Basic Theory of Reduction Congruence for Two Timed Asynchronous π-Calculi

Martin Berger

Dept. of Computer Science, Queen Mary, Univ. of London.

Abstract. We study reduction congruence, the most widely used notion of equality for the asynchronous π-calculus with timers, and derive several alternative characterisations, one of them being a labelled asynchronous bisimilarity. These results are adapted to an asynchronous π-calculus with timers, locations and message failure. In addition we investigate the problem of how to distribute value-passing processes in a semantics-preserving way.

1 Introduction

The π-calculus has been used to good effect as a tool for modelling and reasoning about computation [6, 7, 18, 23, 26]. Unfortunately, it appears incomplete for compositional representation and verification of distributed systems. An important instance of what cannot be covered convincingly are network protocols, for example TCP, that implement reliable (under some mild constraints about the probability of message failures) FIFO channels on top of an unreliable message passing fabric. Typically, such protocols start a timer when sending a message and, if the corresponding acknowledgement doesn't arrive early enough or not at all, a time-out initiates a retransmission. Timed Automata, Time(d) Petri Nets, Timed CCS and many other formalisms have been proposed to help express this or similar phenomena. Unfortunately, they all seem insufficient to give *convincing* accounts of advanced programming languages containing primitives for distribution, such as Java or the POSIX libraries. The two key shortcomings are the lack in expressivity of the underlying non-distributed formalism (e.g. finite automata or CCS do not allow precise and compositional modelling of Java's non-distributed core) and incomplete integration of the different features that are believed to be necessary for modelling distributed systems (e.g. [1] lacks timing and many timed process algebras do not feature message failures among their primitive operations). As an initial move towards overcoming this expressivity gap, [5] augmented the asynchronous π-calculus with a timer, with locations, message-loss, location failure and the ability to save process state. The present text, a partial summary of [4], takes the next step and starts the study of two extensions in earnest by investigating the natural equality for π_t, the asynchronous π-calculus with timers and π_{mlt}, the asynchronous π-calculus with timers and message failure.

The remainder has two main parts. First, several characterisations of π_t's reduction congruence, the canonical equivalence for asynchronous π-calculi [12, 17], are obtained. The most useful of those is as a labelled bisimilarity. For other untyped π-calculi, weak labelled characterisations of reduction congruence have not been forthcoming, only sound approximations. π_t is interesting because it allows to study the effect of combining discrete timing and name passing interaction, a line of inquiry long overdue. It also paves the way for the second part which studies π_{mlt}, a minimal extension of π_t allowing convenient expression of basic distributed algorithms such as the aforementioned network protocol. We show that reasoning about π_{mlt} can be broken down into two parts: reasoning about processes, i.e. reasoing in π_t, and reasoning about distributed interaction. A related aim is to devise a translation $(\cdot)^\bullet$ that allows to take a non-distributed process $\mathsf{P} \,|\, \mathsf{Q}$ and locate it as $[\mathsf{P}^\bullet] \,|\, [\mathsf{Q}^\bullet]$ in a semantics preserving way (here $[\cdot]$ denotes location). This may help to reason about some properties of distributed processes using tools from centralised computing. That may not be possible for arbitrary P and Q but we identify a translation that works for a restricted class of processes and may be a good starting point for further investigations. The other main contribution of this second part is a characterisation of π_{mlt}'s reduction congruence by a barbed congruence [21], and a sound approximation by a labelled bisimilarity.

2 Adding Discrete Timing to π

2.1 Syntax and Semantics of π_t

The π-calculus [16, 20] is a simple syntax for modelling computation as name-passing interaction. The key operational rule is

$$(\textsc{Com}) \quad \overline{x}\langle\vec{y}\rangle \,|\, x(\vec{v}).\mathsf{P} \rightarrow \mathsf{P}\{\vec{y}/\vec{v}\}$$

where the process $\overline{x}\langle\vec{y}\rangle$ sends the data $\vec{y}$ along a channel x (drawn from a countably infinite set $\mathcal{N}$ of *names*) and another process $x(\vec{v}).\mathsf{P}$ waits to receive data on the same channel x, called *input subject*. When the interaction happens, $x(\vec{v}).\mathsf{P}$ evolves into $\mathsf{P}\{\vec{y}/\vec{v}\}$. The operator $|$ is *parallel composition*. Finitely describable infinitary behaviour is achieved by way of a *replication* operation ! and interaction of P at x with the environment can be prevented by the *restriction* $(\nu x)\mathsf{P}$.

Our new syntax $\mathsf{timer}^t(x(\vec{v}).\mathsf{P}, \mathsf{Q})$, with $t > 0$ being an integer, is straightforward and completely standard. It supports two operations: (1) the *time-out* which means that after t steps it turns into Q, unless (2) it has been *stopped*, i.e. that a message has been received by the timer at x.

$$(\textsc{Stop}) \quad \mathsf{timer}^{t+1}(x(\vec{v}).\mathsf{P}, \mathsf{Q}) \,|\, \overline{x}\langle\vec{y}\rangle \rightarrow \mathsf{P}\{\vec{y}/\vec{v}\}$$

The resulting calculus is given by the following grammar.

$$\mathsf{P} ::= x(\vec{y}).\mathsf{P} \;\;|\;\; \overline{x}\langle\vec{y}\rangle \;\;|\;\; \mathsf{P}|\mathsf{Q} \;\;|\;\; (\nu x)\mathsf{P} \;\;|\;\; \mathsf{timer}^t(x(\vec{v}).\mathsf{P}, \mathsf{Q}) \;\;|\;\; !x(\vec{v}).\mathsf{P} \;\;|\;\; 0$$

We often abbreviate $\overline{x}\langle\rangle$ to $\overline{x}$ and write $x.\mathsf{P}$ in $x().\mathsf{P}$'s stead. The asynchronous π-calculus is a sub-calculus of π_t. The free and bound names of timers are:

$\mathsf{fn}(\mathsf{timer}^t(x(\vec{v}).\mathsf{P},\mathsf{Q})) = \mathsf{fn}(x(\vec{v}).\mathsf{P}) \cup \mathsf{fn}(\mathsf{Q})$, $\mathsf{bn}(\mathsf{timer}^t(x(\vec{v}).\mathsf{P},\mathsf{Q})) = \mathsf{bn}(x(\vec{v}).\mathsf{P}) \cup \mathsf{bn}(\mathsf{Q})$. Structural congruence $\equiv$ is defined by the same axioms as in the asynchronous π-calculus, but over our extended syntax.

The flow of time is communicated at each step in the computation by a *time stepper function* ϕ, which acts on processes. It models the implicit broadcast of time passing and works equally well for labelled transitions and reductions.

$$\phi(\mathsf{P}) = \begin{cases} \mathsf{timer}^{t-1}(x(\vec{v}).\mathsf{Q},\mathsf{R}) & \mathsf{P} = \mathsf{timer}^t(x(\vec{v}).\mathsf{Q},\mathsf{R}), t > 1 \\ \mathsf{R} & \mathsf{P} = \mathsf{timer}^t(x(\vec{v}).\mathsf{Q},\mathsf{R}), t \leq 1 \\ \phi(\mathsf{Q})|\phi(\mathsf{R}) & \mathsf{P} = \mathsf{Q}|\mathsf{R} \\ (\nu x)\phi(\mathsf{Q}) & \mathsf{P} = (\nu x)\mathsf{Q} \\ \mathsf{P} & \text{otherwise.} \end{cases}$$

Here is how time stepping is used.

$$(\text{PAR}) \quad \mathsf{P} \to \mathsf{P}' \;\Rightarrow\; \mathsf{P}|\mathsf{Q} \to \mathsf{P}'|\phi(\mathsf{Q})$$

The only difference with the corresponding rule of untimed calculi is that we have $\phi(\mathsf{Q})$ rather than Q in the resulting process of the conclusion. It ensures that each *active timer*, that is any timer not under a prefix, is ticked one unit at each interaction. The additional rule

$$(\text{IDLE}) \quad \mathsf{P} \to \phi(\mathsf{P})$$

prevents the flow of time from ever being halted by deadlocked processes. This means, π_t does not enforce progress assumptions that can be found in many models of timed computations. It is possible to add progress requirements later on top of π_t. Here are the remaining reduction rules.

$$(\text{REP}) \quad !x(\vec{v}).\mathsf{P} \mid \overline{x}\langle\vec{y}\rangle \;\to\; !x(\vec{v}).\mathsf{P} \mid \mathsf{P}\{\vec{y}/\vec{v}\}$$

$$(\text{RES}) \quad \mathsf{P} \to \mathsf{Q} \Rightarrow (\nu x)\mathsf{P} \to (\nu x)\mathsf{Q} \qquad (\text{CONG}) \quad \mathsf{P} \equiv \mathsf{P}' \to \mathsf{Q}' \equiv \mathsf{Q} \Rightarrow \mathsf{P} \to \mathsf{Q}$$

The corresponding labelled synchronous semantics is obtained from the conventional synchronous semantics [15] of the asynchronous π-calculus with the following new rules, the first of which replacing that for parallel composition.

