A real-time extension to the π-calculus.

Technical Report 2009-557

Ernesto Posse
Applied Formal Methods Group – Software Technology Lab
School of Computing – Queen's University
Kingson, Ontario, Canada
eposse@cs.queensu.ca

May 21, 2009

Abstract

We propose an extension to the asynchronous π-calculus, the π_{klt}-calculus, for real-time systems where processes execute over dense time, and have the ability to delay actions, measure time, and impose timeouts. Other extensions include explicit data-structures and pattern-matching. We present the calculus syntax and operational semantics and establish sufficient conditions for *legitimacy* (a.k.a. *finite variability*), and introduce the notion of *timed-compositionality* and the associated *timed-congruence*. We also develop an Abstract Machine for this calculus and establish its soundness w.r.t. the given operational semantics.

1 Introduction

Process algebras [2, 32, 16] such as CSP [18, 32, 37], CCS [23, 24] and ACP [7] have become a well-established approach to modelling and reasoning about concurrent systems. They provide a concise and effective means to describe and analyze system behaviour.

Departing from their origins in Theoretical Computer Science, process algebras are finding applications in domains as diverse as Systems Biology [12], Web Services [34], Business Process Modelling [9] and Embedded Systems [10]. Nevertheless, the direct applicability of "classical" process algebras to these domains has been limited by their own capabilities as they do not take into account issues such as time, mobility and spatial distribution, demanded by these applications. This has sparked an interest in the definition of new process algebras that tackle these issues.

Consider the issue of time. In classical process algebras time is abstracted: advance in time means change of state, where state changes are the result of transitions or steps, and therefore time is discrete. Hence the notion of time and that of a computational step are interlocked: if a system stops, time stops. However this is not very realistic. Identifying the passage of time with computational steps does not result in a clear approach for the description of timing requirements, making these algebras unsuitable to model and reason about real-time properties of dynamic systems. This has lead to the definition of a number of process calculi which extend the existing models with an explicit notion of time and some appropriate constructs (e.g. [3, 4, 13, 26, 37, 39, 17]).

Similarly, the issue of mobility lead to the creation of the π-calculus [25]. In the π-calculus one can describe structural changes by modifying the network of communication channels between processes. However, as the classical process algebras, it does not have an explicit notion of time.

There have been a few process algebras which extend the π-calculus with an explicit notion of time. One of the best known is the stochastic π-calculus [30], but others have been proposed as well, such as the πRT-calculus [22], the timed-π [14], and the $TD\pi$-calculus [31]. These calculi allow the description of *delayable* or *durable* actions (usually probabilistic delays) with emphasis on modelling the duration of communication. This contrasts with the discrete-event world view [40] where one is more interested on the timing and scheduling of events than the duration of interaction between components.[1] The ability of delaying an arbitrary process (not just interaction) seems to provide a more flexible modelling approach. Furthermore, in the context of time-sensitive discrete-event systems, when waiting for input, the possible continuation states depend not only on timeouts but also on the amount of time the system spends waiting. Therefore the ability to measure the passage of time, and the ability to change behaviour if an event has not occurred within some time-constraint are fundamental operations. While some of the calculi mentioned include timeouts, none include a mechanism to observe the duration of actions within the language (*i.e.*, as a construct). Moreover, most of these algebras assume a discrete-time model which leads to an awkward and inefficient approximation of real-time behaviour. Furthermore, all of these calculi are based on the synchronous π-calculus, thus allow mixed-guarded choice which is difficult to implement in practice [27].

In this paper we introduce the π_{klt}-calculus, which extends the asynchronous π-calculus with basic real-time capabilities, namely, process delay, time observation and timeouts, and we look into two fundamental issues which have lacked attention in the context of timed π-calculi. The first concerns the issue of timed behavioural equivalence, and the second concerns progress in dense-time.

Perhaps the most comprehensive study of timed equivalences for timed π-calculi is found in [11] and [5]. These papers study extensions to the π-calculus in which actions have associated timers over discrete time. In [11] some forms of timed barbed bisimilarity are studied, while [5] presents asynchronous bisimilarities. Nevertheless, these equivalences are very stringent, as they require an exact match in the timing of the transitions of the processes being compared,

[1] It also contrasts with the so-called *synchrony hypothesis* of synchronous languages for reactive systems such as ESTEREL [8], LUSTRE [15] and SIGNAL [21], which assumes that actions, and in particular interactions are instantaneous.

for all future behaviours, and *as far in the future as the processes can run*. In the context of real-time systems one of the main concerns is to meet so-called *hard constraints*: to ensure a system's response within a certain amount of time T. In this context all late responses are failures and therefore we can restrict our equivalence checking to *equivalence up-to time T*. This in turn, leads to the issue of compositionality: when is it safe to replace one process by another in a timed context? In this paper we define such a behavioural equivalence up-to a given time, and show it to be compositional in our calculus.

The second major issue that we address concerns progress over dense-time and the property known as *legitimacy* (a.k.a. *finite variability*). A legitimate system is one that can perform only a finite number of actions in a finite amount of time. Hence, an illegitimate system may present divergent behaviour (in which time does not advance) or Zeno-behaviour (in which time advances but never beyond a certain point). From a practical perspective it is desirable to avoid illegitimate behaviours. While this issue has been explored in the context of Timed CSP [36], it has been neglected in the presence of mobility. We address this issue here and establish sufficient conditions for legitimacy of a process in our calculus. We contrast our approach with that followed by some timed process algebras such as Timed CSP which avoid this issue by requiring all recursions to have a strictly positive time guard, and thus, forbid illegitimate processes *by construction*. We argue nevertheless, that such an approach is overzealous, as it is possible to combine some processes that do not satisfy the time-guard requirement with legitimate processes to obtain composite legitimate systems.

Our calculus is intended to help bridge the gap between foundational process algebras and realistic languages and therefore it supports basic data structures and pattern matching. In addition to the issues sketched above, we develop an Abstract Machine for which we establish soundness w.r.t. the operational semantics. The inclusion of a real-time component as well as pattern-matching entail substantial differences with existing Abstract Machines for the π-calculus.

The contributions of this paper are: a process algebra that supports mobility and real-time with some higher-level features; a formal operational semantics, including a new timed observational equivalence, the notion of timed compositionality and timed congruence, and the establishment of sufficient conditions for legitimacy; a sound abstract machine to support execution with a working implementation.

The remainder of the paper is organized as follows: Section 2 introduces the core calculus, its syntax, its operational semantics and its basic theory. Section Section 3 develops an abstract machine for the core calculus. In section Section 4 we contrast this language and theory with some related work. Finally section Section 5 gives some concluding remarks. The apendices contain proofs and proof sketches.

P	$::=$	$\surd$	done
	$\mid$	$x!E$	trigger
	$\mid$	$\sum_{i \in I} x_i?F_i@y_i \to P_i$	listener
	$\mid$	$\nu x.P$	new/hide
	$\mid$	$\Delta E \to P$	delay/timer
	$\mid$	$P_1 \parallel P_2$	parallel composition
	$\mid$	$A(x_1, ..., x_n)$	process instantiation
E	$::=$	$n \mid \mathsf{true} \mid \mathsf{false} \mid$ "s" $\mid x \mid \langle E_1, ..., E_m \rangle$	
	$\mid$	$op\, E \mid E_1\, op\, E_2 \mid f(E_1, ..., E_m)$	
F	$::=$	$n \mid \mathsf{true} \mid \mathsf{false} \mid$ "s" $\mid x \mid \langle F_1, ..., F_m \rangle$	

Table 1: π_{klt} syntax

2 Timed, mobile processes: the π_{klt}-calculus

The π_{klt}-calculus is used to describe the behaviour of timed, concurrent, interacting processes. It provides operators to compose processes in parallel, to describe communication via events, or equivalently via message-passing over channels, to limit the scope of events, to delay processes and to observe the passage of time.

Unlike other basic process algebras the π_{klt}-calculus provides some higher-level constructs, in particular, it allows the use of complex expressions and data-structures in messages, and uses pattern-matching as a mechanism to extract information from data. The reason behind this design decision is to present this calculus as an attempt to close the gap between foundational process algebras and more realistic languages.

2.1 Syntax and informal description

Definition 1. (Syntax) The set $\mathcal{P}$ of π_{klt} **terms**, the set $\mathcal{E}$ of **expressions** and the set of **patterns** $\mathcal{F}$ are defined by the BNF in Table 1. Here P, P_i range over *process terms*, $x, y, ...$ range over the set of **(channel or variable) names**, A ranges over the set of **process names**, E ranges over *expressions*, and F ranges over *patterns*. Process definitions have the form: $A(x_1, ..., x_n) \stackrel{def}{=} P$. op ranges over standard arithmetic/relational operators, n ranges over floating point numbers, s ranges over strings, and f ranges over function names, with function definitions having the form: $f(x_1, ..., x_n) \stackrel{def}{=} E$.

The process $\surd$ simply terminates. The process $x!E$ triggers an event x and associates this event with the value of expression E. Alternatively, one can say that it sends the message E through channel x. This process performs communication by *unicasting*: if there are multiple listeners, only one of them accepts the message, and the choice is non-deterministic. Mobility is achieved in the same way as in the π-calculus since event/channel names are expressions,

P	$::=$	$\cdots$	
	$\mid$	$P_1; P_2$	sequential composition
	$\mid$	$(\sum_{i\in I} x_i?F_i@y_i \to P_i) \overset{E}{\triangleright} Q$	listener with timeout
	$\mid$	$\mathsf{match}\ E\ \mathsf{with}\ F_1 \to P_1 \mid \cdots \mid F_n \to P_n$	explicit pattern matching
	$\mid$	$\mathsf{if}\ E\ \mathsf{then}\ P\ \mathsf{else}\ Q$	conditional

Table 2: Some derived constructs

and so they can be sent as messages. The process $\sum_{i\in I} \beta_i \to P_i$ is a *listener*, consisting of a list of alternative input guarded processes $\beta_i \to P_i$. Each *input guard* β_i is of the form $x_i?F_i@y_i$, where x_i is an event/channel name, F_i is a pattern, and y_i is a variable. This process listens to all events (channels) x_i, and when x_i is triggered with a value v that matches the pattern F_i, the corresponding process P_i is executed with y_i bound to the amount of time the listener waited, and the alternatives are discarded[2]. A listener process represents, thus, a process in a state with external choice. Pattern-matching of inputs means that the input value must have the same "shape" as the pattern, and if successful, the free names in the pattern are bound to the corresponding values of the input[3]. For example, the value $\langle 3, \mathsf{true}, 7\rangle$ matches the pattern $\langle 3, x, y\rangle$ with the resulting binding $\{^{\mathsf{true}}/_x, {}^7/_y\}$. The scope of these bindings is the corresponding P_i. The suffixes F_i and $@y_i$ are optional: $x? \to P$ is equivalent to $x?y@z \to P$ for some fresh names y and z. Sometimes we write listeners in infix notation: $x_1?F_1@y_1 \to P_1 + \cdots + x_n?F_n@y_n \to P_n$. The process $\nu x.P$ hides the name x from the environment, so that it is private to P. Alternatively, $\nu x.P$ can be seen as the creation of a new name, i.e. a new event or channel, whose scope is P. We write $\nu x_1, x_2, ..., x_n.P$ for the process term $\nu x_1.\nu x_2....\nu x_n.P$. The process $\Delta E \to P$ is a *delay* or *timer*: it delays the execution of process P by an amount of time equal to the value of the expression E. The process $P_1 \parallel P_2$ is the parallel composition of P_1 and P_2. In its generalized form, we write $\Pi_{i\in\{1,\ldots,n\}} P_i$ for $P_1 \parallel P_2 \parallel \cdots \parallel P_n$. The process $A(y_1, ..., y_n)$ creates a new instance of a process defined by $A(x_1, ..., x_n) \overset{def}{=} P$, where the ports $x_1, ..., x_n$ are substituted in the body P by the events/channels (or values) $y_1, ..., y_n$.

There are several derived process terms such as the ones shown in Table 2. Here we only mention a few of them. These are all defined in terms of the operators above. See [28] for details. The process $P_1; P_2$ is the sequential composition of P_1 and P_2, this is, P_1 must terminate before beginning P_2. The process term $(\sum_{i\in I} \beta_i \to P_i) \overset{E}{\triangleright} Q$ represents a listener process with a timeout. If after an amount of time determined by the value of the expression E, none of the events have been triggered, control passes to Q. We define this term by combining a listener with a delay as follows:

[2] Note that to enable an input guard it is not enough for the event to be triggered: the event's value must match the guard's pattern as well.

[3] This is essentially the same as pattern-matching as found in languages such as ML or Haskell, albeit, without a type system, the structure matched is that given by tuples.

$(\Sigma_{i\in I}\beta_i \rightarrow P_i) \overset{E}{\triangleright} Q \overset{def}{=} \nu s.((\Sigma_{i\in I}\beta_i \rightarrow P_i + s? \rightarrow Q) \parallel \Delta E \rightarrow s!)$. The process $\mathsf{match}\ E\ \mathsf{with}\ F_1 \rightarrow P_1|\cdots|F_n \rightarrow P_n$ evaluates the expression E and attempts to match it with each pattern F_i. If a pattern F_i matches then the corresponding process P_i is executed. This construct is syntactic sugar for $\nu x.(x!E \parallel x?F_1 \rightarrow P_1 + \cdots + x?F_n \rightarrow P_n)$. Finally, the process $\mathsf{if}\ E\ \mathsf{then}\ P\ \mathsf{else}\ Q$ is shorthand for $\mathsf{match}\ E\ \mathsf{with}\ \mathsf{true} \rightarrow P|\mathsf{false} \rightarrow Q$.

As in the asynchronous π-calculus, an output with a continuation $x!E \rightarrow P$ is syntactic sugar for $x!E \parallel P$.

The suffix $@y_i$ of input guards is inherited from Timed CSP, but it is absent in all other timed variants of the π-calculus. This construct gives the calculus the power to measure the timing of events, and determine future behaviour accordingly.

2.2 Operational semantics

We now define formally the semantics. Let $\mathcal{N}$ denote the set of all possible names (including event names). Let $\mathcal{V}$ denote the universe of possible values including booleans, real numbers, strings, tuples of values and event (channel) names. We write $\mathrm{n}(v)$ for the set of all event names occurring in the value v. To simplify the presentation we assume we have a function $eval : \mathcal{E} \rightarrow \mathcal{V}$ that given an expression returns its value.[4] A sequence of names or values $x_1, ..., x_n$ is abbreviated as $\vec{x}$. A *substitution* is a function $\sigma : \mathcal{N} \rightarrow \mathcal{V}$. We write $\{^{v_1}/_{x_1}, ..., ^{v_n}/_{x_n}\}$ or $\{^{\vec{v}}/_{\vec{x}}\}$ for the substitution σ where $\sigma(x_1) = v_1$, ..., $\sigma(x_n) = v_n$ and $\sigma(z) = z$ for all $z \notin \{x_1, \ldots, x_n\}$. We write $\mathrm{dom}(\sigma)$ for $\{x_1, ..., x_n\}$. Furthermore, we write $\sigma \cup \sigma'$ for the substitution that extends σ with the bindings of σ'. Substitution is generalized to processes as a function $\sigma : \mathcal{P} \rightarrow \mathcal{P}$ in the natural way performing the necessary renamings to avoid capture of free names as usual. We write $P\sigma$ for $\sigma(P)$ denoting the process where all free occurrences of each x in σ have been (simultaneously) substituted by $\sigma(x)$. We denote $\mathcal{M}$ the set of all name substitutions. We write $P \equiv_\alpha Q$ if Q can be obtained from P by renaming of bound names. We denote $\mathrm{fn}(P)$ the set of free names of P (i.e. names not bound by either ν or an input guard). We use $\mathbb{R}_0^+$ to denote the non-negative reals.

Pattern-matching

Pattern matching is formally defined by a function *match* which takes as input a pattern, a datum (*i.e.*, a concrete value) and a substitution and returns either a new substitution which extends the original substitution with the appropriate bindings, or an empty substitution if the datum does not match the pattern. The substitution provided as input is used to ensure that all occurrences of a variable in a tuple match the same data.

[4] We do not need a name environment, as all expressions will be closed, since the appropriate substitutions of free names for values are performed before evaluation takes place.

$$
\frac{P \equiv_\alpha P'}{P \equiv P'} \qquad \nu x.\surd \equiv \surd \qquad \nu x.\nu y.P \equiv \nu y.\nu x.P
$$

$$
P \parallel \surd \equiv P \qquad P_1 \parallel P_2 \equiv P_2 \parallel P_1 \qquad P_1 \parallel (P_2 \parallel P_3) \equiv (P_1 \parallel P_2) \parallel P_3
$$

$$
\beta_1 \rightarrow P_1 + \beta_2 \rightarrow P_2 \equiv \beta_2 \rightarrow P_2 + \beta_1 \rightarrow P_1
$$

$$
\beta_1 \rightarrow P_1 + (\beta_2 \rightarrow P_2 + \beta_3 \rightarrow P_3) \equiv (\beta_1 \rightarrow P_1 + \beta_2 \rightarrow P_2) + \beta_3 \rightarrow P_3
$$

$$
\frac{x \notin fn(P)}{P \parallel \nu x.Q \equiv \nu x.(P \parallel Q)} \qquad \frac{A(x_1, ..., x_n) \stackrel{def}{=} P}{A(y_1, ..., y_n) \equiv P\{{}^{y_1}/_{x_1}, ..., {}^{y_n}/_{x_n}\}}
$$

$$
\Delta E \rightarrow (P_1 \parallel P_2) \equiv \Delta E \rightarrow P_1 \parallel \Delta E \rightarrow P_2
$$

Table 3: Axioms for structural congruence of processes.

Definition 2. (Pattern matching) Let $match : \mathcal{F} \times \mathcal{V} \times \mathcal{M} \rightarrow \mathcal{M}$ be a function defined as:

$$
match(F, v, \sigma) \stackrel{def}{=} \begin{cases} \sigma & \text{if } F = v \text{ and } v \in \mathcal{B} \\ & \text{ or } F \in \mathrm{dom}(\sigma) \text{ and } \sigma(F) = v \\ \sigma \cup \{{}^{v}/_{F}\} & \text{if } F \in \mathcal{N} \text{ and } F \notin \mathrm{dom}(\sigma) \\ \sigma_n & \text{if } F = \langle F_1, ..., F_n \rangle \text{ and } v = \langle v_1, ..., v_n \rangle, \\ & \text{ where } \forall i \in \{1, ..., n\},\ \sigma_i \stackrel{def}{=} match(F_i, v_i, \sigma_{i-1}) \\ & \text{ and } \sigma_0 \stackrel{def}{=} \sigma \\ \emptyset & \text{otherwise} \end{cases}
$$

where $\mathcal{B}$ is the set of basic constants (i.e. non-tuple values).

The first case returns the given substitution when the pattern is a basic constant identical to the datum or it is a variable whose associated value in the substitution is identical to the datum. The second case returns the given substitution extended with a new association, binding the pattern to the datum when the pattern is a variable not in the given substitution. The third case matches tuples: each item in the tuple datum is matched against the corresponding item in the pattern tuple from left to right. The result is the substitution yielded by the last match. The last case returns the empty substitution when there is a mismatch, since all other cases have failed.

Structural congruence

Any well-defined semantics must ensure that processes which are syntactically equivalent should behave in the same way. We now define such an equivalence relation, called *structural congruence*.

Definition 3. (Structural congruence over process terms) The relation $\equiv \subseteq \mathcal{P} \times \mathcal{P}$ is defined to be the smallest congruence[5] over $\mathcal{P}$ which satisfies the axioms shown in Table 3.

[5]Note that since $\equiv$ is defined to be a congruence relation, in addition to these axioms, it

Proposition 1. *Every process is structurally congruent to a process of the form:* $\nu x_1, ..., x_n.(P_1 \parallel \cdots \parallel P_k)$*, where each* P_i *is either a trigger, a listener, a delay or a process instantiation. We call such form "canonical normal form".*

Timed labelled-transitions systems

The meaning of a π_{klt} term is formally a *timed labelled transition system*, this is, a transition system in which we distinguish between transitions due to actions and evolution (passage of time). Formally, a *timed label transition system* or *TLTS* is a tuple $(\mathcal{S}, \mathcal{L}, \rightarrow, \rightsquigarrow)$ where $\mathcal{S}$ is a set of states, $\mathcal{L}$ is a set of *labels*, $\rightarrow \subseteq \mathcal{S} \times \mathcal{L} \times \mathcal{S}$ is a *transition relation* and $\rightsquigarrow \subseteq \mathcal{S} \times \mathbb{R}_0^+ \times \mathcal{S}$ is an *evolution relation*. A *rooted TLTS* is a tuple $(\mathcal{S}, s_0, \mathcal{L}, \rightarrow, \rightsquigarrow)$ where $(\mathcal{S}, \mathcal{L}, \rightarrow, \rightsquigarrow)$ is a TLTS and $s_0 \in \mathcal{S}$ is a distinguished *initial state*. We write $s \xrightarrow{a} s'$ for $(s, a, s') \in \rightarrow$ and $s \overset{d}{\rightsquigarrow} s'$ for $(s, d, s') \in \rightsquigarrow$. We write $s \xrightarrow{a}$ to mean that $\exists s' \in \mathcal{S}.\, s \xrightarrow{a} s'$. Typically the set $\mathcal{L}$ contains a distinguished element τ used to denote *unobservable* or *silent* actions. We write $s \overset{a}{\Longrightarrow} s'$ for $s(\xrightarrow{\tau})^* \xrightarrow{a} (\xrightarrow{\tau})^* s'$.

The notion of a TLTS characterizes a "two-dimensional" nature of time: transitions describe instantaneous actions and therefore a sequence of transitions describes an execution within the same instant of real-time, whereas evolution describes the passage of real-time itself.

Definition 4. (Process transitions and evolution) The meaning of a π_{klt} term P_0 is a rooted TLTS $\mathcal{T} = (\mathcal{P}, P_0, \mathcal{A}, \rightarrow, \rightsquigarrow)$ where $\mathcal{A}$ is the set of action labels described below, the relation $\rightarrow \subseteq \mathcal{P} \times \mathcal{A} \times \mathcal{P}$ is the smallest transition relation satisfying the inference rules in Table 4 and $\rightsquigarrow \subseteq \mathcal{P} \times \mathbb{R}_0^+ \times \mathcal{P}$ is the smallest evolution relation satisfying the inference rules in Table 5. The elements of $\mathcal{A}$ are actions of the form τ (silent action), $x!v$ (trigger), or $x?v$ (reception), where v is a constant or a name. We let α range over $\mathcal{A}$. We write $\mathrm{fn}(\alpha)$ (resp. $\mathrm{bn}(\alpha)$) for the set of free (resp. bound) names of the action α. These are defined by the following table:

α	τ	$x!v$	$x?v$
$\mathrm{fn}(\alpha)$	$\emptyset$	$\{x\} \cup \mathrm{n}(v)$	$\{x\}$
$\mathrm{bn}(\alpha)$	$\emptyset$	$\emptyset$	$\mathrm{n}(v)$

We impose an additional constraint on the TLTS to guarantee maximal progress: if $P \xrightarrow{\tau}$ then $P \overset{d}{\not\rightsquigarrow}$. This ensures that internal actions are urgent.

The rules for parallel composition seem to lack symmetry as they only describe the transitions of the first process. Nevertheless, this symmetry is recovered by using the congruence rule with the commutativity axiom for structural congruence. Note also that while there is no replication operator as in several presentations of the π-calculus, we allow recursive definitions, thus achieving the same result.

