varepsilon, \varepsilon), \nu)$, with $1 < \nu(x) \leq 2$.

The rest of this section is organised as follows. Sections 5.1 and 5.2 introduce two conditions (called LESP and NDP, respectively), which are then used in Section 5.3 to establish our preservation results. Section 5.4 gives a decidable approximation of LESP, which allows to exploit our results in practice. In Section 5.5 we prove that behaviour and progress are preserved when passing from the non-urgent to the urgent semantics.

5.1 LESP: a property on sending edges

The problem observed in Example 9 could be approached by imposing some conditions on the *syntax* of CTAs, to exclude those which, like $\mathsf{A'_s}$ above, do not progress "by their own". Here we will focus instead on more general conditions on the *semantics* of CTAs, that also avoid other counter-examples like the one below.

Example 10. Let $S = (\mathsf{A_p}, \mathsf{A_q})$, and let $S' = (\mathsf{A'_p}, \mathsf{A_q})$, where:



We have that $\mathsf{A}'_{\mathtt{p}} \sqsubseteq \mathsf{A}_{\mathtt{p}}$ for every point-wise refinement in Definition 8, 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 $\gamma_0 \xrightarrow{2} \gamma = ((p_0, q_0), \varepsilon, \nu_0 + 2)$, but there are no t and $\alpha \in \mathsf{Act}$ such that $\gamma \xrightarrow{t} \xrightarrow{\alpha}$.

The problem in Examples 9 and 10 is that a latest-enabled sending edge, which was crucial for making execution progress, is lost after the refinement. As we will show, preserving latest-enabled sending edges is useful to preserve behaviour and progress. We formalise this condition in Definition 16.

Definition 16 (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 11. Recall S and S' from Example 10. 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 4). In Section 5.4 we give a sound, compositional, and practically useful, approximation of LESP.

Theorem 4 (Undecidability of LESP). *Establishing whether restrictions of the system refinements $\sqsubseteq_{sr}$, $\sqsubseteq_{a}$, $\sqsubseteq_{srp}$ are LESP is undecidable.*

Proof. See page 30.

5.2 NDP: a property on receiving edges

The following example shows that LESP alone is not enough to preserve behaviour or progress. In fact, further problems may arise when, after refinement, the ability of a CTA to receive messages is compromised.

Example 12. Recall $\mathsf{A}_{\mathtt{s}}$, $\mathsf{A}_{\mathtt{r}}$, $\mathsf{A}_{\mathtt{r}}{}^{\sqsubseteq_{sr}}$ from Example 4, and let $S = (\mathsf{A}_{\mathtt{s}}, \mathsf{A}_{\mathtt{r}})$, and $S' = (\mathsf{A}_{\mathtt{s}}, \mathsf{A}_{\mathtt{r}}{}^{\sqsubseteq_{sr}})$. We have $S' \sqsubseteq_{sr} S$. The relation $\mathcal{R} = \{(S', S)\}$ is LESP, 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.

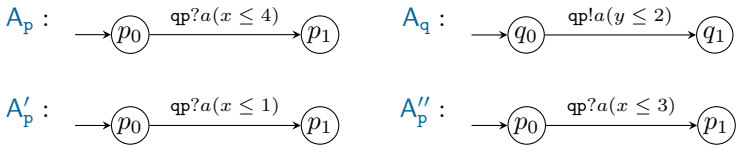
Intuitively, the problem is that narrowing the constraints of receiving edges may disable them before the message has been sent. To detect this situation, we introduce a property ensuring that a refinement preserves the non-deferrable actions (Definition 17).

Send/receive restriction	Send restriction/receive procrastination	Asymmetric restriction
✗ NDP-only (Example 10) ✗ LESP-only (Example 12) ✓ LESP+NDP (Theorem 6)	✗ NDP-only (Example 10) ✓ LESP (Theorem 7)	✗ LESP+NDP (Example 14)

Table 2: Summary of preservation results. ✓ means that preservation (of any kind) holds for a refinement under the given conditions, while ✗ means it does not hold.

Definition 17 (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 5). Remarkably, we will show later that $\sqsubseteq_{srp}$ guarantees NDP by construction.

Theorem 5 (NDP undecidability). *Establishing whether restrictions of the system refinements $\sqsubseteq_{sr}$, $\sqsubseteq_{a}$, $\sqsubseteq_{srp}$ are NDP is undecidable.*

Proof. See page 30.

5.3 Preservation results

In this section we present our main results about preservation upon refinements. Our results, both positive and negative, are also summarized in Table 2.

We already know from Example 10 that neither behaviour nor progress are preserved in general. Since the refinement considered in that example is NDP but not LESP, we also infer that preservation does not hold for *NDP-only* restrictions. Further, from Example 12 we know that even LESP-only restrictions of $\sqsubseteq_{sr}$ do not preserve behaviour and progress One of our main positive results is that LESP+NDP restrictions of $\sqsubseteq_{sr}$ (i.e., restrictions satisfying both conditions) do preserve both behaviour and progress.