$$\mathsf{P} \xrightarrow{l} \mathsf{P}', \mathsf{bn}(l) \cap \mathsf{fn}(\mathsf{Q}) = \emptyset \;\Rightarrow\; \mathsf{P} \mid \mathsf{Q} \xrightarrow{l} \mathsf{P}' \mid \phi(\mathsf{Q})$$

$$\mathsf{timer}^{t+1}(x(\vec{v}).\mathsf{P},\mathsf{Q}) \xrightarrow{x(\vec{z})} \mathsf{P}\{\vec{z}/\vec{v}\} \qquad\qquad \mathsf{P} \xrightarrow{\tau} \phi(\mathsf{P})$$

Labels are given as $l ::= \tau \mid x(\vec{y}) \mid \overline{x}\langle(\nu\vec{y})\vec{z}\rangle$. *Contexts* are standard except for two new rules: $C[\cdot] ::= \ldots \mid \mathsf{timer}^t(x(\vec{v}).\mathsf{P}, C'[\cdot]) \mid \mathsf{timer}^t(x(\vec{v}).C'[\cdot],\mathsf{P})$. A binary relation $\mathcal{R}$ on processes is a *π_t-congruence* if it is an equivalence, if $\equiv\, \subseteq \mathcal{R}$ and if $\mathsf{P}\ \mathcal{R}\ \mathsf{Q}$ implies $C[\mathsf{P}]\ \mathcal{R}\ C[\mathsf{Q}]$ for all contexts $C[\cdot]$. The *strong barb* $\downarrow_x$ for π_t is defined (up to $\equiv$) by (1) $\overline{x}\langle\vec{y}\rangle\downarrow_x$; (2) $\mathsf{P}\downarrow_x \;\Rightarrow\; \mathsf{P}|\mathsf{Q}\downarrow_x$; and (3) $\mathsf{P}\downarrow_x$, $x \neq y \Rightarrow (\nu y)\mathsf{P}\downarrow_x$. A symmetric binary relation $\mathcal{R}$ on processes is a *strong barbed bisimulation* if it is a π_t-congruence and if $\mathsf{P}\ \mathcal{R}\ \mathsf{Q}$ implies the following: (1) for all names x: $\mathsf{P}\downarrow_x$ implies $\mathsf{Q}\downarrow_x$; and (2) whenever $\mathsf{P} \rightarrow \mathsf{P}'$ then there is a process Q' such that $\mathsf{Q} \rightarrow \mathsf{Q}'$ and $\mathsf{P}' \mathcal{R} \mathsf{Q}'$. The largest strong barbed bisimulation $\overset{rc}{\sim}$ is *strong reduction congruence*. The corresponding notions of *barbed bisimulation* and *reduction congruence* $\overset{rc}{\approx}$ are derived by replacing $\downarrow_x$ with $\Downarrow_x$ and $\rightarrow$ with $\twoheadrightarrow$. Here $\twoheadrightarrow$ is the transitive and reflexive closure of $\rightarrow$ and $P \Downarrow_x$ means $P \twoheadrightarrow Q\downarrow_x$ for some Q. A binary relation $\mathcal{R}$ on processes is *time-closed* if $\mathsf{P} \mathcal{R} \mathsf{Q}$ implies $\phi(\mathsf{P}) \mathcal{R} \phi(\mathsf{Q})$. It will later emerge that $\overset{rc}{\approx}$ and $\overset{rc}{\sim}$ are time-closed.

Examples (1).

1. The process $\mathsf{delay}^t(\mathsf{P}) = (\nu x)\mathsf{timer}^t(x.\mathsf{0}, \mathsf{P})$ implements a *delay operator*, assuming $x \notin \mathsf{fn}(\mathsf{P})$. For t units of time, it cannot interact at all, it behaves like $\mathsf{0}$, but then it evolves into P. It is comparable to the `sleep` operator in Java and can be used to implement cyclic behaviour: $(\nu x)(\overline{x} \mid !x.\mathsf{delay}^t(\mathsf{P} \mid \overline{x}))$ $(x \notin \mathsf{fn}(\mathsf{P}))$ which spawns P every $t+1$ units of time. The delay operator is crucial in the proof of Theorem 2.
2. The next example shows that we only need $\mathsf{timer}^1(x(\vec{v}).\mathsf{P}, \mathsf{Q})$ as timing construct. As all others can be built up by iteration of this basic form. Define $\mathsf{T}_1 = \mathsf{timer}^1(x(\vec{v}).\mathsf{P}, \mathsf{Q})$ and $\mathsf{T}_{t+1} = \mathsf{timer}^1(x(\vec{v}).\mathsf{P}, \mathsf{T}_t)$. Then $\mathsf{T}_t \overset{rc}{\sim} \mathsf{timer}^t(x(\vec{v}).\mathsf{P}, \mathsf{Q})$ for all $t > 0$.
3. Assume that P is a process of the form $x(\vec{v}).\mathsf{Q}$. Define P^0 to be $\mathsf{0}$ and let $\mathsf{P}^{n+1} = \mathsf{timer}^1(\mathsf{P}, \mathsf{P}^n)$. Then P^n is a process that offers the service P for n time units when it becomes unguarded. Note that P is offered only once. $\mathsf{delay}^n(\mathsf{P}^m)$ also offers P for n units of time, but not straight away. Instead the service is available only after m units of time.
4. A variant of the previous example. Assume $\mathsf{P} = x(\vec{v}).\mathsf{Q}$ with $\mathsf{fn}(\mathsf{P}) = \{x\}, x \notin \{\vec{v}\}$. Let $\mathsf{P}^0 = \mathsf{0}$ and set $\mathsf{P}^{n+1} = \mathsf{timer}^1(x(\vec{v}).(\mathsf{Q} \mid \mathsf{P}^n), \mathsf{P}^n)$. Now P is offered for repeated use in P^n, for n units of time, so we may invoke P up to n times.

2.2 Why a Novel Kind of Timer?

Before getting on with the technical development, we'd like to summarise the key reasons for devising our own reduction-based account of discrete timing rather than adapting one of the existing constructs.

- A key design objective was simplicity and preservation of as much established π-calculus technology as possible. That ruled out labelled transitions with dedicated time passing actions to communicate the flow of time. The ability to use the simpler reduction semantics is advantageous because it is sometimes difficult to find suitable labelled semantics. It is trivial to adapt the timer proposed here to other models of computing, from Ambient Calculi [11], to λ-calculi and Abstract State Machines [9]. This is currently not possible for labelled-transition based approaches to timing.
- Some previous proposals exhibit behavioural anomalies, such as timers being able to stop themselves. This is caused, to put it simplistically, by less than ideal combinations of progress assumptions, the ability for time to pass under

unrestricted sums and computational steps having zero duration. The calculi proposed here do not suffer from these shortcomings.

– Finally, we must emphasise that our timer is different from those where time-flow is communicated by labelled transitions only in its syntactic presentation. Its behaviour is essentially identical. Semantically relevant differences between our calculus and its alternatives are a consequence of other design choices, for example progress assumptions or the presence of mixed choice, not of the presentation of timers by way of time-steppers.

Our design of π_t and π_{mlt} is discussed in great detail in [4], which also contains comparisons with the alternative approaches.

2.3 The Maximal Sound Theory

Reduction congruence is often seen to be the most canonical equivalence for asynchronous π-calculi. This section looks at its incarnation for π_t. The presentation is close to [17] to facilitate comparison, but due to timers, proofs are quite different.

A *logic* is a pair $\mathcal{L} = (F, \vdash)$ comprising a set F of *formulae* and an *entailment relation* $\vdash \subseteq \mathcal{P}(F) \times F$. In this section, F will always be pairs of π_t-processes. References to the underlying logic $\mathcal{L}$ will often be omitted. A set $\mathcal{T}$ of formulae is a π_t*-theory*, or simply a *theory*, and its members are *axioms*. We write $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$ whenever $(\mathcal{T}, (\mathsf{P}, \mathsf{Q})) \in \vdash$ and call (P, Q) a *theorem* or *consequence* of $\mathcal{T}$ in $\mathcal{L}$. If $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$ is not derivable, we write $\mathcal{T} \nvdash \mathsf{P} = \mathsf{Q}$. The set of all consequences of $\mathcal{T}$ in $\mathcal{L}$ is denoted $|\,\mathcal{T}\,|_{\mathcal{L}}$ (with the subscript $\mathcal{L}$ often omitted). $\mathcal{T}$ is *consistent* if $|\,\mathcal{T}\,|$ does not equate all processes, otherwise it is *inconsistent*. $\mathcal{T}$ is *reduction-closed* if $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$ and $\mathsf{P} \twoheadrightarrow \mathsf{P}'$ implies the existence of a reduction sequence $\mathsf{Q} \twoheadrightarrow \mathsf{Q}'$ such that $\mathcal{T} \vdash \mathsf{P}' = \mathsf{Q}'$. $\mathcal{T}$ is *strongly reduction-closed* if $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$ and $\mathsf{P} \rightarrow \mathsf{P}'$ implies the existence of a reduction $\mathsf{Q} \rightarrow \mathsf{Q}'$ such that $\mathcal{T} \vdash \mathsf{P}' = \mathsf{Q}'$. In this section we only use π_t*-logics* $(\mathcal{T}, \vdash)$ whose entailment is inductively defined such that $|\,\mathcal{T}\,|$ is a π_t-congruence containing $\mathcal{T}$. $\mathcal{T}$ is *time-closed* if $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$ implies $\mathcal{T} \vdash \phi(\mathsf{P}) = \phi(\mathsf{Q})$.

