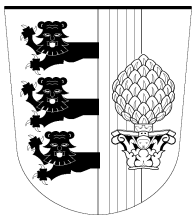
Comparing the worst-Case efficiency of asynchronous systems with PAFAS

Flavio Corradini, Walter Vogler, Lars Jenner

Angaben zur Veröffentlichung / Publication details:

Corradini, Flavio, Walter Vogler, and Lars Jenner. 2000. "Comparing the worst-Case efficiency of asynchronous systems with PAFAS." Augsburg: Universität Augsburg.

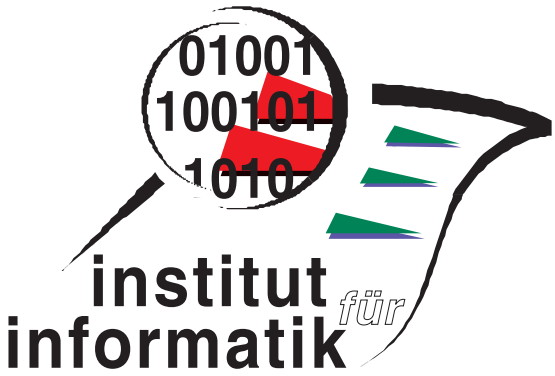
Universität Augsburg



Comparing the Worst-Case Efficiency of Asynchronous Systems with PAFAS

Flavio Corradini, Walter Vogler, Lars Jenner

Report 2000-6 Dezember 2000



Institut für Informatik
D-86135 Augsburg

Comparing the Worst-Case Efficiency of Asynchronous Systems with PAFAS

Flavio Corradini*
Dipartimento di Matematica Pura ed Applicata, Università di L'Aquila, L'Aquila, Italy
flavio@univaq.it
Walter Vogler, Lars Jenner†
Institut für Informatik, Universität Augsburg, D-86135 Augsburg, Germany
Walter.Vogler@informatik.uni-augsburg.de

Abstract

A timed CCS-like process algebra PAFAS and a testing scenario are developed for evaluating the temporal worst-case efficiency of asynchronous concurrent systems. Each action is associated with a maximal time delay, which allows components to work with arbitrary relative speeds; for simplicity, the maximal delay is 1 or 0, but time is continuous. The canonical testing preorder associated to our timed testing compares worst-case efficiency; we show that this efficiency preorder can equivalently be defined considering only discrete time, which is of course much simpler. Then we characterize the efficiency preorder with some kind of refusal traces; despite the rather weak control an asynchronous test environment has, this gives quite detailed insight into the temporal system behaviour. Since the preorder is not a precongruence for choice, we refine it to the efficiency precongruence, which is a precongruence for all operators of the algebra including recursion.

1 Motivation and Introduction

Classical process algebras like CCS model asynchronous systems, where the components have arbitrary relative speeds. To consider the temporal behaviour, several timed process algebras have been proposed, where usually systems are regarded as synchronous, i.e. have components with fixed speeds. The easiest of these is SCCS [20], since terms are essentially the same as for CCS; the natural choice to fix the speeds of components is to assume that each action takes one unit of time; so SCCS-semantics differs from CCS-semantics essentially by excluding runs where one component performs many actions while another performs just one.

Our aim is to compare the temporal worst-case efficiency of asynchronous concurrent systems modelled with a process algebra, and – as in the case of SCCS – we want to keep things simple by using (almost) just classical CCS-like process terms. Furthermore, we

*This work was supported by MURST progetto 'Saladin: Software Architectures and Languages to Coordinate Distributed Mobile Components'.
†This work was supported by the DFG-project 'Halbordnungstesten'.

will use a variant of (must-)testing [12], where the testing preorder can be interpreted as comparing efficiency.

A usual treatment of asynchronous systems with a timed process algebra is to allow arbitrary idling before each action [20, 23]; this achieves arbitrary relative speeds, but is not suitable for defining worst-case runs since each action already can take arbitrarily long. Here, we assume that each action is performed within a given time bound. Since an action can also be performed earlier, it is still possible that one component performs many actions while another performs just one; thus, despite assuming an upper time bound, we deal with arbitrary relative speeds, i.e. with truly asynchronous systems. Also e.g. [18] uses upper time bounds for the performance evaluation of asynchronous systems in the area of distributed algorithms.

To keep things simple as in SCCS, we could take 1 as a common upper time bound for all actions, and we have done so in a preliminary version of this paper [16]. To demonstrate the generality of our results, our process algebra PAFAS (process algebra for asynchronous systems[1] also has urgent actions with upper time bound 0. Even when studying processes where all bounds are 1, restricted use of such actions is necessary in order to describe states reached after some time has passed. To have urgent actions as first class citizens enhances the modelling power, since it is often very natural to consider the time taken by some actions as negligible compared to others. It might also help to find an axiomatization; compare [34], where a fragment of the algebra in [16] is axiomatized. Due to our notation, we interpret ordinary CCS-like processes automatically as processes where all actions have bound 1.

We compare processes via the testing approach developed by [12] and extended to timed testing in a Petri net framework in [31, 17], where a timed test is a test environment together with a time bound. A process is embedded into the environment essentially via parallel composition and satisfies a timed test, if success is reached before the time bound in *every* run of the composed system, i.e. even in the worst case. If some process P satisfies each timed test satisfied by a process Q, then P may be successful in more environments than specified by Q, but it may also be successful in the same environments within a shorter time; therefore, we call it a *faster implementation* of Q, and the testing preorder is naturally an *efficiency preorder*.

To define timed testing formally, we have to define runs of asynchronous systems. In Section 2, we develop a suitable semantics where time is continuous; we try to formalize our intuitive ideas as directly as possible without anticipating any specific treatment that might be necessary to obtain a precongruence in the end. As a sort of sanity check, we show in Section 3 that (most of) our processes do not have time-stops, i.e. every finite run can be extended such that time grows arbitrarily; this result depends on the restriction to time-guarded recursion.

It has turned out that the classical embedding in the test environment with just parallel composition leads to a testing preorder which – surprisingly – is not a precongruence for prefixing; instead of refining the preorder to the coarsest such precongruence (cf. [15]), we get this precongruence directly by using a slightly different, but also intuitive embedding.

Using continuous time is difficult to handle; e.g. initially each process can make uncountably many different time steps. Our first main result in Section 4 shows that realism and simplicity can be reconciled: we define an analogous efficiency preorder based on discrete time behaviour and show its coincidence with the first one. In Section 5, as usual in a

[1] Or process algebra for faster asynchronous systems, but the comparison of speed of course relies on testing, and not on the algebra alone.

testing approach, we characterize the efficiency preorder – here with some kind of refusal traces. The important point with this second main result is that test environments are asynchronous systems, hence 'temporally weak', but nevertheless reveal the temporal behaviour of tested processes quite in detail; correspondingly, the construction of revealing tests is a little involved.

These two main results are translations (and extensions to urgent actions) from corresponding results for Petri nets in [17] to a process algebra setting. On the one hand, this demonstrates that the underlying ideas are not model-dependent; on the other hand, the developments here are quite different, in particular since process algebras are much more powerful than finite safe Petri nets; see e.g. the progress preorder in Section 4.

We also provide precongruence results for parallel composition, hiding, relabelling and prefixing. In Section 6 we refine the efficiency preorder to a precongruence also for choice: as usual, we additionally have to take into account the (initial) stability of processes; furthermore, we have to know to some degree which behaviour starts with an internal action. Quite surprisingly, although we consider a preorder, the condition on stability is not only an implication but an equivalence. The resulting efficiency precongruence is then shown to be the coarsest precongruence for all operators of our process algebra that respects inclusion of discrete behaviour. We also provide a precongruence result for recursion; for this, we added to the algebra of [16] the complete time-stop Ω.

We close with a small example in Section7; work on more elaborate examples is in progress. In the Petri net setting, different implementations of a bounded buffer are compared in [32, 17], while [33, 7] study token-ring solutions to the problem of mutual exclusion.

2 Continuously Timed Processes and Tests

We will study a *CCS*-like process algebra with *TCSP*-like parallel composition $\|_A$, where A is the set of actions components have to synchronize on. Processes will perform (atomic) actions instantaneously within some time bound; we interpret ordinary *CCS*-like processes such that this time bound is 1, but also consider urgent actions with time bound 0. Time passes (in this section) continuously between action occurrences. For example, process $a.P$ will idle and then perform action a at some time point in the real interval $[0;1]$, evolving to P. To model this, we additionally have to introduce continuously timed actions $\langle a, r\rangle$, which carry a 'timer' r whose initial value can be chosen from the interval $[0;1]$ of real numbers. Whenever time passes globally by a certain amount, the timer of a locally activated action will be decreased accordingly. Thus, we interpret *CCS*-like processes as so-called *initial* processes where all timer values are 1; e.g. $a.b + c$ is $\langle a,1\rangle.\langle b,1\rangle + \langle c,1\rangle$, which becomes $\langle a,.7\rangle.\langle b,1\rangle + \langle c,.7\rangle$ after time .3 and $\langle a,0\rangle.\langle b,1\rangle + \langle c,0\rangle$ (which we also write as $\underline{a}.b + \underline{c}$) after additional time .7. Processes with timer-values $\notin \{0,1\}$ only interest us if they occur in runs from processes with discrete timer-values.

Timer value 0 denotes that the idle-time of the respective action has elapsed, hence it must either occur or be deactivated (in the example: a by c or vice versa) before time may pass further – unless it has to wait for synchronization with another component (i.e. our processes are *patient*). E.g. process $\langle a,1\rangle.P$ can idle, process $\langle a,0\rangle.Q$ cannot, and as a stand-alone process it has no reason to wait; but as a component in $(\langle a,0\rangle.Q)\|_{\{a\}}(\langle a,1\rangle.P)$, it has to wait for synchronization on a, and this can take up to time 1, since component $\langle a,1\rangle.P$ may idle this long.

We also use two distinguished actions: τ represents internal activity that is unobservable for other components; in the testing scenario, ω is reserved for observers (test processes), which use this action in order to signal success of a test.

Definition 2.1 $\mathbb{A}$ is an infinite set of actions with a special action ω – the *success action*; additionally, τ is the *internal action*. We define $\mathbb{A}_\tau = \mathbb{A} \cup \{\tau\}$. Elements of $\mathbb{A}$ are denoted by $a, b, c, \ldots$ and those of $\mathbb{A}_\tau$ are denoted by $\alpha, \beta, \ldots$.