The (CHOICE) rule requires a match between the pattern and the value associated with the event provided by the environment. If the match is successful,

satisfies those of an equivalence relation and of a congruence: it is preserved by all contexts in the language.

$$(\textsc{trig})\ x!E \xrightarrow{x!eval(E)} \surd \qquad (\textsc{choice})\ \frac{\sigma = match(F_i, v, \emptyset) \qquad \sigma \neq \emptyset}{\sum_{i\in I} x_i?F_i@y_i \to P_i \xrightarrow{x_i?v} P_i(\sigma \cup \{{}^0/_{y_i}\})}$$

$$(\textsc{new})\ \frac{P \xrightarrow{\alpha} P' \qquad x \notin \mathrm{n}(\alpha)}{\nu x.P \xrightarrow{\alpha} \nu x.P'} \qquad (\textsc{delay})\ \frac{eval(E) = 0}{\Delta E \to P \xrightarrow{\tau} P}$$

$$(\textsc{par})\ \frac{P \xrightarrow{\alpha} P' \qquad \mathrm{bn}(\alpha) \cap \mathrm{fn}(Q) = \emptyset}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q} \qquad (\textsc{comm})\ \frac{P \xrightarrow{x!v} P' \qquad Q \xrightarrow{x?v} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$$

$$(\textsc{cngr})\ \frac{P \xrightarrow{\alpha} P' \quad P \equiv Q \quad P' \equiv Q'}{Q \xrightarrow{\alpha} Q'}$$

Table 4: Process transitions.

$$(\textsc{tidle})\ \surd \overset{d}{\rightsquigarrow} \surd \qquad (\textsc{ttrig})\ x!E \overset{d}{\rightsquigarrow} x!E$$

$$(\textsc{tchoice})\ \sum_{i\in I} x_i?F_i@y_i \to P_i \overset{d}{\rightsquigarrow} \sum_{i\in I} x_i?F_i@y_i \to P_i\{{}^{y_i+d}/_{y_i}\}$$

$$(\textsc{tnew})\ \frac{P \overset{d}{\rightsquigarrow} P'}{\nu x.P \overset{d}{\rightsquigarrow} \nu x.P'} \qquad (\textsc{tdelay})\ \frac{0 \leqslant d \leqslant eval(E)}{\Delta E \to P \overset{d}{\rightsquigarrow} \Delta(E-d) \to P}$$

$$(\textsc{tpar})\ \frac{P \overset{d}{\rightsquigarrow} P' \qquad Q \overset{d}{\rightsquigarrow} Q'}{P \parallel Q \overset{d}{\rightsquigarrow} P' \parallel Q'} \qquad (\textsc{tcngr})\ \frac{P \overset{d}{\rightsquigarrow} P' \quad P \equiv Q \quad P' \equiv Q'}{Q \overset{d}{\rightsquigarrow} Q'}$$

Table 5: Process evolution.

the resulting substitution is applied to the corresponding continuation, binding the elapsed time variable y_i to 0. This may seem a bit counterintuitive at first, but it is explained by the (TCHOICE) rule: when time advances d time units, the elapsed time variables are incremented by d so when a listener's transition is triggered the value of this variable will be the correct amount.

An example

To illustrate the semantics, we look at a short example. Consider the term $x?\langle 4, z\rangle@e \to \Delta(5 - e) \to z!e$. This term blocks waiting from an event x to occur, with a pair $\langle 4, z\rangle$ as data, and then responds 5 time units after it began waiting, by triggering the received event z with the time elapsed between the it started waiting and the occurrence of x. Its TLTS will contain the following execution:

$$
\begin{array}{ll}
\multicolumn{2}{l}{x?\langle 4, z\rangle@e \to \Delta(5 - e) \to z!e} \\
\quad \overset{3.2}{\rightsquigarrow} & x?\langle 4, z\rangle@e \to \Delta(5 - (e + 3.2)) \to z!(e + 3.2) \\
\quad \xrightarrow{x?\langle 4,y\rangle} & \Delta(5 - (0 + 3.2)) \to y!(0 + 3.2) \\
\quad \overset{1.8}{\rightsquigarrow} & \Delta 0 \to y!(0 + 3.2) \\
\quad \xrightarrow{\tau} & y!(0 + 3.2) \\
\quad \xrightarrow{y!3.2} & \surd
\end{array}
$$

Some properties

The operational semantics entails the following elementary properties.

Proposition 2.

1. *(Zero time advance) For any $P \in \mathcal{P}$, $P \overset{0}{\rightsquigarrow} P$.*
2. *(Time determinacy) For any $P, P', P'' \in \mathcal{P}$, $d \in \mathbb{R}_0^+$, if $P \overset{d}{\rightsquigarrow} P'$ and $P \overset{d}{\rightsquigarrow} P''$, then $P' \equiv P''$.*
3. *(Time continuity) For any $P, P' \in \mathcal{P}$, $d, d' \in \mathbb{R}_0^+$, $P \overset{d+d'}{\rightsquigarrow} P'$ if and only if there is a P'' such that $P \overset{d}{\rightsquigarrow} P''$ and $P'' \overset{d'}{\rightsquigarrow} P'$.*
4. *(Persistency of offers) For any $P, P' \in \mathcal{P}$, $d \in \mathbb{R}_0^+$, and any $\alpha \neq \tau$, if $P \xrightarrow{\alpha}$ and $P \overset{d}{\rightsquigarrow} P'$ then $P' \xrightarrow{\alpha}$.*

2.3 Timed equivalence

Several timed variants of bisimilarity for timed π-calculi have been studied in [11], [5] and [6]. The calculi studied by these, extend the π-calculus with discrete time, rather than dense time. Furthermore, the variants of bisimilarity presented, identify processes based on their entire future behaviour. But this is too stringent, and as explained in the introduction, all behaviours that violate hard response-time constraints of real-time systems are considered to be failures. Therefore we can weaken our comparison criteria to behaviours up to a given deadline.

Consider the following processes:

$$\begin{array}{rcl} A & \stackrel{def}{=} & (a? \to P) \overset{3}{\triangleright} Q \\ B & \stackrel{def}{=} & (a? \to P) \overset{5}{\triangleright} Q \end{array}$$

These systems cannot be identified under standard notions of bisimilarity. To see these, recall the definition of timeout. We can expand these to

$$\begin{array}{rcl} A & = & \nu s.((a? \to P + s? \to Q) \parallel \Delta 3 \to s!) \\ B & = & \nu s.((a? \to P + s? \to Q) \parallel \Delta 5 \to s!) \end{array}$$

Then we see that A has the following execution:

$$\begin{array}{rcl} A & \stackrel{3}{\rightsquigarrow} & \nu s.((a? \to P + s? \to Q) \parallel \Delta 0 \to s!) \\ & \stackrel{\tau}{\to} & \nu s.((a? \to P + s? \to Q) \parallel s!) \\ & \stackrel{\tau}{\to} & Q \end{array}$$

but this cannot be matched by B:

$$B \stackrel{3}{\rightsquigarrow} \nu s.((a? \to P + s? \to Q) \parallel \Delta 2 \to s!) \stackrel{\tau}{\not\to}$$

Hence, A and B cannot be not identified by any of the existing timed bisimilarities for timed π-calculi, such as those in [11] or [5] or bisimilarities that match evolution directly (*i.e.*, $P \stackrel{d}{\rightsquigarrow} P'$ implies $Q \stackrel{d}{\rightsquigarrow} Q'$ with P' bisimilar to Q'). Either way, at time 3 the process A has an interaction that cannot be matched by B. Nevertheless, before time 3, both A and B have exactly the same transitions ($a?$) and evolutions. But if we have a hard constraint requiring interaction before 3 time units, we don't care about their behaviour beyond time 3, and so it makes sense to identify the two processes up-to time 3.

In [36], Schneider introduced a notion of *timed bisimilarity up-to time* T to compare the behaviour of Timed CSP processes. A good notion of behavioural/observational equivalence is one which satisfies the property that whenever two processes are identified, no observer or context can distinguish between them. Such a property is satisfied by an equivalence relation which is preserved by all combinators or operators of the language, in other words, by a congruence relation (compositionality). Unfortunately Schneider's equivalence is not a congruence in the context of timed π-calculi, because, as ground bisimilarity, it is not preserved by listeners (input). In the theory of the π-calculus several alternative definitions of bisimilarity have been explored to ensure compositionality. Sangiorgi's *open bisimilarity* [35] has the desired feature: it is a congruence for all π-calculus operators. This suggests the following equivalence for dense-time π-calculi which combines Schneider's timed bisimilarity with Sangiorgi's open bisimilarity.

Definition 5. (Open timed-bisimulation) Let $\mathcal{S}$ be a set of terms in some language for which there is a notion of substitution defined, where substitutions are functions $\sigma : \mathcal{S} \to \mathcal{S}$. Let $(\mathcal{S}, \mathcal{L}, \to, \leadsto)$ a TLTS over $\mathcal{S}$. A relation $R \subseteq \mathcal{S} \times \mathbb{R}_0^+ \times \mathcal{S}$, is called an ***open timed-bisimulation*** if for all $t \in \mathbb{R}_0^+$, whenever $(P, t, Q) \in R$ then, for any substitution $\sigma : \mathcal{S} \to \mathcal{S}$ and any $d \in \mathbb{R}_0^+$ such that $d < t$:

1. $\forall \alpha \in \mathcal{L}. \forall P' \in \mathcal{S}. P\sigma \xrightarrow{\alpha} P' \Rightarrow \exists Q' \in \mathcal{S}. Q\sigma \xrightarrow{\alpha} Q' \wedge (P', t, Q') \in R$
2. $\forall \alpha \in \mathcal{L}. \forall Q' \in \mathcal{S}. Q\sigma \xrightarrow{\alpha} Q' \Rightarrow \exists P' \in \mathcal{S}. P\sigma \xrightarrow{\alpha} P' \wedge (P', t, Q') \in R$
3. $\forall P' \in \mathcal{S}. P\sigma \overset{d}{\leadsto} P' \Rightarrow \exists Q' \in \mathcal{S}. Q\sigma \overset{d}{\leadsto} Q' \wedge (P', t-d, Q') \in R$
4. $\forall Q' \in \mathcal{S}. Q\sigma \overset{d}{\leadsto} Q' \Rightarrow \exists P' \in \mathcal{S}. P\sigma \overset{d}{\leadsto} P' \wedge (P', t-d, Q') \in R$

Let

$$\Leftrightarrow \overset{def}{=} \{(P, u, Q) \in \mathcal{S} \times \mathbb{R}_0^+ \times \mathcal{S} \mid \exists R.\, R \text{ is an open time-bisimulation} \,\&\, (P, u, Q) \in R\}$$

We write $P \Leftrightarrow_u Q$ for $(P, u, Q) \in \Leftrightarrow$ and say that P and Q are ***open timed-bisimilar up to*** u. If $P \Leftrightarrow_u Q$, we call u the ***time-limit of the bisimulation***. For any given $t \in \mathbb{R}_0^+$, $\Leftrightarrow_t \overset{def}{=} \{(P, Q) \in \mathcal{S} \times \mathcal{S} \mid (P, t, Q) \in \Leftrightarrow\}$.

This definition is applicable to any TLTS for which states are terms in a language with an appropriate notion of substitution. In our context we will assume that $\mathcal{S}$ is $\mathcal{P}$, the set of π_{klt} process terms, and the TLTS is that given in definition 4.

First notice that, as Sangiorgi's open bisimilarity, this definition quantifies over substitutions σ. This is to ensure listener processes are identified whenever possible. Also notice, that the first two items are the same as for standard (untimed) open bisimulation, if we ignore t. It is the second two items which make a difference in the definition.

Now, let us revisit the processes $A \overset{def}{=} (a? \to P) \overset{3}{\triangleright} Q$ and $B \overset{def}{=} (a? \to P) \overset{5}{\triangleright} Q$. We have that $A \Leftrightarrow_3 B$, as the two processes have the same transitions and evolution up to any time strictly less than 3. Figure 1 on page 13 illustrates this: any state at time $d < 3$, both A and B have an $a?$ transition which leads them to P, and any process P is bisimilar with itself up to any time.

Open timed-bisimulation satisfies the following properties for any TLTS.

Proposition 3. *For any TLTS $M = (\mathcal{S}, \mathcal{L}, \to, \leadsto)$,*

1. *For any $t \in \mathbb{R}_0^+$, $\Leftrightarrow_t$ is an equivalence relation.*
2. *$\Leftrightarrow$ is an open timed-bisimulation.*
3. *$\Leftrightarrow$ is the largest open timed-bisimulation.*

Furthermore, we have the following:

$$\begin{tikzcd}
A \arrow[r, dotted, no head, "R_3"] \arrow[d, squiggly, "d"'] & B \arrow[d, squiggly, "d"'] \\
(a? \to P) \overset{3-d}{\triangleright} Q \arrow[r, dotted, no head, "R_{3-d}"] \arrow[d, "a?"'] & (a? \to P) \overset{5-d}{\triangleright} Q \arrow[d, "a?"'] \\
P \arrow[r, dotted, no head, "R_{3-d}"] & P
\end{tikzcd}$$

Figure 1: Time bisimulation. Here R_t denotes $\{(P,Q) \,|\, (P,t,Q) \in R\}$.

Proposition 4. *For any TLTS $M = (\mathcal{S}, \mathcal{L}, \to, \rightsquigarrow)$, any $t, u \in \mathbb{R}_0^+$, and any $P, P', P'' \in \mathcal{S}$:*

1. *If $P \mathrel{\underline{\leftrightarrow}}_t P'$ then for any $u \leqslant t$, $P \mathrel{\underline{\leftrightarrow}}_u P'$*
2. *If $P \mathrel{\underline{\leftrightarrow}}_t P'$ and $P' \mathrel{\underline{\leftrightarrow}}_u P''$ then $P \mathrel{\underline{\leftrightarrow}}_{\min\{t,u\}} P''$*

Timed compositionality

Now we focus on the compositionality properties of open timed-bisimilarity. First, we have that it is closed under substitutions:

Lemma 1. *For any substitution σ, and any $t \in \mathbb{R}_0^+$, if $P \mathrel{\underline{\leftrightarrow}}_t Q$ then $P\sigma \mathrel{\underline{\leftrightarrow}}_t Q\sigma$.*

As mentioned above, a good observational equivalence should be a congruence. However, as we have argued, we are only interested in equivalence up-to some time T, and therefore we only care about the observable behaviour up-to time T, which means that the equivalence must be preserved by our operators only up to that time. Hence we are after a notion of *timed-compositionality*, characterized by a *timed-congruence*, which we now formally define:

Definition 6. (Timed-congruence) Given some set $\mathcal{S}$, and a ternary relation $R \subseteq \mathcal{S} \times \mathbb{R}_0^+ \times \mathcal{S}$, we define R's t-***projection*** to be the binary relation $R_t \stackrel{def}{=} \{(p,q) \in \mathcal{S} \times \mathcal{S} \,|\, (p,t,q) \in R\}$. R is called a t-***congruence*** iff R_t is a congruence. R is called a ***timed-congruence*** if it is a t-congruence for all $t \in \mathbb{R}_0^+$. R is called a ***timed-congruence up-to*** u iff it is a t-congruence for all $0 \leqslant t \leqslant u$.

Open-timed bisimilarity satisfies the stronger property stated by the following, obtained using Lemma 1:

Lemma 2. *For any $P, P', Q_1, ..., Q_n \in \mathcal{P}$, and any $t \in \mathbb{R}_0^+$, if $P \mathrel{\underline{\leftrightarrow}}_t P'$ then:*

1. *$\Delta E \to P \mathrel{\underline{\leftrightarrow}}_{t+e} \Delta E \to P'$ where $e = eval(E)$*
2. *$\nu x.P \mathrel{\underline{\leftrightarrow}}_t \nu x.P'$*

3. $P \parallel Q \Leftrightarrow_t P' \parallel Q$
4. $x?F@y \to P + \sum_{i=1}^{n} \beta_i \to Q_i \Leftrightarrow_t x?F@y \to P' + \sum_{i=1}^{n} \beta_i \to Q_i$ *where each* β_i is of the form $x_i?F_i@y_i$.

The immediate consequence, which follows from Proposition 4 and Lemma 2 is timed compositionality:

Theorem 1. $\Leftrightarrow_t$ *is a timed-congruence up-to* t *and* $\Leftrightarrow$ *is a timed-congruence.*

We also obtain the following properties as a consequence of Lemma 2, which guarantees equivalence up to the least upper bound of the pairwise time-equivalences:

Corollary 1. *For any families of terms* $\{P_i \in \mathcal{P}\}_{i \in I}$ *and* $\{Q_i \in \mathcal{P}\}_{i \in I}$*, if for each* $i \in I$*,* $P_i \Leftrightarrow_{t_i} Q_i$ *then*

1. $\prod_{i \in I} P_i \Leftrightarrow_{\min\{t_i | i \in I\}} \prod_{i \in I} Q_i$
2. $\sum_{i \in I} x_i?F_i@y_i \to P_i \Leftrightarrow_{\min\{t_i | i \in I\}} \sum_{i \in I} x_i?F_i@y_i \to Q_i$

2.4 Legitimacy

We will now turn our attention to another fundamental issue in the description of timed systems: legitimacy, a.k.a. finite variability. Consider the following process:

$$L(x) \stackrel{def}{=} \nu u.((u? \to L(x)) \parallel u!)$$

This process triggers a local event u which enables it to recursively call itself. In other words, it performs an internal action (the interaction on u) and loops indefinitely. This means that it has an infinite execution of internal actions:

$$\gamma_0 = L(x) \xrightarrow{\tau} L(x) \xrightarrow{\tau} L(x) \xrightarrow{\tau} \cdots$$

This kind of execution is called *instantaneous divergence*, since the system diverges at a single instant in time (it always has τ transitions, thus never reaching a stable state and forever proscribing any time advance).

There are systems that, even though they are not instantaneous divergent, do exhibit undesirable behaviour. Consider the following:

$$\begin{aligned} Z(n) &\stackrel{def}{=} \nu u.((u? \to Z'(n)) \parallel u!) \\ Z'(n) &\stackrel{def}{=} \Delta 2^{-n} \to Z(n+1) \end{aligned}$$

This also produces an infinite execution:

$$\gamma_1 = Z(1) \xrightarrow{\tau} Z'(1) \stackrel{1/2}{\rightsquigarrow} Z(2) \xrightarrow{\tau} Z'(2) \stackrel{1/4}{\rightsquigarrow} Z(3) \xrightarrow{\tau} Z'(3) \stackrel{1/8}{\rightsquigarrow} \cdots$$

While in this execution time advances, it never progresses beyond time 2. This sort of execution is known as *Zeno behaviour*.

Instantaneous divergence and Zeno behaviour have something in common: they represent behaviours of a process performing an infinite number of actions in a finite amount of time. Those behaviours are not realistic, and we should avoid them. This gives rise to a fundamental question: how can we know if a system will behave in a way that time always progresses? In other words, we want to know what are the sufficient conditions for a system to have properly defined timed behaviour.

A system which can only perform a finite amount of actions in a finite amount of time is called *legitimate*. Some timed process algebras such as existing timed π-calculi of [11], [5] and [6], avoid the issue by restricting the time base to a discrete domain. This eliminates Zeno-behaviours but not divergent behaviours. Others, which accept dense-time domains, such as Timed CSP avoid the issue by requiring all recursions to have a strictly positive time guard, and thus, forbid *by construction*, illegitimate processes. We argue nevertheless, that such an approach disallows perfectly useful processes, as it will be illustrated below. It is preferable to be aware under what conditions we can guarantee legitimacy, without sacrificing expressiveness.

Now we formalize the notion of legitimacy and establish sufficient conditions for legitimacy.

Legitimate executions

Because of the properties of Proposition 2, every execution can be written in the following form:

$$\gamma = P_0 \overset{d_0}{\rightsquigarrow} P_0' \xrightarrow{\alpha_0} P_1 \overset{d_1}{\rightsquigarrow} P_1' \xrightarrow{\alpha_1} P_2 \overset{d_2}{\rightsquigarrow} P_2' \xrightarrow{\alpha_2} \cdots$$

This allows to define the *length* of the execution as $\mathrm{len}(\gamma) \overset{def}{=} n$ where n is the number of transitions, the execution's *duration* as $\mathrm{dur}(\gamma) \overset{def}{=} \sum_{i=0}^{\mathrm{len}(\gamma)} d_i$ and its *action trace* as $\mathrm{atr}(\gamma) \overset{def}{=} \langle \alpha_0, \alpha_1, \alpha_2, ... \rangle$.

Definition 7. (Legitimacy) A ***legitimate execution*** is an execution γ such that $\mathrm{len}(\gamma) = \infty$ implies that $\mathrm{dur}(\gamma) = \infty$. A ***legitimate process*** is a process such that all its executions are legitimate.

This definition implies that every finite execution is legitimate so illegitimate behaviours can arise only from infinite executions. In our calculus, the only way to obtain an infinite execution is via recursion. Hence we have to look at the delays between recursive invocations.

Well-timed definitions

Suppose that a process P may invoke some definition $A(\vec{x}) \overset{def}{=} Q$. Since this invocation may occur anywhere inside P, some amount of time may pass from the beginning of P until A is actually invoked. This amount of time may depend on P itself or its environment, if the invocation is inside a listener. We define a function md_A which, for a process P, gives the minimum delay between the start of P and any invocation of A in P.

Definition 8. (Minimum delay for invocation) Let $A(\vec{x}) \stackrel{def}{=} Q$ be some process definition. Define $md_A : \mathcal{P} \to \mathbb{R}_0^+ \cup \{\infty\}$ as follows:

$$
\begin{array}{rcl}
md_A(\surd) & \stackrel{def}{=} & \infty \\
md_A(x!E) & \stackrel{def}{=} & \infty \\
md_A(\Delta E \to P) & \stackrel{def}{=} & eval(E) + md_A(P) \\
md_A(\nu x.P) & \stackrel{def}{=} & md_A(P) \\
md_A(P_1 \parallel P_2) & \stackrel{def}{=} & \min\{md_A(P_1), md_A(P_2)\} \\
md_A(\Sigma_{i\in I} x_i?F_i@y_i \to P_i) & \stackrel{def}{=} & \min\{md_A(P_i) \mid i \in I\} \\
md_A(A(\vec{y})) & \stackrel{def}{=} & 0 \\
md_A(B(\vec{y})) & \stackrel{def}{=} & md_A(Q'\{\vec{y}/\vec{x}\}) \\
 & & \text{where } B \neq A \text{ and } B(\vec{x}) \stackrel{def}{=} Q'
\end{array}
$$

We say that P is *t-guarded for A*, if $t \leqslant md_A(P)$.

Using this notion we can define what it means for a process definition to have proper timed behaviour.

Definition 9. (Well-timed definition) A definition $A(\vec{x}) \stackrel{def}{=} P$ with $\vec{x} = x_1, ..., x_n$, is said to be ***well-timed*** if there is a $b \in \mathbb{R}_0^+$ such that $b > 0$ and for all $\vec{v} = v_1, ..., v_n$, where each $v_i \in \mathcal{V}$, $md_A(P\{\vec{v}/\vec{x}\}) \geqslant b$.

This definition states that there must be a strictly positive lower bound on the minimal delay before any possible instantiation of A in its body. Notice that this includes any indirect invocation of A, by definition of minimal delay.

We will need the following, which states that transitions and evolutions are preserved and reflected by substitution.

Proposition 5. *For any $P, P' \in \mathcal{P}$, $d \in \mathbb{R}_0^+$, $\alpha \in \mathcal{A}$, and any substitution σ,*

1. *if $P \xrightarrow{\alpha} P'$ then $P\sigma \xrightarrow{\sigma(\alpha)} P'\sigma$*
2. *if $P \stackrel{d}{\leadsto} P'$ then $P\sigma \stackrel{d}{\leadsto} P'\sigma$*
3. *if $P\sigma \xrightarrow{\alpha} P'$ then there is a α' and a Q such that $P \xrightarrow{\alpha'} Q$ with $\alpha = \sigma(\alpha')$ and $P' = Q\sigma$.*
4. *if $P\sigma \stackrel{d}{\leadsto} P'$ then there is a Q such that $P \stackrel{d}{\leadsto} Q$ and $P' = Q\sigma$*

This proposition allows us to conclude that legitimacy is preserved by substitutions.

Lemma 3. *Let $P \in \mathcal{P}$, and σ any substitution. If P is legitimate, so is $P\sigma$.*

This leads us to the main result:

Theorem 2. ***(Sufficient conditions for legitimacy)*** *Let D be a finite set of process definitions and P a process which invokes only definitions in D. If all definitions in D are well-timed then P is legitimate.*

Examples

Let us revisit the examples at the beginning of this section. First, we saw that the definition L leads to a divergent execution γ_0 which is illegitimate, since $\text{len}(\gamma_0) = \infty$ but $\text{dur}(\gamma_0) = 0$. In this case, we have that the problem is that the definition of L is not well-timed: the minimum delay before any recursive invocation of L is $md_L(P\{^y/_x\}) = 0$ for any y, where $P = \nu u.((u? \to L(x)) \parallel u!)$, and therefore there is no $b > 0$ such that $md_L(P\{^y/_x\}) \geqslant b$ for any y.