As is well-known, there is no unique largest consistent and reduction-closed theory (Theorem 1.2 below), so we have to impose a mild additional constraint. Preservation of weak barbs is a popular choice, but requires a notion of observation. Alas, it is not apriori clear what observing timed computations may entail. Fortunately, we can do without a notion of observation and will *prove* in Theorem 1 that $\downarrow_x$ defined above is in fact a correct notion of barb. A process P is *insensitive* if it can never interact with any other process, i.e. $\mathsf{P} \twoheadrightarrow \mathsf{Q}$ implies $\mathsf{an}(\mathsf{Q}) = \emptyset$. Here $\mathsf{an}(\mathsf{P})$, the *active names* of P, is given by induction on the syntax of P: $\mathsf{an}((\nu x)\mathsf{P}) = \mathsf{an}(\mathsf{P}) \setminus \{x\}$, $\mathsf{an}(\mathsf{P}|\mathsf{Q}) = \mathsf{an}(\mathsf{P}) \cup \mathsf{an}(\mathsf{Q})$, $\mathsf{an}(0) = \emptyset$ and $\mathsf{an}(x(\vec{v}).\mathsf{P}) = \mathsf{an}(!x(\vec{v}).\mathsf{P}) = \mathsf{an}(\mathsf{timer}^t(x(\vec{v}).\mathsf{P}, Q)) = \mathsf{an}(\overline{x}\langle\vec{y}\rangle) = \{x\}$. A π_t theory is *sound* if it is consistent, reduction-closed and equates any two insensitive terms.

The dramatic semantic effect of timers becomes apparent in the next proposition: we are guaranteed *strong* reduction-closure despite having stipulated only reduction-closure.

Proposition 1. *Let $\mathcal{T}$ be sound. (1) If $\mathcal{T} \vdash P = Q$, then: $\mathsf{P}\downarrow_x$ if and only if $\mathsf{Q}\downarrow_x$. (2) If $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$ then for all appropriate $\vec{x}$, $\vec{v}$: $\mathcal{T} \vdash \mathsf{P}\{\vec{x}/\vec{v}\} = \mathsf{Q}\{\vec{x}/\vec{v}\}$. (3) If $\mathcal{T}$ is a sound theory, then $\mathcal{T}$ is time-closed. (4) $\mathcal{T}$ is reduction-closed if and only if, whenever $\mathcal{T} \vdash \mathsf{P} = \mathsf{Q}$, then, for all contexts $C[\cdot]$, $C[\mathsf{P}] \rightarrow \mathsf{P}'$ implies $C[\mathsf{Q}] \rightarrow \mathsf{Q}'$, for some Q' with $\mathcal{T} \vdash \mathsf{P}' = \mathsf{Q}'$.*

The key reason why requiring reduction-closure and congruency gives strong reduction-closure is (roughly) that we can use a process like $\mathsf{timer}^1(x(\vec{v}).\overline{a}, \mathsf{0})$ to detect and signal the fact that $\mathsf{P}\downarrow_x$ by running both in parallel. After the first step of the clock, that ability disappears forever. Hence any process that wishes to be equated to P by a sound theory better be able to match any of P's strong barbs immediately and not only after some reduction steps.

With $\mathcal{T}_{\max} = \bigcup\{\mathcal{T} \mid \mathcal{T}$ is a sound theory$\}$ we can now state the existence and various alternative presentations of the maximal sound theory.

Theorem 1. *(1) $\mathcal{T}_{max}$ is the unique sound theory such that $|\,\mathcal{T}\,| \subseteq |\,\mathcal{T}_{max}\,|$ for all sound theories $\mathcal{T}$. $\mathcal{T}_{max}$ is called the* maximum sound theory. *(2) There is no largest consistent, reduction-closed theory. (3) $|\,\mathcal{T}_{max}\,| = \mathcal{T}_{\max} = \overset{rc}{\sim} = \overset{rc}{\approx}$.*

2.4 Labelled Semantics

Reduction based equivalences are sometimes hard to use. To make reasoning easier, labelled semantics and associated notions of bisimilarities have been developed for many untimed calculi. We shall now do the same for π_t. A symmetric binary relation $\mathcal{R}$ is a *strong synchronous bisimulation* if $\mathsf{P}\ \mathcal{R}\ \mathsf{Q}$ and $\mathsf{P} \xrightarrow{l} \mathsf{P}'$ means that there is a synchronous transition $\mathsf{Q} \xrightarrow{l} \mathsf{Q}'$ with $\mathsf{P}'\ \mathcal{R}\ \mathsf{Q}'$. The largest strong synchronous bisimulation $\sim$ is *strong synchronous bisimilarity.* Weak bisimilarity $\approx$ is defined by replacing $\mathsf{Q} \xrightarrow{l} \mathsf{Q}'$ with $\mathsf{Q} \overset{\hat{l}}{\twoheadrightarrow} \mathsf{Q}'$ ($\hat{\cdot}$ is the usual τ-erasing operation).

The failure of the various synchronous bisimilarities to equate fw_{xx} with $\mathsf{0}$ has lead to asynchronous transitions [15] which model asynchronous observers. Since $\mathcal{T}_{max}$, unlike $\approx$ and $\sim$, equates fw_{xx} and $\mathsf{0}$, asynchronous bisimilarity might also be interesting in π_t (here $\mathsf{fw}_{xy} = !x(\vec{v}).\overline{y}\langle\vec{v}\rangle$). But what are asynchronous transitions $\xrightarrow{l}_a$? Unfortunately, the straightforward adaptation to π_t of the transitions introduced in [15] does not work, because the obvious rule for parallel composition

$$\mathsf{P} \xrightarrow{l}_a \mathsf{P}', \mathsf{bn}(l) \cap \mathsf{fn}(\mathsf{Q}) = \emptyset \quad \Rightarrow \quad \mathsf{P} \mid \mathsf{Q} \xrightarrow{l}_a \mathsf{P}' \mid \phi(\mathsf{Q}) \qquad (1)$$

does not connect asynchrony well with time passing. To see what goes wrong consider what it means to be an asynchronous observer. Interacting with a process to detect that it sends a message consumes one unit of time. The (PAR) rule and its labelled counterpart (1) ensure that this time-step permeates all processes. Dually, testing that a process is inputting involves sending a message. But asynchronously entails that the observer cannot know exactly when the message has been consumed. Hence the observation $\xrightarrow{x(\vec{v})}_a$ should not be associated with a time step, for otherwise a judiciously set timer could detect that interaction by the time it takes. So the rule (1) for parallel composition above may work incorrectly. We propose to split it in two:

- $\mathsf{P} \xrightarrow{l}_a \mathsf{P}'$, $l \neq x(\vec{v})$, $\mathsf{bn}(l) \cap \mathsf{fn}(\mathsf{Q}) = \emptyset \;\Rightarrow\; \mathsf{P} \mid \mathsf{Q} \xrightarrow{l}_a \mathsf{P}' \mid \phi(\mathsf{Q})$
- $\mathsf{P} \xrightarrow{x(\vec{y})}_a \mathsf{P}'$, $\mathsf{bn}(l) \cap \mathsf{fn}(\mathsf{Q}) = \emptyset \;\Rightarrow\; \mathsf{P} \mid \mathsf{Q} \xrightarrow{x(\vec{y})}_a \mathsf{P}' \mid \mathsf{Q}$

The remaining rules for the inductive definition of $\xrightarrow{l}_a$ are here:

$$\overline{x}\langle\vec{y}\rangle \xrightarrow{\overline{x}\langle\vec{y}\rangle}_a \mathsf{0} \qquad\qquad \mathsf{P} \xrightarrow{l}_a \mathsf{Q}, x \notin \mathsf{fn}(l) \cup \mathsf{bn}(l) \Rightarrow (\nu x)\mathsf{P} \xrightarrow{l}_a (\nu x)\mathsf{Q}$$
$$\overline{x}\langle\vec{y}\rangle \mid x(\vec{v}).\mathsf{Q} \xrightarrow{\tau}_a \mathsf{Q}\{\vec{y}/\vec{v}\} \qquad\qquad \overline{x}\langle\vec{y}\rangle \mid !x(\vec{v}).\mathsf{Q} \xrightarrow{\tau}_a \mathsf{Q}\{\vec{y}/\vec{v}\} \mid !x(\vec{v}).\mathsf{Q}$$
$$\overline{x}\langle\vec{y}\rangle \mid \mathsf{timer}^t(x(\vec{v}).\mathsf{Q}, \mathsf{R}) \xrightarrow{\tau}_a \mathsf{Q}\{\vec{y}/\vec{v}\} \qquad\qquad P \xrightarrow{\tau}_a \phi(P)$$
$$\mathsf{P} \equiv \mathsf{P}', \mathsf{P}' \xrightarrow{l}_a \mathsf{Q}', \mathsf{Q}' \equiv \mathsf{Q} \Rightarrow \mathsf{P} \xrightarrow{l}_a \mathsf{Q} \qquad\qquad \mathsf{0} \xrightarrow{x(\vec{z})}_a \overline{x}\langle\vec{z}\rangle$$
$$\mathsf{P} \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle}_a \mathsf{Q}, a \neq x, a \in \{\vec{z}\} \setminus \{\vec{y}\} \Rightarrow (\nu a)\mathsf{P} \xrightarrow{\overline{x}\langle(\nu\vec{y},a)\vec{z}\rangle}_a \mathsf{Q}$$