Let $\mathbb{T} = [0; 1] \subseteq \mathbb{R}_0^+$ be the set of real numbers in the interval $[0; 1]$. Elements from $\mathbb{T}$ are denoted e.g. by ρ or r.

For $\alpha \in \mathbb{A}_\tau$ and $r \in \mathbb{T}$, $\langle \alpha, r \rangle$ is a *continuously timed action* with timer r. We use α as a shorthand for $\langle \alpha, 1 \rangle$ and $\underline{\alpha}$ as a shorthand for $\langle \alpha, 0 \rangle$, which we call an *urgent* action.

Definition 2.2 (*continuously timed and initial process terms and processes*) Let $\Phi : \mathbb{A}_\tau \to \mathbb{A}_\tau$ be a function such that the set $\{\alpha \in \mathbb{A}_\tau \mid \emptyset \neq \Phi^{-1}(\alpha) \neq \{\alpha\}\}$ is finite, $\Phi^{-1}(\omega) \subseteq \{\omega\}$ and $\Phi(\tau) = \tau$; then Φ is a *general relabelling function*.

A *general process term* P is generated by the following grammar:

$$P \quad ::= \quad \mathbf{0} \;\mid\; \Omega \;\mid\; x \;\mid\; \langle \alpha, r \rangle.P \;\mid\; P + P \;\mid\; P\|_A P \;\mid\; P[\Phi] \;\mid\; \mu x.P$$

where $\mathbf{0}$ *(Nil)* and Ω *(Timestop)* are constants, $x \in \mathcal{X} = \{x, y, z, \ldots\}$ is a *(process) variable*, $\langle \alpha, r \rangle$ a continuously timed action, Φ a general relabelling function and $A \subseteq \mathbb{A}$ possibly infinite. Additionally, we only allow *guarded* recursion, where *also internal* timed actions $\langle \tau, r \rangle$ may serve as a guard; see below. The set of general process terms is denoted by $\tilde{\mathbb{L}}$. Significant subsets of $\tilde{\mathbb{L}}$ are:

- $\tilde{\mathbb{P}}$, the set of *discrete process terms*. These are $\tilde{\mathbb{L}}$-terms where all r are from $\{0, 1\}$ and recursion is *time-guarded* (see below).
- $\tilde{\mathbb{P}}_c$, the set of *continuous process terms*. These are $\tilde{\mathbb{L}}$-terms generated by the following grammar (continuous timers, $r \in (0, 1)$, can appear only at the top level):

  $$P \quad ::= \quad \mathbf{0} \;\mid\; x \;\mid\; \langle \alpha, r \rangle.Q \;\mid\; P + P \;\mid\; P\|_A P \;\mid\; P[\Phi] \;\mid\; \mu x.Q$$

  where Q and $\mu x.Q$ are $\tilde{\mathbb{P}}$-terms – hence, recursion is time-guarded.
- $\tilde{\mathbb{P}}_1$, the set of *initial process terms*. These are Ω-free $\tilde{\mathbb{L}}$-terms and the choice of r is restricted to $r = 1$.

A term P is closed and called a *process* if all variables x in P are bound by the corresponding μx-operator. The set of closed terms of $\tilde{\mathbb{L}}$, $\tilde{\mathbb{P}}$, $\tilde{\mathbb{P}}_c$ and $\tilde{\mathbb{P}}_1$ is obtained by removing $\tilde{}$ from the corresponding set. Thus, $\mathbb{L}$ denotes the set of general processes, $\mathbb{P}$ the set of discrete processes, $\mathbb{P}_c$ the set of continuous processes and $\mathbb{P}_1$ the set of initial processes.

As explained above, $\mathbb{P}$ is the set of processes we are mainly interested in; $\mathbb{P}_1$ is an important subset, since it corresponds to ordinary CCS-like processes. The set $\mathbb{P}_c$ is of interest, since the processes in $\mathbb{P}$ can reach processes in $\mathbb{P}_c$ when they let pass a non-discrete amount of time; due to our treatment of recursion, we will actually need terms from $\tilde{\mathbb{P}}_c$ in the definition of our operational semantics of $\mathbb{P}$-processes.

With $\equiv$ we denote syntactical equality of process terms.

$\mathbf{0}$ is the Nil-process, which cannot perform any action, but may let time pass without limit; a trailing $\mathbf{0}$ will often be omitted, so e.g. $a.b + c$ abbreviates $a.b.\mathbf{0} + c.\mathbf{0}$. Ω cannot perform any action and does not allow time to pass any further; such a time-stop process is not realistic and only introduced to show a precongruence result for recursion. $x \in \mathcal{X}$ is a process variable used for recursion. $\langle \alpha, r \rangle.P$ is (action-) prefixing, known from *CCS*, where $\langle \alpha, r \rangle.P$ is ready to perform action α at some time in $[0, r]$. $P_1 + P_2$ models the choice (sum) of two conflicting processes P_1 and P_2. $P_1 \|_A P_2$ is the parallel composition of two processes P_1 and P_2 that run in parallel and have to synchronize on all actions from A; this synchronization discipline is inspired from *TCSP*.

The general relabelling operation $P[\Phi]$ subsumes the classically distinguished operations relabelling and hiding. These can be understood as special cases of a general relabelling in the following way: if Φ satisfies the condition $\Phi^{-1}(\tau) = \{\tau\}$, then Φ is a *(classical) relabelling function*; if for a set $A \subseteq \mathbb{A}$ Φ satisfies the conditions $\Phi|_A = \tau$ and $\Phi|_{\mathbb{A}_\tau \setminus A} = id_{\mathbb{A}_\tau}$, then we consider P/A to be a notation equivalent to $P[\Phi]$, where A is called a *hiding set*. The restrictions on general relabelling functions serve several purposes: $\Phi(\tau) = \tau$ ensures that τ cannot be made visible by relabelling, and $\Phi^{-1}(\omega) \subseteq \{\omega\}$ ensures that testable processes will be closed under general relabelling. The finiteness of the set $\{\alpha \in \mathbb{A}_\tau \mid \emptyset \neq \Phi^{-1}(\alpha) \neq \{\alpha\}\}$ will ensure later on that the number of different actions ever performable by a given c-process is finite; note that we allow infinite hiding sets, however.

$\mu x.P$ models recursion. Ω or some $x \in \mathcal{X}$ is *guarded* in a general process term $P \in \tilde{\mathbb{L}}$, if each occurrence of Ω or x is in a subterm $\langle \alpha, r \rangle.Q$ of P where $\alpha \in \mathbb{A}_\tau$; note that *also internal* timed actions $\langle \tau, r \rangle$ may serve as a *guard*. We speak of *time-guarded* if $r = 1$. In this paper, we only consider general process terms, discrete process terms resp., $\mu x.P$ where x is guarded, time-guarded resp., in P. We say that $P \in \tilde{\mathbb{L}}$ is *(time-)guarded* if Ω and all $x \in \mathcal{X}$ are (time-)guarded in P. Note: general processes are guarded and continuous processes are time-guarded.

In order to economize on parentheses, precedence of the operators in decreasing order is as follows: relabelling, prefix, recursion, parallel composition, choice.

Whenever we perform syntactical substitution $P\{Q/x\}$, we assume $\mathit{free}(Q) \cap \mathit{bound}(P) = \emptyset$ (Barendregt convention), where $\mathit{free}(P)$ and $\mathit{bound}(P)$ denote the sets of free resp. bound variables in P. If $\mathcal{S}$ is a function $\mathcal{S} : \mathcal{X} \mapsto \tilde{\mathbb{L}}$, then $\mathcal{S}$ denotes a *simultaneous substitution* of all variables, and we write $[P]_{\mathcal{S}}$ for $P\{\mathcal{S}(x)/x, \mathcal{S}(y)/y, \ldots\}$.

We intend choice to be commutative and associative; therefore, $\sum_{i \in I} P_i$ is used as a shorthand for the sum of all $P_i \in \tilde{\mathbb{L}}$, where i is in a finite indexing set I. We define $\sum_{i \in \emptyset} P_i \equiv \mathbf{0}$, and if $|I| = 1$, then $\sum_{j \in \{i\}} P_j \equiv P_i$.

Now the purely functional behaviour of process terms (i.e. which actions they can perform) is given by the following operational semantics.

Definition 2.3 (*Operational semantics of functional behaviour*) The following SOS-rules define the transition relation $\xrightarrow{\alpha} \subseteq (\tilde{\mathbb{L}} \times \tilde{\mathbb{L}})$ for each $\alpha \in \mathbb{A}_\tau$. We always require the *side-condition* that $(P, P') \in \xrightarrow{\alpha}$ implies that P is guarded. As usual, we write $P \xrightarrow{\alpha} P'$ if $(P, P') \in \xrightarrow{\alpha}$ and $P \xrightarrow{\alpha}$ if there exists a $P' \in \tilde{\mathbb{L}}$ such that $(P, P') \in \xrightarrow{\alpha}$, and similar conventions will apply later on.

$$\mathrm{Pref}_a \ \frac{}{\langle \alpha, r \rangle.P \xrightarrow{\alpha} P} \qquad \mathrm{Par}_{a1} \ \frac{\alpha \notin A,\ P_1 \xrightarrow{\alpha} P_1'}{P_1 \|_A P_2 \xrightarrow{\alpha} P_1' \|_A P_2} \qquad \mathrm{Par}_{a2} \ \frac{\alpha \in A,\ P_1 \xrightarrow{\alpha} P_1',\ P_2 \xrightarrow{\alpha} P_2'}{P_1 \|_A P_2 \xrightarrow{\alpha} P_1' \|_A P_2'}$$

$$\text{Sum}_a \ \frac{P_1 \xrightarrow{\alpha} P_1'}{P_1 + P_2 \xrightarrow{\alpha} P_1'} \qquad \text{Rel}_a \ \frac{P \xrightarrow{\alpha} P'}{P[\Phi] \xrightarrow{\Phi(\alpha)} P'[\Phi]} \qquad \text{Rec}_a \ \frac{P \xrightarrow{\alpha} P'}{\mu x.P \xrightarrow{\alpha} P'\{\mu x.P/x\}}$$

Additionally, there are symmetric rules for Par_{a1} and Sum_a for actions of P_2. Finally, $\mathcal{A}(P) = \{\alpha \in \mathbb{A}_\tau \mid P \xrightarrow{\alpha}\}$ is the set of *activated actions* of P.