In the second example, we saw how the definition of Z led to an infinite execution γ_1 where we have that $\text{len}(\gamma_1) = \infty$ but $\text{dur}(\gamma_1) = \Sigma_{i=0}^{\infty} {}^1/_{2^i} = 2$, so it is illegitimate. In this case, we can check that $md_Z(P\{^y/_n\}) = 2^{-y}$ for any y where $P = \nu u.((u? \to Z'(n)) \parallel u!)$. But there is no $b > 0$ such that $b \leqslant 2^{-y}$ for all y. Hence, the definition of Z is not well-timed.

On the other hand, consider the following variation of Z:

$$\begin{aligned} B(n) &\stackrel{def}{=} \nu u.((u? \to B'(n)) \parallel u!) \\ B'(n) &\stackrel{def}{=} \Delta(1 + 2^{-n}) \to B(n+1) \end{aligned}$$

In this case, we have the execution

$$\gamma_2 = B(1) \xrightarrow{\tau} B'(1) \overset{1+1/2}{\rightsquigarrow} B(2) \xrightarrow{\tau} B'(2) \overset{1+1/4}{\rightsquigarrow} B(3) \xrightarrow{\tau} B'(3) \cdots$$

So $\text{dur}(\gamma_2) = \Sigma_{i=0}^{\infty}(1 + {}^1/_{2^i}) = \infty$, and therefore it is a legitimate execution. This is because $md_Z(P\{^y/_n\}) = 1 + 2^{-y}$ for any y where $P = \nu u.((u? \to B'(n)) \parallel u!)$. But there is a $b > 0$ such that $b \leqslant 1 + 2^{-y}$ for all y, namely $b = 1$, and so B has a well-timed definition.

Legitimacy and non-well-timed definitions

We remarked that some timed process algebras such as Timed CSP require all recursions to have a strictly positive time guard, and thus, forbid *by construction*, illegitimate processes. But such requirement is a bit excessive. Consider for example the following process definition:

$$C(a,b) \stackrel{def}{=} a? \to b! \to C(a,b)$$

It is easy to establish that $md_C(P\{^{a'}/_a, {}^{b'}/_b\}) = 0$ for any a', b' where $P = a? \to b! \to C(a,b)$. Therefore this definition is not well-timed. But should we forbid such definition? Consider the process $D(a,b) \stackrel{def}{=} a! \to b? \to D(a,b)$. If we combine an instance of C with an instance of D, as in $E = \nu a,b.(C(a,b) \parallel D(a,b))$ we obtain a divergent execution

$$E \xrightarrow{\tau} \nu a,b.(b! \to C(a,b) \parallel b? \to D(a,b)) \xrightarrow{\tau} E \xrightarrow{\tau} \cdots$$

But what if we consider the following alternative context: $D'(a,b) \stackrel{def}{=} a! \to \Delta 0.1 \to b? \to D'(a,b)$? Then the process $E' = \nu a,b.(C(a,b) \parallel D'(a,b))$ has a legitimate execution:

$$\begin{array}{lll} E' & \xrightarrow{\tau} & \nu a, b.(b! \to C(a,b) \parallel \Delta 0.1 \to b? \to D'(a,b)) \\ & \overset{0.1}{\rightsquigarrow} & \nu a, b.(b! \to C(a,b) \parallel \Delta 0 \to b? \to D'(a,b)) \\ & \xrightarrow{\tau} & \nu a, b.(b! \to C(a,b) \parallel b? \to D'(a,b)) \\ & \xrightarrow{\tau} & E' \\ & \ldots & \end{array}$$

This does not contradict Theorem 2, because the Theorem establishes sufficient, but not necessary conditions for legitimacy. Hence, some process definitions like C, while not well-timed, may be combined with other processes to obtain legitimate systems, and therefore they should not be dismissed by construction.

Legitimacy and time-bisimulation

We introduced the notion of open time-bisimilarity as a means to compare the behaviour of processes up to a given time. How is open time-bisimilarity related to legitimacy? Naturally, if two processes are equivalent, their evolution over time must be equivalent, and therefore, they should both be legitimate or both illegitimate. We prove formally this statement. First, we need the following Lemma.

Lemma 4. *Let $P, Q \in \mathcal{P}$. If $P \leftrightharpoons_t Q$ and γ_P is an execution beginning with P such that $\mathrm{dur}(\gamma_P) < t$, then there is an execution γ_Q, starting at Q, such that $\mathrm{atr}(\gamma_P) = \mathrm{atr}(\gamma_Q)$, $\mathrm{len}(\gamma_P) = \mathrm{len}(\gamma_Q)$ and $\mathrm{dur}(\gamma_P) = \mathrm{dur}(\gamma_Q)$.*

From this, the result follows: open-time-bisimilarity preserves legitimacy (and illegitimacy.)

Theorem 3. *Let $P, Q \in \mathcal{P}$. If $P \leftrightharpoons_t Q$ for all $t \in \mathbb{R}_0^+$ then P is legitimate if and only if Q is legitimate.*

3 An Abstract Machine for the π_{klt}-calculus

Process algebras can be used strictly as specification languages, but when we see them as modelling or programming languages we become interested in how the "models" or programs are to be executed. An operational semantics for such a language given as a set of Plotkin-style SOS rules, describes execution, but at a very high-level of abstraction. It is not always clear how a set of inductively defined rules defining an LTS, or even a non-labelled reduction system, is to be executed by a machine. A commonly accepted approach is to define an *Abstract Machine* for the language which captures the operational semantics prescribed by the inference rules, and which describes computation at an appropriate level of abstraction such that it can be easily implemented. In this section we describe such Abstract Machine for the π_{klt} calculus and proof it is sound w.r.t. the operational semantics.

3.1 Abstract Machine specification

When ignoring time, our Abstract Machine is quite similar to Turner's Abstract Machine for the π-calculus [38]. But unlike Turner's we have to take into account evolution over real-time. In spite of supporting dense time, our calculus can be classified as a discrete-event formalism [40], since changes of state occur only at specific instants. Therefore, we can simulate the execution of a process by means of event-scheduling. The key idea is to treat each π_{klt} term as a *simulation event* to be executed by an event-scheduler (and not to be confused with a *communication event* in the language itself). Such event scheduler forms the heart of our Abstract Machine.

The global queue

The event-scheduler contains a queue of simulation events (terms) to be executed, but rather than store them all in a single linear queue, we divide them into *time-slots*, i.e., sequences of all simulation events to be executed at a given instant in time. Hence the global event queue is a time-ordered queue of time-slots, each of which is a queue of terms. We describe the operation of our Abstract Machine by showing how it evolves in these two "dimensions" of time: the "vertical dimension" which corresponds to the execution of all terms in a single time-slot, and the "horizontal dimension", which corresponds to the advance in time, i.e. the progress of the global queue.

Definition 10. (Global queue) The set $\mathcal{R}$ of ***global queue states***, ranging over R is defined by the following BNF, where T ranges over the set $\mathcal{T}$ of ***time-slots***:

$$\begin{array}{lcl} R & ::= & (t_1, T_1) \cdot (t_2, T_2) \cdot \cdots \cdot (t_n, T_n) \quad | \quad \langle\rangle \\ T & ::= & P_1 :: P_2 :: \cdots :: P_m \quad | \quad \epsilon \end{array}$$

where each $P_i \in \mathcal{P}$, each $t_i \in \mathbb{R}_0^+$, and for each $i \geqslant 1$, $t_i < t_{i+1}$.

Event observers and the heap

Processes interact through (communication) events (a.k.a. channels) introduced by the ν operator. Furthermore, multiple processes can trigger and/or listen to the same event. Hence each such event can affect multiple processes. So there can be multiple attempts to interact through it, and therefore we need to keep track of each of these requests in an *observer set*[6], containing *observers, i.e.,* requests to either trigger the event or listen to it[7]. We also define the *heap*, which is essentially a map associating each event name to its observer set.

Definition 11. (Event observers and heap) The set of ***event observers***, ranged over by O, ***observer sets***, ranged over by Q and the set $\mathcal{H}$ of ***heaps***,

[6] Analogous to "channel queues" in Turner's terminology.

[7] In Turner's terminology, triggers are "writers" and listeners are "readers".

ranged over by H, are defined by the following BNF:

$$\begin{array}{lcl} O & ::= & !v \quad | \quad ?(F, y, P, t, c) \\ Q & ::= & \{O_1, O_2, \cdots, O_n\} \quad | \quad \emptyset \\ H & ::= & x_1 \mapsto Q_1, x_2 \mapsto Q_2, ..., x_m \mapsto Q_m \quad | \quad \epsilon \end{array}$$

where, as usual, $v \in \mathcal{V}$, $F \in \mathcal{F}$, $P \in \mathcal{P}$, and $t \in \mathbb{R}_0^+$.

We denote $H\{x \mapsto Q\}$ for the heap where the entry for x is updated to Q. Formally:

$$\begin{array}{rcll} \epsilon\{x \mapsto Q\} & \stackrel{def}{=} & x \mapsto Q & \\ (H, x \mapsto Q)\{x \mapsto Q'\} & \stackrel{def}{=} & H, x \mapsto Q' & \\ (H, x' \mapsto Q)\{x \mapsto Q'\} & \stackrel{def}{=} & H\{x \mapsto Q'\}, x' \mapsto Q & \quad \text{where } x \neq x' \end{array}$$

We extend this notation to indexed sets of names: $H\{x_i \mapsto Q_i\}_{i \in I}$ stands for $H\{x_1 \mapsto Q_1\} \cdots \{x_n \mapsto Q_n\}$ for $I = \{1, ..., n\}$.

Observers of the form $!v$ are *output observers*, and denote an attempt to trigger the given event with a value v. Observers of the form $?(F, y, P, t, c)$ denote *input observers*, with a pattern F to match, elapsed-time variable y, body P to execute when the event is triggered with a matching value and which start listening at time t. This tag t is necessary in order to assign the correct elapsed time to y once the event is triggered. Such time-stamp is not present in Turner's machine, since his is "time agnostic". The last item, c is a tag used to identify the original π_{klt} listener, so that each branch of a listener $\sum_{i \in I} x_i?F_i@y_i \to P_i$ will have an observer $?(F_i, y_i, P_i, t, c)$ sharing the same identifier c. This is also absent from Turner's machine, since he does not implement the choice operator.

We assume the standard set theoretical operations are available for observer sets: *e.g.*, $Q \cup \{O\}$ denotes the observer set that adds O to Q, $Q \backslash O$ is the set that results from removing O from Q, $O \in Q$ tests for membership, etc.

Note also, that unlike Turner's machine, any given non-empty observer set can contain both inputs and outputs simultaneously, because it is possible that the value associated with a triggered event does not match any of the patterns of the available input observers and thus the corresponding output observer must be suspended with the existing inputs on the same observer set. Hence the following auxiliary definitions will be useful to extract the relevant observers from a set.

Definition 12. Given an observer set Q, we denote:

$$\begin{array}{rcl} \mathit{inputs}(Q) & \stackrel{def}{=} & \{O \in Q \,|\, O \text{ is of the form } ?(F, y, P, t, c)\} \\ \mathit{outputs}(Q) & \stackrel{def}{=} & \{O \in Q \,|\, O \text{ is of the form } !v\} \end{array}$$

$$\mathit{patt}(?(F, y, P, t, c)) \stackrel{def}{=} F \qquad \mathit{tag}(?(F, y, P, t, c)) \stackrel{def}{=} c \qquad \mathit{val}(!v) \stackrel{def}{=} v$$

and we define the following functions:

$$\begin{aligned} inms(v, Q) &\stackrel{def}{=} \{(O, \sigma) \mid O \in inputs(Q),\ \sigma = match(patt(O), v, \emptyset),\ \sigma \neq \emptyset\} \\ outms(F, Q) &\stackrel{def}{=} \{(O, \sigma) \mid O \in outputs(Q),\ \sigma = match(F, val(O), \emptyset),\ \sigma \neq \emptyset\} \end{aligned}$$

which give us the set of observers and bindings for successful matches between a value v (resp. a pattern F) and the patterns (resp. values) available as input (resp. output) observers in the set.

Furthermore, we need the ability to remove input observers from all branches of a listener once a branch has been triggered. To this end, we define the following which removes all c tagged inputs from an observer set Q:

$$withdraw(Q, c) \stackrel{def}{=} \{O \in inputs(Q) \mid tag(O) \neq c\} \cup outputs(Q)$$

which we use to define the following function that removes all such inputs anywhere in the heap[8]:

$$\begin{aligned} remalts(\epsilon, c) &\stackrel{def}{=} \epsilon \\ remalts((H, x \mapsto Q), c) &\stackrel{def}{=} remalts(H, c), x \mapsto withdraw(Q, c) \end{aligned}$$

Executing time-slots

We now describe the "vertical dimension" of time, *i.e.*, the execution of terms within one time-slot.

Definition 13. (Time-slot execution) The behaviour of time-slots at a time $t \in \mathbb{R}_0^+$ is defined as the smallest relation $\rightarrow_t \subseteq (\mathcal{H} \times \mathcal{T}) \times (\mathcal{H} \times \mathcal{T})$ satisfying the axioms in Table 6.

The rule (NIL) simply ignores a terminated process. The (SPAWN) rules break a parallel composition into its components and schedules their execution in the current time-slot in an arbitrary order. Unlike Turner's machine, this is non-deterministic. The (RESTR) rule allocates a new spot for the new event, and initializes its observer set to empty.

The (OUT-F) rule describes the case when we trigger an event x, and there is no matching listener in x's observer set (output failure). In this case we simply add a new trigger observer to x's observer set and continue. The (OUT-S) rule (output success) applies when there is a matching observer O with σ being the resulting bindings. In this case, we remove all observers belonging to the listener which has matched the event (those tagged with c), and then execute the body P of the observer, applying the substitution σ extended with the binding of the elapsed time variable y to the difference between the current time t and the time u when the receiver started listening. Note that since there might be more

[8]This definition is inefficient since it traverses the entire heap. In practice, the input observers contain a list of pointers to the relevant heap entries to remove the alternatives efficiently.

$$(\textsc{nil})\quad \frac{-}{(H, \surd :: T) \rightarrow_t (H, T)}$$

$$(\textsc{spawn}_1)\quad \frac{-}{(H, (P_1 \parallel P_2) :: T) \rightarrow_t (H, T :: P_1 :: P_2)}$$

$$(\textsc{spawn}_2)\quad \frac{-}{(H, (P_1 \parallel P_2) :: T) \rightarrow_t (H, T :: P_2 :: P_1)}$$

$$(\textsc{restr})\quad \frac{k \text{ fresh}}{(H, \nu x.P :: T) \rightarrow_t (H\{k \mapsto \emptyset\}, P\{^k/_x\} :: T)}$$

$$(\textsc{out-f})\quad \frac{H(x) = Q \qquad v = eval(E) \qquad inms(v, Q) = \emptyset}{(H, x!E :: T) \rightarrow_t (H\{x \mapsto Q \cup \{!v\}\}, T)}$$

$$(\textsc{out-s})\quad \frac{H(x) = Q \qquad (O, \sigma) \in inms(eval(E), Q) \neq \emptyset \qquad O = ?(F, y, P, u, c)}{(H, x!E :: T) \rightarrow_t (remalts(H, c), P(\sigma \cup \{^{t-u}/_y\}) :: T)}$$

$$(\textsc{inp-f})\quad \frac{c \text{ fresh} \qquad \forall i \in I. \quad H(x_i) = Q_i \qquad outms(F_i, Q_i) = \emptyset}{(H, \sum_{i \in I} x_i?F_i@y_i \rightarrow P_i :: T) \rightarrow_t (H\{x_i \mapsto Q_i \cup \{?(F_i, y_i, P_i, t, c)\}\}_{i \in I}, T)}$$

$$(\textsc{inp-s})\quad \frac{\exists i \in I. \quad H(x_i) = Q_i \qquad (O, \sigma) \in outms(F_i, Q_i) \neq \emptyset}{(H, \sum_{i \in I} x_i?F_i@y_i \rightarrow P_i :: T) \rightarrow_t (H\{x_i \mapsto Q_i \backslash O\}, P_i(\sigma \cup \{^0/_{y_i}\}) :: T)}$$

$$(\textsc{inst})\quad \frac{A(\vec{x}) \stackrel{def}{=} P}{(H, A(\vec{v}) :: T) \rightarrow_t (H, T :: P\{^{\vec{v}}/_{\vec{x}}\})}$$

Table 6: Time-slot execution

than one successful match, the choice is non-deterministic, contrasting again with Turner's machine.

The rules (INP-S) and (INP-F) are the dual of (OUT-S) and (OUT-F). In rule (INP-F), when attempting to execute a listener, if there are no matching triggers in the relevant observer sets, we simply add the appropriate observers to the corresponding observer sets. Note that the added observers are tagged with the current time t, and with the same tag c. On the other hand, in rule (INP-S), one of the branches succeeds in matching the pattern with an observer O and binding σ. In this case, we remove the output observer from the event's observer set and execute the body of the corresponding branch, applying the substitution σ and binding y_i to 0, since the listener did not have to wait.[9]

Finally, the rule, (INST) deals with process instantiations. This simply assumes the set of process definitions is available, and schedules the execution of the body of the definition replacing its parameters by the arguments provided by the instantiation.[10]

Note that there is no rule associated with the Δ operator. We specify its behaviour in the description of the global scheduler below.

Global event scheduler

Now that we have the execution rules for time-slots, we can define the behaviour of the global event scheduler. For this purpose we will assume we have a function $insort : \mathbb{R}_0^+ \times \mathcal{P} \times \mathcal{R} \to \mathcal{R}$ which, given a time, inserts a term in the appropriate time-slot in the global queue, i.e., which satisfies the following:

$$
\begin{aligned}
&insort(t, P, \langle\rangle) = (t, P)\\
&insort(t, P, (t_1, T_1) \cdot \cdots \cdot (t_i, T_i) \cdot \cdots \cdot (t_n, T_n))\\
&\qquad = (t_1, T_1) \cdot \cdots \cdot (t_i, T_i :: P) \cdot \cdots \cdot (t_n, T_n)\\
&\qquad\qquad \text{if } t = t_i \text{ for some } i \in \{1, ..., n\}\\
&insort(t, P, (t_1, T_1) \cdot \cdots \cdot (t_i, T_i) \cdot (t_{i+1}, T_{i+1}) \cdot \cdots \cdot (t_n, T_n))\\
&\qquad = (t_1, T_1) \cdot \cdots \cdot (t_i, T_i) \cdot (t, P) \cdot (t_{i+1}, T_{i+1}) \cdot \cdots \cdot (t_n, T_n)\\
&\qquad\qquad \text{if } t \notin \{t_1, ..., t_n\} \text{ and } t_i < t < t_{i+1}
\end{aligned}
$$

Now we can define the global scheduler.

Definition 14. (Scheduler) The behaviour of the scheduler is given by the smallest relations $\hookrightarrow_0, \hookrightarrow_1, \hookrightarrow_2 \subseteq (\mathcal{H} \times \mathcal{R}) \times (\mathcal{H} \times \mathcal{R})$ which satisfy the rules in Table 7.[11]

The rule (TIMESLOT) states that as long as there are terms in the current time-slot then they are executed. The (ADVANCE) rule states that when the current time-slot is empty, simply move on to the next available time-slot. Finally, the (SCHED) rule describes the behaviour of the delay operator: to execute $\Delta E \to P$ simply compute the value d of E and insert P at time $t + d$ (where t

[9] Note that in this case we don't have to traverse the heap to remove input observers because it is a listener which is being executed and we have not added any input observers for it.

[10] In the actual implementation the heap is replaced by a structured environment that supports lexical scoping and which stores not only named events but also process definitions.

[11] The subscripts denote "vertical" step (0), "horizontal" step (1), and "scheduling" (2).

(TIMESLOT)	$\dfrac{(H,T) \to_t (H',T') \qquad T \neq \epsilon}{(H,(t,T)\cdot R) \hookrightarrow_0 (H',(t,T')\cdot R)}$	
(ADVANCE)	$\dfrac{-}{(H,(t,\epsilon)\cdot R) \hookrightarrow_1 (H,R)}$	
(SCHED)	$\dfrac{d = eval(E)}{(H,(t,\Delta E \to P :: T)\cdot R) \hookrightarrow_2 (H, insort(t+d,P,(t,T)\cdot R))}$	

Table 7: Scheduler.

is the current time). Note that this is inserted in $(t,T)\cdot R$ because the value of E may be 0. Also note that this may create a new time-slot, if there was none at time $t+d$.

Example

We illustrate the Abstract Machine with a sample execution. Consider the processes $Q \stackrel{def}{=} \Delta 3.2 \to x!1$ and $P \stackrel{def}{=} x?1@e \to P_1$ where $P_1 \stackrel{def}{=} \Delta(5-e) \to P_2$ for some P_2. Under the TLTS semantics we have the following execution for $\nu x.(P \parallel Q)$:

$$\begin{array}{lcl}
\nu x.(P \parallel Q) & \stackrel{3.2}{\rightsquigarrow} & \nu x.(x?1@e \to P_1\{^{e+3.2}/_e\} \parallel \Delta 0 \to x!1) \\
 & \xrightarrow{\tau} & \nu x.(x?1@e \to P_1\{^{e+3.2}/_e\} \parallel x!1) \\
 & \xrightarrow{\tau} & \nu x.(P_1\{^{e+3.2}/_e\}\{^0/_e\}) \\
 & \equiv & \nu x.(\Delta(5-(e+3.2)) \to P_2\{^{e+3.2}/_e\})\{^0/_e\} \\
 & \equiv & \nu x.\Delta(5-(0+3.2)) \to P_2\{^{e+3.2}/_e\}\{^0/_e\} \\
 & \stackrel{1.8}{\rightsquigarrow} & \nu x.\Delta 0 \to P_2\{^{e+3.2}/_e\}\{^0/_e\} \\
 & \xrightarrow{\tau} & \nu x.P_2\{^{e+3.2}/_e\}\{^0/_e\} \\
 & \equiv & \nu x.P_2\{^{3.2}/_e\}
\end{array}$$

Now we'll see how this is mimicked by the Abstract Machine. Suppose that the current time is 7.[12] Let us see the execution of the time-slot containing only $\nu x.(P \parallel Q)$, assuming the heap is initially empty (just to simplify notation):

$$\begin{array}{lcll}
(\epsilon, \nu x.(P \parallel Q)) & \to_7 & (k \mapsto \emptyset, (P \parallel Q)\{^k/_x\}) & (\text{RESTR})\ k \text{ fresh} \\
 & \equiv & (k \mapsto \emptyset, (P\{^k/_x\} \parallel Q\{^k/_x\})) & \\
 & \to_7 & (k \mapsto \emptyset, P\{^k/_x\} :: Q\{^k/_x\}) & (\text{SPAWN}_1) \\
 & \to_7 & (k \mapsto \{?(1,e,P_1\{^k/_x\},7,c)\}, Q\{^k/_x\}) & (\text{INP-F})\ c \text{ fresh}
\end{array}$$

Hence, at the global queue level we have:

[12] Any time will do.

$$
\begin{array}{lll}
(\epsilon, (7, \nu x.(P \parallel Q))) & & \\
\hookrightarrow_0 & (k \mapsto \emptyset, (7, (P \parallel Q)\{^k/_x\})) & (\textsc{timeslot}) \\
\equiv & (k \mapsto \emptyset, (7, (P\{^k/_x\} \parallel Q\{^k/_x\}))) & \\
\hookrightarrow_0 & (k \mapsto \emptyset, (7, P\{^k/_x\} :: Q\{^k/_x\})) & (\textsc{timeslot}) \\
\hookrightarrow_0 & (k \mapsto \{?(1, e, P_1\{^k/_x\}, 7, c)\}, (7, Q\{^k/_x\})) & (\textsc{timeslot}) \\
\equiv & (k \mapsto \{?(1, e, P_1\{^k/_x\}, 7, c)\}, (7, \Delta 3.2 \to k!1)) & \\
\hookrightarrow_2 & (k \mapsto \{?(1, e, P_1\{^k/_x\}, 7, c)\}, \mathit{insort}(7+3.2, k!1, (7, \epsilon))) & (\textsc{sched}) \\
\equiv & (k \mapsto \{?(1, e, P_1\{^k/_x\}, 7, c)\}, (7, \epsilon) \cdot (10.2, k!1)) & \\
\hookrightarrow_1 & (k \mapsto \{?(1, e, P_1\{^k/_x\}, 7, c)\}, (10.2, k!1)) & (\textsc{advance}) \\
\hookrightarrow_0 & (k \mapsto \emptyset, (10.2, P_1\{^k/_x\}\{^{10.2-7}/_e\})) & (\textsc{out-s}) + (\textsc{timeslot}) \\
\equiv & (k \mapsto \emptyset, (10.2, \Delta(5-3.2) \to P_2\{^{3.2}/_e\}\{^k/_x\})) & \\
\hookrightarrow_2 & (k \mapsto \emptyset, (10.2, \epsilon) \cdot (12, P_2\{^{3.2}/_e\}\{^k/_x\})) & (\textsc{sched}) \\
\hookrightarrow_1 & (k \mapsto \emptyset, (12, P_2\{^{3.2}/_e\}\{^k/_x\})) & (\textsc{advance})
\end{array}
$$

Here we used ($\textsc{spawn}_1$) which selected the left process P to be executed first. This resulted in registering the input observer in k's observer set. If we had used ($\textsc{spawn}_2$) instead, then Q would have been executed first, resulting in the scheduling happening first, but P would remain in the time-slot for time 7, and thus it would be executed before advancing in time.