The set of labels is the same as for synchronous transitions. *Strong asynchronous bisimilarity* $\sim_a$ and its weak counterpart $\approx_a$ are defined just as (strong) synchronous bisimilarity except that $\xrightarrow{l}$ is replaced with $\xrightarrow{l}_a$. The next lemma shows that timers also wreak havoc with labelled equivalences.

Lemma 1. *Neither $\approx$ nor $\sim$, $\approx_a$ and $\sim_a$ are closed under parallel composition.*

As an example of what may go wrong, note that $\mathsf{P} \stackrel{def}{=} (\nu x)(\overline{x} \mid \mathsf{timer}^1(x.\overline{y}, \mathsf{0})) \sim (\nu x)(\overline{x} \mid \mathsf{timer}^1(x.\mathsf{0}, \overline{y})) \stackrel{def}{=} \mathsf{Q}$ means $\mathsf{P} \mathrel{\mathcal{R}} \mathsf{Q}$ ($\mathcal{R}$ is any of the four equivalences in Lemma 1) but $\mathsf{Q} \mid \overline{a} \xrightarrow{\overline{a}}\xrightarrow{\overline{y}} (\nu x)\overline{x}$ cannot be matched by $\mathsf{P} \mid \overline{a}$.

This failure of closure under parallel composition is caused by lacking time-closure. Let $\sim'_a$ be the largest strong, asynchronous bisimulation that is also time-closed, with $\approx'$, $\sim'$ and $\approx'_a$ being defined similarly. Its easy to show that these four new equivalences are closed under parallel composition. Still, this does not guarantee congruency.

Proposition 2. *Assume x, y, a, b are fresh and distinct names. Define*

$$\mathsf{P} = (\nu a)(\overline{x}\langle a\rangle \mid !y(v).\overline{v})$$
$$\mathsf{Q} = (\nu a)(\overline{x}\langle a\rangle \mid !y(v).\overline{v} \;\mid \mathsf{timer}^1(y(v).(\overline{v} \mid \mathsf{timer}^1(a.\overline{b}, \mathsf{0})), \mathsf{0})).$$

If $\mathcal{R}$ is one of $\approx$, $\sim$, $\approx_a$, $\sim_a$, $\approx'$, $\sim'$, $\approx'_a$ or $\sim'_a$, then $\mathsf{P} \mathrel{\mathcal{R}} \mathsf{Q}$ but not $\mathsf{P}\{x/y\} \mathrel{\mathcal{R}} \mathsf{Q}\{x/y\}$. Consequently, $\mathcal{R}$ cannot be closed under any of the three available forms of input prefixing.

In the asynchronous π-calculus, various reasonable equivalences are congruences. That this fails for π_t hints at renaming carrying non-trivial computational content. Interestingly, our example uses nested timers. It is conceivable that prohibiting nesting of timers results in a subcalculus where the relevant equivalences are renaming-closed. The next result shows that failure of renaming-closure is the only defect $\sim'_a$ has vis-a-vis congruency. Define $\sim^c_a$ as the largest strong, time-closed, asynchronous bisimulation that is also renaming-closed.

Proposition 3. *$\sim^c_a$ is the largest strong asynchronous bisimulation contained in $\sim_a$ that is also a congruence.*

The processes P and Q, defined just after Lemma 1, also show that fully abstract and compositional encodings $[\![\cdot]\!]$ of π_t into the asynchronous π-calculus are impossible, when the equivalence $\mathcal{R}$ on the source of the encoding is one of those mentioned in Lemma 1. Otherwise we could derive $\mathsf{P} \sim \mathsf{Q} \Rightarrow [\![\mathsf{P}]\!] \sim [\![\mathsf{Q}]\!] \Rightarrow [\![\mathsf{P}]\!] \,|\, [\![\overline{a}]\!] \bowtie [\![\mathsf{Q}]\!] \,|\, [\![\overline{a}]\!] \Rightarrow [\![\mathsf{P} \,|\, \overline{a}]\!] \bowtie [\![\mathsf{Q} \,|\, \overline{a}]\!] \Rightarrow \mathsf{P} \,|\, \overline{a} \sim \mathsf{Q} \,|\, \overline{a}$ (the target's equivalence $\bowtie$ is only required to be closed under parallel composition for the encoding to be contradictory). The converse question is also interesting: can untimed subcalculi of π_t, for example the asynchronous π-calculus, be embedded? Once again the answer seems mostly negative: a translation $[\![\cdot]\!]$ from π_a into π_t is *barb-expansive* if for all P and all names x we can find an integer $n > 0$ such that $d([\![\mathsf{P}]\!], x) \geq n \cdot d(\mathsf{P}, x)$. Here $d(\mathsf{P}, x)$ is the least n such that $\mathsf{P} \underbrace{\rightarrow \cdots \rightarrow}_{n} \mathsf{Q}\downarrow_x$ and ω if no such n exists. Then one can easily show the following. Assume the chosen π_a-equivalence equates $\overline{x}$ with $\tau.\overline{x}$. If $[\![\cdot]\!]$ is a barb-expansive mapping from π_a into π_t, then it cannot be complete with reduction congruence being π_t's equivalence. In particular, the syntactic inclusion of π_a into π_t cannot be fully abstract.

2.5 Characterising $\mathcal{T}_{max}$ as $\sim^c_a$.

In the asynchronous π-calculus, asynchronous bisimilarity soundly approximates the corresponding maximal theory, but does not characterise it, a counterexample being $\overline{x}\langle y\rangle \,|\, \mathsf{eq}_{yz}$ and $\overline{x}\langle z\rangle \,|\, \mathsf{eq}_{yz}$, where $\mathsf{eq}_{yz} = \mathsf{fw}_{yz} \,|\, \mathsf{fw}_{zy}$ [17]. The reason for their semantic equality is that eq_{yz} turns any observation on y into a weak observation on z and vice versa. There is no way for a process in the asynchronous π-calculus to detect whether a name has come via eq_{yz} or not. In π_t this is different because forwarding takes time. This leads to the following labelled characterisation of reduction congruence.

Theorem 2. *$\mathcal{T}_{max} = \sim^c_a \subsetneq \sim'_a \subsetneq \sim_a \subsetneq \approx'_a \subsetneq \approx_a$. In addition $\sim \subsetneq \approx$ and $\approx \subsetneq \approx_a$.*

The proof is straightforward, except for showing $\mathcal{T}_{\max} \subseteq \sim^c_a$. The key difficulty is to establish that $\mathcal{T}_{\max} \vdash \mathsf{P} = \mathsf{Q}$ and $\mathsf{P} \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle} \mathsf{P}'$ together imply $\mathsf{Q} \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle} \mathsf{Q}'$ for some Q' with $\mathcal{T}_{\max} \vdash \mathsf{P}' = \mathsf{Q}'$. Simplifying greatly, the proof uses a context like

$$C[\cdot] = [\cdot] \mid x(\vec{v}).\Pi_{z_i \in \vec{z}} \Pi_{j=1}^{f(i)} \overline{z_i}\langle \ldots \rangle$$

which receives a tuple of names at x and encodes at what positions in the tuple $\vec{v}$ a name w was received by encoding these positions through the number of uninterrupted (even by τ) outputs of w. Here $\Pi_{i\in\{1,\ldots,n\}}\mathsf{P}_i \equiv \mathsf{P}_1|\ldots|\mathsf{P}_n$ and f is a suitable function allowing this encoding. The construction of f is delicate and omitted for brevity, but we cannot use simple functions like the identity $i \mapsto i$, because $C[\cdot]$ must be able to distinguish, for example, $\overline{x}\langle abaa\rangle$ from $\overline{x}\langle aaab\rangle$. Both have the same number of as and bs. This is why we must code up not only how many times a name occurs in $\vec{y}$ but also at which positions. Using the observational capabilities of timers, we can distinguish processes that can output a fixed name n times, but not $n+1$ times in an uninterrupted row from processes that can do more than n uninterrupted outputs of that name. Thus the sketched construction of $C[\cdot]$ ensures that $\mathcal{T}_{\max} \vdash C[\mathsf{P}] = C[\mathsf{Q}]$ can only hold if Q can do *exactly* the same initial outputs as P, which is what was needed to be shown. The actual proof is more complicated and can be found in [4].