Theorem 6. *LESP+NDP restrictions of $\sqsubseteq_{sr}$ preserve behaviour, global and local progress.*

Proof. See page 34.

Since, by definition, the refinement $\sqsubseteq_{srp}$ is stricter than $\sqsubseteq_{sr}$, the positive result about $\sqsubseteq_{sr}$ in Theorem 6 also applies to $\sqsubseteq_{srp}$. In Lemma 5 we will also show that $\sqsubseteq_{srp}$ is a NDP restriction of $\sqsubseteq_{sr}$. Therefore, for $\sqsubseteq_{srp}$ we can weaken the hypotheses about LESP+NDP used in the case of $\sqsubseteq_{sr}$, by only requiring LESP (Theorem 7). This has useful practical consequences, since in the following section we will devise a decidable approximation of LESP (recall from Theorem 4 that LESP is undecidable).

Lemma 5. *$\sqsubseteq_{srp}$ is a NDP restriction of $\sqsubseteq_{sr}$.*

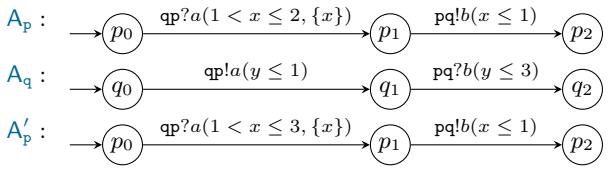
Proof. See page 34.

Theorem 7. *LESP restrictions of $\sqsubseteq_{srp}$ preserve behaviour, global and local progress.*

Proof. See page 34.

Quite surprisingly, for asymmetric restriction refinement we only have negative results about preservation. Indeed, Example 14 shows that $\sqsubseteq_a$ does not preserve behaviour nor progress, not even if considering LESP+NDP restrictions, not even if only considering CTAs without mixed states.

Example 14 (Asymmetric restriction refinement). Consider the following CTAs:



Let $S = (\mathsf{A_p}, \mathsf{A_q})$ and $S' = (\mathsf{A'_p}, \mathsf{A_q})$. We have that $\mathsf{A'_p} \sqsubseteq_a \mathsf{A_p}$, and so $S' \sqsubseteq_a S$. The relation $\mathcal{R} = \{(S', S)\}$ is LESP+NDP. 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.

5.4 A sound approximation of LESP

In this section we devise a decidable condition on point-wise refinements that implies LESP of the corresponding system refinements. We stress that this condition is on point-wise refinements, hence it is suitable for compositional reasoning. This condition, which we call LLESP after *locally* latest-enabled send preservation, is in fact an approximation of LESP (Theorem 8). We start with some auxiliary definitions.

Definition 18. *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:*

$$\begin{aligned} Pre_q^{\mathsf{A}} &\stackrel{def}{=} \{\nu_0 \mid q_0 = q\} \cup \{\nu \mid \exists q', \ell, \nu' : (q', \ell, q) \in E \ \wedge\ \nu' \in [\![\mathsf{guard}(\ell)]\!] \wedge \\ &\qquad\qquad \lambda = \mathsf{reset}(\ell) \ \wedge\ \nu = \lambda(\nu')\} \\ Les_q^{\mathsf{A}} &\stackrel{def}{=} \{\nu \mid q \text{ has a latest-enabled sending edge in } \nu\} \\ Post_q^{\mathsf{A}}(K) &\stackrel{def}{=} \left\{\nu + t \,\middle|\, \nu \in K \ \wedge\ (\nu \in Les_q^{\mathsf{A}} \implies \nu + t \in Les_q^{\mathsf{A}})\right\} \end{aligned}$$

We briefly comment the definition above. The set Les_q^{A} is self-explanatory, and its use is auxiliary to the definition of $Post$. The sets Pre_q^{A} and $Post_q^{\mathsf{A}}(K)$ are useful for obtaining an over-approximation, denoted $Post_q^{\mathsf{A}}(Pre_q^{\mathsf{A}})$, 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. This is needed because the LESP property on $S' \ \mathcal{R}\ S$ (Definition 16) holds if latest-enabled sending edges are preserved in reachable configurations of S'. The set Pre_q^{A} 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. This is an over-approximation, as it only considers the edges entering in q and not constraints of earlier parts of the run. Instead, $Post_q^{\mathsf{A}}(K)$ 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 $\nu' \in Post_q^{\mathsf{A}}(K)$. This is obtained by defining $Post_q^{\mathsf{A}}(K)$ as the set of clock valuations that would satisfy item 3d of Definition 4 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 an run ending with a delay (the original run followed by a null delay), the set $Post_q^{\mathsf{A}}(Pre_q^{\mathsf{A}})$ contains the clock valuations we were looking for.

We can now introduce a decidable and compositional approximation of LESP.