Except for Pref_a and Rec_a, these rules are standard. Pref_a allows an activated action to occur disregarding the value of its timer; in particular, $\alpha.P$ can perform α just as e.g. in CCS. Additionally, passage of time will never deactivate actions or activate new ones, and we capture all behaviour that is possible in the standard *CCS*-like setting without time. Note that rule Rec_a implicitly makes use of guarded recursion. It forces us to define an operational semantics not only for general processes but also for general process terms (in the premise of Rec_a). On the other hand, it will simplify proofs of operational properties, since it connects induction on inferences of a transition with induction on the structure of a general process.

Due to the rules for choice, it should be clear that any sensible semantics will make choice indeed commutative and associative; in particular, this will be the case in this paper; we will not mention this any further.

The set of activated actions of a general process term P describes its immediate functional behaviour. Just as $\mathcal{A}(\Omega)$, $\mathcal{A}(x)$ is empty for process variables $x \in \mathcal{X}$, reflecting that unbound occurrence of a variable means incomplete specification, which Ω stands for. Note that $\mathcal{A}(P)$ records only actions, not the possibly various timer values associated with the same action in a process. Instead via the operational semantics, $\mathcal{A}(P)$ can equivalently be determined inductively from the syntactical structure of P alone; see the proposition below.

The set of activated actions will be preserved both along passage of time and under substitution of guarded variables; furthermore, due to the definition of general relabelling functions, $\mathcal{A}(P)$ is always finite, and this will be used for the characterization of our testing preorder.

Proposition 2.4 Let $P, Q, R \in \tilde{\mathbb{L}}$ be general process terms.

1. $\mathcal{A}(P)$ is finite.
2. Let $\alpha \in \mathbb{A}_\tau$ and $x \in \mathcal{X}$ be guarded in P. Then, $P\{Q/x\} \xrightarrow{\alpha} R$ if and only if there exists $P' \in \tilde{\mathbb{L}}$ with $P \xrightarrow{\alpha} P'$ and $R \equiv P'\{Q/x\}$; in particular $\mathcal{A}(P) = \mathcal{A}(P\{Q/x\})$.
3. For P not guarded, $\mathcal{A}(P) = \emptyset$; for guarded P, $\mathcal{A}(P)$ can be calculated by structural induction:

   Nil: $\mathcal{A}(\mathbf{0}) = \emptyset$
   Pref: $\mathcal{A}(\langle \alpha, r \rangle.P) = \{\alpha\}$ for all $\alpha \in \mathbb{A}_\tau$
   Sum: $\mathcal{A}(P_1 + P_2) = \mathcal{A}(P_1) \cup \mathcal{A}(P_2)$
   Par: $\mathcal{A}(P_1 \|_A P_2) = (\mathcal{A}(P_1) \cap \mathcal{A}(P_2) \cap A) \cup ((\mathcal{A}(P_1) \cup \mathcal{A}(P_2)) \setminus A)$
   Rel: $\mathcal{A}(P[\Phi]) = \Phi(\mathcal{A}(P))$
   Rec: $\mathcal{A}(\mu x.P) = \mathcal{A}(P)$

Proof:

1. Induction on the structure of P.

2. Induction on the structure of P:

Nil, Stop: $\mathbf{0} \equiv \mathbf{0}\{Q/x\} \xrightarrow{\alpha}$ for no $\alpha \in \mathbb{A}_\tau$; Ω is analogous.

Var: x guarded in $P \equiv y$ implies $x \not\equiv y$; this case is analogous to Nil.

Pref: x is guarded in $\langle \alpha, r \rangle.P$ and $(\langle \alpha, r \rangle.P)\{Q/x\} \equiv \langle \alpha, r \rangle.(P\{Q/x\}) \xrightarrow{\alpha} P\{Q/x\}$, which is unique, and $\langle \alpha, r \rangle.P \xrightarrow{\alpha} P$, which is unique.

Sum, Par, Rel: straightforward induction.

Rec: $\mu x.P \xrightarrow{\alpha} P'\{\mu x.P/x\}$ due to $P \xrightarrow{\alpha} P'$ by rule Rec_a; then x is bound in $\mu x.P$ and $P'\{\mu x.P/x\}$, hence $(\mu x.P)\{Q/x\} \equiv \mu x.P \xrightarrow{\alpha} P'\{\mu x.P/x\} \equiv (P'\{\mu x.P/x\})\{Q/x\}$. Now let $x \not\equiv y$; then $(\mu y.P)\{Q/x\} \equiv \mu y.(P\{Q/x\})$ since by Barendregt convention y is not free in Q.

By rule Rec_a, $(\mu y.P)\{Q/x\} \equiv \mu y.(P\{Q/x\}) \xrightarrow{\alpha} R$ iff $R \equiv R'\{\mu y.(P\{Q/x\})/y\}$ and $P\{Q/x\} \xrightarrow{\alpha} R'$ for some R' iff (by ind.) $R \equiv R'\{\mu y.(P\{Q/x\})/y\}$ and $R' \equiv P'\{Q/x\}$ with $P \xrightarrow{\alpha} P'$ for some R' and P' iff $R \equiv (P'\{Q/x\})\{\mu y.(P\{Q/x\})/y\} \equiv (P'\{\mu y.P/y\})\{Q/x\}$ and $P \xrightarrow{\alpha} P'$ for some P' iff $R \equiv P''\{Q/x\}$ and $P'' \equiv P'\{\mu y.P/y\}$ and $P \xrightarrow{\alpha} P'$ for some P' and P'' iff $\mu y.P \xrightarrow{\alpha} P''$ and $R \equiv P''\{Q/x\}$ for some P''.

3. Clear from the side-condition in the above definition, by induction on the inference of $P \xrightarrow{\alpha}$ resp.

Nil: $\mathbf{0} \xrightarrow{\alpha}$ for no $\alpha \in \mathbb{A}_\tau$, hence $\mathcal{A}(\mathbf{0}) = \emptyset$.

Pref: $\langle \alpha, r \rangle.P \xrightarrow{\beta}$ iff $\alpha = \beta$, hence $\mathcal{A}(\langle \alpha, r \rangle.P) = \{\alpha\}$

Sum: $P_1 + P_2 \xrightarrow{\alpha} \Leftrightarrow P_1 \xrightarrow{\alpha} \vee P_2 \xrightarrow{\alpha} \Leftrightarrow \alpha \in \mathcal{A}(P_1) \vee \alpha \in \mathcal{A}(P_2)$, hence $\mathcal{A}(P_1 + P_2) = \mathcal{A}(P_1) \cup \mathcal{A}(P_2)$.

Par$_1$: $\alpha \notin A \wedge P_1 \|_A P_2 \xrightarrow{\alpha} \Leftrightarrow \alpha \notin A \wedge (P_1 \xrightarrow{\alpha} \vee P_2 \xrightarrow{\alpha}) \Leftrightarrow \alpha \notin A \wedge (\alpha \in \mathcal{A}(P_1) \vee \alpha \in \mathcal{A}(P_2))$, hence iff $\alpha \in \mathcal{A}(P_1) \cup \mathcal{A}(P_2)$

Par$_2$: $\alpha \in A \wedge P_1 \|_A P_2 \xrightarrow{\alpha} \Leftrightarrow \alpha \in A \wedge (P_1 \xrightarrow{\alpha} \wedge P_2 \xrightarrow{\alpha}) \Leftrightarrow \alpha \in A \wedge (\alpha \in \mathcal{A}(P_1) \wedge \alpha \in \mathcal{A}(P_2))$, hence iff $\alpha \in \mathcal{A}(P_1) \cap \mathcal{A}(P_2)$

Rel: $P[\Phi] \xrightarrow{\alpha} \Leftrightarrow \exists \beta \in \Phi^{-1}(\alpha) : P \xrightarrow{\beta} \Leftrightarrow \exists \beta \in \mathcal{A}(P) : \Phi(\beta) = \alpha \Leftrightarrow \alpha \in \Phi(\mathcal{A}(P))$

Rec: $\mu x.P \xrightarrow{\alpha} \Leftrightarrow P \xrightarrow{\alpha} \Leftrightarrow \alpha \in \mathcal{A}(P)$.

□

As a first step to define timed behaviour, we now give operational rules for the passage of 'wait-time': all components of a system participate in a global time step, and this passage of time is recorded for locally activated actions by decreasing their annotated timer in rule Pref_c. Note that time passes disregarding elapsed timers; as in the example at the beginning of this section, this might be necessary for a component when waiting for a synchronization partner, explaining the name 'wait-time'.

Definition 2.5 (*Operational semantics for wait-time*) Via the following SOS-rules, a relation $\overset{\rho}{\rightsquigarrow}_c \subseteq (\tilde{\mathbb{L}} \times \tilde{\mathbb{L}})$ is defined for each $\rho \in \mathbb{T}$:

Nil_c $\dfrac{}{\mathbf{0} \overset{\rho}{\rightsquigarrow}_c \mathbf{0}}$ $\qquad$ Pref_c $\dfrac{r' = \max(r-\rho, 0)}{\langle \alpha, r\rangle.P \overset{\rho}{\rightsquigarrow}_c \langle \alpha, r'\rangle.P}$ $\qquad$ Sum_c $\dfrac{P_1 \overset{\rho}{\rightsquigarrow}_c P_1',\ P_2 \overset{\rho}{\rightsquigarrow}_c P_2'}{P_1 + P_2 \overset{\rho}{\rightsquigarrow}_c P_1' + P_2'}$

Rel_c $\dfrac{P \overset{\rho}{\rightsquigarrow}_c P'}{P[\Phi] \overset{\rho}{\rightsquigarrow}_c P'[\Phi]}$ $\qquad$ Par_c $\dfrac{P_1 \overset{\rho}{\rightsquigarrow}_c P_1',\ P_2 \overset{\rho}{\rightsquigarrow}_c P_2'}{P_1 \|_A P_2 \overset{\rho}{\rightsquigarrow}_c P_1' \|_A P_2'}$ $\qquad$ Rec_c $\dfrac{P \overset{\rho}{\rightsquigarrow}_c P'}{\mu x.P \overset{\rho}{\rightsquigarrow}_c P'\{\mu x.P/x\}}$

Note that a process variable $x \in \mathcal{X}$ – as Ω – has no time semantics, reflecting the fact that an unbound occurrence of a variable means incomplete specification.