3.2 Soundness

We now establish that reductions in our Abstract Machine correspond to valid π_{klt} executions following the same approach as that in [38] for the π-calculus. To do this we first encode the states of our Abstract Machine as π_{klt} terms. We need to encode the heap, its observer sets, time-slots and the global queue. We use $x \in H$ to denote that there is an entry for x in the heap H.

Definition 15. (Encoding the machine state) The ***set of triggers*** in the heap entry for x and the ***set of all triggers*** in the heap are given by:

$$
\begin{array}{rcl}
\mathit{triggers}(Q, x) & \stackrel{def}{=} & \{x!v \mid !v \in \mathit{outputs}(Q)\} \\
\mathit{alltriggers}(H) & \stackrel{def}{=} & \bigcup_{x \in H} \mathit{triggers}(H(x))
\end{array}
$$

The ***set of branches of a listener with tag*** c is given by:

$$
\begin{array}{rcl}
\mathit{branches}(H, c, t) & \stackrel{def}{=} & \bigcup_{x \in H} \mathit{alts}(H(x), c, x, t) \\
\mathit{alts}(Q, c, x, t) & \stackrel{def}{=} & \{x?F@y \to P\{^{y+(t-u)}/_y\} \mid ?(F, y, P, u, c) \in \mathit{inputs}(Q)\}
\end{array}
$$

The ***set of all listeners in the heap*** is given by:

$$
\mathit{alllisteners}(H, t) \stackrel{def}{=} \{\sum \mathit{branches}(H, c, t) \mid c \in \mathit{alltags}(H)\}
$$

where

$$
\begin{aligned}
alltags(H) &\stackrel{def}{=} \bigcup_{x \in H} tags(H(x)) \\
tags(Q) &\stackrel{def}{=} \{tag(O) \,|\, O \in inputs(Q)\}
\end{aligned}
$$

The ***encoding of the heap*** H ***at time*** t, is given by:

$$[\![H]\!]_t \stackrel{def}{=} (\prod alllisteners(H,t)) \parallel (\prod alltriggers(H))$$

The ***encoding of a time-slot*** $T = P_1 :: P_2 :: \cdots :: P_n$ is:

$$[\![T]\!] \stackrel{def}{=} P_1 \parallel P_2 \parallel \cdots \parallel P_n$$

The ***encoding of a heap/time-slot pair at time*** t is:

$$[\![(H,T)]\!]_t \stackrel{def}{=} \nu x_1, ..., x_k.([\![H]\!]_t \parallel [\![T]\!])$$

The ***encoding of the global queue*** $R = (t_1, T_1) \cdot (t_2, T_2) \cdot ... \cdot (t_m, T_m)$ is:

$$[\![R]\!] \stackrel{def}{=} [\![T_1]\!] \parallel \Delta(t_2 - t_1) \to [\![T_2]\!] \parallel \cdots \parallel \Delta(t_m - t_1) \to [\![T_m]\!]$$

The ***encoding of the machine state*** (H, R) is defined as:

$$[\![(H,R)]\!] \stackrel{def}{=} \nu x_1, ..., x_k.([\![H]\!]_{curtime(R)} \parallel [\![R]\!])$$

where $\{x_1, ..., x_k\}$ are the names of entries in the heap H and where $curtime((t,T) \cdot R) \stackrel{def}{=} t$ is the time-stamp of the first time-slot.[13]

A few comments about this encoding. First, note that to create a single listener we have to quantify over the possible entries in the heap. This is because each branch of a listener may listen to different events/channels, and therefore, the corresponding input observers may be dispersed over multiple heap entries. This is why we use the tags c, to identify all input observers which belong to the same listener. Second, note that the definition *alts* which gives us the alternative branches of a listener in some observer set, substitutes $y + (t - u)$ for y in P, where t is the current time and u is the time-stamp of the input observer, *i.e.*, when the listener began listening. This is because the encoding corresponds to taking a "snapshot" of the machine's state at time t, but each listener may have registered at some time $u \leqslant t$ so we have to take into account the time already elapsed. The encoding $[\![R]\!]$ for the global queue considers the first time-slot to represent the current one, so all future time-slots are delayed relative to the current time t_1.

In the following we will write $\Longrightarrow$ for $(\xrightarrow{\tau}) \equiv \cup \equiv$, this is $P \Longrightarrow Q$ if there is zero or one reductions from P to Q (modulo structural congruence). We first establish the consistency of execution within one time-slot at time t.

[13]Note that $[\![(H,R)]\!]$ is in canonical normal form (cf. Proposition 1).

Lemma 5. ***(Sound time-slot execution)***
If $(H,T) \rightarrow_t (H',T')$ *then* $[\![(H,T)]\!]_t \Longrightarrow [\![(H',T')]\!]_t$

From this, the validity of the scheduler's rules follows:

Theorem 4. ***(Abstract Machine Validity)***

1. *If* $(H,R) \hookrightarrow_0 (H',R')$ *then* $[\![(H,R)]\!] \Longrightarrow [\![(H',R')]\!]$

2. *If* $(H,R) \hookrightarrow_1 (H',R')$ *then* $[\![(H,R)]\!] \overset{d}{\leadsto}\!\!\Longrightarrow [\![(H',R')]\!]$ with $d = t_2 - t_1$ where $R = (t_1,T_1) \cdot (t_2,T_2) \cdot \ldots$

3. *If* $(H,R) \hookrightarrow_2 (H',R')$ *then* $[\![(H,R)]\!] \equiv [\![(H',R')]\!]$

3.3 Implementation: kiltera

We have implemented the Abstract Machine described in this Section, forming the basis of a language called kiltera. This language includes all the basic and derived constructs mentioned in this paper, plus some which we have not mentioned, such as event/channel arrays, indexed parallel composition, and loops, all defined in terms of the core. Table 8 shows the ASCII syntax equivalent of π_{klt} terms in kiltera.

The implementation follows the specification quite closely, but there is one difference, for the sake of efficiency. While the Abstract Machine presented here uses substitution, the actual implementation uses frame environments to support lexical scoping, according to the standard approach followed by functional programming languages. This also allows us to support local variable definitions and nested process definitions. Furthermore, process definitions result in closures which are treated as first class values and thus can be communicated.

The language also supports some primitives for mobile distributed computation and the implementation supports truly distributed (virtual time) simulation based on Time-Warp [19].

4 Related work

There have been several timed extensions to the π-calculus. We now contrast them with π_{klt}. Most of the timed extensions assume a discrete time model, and none include a time-observing construct as $@y$ in π_{klt} listeners. Most are based on the synchronous π-calculus, whereas π_{klt} is based on asynchronous communication. Only a couple have reported on notions of behavioural equivalence and compositionality. To the best of our knowledge, none of the previous work on timed π-calculi has studied time-bounded equivalences, none has dealt with the issue of legitimacy (finite variability) and no Abstract Machine has been proposed. Furthermore the only other language in this family, apart from ours, with an actual implementation is the stochastic π-calculus.

π_{klt} notation	kiltera ASCII notation
$\surd$	`done`
$x!E$	`trigger x with E`

$x_1?F_1@y_1 \to P_1 + \cdots + x_n?F_n@y_n \to P_n$	`when`
	`  x_1 with F_1 after y_1 ->`
	`    P_1`
	`|    ...`
	`...`
	`| x_n with F_n after y_n ->`
	`    P_n`

$\nu x_1, ..., x_n.P$	`event x_1, ..., x_n in` `  P`
$\Delta E \to P$	`wait E ->` `  P`

$P_1 \parallel \cdots \parallel P_n$	`par`
	`  P_1`
	`  ...`
	`  P_n`

$A(x_1, ..., x_n)$	`A[x_1,...,x_n]`
$A(x_1, ..., x_n) \stackrel{def}{=} P$	`process A[x_1,...,x_n]:` `  P`

Table 8: kiltera notation for π_{klt} syntax

The stochastic π-calculus

The stochastic π-calculus [30] makes a small but significant modification to the syntax and semantics of the (synchronous) π-calculus, by associating *rat*es to (input and output) actions. The meaning of this rate is that the given action is completed only after an amount of time which is drawn from an exponential distribution determined by the rate. As π_{klt}, processes execute with respect to a global clock over continuous time. But, unlike π_{klt}, the amount of delay is a random variable over a distribution, and therefore, transitions and their timing, are probabilistic. Semantically, each transition also has a rate, which depends on the rate of the corresponding term. This implies that communication between processes has a rate which depends on the rates of the participating input and output actions.

The probabilistic nature of this calculus results in a theory which is substantially different from ours, in scope and purpose. While both the stochastic π-calculus and π_{klt} can be used to reason about quantitative aspects of the timing behaviour of systems, the emphasis of the stochastic π theory is on probabilistic analysis and performance evaluation, whereas π_{klt}'s theory is focused on the discrete-event behaviour of systems, focusing in properties of legitimacy, behavioural equivalence and compositionality. We have not seen any comparable results in the theory of the stochastic π-calculus.

The πRT-calculus

The calculus introduced in [22] extends the π-calculus with a timeout operator from Timed CSP [37, 36] as well as allowing transmission of time values. Like π_{klt}, it has a model of computation based on a global clock, but unlike π_{klt}, time is discrete. The model of computation includes several properties similar to π_{klt}'s, such maximal progress, time determinacy and time continuity, Nevertheless, the status of these properties in their theory is not clear: are they assumptions or derived properties? To the best of our knowledge, the theory of this calculus has not been developed. The authors do not report any results regarding the issues we have treated in this paper, namely legitimacy, equivalence and compositionality.

The $T\pi$ and $TD\pi$-calculi

Of the timed-π calculi we surveyed, we have found the $T\pi$ and $TD\pi$-calculi [31] to have one of the most developed theories. This algebra extends the π-calculus in several ways: 1) computation proceeds with respect to a global clock over *discrete* time; 2) input and output actions (*i.e.*, listeners and triggers), have associated timers, which represent timeouts; 3) it supports distributed computation in a sense similar to π_{dklt}, where processes execute in locations, and can move between locations; 4) it has a type system which assigns capabilities to channels (*e.g.*, read or write), and locations (*e.g.*, can move). Its theory is centered around several notions of *barbed bisimulation*, which identify processes based on the satisfaction of certain predicates known as *barbs*. Different variants of this equivalence are proposed which distinguish processes depending on their type information, and/or their timing behaviour.

There are several obvious similarities as well as differences between π_{klt} and this calculus. First, both have a notion of timed execution with respect to a global clock, but while in $TD\pi$ time is discrete, in π_{klt} it is continuous. Second, the timer tags on channels of $TD\pi$ are easily emulated with timeouts in π_{klt}, but while π_{klt} provides a mechanism to observe the elapsed time, $TD\pi$ does not. Furthermore, unlike πRT and π_{klt}, time values are not transmittable as data. Third, $TD\pi$ provides a type system, while π_{klt}'s processes are untyped. The remaining distinction are with respect to the model of distributed processing, so we leave the comparison of these features to the second part of this paper.

As with the previously discussed variants, the theory does not provide any results regarding legitimacy, unlike our theory. They do discuss a timed barbed bisimilarity relation, but it is a much more stringent equivalence than our own open-timed bisimilarity. Whereas their equivalence requires an exact match in the timing of transitions, our equivalence requires match *up to* a given time. For example, consider the processes $(a? \to P) \overset{3}{\triangleright} Q_1$ and $(a? \to P) \overset{5}{\triangleright} Q_2$. In π_{klt}, we are able to show these processes as equivalent *up to* time 3, which includes equivalence of all intermediate states between time 0 (included) and time 3 (excluded). The corresponding $TD\pi$ processes could not be identified at all, because their timeouts are different. This makes timed barbed bisimilarity in $TD\pi$ a very stringent equivalence. Furthermore there are no results regarding its compositionality, whether it is a congruence, whether it is preserved by the operators in the language. We suspect that in fact, it is not a congruence, for the same reasons that strong, ground bisimilarity is not a congruence in the π-calculus, or (non-open) timed-bisimilarity is not a congruence in π_{klt}: it ignores transmitted values over channels.

The π_t and π_{mlt}-calculi

The π_t and π_{mlt}-calculi [5, 6] are similar to the $T\pi$ and $TD\pi$-calculi. They support time, and in the case of π_{mlt}, locations and message failures. Like π_{klt} communication is asynchronous. Nevertheless, time is discrete. Their timer construct corresponds in π_{klt} to a single-branching listener with a timeout. The theory studies several bisimilarities, establishing compositionality of strong, timed-closed, renaming-closed asynchronous bisimilarity. While the requirements of time-closedness and renaming-closedness are similar to those of our open timed-bisimilarity, their equivalence is defined only in the discrete-time setting, and identifies processes with respect to the entire future, unlike our equivalence *up-to* some time.

The timed-π calculus

Another timed extension to the π-calculus was provided in [14]. This extension introduces a delay operator and an *integral* operator, which is nothing but a sum (a choice operator) over a continuous time domain. Unfortunately there is nothing much to comment about this, since no theory is developed for this language, no results regarding legitimacy, equivalence, compositionality, or any other interesting property. Furthermore, there is no mention of an implementation either.

The SpacePi calculus

Another related process algebra recently introduced is SpacePi [20], an extension to the π-calculus with both continuous time and space. In this algebra, processes have an associated position in the euclidean vector space $\mathbb{R}^n$, as well as a movement function which updates the process's position. Communication channels have an associated *radius*, which determines how far a process can send or receive a message. Time, while continuous, is divided in computation intervals, so that the movement function specifies where a process will be at the end of the interval, if it does not interact with other processes.

This calculus is quite different than π_{klt}. While it presents some novel features, like radius for channels, it seems to impose a rather unnatural structure to time in the form of time-intervals, apparently, to justify the meaning of the movement functions. The result is that the modeller is forced to be aware of such imposition, and develop models accordingly. In addition, to the best of our knowledge, there are no theoretical results for this language, no properties regarding its semantics, such as conditions for legitimacy, or adequate notions of behavioural equivalence. Furthermore, there are no tools yet to support modelling and simulation with this calculus.

The ϕ-calculus

The ϕ-calculus [33] is another timed variant of the π-calculus for hybrid, embedded systems. Its distinguishing characteristic is the ability to describe hybrid systems consisting of an *environment* which runs over continuous time and a *process expression* which performs discrete actions which may change the continuous environment. The emphasis of this calculus is on hybrid systems, whereas π_{klt} is concerned with discrete-event systems.

Previous version of kiltera

To end, we would like to mention that the language presented in this paper was first published in [29], where rendez-vous was the main form of communication. In the author's thesis, this was changed to asynchronous communication, added multicasting and a form of *ephemeral* or *transient* interaction. In this paper we have decided to trim the language back to make it closer to the π-calculus, due to several difficulties introduced by mixing multicasting, transient interaction and time. This allows us to take advantage of a large body of work that is already in place for the π-calculus.

5 Conclusions and future work

We have introduced the π_{klt}-calculus, a dense-time extension to the asynchronous π-calculus which adds some high-level features such as pattern-matching. We have given an operational semantics in terms of timed-labelled transition systems, and developed its basic theory, including a notion of timed compositionality and a timed-bisimilarity up-to a given time, which is shown to be a timed-congruence. We also explored the issue of legitimate process definitions and established sufficient conditions for legitimacy. We developed an Abstract Machine for the calculus, based on event-scheduling, and established its soundness with respect to the operational semantics. We have implemented this in a language called kiltera. We view this language as a step towards filling the gap between foundational process algebras and realistic languages, specially in the context of real-time mobile systems.

In the second part we will present the distributed computation extensions to the core language, as well as a distributed Abstract Machine, based on the TimeWarp algorithm [19]. These have been implemented in the kiltera simulator.

There are multiple directions for future research. First, concerning the core theory, an axiomatization of open-time bisimilarity would be interesting to support automatic verification. Weaker variants of this equivalence are also of interest, as well as fully-abstract denotational models, and comparisons with similar models. The aplicability of real-time temporal logics and model-checking approaches can be valuable. At the meta-theory level it might be interesting to explore the idea of rule formats for timed process algebras which yield timed-congruences, in the same vein as those described in [1].

From the point of view of language features, several issues should be explored, including type systems, and with respect to the distributed features, active process migration. The implementation is mostly a proof of concept, with the simulator written entirely in Python, and thus it is not very fast. Optimization strategies for the Abstract Machine are also issues to be explored.

The implementation is available at `http://www.kiltera.org`.

References

[1] L. Aceto, W.J. Fokkink, and C. Verhoef. Structural Operational Semantics. In J.A. Bergstra, A. Ponse, S.A. Smolka, editor, *Handbook of Process Algebra*, chapter 3, pages 197–292. Elsevier, 2000.

[2] J. C. M. Baeten. A brief history of process algebra. *Theoretical Compututer Science*, 335(2-3):131–146, 2005.

[3] J. C. M. Baeten and J. A. Bergstra. Real Time Process Algebra. *Formal Aspects of Computing*, 3:142–188, 1991.

[4] J. C. M. Baeten, D. A. van Beek, and J. E. Rooda. Process algebra for dynamic system modeling. Technical report, Technische Universiteit Eindhoven, 2006.

[5] M. Berger. Basic Theory of Reduction Congruence for Two Timed Asynchronous π-Calculi. In *CONCUR 2004 - Concurrency Theory*, volume 3170 of *Lecture Notes in Computer Science*, pages 115 – 130. Springer, 2004.

[6] M. Berger and N. Yoshida. Timed, Distributed, Probabilistic, Typed Processes. In *Programming Languages and Systems*, volume 4807 of *Lecture Notes in Computer Science*, pages 158–174. Springer, 2007.

[7] J. A. Bergstra and J. W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1–3):109–137, 1984.

[8] G. Berry. *The Foundations of Esterel*. MIT Press, 2000. Editors: G. Plotkin, C. Stirling and M. Tofte.

[9] K. R. Braghetto, J. E. Ferreira, and C. Pu. Using process algebra to control the execution of business processes. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 128–129. ACM, 2008.

[10] A. M. K. Cheng. A survey of formal verification methods and tools for embedded and real-time systems. 2:184–195, August 12 2007.

[11] G. Ciobanu. Behaviour equivalences in timed distributed pi-calculus. In Martin Wirsing, Jean-Pierre Banâtre, Matthias M. Hölzl, and Axel Rauschmayer, editors, *Software-Intensive Systems and New Computing Paradigms - Challenges and Visions*, volume 5380 of *Lecture Notes in Computer Science*, pages 190–208. Springer, 2008.

[12] F. Ciocchetta and J. Hillston. *Formal Methods for Computational Systems Biology*, volume 5016 of *Lecture Notes in Computer Science*, chapter Process Algebras in Systems Biology, pages 265–312. Springer, 2008.

[13] C. Fidge and J. Zic. An expressive real-time CCS. In *Proceedings of the Australasian conference on Parallel and Real-time systems*, pages 365–372, 1995.

[14] M. Fischer. A new time extension to the π-calculus based on time consuming transition semantics. In Christoph Grimm, editor, *Languages for System Specification*, ChDL series. Springer-Verlag, 2004.

[15] N. Halbwachs, P. Caspi, and D. Pilaud. The synchronous dataflow programming language Lustre. In *Another Look at Real Time Programming, Proceedings of the IEEE, Special Issue*, September 1991.

[16] M. Hennessy. *Algebraic Theory of Processes*. MIT Press, 1988.

[17] M. Hennessy and T. Regan. A temporal process algebra. In Juan Quemada, José A. Mañas, and Enrique Vázquez, editors, *Proceedings of the Third International Conference on Formal Description Techniques for Distributed Systems and Communications Protocols (FORTE 90)*, pages 33–48. North-Holland, 1990.

[18] C. A. R. Hoare. Communicating Sequential Processes. *Communications of the ACM*, 21(8):666–677, August 1978.

[19] D. R. Jefferson. Virtual Time. *ACM-TOPLAS*, 7(3):404–425, July 1985.

[20] M. John, R. Ewald, and A. M. Uhrmacher. A Spatial Extension to the π-calculus. *Electronic Notes in Theoretical Computer Science*, 194(3):133–148, January 2008.

[21] P. Le Guernic, M. Le Borgue, T. Gauthier, and C. Le Maire. Programming real-time applications with Signal. In *Another Look at Real Time Programming. Proceedings of the IEEE, Special Issue*, September 1991.

[22] J. Y. Lee and J. Zic. On modeling real-time mobile processes. In *Proceedings of the twenty-fifth Australasian conference on Computer Science ACSC'02*, pages 139–147, January 2002.

[23] R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag, 1980.

[24] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[25] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, parts I and II. Reports ECS-LFCS-89-85 and 86, Computer Science Dept., University of Edinburgh, March 1989.

[26] F. Moller and C. Tofts. A temporal calculus of communicating systems. In J. C. M. Baeten and J. W. Klop, editors, *CONCUR '90: Theories of Concurrency: Unification and Extension*, volume 458 of *Lecture Notes in Computer Science*, pages 401–415. Springer-Verlag, 1990.

[27] U. Nestmann and B. C. Pierce. Decoding choice encodings. *Information and Computation*, 163(1):1–59, 2000.

[28] E. Posse. *Modelling and simulation of dynamic structure discrete-event systems.* Ph.D. thesis, School of Computer Science – McGill University, October 2008.

[29] E. Posse and H. Vangheluwe. kiltera: A simulation language for timed, dynamic structure systems. In *Proceedings of the 40th Annual Simulation Symposium*, 2007.

[30] C. Priami. Stochastic π-calculus. *Computer journal*, 38(7):578–589, 1995.

[31] C. Prisacariu and G. Ciobanu. Timed Distributed π-Calculus. Technical report, Institute of Computer Science, Romanian Academy, 2005.

[32] W. A. Roscoe. *Theory and Practice of Concurrency.* Prentice-Hall, 1998.

[33] W. C. Rounds and H. Song. The ϕ-calculus – a hybrid extension of the π-calculus to embedded systems. Technical Report CSE-TR-458-02, Dept. of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, 2002.

[34] G. Salaün, L. Bordeaux, and M. Schaerf. Describing and reasoning on web services using process algebra. In *ICWS '04: Proceedings of the IEEE International Conference on Web Services*, page 43, Washington, DC, USA, 2004. IEEE Computer Society.

[35] D. Sangiorgi. A theory of bisimulation for the π-calculus. Technical Report ECS-LFCS-93-270, 1993.

[36] S. Schneider. An operational semantics for Timed CSP. *Information and Computation*, 1995.

[37] S. Schneider. *Concurrent and Real-time Systems: The CSP Approach.* John Wiley & Sons, Ltd., 2000.

[38] D. N. Turner. *The polymorphic Pi-calculus: Theory and Implementation.* Ph.d. thesis, University of Edinburgh, 1996.

[39] Y. Wang. CCS + time = an interleaving model for real time systems. In *Proceedings of ICALP'91 – Annual International Colloquium on Automata, Languages and Programming*, 1991.

[40] B. P. Zeigler, H. Praehofer, and T. G. Kim. *Theory of Modeling and Simulation*. Academic Press, second edition, 2000.

A Basic properties

Proposition 1. *Every process is structurally congruent to a process of the form: $\nu x_1, ..., x_n.(P_1 \parallel \cdots \parallel P_k)$, where each P_i is either a trigger, a listener, a delay or a process instantiation. We call such form "canonical normal form".*

Proof. (Sketch) Using the scope extrusion axiom for structural congruence ($P \parallel \nu x.Q \equiv \nu x.(P \parallel Q)$ if $x \notin \text{fn}(P)$) together with the α-conversion axiom ($P \equiv Q$ if $P \equiv_\alpha Q$) we can extract, from all parallel subterms, any νx which is not inside a listener or delay, to the outermost position. □

Proposition 2.