Definition 19 (Locally latest-enabled send preservation). *We say that a point-wise refinement* $\sqsubseteq$ *is* locally latest-enabled 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* $\mathsf{A} \sqsubseteq \mathsf{A}'$*, and for all* $q \in Q$*:*

$$Post_q^{\mathsf{A}}(Pre_q^{\mathsf{A}}) \cap Les_q^{\mathsf{A}'} \subseteq Post_q^{\mathsf{A}}(Pre_q^{\mathsf{A}}) \cap Les_q^{\mathsf{A}}$$

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

Theorem 8 (From LLESP to LESP). *System refinements induced by LLESP point-wise refinements are LESP. Furthermore, LLESP is decidable.*

Proof. See page 36.

5.5 Refinements under the urgent semantics

We now relate the urgent and non-urgent semantics. Since the urgent semantics only restricts the behaviour of systems (by dropping some timed transitions), it comes without surprise that urgent semantics preserves the behaviour of the non-urgent one.

Theorem 9 (Preservation of behaviour). *For all systems S, the relation*

$$\{((\gamma, 1), (\gamma, 2)) \mid \gamma \text{ is a configuration of } S\}$$

between states of $[\![S]\!]_u \uplus [\![S]\!]$ is a timed simulation.

Proof. See page 37.

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 15.

Example 15 (Progress - non-urgent vs. urgent - part 1). Consider the following system:

A run of $(\mathsf{A_s}, \mathsf{A_r})$ with the non-urgent semantics is:

$$\gamma_0 \xrightarrow{\mathtt{sr}!a} \xrightarrow{3} \gamma = ((q_1, q'_0), (a, \varepsilon), \nu_0 + 3) \xrightarrow{t} \xrightarrow{\mathtt{sr}?a} \qquad \forall t \in \mathbb{R}_{\geq 0}$$

The corresponding run with the urgent semantics would be:

$$\gamma_0 \xrightarrow{\mathtt{sr}!a}_u \xrightarrow{3}_u \gamma \not\xrightarrow{\alpha}_u \qquad \forall \alpha \neq 0$$

Note that the run with the non-urgent semantics leads to a final state, whereas the one with the urgent semantics does not.

The issue highlighted by Example 15 is subtle: if there is not a precise point in time in which a guard becomes enabled (e.g. in $x > 3$), then the run may get stuck. This problem is known in literature [12], and it can be dealt with a minor syntactic restriction on the guards of CTAs. In Definition 20 we introduce a restriction on guards that guarantees that urgent semantics preserves progress. The condition, which generalises the notion of *right-open time progress* of [12] (to deal with non-convex guards), corresponds to forbidding guards defined as the conjunction of subguards of the form $x > c$ (but still allowing subguards of the form $x \geq c$). We provide a definition based on sets of clock valuations, in order to keep our results independent from the syntax of guards.

Definition 20 (Fully left closed). *For all ν, and for all sets of clock valuations K, let:*

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

and let $\inf Z$ *denote the infimum of a set Z. We say that 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 16 (Progress - non-urgent vs. urgent - part 2). The guard $x > 3$ in Example 15 is not fully left closed. Indeed, $\inf D_{\nu_0}([\![x > 3]\!]) = \inf\{t \mid t > 3\} = 3$, 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 15 where guard $x > 3$ is replaced by $x \geq 3$. The run

$$\gamma_0 \xrightarrow{\mathtt{sr}!a}_u \xrightarrow{3}_u \gamma$$

would not get stuck and allow $\gamma \xrightarrow{\mathtt{sr}?a}$.

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

Theorem 10 (Preservation of progress vs. urgency). *For all input fully left closed systems S, if S enjoys global (resp. local) progress under non-urgent semantics, then S enjoys global (resp. local) progress under urgent semantics.*

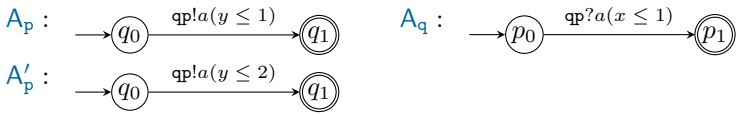
Proof. See page 37.

6 Conclusions and related work

We have introduced three decidable notions of compositional timed refinement for distributed systems with asynchronous communications, and studied their ability of preserving three crucial properties: behaviour, global progress and local progress. Two of these refinements, $\sqsubseteq_{sr}$ and $\sqsubseteq_{srp}$, restrict the model's behaviour. One of them, $\sqsubseteq_a$, reflects the asymmetric nature of (untimed) behavioural subtyping, in the sense that it restricts outputs and relaxes inputs. All refinements, except $\sqsubseteq_a$, turned out to preserve behaviour, global progress and local progress of the original system, under a few additional conditions. The negative results for $\sqsubseteq_a$ suggest that the asymmetric approach does not scale to the timed scenario.

The purpose of our work was to provide formal basis to support implementation of well-behaved systems from well-behaved models. However, as we have shown in Section 1, the existing semantics of CTAs have limitations in this sense. We have therefore introduced two new semantics of CTAs: (i) a more general non-urgent semantics that is closer to known semantics, and (ii) an urgent semantics that models the behaviour of concrete implementations. The more general semantics allows us to analyse the properties of systems upon incremental refinements (e.g. thanks to the results in Section 5). Moreover, the preservation results from non-urgent to urgent semantics given in Section 5.5 pave the way to implementations of refinements (e.g. derived incrementally using the non-urgent semantics, and relying on the results in Section 5) still preserve behaviour and progress.