Lemma 2.6 (*Closure Properties of Action and Wait-time Transitions*)

1. $P \in \tilde{\mathbb{P}}_c$ $(\tilde{\mathbb{P}})$, $Q \in \tilde{\mathbb{P}}$ and $y \in \mathcal{X}$ time-guarded in P but not free in Q imply y time-guarded in $P\{Q/x\}$.
2. $P, Q \in \tilde{\mathbb{P}}$ and $x \in \mathcal{X}$ imply $P\{Q/x\} \in \tilde{\mathbb{P}}$.
3. $P \in \tilde{\mathbb{P}}_c$, $Q \in \tilde{\mathbb{P}}$ and $x \in \mathcal{X}$ imply $P\{Q/x\} \in \tilde{\mathbb{P}}_c$.
4. For $P \in \tilde{\mathbb{P}}$ $(\mathbb{P})$, $P \overset{\alpha}{\rightarrow} Q$ or $P \overset{1}{\rightsquigarrow}_c Q$ imply $Q \in \tilde{\mathbb{P}}$ $(\mathbb{P})$.
5. For $P \in \tilde{\mathbb{P}}_c$ $(\mathbb{P}_c)$, $P \overset{\alpha}{\rightarrow} Q$ or $P \overset{\rho}{\rightsquigarrow}_c Q$ imply $Q \in \tilde{\mathbb{P}}_c$ $(\mathbb{P}_c)$.
6. $P \in \tilde{\mathbb{P}}_c$ $(\mathbb{P}_c)$ and $P \overset{1}{\rightsquigarrow}_c Q$ imply $Q \in \tilde{\mathbb{P}}$ $(\mathbb{P})$.

Proof: 1. This is immediate since each occurrence of y in $Q \in \tilde{\mathbb{P}}$ is bound and therefore time-guarded.

2. By induction on the structure of P.

Nil,Stop: If $P \equiv \mathbf{0}$ then $P\{Q/x\} \equiv \mathbf{0} \in \tilde{\mathbb{P}}$. If $P \equiv \Omega$ then $P\{Q/x\} \equiv \Omega \in \tilde{\mathbb{P}}$.

Var: $P \equiv z$. If $z \neq x$ then $z\{Q/x\} \equiv z \in \tilde{\mathbb{P}}$. If $z = x$ then $x\{Q/x\} \equiv Q \in \tilde{\mathbb{P}}$.

Pref: $P \equiv \langle \alpha, r\rangle.P_1$. By induction hypothesis $P_1\{Q/x\} \in \tilde{\mathbb{P}}$. Thus also $\langle \alpha, r\rangle.(P_1\{Q/x\}) \equiv (\langle \alpha, r\rangle.P_1)\{Q/x\}$.

Sum: $P \equiv P_1 + P_2$. By induction hypothesis $P_1\{Q/x\} \in \tilde{\mathbb{P}}$ and $P_2\{Q/x\} \in \tilde{\mathbb{P}}$. Thus also $(P_1\{Q/x\}) + (P_2\{Q/x\}) \equiv (P_1 + P_2)\{Q/x\} \in \tilde{\mathbb{P}}$.

Par, Rel: analogously to Sum.

Rec: $P \equiv \mu z.P_1$. If $z = x$ then $(\mu z.P_1)\{Q/x\} \equiv \mu z.P_1 \in \tilde{\mathbb{P}}$. Assume $z \neq x$ and z not free in Q by Barendregt convention. Then $(\mu z.P_1)\{Q/x\} \equiv \mu z.(P_1\{Q/x\})$. By induction hypothesis $P_1\{Q/x\} \in \tilde{\mathbb{P}}$. Since z is time-guarded in P_1 and z not free in Q, by Item 1. we have z is time-guarded in $P_1\{Q/x\}$. Thus $\mu z.(P_1\{Q/x\}) \in \tilde{\mathbb{P}}$.

3. A trivial adaptation of Item 2.

4. By induction on the structure of P.

Nil: $P \equiv \mathbf{0}$. Case $P \overset{\alpha}{\rightarrow} Q$ is not possible while $\mathbf{0} \overset{\rho}{\rightsquigarrow}_c \mathbf{0} \in \mathbb{P}$.

Stop,Var: Cases $P \equiv x$ and $P \equiv \Omega$ are not possible

Pref: $P \equiv \langle \alpha, r\rangle.P_1$. Then $\langle \alpha, r\rangle.P_1 \xrightarrow{\alpha} P_1$ and $\langle \alpha, r\rangle.P_1 \overset{\rho}{\rightsquigarrow}_c \langle \alpha, r'\rangle.P_1$, where $r' = \max(r - \rho, 0)$. Both P_1 and $\langle \alpha, r'\rangle.P_1$ are in $\tilde{\mathbb{P}}$ and if $P \in \mathbb{P}$ then P_1 and $\langle \alpha, r'\rangle.P_1$ are also in $\mathbb{P}$.

Sum: $P \equiv P_1 + P_2$. Then $P_1 + P_2 \xrightarrow{\alpha} P_1'$ if either $P_1 \xrightarrow{\alpha} P_1'$ or $P_2 \xrightarrow{\alpha} P_1'$. Assume $P_1 \xrightarrow{\alpha} P_1'$ (the other case is symmetric). By induction hypothesis $P_1' \in \tilde{\mathbb{P}}$.

If $P_1 + P_2 \overset{\rho}{\rightsquigarrow}_c P_1' + P_2'$ if both $P_1 \overset{\rho}{\rightsquigarrow}_c P_1'$ and $P_2 \overset{\rho}{\rightsquigarrow}_c P_1'$. By induction hypothesis $P_1' \in \tilde{\mathbb{P}}$ and $P_2' \in \tilde{\mathbb{P}}$. Thus also $P_1' + P_2' \in \tilde{\mathbb{P}}$.

Moreover, is $P \in \mathbb{P}$, that is $P_1, P_2 \in \mathbb{P}$, then both P_1' and $P_1' + P_2'$ are also in $\mathbb{P}$.

Par, Rel: analogously to Sum.

Rec: $P \equiv \mu x.P_1$. Assume $\mu x.P_1 \xrightarrow{\alpha} P'\{\mu x.P_1/x\}$ due to $P_1 \xrightarrow{\alpha} P'$. By induction hypothesis $P' \in \tilde{\mathbb{P}}$. By $\mu x.P_1 \in \tilde{\mathbb{P}}$ and Item 2. we have $P'\{\mu x.P_1/x\} \in \tilde{\mathbb{P}}$. Moreover, if $\mu x.P_1 \in \mathbb{P}$ then x is the unique free variable in P_1. Simple inductive reasoning shows that x is the unique free variable in P_1'. Thus $P'\{\mu x.P_1/x\} \in \mathbb{P}$.

Case $\mu x.P_1 \overset{\rho}{\rightsquigarrow}_c P'\{\mu x.P_1/x\}$ is analogous.

5. A trivial adaptation of Item 4.

6. By induction on the structure of P.

Nil: $P \equiv \mathbf{0}$. In this case $\mathbf{0} \overset{1}{\rightsquigarrow}_c \mathbf{0} \in \mathbb{P}$.

Stop,Var: $P \equiv \Omega$. This case is not possible since $\Omega \overset{1}{\not\rightsquigarrow}_c$. Similarly for $P \equiv x$.

Pref: $P \equiv \langle \alpha, r\rangle.P_1$. Then $\langle \alpha, r\rangle.P_1 \overset{1}{\rightsquigarrow}_c \langle \alpha, 0\rangle.P_1 \in \tilde{\mathbb{P}}$. Of course, if $P \in \mathbb{P}_c$ then $\langle \alpha, 0\rangle.P_1 \in \mathbb{P}$.

Sum: $P \equiv P_1 + P_2$. $P_1 + P_2 \overset{1}{\rightsquigarrow}_c Q_1 + Q_2$ then $P_1 \overset{1}{\rightsquigarrow}_c Q_1$ and $P_2 \overset{1}{\rightsquigarrow}_c Q_2$. By induction hypothesis $P_1, P_2 \in \tilde{\mathbb{P}}_c$ ($\mathbb{P}_c$) implies $Q_1, Q_2 \in \tilde{\mathbb{P}}_c$ ($\mathbb{P}_c$). Thus also $Q_1 + Q_2 \in \tilde{\mathbb{P}}_c$ ($\mathbb{P}_c$).

Par,Rel: analogously to Sum.

Rec: $P = \mu x.P_1$. In this case $\mu x.P_1 \in \tilde{\mathbb{P}}$. By the operational semantics $\mu x.P_1 \overset{1}{\rightsquigarrow}_c Q_1\{\mu x.P_1/x\}$ if $P_1 \overset{1}{\rightsquigarrow}_c Q_1$ and $Q_1 \in \tilde{\mathbb{P}}$ by Item 4. Thus, $Q_1\{\mu x.P_1/x\} \in \tilde{\mathbb{P}}$ by Item 2. Of course, if $P \in \mathbb{P}$ then also $Q_1\{\mu x.P_1/x\} \in \mathbb{P}$.

□

Lemma 2.7 (*Properties of passage of wait-time*) $P, P', P'', Q, R \in \tilde{\mathbb{L}}$ be general process terms and $\rho, \rho', \rho + \rho' \in \mathbb{T}$.

1. $P \overset{\rho}{\rightsquigarrow}_c$ if and only if P is guarded, and if $P \overset{\rho}{\rightsquigarrow}_c P'$, then P' is guarded.
2. If $P \overset{\rho}{\rightsquigarrow}_c P'$ and $P \overset{\rho}{\rightsquigarrow}_c P''$, then $P' \equiv P''$.
3. Let $x \in \mathcal{X}$ be guarded in P; then: $P\{Q/x\} \overset{\rho}{\rightsquigarrow}_c R$ if and only if there exists $P' \in \tilde{\mathbb{L}}$ with $P \overset{\rho}{\rightsquigarrow}_c P'$ and $R \equiv P'\{Q/x\}$.
4. If $P \overset{\rho}{\rightsquigarrow}_c P'$, then $\mathcal{A}(P) = \mathcal{A}(P')$.