1. *(Zero time advance) For any $P \in \mathcal{P}$, $P \overset{0}{\leadsto} P$.*
2. *(Time determinacy) For any $P, P', P'' \in \mathcal{P}$, $d \in \mathbb{R}_0^+$, if $P \overset{d}{\leadsto} P'$ and $P \overset{d}{\leadsto} P''$, then $P' \equiv P''$.*
3. *(Time continuity) For any $P, P' \in \mathcal{P}$, $d, d' \in \mathbb{R}_0^+$, $P \overset{d+d'}{\leadsto} P'$ if and only if there is a P'' such that $P \overset{d}{\leadsto} P''$ and $P'' \overset{d'}{\leadsto} P'$.*
4. *(Persistency of offers) For any $P, P' \in \mathcal{P}$, $d \in \mathbb{R}_0^+$, and any $\alpha \neq \tau$, if $P \xrightarrow{\alpha}$ and $P \overset{d}{\leadsto} P'$ then $P' \xrightarrow{\alpha}$.*

Proof. All these can be proven using standard induction on the structure of P. We show here time determinacy and time continuity. The rest are similar. We begin with time determinacy.

Case 1. $P \equiv \surd$, $P \equiv x!E$, $P \equiv \Delta E \to P'$, or $P \equiv \sum_{i \in I} \beta_i \to P_i$. In each of these cases, there is only one axiom that can be applied, or the (TCNGR) rule, and therefore the result is uniquely determined up-to $\equiv$.

Case 2. $P \equiv \nu x.P_1$. Assume that the statement holds for P_1 (which is a subterm of P): if $P_1 \overset{d}{\leadsto} P_1'$ and $P_1 \overset{d}{\leadsto} P_1''$ then $P_1' \equiv P_1''$. Suppose that $P \overset{d}{\leadsto} P'$ and $P \overset{d}{\leadsto} P''$. Each of these must be the conclusion of the (TNEW) rule. Therefore $P' \equiv \nu x.P_1'$ for some P_1' such that $P_1 \overset{d}{\leadsto} P_1'$. Similarly $P'' \equiv \nu x.P_1''$ for some P_1'' such that $P_1 \overset{d}{\leadsto} P_1''$. Hence, by induction hypothesis, $P_1' \equiv P_1''$, and since $\equiv$ is a congruence, $\nu x.P_1' \equiv \nu x.P_1''$, this is $P' \equiv P''$ are required.

Case 3. $P \equiv P_1 \parallel P_2$. Assume that the statement holds for each sub-term P_1 and P_2. Suppose that $P \overset{d}{\rightsquigarrow} P'$ and $P \overset{d}{\rightsquigarrow} P''$. Each case must be the conclusion of (TPAR). Therefore, $P' \equiv P_1' \parallel P_2'$ for some P_1' such that $P_1 \overset{d}{\rightsquigarrow} P_1'$ and some P_2' such that $P_2 \overset{d}{\rightsquigarrow} P_2'$. Similarly, since $P \overset{d}{\rightsquigarrow} P''$, we have that $P'' \equiv P_1'' \parallel P_2''$ for some P_1'' such that $P_1 \overset{d}{\rightsquigarrow} P_1''$ and some P_2' such that $P_2 \overset{d}{\rightsquigarrow} P_2''$. Hence, by our induction hypothesis, we conclude that $P_1' \equiv P_1''$ and $P_2' \equiv P_2''$. And since $\equiv$ is a congruence, we have that $P_1' \parallel P_2' \equiv P_1'' \parallel P_2''$. This is, $P' \equiv P''$ as required.

Now we show time continuity.

Case 1. $P \equiv \surd$ or $P \equiv x!E$ ($\Leftarrow$) Assume there is a P'' such that $P \overset{d}{\rightsquigarrow} P''$ and $P'' \overset{d'}{\rightsquigarrow} P'$. Then $P'' \equiv P$ and $P' \equiv P$. So, by (TIDLE) if $P \equiv \surd$ or by (TTRIG) if $P \equiv x!E$, $P \overset{d+d'}{\rightsquigarrow} P'$. ($\Rightarrow$) Dually, whenever $P \overset{d+d'}{\rightsquigarrow} P'$, it is possible to find a P'', namely $P'' \equiv P$, such that $P \overset{d}{\rightsquigarrow} P''$ and $P'' \overset{d'}{\rightsquigarrow} P'$.

Case 2. $P \equiv \Delta E \to P_1$. ($\Leftarrow$) Assume there is a P'' such that $P \overset{d}{\rightsquigarrow} P''$ and $P'' \overset{d'}{\rightsquigarrow} P'$. Then $P'' \equiv \Delta(E - d) \to P_1$ where $0 \leqslant d \leqslant \mathit{eval}(E)$. Therefore, $P' \equiv \Delta((E - d) - d') \to P_1$ where $0 \leqslant d' \leqslant \mathit{eval}(E - d)$. But this means that $P' \equiv \Delta(E-(d+d')) \to P_1$ and $0 \leqslant d+d' \leqslant \mathit{eval}(E)$, so by (TDELAY), $P \overset{d+d'}{\rightsquigarrow} P'$. ($\Rightarrow$) Suppose $P \overset{d+d'}{\rightsquigarrow} P'$. Then, it must be the case that $P' \equiv \Delta(E-(d+d')) \to P_1$ with $0 \leqslant d+d' \leqslant \mathit{eval}(E)$. Define $P'' \equiv \Delta(E-d) \to P_1$. Then $P \overset{d}{\rightsquigarrow} P''$ and $P'' \overset{d'}{\rightsquigarrow} P'$.

Case 3. $P \equiv \nu x.P_1$. Assume the statement is true for the sub-term P_1. ($\Leftarrow$) Suppose there is a P'' such that $P \overset{d}{\rightsquigarrow} P''$ and $P'' \overset{d'}{\rightsquigarrow} P'$. Then $P'' \equiv \nu x.P_1''$ for some P_1'' such that $P_1 \overset{d}{\rightsquigarrow} P_1''$. Hence it must be that $P' \equiv \nu x.P_1'$ for some P_1' such that $P_1'' \overset{d'}{\rightsquigarrow} P_1'$. This implies that $P_1 \overset{d+d'}{\rightsquigarrow} P_1'$ by the induction hypothesis, and this in turn implies that $\nu x.P_1 \overset{d+d'}{\rightsquigarrow} \nu x.P_1''$ by (TNEW), in other words, $P \overset{d+d'}{\rightsquigarrow} P'$. ($\Rightarrow$) Suppose $P \overset{d+d'}{\rightsquigarrow} P'$. Then, $P' \equiv \nu x.P_1'$ for some P_1' with $P_1 \overset{d+d'}{\rightsquigarrow} P_1'$. So, by induction hypothesis, there is a P_1'' such that $P_1 \overset{d}{\rightsquigarrow} P_1''$ and $P_1'' \overset{d'}{\rightsquigarrow} P_1'$. Define $P'' \equiv \nu x.P_1''$. It follows that $P \overset{d}{\rightsquigarrow} P''$ and $P'' \overset{d'}{\rightsquigarrow} P'$ by (TNEW).

Case 4. $P \equiv P_1 \parallel P_2$. Assume the statement is true for the sub-terms P_1 and P_2. ($\Leftarrow$) Suppose there is a P'' such that $P \overset{d}{\rightsquigarrow} P''$ and $P'' \overset{d'}{\rightsquigarrow} P'$. Then $P'' \equiv P_1'' \parallel P_2''$ for some P_1'' and P_2'' where $P_1 \overset{d}{\rightsquigarrow} P_1''$ and $P_2 \overset{d}{\rightsquigarrow} P_2''$. Hence, it is also the case that $P' \equiv P_1' \parallel P_2'$ for some P_1' and P_2' where $P_1'' \overset{d'}{\rightsquigarrow} P_1'$ and $P_2'' \overset{d'}{\rightsquigarrow} P_2'$. We conclude, by induction hypothesis, that $P_1 \overset{d+d'}{\rightsquigarrow} P_1'$ and $P_2 \overset{d+d'}{\rightsquigarrow} P_2'$. So, by (TPAR), we have that $P_1 \parallel P_2 \overset{d+d'}{\rightsquigarrow} P_1' \parallel P_2'$, this is, $P \overset{d+d'}{\rightsquigarrow} P'$.

($\Rightarrow$) Suppose $P \overset{d+d'}{\rightsquigarrow} P'$. Then, $P' \equiv P'_1 \parallel P'_2$ for some P'_1 and P'_2 where $P_1 \overset{d+d'}{\rightsquigarrow} P'_1$ and $P_2 \overset{d+d'}{\rightsquigarrow} P'_2$. So, by induction hypothesis, there are P''_1 and P''_2 such that $P_1 \overset{d}{\rightsquigarrow} P''_1 \overset{d'}{\rightsquigarrow} P'_1$ and $P_2 \overset{d}{\rightsquigarrow} P''_2 \overset{d'}{\rightsquigarrow} P'_2$. Therefore, by using (TPAR), we obtain that $P_1 \parallel P_2 \overset{d}{\rightsquigarrow} P''_1 \parallel P''_2 \overset{d'}{\rightsquigarrow} P'_1 \parallel P'_2$. This is, $P \overset{d}{\rightsquigarrow} P''$ and $P'' \overset{d'}{\rightsquigarrow} P'$, where $P'' \equiv P''_1 \parallel P''_2$.

Case 5. $P \equiv \Sigma_{i\in I}\beta_i \to P_i$, where for each $i \in I$, β_i is of the form $x_i?F_i@y_i$. ($\Leftarrow$) Suppose there is a P'' such that $P \overset{d}{\rightsquigarrow} P''$ and $P'' \overset{d'}{\rightsquigarrow} P'$. Then $P'' \equiv \Sigma_{i\in I}\beta_i \to P''_i$ where $P''_i \equiv P_i\{^{y_i+d}/_{y_i}\}$. Hence, $P' \equiv \Sigma_{i\in I}\beta_i \to P'_i$ where $P'_i \equiv P''_i\{^{y_i+d'}/_{y_i}\}$. So we have that $P'_i \equiv P_i\{^{y_i+d}/_{y_i}\}\{^{y_i+d'}/_{y_i}\} \equiv P_i\{^{y_i+(d+d')}/_{y_i}\}$ by composition of substitutions. Hence, $P \overset{d+d'}{\rightsquigarrow} P'$ by (TCHOICE). ($\Leftarrow$) Suppose $P \overset{d+d'}{\rightsquigarrow} P'$. So $P' \equiv \Sigma_{i\in I}\beta_i \to P'_i$ with $P'_i \equiv P_i\{^{y_i+(d+d')}/_{y_i}\}$. Define $P'' \equiv \Sigma_{i\in I}\beta_i \to P''_i$ with $P''_i \equiv P_i\{^{y_i+d}/_{y_i}\}$. Hence, $P''_i\{^{y_i+d'}/_{y_i}\} \equiv P_i\{^{y_i+d}/_{y_i}\}\{^{y_i+d'}/_{y_i}\} \equiv P_i\{^{y_i+(d+d')}/_{y_i}\} \equiv P'_i$. So, by (TCHOICE), $P'' \overset{d'}{\rightsquigarrow} P'$ and by definition of P'' and (TCHOICE), we also have that $P \overset{d}{\rightsquigarrow} P''$, as required.

□

B Timed-equivalence

Proposition 3. *For any TLTS $M = (\mathcal{S}, \mathcal{L}, \to, \rightsquigarrow)$,*

1. *For any $t \in \mathbb{R}_0^+$, $\Leftrightarrow_t$ is an equivalence relation.*
2. *$\Leftrightarrow$ is an open timed-bisimulation.*
3. *$\Leftrightarrow$ is the largest open timed-bisimulation.*

Proof. We only show the second, for illustration. Assume that $P \Leftrightarrow_t Q$, *i.e.* that $(P, t, Q) \in \Leftrightarrow$. By definition of $\Leftrightarrow$, there must be an open timed-bisimulation R such that $(P, t, Q) \in R$. We now check that $\Leftrightarrow$ satisfies the four conditions of open timed-bisimulation.

1. Suppose that $P\sigma \xrightarrow{\alpha} P'$ for some σ, α and P'. Then, $Q\sigma \xrightarrow{\alpha} Q'$ and $(P', t, Q') \in R$, since R is an open timed-bisimulation. But since there is an open timed-bisimulation that contains (P', t, Q'), then $P' \Leftrightarrow_t Q'$.
2. This is analogous to the previous item.
3. Suppose that $d < t$ and $P\sigma \overset{d}{\rightsquigarrow} P'$. Then $Q\sigma \overset{d}{\rightsquigarrow} Q'$ and $(P', t-d, Q') \in R$, since R is an open timed-bisimulation. But since there is an open timed-bisimulation that contains $(P', t-d, Q')$, then $P' \Leftrightarrow_{t-d} Q'$.
4. This is analogous to the previous item.

□

Proposition 4. *For any TLTS $M = (\mathcal{S}, \mathcal{L}, \rightarrow, \rightsquigarrow)$, any $t, u \in \mathbb{R}_0^+$, and any $P, P', P'' \in \mathcal{S}$:*

1. *If $P \Leftrightarrow_t P'$ then for any $u \leqslant t$, $P \Leftrightarrow_u P'$*
2. *If $P \Leftrightarrow_t P'$ and $P' \Leftrightarrow_u P''$ then $P \Leftrightarrow_{\min\{t,u\}} P''$*

Proof. We show 1 first. Pick any $t \in \mathbb{R}_0^+$. Consider the following relation: $R \stackrel{def}{=} \bigcup_{0 \leqslant z \leqslant t} R_z$ where

$$R_z \stackrel{def}{=} \{(P, u, Q) \in \mathcal{P} \times \mathbb{R}_0^+ \times \mathcal{P} \,|\, u \leqslant z \text{ and } P \Leftrightarrow_z Q\}$$

We claim that R is an open timed-bisimulation. To see this, take any $(P, u, Q) \in R$. Then there must be a $z \in \mathbb{R}_0^+$ such that $z \leqslant t$ and $(P, u, Q) \in R_z$. Hence $u \leqslant z$ and $P \Leftrightarrow_z Q$. Now we check the four conditions of open timed-bisimulation:

1. Suppose that $P\sigma \xrightarrow{\alpha} P'$ for some σ, α and P'. Since $P \Leftrightarrow_z Q$ we conclude that $Q\sigma \xrightarrow{\alpha} Q'$ for some Q' and $P' \Leftrightarrow_z Q'$. Since $u \leqslant z$ and $P' \Leftrightarrow_z Q'$ hold, we conclude that $(P', u, Q') \in R_z$ and therefore $(P', u, Q') \in R$.
2. Analogous to the previous item.
3. Suppose that $d < u$ and $P\sigma \stackrel{d}{\rightsquigarrow} P'$ for some σ and P'. Since $u \leqslant z$ we have that $d < z$, and since $P \Leftrightarrow_z Q$ we conclude that $Q\sigma \stackrel{d}{\rightsquigarrow} Q'$ for some Q' and $P' \Leftrightarrow_{z-d} Q'$. Since $u \leqslant z$, we have that $u - d \leqslant z - d$, and since $P' \Leftrightarrow_{z-d} Q'$, we obtain that $(P', u - d, Q') \in R_{z-d}$, but $d < u \leqslant z$ so $0 < z - d$ and $z \leqslant t$ which means that $z - d \leqslant t$, so $0 \leqslant z - d \leqslant t$ which implies that $(P', u - d, Q') \in R$.
4. Analogous to the previous item.

So we conclude that R is indeed an open timed-bisimulation. Suppose that $P \Leftrightarrow_t Q$ and $u \leqslant t$. Then $(P, u, Q) \in R_t$, which means that $(P, u, Q) \in R$. In other words, $P \Leftrightarrow_u Q$.

Now we show 2. Assume that $P \Leftrightarrow_t P'$ and $P' \Leftrightarrow_u P''$. Let $m = \min t, u$ so $m \leqslant t$ and $m \leqslant u$. Hence, by what we just proves (1), we have that $P \Leftrightarrow_m P'$ and $P' \Leftrightarrow_m P''$, which by transitivity implies $P \Leftrightarrow_m P''$. □

Lemma 1. *For any substitution σ, and any $t \in \mathbb{R}_0^+$, if $P \Leftrightarrow_t Q$ then $P\sigma \Leftrightarrow_t Q\sigma$.*

Proof. Let $R = \bigcup_{t \in \mathbb{R}_0^+} R_t$ with $R_t \stackrel{def}{=} \{(P\sigma, t, Q\sigma) \,|\, \sigma \text{is any substitution and } P \Leftrightarrow_t Q\}$. We claim that R is an open timed-bisimulation. First, note that if $P \Leftrightarrow_t Q$ then $(P, t, Q) \in R_t$ (and thus $(P, t, Q) \in R)$), since the identity function is a substitution. Assume that $(P, Q) \in R$. Then $(P, Q) \in R_t$ for some t and $P = P_0\sigma_0$ and $Q = Q_0\sigma_0$ for some substitution σ_0 and some P_0, Q_0 with $P_0 \Leftrightarrow_t Q_0$. Take any substitution σ. Then

1. Suppose $P\sigma \xrightarrow{\alpha} P'$. This is, $P_0\sigma_0\sigma \xrightarrow{\alpha} P'$. But $P_0 \Leftrightarrow_t Q_0$, so $Q_0\sigma_0\sigma \xrightarrow{\alpha} Q'$ with $P' \Leftrightarrow_t Q'$, which implies that $(P', t, Q') \in R_t$ and thus $Q\sigma \xrightarrow{\alpha} Q'$ and $(P', t, Q') \in R$ as required.
2. This is the dual of the previous case.
3. Suppose $d < t$ and $P\sigma \overset{d}{\rightsquigarrow} P'$. This is, $P_0\sigma_o\sigma \overset{d}{\rightsquigarrow} P'$. Since $P_0 \Leftrightarrow_t Q_0$, we have that $Q_0\sigma_0\sigma \overset{d}{\rightsquigarrow} Q'$ and $P' \Leftrightarrow_{t-d} Q'$. But this implies that $(P', t - d, Q') \in R_{t-d}$. Therefore $Q\sigma \overset{d}{\rightsquigarrow} Q'$ and $(P', t - d, Q') \in R$.
4. This is the dual of the previous case.

We conclude that R is an open timed-bisimulation. So, for any σ, if $P \Leftrightarrow_t Q$ then $(P\sigma, t, Q\sigma) \in R_t$, and therefore $(P\sigma, t, Q\sigma) \in R$, *i.e.* $P\sigma \Leftrightarrow_t Q\sigma$. □

Lemma 2. *For any $P, P', Q_1, ..., Q_n \in \mathcal{P}$, and any $t \in \mathbb{R}_0^+$, if $P \Leftrightarrow_t P'$ then:*

1. $\Delta E \to P \Leftrightarrow_{t+e} \Delta E \to P'$ *where* $e = eval(E)$
2. $\nu x.P \Leftrightarrow_t \nu x.P'$
3. $P \parallel Q \Leftrightarrow_t P' \parallel Q$
4. $x?F@y \to P + \sum_{i=1}^{n} \beta_i \to Q_i \Leftrightarrow_t x?F@y \to P' + \sum_{i=1}^{n} \beta_i \to Q_i$ *where each* β_i is of the form $x_i?F_i@y_i$.

Proof.

1. We first show that $\Delta E \to P \Leftrightarrow_{t+e} \Delta E \to P'$ where $e = eval(E)$ given that $P \Leftrightarrow_t P'$. Let $R = \bigcup_{u \in \mathbb{R}_0^+} R_u$ with $R_u \overset{def}{=} R'_u \cup \Leftrightarrow_u$ where

$$R'_u \overset{def}{=} \{(\Delta E \to P_1, u, \Delta E \to P_2) \,|\, P_1 \Leftrightarrow_{u-e} P_2 \,\&\, e = eval(E) \leqslant u\}$$

We claim that R is an open timed-bisimulation. Suppose that $(Q_1, u, Q_2) \in R$. Then $(Q_1, u, Q_2) \in R_u$. Hence either $(Q_1, u, Q_2) \in R'_u$ or $Q_1 \Leftrightarrow_u Q_2$.

Case 1. $(Q_1, u, Q_2) \in R'_u$. Then $Q_i \overset{def}{=} \Delta E \to P_i$ for $i \in \{1, 2\}$ and $P_1 \Leftrightarrow_{u-e} P_2$ with $e = eval(E) \leqslant u$. Now we check the four conditions for open timed-bisimulation:

(a) Suppose $Q_1\sigma \xrightarrow{\alpha} Q'_1$. This transition could be derived only by the (DELAY) rule. Hence, $\alpha = \tau$, $e = eval(E\sigma) = 0$ and $Q'_1 \equiv P_1\sigma$. Since $e = 0$, this transition can be matched, using (DELAY), by $Q_2\sigma \xrightarrow{\alpha} Q'_2$ where $Q'_2 \equiv P_2\sigma$. But $P_1 \Leftrightarrow_{u-e} P_2$ and $\Leftrightarrow_t$ is closed under substitution (Lemma 1), therefore, $Q'_1 \Leftrightarrow_{u-e} Q'_2$, but $e = 0$, so $Q'_1 \Leftrightarrow_u Q'_2$, which implies that $(Q'_1, u, Q'_2) \in R$.

(b) This is the dual of the previous item.

(c) Suppose $d < u$ and $Q_1\sigma \overset{d}{\leadsto} Q_1'$. This evolution could be derived only by the (TDELAY) rule. Hence $Q_1' \equiv \Delta(E-d)\sigma \to P_1\sigma$. Let $e' = eval((E-d)\sigma) = e - d$ where $e = eval(E\sigma)$. This can be matched by $Q_2\sigma \overset{d}{\leadsto} Q_2'$ where $Q_2' \equiv \Delta(E-d)\sigma \to P_2\sigma$. We know that $P_1 \Leftrightarrow_{u-e} P_2$, and since $\Leftrightarrow_t$ is closed under substitution (Lemma 1), we have that $P_1\sigma \Leftrightarrow_{u-e} P_2\sigma$. But note that $e = e'+d$, so $u - e = u - e' - d = (u-d) - e'$. Furthermore, since $e \leqslant u$ we have that $e' \leqslant u - d$. Also note that, by definition,

$$R'_{u-d} = \{(\Delta E' \to P_1', u-d, \Delta E' \to P_2') \mid \\ P_1' \Leftrightarrow_{(u-d)-e'} P_2' \,\&\, e' = eval(E') \leqslant u - d\}$$

Hence, by taking $P_1' \equiv P_1\sigma$, $P_2' \equiv P_2\sigma$ and $E' = (E-d)\sigma$ we have that $(Q_1', u-d, Q_2') \in R'_{u-d}$ and therefore $(Q_1', u-d, Q_2') \in R_{u-d}$, and $(Q_1', u-d, Q_2') \in R$ as required.

(d) This is the dual of the previous item.

Case 2. $Q_1 \Leftrightarrow_u Q_2$. We check the four conditions for open time-bisimulation:

(a) Suppose $Q_1\sigma \xrightarrow{\eta} Q_1'$. Then $Q_2\sigma \xrightarrow{\eta} Q_2'$ and $Q_1' \Leftrightarrow_u Q_2'$, hence $(Q_1', u, Q_2') \in R$.

(b) This is the dual of the previous item.

(c) Suppose $d < u$ and $Q_1\sigma \overset{d}{\leadsto} Q_1'$. Then $Q_2\sigma \overset{d}{\leadsto} Q_2'$ and $Q_1' \Leftrightarrow_{u-d} Q_2'$, hence $(Q_1', u-d, Q_2') \in R$.

(d) This is the dual of the previous item.

We conclude that R is indeed an open timed-bisimulation. Now, suppose that $P_1 \Leftrightarrow_t P_2$. Then we have that $(\Delta E \to P_1, t+e, \Delta E \to P_2) \in R'_{t+e}$, where $e = eval(E)$, and so $(\Delta E \to P_1, t+e, \Delta E \to P_2) \in R$, which means that $\Delta E \to P_1 \Leftrightarrow_{t+e} \Delta E \to P_2$, as required.