Semantics. The semantics of CTAs in [29] was introduced for studying decidability issues for timed languages: their decidability results are general, and apply also to our semantics. To achieve such goals, [29] adopts the usual language-based approach used in computability theory. However, 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 final:



The systems $S = (\mathsf{A_p}, \mathsf{A_q})$ and $S' = (\mathsf{A'_p}, \mathsf{A_q})$ accept the same language, namely $t_0 \ \mathtt{qp}!a \ t_1 \ \mathtt{qp}?a \ t_2$ with $t_0 + t_1 \leq 1$ and $t_2 \in \mathbb{R}_{\geq 0}$. However, the two systems have a fundamental difference: S enjoys progress, while S' does not. In general, as discussed in Section 1, the semantics in [29] allows time steps that make the guards of all the available actions unsatisfiable. Intuitively, one could rule out undesirable executions *a posteriori* by considering only those that terminate in final states, or visit them infinitely often. However, it is much more standard to introduce some way for disciplining the flow of time, giving the power of enforcing the execution of discrete actions. This is supported by automata-based formalisms through, e.g., invariants [8, 27] and deadlines [12], and by a variety of primitives in timed process algebras [36]. Summing up, the benefits of our semantics are: (i) they allow us to rule out undesirable executions without filtering them *a posteriori*; (ii) they facilitate the comparison of behaviours, based, e.g., on (bi)simulation; (iii) they allow us to reason on standard properties of protocols (e.g., progress) without the need of introducing *ad hoc* final states.

Urgency. Our urgent semantics forbids delays when read actions are possible, hence emulating the receive primitives of common programming languages. Some mechanisms for expressing urgent behaviour are present in timed process algebras [36]. In particular, our notion of urgency, being enforced by communications, can be seen as an extension to the asynchronous setting of the maximal progress assumption in Timed CCS [41]. This condition, also known as τ-urgency [36], disables time passing when a synchronisation is possible. In the context of timed automata, some extensions to deal with urgency have been introduced: urgent locations and channels in Uppaal [7, 8], and the already mentioned timed automata with deadlines of [12]. In [29], the term "urgent" has been used to refer to their semantics. However, their notion of urgency is different from ours: in [29], it means that read actions have priority over invisible actions. We use the word urgency with the traditional meaning of [12, 36], instead.

Refinements. The notions of refinement introduced in this paper are related to the concept of subtyping in session types [23, 26]. Asymmetric refinement is a naïve attempt to extend the output co-variant and input contra-variant approach to subtyping of [23] to time constraints. Differently from these works, we focus on "purely timed" refinements (structure preserving mappings that only affect guards). Other standard techniques to deal with the branching structure of CTAs are orthogonal to the present work. In [11] a (session) type system for a timed π-calculus was introduced. There, systems are modelled as timed session types (which can be mapped into CTAs [10]). Systems

refinements are presented as processes in a calculus, for which (behavioural) typing rules ensure properties similar to our behaviour preservation and global progress. The input/output primitives in the calculus of [11] can roughly be modelled with CTAs that contains only equality in guards. Therefore, our send/receive restriction refinement could be seen, up to minor adaptation, a generalization of the type system in [11]. Other notions of timed refinements in the timed setting have been studied in [5, 20, 22]. All of them consider synchronous communication.