5. $P \overset{\rho+\rho'}{\rightsquigarrow_c} P''$ if and only if $P \overset{\rho}{\rightsquigarrow_c} P' \overset{\rho'}{\rightsquigarrow_c} P''$ for some P'.

Proof: 1. By induction on the structure of P; informally, the argument is that a composite term can do a time step iff its operands can do it – except for prefix-terms, where the prefix is essentially preserved.

Nil: $\mathbf{0}$ is guarded and $\mathbf{0} \overset{\rho}{\rightsquigarrow_c} \mathbf{0}$.

Var: $P \equiv x$ for $x \in \mathcal{X}$ is not guarded, and for no ρ: $x \overset{\rho}{\rightsquigarrow_c}$.

Pref: $\langle \alpha, r \rangle.P$ is guarded and $\langle \alpha, r \rangle.P \overset{\rho}{\rightsquigarrow_c} \langle \alpha, r' \rangle.P$ with $r' = \max(r - \rho, 0)$, and $\langle \alpha, r' \rangle.P$ is guarded, too.

Sum: $P \equiv P_1 + P_2 \overset{\rho}{\rightsquigarrow_c}$ iff for $i = 1, 2$: $P_i \overset{\rho}{\rightsquigarrow_c} P_i'$ for some P_i', hence by ind. iff both P_1 and P_2 are guarded, in which case both P_1' and P_2' are guarded, too. Thus, $P \equiv P_1 + P_2 \overset{\rho}{\rightsquigarrow_c}$ iff P is guarded and if $P \overset{\rho}{\rightsquigarrow_c} P' \equiv P_1' + P_2'$, then P' is guarded.

Par: analogously to Sum.

Rel: similar to Sum.

Rec: $\mu x.P$ is guarded iff P is guarded iff by ind. $P \overset{\rho}{\rightsquigarrow_c} P'$ for some guarded P' iff $\mu x.P \overset{\rho}{\rightsquigarrow_c} P'\{\mu x.P/x\}$, which is guarded, too.

2. by induction on the inference of $P \overset{\rho}{\rightsquigarrow_c} P'$, $P \overset{\rho}{\rightsquigarrow_c} P''$ resp.

3. similar to 2.4.2

4. By induction on the inference of $P \overset{\rho}{\rightsquigarrow_c} P'$. We only consider the Rec-case in the induction, since the other cases are clear:

By rule Rec_c, $\mu x.P \overset{\rho}{\rightsquigarrow_c} R$ implies $R \equiv P'\{\mu x.P/x\}$ and $P \overset{\rho}{\rightsquigarrow_c} P'$ for some P' and by 1., x is guarded in P and P'. Now by induction and Proposition 2.4.2 and .3: $\mathcal{A}(\mu x.P) = \mathcal{A}(P) = \mathcal{A}(P') = \mathcal{A}(P'\{\mu x.P/x\}) = \mathcal{A}(R)$.

5. Induction:

Nil: $\mathbf{0} \overset{\rho+\rho'}{\rightsquigarrow_c} \mathbf{0}$ and $\mathbf{0} \overset{\rho}{\rightsquigarrow_c} \mathbf{0} \overset{\rho'}{\rightsquigarrow_c} \mathbf{0}$.

Var, Stop: For $x \in \mathcal{X}$: neither $x \overset{\rho}{\rightsquigarrow_c}$ nor $x \overset{\rho+\rho'}{\rightsquigarrow_c}$; similar for Ω.

Pref: $\langle \alpha, r \rangle.P \overset{\rho+\rho'}{\rightsquigarrow_c} \langle \alpha, \max(r - (\rho + \rho'), 0) \rangle.P$ and $\langle \alpha, r \rangle.P \overset{\rho}{\rightsquigarrow_c} \langle \alpha, \max(r - \rho, 0) \rangle.P \overset{\rho'}{\rightsquigarrow_c} \langle \alpha, \max(\max(r - \rho, 0) - \rho', 0) \rangle.P$ and $\max(\max(r - \rho, 0) - \rho', 0) = \max(r - (\rho + \rho'), 0)$.

Sum: $P_1 + P_2 \overset{\rho+\rho'}{\rightsquigarrow_c} P_1'' + P_2'' \Leftrightarrow \forall_{i=1,2}\ P_i \overset{\rho+\rho'}{\rightsquigarrow_c} P_i'' \Leftrightarrow$ (ind.) $\forall_{i=1,2} \exists P_i' : P_i \overset{\rho}{\rightsquigarrow_c} P_i' \overset{\rho'}{\rightsquigarrow_c} P_i'' \Leftrightarrow$ $\forall_{i=1,2} \exists P_i' : P_1 + P_2 \overset{\rho}{\rightsquigarrow_c} P_1' + P_2' \overset{\rho'}{\rightsquigarrow_c} P_1'' + P_2''$.

Par: analogously to Sum.

Sum, Par, Rel: induction

Rec: $\mu x.P \overset{\rho+\rho'}{\rightsquigarrow}_c R$ iff (by rule Rec_c) $P \overset{\rho+\rho'}{\rightsquigarrow}_c P''$ for some P'' and $R \equiv P''\{\mu x.P/x\}$ iff (by induction) $\exists P', P'' : P \overset{\rho}{\rightsquigarrow}_c P' \overset{\rho'}{\rightsquigarrow}_c P'' \wedge R \equiv P''\{\mu x.P/x\}$ iff (again by rule Rec_c and by 3. since x is guarded in P and P') $\exists P', P'' : \mu x.P \overset{\rho}{\rightsquigarrow}_c P'\{\mu x.P/x\} \overset{\rho'}{\rightsquigarrow}_c P''\{\mu x.P/x\} \equiv R$.

□

The operational semantics of wait-time allows general processes to wait forever, but our intention was that an urgent action has to occur or be disabled (– unless it has to wait for a synchronization partner). We will enforce this using an auxiliary function that calculates for a given action α its residual time $\mathcal{R}(\alpha, P)$ in a general process term P, i.e. the time until it becomes urgent.

Definition 2.8 (*Residual time of actions and general process terms*) The *residual time* $\mathcal{R}(\alpha, P)$ of an action $\alpha \in \mathbb{A}_\tau$ in a guarded general process term $P \in \tilde{\mathbb{L}}$ is defined by:

Nil, Stop, Var: $\mathcal{R}(\alpha, \mathbf{0}) = 1$

Pref: $\mathcal{R}(\alpha, \langle \beta, r \rangle.P) = \begin{cases} r & \text{if } \alpha = \beta \\ 1 & \text{otherwise} \end{cases}$

Sum: $\mathcal{R}(\alpha, P_1 + P_2) = \min(\mathcal{R}(\alpha, P_1), \mathcal{R}(\alpha, P_2))$

Par: $\mathcal{R}(\alpha, P_1 \|_A P_2) = \begin{cases} \max(\mathcal{R}(\alpha, P_1), \mathcal{R}(\alpha, P_2)) & \text{if } \alpha \in A \\ \min(\mathcal{R}(\alpha, P_1), \mathcal{R}(\alpha, P_2)) & \text{if } \alpha \notin A \end{cases}$

Rel: $\mathcal{R}(\alpha, P[\Phi]) = \min\{\mathcal{R}(\beta, P) \mid \beta \in \Phi^{-1}(\alpha)\}$

Rec: $\mathcal{R}(\alpha, \mu x.P) = \mathcal{R}(\alpha, P)$

In this definition, we put $\min \emptyset := 1$. The residual time of a guarded general process term $P \in \tilde{\mathbb{L}}$ is $\mathcal{R}(P) = \min\{\mathcal{R}(\alpha, P) \mid \alpha \in \mathcal{A}(P)\}$. $\mathcal{R}(\alpha, P)$ and $\mathcal{R}(P)$ for unguarded P are irrelevant and, hence, undefined.

We have chosen $\mathcal{R}(\alpha, \mathbf{0}) = 1$ mainly for technical reasons (cf. Proposition 2.9.1 below). The Par-case will realize the desired behaviour of waiting in a parallel composition: if P_1 and P_2 have to synchronize on α, then the residual time of α in $P_1 \|_A P_2$ is determined by the 'slower' component with larger residual time; if P_1 and P_2 do not have to synchronize on α, the 'faster' component determines the maximal possible delay of α in $P_1 \|_A P_2$.

Observe that in the Rel-case $\Phi^{-1}(\alpha)$ may be empty (where $\min \emptyset = 1$) or infinite; for the latter case, we will show below that for any $P \in \tilde{\mathbb{L}}$ there are only finitely many $\beta \in \mathbb{A}_\tau$ with $\mathcal{R}(\beta, P) \neq 1$ (Proposition 2.9.1 together with Proposition 2.4.1), such that the set $\{\mathcal{R}(\beta, P) \mid \beta \in \Phi^{-1}(\alpha)\}$ is finite and $\mathcal{R}(\alpha, P[\Phi])$ exists. Similarly, $\mathcal{R}(P)$ exists for each general process term P, and, hence, the residual time is well-defined in all cases.

In the following proposition, we ascertain that only activated actions of a general process term can have a residual time less than 1, and that the residual time of each action is preserved under substitution of guarded variables. Additionally, we show how the residual time of a general process term can be calculated directly from the residual times of its components, provided there is no parallel composition with synchronisation:

Proposition 2.9 Let $P, Q \in \tilde{\mathbb{L}}$ be guarded general process terms, $\alpha \in \mathbb{A}_\tau$ and $x \in \mathcal{X}$.

1. $\mathcal{R}(\alpha, P) \in \mathbb{T}$, and $\mathcal{R}(\alpha, P) \neq 1$ implies $\alpha \in \mathcal{A}(P)$.

2. $\mathcal{R}(\alpha, P) = \mathcal{R}(\alpha, P\{Q/x\})$, thus $\mathcal{R}(P) = \mathcal{R}(P\{Q/x\})$.