2. Now we show that $\nu x.P \Leftrightarrow_t \nu x.P'$ given that $P \Leftrightarrow_t P'$. Let $R = \bigcup_{u \in \mathbb{R}_0^+} R_u$ where

$$R_u \overset{def}{=} \{(\nu x.P_1, u, \nu x.P_2) \mid P_1 \Leftrightarrow_u P_2\}$$

We claim that R is an open time-bisimulation. Let $(\nu x.P_1, u, \nu x.P_2) \in R$. So $(\nu x.P_1, u, \nu x.P_2) \in R_u$ and therefore $P_1 \Leftrightarrow_u P_2$. We proceed to check the four conditions of open time-bisimulation:

(a) Suppose $(\nu x.P_1)\sigma \xrightarrow{\alpha} X$. For simplicity assume that x does not occur in σ either as a source or as a target.[14] This way we can assume that $(\nu x.P_1)\sigma = \nu x.P_1$ and $(\nu x.P_2)\sigma = \nu x.P_2\sigma$. The transition must have been derived by (NEW):

$$\frac{P_1\sigma \xrightarrow{\alpha} P_1' \qquad x \notin \mathrm{n}(()\alpha)}{\nu x.P_1\sigma \xrightarrow{\alpha} X \equiv \nu x.P_1'}$$

[14] If x occurs in σ we can simply rename it with a fresh name x' to $\nu x'.P_1\{x'/x\}$ and $\nu x'.P_2\{x'/x\}$.

Hence $x \notin \mathrm{n}(\alpha)$ and $P_1\sigma \xrightarrow{\alpha} P_1'$. But $P_1 \Leftrightarrow_u P_2$ so the transition $P_1\sigma \xrightarrow{\alpha} P_1'$ can be matched by $P_2\sigma \xrightarrow{\alpha} P_2'$ with $P_1' \Leftrightarrow_u P_2'$. So by (NEW), $\nu x.P_2\sigma \xrightarrow{\alpha} Y \equiv \nu x.P_2'$. Since $P_1' \Leftrightarrow_u P_2'$, we have that $(\nu x.P_1', u, \nu x.P_2') \in R_u$, *i.e.* $(X, u, Y) \in R_u$ and therefore $(X, u, Y) \in R$.

(b) Suppose $(\nu x.P_2)\sigma \xrightarrow{\eta} Y$. This is the dual of the previous item.

(c) Suppose $d < u$ and $(\nu x.P_1)\sigma \overset{d}{\rightsquigarrow} X$. For simplicity assume that x does not occur in σ either as a source or as a target. If it is we can simply rename it with a fresh name. This way we can assume that $(\nu x.P_1)\sigma = \nu x.P_1$ and $(\nu x.P_2)\sigma = \nu x.P_2\sigma$. This transition could have been derived only by (TNEW):

$$\frac{P_1\sigma \overset{d}{\rightsquigarrow} P_1'}{\nu x.P_1\sigma \overset{d}{\rightsquigarrow} X \equiv \nu x.P_1'}$$

So it must be that $P_1\sigma \overset{d}{\rightsquigarrow} P_1'$. But $P_1 \Leftrightarrow_u P_2$, and so $P_1\sigma \overset{d}{\rightsquigarrow} P_1'$ can be matched by $P_2\sigma \overset{d}{\rightsquigarrow} P_2'$ and $P_1' \Leftrightarrow_{u-d} P_2'$. Therefore by (TNEW), $\nu x.P_2\sigma \overset{d}{\rightsquigarrow} Y \equiv \nu x.P_2'$. Since $P_1' \Leftrightarrow_{u-d} P_2'$, we have that $(\nu x.P_1', u - d, \nu x.P_2') \in R_{u-d}$, *i.e.* $(X, u-d, Y) \in R_{u-d}$ and therefore $(X, u - d, Y) \in R$.

(d) Suppose $d < u$ and $(\nu x.P_2)\sigma \overset{d}{\rightsquigarrow} Y$. This is the dual of the previous item.

We conclude that R is an open time-bisimulation. Suppose $P_1 \Leftrightarrow_t P_2$. Therefore $(\nu x.P_1, t, \nu x.P_2) \in R_t$ and so $(\nu x.P_1, t, \nu x.P_2) \in R$ which means that $\nu x.P_1 \Leftrightarrow_t \nu x.P_2$.

3. Now we prove that $P \parallel Q \Leftrightarrow_t P' \parallel Q$ given that $P \Leftrightarrow_t P'$. Let $R \overset{def}{=} \bigcup_{t \in \mathbb{R}_0^+} R_t$ where

$$R_t \overset{def}{=} \{(P_1 \parallel Q, t, P_2 \parallel Q) \mid P_1 \Leftrightarrow_t P_2,\ Q \text{ is any process}\}$$

We claim that R is an open time-bisimulation. From this the result follows directly. Suppose $(P_1 \parallel Q, t, P_2 \parallel Q) \in R$. So $(P_1 \parallel Q, t, P_2 \parallel Q) \in R_t$. Hence $P_1 \Leftrightarrow_t P_2$ by definition of R_t. Now we proceed to check each of the four conditions of time-bisimulation:

(a) Suppose that $(P_1 \parallel Q)\sigma \xrightarrow{\alpha} X$. We need to show that there is a Y such that $(P_2 \parallel Q)\sigma \xrightarrow{\alpha} Y$ with $(X, t, Y) \in R$. Note that $(P_1 \parallel Q)\sigma \equiv P_1\sigma \parallel Q\sigma$ by definition of substitution. We proceed by considering the possible cases of how this transition was derived:

Case 1. The transition was derived by rule (PAR):

$$\frac{P_1\sigma \xrightarrow{\alpha} P_1' \qquad \mathrm{bn}(\alpha) \cap \mathrm{fn}(Q\sigma) = \emptyset}{P_1\sigma \parallel Q\sigma \xrightarrow{\alpha} X \equiv P_1' \parallel Q\sigma}$$

so it must be the case that $P_1\sigma \xrightarrow{\alpha} P_1'$ and $\mathrm{bn}(\alpha) \cap \mathrm{fn}(Q\sigma) = \emptyset$. But we know that $P_1 \Leftrightarrow_t P_2$, so $P_2\sigma \xrightarrow{\alpha} P_2'$ and $P_1' \Leftrightarrow_t P_2'$. So by applying (PAR), we have that $P_2\sigma \parallel Q\sigma \xrightarrow{\alpha} Y \equiv P_2' \parallel Q\sigma$, and since $P_1' \Leftrightarrow_t P_2'$, we have that $(P_1' \parallel Q\sigma, t, P_2' \parallel Q\sigma) \in R_t$, *i.e.* $(X, t, Y) \in R_t$ and so $(X, t, Y) \in R$.

Case 2. The transition was derived by rule (COMM): In this case we have two possibilities: either the input $x?v$ was performed by $P_1\sigma$ or it was performed by $Q\sigma$. Both cases are analogous, so we'll show only the first:

$$\frac{P_1\sigma \xrightarrow{x?v} P_1' \qquad Q\sigma \xrightarrow{x!v} Q'}{P_1\sigma \parallel Q\sigma \xrightarrow{\tau} X \equiv P_1' \parallel Q'}$$

so it must be the case that $P_1\sigma \xrightarrow{x?v} P_1'$ and $Q\sigma \xrightarrow{x!v} Q'$. But we know that $P_1 \Leftrightarrow_t P_2$, so $P_2\sigma \xrightarrow{x?v} P_2'$ and $P_1' \Leftrightarrow_t P_2'$. So by applying (PAR), we have that $P_2\sigma \parallel Q\sigma \xrightarrow{\tau} Y \equiv P_2' \parallel Q$, and since $P_1' \Leftrightarrow_t P_2'$, we have that $(P_1' \parallel Q, t, P_2' \parallel Q) \in R_t$, *i.e.* $(X, t, Y) \in R_t$ and so $(X, t, Y) \in R$.

(b) Suppose that $(P_2 \parallel Q)\sigma \xrightarrow{\alpha} Y$. This is analogous to the previous case.

(c) Suppose that $d < t$ and $(P_1 \parallel Q)\sigma \equiv P_1\sigma \parallel Q\sigma \overset{d}{\rightsquigarrow} X$. We need to show that there is a Y such that $(P_2 \parallel Q)\sigma \equiv P_2\sigma \parallel Q\sigma \overset{d}{\rightsquigarrow} Y$ and $(X, t-d, Y) \in R$. As before, we proceed by considering the possible cases of how this evolution was derived. But this could have been derived only by (TPAR):

$$\frac{P_1\sigma \overset{d}{\rightsquigarrow} P_1' \qquad Q\sigma \overset{d}{\rightsquigarrow} Q'}{P_1\sigma \parallel Q\sigma \overset{d}{\rightsquigarrow} X \equiv P_1' \parallel Q'}$$

so it must be that $P_1\sigma \overset{d}{\rightsquigarrow} P_1'$ and $Q\sigma \overset{d}{\rightsquigarrow} Q'$. But since $P_1 \Leftrightarrow_t P_2$, $P_2\sigma \overset{d}{\rightsquigarrow} P_2'$ and $P_1' \Leftrightarrow_{t-d} P_2'$. By (TPAR), we obtain $P_2\sigma \parallel Q\sigma \overset{d}{\rightsquigarrow} Y \equiv P_2' \parallel Q'$, and since $P_1' \Leftrightarrow_{t-d} P_2'$, we conclude that $(P_1' \parallel Q', t-d, P_2' \parallel Q') \in R_{t-d}$, *i.e.* $(X, t-d, Y) \in R_{t-d}$ and so $(X, t-d, Y) \in R$.

(d) Suppose that $d < t$ and $(P_2 \parallel Q)\sigma \equiv P_2\sigma \parallel Q\sigma \overset{d}{\rightsquigarrow} Y$. This is analogous to the previous case.

Now that we have established that R is an open timed-bisimulation, the result follows, because if we assume that $P \Leftrightarrow_t P'$, then there is an open timed-bisimulation, R, such that $(P \parallel Q, t, P' \parallel Q) \in R$, for any Q, i.e., $P \parallel Q \Leftrightarrow_t P' \parallel Q$.

4. Finally we prove that given $P \Leftrightarrow_t P'$, $\beta \to P + \sum_{i=1}^{n} \beta_i \to Q_i \Leftrightarrow_t \beta \to P' + \sum_{i=1}^{n} \beta_i \to Q_i$ where β is of the form $x?F@y$ and each β_i is of the form $x_i?F_i@y_i$. Let $R = \bigcup_{u \in \mathbb{R}_0^+} R_u$ with $R_u \overset{def}{=} R'_u \cup \Leftrightarrow_u$ where

$$R'_u \overset{def}{=} \{(\beta \to P + \Sigma_{i\in I}\beta_i \to Q_i, u, \beta \to P' + \Sigma_{i\in I}\beta_i \to Q_i) \mid P \Leftrightarrow_u P'\}$$

We first claim that R is an open time-bisimulation. Suppose that $(P_1, u, P_2) \in R$. Then $(P_1, u, P_2) \in R_u$. Hence either $(P_1, u, P_2) \in R'_u$ or $P_1 \Leftrightarrow_u P_2$.

Case 1. $(P_1, u, P_2) \in R'_u$. Then $P_1 \equiv \beta \to P + \Sigma_{i\in I}\beta_i \to Q_i$, $P_2 \equiv \beta \to P' + \Sigma_{i\in I}\beta_i \to Q_i$, and $P \Leftrightarrow_u P'$. Note that $P_1\sigma \equiv \beta\sigma \to P\sigma + \Sigma_{i\in I}\beta_i\sigma \to Q_i\sigma$ and $P_2\sigma \equiv \beta\sigma \to P'\sigma + \Sigma_{i\in I}\beta_i\sigma \to Q_i\sigma$. We now check the four conditions of open timed-bisimulation:[15]

(a) Suppose $P_1\sigma \xrightarrow{\alpha} P'_1$. This transition could be derived only by the (CHOICE) rule. There are two possibilities, either the transition was taken by the $\beta\sigma$ branch, or by any of the $\beta_i\sigma$ branches.
Subcase 1: Suppose that the $\beta\sigma$ transition was taken. Then, $\alpha = \sigma(x?v)$ for some v where $\sigma' = match(F, v, \emptyset) \neq \emptyset$, and $P'_1 = P\sigma\sigma'\{0/y\}$. This transition can by matched by $P_2\sigma$, since it has the same guard $\beta\sigma$, so $P_2\sigma \xrightarrow{\alpha} P'_2$ where $P'_2 = P'\sigma\sigma'\{0/y\}$. But we know that $P \Leftrightarrow_u P'$, and since $\Leftrightarrow_u$ is closed under substitution (Lemma 1) we have that $P'_1 \Leftrightarrow_u P'_2$ which entails that $(P'_1, u, P'_2) \in R_u$ and therefore $(P'_1, u, P'_2) \in R$.
Subcase 2: Suppose that some $\beta_i\sigma$ transition was enabled instead. Then, $\alpha = \sigma(x_i?v)$ for some v where $\sigma' = match(F_i, v, \emptyset) \neq \emptyset$, and $P'_1 = Q\sigma\sigma'\{0/y_i\}$. This transition can by matched by $P_2\sigma$, since it has the same guard $\beta\sigma$, so $P_2\sigma \xrightarrow{\alpha} P'_2$ where $P'_2 = Q\sigma\sigma'\{0/y_i\}$. Hence $P'_1 = P'_2$ which implies that $P'_1 \Leftrightarrow_u P'_2$ and so $(P'_1, u, P'_2) \in R_u$ and therefore $(P'_1, u, P'_2) \in R$.

(b) Suppose $P_2\sigma \xrightarrow{\alpha} P'_2$. This case is the exact symmetric of the previous one.

(c) Suppose $d < u$ and $P_1\sigma \overset{d}{\rightsquigarrow} P'_1$. This could be derived only by (TCHOICE). This implies that $P'_1 \equiv \beta\sigma \to P\sigma\{y+d/y\} + \Sigma_{i\in I}\beta_i\sigma \to Q_i\sigma\{y_i+d/y_i\}$. This can be matched by $P_2\sigma \overset{d}{\rightsquigarrow} P'_2$ where $P'_2 \equiv \beta\sigma \to P'\sigma\{y+d/y\} + \Sigma_{i\in I}\beta_i\sigma \to Q_i\sigma\{y_i+d/y_i\}$. Since $P \Leftrightarrow_u P'$, by closure under substitution (Lemma 1) we have that $P\sigma\{y+d/y\} \Leftrightarrow_u P'\sigma\{y+d/y\}$. Furthermore, we have that, $u-d \leqslant u$

[15] For simplicity we assume that bound variables in P_1 and P_2 have been renamed if necessary.

because $0 \leqslant d < u$, and therefore $P\sigma\{^{y+d}/_y\} \Leftrightarrow_{u-d} P'\sigma\{^{y+d}/_y\}$ by Proposition 4. Hence $(P_1', u-d, P_2') \in R_u'$ and so $(P_1', u-d, P_2') \in R$.

(d) Suppose $d < u$ and $P_2\sigma \overset{d}{\rightsquigarrow} P_2'$. As in the previous case.

Case 2. $P_1 \Leftrightarrow_u P_2$. We check the four conditions for open time-bisimulation:

(a) Suppose $P_1\sigma \xrightarrow{\alpha} P_1'$. Then $P_2\sigma \xrightarrow{\alpha} P_2'$ and $P_1' \Leftrightarrow_u P_2'$, hence $(P_1', u, P_2') \in R$.

(b) This is the dual of the previous item.

(c) Suppose $d < u$ and $P_1\sigma \overset{d}{\rightsquigarrow} P_1'$. Then $P_2\sigma \overset{d}{\rightsquigarrow} P_2'$ and $P_1' \Leftrightarrow_{u-d} P_2'$, hence $(P_1', u-d, P_2') \in R$.

(d) This is the dual of the previous item.

Now that we have established that R is an open timed-bisimulation, the result follows, because if we assume that $P \Leftrightarrow_t P'$, then there is an open timed-bisimulation, R, such that $(\beta \rightarrow P + \sum_{i \in I} \beta_i \rightarrow Q_i, t, \beta \rightarrow P' + \sum_{i \in I} \beta_i \rightarrow Q_i) \in R$, i.e., $\beta \rightarrow P + \sum_{i \in I} \beta_i \rightarrow Q_i \Leftrightarrow_t \beta \rightarrow P' + \sum_{i \in I} \beta_i \rightarrow Q_i$.

□

Theorem 1. $\Leftrightarrow_t$ *is a timed-congruence up-to* t *and* $\Leftrightarrow$ *is a timed-congruence.*

Proof. This follows from Lemma 2 together with Proposition 4: For any given $t \in \mathbb{R}_0^+$, given $P \Leftrightarrow_t P'$ we obtain $\Delta E \rightarrow P \Leftrightarrow_{t+e} \Delta E \rightarrow P'$ where $e = eval(E)$ by Lemma 2, which implies $\Delta E \rightarrow P \Leftrightarrow_t \Delta E \rightarrow P'$ by Proposition 4, since $t \leqslant t + e$, and so $\Leftrightarrow_t$ is preserved by delays. The rest of the items in Lemma 2 show that it is also preserved by all other operators, hence, for any $t \in \mathbb{R}_0^+$, $\Leftrightarrow$ is a t-congruence, and so $\Leftrightarrow$ is a timed-congruence. Furthermore, $\Leftrightarrow_t$ is a timed-congruence up-to t because each $\Leftrightarrow_u$ is a congruence for each $0 \leqslant u \leqslant t$. □

Corollary 1. *For any families of terms* $\{P_i \in \mathcal{P}\}_{i \in I}$ *and* $\{Q_i \in \mathcal{P}\}_{i \in I}$*, if for each* $i \in I$*,* $P_i \Leftrightarrow_{t_i} Q_i$ *then*

1. $\prod_{i \in I} P_i \Leftrightarrow_{\min\{t_i | i \in I\}} \prod_{i \in I} Q_i$

2. $\sum_{i \in I} x_i?F_i@y_i \rightarrow P_i \Leftrightarrow_{\min\{t_i | i \in I\}} \sum_{i \in I} x_i?F_i@y_i \rightarrow Q_i$

Proof. This is a consequence of Lemma 2. □

1. By induction on the size of I. Let $I = \{1, ..., n-1\}$, $P = \prod_{i \in I} P_i$, $Q = \prod_{i \in I} Q_i$ and $M = \min\{t_i | t_i \in I\}$. Our goal is to show that $P' \Leftrightarrow_{M'} Q'$ where $P' = P \parallel P_n$, $Q' = Q \parallel Q_n$ and $M' = \min\{t_i | t_i \in I'\}$ with $I' = I \cup \{n\}$ assuming the induction hypothesis that $P \Leftrightarrow_M Q$. From $P \Leftrightarrow_M Q$ we get by Lemma 2 that $P \parallel P_n \Leftrightarrow_M Q \parallel P_n$. Also, by Lemma 2 we have that $Q \parallel P_n \Leftrightarrow_{t_n} Q \parallel Q_n$ since $P_n \Leftrightarrow_{t_n} Q_n$ by assumption. Hence, by Proposition 4 we have that $P \parallel P_n \Leftrightarrow_{\min\{M, t_n\}} Q \parallel Q_n$. But $\min\{M, t_n\} = M'$ and so $P' \Leftrightarrow_{M'} Q'$.

2. Also by induction. The argument mimicks the previous one, replacing $\|$ with $+$.

C Legitimacy

Proposition 5. *For any $P, P' \in \mathcal{P}$, $d \in \mathbb{R}_0^+$, $\alpha \in \mathcal{A}$, and any substitution σ,*

1. *if $P \xrightarrow{\alpha} P'$ then $P\sigma \xrightarrow{\sigma(\alpha)} P'\sigma$*
2. *if $P \overset{d}{\rightsquigarrow} P'$ then $P\sigma \overset{d}{\rightsquigarrow} P'\sigma$*
3. *if $P\sigma \xrightarrow{\alpha} P'$ then there is a α' and a Q such that $P \xrightarrow{\alpha'} Q$ with $\alpha = \sigma(\alpha')$ and $P' = Q\sigma$.*
4. *if $P\sigma \overset{d}{\rightsquigarrow} P'$ then there is a Q such that $P \overset{d}{\rightsquigarrow} Q$ and $P' = Q\sigma$*

Proof. We'll show only the first two items. By induction on the derivation of $P \xrightarrow{\alpha} P'$ and $P \overset{d}{\rightsquigarrow} P'$ respectively. We proceed by case analysis on the structure of P.

Case 1. $P \equiv \surd$. In this case, P has no transitions, so *1* is vacuously true. For 2 note that $\surd\sigma \equiv \surd$ and $\surd \overset{d}{\rightsquigarrow} \surd$, hence $P\sigma \overset{d}{\rightsquigarrow} P'\sigma$.

Case 2. $P \equiv x!E$. This has only one transition: $P \xrightarrow{\alpha} \surd$ where $\alpha = x!eval(E)$. By definition of substitution, $P\sigma \equiv \sigma(x)!E\sigma$. But by (TRIG), $P\sigma \xrightarrow{\alpha'} \surd$, where $\alpha' \equiv \sigma(x)!eval(E\sigma) = \sigma(\alpha)$. For evolution, suppose that $P \overset{d}{\rightsquigarrow} P'$. Then $P' \equiv P$ and so $P'\sigma \equiv P\sigma$. By (TTRIG), $P\sigma \overset{d}{\rightsquigarrow} P\sigma \equiv P'\sigma$.

Case 3. $P \equiv \Delta E \to Q$. We have that $P\sigma \equiv \Delta E\sigma \to Q\sigma$. *1)* The only transition that P has is from (DELAY), which implies that if $P \xrightarrow{\alpha} P'$ then $\alpha = \tau$, $eval(E) = 0$, and $P' \equiv Q$. This means that $eval(E\sigma) = 0$, and so, by (DELAY), $P\sigma \xrightarrow{\alpha} P'\sigma$. *2)* If $P \overset{d}{\rightsquigarrow} P'$ then $0 \leqslant d \leqslant eval(E)$ and $P' \equiv \Delta(E - d) \to Q$. So, $P'\sigma \equiv \Delta(E\sigma - d) \to Q\sigma$. But this means that $P\sigma \overset{d}{\rightsquigarrow} P'\sigma$ by (TDELAY).

Case 4. $P \equiv \nu x.Q$. Let σ be any substitution. We can assume, without loss of generality that x does not occur in σ[16]. We now show each item:
1) A transition $P \xrightarrow{\alpha} P'$ must have been derived by (NEW). Then $P' \equiv \nu x.Q'$ for some Q' such that $Q \xrightarrow{\alpha} Q'$ with $x\mathrm{n}(\alpha)$. Then by induction hypothesis, $Q\sigma \xrightarrow{\sigma(\alpha)} Q'\sigma$. Hence, by (NEW), $\nu x.Q\sigma \xrightarrow{\sigma(\alpha)} \nu x.Q'\sigma$. But we assumed that x does not occur in σ, so we have $(\nu x.Q)\sigma \xrightarrow{\sigma(\alpha)} (\nu x.Q')\sigma$.
2) An evolution $P \overset{d}{\rightsquigarrow} P'$must have been derived with (TNEW). Hence $P' \equiv$

[16] If x occurs in σ, we can simply rename x for a fresh name.

$\nu x.Q'$ for some Q' such that $Q \overset{d}{\rightsquigarrow} Q'$. By induction we have that $Q\sigma \overset{d}{\rightsquigarrow} Q'\sigma$, and so, by (TNEW), we have $\nu x.Q\sigma \overset{d}{\rightsquigarrow} \nu x.Q'\sigma$, which implies $(\nu x.Q)\sigma \overset{d}{\rightsquigarrow} (\nu x.Q')\sigma$.