Properties and preservation. Timed simulation we used [18] is nowadays classical. The notion of progress is well known both in the untimed [6] and timed setting [5, 10, 11, 37], called compliance in the last. Preservation of properties upon refinement has traditionally an important role in session subtyping. Indeed, subtyping of binary synchronous session types admits an elegant characterization as the largest progress preserving relation [4]. There are some other interesting properties of systems, such as non-zenoness and eventual reception of messages in a queue (studied in [10] for CTA) which we have not studied. These properties seem not preserved by our refinements in the general case. A characterization of the subclass of CTAs for which they are preserved is left as a future work.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. TCS 126, 183–235 (1994)
2. Advanced Message Queuing protocols (AMQP) homepage. http://jira.amqp.org/confluence/display/AMQP/Advanced+Message+Queuing+Protocol
3. Armstrong, J.: Programming Erlang: Software for a Concurrent World. Pragmatic Bookshelf (2007)
4. Barbanera, F., de'Liguoro, U.: Sub-behaviour relations for session-based client/server systems. MSCS 25(6), 1339–1381 (2015)
5. Bartoletti, M., Cimoli, T., Murgia, M., Podda, A.S., Pompianu, L.: Compliance and subtyping in timed session types. In: FORTE. LNCS, vol. 9039, pp. 161–177. Springer (2015)
6. Bartoletti, M., Scalas, A., Zunino, R.: A semantic deconstruction of session types. In: Proc. CONCUR. LNCS, vol. 8704, pp. 402–418. Springer (2014)
7. Behrmann, G., David, A., Larsen, K.G.: A tutorial on UPPAAL. In: SFM. LNCS, vol. 3185, pp. 200–236. Springer (2004)
8. Bengtsson, J., Yi, W.: Timed automata: Semantics, algorithms and tools. In: Lectures on Concurrency and Petri Nets. LNCS, vol. 3098, pp. 87–124. Springer (2003)
9. Bettini, L., Coppo, M., D'Antoni, L., Luca, M.D., Dezani-Ciancaglini, M., Yoshida, N.: Global progress in dynamically interleaved multiparty sessions. In: CONCUR. LNCS, vol. 5201, pp. 418–433 (2008)
10. Bocchi, L., Lange, J., Yoshida, N.: Meeting deadlines together. In: 26th International Conference on Concurrency Theory, CONCUR 2015. LIPIcs, vol. 42, pp. 283–296. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2015)
11. Bocchi, L., Yang, W., Yoshida, N.: Timed Multiparty Session Types, LNCS, vol. 8704, pp. 419–434. Springer (2014)
12. Bornot, S., Sifakis, J., Tripakis, S.: Modeling urgency in timed systems. In: COMPOS. LNCS, vol. 1536, pp. 103–129. Springer (1997)
13. Brand, D., Zafiropulo, P.: On communicating finite-state machines. J. ACM 30(2), 323–342 (1983)
14. Bravetti, M., Carbone, M., Zavattaro, G.: Undecidability of asynchronous session subtyping. Information and Computation (2017)
15. Bruno, E.J., Bollella, G.: Real-Time Java Programming: With Java RTS. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edn. (2009)
16. Capecchi, S., Giachino, E., Yoshida, N.: Global Escape in Multiparty Sessions. In: FSTTCS. Leibniz International Proceedings in Informatics (LIPIcs), vol. 8, pp. 338–351 (2010)
17. Cattani, S., Kwiatkowska, M.Z.: A refinement-based process algebra for timed automata. Formal Asp. Comput. 17(2), 138–159 (2005)
18. Cerans, K.: Decidability of bisimulation equivalences for parallel timer processes. In: CAV. LNCS, vol. 663, pp. 302–315. Springer (1992)
19. Chen, T.c., Dezani-Ciancaglini, M., Scalas, A., Yoshida, N.: On the Preciseness of Subtyping in Session Types. LMCS 13(2) (2017)
20. Chilton, C., Kwiatkowska, M.Z., Wang, X.: Revisiting timed specification theories: A linear-time perspective. In: FORMATS. LNCS, vol. 7595, pp. 75–90. Springer (2012)
21. Clemente, L., Herbreteau, F., Stainer, A., Sutre, G.: Reachability of communicating timed processes. In: FOSSACS, LNCS, vol. 7794, pp. 81–96. Springer (2013)
22. David, A., Larsen, K.G., Legay, A., Nyman, U., Traonouez, L., Wasowski, A.: Real-time specifications. STTT 17(1), 17–45 (2015)
23. Demangeon, R., Honda, K.: Full Abstraction in a Subtyped pi-Calculus with Linear Types, LNCS, vol. 6901, pp. 280–296. Springer (2011)
24. Dierks, H., Kupferschmid, S., Larsen, K.G.: Automatic Abstraction Refinement for Timed Automata, LNCS, vol. 4763, pp. 114–129. Springer (2007)

25. Fecher, H., Majster-Cederbaum, M.E., Wu, J.: Refinement of actions in a real-time process algebra with a true concurrency model. ENTCS 70(3), 260–280 (2002)
26. Gay, S., Hole, M.: Subtyping for Session Types in the Pi-Calculus. Acta Informatica 42(2/3), 191–225 (2005)
27. Henzinger, T.A., Nicollin, X., Sifakis, J., Yovine, S.: Symbolic model checking for real-time systems. Inf. Comput. 111(2), 193–244 (1994)
28. Hu, R., Neykova, R., Yoshida, N., Demangeon, R., Honda, K.: Practical interruptible conversations - distributed dynamic verification with session types and Python. In: RV. LNCS, vol. 8174, pp. 130–148 (2013)
29. Krcal, P., Yi, W.: Communicating timed automata: The more synchronous, the more difficult to verify. In: CAV. LNCS, vol. 4144, pp. 243–257 (2006)
30. Lange, J., Yoshida, N.: On the Undecidability of Asynchronous Session Subtyping, pp. 441–457. Springer (2017)
31. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a nutshell. Int. Journal on Software Tools for Technology Transfer 1, 134–152 (1997)
32. Lynch, N., Segala, R., Vaandrager, F.: Hybrid i/o automata. Inf. Comput. 185(1), 105–157 (Aug 2003)
33. Mostrous, D.: Session Types in Concurrent Calculi: Higher-Order Processes and Objects. Ph.D. thesis, Imperial College London (November 2009)
34. Mostrous, D., Yoshida, N.: Session-based communication optimisation for higher-order mobile processes. In: TLCA. LNCS, vol. 5608, pp. 203–218 (2009)
35. Mostrous, D., Yoshida, N., Honda, K.: Global principal typing in partially commutative asynchronous sessions. In: ESOP. LNCS, vol. 5502, pp. 316–332 (2009)
36. Nicollin, X., Sifakis, J.: An overview and synthesis on timed process algebras. In: CAV. LNCS, vol. 575, pp. 376–398 (1991)
37. Padovani, L.: Fair subtyping for multi-party session types. MSCS 26(3), 424–464 (2016)
38. Schneider, S.: Concurrent and Real Time Systems: The CSP Approach. John Wiley & Sons, Inc., New York, NY, USA, 1st edn. (1999)
39. Vinoski, S.: Advanced Message Queuing Protocol. IEEE Internet Computing 10(6), 87–89 (2006)
40. Wang, W., Jiao, L.: Trace Abstraction Refinement for Timed Automata, LNCS, vol. 8837, pp. 396–410. Springer (2014)
41. Yi, W.: CCS + time = an interleaving model for real time systems. In: ICALP. LNCS, vol. 510, pp. 217–228. Springer (1991)
42. Yovine, S.: Kronos: A verification tool for real-time systems. (kronos user's manual release 2.2). International Journal on Software Tools for Technology Transfer 1, 123–133 (1997)