3. Except for parallel composition, $\mathcal{R}(P)$ may be calculated directly:

$$\mathcal{R}(\mathbf{0}) = 1$$
$$\mathcal{R}(\langle \alpha, r \rangle.P) = r$$
$$\mathcal{R}(P + Q) = \min(\mathcal{R}(P), \mathcal{R}(Q))$$
$$\mathcal{R}(P[\Phi]) = \mathcal{R}(\mu x.P) = \mathcal{R}(P)$$

Proof: **1.** Apply induction on the structure of P; observe Proposition 2.4.3 and that $\mathcal{R}(\alpha, P\|_A Q) \neq 1$ iff either $\alpha \in A$ and $\mathcal{R}(\alpha, P) \neq 1 \neq \mathcal{R}(\alpha, Q)$ or $\alpha \notin A$ and $\mathcal{R}(\alpha, P) \neq 1$ or $\mathcal{R}(\alpha, Q) \neq 1$; note that the respective induction step shows that $\mathcal{R}(\alpha, P[\Phi])$ is well-defined.

2. Induction on the structure of P.

3. Mostly clear; we exploit the finiteness of $\mathcal{A}(P)$ and the restriction on general relabelling functions Φ in order to swap minima:

- $\mathcal{R}(P_1 + P_2) = \min_{\alpha \in \mathcal{A}(P_1+P_2)} \mathcal{R}(\alpha, P_1 + P_2) = \min_{\alpha \in \mathcal{A}(P_1) \cup \mathcal{A}(P_2)} \min(\mathcal{R}(\alpha, P_1), \mathcal{R}(\alpha, P_2))$
  $= \min(\min_{\alpha \in \mathcal{A}(P_1) \cup \mathcal{A}(P_2)} \mathcal{R}(\alpha, P_1), \min_{\alpha \in \mathcal{A}(P_1) \cup \mathcal{A}(P_2)} \mathcal{R}(\alpha, P_2))$
  $\stackrel{1.}{=} \min(\min_{\alpha \in \mathcal{A}(P_1)} \mathcal{R}(\alpha, P_1), \min_{\alpha \in \mathcal{A}(P_2)} \mathcal{R}(\alpha, P_2)) = \min(\mathcal{R}(P_1), \mathcal{R}(P_2))$.

- $\mathcal{R}(P[\Phi]) = \min_{\alpha \in \mathcal{A}(P[\Phi])} \mathcal{R}(\alpha, P[\Phi]) = \min_{\alpha \in \Phi(\mathcal{A}(P))} \min_{\beta \in \Phi^{-1}(\alpha) \cap \mathcal{A}(P)} \mathcal{R}(\beta, P) \stackrel{1.}{=}$
  $\min_{\beta \in \mathcal{A}(P)} \mathcal{R}(\beta, P) = \mathcal{R}(P)$.

□

The effect of waiting on the residual time of activated actions is described by the following lemma: if time advances by amount ρ, then the residual time of an activated action is decreased by the same amount, unless it is already less than ρ, in which case it is zero afterwards. This behaviour is realized locally by rule Pref_c of Definition 2.5.

Lemma 2.10 For general process terms $P, P' \in \tilde{\mathbb{L}}$ and $\rho \in \mathbb{T}$ let $P \stackrel{\rho}{\rightsquigarrow}_c P'$; then for all $\alpha \in \mathcal{A}(P) = \mathcal{A}(P')$ we have either $\mathcal{R}(\alpha, P) - \mathcal{R}(\alpha, P') = \rho$, or $\mathcal{R}(\alpha, P) < \rho$ and $\mathcal{R}(\alpha, P') = 0$.

Proof: In this proof, we will deal with minima and maxima when calculating residual times. In these calculations, we will often use the following properties without mentioning it:

Let I be a finite set, $\rho \in \mathbb{T}$ and $(x_i)_{i \in I}$, $(y_i)_{i \in I}$ be families of real numbers.

1. $\min_{i \in I}(x_i - y_i) \leq \min_{i \in I} x_i - \min_{i \in I} y_i$.
2. If $x_i - y_i \leq \rho$ for all $i \in I$, then $\min_{i \in I} x_i - \min_{i \in I} y_i \leq \rho$.
3. If $x_i - y_i = \rho$ for all $i \in I$, then $\min_{i \in I} x_i - \min_{i \in I} y_i = \rho$.
4. $\max_{i \in I}(x_i - y_i) \geq \max_{i \in I} x_i - \max_{i \in I} y_i$.
5. If $x_i - y_i \leq \rho$ for all $i \in I$, then $\max_{i \in I} x_i - \max_{i \in I} y_i \leq \rho$.
6. If $x_i - y_i = \rho$ for all $i \in I$, then $\max_{i \in I} x_i - \max_{i \in I} y_i = \rho$.

Proof: 1. Let $\min_{i\in I}\, x_i = x_j$ with $j \in I$; then $\min_{i\in I}\,(x_i - y_i) \leq x_j - y_j \leq x_j - \min_{i\in I}\, y_i = \min_{i\in I}\, x_i - \min_{i\in I}\, y_i$.

2. Let $\min_{i\in I}\, y_i = y_k$ with $k \in I$; then $\min_{i\in I}\, x_i - \min_{i\in I}\, y_i = \min_{i\in I}\, x_i - y_k \leq x_k - y_k \leq \rho$.

3. Follows from 1. and 2.

4. Let $\max_{i\in I}\, x_i = x_j$ with $j \in I$; then $\max_{i\in I}\,(x_i - y_i) \geq x_j - y_j \geq x_j - \max_{i\in I}\, y_i = \max_{i\in I}\, x_i - \max_{i\in I}\, y_i$.

5. Follows from 4.

6. $\max_{i\in I}\, x_i - \max_{i\in I}\, y_i \leq \rho$ by 4. Let $\max_{i\in I}\, y_i = y_k$ with $k \in I$; then $\rho = x_k - y_k \leq \max_{i\in I}\, x_i - y_k = \max_{i\in I}\, x_i - \max_{i\in I}\, y_i$. □

We now perform induction on the inference of $P \overset{\rho}{\rightsquigarrow}_c P'$, using Lemma 2.7.4, which also gives $\mathcal{A}(P) = \mathcal{A}(P')$; interesting cases are:

Sum: $P_1 + P_2 \overset{\rho}{\rightsquigarrow}_c P_1' + P_2' \Rightarrow \forall_{i=1,2}\, P_i \overset{\rho}{\rightsquigarrow}_c P_i' \Rightarrow \forall_{i=1,2}\forall_{\alpha\in\mathcal{A}(P_i)=\mathcal{A}(P_i')}\, (\mathcal{R}(\alpha, P_i) - \mathcal{R}(\alpha, P_i') = \rho \vee (\mathcal{R}(\alpha, P_i) < \rho \wedge \mathcal{R}(\alpha, P_i') = 0))$. For $\alpha \in \mathcal{A}(P_1 + P_2) = \mathcal{A}(P_1' + P_2')$ one of the following cases applies:

1. $\exists_{i=1,2}\, \mathcal{R}(\alpha, P_i) < \rho \wedge \mathcal{R}(\alpha, P_i') = 0$; then $\mathcal{R}(\alpha, P_1 + P_2) = \min_{i=1,2} \mathcal{R}(\alpha, P_i) < \rho$ and $\mathcal{R}(\alpha, P_1' + P_2') = \min_{i=1,2} \mathcal{R}(\alpha, P_i') = 0$.
2. $\forall_{i=1,2}\, \mathcal{R}(\alpha, P_i) - \mathcal{R}(\alpha, P_i') = \rho$; then $\mathcal{R}(\alpha, P_1 + P_2) - \mathcal{R}(\alpha, P_1' + P_2') = \min_{i=1,2} \mathcal{R}(\alpha, P_i) - \min_{i=1,2} \mathcal{R}(\alpha, P_i') = \rho$.
3. $\mathcal{R}(\alpha, P_1) - \mathcal{R}(\alpha, P_1') = \rho \wedge \mathcal{R}(\alpha, P_2) = \mathcal{R}(\alpha, P_2') = 1$ by 2.9.1 (or vice versa); then $\mathcal{R}(\alpha, P_1 + P_2) - \mathcal{R}(\alpha, P_1' + P_2') = \min_{i=1,2} \mathcal{R}(\alpha, P_i) - \min_{i=1,2} \mathcal{R}(\alpha, P_i') = \mathcal{R}(\alpha, P_1) - \mathcal{R}(\alpha, P_1') = \rho$.

Par: $P_1 \|_A P_2 \overset{\rho}{\rightsquigarrow}_c P_1' \|_A P_2' \Rightarrow \forall_{i=1,2}\, P_i \overset{\rho}{\rightsquigarrow}_c P_i'$.
For any $a \in (\mathcal{A}(P_1) \cap \mathcal{A}(P_2)) \cap A$ by ind. one of the following cases applies:

1. $\forall_{i=1,2}\, \mathcal{R}(a, P_i) < \rho \wedge \mathcal{R}(a, P_i') = 0$; then $\mathcal{R}(a, P_1 \|_A P_2) = \max_{i=1,2} \mathcal{R}(a, P_i) < \rho$ and $\mathcal{R}(a, P_1' \|_A P_2') = \max_{i=1,2} \mathcal{R}(a, P_i') = 0$.
2. $\mathcal{R}(a, P_1) - \mathcal{R}(a, P_1') = \rho \wedge \mathcal{R}(a, P_1) \geq \mathcal{R}(a, P_2)$ (or vice versa);
   then $\mathcal{R}(a, P_1 \|_A P_2) - \mathcal{R}(a, P_1' \|_A P_2') = \max_{i=1,2} \mathcal{R}(a, P_i) - \max_{i=1,2} \mathcal{R}(a, P_i') = \mathcal{R}(a, P_1) - \mathcal{R}(a, P_1') = \rho$.

For $\alpha \in (\mathcal{A}(P_1) \cup \mathcal{A}(P_2)) \setminus A$ we argue as for sum.