Case 5. $P \equiv P_l \parallel P_r$. Let σ be any substitution. Assume, without loss of generality that names of σ (either as sources or as targets) are not bound names of P, and also that the bound names of P_l are not free in P_r.[17]
1) The transition $P \xrightarrow{\alpha} P'$ must have been derived (PAR) or (COMM).
Sub-case 1) It was derived by (PAR). Then $P' \equiv P_l' \parallel P_r$ for some P_l' such that $P_l \xrightarrow{\alpha} P_l'$ and $\mathrm{bn}(\alpha) \cap \mathrm{fn}(P_r) = \emptyset$. We know that $\mathrm{bn}(\alpha) \subseteq \mathrm{bn}(P_l)$ and so these bound names do not ocurr in σ by our assumption. Hence $\mathrm{bn}(\sigma(\alpha)) = \mathrm{bn}(\alpha)$. Since the target names of σ are not bound names in P (and therefore in P_l) then $\mathrm{fn}(P_r\sigma) \cap \mathrm{bn}(P_l) = \emptyset$ and therefore $\mathrm{fn}(P_r\sigma) \cap \mathrm{bn}(\sigma(\alpha)) = \emptyset$. This, and the induction hypothesis, $P_l\sigma \xrightarrow{\sigma(\alpha)} P_l'\sigma$, imply, by (PAR), that $P\sigma \equiv P_l\sigma \parallel P_r\sigma \xrightarrow{\sigma(\alpha)} P_l'\sigma \parallel P_r\sigma \equiv P'\sigma$.
Sub-case 2) It was derived by (COMM). Then $\alpha = \tau$ and $P' \equiv P_l' \parallel P_r'$ for some P_l' and P_r' such that $P_l \xrightarrow{x!v} P_l'$ and $P_r \xrightarrow{x?v} P_r'$ for some x and v. By induction hypothesis, $P_l\sigma \xrightarrow{\sigma(x!v)} P_l'\sigma$ and $P_r\sigma \xrightarrow{\sigma(x?v)} P_r'\sigma$. But since $\sigma(x!v) = \sigma(x)!\sigma(v)$ and $\sigma(x?v) = \sigma(x)?\sigma(v)$,[18] we conclude, by (COMM), that $P\sigma \equiv P_l\sigma \parallel P_r\sigma \xrightarrow{\sigma(\alpha)} P_l'\sigma \parallel P_r'\sigma \equiv P'\sigma$ since $\sigma(\tau) = \tau$.
2) The evolution $P \overset{d}{\rightsquigarrow} P'$ must have been derived by (TPAR). Hence $P' \equiv P_l' \parallel P_r'$ for some P_l' and P_r' such that $P_l \overset{d}{\rightsquigarrow} P_l'$ and $P_r \overset{d}{\rightsquigarrow} P_r'$. By induction hypothesis, $P_l\sigma \overset{d}{\rightsquigarrow} P_l'\sigma$ and $P_r\sigma \overset{d}{\rightsquigarrow} P_r'\sigma$, from which we conclude, by (TPAR), that $P\sigma \equiv P_l\sigma \parallel P_r\sigma \overset{d}{\rightsquigarrow} P_l'\sigma \parallel P_r'\sigma \equiv P'\sigma$.

Case 6. $P \equiv \Sigma_{i\in I}\beta_i \rightarrow P_i$ where each $\beta_i = x_i?F_i@y_i$. Without loss of generality, we assume that the bound names of the guards do not occur in σ[19]. Then we have that $P\sigma \equiv \Sigma_{i\in I}\beta_i\sigma \rightarrow P_i\sigma$, where each $\beta_i\sigma = \sigma(x_i)?F_i@y_i$.
1) A transition $P \xrightarrow{\alpha} P'$ must have been derived with (CHOICE), and so $\alpha = x_k?v$ for some x_k and v, and $P' \equiv P_k\sigma''$ for some substitution $\sigma'' = \sigma' \cup \{0/y_k\}$ where $\sigma' = match(F_k, v, \emptyset) \neq \emptyset$ with $\beta_k = x_k?F_k@y_k$. Let $\varsigma' \overset{def}{=} match(F_k, \sigma(v), \emptyset)$ and $\varsigma'' \overset{def}{=} \varsigma' \cup \{0/y_k\}$. Since the structure and constants of $\sigma(v)$ are the same of those of v, we have that σ' and ς' are essentially the same, modulo σ, this is, $\varsigma'(z) = \sigma(\sigma'(z))$ for any $z \in \mathcal{N}$, and $\varsigma' = \emptyset \Leftrightarrow \sigma' = \emptyset$. Since this transition occurred, $\sigma' \neq \emptyset$ and therefore $\varsigma' \neq \emptyset$. This allows us to conclude that $P\sigma \xrightarrow{\sigma(\alpha)}$

[17]If this is not the case, as usual we can rename bound names of P with fresh names not occurring in σ or P_r.

[18]Note that this is a substitution σ applied to an input *action* $\alpha = x?v$, not to an inpur *process* $x?y \rightarrow P$. Substitution for processes is defined in the standard way to avoid capture of free variables. The key is that in a transition $P \xrightarrow{x?v} P'$ the action $x?v$ represents the action provided by the environment, i.e., the value v is not the bound name in P, but the data sent through channel x. Hence, v is not bound in the action, whereas y is bound in $x?y \rightarrow P$.

[19]If they occur in σ, we can simply rename them.

P'' where $\sigma(\alpha) = \sigma(x_k)?\sigma(v)$ and $P'' \equiv P_k\sigma\varsigma''$ by (CHOICE). We claim that $\sigma\varsigma'' = \sigma''\sigma$. To see this, consider any name $z \in \mathcal{N}$. We have three possibilities: a) $z = y_k$, b) $z \in \mathrm{n}(F_k)$ or c) $z \notin \mathrm{n}(F_k) \cup \{y_k\}$. In each case we have that the substitutions coincide: a) $\varsigma''(\sigma(y_k)) = \varsigma''(y_k)$ since y_k does not occur in σ and a substitution is always defined to be the identity on names which are not in its source. Hence $\varsigma''(\sigma(y_k)) = 0$ by definition of ς''. On the other hand, $\sigma(\sigma''(y_k)) = \sigma(0) = 0$, by definition of σ''. b) $\varsigma''(\sigma(z)) = \varsigma''(z)$ since z does not occur in σ's source; but we know that $\varsigma'(z) = \sigma(\sigma'(z))$, hence $\varsigma''(\sigma(z)) = \sigma(\sigma'(z))$. c) $\varsigma''(\sigma(z)) = \sigma(z)$ since none of the variables in $\mathrm{n}(F_k) \cup \{y_k\}$ occur in the targets of σ, and therefore ς'' acts as the identity on anything that does not have names in $\mathrm{n}(F_k) \cup \{y_k\}$. On the other hand, $\sigma(\sigma''(z)) = \sigma(z)$ for the same reason: σ'' is the identity on names not in $\mathrm{n}(F_k) \cup \{y_k\}$. In the three cases we have seen that $\sigma\varsigma'' = \sigma''\sigma$, and therefore $P_k\sigma\varsigma'' \equiv P_k\sigma''\sigma$ so we have that $P'' \equiv P'\sigma$ and therefore $P\sigma \xrightarrow{\sigma(\alpha)} P'\sigma$.
2) An evolution $P \overset{d}{\rightsquigarrow} P'$ must have been derived with (TCHOICE), so $P' \equiv \Sigma_{i\in I}\beta_i \to P_i'$ where $P_i' \equiv P_i\{^{y_i+d}/_{y_i}\}$. So $P'\sigma \equiv \Sigma_{i\in I}\beta_i\sigma \to P_i'\sigma$. On the other hand we have, by (TCHOICE), that $P\sigma \overset{d}{\rightsquigarrow} \Sigma_{i\in I}\beta_i\sigma \to Q_i$ where $Q_i \equiv P_i\sigma\{^{y_i+d}/_{y_i}\}$. But since none of the y_i occur in σ, we have that $\{^{y_i+d}/_{y_i}\}\sigma = \sigma\{^{y_i+d}/_{y_i}\}$, so $Q_i \equiv P_i\{^{y_i+d}/_{y_i}\}\sigma \equiv P_i'\sigma$, and therefore $P\sigma \overset{d}{\rightsquigarrow} P'\sigma$.

Case 7. $P \equiv A(\vec{y})$. Let σ be any substitution. We assume that the definition of A is *closed*, *i.e.* if $A(\vec{x}) \overset{def}{=} Q$ then $\mathrm{fn}(Q) \subseteq \vec{x}$.
1) A transition $P \xrightarrow{\alpha} P'$ must have been obtained by (CNGR), and therefore $Q\{\vec{y}/\vec{x}\} \xrightarrow{\alpha} Q'$ for some Q and Q'. By induction hypothesis we have that $Q\{\vec{y}/\vec{x}\}\sigma \xrightarrow{\sigma(\alpha)} Q'\sigma$. Without loss of generality we can assume that none of the names in $\vec{x}$ occur in σ[20]. Hence $\{\vec{y}/\vec{x}\}\sigma = \sigma\{\sigma(\vec{y})/\vec{x}\}$ and so $Q\{\vec{y}/\vec{x}\}\sigma = Q\sigma\{\sigma(\vec{y})/\vec{x}\}$, but the definition of A is closed, $\mathrm{fn}(Q) \subseteq \vec{x}$ which means that $Q\sigma = Q$ since none of the names in $\vec{x}$ occurs in σ. Therefore $Q\{\vec{y}/\vec{x}\}\sigma = Q\{\sigma(\vec{y})/\vec{x}\}$ and so $Q\{\sigma(\vec{y})/\vec{x}\} \xrightarrow{\sigma(\alpha)} Q'\sigma$, from which we conclude, by (CNGR), that $A(\sigma(\vec{y})) \xrightarrow{\sigma(\alpha)} Q'\sigma$, or in other words $P\sigma \xrightarrow{\sigma(\alpha)} P'\sigma$ where $P' \equiv Q'$.
2) The evolution $P \overset{d}{\rightsquigarrow} P'$ must be derived by (TCNGR), which implies that $d = 0$ and $P' \equiv P$. Hence $P\sigma \equiv A(\sigma(\vec{y})) \overset{0}{\rightsquigarrow} A(\sigma(\vec{y})) \equiv P'\sigma$.

□

Lemma 3. *Let $P \in \mathcal{P}$, and σ any substitution. If P is legitimate, so is $P\sigma$.*

Proof. Let γ_σ be any execution of $P\sigma$:

$$\gamma_\sigma = P\sigma \overset{d_0}{\rightsquigarrow} Q_0' \xrightarrow{\alpha_0} Q_1 \overset{d_1}{\rightsquigarrow} Q_1' \xrightarrow{\alpha_1} Q_2 \overset{d_2}{\rightsquigarrow} Q_2' \xrightarrow{\alpha_2} \cdots$$

Then, by Proposition 5, this execution has the form

$$P\sigma \overset{d_0}{\rightsquigarrow} P_0'\sigma \xrightarrow{\sigma(\alpha_0')} P_1\sigma \overset{d_1}{\rightsquigarrow} P_1'\sigma \xrightarrow{\sigma(\alpha_1')} P_2\sigma \overset{d_2}{\rightsquigarrow} P_2'\sigma \xrightarrow{\sigma(\alpha_2')} \cdots$$

[20] If they occur we can rename them in Q by fresh names.

for some $P'_0, P_1, P'_1, P_2, P'_2, ...$ and some $\alpha'_0, \alpha'_1, \alpha'_2, ...$ such that

$$\gamma = P \overset{d_0}{\rightsquigarrow} P'_0 \xrightarrow{\alpha'_0} P_1 \overset{d_1}{\rightsquigarrow} P'_1 \xrightarrow{\alpha'_1} P_2 \overset{d_2}{\rightsquigarrow} P'_2 \xrightarrow{\alpha'_2} \cdots$$

Note that $\text{len}(\gamma) = \text{len}(\gamma_\sigma)$, since there is a one-to-one relation between evolutions and transitions in γ with those in γ_σ. Furthermore, $\text{dur}(\gamma) = \text{dur}(\gamma_\sigma)$ since the substitution preserves the amount of time in each evolution. But we assumed that P is legitimate, and therefore, γ is a legitimate execution. Hence either $\text{len}(\gamma) < \infty$ or $\text{len}(\gamma) = \infty$ and $\text{dur}(\gamma) = \infty$. But this is to say that either $\text{len}(\gamma_\sigma) < \infty$ or $\text{len}(\gamma_\sigma) = \infty$ and $\text{dur}(\gamma_\sigma) = \infty$, *i.e.* that $P\sigma$ is legitimate. $\square$

Theorem 2. ***(Sufficient conditions for legitimacy)*** *Let D be a finite set of process definitions and P a process which invokes only definitions in D. If all definitions in D are well-timed then P is legitimate.*

Proof. By induction on the structure of P.

Case 1. $P \equiv \surd$ or $P \equiv x!E$. These processes have no infinite executions, so they are always legitimate.

Case 2. $P \equiv \Delta E \to Q$. Assume the statement holds for Q. Let γ be any execution of P. If it is finite, then it is legitimate, by definition. Assume that γ is infinite. Then, it must begin by $P \overset{e}{\rightsquigarrow} P' \xrightarrow{\tau} Q$ where $e = eval(E)$. Let γ_1 be the remainder of the execution, starting from Q. By induction hypothesis, γ_1 must be legitimate, so $\text{dur}(\gamma_1) = \infty$. But $\text{dur}(\gamma) = e + \text{dur}(\gamma_1)$, so $\text{dur}(\gamma) = \infty$ as required.

Case 3. $P \equiv \nu x.Q$. Assume the statement holds for Q. Let γ be any execution of P. If it is finite, then it is legitimate, by definition. Assume that γ is infinite. Then, it must begin with $P \overset{d}{\rightsquigarrow} P'_1 \xrightarrow{\alpha_1} P_1$. The evolution must have been obtained by (TNEW), which means that $P'_1 \equiv \nu x.Q'_1$ for some Q'_1 and $Q \overset{d}{\rightsquigarrow} Q'_1$ by a shorter inference. Since $P'_1 \equiv \nu x.Q'_1$, the transition must have been obtained by (NEW), which implies that $P_1 \equiv \nu x.Q_1$ for some Q_1 such that $Q'_1 \xrightarrow{\alpha_1} Q_1$. By repeating this over γ we obtain an execution γ_1 starting from Q. By induction hypothesis, γ_1 must be legitimate, so $\text{dur}(\gamma_1) = \infty$. But $\text{dur}(\gamma) = \text{dur}(\gamma_1)$, because it has exactly the same evolutions, so $\text{dur}(\gamma) = \infty$ as required.

Case 4. $P \equiv P_l \parallel P_r$. Assume the statement holds for P_l and P_r. Let γ be any execution of P. If it is finite, then it is legitimate, by definition. Assume that γ is infinite. Then it must begin with a pair $P \overset{d}{\rightsquigarrow} P'_l \parallel P'_r$ and $P'_l \parallel P'_r \xrightarrow{\alpha} P^{(1)}_l \parallel P^{(1)}_r$. The evolution must have been the result of (TPAR), and therefore $P_l \overset{d}{\rightsquigarrow} P'_l$ and $P_r \overset{d}{\rightsquigarrow} P'_r$. The transition may have been the result of either (PAR) or (COMM). If it was the result of (COMM), then we have that $P'_l \xrightarrow{\alpha_l} P^{(1)}_l$ and $P'_r \xrightarrow{\alpha_r} P^{(1)}_r$ for some complementary actions α_l and α_r. If it was the result of a (PAR) rule, then we have that either $P'_l \xrightarrow{\alpha} P^{(1)}_l$ or $P'_r \xrightarrow{\alpha} P^{(1)}_r$. In either case we can build a pair of infinite executions γ_l and γ_r each beginning with P_l and P_r respectively. But by induction hypothesis, γ_l and γ_r are legitimate, so $\mathrm{dur}(\gamma_l) = \infty$ and $\mathrm{dur}(\gamma_r) = \infty$. But the evolutions of P and its derivatives are exactly the same as those of P_l and P_r, since evolution of a parallel composition is the evolution of its parts by equal amounts of time as stated by the (TPAR) rule. This means that $\mathrm{dur}(\gamma) = \mathrm{dur}(\gamma_l) = \mathrm{dur}(\gamma_r) = \infty$, and therefore γ is legitimate.

Case 5. $P \equiv \Sigma_{i\in I}\beta_i \to P_i$. Assume the statement holds for each P_i. Let γ be any execution of P. If it is finite, then it is legitimate, by definition. Assume that γ is infinite. Then, it must begin with $P \overset{d}{\rightsquigarrow} P' \xrightarrow{x_k?v} P'_k\sigma$ for some d, x_k and v, where $\beta_k = x_k?F_k@y_k$, $\sigma = match(F_k, v, \emptyset) \cup \{^0/_{y_k}\}$, $P' = \sum_{i\in I} \beta_i \to P_i\{^{y_i+d}/_{y_i}\}$ and $P'_k = P_k\{^{y_k+d}/_{y_k}\}$. Let γ_1 be the remainder of the execution, starting from $P'_k\sigma$. By induction hypothesis, any execution starting from P_k must be legitimate, hence, by Lemma 3, γ_1 must be legitimate as well, so $\mathrm{dur}(\gamma_1) = \infty$. But $\mathrm{dur}(\gamma) = d + \mathrm{dur}(\gamma_1)$, so $\mathrm{dur}(\gamma) = \infty$ as required.

Case 6. $P \equiv A(\vec{y})$. Let γ be any execution of P. If it is finite, then it is legitimate, by definition. Assume that γ is infinite. Then, since D is finite, there must be a process definition $B \in D$ which is invoked infinitely often, *i.e.* γ contains an infinite number of occurrences $B(\vec{z}_1)$, $B(\vec{z}_2)$, $B(\vec{z}_3)$, ... Hence γ has the form

$$P \overset{\gamma_0}{\Longrightarrow} B(\vec{z}_1) \overset{\gamma_1}{\Longrightarrow} B(\vec{z}_2) \overset{\gamma_2}{\Longrightarrow} B(\vec{z}_3) \overset{\gamma_3}{\Longrightarrow} \cdots$$

where each γ_i is a finite execution between each invocation of B. Since the definition $B(\vec{x}) \overset{def}{=} Q$ is well-timed, there is a $b > 0$ such that for all $\vec{z}_k$, $md_B(Q\{^{\vec{z}_k}/_{\vec{x}}\}) \geqslant b$. But this means that there is a $b > 0$ such that for all $k \geqslant 1$, $\mathrm{dur}(\gamma_k) \geqslant md_B(Q\{^{\vec{z}_k}/_{\vec{x}}\}) \geqslant b$. Hence $\mathrm{dur}(\gamma) = \Sigma_{k=0}^{\infty}\mathrm{dur}(\gamma_k) \geqslant \mathrm{dur}(\gamma_0) + \Sigma_{k=1}^{\infty} b = \infty$ as required.

□

Lemma 4. *Let $P, Q \in \mathcal{P}$. If $P \leftrightharpoons_t Q$ and γ_P is an execution beginning with P such that* $\mathrm{dur}(\gamma_P) < t$*, then there is an execution γ_Q, starting at Q, such that* $\mathrm{atr}(\gamma_P) = \mathrm{atr}(\gamma_Q)$, $\mathrm{len}(\gamma_P) = \mathrm{len}(\gamma_Q)$ *and* $\mathrm{dur}(\gamma_P) = \mathrm{dur}(\gamma_Q)$.

Proof. The execution γ_P must be of the form

$$P \overset{d_0}{\rightsquigarrow} P'_0 \xrightarrow{\alpha_0} P_1 \overset{d_1}{\rightsquigarrow} P'_1 \xrightarrow{\alpha_1} P_2 \overset{d_2}{\rightsquigarrow} P'_2 \xrightarrow{\alpha_2} \cdots$$

We can show, by induction on i, that for each pair evolution/transition $P_i \overset{d_i}{\rightsquigarrow} P'_i \xrightarrow{\alpha_i} P_{i+1}$ there is a pair evolution/transition $Q_i \overset{d_i}{\rightsquigarrow} Q'_i \xrightarrow{\alpha_i} Q_{i+1}$ where $P_i \leftrightharpoons_{u_i} Q_i$ with $u_i = t - \Sigma_{j=0}^{i-1} d_j$. The first pair is $P \overset{d_0}{\rightsquigarrow} P'_0 \xrightarrow{\alpha_0} P_1$, where $d_0 < t$. But since $P \leftrightharpoons_t Q$, then $Q \overset{d_0}{\rightsquigarrow} Q'_1$ with $P'_0 \leftrightharpoons_{t-d_0} Q'_0$, which in turn implies that $Q'_0 \xrightarrow{\alpha_0} Q_1$ with $P_1 \leftrightharpoons_{t-d_0} Q_1$. So we can form $Q \overset{d_0}{\rightsquigarrow} Q'_0 \xrightarrow{\alpha_0} Q_1$. So in general, assuming that the statement holds for all $k < i$, we show it for i. In particular, we assume it holds for $k = i-1$: $P_{i-1} \leftrightharpoons_{u_{i-1}} Q_{i-1}$ with $u_{i-1} = t - \Sigma_{j=0}^{i-2} d_j$. We have that $P_{i-1} \overset{d_{i-1}}{\rightsquigarrow} P'_{i-1} \xrightarrow{\alpha_{i-1}} P_i$. But note that $\Sigma_{j=0}^{i-1} d_j \leqslant \mathrm{dur}(\gamma_P) < t$ and so $d_{i-1} + \Sigma_{j=0}^{i-2} d_j < t$. Therefore $d_{i-1} < u_{i-1}$. Hence, from $P_{i-1} \Leftrightarrow_{u_{i-1}} Q_{i-1}$ and $P_{i-1} \overset{d_{i-1}}{\rightsquigarrow} P'_{i-1}$ we deduce that $Q_{i-1} \overset{d_{i-1}}{\rightsquigarrow} Q'_{i-1}$ with $P'_{i-1} \Leftrightarrow_{u'_{i-1}} Q'_{i-1}$ where $u'_{i-1} = u_{i-1} - d_{i-1} = t - (d_{i-1} + \Sigma_{j=0}^{i-2} d_j) = t - \Sigma_{j=0}^{i-1} d_j$. This in turn implies, together with $P'_{i-1} \xrightarrow{\alpha_{i-1}} P_i$, that $Q'_{i-1} \xrightarrow{\alpha_{i-1}} Q_i$ and $P_i \Leftrightarrow_{u_i} Q_i$ where $u_i = u'_{i-1} = t - \Sigma_{j=0}^{i-1} d_j$, and so $Q_{i-1} \overset{d_{i-1}}{\rightsquigarrow} Q'_{i-1} \xrightarrow{\alpha_{i-1}} Q_i$. So we can build an execution γ_Q starting with Q with the form

$$Q \overset{d_0}{\rightsquigarrow} Q'_0 \xrightarrow{\alpha_0} Q_1 \overset{d_1}{\rightsquigarrow} Q'_1 \xrightarrow{\alpha_1} Q_2 \overset{d_2}{\rightsquigarrow} Q'_2 \xrightarrow{\alpha_2} \cdots$$

which has the same trace and therefore, $\mathrm{atr}(\gamma_P) = \mathrm{atr}(\gamma_Q)$, $\mathrm{len}(\gamma_P) = \mathrm{len}(\gamma_Q)$ and $\mathrm{dur}(\gamma_P) = \mathrm{dur}(\gamma_Q)$. □

Theorem 3. *Let $P, Q \in \mathcal{P}$. If $P \Leftrightarrow_t Q$ for all $t \in \mathbb{R}_0^+$ then P is legitimate if and only if Q is legitimate.*

Proof. It is enough to prove that if Q is legitimate then so is P. We prove this by contradiction. Assume that Q is legitimate, but P is not. Then there is an infinite execution γ_P starting at P such that $\mathrm{dur}(\gamma_P) = t < \infty$ for some $t \in \mathbb{R}_0^+$. But $P \Leftrightarrow_t Q$ by assumption, so by Lemma 4, there must be an execution γ_Q beginning with Q such that $\mathrm{len}(\gamma_P) = \mathrm{len}(\gamma_Q)$ and $\mathrm{dur}(\gamma_P) = \mathrm{dur}(\gamma_Q)$. But this is to say that $\mathrm{len}(\gamma_Q) = \infty$ and $\mathrm{dur}(\gamma_Q) = t < \infty$, *i.e.* Q is illegitimate. A contradiction. □

D Abstract Machine Soundness

Lemma 5. ***(Correct time-slot execution)***
If $(H, T) \rightarrow_t (H', T')$ then $[\![(H, T)]\!]_t \Longrightarrow [\![(H', T')]\!]_t$

Proof. We proceed by case analysis on the rules for time-slot transitions. In this proof we'll use the following notational shortcuts: $A = \Pi alllisteners(H, t)$, $B = \Pi alltriggers(H)$

Case 1. Rule (NIL). $(H, \surd :: T) \rightarrow_t (H, T)$

$$\begin{array}{rcl} [\![(H, \surd :: T)]\!]_t & = & \nu\tilde{x}.([\![H]\!]_t \parallel [\![\surd :: T]\!]) \\ & = & \nu\tilde{x}.([\![H]\!]_t \parallel \surd \parallel [\![T]\!]) \\ & \equiv & \nu\tilde{x}.([\![H]\!]_t \parallel [\![T]\!]) \\ & = & [\![(H, T)]\!]_t \end{array}$$

Case 2. Rule (SPAWN$_1$). $(H, (P_1 \parallel P_2) :: T) \rightarrow_t (H, T :: P_1 :: P_2)$

$$\begin{array}{rcl} [\![(H, (P_1 \parallel P_2) :: T)]\!] & = & \nu\tilde{x}.([\![H]\!]_t \parallel [\![(P_1 \parallel P_2) :: T]\!]) \\ & = & \nu\tilde{x}.([\![H]\!]_t \parallel (P_1 \parallel P_2) \parallel [\![T]\!]) \\ & \equiv & \nu\tilde{x}.([\![H]\!]_t \parallel [\![T]\!] \parallel P_1 \parallel P_2) \\ & = & [\![(H, T)]\!]_t \end{array}$$

Case 3. Rule (SPAWN$_2$). As the previous case.