A Proofs for Section 2

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 $\delta' \leq_\nu \delta$. □

Proof of Lemma 2

For Item 1, 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 2, 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 3, 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, $\delta \leq_{\nu+t} \delta'$. □

Lemma 6. *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 3

Time determinism follows immediately by items 3a to 3c of definition 4. Time additivity follows by lemma 6. □

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 (3e) of Definition 4 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 $\mathtt{act}(e) = \mathtt{sp}?$. 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 (3d) of Definition 4 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 (3e) of Definition 4 holds as well, because the edge e itself remains

non-deferrable in $\nu' + t''$. Therefore, by condition (2) of Definition 4, 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 $\mathtt{act}(e) = \mathtt{ps!}$. We show that the transition $\gamma' \xrightarrow{t''} \gamma''$ is admitted by our semantics. Condition (3d) of Definition 4 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 (3e) holds trivially, because in this case we are assuming all receiving edges to be deferrable. Further, by condition (1) of Definition 4, $\gamma'' \xrightarrow{\mathtt{ps!}}$. Therefore we conclude that $\gamma' \xrightarrow{t''} \xrightarrow{\mathtt{ps!}}$.
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 $\mathtt{act}(e_s) = \mathtt{ps!}$. We show that the transition $\gamma' \xrightarrow{t''} \gamma''$ is admitted by our semantics. Condition (3d) of Definition 4 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 (3e) 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 4, we obtain the thesis $\gamma' \xrightarrow{t''} \xrightarrow{\mathtt{ps!}}$.
   (b) otherwise, if $t_r < t_s$, we show that the latest receiving action is preserved. Let $t'' = t_r$, and let $\mathtt{act}(e_r) = \mathtt{sp?}$. We show that the transition $\gamma' \xrightarrow{t''} \gamma''$ is admitted by our semantics. Condition (3d) of Definition 4 holds: indeed, by the choice of e_s and of t'' it follows that e_s is latest-enabled in γ''. Condition (3e) 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 4, we obtain the thesis $\gamma' \xrightarrow{t''} \xrightarrow{\mathtt{sp?}}$. □

B Proofs for Section 4

Proof of Lemma 4

We prove the contrapositive. Assume that S has does *not* enjoy global progress, i.e. by Definition 13 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 5, $\rho = \gamma$ is a maximal run of γ. Since γ is not final, there exists a participant $\mathtt{p}$ such that $q_{\mathtt{p}}$ is not final. Clearly, since the run γ has no transitions, there is no n such that $\mathsf{subj}(act_n(\rho)) = \mathtt{p}$. Therefore, by Definition 14, we conclude that S does not enjoy local progress. □

C Proofs for Section 5

C.1 Proofs for Section 5.1

Proof of Theorem 4 (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 $\sqsubseteq \in \{\sqsubseteq_{sr}, \sqsubseteq_a, \sqsubseteq_{srp}\}$ 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. □

C.2 Proofs for Section 5.2

Proof of Theorem 5 (Sketch)

The argument is similar to the one in the proof of theorem 4 above. We just show how to construct the two systems S' and S'' from the system S (containing A) and state q of A. Let A' be as A, except that the only edge with source q is the self-loop $(q, \mathtt{pr}!a(\mathit{true}), q)$, where $\mathtt{r}$ is not a participant in S. Let $\mathsf{A}'_{\mathtt{r}}$ be a CTA with just a state q_0 and a self loop $(q_0, \mathtt{pr}?a(\mathit{true}), q_0)$. Let S' be as S, except for A' in place of A and $\mathsf{A}'_{\mathtt{r}}$ added. Let $\mathsf{A}''_{\mathtt{r}}$ be a CTA with just a state q_0 and a self loop $(q_0, \mathtt{pr}?a(\mathit{false}), q_0)$. The relation $\{(S', S'')\}$ is not NDP iff q is reachable in S, by construction. Hence NDP checking is undecidible. □