Examples (2). The next few examples show how easy it is to reason about $\mathcal{T}_{max}$ with $\sim^c_a$.

1. The identity forwarder fw_{aa} and $\mathsf{0}$ are strongly reduction congruent. To see this, define $\mathcal{R}$ up to $\equiv$ by $\mathsf{fw}_{xx} \mid \Pi_i\overline{y_i}\langle\vec{z_i}\rangle \; \mathcal{R} \; \Pi_i\overline{y_i}\langle\vec{z_i}\rangle$ whenever $\{y_i, \vec{z_i}\} \subseteq \mathcal{N}$. Obviously $\mathcal{R}$ is time- and renaming closed. Since all occurring processes are timer-free, idle transitions can trivially be matched. The only vaguely interesting transition $\mathsf{fw}_{xx} \mid \overline{x}\langle\vec{a}\rangle \mid \Pi_i\overline{y_i}\langle\vec{z_i}\rangle \xrightarrow{\tau}_a \mathsf{fw}_{xx} \mid \overline{x}\langle\vec{a}\rangle \mid \Pi_i\overline{y_i}\langle\vec{z_i}\rangle$ is clearly matched by the idle transition $\overline{x}\langle\vec{a}\rangle \mid \Pi_i\overline{y_i}\langle\vec{z_i}\rangle \xrightarrow{\tau}_a \overline{x}\langle\vec{a}\rangle \mid \Pi_i\overline{y_i}\langle\vec{z_i}\rangle$.
2. To see that $\mathcal{T}_{max} \vdash \mathsf{T}_t = \mathsf{timer}^t(x(\vec{v}).\mathsf{P}, \mathsf{Q})$, simply define the relation $\mathcal{R}$ by $\mathsf{T}_t \mid \Pi^n_{i=1}\overline{x_i}\langle\vec{y_i}\rangle \; \mathcal{R} \; \mathsf{timer}^t(x(\vec{v}).\mathsf{P}, \mathsf{Q}) \mid \Pi^n_{i=1}\overline{x_i}\langle\vec{y_i}\rangle$. Verification that $\mathcal{R}$ has all the required closure properties is easy.
3. Parallel composition and delay operators commute, i.e. $\mathcal{T}_{max} \vdash \mathsf{delay}^t(\mathsf{P}|\mathsf{Q}) = \mathsf{delay}^t(\mathsf{P}) \mid \mathsf{delay}^t(\mathsf{Q})$: consider $\mathcal{R}$ given by $\mathsf{delay}^t(\mathsf{P}|\mathsf{Q}) \; \mathcal{R} \; \mathsf{delay}^t(\mathsf{P}) \mid \mathsf{delay}^t(\mathsf{Q})$. It is again straightforward to verify that $\mathcal{R} \cup \mathsf{id}$ is a renaming-closed, time-closed, asynchronous bisimulation.

Locality. A process P is *local* no input is bound by another input, i.e. we do not allow processes like $x(y).y(v).\mathsf{P}$. We denote π_t restricted to local processes by π_t^{loc}. Local processes are convenient for modelling distributed computing.

Theorem 3. *All results stated so far also hold in* π_t^{loc}.

3 Adding Location and Message Failure

One of the main uses of timers is to unblock computations after they became stuck due to some fault such as a lost message. This is inconvenient to model in π_t because it lacks message failures. To explore timers in a more realistic setting, this section augments π_t with locations and non-byzantine message failure, obtaining π_{mlt}.

3.1 Syntax and Semantics of π_{mlt}

Processes in π_{mlt}, called *networks* and closely related to, but not identical with [5], are parallel compositions of *messages in transit* $\overline{x}\langle\vec{y}\rangle$ and *locations* or *sites* $[\mathsf{P}]_A$ which execute π_t processes. Restriction of names is also possible for networks using (νx). For simplicity P must be *local* and the subscript A contains the free names that $[\mathsf{P}]_A$ may use to receive data on. Messages in transit have left their source location but not yet arrived at the destination. Message failure occurs only in transit and can involve loss and duplication of messages.

In summary, our networks are generated by the grammar below.

$$\mathsf{N} ::= \overline{x}\langle\vec{y}\rangle \ \mid\ [\mathsf{P}]_A \ \mid\ \mathsf{N}_1|\mathsf{N}_2 \ \mid\ (\nu x)\mathsf{N} \ \mid\ \mathsf{0}$$

N is *well-formed*, written $\vdash \mathsf{N}$, if $\vdash \mathsf{N}$ is derivable using the following rules. (1) $\vdash \mathsf{0}$ is always derivable; (2) $\vdash [\mathsf{P}]_A$ if P is local and each free input subject in P is in A; (3) $\vdash \mathsf{N}_1|\mathsf{N}_2$ if $\vdash \mathsf{N}_1$ and $\vdash \mathsf{N}_2$ and, moreover, $\mathsf{ap}(\mathsf{N}_1) \cap \mathsf{ap}(\mathsf{N}_2) = \emptyset$; (4) $\vdash (\nu x)\mathsf{N}$ if $\vdash \mathsf{N}$. Here the *access points* $\mathsf{ap}(\mathsf{N})$ of a network N are given by: $\mathsf{ap}([\mathsf{P}]_A) = A$, $\mathsf{ap}(\mathsf{N}_1|\mathsf{N}_2) = \mathsf{ap}(\mathsf{N}_1) \cup \mathsf{ap}(\mathsf{N}_2)$ and $\mathsf{ap}((\nu x)\mathsf{N}) = \mathsf{ap}(\mathsf{N}) \setminus \{x\}$. The *free names* of networks are given by $\mathsf{fn}(\overline{x}\langle\vec{y}\rangle) = \{x, \vec{y}\}$, $\mathsf{fn}([\mathsf{P}]_A) = \mathsf{fn}(\mathsf{P}) \cup A$, $\mathsf{fn}(\mathsf{M}|\mathsf{N}) = \mathsf{fn}(\mathsf{M}) \cup \mathsf{fn}(\mathsf{N})$, $\mathsf{fn}((\nu x)\mathsf{N}) = \mathsf{fn}(\mathsf{N}) \setminus \{x\}$, $\mathsf{fn}(\mathsf{0}) = \emptyset$. Bound names are omitted. In the remainder of this text, we assume that expressions involving networks such as $[\mathsf{P}]_A$ are well-formed. In particular, quantifications like: "for all P and all A, $[\mathsf{P}]_A$ has property X" or even "for all P, $[\mathsf{P}]_A$ has property X" abbreviate the statement: "for all P and all A such that $[\mathsf{P}]_A$ is well-formed, $[\mathsf{P}]_A$ has property X". On networks, $\equiv$ is generated by the axioms below.

$$\begin{array}{ll}
\mathsf{M} \equiv_\alpha \mathsf{N} \Rightarrow \mathsf{M} \equiv \mathsf{N} & \mathsf{M} \,|\, \mathsf{N} \equiv \mathsf{N} \,|\, \mathsf{M} \\
\mathsf{L} \,|\, (\mathsf{M} \,|\, \mathsf{N}) \equiv (\mathsf{L} \,|\, \mathsf{M}) \,|\, \mathsf{N} & \mathsf{M} \,|\, \mathsf{0} \equiv \mathsf{M} \\
x \notin \mathsf{fn}(\mathsf{M}) \Rightarrow \mathsf{M}|(\nu x)\mathsf{N} \equiv (\nu x)(\mathsf{M}|\mathsf{N}) & (\nu x)(\nu y)\mathsf{M} \equiv (\nu y)(\nu x)\mathsf{M} \\
(\nu x)\mathsf{0} \equiv \mathsf{0} & [(\nu x)\mathsf{M}]_A \equiv (\nu x)[\mathsf{M}]_{A\cup\{x\}} \\
[\mathsf{0}]_\emptyset \equiv \mathsf{0} & \mathsf{P} \equiv \mathsf{Q} \Rightarrow [\mathsf{P}]_A \equiv [\mathsf{Q}]_A
\end{array}$$

One of the key objectives in the design of π_{mlt} was to retain the semantics of the underlying π_t, to allow separation of reasoning about networks from reasoning about processes. Hence the first reduction rule.

$$(\textsc{Intra}) \quad \mathsf{P} \to \mathsf{Q} \Rightarrow [\mathsf{P}]_A \to [\mathsf{Q}]_A$$

Inter-site communication happens by message migration.

$$\begin{array}{ll}
(\textsc{Out}) & x \notin A, [\overline{x}\langle\vec{y}\rangle|\mathsf{P}]_A \to [\phi(\mathsf{P})]_A|\overline{x}\langle\vec{y}\rangle \\
(\textsc{In}) & x \in A, [\mathsf{P}]_A|\overline{x}\langle\vec{y}\rangle \to [\mathsf{P} \mid \overline{x}\langle\vec{y}\rangle]_A
\end{array}$$