Rel: $P[\Phi] \overset{\rho}{\rightsquigarrow}_c P'[\Phi] \Rightarrow P \overset{\rho}{\rightsquigarrow}_c P'$
For $\alpha \in \mathcal{A}(P[\Phi])$ we have one of the following cases:

1. $\exists \beta \in \mathcal{A}(P) \cap \Phi^{-1}(\alpha)$: $\mathcal{R}(\beta, P) < \rho \wedge \mathcal{R}(\beta, P') = 0$, then $\mathcal{R}(\alpha, P[\Phi]) < \rho \wedge \mathcal{R}(\alpha, P'[\Phi]) = 0$
2. otherwise:
   $\mathcal{R}(\alpha, P[\Phi]) - \mathcal{R}(\alpha, P'[\Phi]) = \min_{\beta\in\Phi^{-1}(\alpha)} \mathcal{R}(\beta, P) - \min_{\beta\in\Phi^{-1}(\alpha)} \mathcal{R}(\beta, P') = \min_{\beta\in\Phi^{-1}(\alpha)\cap\mathcal{A}(P)} \mathcal{R}(\beta, P) - \min_{\beta\in\Phi^{-1}(\alpha)\cap\mathcal{A}(P)} \mathcal{R}(\beta, P') = \rho$

Rec: $\mu x.P \overset{\rho}{\rightsquigarrow}_c R$ implies $R \equiv P'\{\mu x.P/x\}$ and $P \overset{\rho}{\rightsquigarrow}_c P'$ for some P' by rule Rec_c, hence induction yields

$\forall_{\alpha\in\mathcal{A}(P)}\ \mathcal{R}(\alpha,P)-\mathcal{R}(\alpha,P')=\rho\ \vee\ (\mathcal{R}(\alpha,P)<\rho\ \wedge\ \mathcal{R}(\alpha,P')=0)$
and since P' is guarded by Lemma 2.7.1, with Proposition 2.9.2 follows
$\forall_{\alpha\in\mathcal{A}(\mu x.P)}\mathcal{R}(\alpha,\mu x.P)-\mathcal{R}(\alpha,P'\{\mu x.P/x\})=\mathcal{R}(\alpha,P)-\mathcal{R}(\alpha,P')=\rho$ or
$\mathcal{R}(\alpha,\mu x.P)=\mathcal{R}(\alpha,P)<\rho\ \wedge\ \mathcal{R}(\alpha,P'\{\mu x.P/x\})=\mathcal{R}(\alpha,P')=0$.

The residual time is the time a stand-alone process can idle; thus, we can use it to restrict wait-time to the timed behaviour we had in mind originally and which we call 'idle-time'. Alternatively, idle-time could have been defined via SOS-rules intertwined with the rules for wait-time.

Definition 2.11 (*Passage of idle-time*) For (guarded) $P,P'\in\tilde{\mathbb{L}}$ and $\rho\in\mathbb{T}$ we write $P\stackrel{\rho}{\rightarrow}_c P'$ if $P\stackrel{\rho}{\rightsquigarrow}_c P'$ and $\rho\leq\mathcal{R}(P)$.

Most of the properties of wait-time stated in Lemma 2.7 carry over to idle-time analogously, gathered in Proposition 2.12 below. Note that general processes without activated actions may idle for an arbitrary amount of time by 4., 1. and 3., but if there are activated actions, they may idle at most for time 1 by 5., 1. and 3. Also observe that 3. demonstrates to some degree that functional and timed behaviour are orthogonal: passage of time can neither deactivate actions (persistency) nor can it activate new actions, i.e. we do not have timeouts.

Proposition 2.12 (*Properties of idle-time*) Let $P,P',P'',Q,R\in\tilde{\mathbb{L}}$ be general process terms and $\rho,\rho',\rho+\rho'\in\mathbb{T}$.

1. $P\stackrel{\rho}{\rightarrow}_c$ iff P is guarded and $\rho\leq\mathcal{R}(P)$, and P' is guarded if $P\stackrel{\rho}{\rightarrow}_c P'$.
2. If $P\stackrel{\rho}{\rightarrow}_c P'$ and $P\stackrel{\rho}{\rightarrow}_c P''$, then $P'\equiv P''$. *(time-determinism)*
3. If $P\stackrel{\rho}{\rightarrow}_c P'$, then $\mathcal{A}(P)=\mathcal{A}(P')$. *(orthogonality)*
4. If $\mathcal{A}(P)=\emptyset$ and $P\stackrel{\rho}{\rightarrow}_c P'$, then $\mathcal{R}(P)=\mathcal{R}(P')=1$.
5. If $\mathcal{A}(P)\neq\emptyset$ and $P\stackrel{\rho}{\rightarrow}_c P'$, then $\mathcal{R}(P)-\mathcal{R}(P')=\rho$
6. $P\stackrel{\rho+\rho'}{\rightarrow}_c P''$ if and only if $P\stackrel{\rho}{\rightarrow}_c P'\stackrel{\rho'}{\rightarrow}_c P''$ for some P'. *(continuity)*

Proof: Use Definition 2.11 and:

1. Lemma 2.7.1.
2. Lemma 2.7.2.
3. Lemma 2.7.4.
4. Implication of 3. and Proposition 2.9.1.
5. Lemma 2.10 and Definition 2.8.
6. $P\stackrel{\rho+\rho'}{\rightarrow}_c P''$
$\Leftrightarrow P\stackrel{\rho+\rho'}{\rightsquigarrow}_c P''\ \wedge\ \rho+\rho'\leq\mathcal{R}(P)$
$\Leftrightarrow \exists P':P\stackrel{\rho}{\rightsquigarrow}_c P'\stackrel{\rho'}{\rightsquigarrow}_c P''\ \wedge\ \rho\leq\mathcal{R}(P)\ \wedge\ \rho'\leq\mathcal{R}(P)-\rho$ (by Lemma 2.7.5)
$\Leftrightarrow \exists P':P\stackrel{\rho}{\rightarrow}_c P'\stackrel{\rho'}{\rightsquigarrow}_c P''\ \wedge\ \rho'\leq\mathcal{R}(P')$ (5. and 4., also using $\rho+\rho'\leq 1$)
$\Leftrightarrow \exists P':P\stackrel{\rho}{\rightarrow}_c P'\stackrel{\rho'}{\rightarrow}_c P''$ □

Both, purely functional and timed behaviour of processes will now be combined in the continuous language of processes. As usual, we will abstract from internal behaviour; but note that internal actions gain some 'visibility' in timed behaviour, since their presence possibly allows to pass more time in between the occurrence of visible actions. For technical reasons, we also need a continuous language that records τ's when we compare processes w.r.t. their temporal progress in the next section. Note that we use λ (and not ε) to denote the empty sequence.

Definition 2.13 (*Continuous language of processes*) Let $P, P' \in \tilde{\mathbb{L}}$ be c-process terms. We write $P \xrightarrow{\varepsilon}_c P'$ if either $\varepsilon \in \mathbb{A}_\tau$ and $P \xrightarrow{\varepsilon} P'$, or $\varepsilon \in \mathbb{T}$ and $P \xrightarrow{\varepsilon}_c P'$. We extend this to sequences w and write $P \xrightarrow{w}_c P'$ if $P \equiv P'$ and $w = \lambda$ (the empty sequence) or there exist $Q \in \tilde{\mathbb{L}}$ and $\varepsilon \in (\mathbb{A}_\tau \cup \mathbb{T})$ such that $P \xrightarrow{\varepsilon}_c Q \xrightarrow{w'}_c P'$ and $w = \varepsilon w'$.

For a sequence $w \in (\mathbb{A}_\tau \cup \mathbb{T})^*$ let w/τ be the sequence w with all τ's removed, let $act(w)$ be the sequence of elements from $\mathbb{A}_\tau$ in w, and let the *duration* $\zeta(w)$ of w be the sum of time steps in w; note that $\zeta(w/\tau) = \zeta(w)$. We write $P \stackrel{v}{\Rightarrow}_c P'$, if $P \xrightarrow{w}_c P'$ and $v = w/\tau$.

For a general process $P \in \mathbb{L}$ we define $\mathsf{CL}_\tau(P) = \{w \mid P \xrightarrow{w}_c\}$ to be the *continuous(ly timed) τ-language*, containing the *continuous τ-traces* of P, and $\mathsf{CL}(P) = \{w \mid P \stackrel{w}{\Rightarrow}_c\}$ to be the *continuous language*, containing the *continuous traces* of P.

Note that $\mathsf{CL}_\tau(P) = \mathsf{CL}(P) = \{\lambda\}$ for P not guarded.

Based on the continuous language of general processes, we are now ready to define timed testing and to relate general processes w.r.t. their efficiency, thereby defining an *efficiency preorder*:

Definition 2.14 (*continuously timed tests*) A discrete Ω-free process $P \in \mathbb{P}$ is *testable* if ω does not occur in P. Any *initial* process $O \in \mathbb{P}_1$ may serve as a *test process (observer)*. We write $\|$ for $\|_{\mathbb{A}\setminus\{\omega\}}$.

A *general timed test* is a pair (O, R), where O is a test process and $R \in \mathbb{R}_0^+$ is the *real time bound*. A testable process P *c-satisfies* a general timed test (P *must*$_c$ (O, R)), if each $w \in \mathsf{CL}(\tau.P \| O)$ with $\zeta(w) > R$ contains some ω.

For testable processes P and Q, we call P a *continuously faster implementation* of Q, written $P \sqsupseteq_c Q$, if P c-satisfies all general timed tests that Q c-satisfies.

Note that in contrast to e.g. [12], execution and not only activation of an ω is necessary for satisfaction of a c-timed test. Usually, one considers the behaviour of $P \| O$ when defining a test. This is also done in [15], where it is shown that surprisingly the resulting efficiency preorder is not a precongruence for prefixing and therefore has to be refined afterwards. In order to avoid this complication, we have chosen $\tau.P \| O$ instead (which is shorthand for $(\langle \tau, 1 \rangle.P) \|_{\mathbb{A}\setminus\{\omega\}} O$), getting the refined semantics directly. From an intuitive point of view, the additional τ-prefix represents some internal setup activity before the actual test begins.

In [16] we only tested initial processes, whereas here we more generally test discrete processes. In a testing approach, extending the class of processes under consideration does not always lead to a generalization, since the additional observers can add discriminating power. Here, we really generalize the result from [16], since we extend the class of testable processes but not the class of observers; our results imply that in our setting additional discrete observers would not 'see' more.

Runs with duration less than R may not contain all actions that occur up to time R; hence we only consider runs with a duration greater than the time bound R for test satisfaction.

The operational idea behind this is that – when performing a test – one should certainly wait until time R is up before declaring the test a failure. If an Ω-free continuous process (like $\tau.P \| O$) could have a time-stop, waiting would not be possible and our definition of c-satisfaction would be questionable; we will see below in Corollary 3.10 that this is not the case.