C.3 General properties

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

Definition 21. *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:*

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

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

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

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

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

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 (3d) and (3e) of Definition 4 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 (3e), (3d), 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 7. *Let $\gamma = (\boldsymbol{q}, \boldsymbol{w}, \nu)$ be such that $\gamma_0 \to^* \gamma$. 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 (3e) and (3d) of Definition 4. □

Lemma 8. *Let S be a system of machines, and let γ be any configuration such that $\gamma_0 \to^* \gamma$. 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 $e = (q_\mathsf{p}, \ell, q'_\mathsf{p}) \in BSE_\gamma(t)$. 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 9. *Let S be a system of machines, and let $\gamma = (\boldsymbol{q}, \boldsymbol{w}, \nu)$ be a configuration such that $\gamma_0 \to^* \gamma$. 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 7.
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 $e = (q_\mathsf{p}, \ell, q'_\mathsf{p})$ 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 8, 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 10. *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, $\rho = \gamma_0 \xrightarrow{t_1}_{S_1} \gamma'_0 \xrightarrow{\alpha_1}_{S_1} \gamma_1 \dots \gamma$. According to definition 14, 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 $\mathsf{subj}(act_n(\rho')) = \mathtt{p}$. □

C.4 Proofs for Section 5.3

Lemma 11. *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 4 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 8:

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

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 3a to 3c clearly hold for S_2 as well. It remains to show it is the case also for items 3d and 3e.

For item 3d, 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 8, $[\![\mathsf{guard}(\ell'_{S_1})]\!] \subseteq [\![\mathsf{guard}(\ell'_{S_2})]\!]$, ℓ'_{S_2} is future-enabled in $\nu + t$, and hence item 3d holds for S_2.

For item 3e, 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 8, 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. □

Lemma 12. *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 11. □

Lemma 13. *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 item 3d and item 3e of definition 4 holds trivially for any delay by assumption, $\gamma \xrightarrow{t} (\boldsymbol{q}, \boldsymbol{w}, \nu + t)$, which is stuck. Hence S does not progress: contradiction. □

Lemma 14. *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 12, 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 7, $BRE^{S_2}_{\gamma}(t) = \emptyset$, and, by lemma 8, $BRE^{S_2}_{\gamma}(0) = \emptyset$ 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 $q_{\mathtt{p}}$ of S_2 that has a latest-enabled sending or a non-deferrable edge in γ. Then, by the LESP & NDP assumption, $q_{\mathtt{p}}$ 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 $e_{S_1} = (q_{\mathtt{p}}, \ell_{S_1}, q'_{\mathtt{p}}$

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 9, $\gamma \xrightarrow{t'}_{S_1} \xrightarrow{\alpha'}_{S_1}$ for some t' and $\alpha' \notin \mathbb{R}_{\geq 0}$. □

Lemma 15. *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 $\sqsubseteq_{srp}$. 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 11. 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 14 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. □

Lemma 16. *LESP+NDP restrictions of $\sqsubseteq_{sr}$ preserve local progress.*

Proof. Suppose S_2 has local progress. By lemma 4, S_2 has global progress as well. Then, by lemma 15, maximal runs of S_1 are maximal runs of S_2. Therefore, by lemma 10, S_1 has local progress. □

Proof of Theorem 6

Composition of lemma 12, lemma 14 and lemma 16. □

Proof of Lemma 5

The fact that $\sqsubseteq_{srp}$ is a restriction of $\sqsubseteq_{sr}$ follows by an easy inspection of definition 8. It remains to show $\sqsubseteq_{srp}$ is NDP. Let A_1 and A_2 be systems of machines such that $A_1 \sqsubseteq_{srp} A_2$, 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 8, $\downarrow [\![\mathsf{guard}(\ell)]\!] \subseteq \downarrow [\![\mathsf{guard}(\ell)]\!]$, ℓ' is future-enabled in γ. □

Proof of Theorem 7

Composition of theorem 6 and lemma 5. □

C.5 Local LESP

Lemma 17. *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* 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 $\gamma_0 \to^* \gamma$. 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 $\gamma_0 \to^n \gamma' \to \gamma$ for some $\gamma' = (\boldsymbol{q'}, \boldsymbol{w'}, \nu')$. We proceed by cases on the rule of definition 4 used for deriving $\gamma' \to \gamma$.

- 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 [8], that can be lifted to guards. They are instrumental in the proof of the decidability of LLESP.

Definition 22. *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 receive edge.

Definition 23. *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, simbol $\setminus$ denotes set difference.