Incursion of one time step in $(\textsc{Out})$ is crucial for a smooth integration of π_t^{loc} into π_{mlt}. Message failures arise from the following rules (which deal with messages in transit only).

$$(\textsc{Loss}) \quad \overline{x}\langle\vec{y}\rangle \to \mathsf{0} \qquad\qquad (\textsc{Dupl}) \quad \overline{x}\langle\vec{y}\rangle \to \overline{x}\langle\vec{y}\rangle \,|\, \overline{x}\langle\vec{y}\rangle$$

Many distributed systems offer only weak guarantees on the upper bound of inter-location clock drift. (PAR) reflects this by not synchronising different sites through application of time-stepping.

$$(\text{PAR}) \quad \mathsf{M} \to \mathsf{M}' \Rightarrow \mathsf{M}|\mathsf{N} \to \mathsf{M}'|\mathsf{N}$$

The remaining rules are:

$$(\text{CONG}) \quad \mathsf{M} \equiv \mathsf{M}' \to \mathsf{N}' \equiv \mathsf{N} \Rightarrow \mathsf{M} \to \mathsf{N} \qquad (\text{RES}) \quad \mathsf{M} \to \mathsf{N} \Rightarrow (\nu x)\mathsf{M} \to (\nu x)\mathsf{N}.$$

A binary relation $\mathcal{R}$ on processes is a π_{mlt}*-congruence* if it is an equivalence, if $\equiv\ \subseteq \mathcal{R}$ and if $\mathsf{P}\,\mathcal{R}\,\mathsf{Q}$ implies $C[\mathsf{P}]\,\mathcal{R}\,C[\mathsf{Q}]$ for all network contexts $C[\cdot]$. *Network contexts* are given by the grammar $C[\cdot] ::= [\cdot] \mid C[\cdot]\,|\,\mathsf{N} \mid (\nu x)C[\cdot]$. Barbs are generated by the following rules. $\mathsf{M}\downarrow_x$ and $x \notin \mathsf{ap}(\mathsf{N})$ imply $\mathsf{M}|\mathsf{N}\downarrow_x$, $\mathsf{M}\downarrow_x$ and $x \neq a$ imply $(\nu a)\mathsf{M}\downarrow_x$ and $\overline{x}\langle\vec{y}\rangle\downarrow_x$. A symmetric binary relation $\mathcal{R}$ on networks is a *strong barbed bisimulation* if it is a π_{mlt}-congruence and if $\mathsf{M}\ \mathcal{R}\ \mathsf{N}$ implies: (1) for all names x: $\mathsf{M}\downarrow_x \Rightarrow \mathsf{N}\downarrow_x$; and (2) whenever $\mathsf{M} \to \mathsf{M}'$ then there is a network N' such that $\mathsf{N} \to \mathsf{N}'$ and $\mathsf{M}'\ \mathcal{R}\ \mathsf{N}'$. The largest barbed bisimulation $\overset{rc}{\sim}$ is called *strong reduction congruence*. *Barbed bisimulation* and *reduction congruence* $\overset{rc}{\approx}$ are derived as usual.

Examples (3).

1. Let $\mathsf{fw}_{xy} = !x(v).\overline{y}\langle v\rangle$. Then the network $[\overline{x}\langle a\rangle]_\emptyset \mid [\mathsf{fw}_{xy}]_x \mid [\mathsf{fw}_{yz}]_y \mid [z(v).\mathsf{Q}]_z$ tries to relay the message $\overline{x}\langle a\rangle$ via two intermediate hops to $[z(v).Q]_z$, where it will be used by Q. It can be seen as a distributed version of $\overline{x}\langle a\rangle \mid \mathsf{fw}_{xy} \mid \mathsf{fw}_{yz} \mid z(v).\mathsf{Q}$, but semantically it is rather different, due to message loss and duplication.
2. The next example shows how to deal with message failure.

   $$[(\nu ab)(\overline{x}\langle\vec{y}a\rangle \,|\, \mathsf{timer}^t(a,\overline{b}) \,|\, !b.(\overline{x}\langle\vec{y}a\rangle \,|\, \mathsf{timer}^t(a,\overline{b})))|\mathsf{P}]_A \mid [x(\vec{v}a).(\overline{a} \mid \mathsf{Q})]_B.$$

   The location on the left sends a message to that on right and sets a timer to wait for an acknowledgement. If that doesn't come in time, it resends the original message.
3. We can also locate the time services of Example 1(3) as $[\mathsf{delay}^n(\mathsf{P}^m)]_A$ but because there is no synchronisation of time between sites, this is not very effective: the location is bisimilar to $[\mathsf{P} \oplus 0]_A$.

The last example is indicative of π_{mlt}'s being too asynchronous for realistic models of distributed systems. In other aspects, too, this calculus is overly idealising, for example in its lack of location failure. The point of π_{mlt} is rather to facilitate the study of message failure in isolation, as a first step towards more realistic models.

3.2 The Maximal Sound Theory

The development in this section mirrors that for π_t with proof being similar, albeit more involved because of possible message failure. π_{mlt}-logics $(\mathcal{T}, \vdash)$ are like π_t-logics, except that formulae are now pairs (M, N) of networks such that $\mathsf{ap}(\mathsf{M}) = \mathsf{ap}(\mathsf{N})$.

As in π_t, there is no maximal consistent and reduction-closed theory. A network M is *insensitive* if $\mathsf{an}(\mathsf{N}) = \emptyset$ for all reduction sequences $\mathsf{M} \twoheadrightarrow \mathsf{N}$, where *active names* for networks extend those of processes: $\mathsf{an}(\overline{x}\langle\vec{y}\rangle) = \{x\}$, $\mathsf{an}([\mathsf{P}]_A) = \mathsf{an}(\mathsf{P})$, $\mathsf{an}(\mathsf{M}|\mathsf{N}) = \mathsf{an}(\mathsf{M}) \cup \mathsf{an}(\mathsf{N})$, $\mathsf{an}((\nu x)\mathsf{N}) = \mathsf{an}(\mathsf{N}) \setminus \{x\}$, $\mathsf{an}(0) = \emptyset$. A theory is *sound* if it is consistent, reduction-closed and identifies all insensitive terms. As before, we set $\mathcal{T}_{\max} = \bigcup\{\mathcal{T} \mid \mathcal{T} \text{ is a sound theory}\}$. $\mathcal{T}_{\max}$ is called the *maximum sound theory.*

Theorem 4. *(1) $\mathcal{T}_{max}$ is the unique sound theory such that $|\,\mathcal{T}\,| \subseteq |\,\mathcal{T}_{max}\,|$ for all sound theories $\mathcal{T}$. (2) There is no largest, consistent, reduction-closed theory. (3) $|\,\mathcal{T}_{max}\,| = \mathcal{T}_{max} = \overset{rc}{\approx}$, $\overset{rc}{\sim} \subsetneq \overset{rc}{\approx}$.*

3.3 Labelled Semantics.

As with π_t, we present an asynchronous transition system $\xrightarrow{l}_a$. The induced asynchronous bisimilarity soundly approximates $\mathcal{T}_{max}$, but does not characterise it. Characterisation fails because the timers in different sites are not synchronised. The most interesting rule is that for parallel composition

$$\mathsf{M} \xrightarrow{l}_a \mathsf{M}', \mathsf{bn}(l) \cap \mathsf{fn}(\mathsf{N}) = \emptyset, (l = \overline{x}\langle(\nu\vec{y})\vec{z}\rangle \Rightarrow x \notin \mathsf{ap}(\mathsf{N})) \Rightarrow \mathsf{M}|\mathsf{N} \xrightarrow{l}_a \mathsf{M}'|\mathsf{N}$$

The reason for the side condition $l = \overline{x}\langle(\nu\vec{y})\vec{z}\rangle \Rightarrow x \notin \mathsf{ap}(\mathsf{N})$ is that well-formed observers cannot input on channels that are in $\mathsf{ap}(\mathsf{N})$. The remaining rules follow.