Case 4. Rule (RESTR). $(H, \nu x.P :: T) \to_t (H\{k \mapsto \emptyset\}, P\{{}^{k}/_{x}\} :: T)$ with k fresh. First note that if $H' = H\{k \mapsto \emptyset\}$ then we have that $H'(k) = \emptyset$ and therefore $alts(H'(k), c, k, t) = \emptyset$ for any c and t. This implies that $alllisteners(H', t) = alllisteners(H, t)$ and therefore $[\![H']\!]_t = [\![H]\!]_t$. Hence,

$$\begin{aligned}
[\![(H, \nu x.P :: T)]\!]_t &= \nu\tilde{x}.([\![H]\!]_t \parallel [\![\nu x.P :: T]\!]) \\
&= \nu\tilde{x}.([\![H]\!]_t \parallel \nu x.P \parallel [\![T]\!]) \\
&\equiv \nu\tilde{x}.([\![H]\!]_t \parallel \nu k.P\{{}^{k}/_{x}\} \parallel [\![T]\!]) \\
&\equiv \nu\tilde{x}, k.([\![H]\!]_t \parallel P\{{}^{k}/_{x}\} \parallel [\![T]\!]) \\
&\equiv \nu\tilde{x}, k.([\![H\{k \mapsto \emptyset\}]\!]_t \parallel P\{{}^{k}/_{x}\} \parallel [\![T]\!]) \\
&= [\![(H\{k \mapsto \emptyset\}, P\{{}^{k}/_{x}\} :: T)]\!]_t
\end{aligned}$$

Case 5. Rule (OUT-F). $(H, x!E :: T) \to_t (H\{x \mapsto Q \cup \{!v\}\}, T)$ given that $inms(v, Q) = \emptyset$ where $H(x) = Q$ and $v = eval(E)$.
Let $H' = H\{x \mapsto Q \cup \{!v\}\}$. So $outputs(H'(x)) = \{!v\} \cup outputs(Q) = \{!v\} \cup outputs(H(x))$ and therefore $triggers(H', x) = \{x!v\} \cup triggers(H, x)$ which entails that $alltriggers(H') = \{x!v\} \cup alltriggers(H)$, and so $B' = \Pi alltriggers(H') = x!v \parallel \Pi alltriggers(H) = x!v \parallel B$. On the other hand, notice that $A' = \Pi alllisteners(H', t) = \Pi alllisteners(H, t) = A$ because H' doesn't add any new listeners. Hence,

$$\begin{aligned}
[\![(H, x!E :: T)]\!]_t &= \nu\tilde{x}.([\![H]\!]_t \parallel [\![x!E :: T]\!]) \\
&= \nu\tilde{x}.([\![H]\!]_t \parallel x!E \parallel [\![T]\!]) \\
&= \nu\tilde{x}.(A \parallel B \parallel x!E \parallel [\![T]\!]) \\
&\equiv \nu\tilde{x}.(A' \parallel B' \parallel [\![T]\!]) \\
&= \nu\tilde{x}.([\![H']\!]_t \parallel [\![T]\!]) \\
&= [\![(H', T)]\!]_t
\end{aligned}$$

Case 6. Rule (OUT-S). $(H, x!E :: T) \to_t (remalts(H, c), P(\sigma \cup \{{}^{t-u}/_{y}\}) :: T)$ given that $(O, \sigma) \in inms(v, Q) \neq \emptyset$ where $v = eval(E)$, $H(x) = Q$ and $O = ?(F, y, P, u, c)$.
Let $H' = remalts(H, c)$. Since $(O, \sigma) \in inms(v, Q)$, then we know that $O \in inputs(Q)$ and $\sigma = match(F, v, \emptyset) \neq \emptyset$. Hence, there is a listener on x, with (unique) tag c, with the form $L = \sum branches(H, c, t) = \sum \cup_{z \in H} alts(H(z), c, z, t) = x?F@y \to P\{{}^{y+(t-u)}/_{y}\} + D$ where D are the remaning alternatives. By using (CHOICE) we can deduce $L \xrightarrow{x?v} P\{{}^{y+(t-u)}/_{y}\}\sigma\{{}^{0}/_{y}\} = P(\sigma \cup \{{}^{t-u}/_{y}\})$. Then, by (COMM) we obtain $L \parallel x!E \xrightarrow{\tau} P(\sigma \cup \{{}^{t-u}/_{y}\}) \parallel \surd \equiv P(\sigma \cup \{{}^{t-u}/_{y}\})$. Note also that H' is H with all c-tagged observers removed, so $alltriggers(H') = alltriggers(H)$ and $alllisteners(H', t) = alllisteners(H, t) \backslash \{L\}$ and hence, $B' = \Pi alltriggers(H') = B$ and $A' = \Pi alllisteners(H', t) = \Pi alllisteners(H, t) \backslash \{L\}$, i.e., the set of listeners without L, so $A = A' \parallel L$. By applying (PAR) we obtain,

$B \parallel A \parallel x!E \equiv B \parallel A' \parallel L \parallel x!E \xrightarrow{\tau} B' \parallel A' \parallel P(\sigma \cup \{^{t-u}/_y\})$. Hence,

$$\begin{array}{rcl} [\![(H, x!E :: T)]\!]_t & = & \nu\tilde{x}.([\![H]\!]_t \parallel [\![x!E :: T]\!]) \\ & = & \nu\tilde{x}.([\![H]\!]_t \parallel x!E \parallel [\![T]\!]) \\ & = & \nu\tilde{x}.(A \parallel B \parallel x!E \parallel [\![T]\!]) \\ & \xrightarrow{\tau} & \nu\tilde{x}.(A' \parallel B' \parallel P(\sigma \cup \{^{t-u}/_y\}) \parallel [\![T]\!]) \\ & \equiv & \nu\tilde{x}.([\![H']\!]_t \parallel [\![P(\sigma \cup \{^{t-u}/_y\}) :: T]\!]) \\ & = & [\![(H', P(\sigma \cup \{^{t-u}/_y\}) :: T)]\!]_t \end{array}$$

Case 7. Rule (INP-F). $(H, L :: T) \to_t (H\{x_i \mapsto Q_i \cup \{?(F_i, y_i, P_i, t, c)\}\}_{i\in I}, T)$ where $L = \sum_{i\in I} x_i?F_i@y_i \to P_i$, c is fresh, and for each $i \in I$, $H(x_i) = Q_i$ and $outms(F_i, Q_i) = \emptyset$.
Let $H' = H\{x_i \mapsto Q_i \cup \{?(F_i, y_i, P_i, t, c)\}\}_{i\in I}$. First note that $B' = \Pi alltriggers(H') = B$, sonce no triggers are being added. As for listeners, note that for each $i \in I$, $inputs(H'(x_i)) = \{?(F_i, y_i, P_i, t, c)\} \cup inputs(H(x_i))$, and therefore, $alts(H'(x_i), c, x_i, t) = \{x_i?F_i@y_i \to P_i | ?(F_i, y_i, P_i, t, c) \in inputs(H'(x_i))\}$ since $\{^{y_i+(t-t)}/_{y_i}\} = \{^{y_i+0}/_{y_i}\} = \{^{y_i}/_{y_i}\}$ is the identity substitution. Hence $branches(H', c, t) = \{x_i?F_i@y_i \to P_i\}$. Note that since c is fresh, there is no observer O in H such that $tag(O) = c$. Therefore for all tags c' in H, $c' \neq c$ and so $alltags(H') = alltags(H) \cup \{c\}$, and also $branches(H', c', t) = branches(H, c', t)$. Hence,

$$\begin{array}{rcl} A' & = & \Pi alllisteners(H', t) \\ & = & \Pi\{\sum branches(H', c', t) | c' \in alltags(H')\} \\ & = & \Pi\{\sum branches(H', c', t) | c' \in alltags(H) \cup \{c\}\} \\ & = & \sum branches(H', c, t) \parallel \Pi\{\sum branches(H', c', t) | c' \in alltags(H), c' \neq c\} \\ & = & L \parallel A \end{array}$$

So it follows that

$$\begin{array}{rcl} [\![(H, L :: T)]\!]_t & = & \nu\tilde{x}.([\![H]\!]_t \parallel [\![L :: T]\!]) \\ & = & \nu\tilde{x}.([\![H]\!]_t \parallel L \parallel [\![T]\!]) \\ & = & \nu\tilde{x}.(A \parallel B \parallel L \parallel [\![T]\!]) \\ & \equiv & \nu\tilde{x}.(A' \parallel B' \parallel [\![T]\!]) \\ & = & \nu\tilde{x}.([\![H']\!]_t \parallel [\![T]\!]) \\ & = & [\![(H', T)]\!]_t \end{array}$$

Case 8. Rule (INP-S). $(H, L :: T) \to_t (H\{x_j \mapsto Q_j \backslash O\}, P_j(\sigma \cup \{^0/_{y_j}\}) :: T)$ where $L = \sum_{i\in I} x_i?F_i@y_i \to P_i$, and there is an $j \in I$ for which $H(x_j) = Q_j$ and $(O, \sigma) \in outms(F_j, Q_j) \neq \emptyset$.
Let $H' = H\{x_j \mapsto Q_j \backslash O\}$. Since $(O, \sigma) \in outms(F_j, Q_j) \neq \emptyset$, O must be of the form $!v$ and $\sigma = match(F_j, v, \emptyset) \neq \emptyset$. Hence, there is a trigger on x_j, $x_j!v$. By (CHOICE) we have that $L \xrightarrow{x_j?v} P_j(\sigma \cup \{^0/_{y_j}\})$. By (COMM) we have that $x_j!v \parallel$

$L \xrightarrow{\tau} \surd \parallel P_j(\sigma \cup \{{}^0/_{y_j}\}) \equiv P_j(\sigma \cup \{{}^0/_{y_j}\})$. Since H' is the same as H with an output observer removed, its listeners are the same, so $A' = \Pi alllisteners(H', t) = A$, and its triggers are those of H minus $x_j!v$: $outputs(Q_i \backslash O) = outputs(Q_j) \backslash O$ so $triggers(Q_j \backslash O, x_j) = triggers(Q_j, x_j) \backslash \{x_j!v\}$ and so $B' = \Pi alltriggers(H') = \Pi alltriggers(H) \backslash \{x_j!v\}$, this is, $B = B' \parallel x_j!v$. Hence,

$$\begin{array}{rcl}
[\![(H, L :: T)]\!]_t & = & \nu\tilde{x}.([\![H]\!]_t \parallel [\![L :: T]\!]) \\
 & = & \nu\tilde{x}.([\![H]\!]_t \parallel L \parallel [\![T]\!]) \\
 & = & \nu\tilde{x}.(A \parallel B \parallel L \parallel [\![T]\!]) \\
 & \xrightarrow{\tau} & \nu\tilde{x}.(A' \parallel B' \parallel P(\sigma \cup \{{}^0/_{y_j}\}) \parallel [\![T]\!]) \\
 & \equiv & \nu\tilde{x}.([\![H']\!]_t \parallel [\![P(\sigma \cup \{{}^0/_{y_j}\}) :: T]\!]) \\
 & = & [\![(H', P(\sigma \cup \{{}^0/_{y_j}\}) :: T)]\!]_t
\end{array}$$

Case 9. Rule (INST). $(H, A(\vec{v}) :: T) \rightarrow_t (H, T :: P\{\vec{v}/\vec{x}\})$ where $A(\vec{x}) \stackrel{def}{=} P$. By definition of congruence, $A(\vec{v}) \equiv P\{\vec{v}/\vec{x}\}$, so

$$\begin{array}{rcl}
[\![(H, A(\vec{v}) :: T)]\!]_t & = & \nu\tilde{x}.([\![H]\!]_t \parallel [\![A(\vec{v}) :: T]\!]) \\
 & = & \nu\tilde{x}.([\![H]\!]_t \parallel A(\vec{v}) \parallel [\![T]\!]) \\
 & \equiv & \nu\tilde{x}.([\![H]\!]_t \parallel P\{\vec{v}/\vec{x}\} \parallel [\![T]\!]) \\
 & = & \nu\tilde{x}.([\![H]\!]_t \parallel [\![T :: P\{\vec{v}/\vec{x}\}]\!])
\end{array}$$

$\square$

Theorem 4. ***(Abstract Machine Validity)***

1. *If* $(H, R) \hookrightarrow_0 (H', R')$ *then* $[\![(H, R)]\!] \Longrightarrow [\![(H', R')]\!]$
2. *If* $(H, R) \hookrightarrow_1 (H', R')$ *then* $[\![(H, R)]\!] \stackrel{d}{\leadsto\Longrightarrow} [\![(H', R')]\!]$ with $d = t_2 - t_1$ where $R = (t_1, T_1) \cdot (t_2, T_2) \cdot ...$
3. *If* $(H, R) \hookrightarrow_2 (H', R')$ *then* $[\![(H, R)]\!] \equiv [\![(H', R')]\!]$

Proof. To prove the result, let us rewrite the encoding of the queue $R = (t_1, T_1) \cdot (t_2, T_2) \cdot ... \cdot (t_n, T_n)$ as follows: Let $R = (t_1, T_1) \cdot R_1$ where $R_1 = (t_2, T_2) \cdot ... \cdot (t_n, T_n)$. Define $[\![R]\!] \stackrel{def}{=} [\![T_1]\!] \parallel [\![R_1]\!]_{t_1}$ where the auxiliary function $[\![-]\!]_t$ over queues is defined as $[\![\langle\rangle]\!]_t \stackrel{def}{=} \surd$ and $[\![(t, T) \cdot R_1]\!]_{t_1} \stackrel{def}{=} (\Delta(t - t_1) \rightarrow [\![T]\!]) \parallel [\![R_1]\!]_{t_1}$. It is straightforward to see that the previous definition is just the unfolding of this definition. Intuitively, $[\![R]\!]_t$ encodes the queue R advancing each timer by t (i.e., the delay for each time-slot is substracted t).

Now we prove each of the three items.

1. Assume that $(H,R) \hookrightarrow_0 (H',R')$. Hence it must have been derived using (TIMESLOT) where R is of the form $(t_1,T_1)\cdot R_1$ and R' is of the form $(t_1,T_1')\cdot R_1$ with $T_1 \neq \epsilon$ and $(H,T_1) \rightarrow_t (H',T_1')$. So by Lemma 5 we have that $[\![(H,T_1)]\!]_{t_1} \Longrightarrow [\![(H',T_1')]\!]_{t_1}$. This is, $\nu\vec{x}.([\![H]\!]_{t_1} \parallel [\![T_1]\!]) \Longrightarrow \nu\vec{x}.([\![H']\!]_{t_1} \parallel [\![T_1']\!])$, which, by using the properties of congruence implies that $\nu\vec{x}.([\![H]\!]_{t_1} \parallel [\![T_1]\!] \parallel [\![R_1]\!]_{t_1}) \Longrightarrow \nu\vec{x}.([\![H']\!]_{t_1} \parallel [\![T_1']\!] \parallel [\![R_1]\!]_{t_1})$, so,

$$\begin{array}{rcl} [\![(H,R)]\!] & = & \nu\vec{x}.([\![H]\!]_{t_1} \parallel [\![R]\!]) \\ & = & \nu\vec{x}.([\![H]\!]_{t_1} \parallel [\![T_1]\!] \parallel [\![R_1]\!]_{t_1}) \\ & \Longrightarrow & \nu\vec{x}.([\![H']\!]_{t_1} \parallel [\![T_1']\!] \parallel [\![R_1]\!]_{t_1}) \\ & = & \nu\vec{x}.([\![H']\!]_{t_1} \parallel [\![R']\!]) \\ & = & [\![(H',R')]\!] \end{array}$$

where $[\![R']\!] = [\![(t_1,T_1')\cdot R_1]\!] = [\![T_1']\!] \parallel [\![R_1]\!]_{t_1}$

2. Assume that $(H,R) \hookrightarrow_1 (H',R')$. In this case, it must be that $R = (t_1,\epsilon)\cdot R_1$, $R' = R_1$ and $H' = H$. Let us assume that $R_1 = (t_2,T_2)\cdot R_2$ for some R_2, so that $[\![R_1]\!]_{t_1} = \Delta(t_2-t_1) \rightarrow [\![T_2]\!] \parallel [\![R_2]\!]_{t_1}$. Furthermore, let $d = t_2 - t_1$. First we will show that $[\![R_2]\!]_{t_1} \stackrel{d}{\rightsquigarrow} [\![R_2]\!]_{t_2}$. The remainder of the queue, R_2 can be empty or have more timeslots. If it is empty, then $[\![R_2]\!]_t = \surd$ for any t. Suppose that it is not empty, i.e., $R_2 = (t_3,T_3)\cdot R_3$. Hence $[\![R_2]\!]_{t_1} = \Delta(t_3-t_1) \rightarrow [\![T_3]\!] \parallel [\![R_3]\!]_{t_1}$. Hence, by (TDELAY) we have can show that

$$\begin{array}{rcl} [\![R_2]\!]_{t_1} & \stackrel{d}{\rightsquigarrow} & \Delta(t_3-t_1-d) \rightarrow [\![T_3]\!] \parallel [\![R_3]\!]_{t_1+d} \\ & = & \Delta(t_3-t_2) \rightarrow [\![T_3]\!] \parallel [\![R_3]\!]_{t_2} \\ & = & [\![R_2]\!]_{t_2} \end{array}$$

Now we will show that $[\![H]\!]_{t_1} \stackrel{d}{\rightsquigarrow} [\![H]\!]_{t_2}$. To do this, recall that $[\![H]\!]_{t_1} = \Pi alllisteners(H,t_1) \parallel \Pi alltriggers(H)$. First note that under evolution, the set of triggers in the heap does not change at all, only listeners change, by updating the elapsed-time variables of each branch in each listener, incrementing them with d. Each listener in $[\![H]\!]_{t_1}$ has the form $L = \sum_{i\in I} x_i?F_i@y_i \rightarrow P_i\{^{y_i+(t_1-u)}/_{y_i}\}$ where u is the time when the listener was registered. By using (TCHOICE), each such listener evolves as follows: $L \stackrel{d}{\rightsquigarrow} L' = \sum_{i\in I} x_i?F_i@y_i \rightarrow P_i\{^{y_i+(t_1-u)}/_{y_i}\}\{^{y_i+d}/_{y_i}\}$ but the composition of substitutions $\{^{y_i+(t_1-u)}/_{y_i}\}\{^{y_i+d}/_{y_i}\}$ is the same as the substitution $\{^{y_i+d+(t_1-u)}/_{y_i}\}$ which is the same as $\{^{y_i+(t_2-u)}/_{y_i}\}$ since $d = t_2 - t_1$. Hence each resulting listener is of the form $L' = \sum_{i\in I} x_i?F_i@y_i \rightarrow P_i\{^{y_i+(t_2-u)}/_{y_i}\}$ which when we put all together in parallel gives us $\Pi alllisteners(H,t_2)$ and since the triggers remained the same, by using the (TPAR) rule on all listeners and triggers, we conclude that $[\![H]\!]_{t_1} \stackrel{d}{\rightsquigarrow} \Pi alllisteners(H,t_2) \parallel \Pi alltriggers(H) = [\![H]\!]_{t_2}$.
Having established that $[\![R_2]\!]_{t_1} \stackrel{d}{\rightsquigarrow} [\![R_2]\!]_{t_2}$ and $[\![H]\!]_{t_1} \stackrel{d}{\rightsquigarrow} [\![H]\!]_{t_2}$ we obtain

the result as follows:

$$\begin{array}{rcl}
[\![(H,R)]\!] & = & \nu\vec{x}.([\![H]\!]_{t_1} \parallel [\![R]\!]) \\
& = & \nu\vec{x}.([\![H]\!]_{t_1} \parallel [\![\epsilon]\!] \parallel [\![R_1]\!]_{t_1}) \\
& = & \nu\vec{x}.([\![H]\!]_{t_1} \parallel \surd \parallel \Delta(t_2 - t_1) \to [\![T_2]\!] \parallel [\![R_2]\!]_{t_1}) \\
& \overset{d}{\rightsquigarrow} & \nu\vec{x}.([\![H]\!]_{t_2} \parallel \Delta 0 \to [\![T_2]\!] \parallel [\![R_2]\!]_{t_2}) \\
& \xrightarrow{\tau} & \nu\vec{x}.([\![H]\!]_{t_2} \parallel [\![T_2]\!] \parallel [\![R_2]\!]_{t_2}) \\
& = & \nu\vec{x}.([\![H]\!]_{t_2} \parallel [\![R_1]\!]) \\
& = & [\![(H',R')]\!]
\end{array}$$

3. Assume that $(H,R) \hookrightarrow_2 (H',R')$. Hence, R is of the form $(t_1, \Delta E \to P :: T_1) \cdot R_1$, $H' = H$ and $R' = insort(t_1 + d, P, (t_1, T_1) \cdot R_1)$ where $d = eval(E)$. Suppose that the queue $(t_1, T_1) \cdot R_1$ has the form $(t_1, T_1) \cdot (t_2, T_2) \cdot ... \cdot (t_1 + d, T') \cdot ... \cdot (t_n, T_n)$, where there is a non-empty time-slot $(t_1 + d, T')$ with $T' = P_1 :: P_2 :: \cdots :: P_m$. Hence, we have that

$$\begin{array}{rcl}
[\![R_1]\!]_{t_1} & = & \Delta(t_2 - t_1) \to [\![T_2]\!] \parallel \cdots \parallel \Delta(t_1 + d - t_1) \to [\![T']\!] \parallel \cdots \\
& = & \Delta(t_2 - t_1) \to [\![T_2]\!] \parallel \cdots \parallel \Delta d \to [\![T']\!] \parallel \cdots
\end{array}$$

Let $T'' = T' :: P$, the result of inserting P in the time-slot[21] $t_1 + d$. Then, we have that

$$\begin{array}{rcl}
[\![R']\!] & = & [\![(t_1, T_1) \cdot (t_2, T_2) \cdot ... \cdot (t_1 + d, T'') \cdot ...]\!] \\
& = & [\![T_1]\!] \parallel \Delta(t_2 - t_1) \to [\![T_2]\!] \parallel \cdots \parallel \Delta(t_1 + d - t_1) \to [\![T'']\!] \parallel \cdots \\
& = & [\![T_1]\!] \parallel \Delta(t_2 - t_1) \to [\![T_2]\!] \parallel \cdots \parallel \Delta d \to [\![T'']\!] \parallel \cdots \\
& \equiv & [\![T_1]\!] \parallel \Delta(t_2 - t_1) \to [\![T_2]\!] \parallel \cdots \parallel \Delta d \to [\![T']\!] \parallel \Delta d \to P \parallel \cdots \\
& \equiv & \Delta d \to P \parallel [\![T_1]\!] \parallel \Delta(t_2 - t_1) \to [\![T_2]\!] \parallel \cdots \parallel \Delta d \to [\![T']\!] \parallel \cdots \\
& = & \Delta d \to P \parallel [\![T_1]\!] \parallel [\![R_1]\!]_{t_1}
\end{array}$$

Hence we obtain the following:

$$\begin{array}{rcl}
[\![(H,R)]\!] & = & \nu\vec{x}.([\![H]\!]_{t_1} \parallel [\![R]\!]) \\
& = & \nu\vec{x}.([\![H]\!]_{t_1} \parallel [\![\Delta E \to P :: T_1]\!] \parallel [\![R_1]\!]_{t_1}) \\
& = & \nu\vec{x}.([\![H]\!]_{t_1} \parallel \Delta E \to P \parallel [\![T_1]\!] \parallel [\![R_1]\!]_{t_1}) \\
& \equiv & \nu\vec{x}.([\![H]\!]_{t_1} \parallel [\![R']\!]) \\
& = & [\![(H',R')]\!]
\end{array}$$

□

[21] If we assume that there was no such time-slot, the result is the same.