Lemma 18. *For all A, for all q state of A, and for all K, we have that:*

1. $Pre^{\mathsf{A}}_q = \begin{cases} [\![(\bigvee_{\delta \in \mathsf{RGuards}(q)} \delta) \vee \delta_0]\!] & \text{if } q = q_0 \\ [\![\bigvee_{\delta \in \mathsf{RGuards}(q)} \delta]\!] & \text{otherwise} \end{cases}$
2. $Les^{\mathsf{A}}_q = \downarrow (q^! \setminus \downarrow (q^? \setminus \downarrow q^!))$.

3. $Post^{\mathsf{A}}_q(K) = \uparrow (K \setminus Les^{\mathsf{A}}_q) \cup (\uparrow K \setminus Les^{\mathsf{A}}_q)$.

Proof. Item 1 follows immediately by the semantics of guards in section 2.1. For item 2, first note that $\downarrow (q^! \setminus \downarrow (q^? \setminus \downarrow q^!)) =$

$$\left\{\nu \,\middle|\, \exists t : \nu + t \in q^! \wedge (\forall t' \geq t : \nu + t' \in q^? \implies \exists t'' \geq t' : \nu + t'' \in q^!)\right\} \tag{2}$$

Indeed:

$$\begin{array}{ll}
\downarrow (q^! \setminus \downarrow (q^? \setminus \downarrow q^!)) & = \\
\downarrow (q^! \setminus \downarrow (\left\{\nu \,\middle|\, \nu \in q^?\right\} \setminus \left\{\nu \,\middle|\, \exists t : \nu + t \in q^!\right\})) & = \\
\downarrow (q^! \setminus \downarrow (\left\{\nu \,\middle|\, \nu \in q^? \wedge \forall t : \nu + t \notin q^!\right\})) & = \\
\downarrow (q^! \setminus \left\{\nu \,\middle|\, \exists t : \nu + t \in q^? \wedge \forall t' \geq t : \nu + t' \notin q^!\right\}) & = \\
\downarrow \left\{\nu \,\middle|\, \nu \in q^! \wedge (\forall t : \nu + t \in q^? \implies \exists t' \geq t : \nu + t' \in q^!)\right\} & = \\
\left\{\nu \,\middle|\, \exists t : \nu + t \in q^! \wedge (\forall t' \geq t : \nu + t' \in q^? \implies \exists t'' \geq t' : \nu + t'' \in q^!)\right\}
\end{array}$$

Now, suppose that $\nu \in Les^{\mathsf{A}}_q$, 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:

$$\begin{array}{rl}
Post^{\mathsf{A}}_q(K) & = \left\{\nu + t \,\middle|\, \nu \in K \ \wedge \ (\nu \in Les^{\mathsf{A}}_q \implies \nu + t \in Les^{\mathsf{A}}_q)\right\} \\
& = \left\{\nu + t \,\middle|\, \nu \in K \ \wedge \ (\nu \notin Les^{\mathsf{A}}_q \vee \nu + t \in Les^{\mathsf{A}}_q)\right\} \\
& = \left\{\nu + t \,\middle|\, (\nu \in K \wedge \nu \notin Les^{\mathsf{A}}_q) \vee (\nu \in K \wedge \nu + t \in Les^{\mathsf{A}}_q)\right\} \\
& = \left\{\nu + t \,\middle|\, \nu \in K \wedge \nu \notin Les^{\mathsf{A}}_q\right\} \cup \left\{\nu + t \,\middle|\, \nu \in K \wedge \nu + t \in Les^{\mathsf{A}}_q\right\} \\
& = \uparrow \left\{\nu \,\middle|\, \nu \in K \wedge \nu \notin Les^{\mathsf{A}}_q\right\} \cup (\{\nu + t \,|\, \nu \in K\} \cap \left\{\nu \,\middle|\, \nu \in Les^{\mathsf{A}}_q\right\}) \\
& = \uparrow (K \setminus Les^{\mathsf{A}}_q) \cup (\uparrow K \setminus Les^{\mathsf{A}}_q)
\end{array}$$

□

Proof of Theorem 8

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. $\nu \in Les^{\mathsf{A}^2_{\mathtt{p}}}_{q_{\mathtt{p}}}$. 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 17, $\nu \in Post_{q_\mathtt{p}}^{\mathsf{A}^1_\mathtt{p}}(Pre_{q_\mathtt{p}}^{\mathsf{A}^1_\mathtt{p}})$. 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 $\nu \in Les_{q_\mathtt{p}}^{\mathsf{A}^1_\mathtt{p}}$.

Decidability follows by the fact that, by lemma 18, 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.6 Proofs for Section 5.5

Proof of Theorem 9

A simple inspection of definition 6. □

Proof of Theorem 10

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 $\gamma_0 \to_u^* \gamma = (\boldsymbol{q}, \boldsymbol{w}, \nu)$. First note that, by theorem 9, $\gamma_0 \to^* \gamma$ 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 $\mathtt{p}$ and $t' < t$ such that $q_\mathtt{p}$ has non-deferrable edge in $(\boldsymbol{q}, \boldsymbol{w}, \nu + t')$. Among these edges, pick the minimum element $e = q_\mathtt{p}, \ell, q'_\mathtt{p}$ 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. □