$$\begin{array}{lr}
\mathsf{M} \xrightarrow{l}_a \mathsf{N}, x \notin \mathsf{fn}(l) \cup \mathsf{bn}(l) \Rightarrow (\nu x)\mathsf{M} \xrightarrow{l}_a (\nu x)\mathsf{N} & 0 \xrightarrow{x(\vec{y})}_a \overline{x}\langle\vec{y}\rangle \\
x \notin A \Rightarrow [\mathsf{P}|\overline{x}\langle\vec{y}\rangle]_A \xrightarrow{\tau}_a [\phi(\mathsf{P})]_A \mid \overline{x}\langle\vec{y}\rangle & \overline{x}\langle\vec{y}\rangle \xrightarrow{\overline{x}\langle\vec{y}\rangle}_a 0 \\
x \in A \Rightarrow [\mathsf{P}]_A \mid \overline{x}\langle\vec{y}\rangle \xrightarrow{\tau}_a [\mathsf{P}|\overline{x}\langle\vec{y}\rangle]_A & \overline{x}\langle\vec{z}\rangle \xrightarrow{\tau}_a 0 \\
\mathsf{M} \equiv \mathsf{M}' \xrightarrow{l}_a \mathsf{N}' \equiv \mathsf{N} \Rightarrow \mathsf{M} \xrightarrow{l}_a \mathsf{N} & \overline{x}\langle\vec{z}\rangle \xrightarrow{\tau}_a \overline{x}\langle\vec{z}\rangle \mid \overline{x}\langle\vec{z}\rangle \\
\mathsf{P} \xrightarrow{\tau}_a \mathsf{Q} \Rightarrow [\mathsf{P}]_A \xrightarrow{\tau}_a [\mathsf{Q}]_A & \\
\mathsf{M} \xrightarrow{\overline{x}\langle(\nu\vec{y})\vec{z}\rangle}_a \mathsf{N}, a \neq x, \in \{\vec{z}\} \setminus \{\vec{y}\} \Rightarrow (\nu a)\mathsf{M} \xrightarrow{\overline{x}\langle(\nu\vec{y},a)\vec{z}\rangle}_a \mathsf{N} &
\end{array}$$

A symmetric relation $\mathcal{R}$ is an *strong asynchronous bisimulation* if $(\mathsf{M}, \mathsf{N}) \in \mathcal{R}$ implies whenever $\mathsf{M} \xrightarrow{l}_a \mathsf{M}'$ then there is a transition sequence $\mathsf{N} \xrightarrow{l}_a \mathsf{N}$ such that $\mathsf{M}' \,\mathcal{R}\, \mathsf{N}'$. The largest strong asynchronous bisimulation $\sim_a$ is called strong asynchronous bisimilarity. The largest asynchronous bisimulation $\approx_a$ is defined analogously.

Theorem 5. *(1) If $\mathcal{T}'_{max} \vdash \mathsf{P} = \mathsf{Q}$ then $[\mathsf{P}]_A \sim_a [\mathsf{Q}]_A$, where $\mathcal{T}'_{max}$ is the maximal sound theory on π_t^{loc}; (2) $\approx_a$ is a congruence; (3) $\approx_a$ is not closed under renaming. (4) $\sim_a \subsetneq \approx_a \subsetneq \overset{rc}{\approx}$.*

To see that $\mathcal{T}_{max}$ properly includes $\approx_a$ consider $[\mathsf{P}|\mathsf{eq}_{yz}|\overline{x}\langle y\rangle]_A \not\approx_a [\mathsf{P}|\mathsf{eq}_{yz}|\overline{x}\langle z\rangle]_A$ where A contains y, z and P is an arbitrary process. To verify that these two networks are related by $\mathcal{T}_{max}$, define $\mathcal{T}$ by

$$\begin{array}{ll} [\mathsf{P}|\mathsf{eq}_{yz}|\overline{x}\langle y\rangle|\Pi_{i\in I}\overline{c_i}\langle\vec{d_i}\rangle]_A|\Pi_{j\in J}\overline{a_j}\langle\vec{b_j}\rangle & \mathcal{T}\ [\mathsf{Q}|\mathsf{eq}_{yz}|\overline{x}\langle y\rangle|\Pi_{i\in I}\overline{c_i}\langle\vec{d_i}\rangle]_A|\Pi_{j\in J}\overline{a_j}\langle\vec{b_j}\rangle \\ [\mathsf{P}]|\mathsf{eq}_{yz}|\Pi_{i\in I}\overline{a_i}\langle\vec{b_i}\rangle]_A|\Pi_{j\in J}\overline{c_j}\langle\vec{d_j}\rangle & \mathcal{T}\ [\mathsf{Q}|\mathsf{eq}_{yz}|\Pi_{i\in I}\overline{a_i}\langle\vec{b_i}\rangle]_A|\Pi_{j\in J}\overline{c_j}\langle\vec{d_j}\rangle \end{array}$$

It is possible but laborious to verify that $\mathcal{T} \cup \{(\mathsf{M},\mathsf{N})|\mathsf{M},\mathsf{N}$ insensitive$\}$ is a sound theory.

This theorem shows that π_{mlt} integrates and extends π_t^{loc} in a strong sense. Congruency and failure of renaming-closure can coexist because π_{mlt} does not have prefixing operators.

3.4 Locating Processes

How expressive is π_{mlt} compared with π_t? It might be possible to modify the separation result in [10] to show that π_t cannot (nicely) encode π_{mlt}. The other way round may be more interesting: how is (discretely timed) name-passing affected by message failure? Would it be possible to design a non-distributed process first – without having to worry about distribution – and then scaffold it so that it can function in a distributed setting? This roughly boils down to finding a transformation $(\cdot)^\bullet$ that allows to go from non-located, failure-free processes $\mathsf{P}\,|\,\mathsf{Q}$ to $[\mathsf{P}^\bullet]_A\,|\,[\mathsf{Q}^\bullet]_B$ in a semantics preserving way. Without message failure, that would not be a problem, but loosing messages might lead to deadlocks and duplicated messages may confuse a receiver. We suspect that no appropriate encoding $(\cdot)^\bullet$ could work for all π_t processes. But that does not mean translations must fail for all processes. As an example of a class of processes that allows distribution, let P, Q be timer free and $x \notin \mathsf{fn}(\mathsf{P}) \cup \mathsf{fn}(\mathsf{Q})$. Assume we wanted to distribute $\mathsf{P}\,|\,\overline{x}\langle\vec{y}\rangle$ and $x(\vec{v}).\mathsf{Q}$ as $[(\mathsf{P}\,|\,\overline{x}\langle\vec{y}\rangle)^\bullet]_A\,|\,[(x(\vec{v}).\mathsf{Q})^\bullet]_B$. By the conditions on free names, message duplication is no problem. To overcome message loss, we replace $\overline{x}\langle\vec{y}\rangle$ with $(\nu ab)(\overline{x}\langle\vec{y}a\rangle\,|\,\mathsf{timer}^t(a,\overline{b})\,|\,!b.(\overline{x}\langle\vec{y}a\rangle\,|\,\mathsf{timer}^t(a,\overline{b})))$ and $x(\vec{v}).\mathsf{Q}$ with $x(\vec{v}a).(\overline{a}\,|\,\mathsf{Q})$ (a, b fresh and ignoring the scaffolding of P and Q for brevity), i.e. we do what TCP does to deal with message loss and add an explicit acknowledgement. If that isn't returned in time, the original message is resent. The resulting distributed process is

$$[\mathsf{P}^\bullet\,|\,(\nu ab)(\overline{x}\langle\vec{y}a\rangle\,|\,\mathsf{timer}^t(a,\overline{b})\,|\,!b.(\overline{x}\langle\vec{y}a\rangle\,|\,\mathsf{timer}^t(a,\overline{b})))]_A\ |\ [x(\vec{v}a).(\overline{a}\,|\,\mathsf{Q}^\bullet)]_B.$$

It is equated by $\mathcal{T}_{max}$ with $[\mathsf{P}^\bullet]_A\ |\ [\mathsf{Q}^\bullet\{\vec{y}/\vec{v}\}]_B$ as we sketch later. This translation is quite inefficient, it even introduces divergence, but that does not matter because – due to the absence of inter-site clock synchronisation – $\mathcal{T}_{max}$ is divergence insensitive. More sophisticated variants of our translations are possible, the pragmatically most important being putting an upper bound on the number of retransmissions and making time-out times contingent on the number of failed retransmissions. It would also be possible to dispense with acknowledgement and time-outs altogether: simply use $(\nu a)(\overline{a} \mid !a.(\overline{x}\langle\vec{y}\rangle \mid \overline{a}))$ to flood the receiver with an unbounded number of messages. This brute force approach is semantically sound under the aforementioned constraints, but it has less potential for generalisation and refinement, whether by using less asynchronous equivalences or by limiting the number of retransmissions.

Continuing with the process above, we show that $[\mathsf{P}^\bullet]_A \mid [\mathsf{Q}^\bullet\{\vec{y}/\vec{v}\}]_B$ is related by $\approx_a$ to $[\mathsf{P}^\bullet|(\nu ab)(\overline{x}\langle\vec{y}\rangle|\mathsf{timer}^t(a,\overline{b})|!b.(\overline{x}\langle\vec{y}\rangle|\mathsf{timer}^t(a,\overline{b}))]_A|[x(\vec{v}a).(\overline{a}|\mathsf{Q}^\bullet)]_B$. Set $\mathsf{U}^t = (\mathsf{timer}^t(a,\overline{b}) \mid !b.\overline{x}\langle\vec{y}\rangle \mid \mathsf{timer}^t(a,\overline{b}))$. In addition, let $\mathsf{R} \oplus \mathsf{S}$, the *internal sum of* R *and* S, be the process $(\nu a)(a.\mathsf{R} \mid a.\mathsf{S} \mid \overline{a})$, where a is fresh. Then we can reason in little steps as follows.

$$
\begin{aligned}
&[\mathsf{P}^\bullet \mid (\nu ab)(\overline{x}\langle\vec{y}a\rangle \mid \mathsf{U}^t)]_A \mid [x(v\vec{a}).(\overline{a} \mid \mathsf{Q}^\bullet)]_B \\
&\quad\equiv\ (\nu ab)([\mathsf{P}^\bullet \mid \overline{x}\langle\vec{y}a\rangle \mid \mathsf{U}^t]_{A\cup\{ab\}} \mid [x(v\vec{a}).(\overline{a} \mid \mathsf{Q}^\bullet)]_B) \\
&\quad\approx_a (\nu ab)([\mathsf{P}^\bullet \mid \mathsf{U}^t]_{A\cup\{ab\}} \mid [\overline{x}\langle\vec{y}a\rangle \mid x(\vec{v}a).(\overline{a} \mid \mathsf{Q}^\bullet)]_B) \\
&\quad\approx_a (\nu ab)([\mathsf{P}^\bullet \mid \mathsf{U}^t]_{A\cup\{ab\}} \mid [\overline{a} \mid \mathsf{Q}^\bullet\{\vec{y}/\vec{v}\}]_B) \\
&\quad\approx_a (\nu ab)([\mathsf{P}^\bullet \mid \mathsf{U}^t \mid \overline{a}]_{A\cup\{ab\}} \mid [\mathsf{Q}^\bullet\{\vec{y}/\vec{v}\}]_B) \\
&\quad=\ (\nu ab)([\mathsf{P}^\bullet \mid !b.(\overline{x}\langle\vec{y}a\rangle \mid \mathsf{timer}^t(a,\overline{b})) \mid \mathsf{timer}^t(a,\overline{b}) \mid \overline{a}]_{A\cup\{ab\}} \mid [\mathsf{Q}^\bullet\{\vec{y}/\vec{v}\}]_B) \\
&\quad\approx_a (\nu ab)([\mathsf{P}^\bullet \mid !b.(\overline{x}\langle\vec{y}a\rangle \mid \mathsf{timer}^t(a,\overline{b})) \mid 0 \oplus \overline{b}]_{A\cup\{ab\}} \mid [\mathsf{Q}^\bullet\{\vec{y}/\vec{v}\}]_B) \\
&\quad\approx_a (\nu ab)([\mathsf{P}^\bullet \mid !b.\mathsf{timer}^t(a,\overline{b}) \mid 0 \oplus \overline{b}]_{A\cup\{ab\}} \mid [\mathsf{Q}^\bullet\{\vec{y}/\vec{v}\}]_B) \\
&\quad\approx_a (\nu ab)([\mathsf{P}^\bullet]_{A\cup\{ab\}} \mid [\mathsf{Q}^\bullet\{\vec{y}/\vec{v}\}]_B) \\
&\quad\equiv\ (\nu ab)[\mathsf{P}^\bullet]_{A\cup\{ab\}} \mid [\mathsf{Q}^\bullet\{\vec{y}/\vec{v}\}]_B \\
&\quad\approx_a [\mathsf{P}^\bullet]_A \mid [\mathsf{Q}^\bullet\{\vec{y}/\vec{v}\}]_B
\end{aligned}
$$

The justification of all the individual steps by defining appropriate bisimulations is straightforward, but rather tedious – [4] has all the details.

4 Conclusion

Models of timed computation are legion, we mention [8, 14] in lieu of a comprehensive overview. A close look at the omitted proofs reveals that bound name passing plays no significant role – scope mobility seems orthogonal to timing, at least in this early stage of integration. This promises easy transfer of the presented technology to other timed calculi. Formalisms for distributed computing are also too numerous to survey here. Most closely related are Dpi [24], Nomadic Pict [25] and the Join Calculus [13]. Other influential distributed extensions of π-calculi can be found in [2, 3, 22]. Possibly the most important criticism of π_t is that it is too synchronous, but also too asynchronous for realistic models. Too synchronous because the absence of clock-drift forces many (in)equalities that might be inappropriate, the coincidence of $\overset{rc}{\approx}$ and $\overset{rc}{\sim}$ being an example. Always allowing time to pass by (Idle) means that important progress assumptions can be expressed only indirectly, leading to the charge of too much asynchrony. By modifying the time-stepper ϕ, it is possible to express clock-drift, thus coarsening equivalences. Having all timers to be of the form $\mathsf{timer}^{t \cdot n}(x(\vec{v}).\mathsf{P}, \mathsf{Q})$ for some fixed $n > 1$ may also be an important step towards more liberal equalities. Arbitrary progress assumptions can be studied by semantically restricting the set of valid traces. On the network level, π_{mlt} is also too asynchronous because it puts no constraints on inter-site clock-drift. With modern clock-synchronisation algorithms [19] it is possible to push clock-drift under the average inter-site communication latency (which is still many orders of magnitude above the duration of atomic computational steps). By modifying (Par) at the network level to also apply $\phi(\cdot)$, suitably augmented to allow intersite clock-drift, π_{mlt} may also become more realistic. A multidimensional open problem looming large is the expressive power of π_t and π_{mlt}. One of its most interesting facets is the question if the translation in §3.4 could be refined to allow a larger class of π_t-processes to be mechanically distributed into π_{mlt}.

References

1. Abdulla, P. A., and Jonsson, B. Verifying programs with unreliable channels. *Info. & Comp. 127*, 2 (1996), 91–101.
2. Amadio, R. M. An asynchronous model of locality, failure, and process mobility. In *Proc. COORDINATION 97* (1997), vol. 1282 of *LNCS*.
3. Amadio, R. M., and Prasad, S. Localities and failures. In *Proc. FSTTCS'94* (1994), vol. 880 of *LNCS*.
4. Berger, M. *Towards Abstractions for Distributed Systems*. PhD thesis, Imperial College, London, 2002.
5. Berger, M., and Honda, K. The Two-Phase Commit Protocol in an Extended π-Calculus. In *Proc. EXPRESS'00* (2000), vol. 39 of *ENTCS*.
6. Berger, M., Honda, K., and Yoshida, N. Sequentiality and the π-calculus. In *Proc. TLCA'01* (2001), vol. 2044 of *LNCS*.
7. Berger, M., Honda, K., and Yoshida, N. Genericity and the π-Calculus. In *Proc. FOSSACS'03* (April 2003), no. 2620 in LNCS, Springer, pp. 103–119.
8. Bergstra, J. A., Ponse, A., and Smolka, S. A., Eds. *Handbook of Process Algebra*. Elsevier, 2001.
9. Börger, E., and Stärk, R. *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer, 2003.
10. Carbone, M., and Maffeis, S. On the expressive power of polyadic synchronisation in pi-calculus. In *Proc. EXPRESS'02* (2002), vol. 68 of *ENTCS*.
11. Cardelli, L., and Gordon, A. Mobile ambients. *TCS 240* (2000).
12. Fournet, C., and Gonthier, G. A hierarchy of equivalences for asynchronous calculi. In *Proc. ICALP'98* (1998), no. 1443 in LNCS.
13. Fournet, C., Gonthier, G., Lévy, J.-J., Maranget, L., and Rémy, D. A Calculus of Mobile Agents. In *Proc. CONCUR* (1996), vol. 1119 of *LNCS*.
14. Hennessy, M. Timed process algebras: a tutorial. Tech. Rep. CS 1993:02, University of Sussex, Computer Science Department, 1993.
15. Honda, K. Two bisimilarities in ν-calculus. Tech. Rep. 92-002, Keio University, Department of Computer Science, 1992.

16. Honda, K., and Tokoro, M. On asynchronous communication semantics. In *Object-Based Concurrent Computing* (1992), no. 612, in LNCS.
17. Honda, K., and Yoshida, N. On reduction-based process semantics. *TCS 151* (1995).
18. Honda, K., and Yoshida, N. A uniform type structure for secure information flow. In *POPL'02* (2002), ACM Press, pp. 81–92.
19. Mills, D. Time synchronization server. URL http://www.eecis.udel.edu/~ntp/.
20. Milner, R., Parrow, J., and Walker, D. A calculus of mobile processes, parts I and II. *Info. & Comp. 100*, 1 (1992).
21. Milner, R., and Sangiorgi, D. Barbed bisimulation. In *Proc. ICALP'92* (1992), vol. 623 of *LNCS*.
22. Riely, J., and Hennessy, M. Distributed processes and location failures. *TCS 226* (2001).
23. Sangiorgi, D., and Walker, D. *The π-Calculus: a Theory of Mobile Processes.* Cambridge University Press, 2001.
24. Sewell, P. Global/local subtyping and capability inference for a distributed pi-calculus. In *Proc. ICALP'98* (1998), vol. 1442 of *LNCS*.
25. Wojciechowski, P. *Nomadic Pict: Language and Infrastructure Design for Mobile Computation*. PhD thesis, University of Cambridge, 2000.
26. Yoshida, N., Berger, M., and Honda, K. Strong Normalisation in the π-Calculus. In *Proc. LICS'01* (2001), IEEE, pp. 311–322. The full version to appear in *Journal of Information and Computation*.