At this point, it is by no means clear how to check $P \sqsupseteq_c Q$ for given testable P and Q. Obviously, it is impossible to apply the definition directly, since there are already uncountably many time bounds and, hence, general timed tests to apply. And even if we could decide $P \sqsupseteq_c Q$ from $\mathsf{CL}(P)$ and $\mathsf{CL}(Q)$ only (which is not the case), $\mathsf{CL}(P)$ and $\mathsf{CL}(Q)$ are still uncountable and hard to handle.

3 Time-Stops

In this section, we will show that most processes in $\tilde{\mathbb{P}}_c$ do not show the time-stop phenomenon. A process is a time-stop, if there is a bound on the time that can elapse along its runs (i.e. all infinite runs are Zeno-runs); a process has a time-stop if along a computation it can reach a time-stop. The following definition introduces this new notion:

Definition 3.1 (*Time-Stops*) Let $P \in \tilde{\mathbb{P}}_c$. Then:

- We say that P *is* a *time-stop* if there is some $n \in \mathbb{N}$ such that $P \xrightarrow{w}_c$ implies $\zeta(w) \leq n$;
- We say that P *has* a *time-stop* if there is some $P' \in \tilde{\mathbb{P}}_c$ such that $P \xrightarrow{w}_c P'$ and P' is a time-stop.

In general, open terms (those containing free variables) and terms containing the Ω-process suffer from this phenomenon, because variables $x \in \mathcal{X}$ and Ω do not let time elapse according to the operational rules in Definition 2.5. Thus, according to Definition 3.1, simple processes like $\langle \alpha, r \rangle.x$ or $\langle \alpha, r \rangle.\Omega$ have a time-stop.

The rest of this section is devoted to proving that Ω-free $\mathbb{P}$-processes do not have time-stops. We need some new notation and preliminary results.

First of all we introduce a function *bound* which, roughly speaking, counts the number of prefixes with timer less than 1 that are not in the scope of a prefix with timer 1, and we state some properties that lead to our results at the end of the section.

Definition 3.2 (*Function Bound*) The bound of a process $P \in \tilde{\mathbb{P}}_c$, $\mathrm{bound}(P)$, is defined by:

Nil, Stop, Var:	$\mathrm{bound}(\mathbf{0}) = \mathrm{bound}(\Omega) = \mathrm{bound}(x) = 0$
Pref:	$\mathrm{bound}(\langle \alpha, r \rangle.P) = \begin{cases} 1 + \mathrm{bound}(P) & \text{if } r < 1 \\ 0 & \text{otherwise} \end{cases}$
Sum:	$\mathrm{bound}(P_1 + P_2) = \max(\mathrm{bound}(P_1), \mathrm{bound}(P_2))$
Par:	$\mathrm{bound}(P_1 \|_A P_2) = \mathrm{bound}(P_1) + \mathrm{bound}(P_2)$
Rel:	$\mathrm{bound}(P[\Phi]) = \mathrm{bound}(P)$
Rec:	$\mathrm{bound}(\mu x.P) = \mathrm{bound}(P)$

Lemma 3.3 Let $P \in \tilde{\mathbb{P}}_c$. Then $\mathrm{bound}(P)$ is finite.

Proof: A simple induction on the structure of P. □

Proposition 3.4 Let P be a time-guarded $\tilde{\mathbb{P}}_c$-($\tilde{\mathbb{P}}$-)term, Q be a time-guarded $\tilde{\mathbb{P}}$-term. Let x be a process variable. Then:

(1) $P\{Q/x\}$ is a time-guarded $\tilde{\mathbb{P}}_c$-($\tilde{\mathbb{P}}$-)term;

(2) $\mathrm{bound}(P\{Q/x\}) = \mathrm{bound}(P)$.

Proof: Prove the three statements together by induction on the structure of P. Actually, we only consider the case P being a time-guarded $\tilde{\mathbb{P}}$-term since the proof for $P \in \tilde{\mathbb{P}}_c$ is completely similar.

Nil: $P \equiv \mathbf{0}$. In such a case $P\{Q/x\} = \mathbf{0}$. Hence $P\{Q/x\}$ is time-guarded and $\mathrm{bound}(P\{Q/x\}) = \mathrm{bound}(P)$.

Stop,Var: Cases $P \equiv \Omega$ and $P \equiv y$ are not possible because P is a time-guarded process.

Pref: $P \equiv \langle \alpha, r\rangle.P_1$. We distinguish two cases: $r = 1$ and $r < 1$.

- Assume $r = 1$. Then $(\langle \alpha, 1\rangle.P_1)\{Q/x\} = \langle \alpha, 1\rangle.(P_1\{Q/x\})$ and $P_1\{Q/x\} \in \tilde{\mathbb{P}}$. Hence, $\langle \alpha, 1\rangle.(P_1\{Q/x\}) \in \tilde{\mathbb{P}}$ and it is time-guarded because of the initial prefix. Moreover, $\mathrm{bound}(P\{Q/x\}) = 0 = \mathrm{bound}(P)$.
- Assume $r < 1$. Then $(\langle \alpha, r\rangle.P_1)\{Q/x\} = \langle \alpha, r\rangle.(P_1\{Q/x\})$ and $P_1\{Q/x\} \in \tilde{\mathbb{P}}$. P time-guarded implies P_1 time-guarded. By induction hypothesis $P_1\{Q/x\}$ is a time- guarded $\tilde{\mathbb{P}}$-term. Hence, $\langle \alpha, r\rangle.(P_1\{Q/x\}) \in \tilde{\mathbb{P}}$ and it is time-guarded. Always by induction hypothesis we have $\mathrm{bound}(P_1\{Q/x\}) = \mathrm{bound}(P_1)$. Thus also $\mathrm{bound}(\langle \alpha, r\rangle.(P_1\{Q/x\})) = 1 + \mathrm{bound}(P_1\{Q/x\}) = 1 + \mathrm{bound}(P_1)$ which is in turn equal to $\mathrm{bound}(\langle \alpha, r\rangle.P_1)$.

Sum: $P \equiv P_1 + P_2$. Of course, P_1 and P_2 are time-guarded $\tilde{\mathbb{P}}$-terms. Moreover, $(P_1 + P_2)\{Q/x\} = (P_1\{Q/x\}) + (P_2\{Q/x\})$. By induction hypothesis $P_1\{Q/x\}$ and $P_2\{Q/x\}$ are time-guarded $\tilde{\mathbb{P}}$-terms. Hence also $(P_1 + P_2)\{Q/x\} = (P_1\{Q/x\}) + (P_2\{Q/x\})$ is a time-guarded $\tilde{\mathbb{P}}$-term. By induction hypothesis $\mathrm{bound}(P_1\{Q/x\}) = \mathrm{bound}(P_1)$ and $\mathrm{bound}(P_2\{Q/x\}) = \mathrm{bound}(P_2)$. Hence, $\mathrm{bound}((P_1 + P_2)\{Q/x\}) = \max\{\mathrm{bound}(P_1\{Q/x\}), \mathrm{bound}(P_2\{Q/x\})\} = \max\{\mathrm{bound}(P_1), \mathrm{bound}(P_2)\} = \mathrm{bound}(P_1 + P_2)$.

Par,Rel: analogous to Sum.

Rec: $P \equiv \mu y.P_1$. We distinguish two cases: $x = y$ and $x \neq y$.

In the former case, $x = y$, $P \equiv P\{Q/x\}$ and we are done.

In the latter case, $x \neq y$, we have $(\mu y.P_1)\{Q/x\} = \mu y.(P_1\{Q/x\})$. By induction hypothesis, we have that $P_1\{Q/x\}$ is a time-guarded $\tilde{\mathbb{P}}$-term and $\mathrm{bound}(P_1\{Q/x\}) = \mathrm{bound}(P_1)$. Hence, $(\mu y.P_1)\{Q/x\} \in \tilde{\mathbb{P}}$ is a time-guarded $\tilde{\mathbb{P}}$-term and $\mathrm{bound}(\mu y.(P_1\{Q/x\})) = \mathrm{bound}(P_1\{Q/x\}) = \mathrm{bound}(P_1) = \mathrm{bound}(\mu y.P_1)$.

□

Proposition 3.5 Let P be a time-guarded $\tilde{\mathbb{P}}_c$-term. If $\mathcal{R}(\alpha, P) < 1$ then there exists a time-guarded $Q \in \tilde{\mathbb{P}}_c$ such that $P \xrightarrow{\alpha} Q$ and $\mathrm{bound}(Q) < \mathrm{bound}(P)$.

Proof: By induction on the structure of P.

Nil: $P \equiv \mathbf{0}$. This case is not possible because $\mathcal{R}(\alpha, P) = 1$.

Stop,Var: These cases are not possible because P is time-guarded.

Pref: $P \equiv \langle \beta, r \rangle.P_1$ and $P_1 \in \tilde{\mathbb{P}}$. $\mathcal{R}(\alpha, P) < 1$ implies $\alpha = \beta$ and $r < 1$. Since P is time-guarded, P_1 is also time-guarded. Moreover, $P \xrightarrow{\alpha} P_1$ and $\text{bound}(P) = 1 + \text{bound}(P_1) > \text{bound}(P_1)$.

Sum: $P \equiv P_1 + P_2$. Then both P_1 and P_2 are time-guarded $\tilde{\mathbb{P}}_c$-terms. Assume $\mathcal{R}(\alpha, P) = \min\{\mathcal{R}(\alpha, P_1), \mathcal{R}(\alpha, P_2)\} < 1$. Then $\mathcal{R}(\alpha, P_1) < 1$ or $\mathcal{R}(\alpha, P_2) < 1$. If $\mathcal{R}(\alpha, P_1) < 1$ (symmetrically if $\mathcal{R}(\alpha, P_2) < 1$), then by induction hypothesis there is $Q \in \tilde{\mathbb{P}}_c$ such that $P_1 \xrightarrow{\alpha} Q$ and $\text{bound}(P_1) > \text{bound}(Q)$. Then $P_1 + P_2 \xrightarrow{\alpha} Q$ and $\text{bound}(P_1 + P_2) = \max\{\text{bound}(P_1), \text{bound}(P_2)\} > \text{bound}(Q)$.

Par: $P \equiv P_1 \|_A P_2$. Then both P_1 and P_2 are $\tilde{\mathbb{P}}_c$-terms. Assume $\mathcal{R}(\alpha, P) < 1$. We consider two cases: $\alpha \in A$ and $\alpha \notin A$.

- $\alpha \in A$. Then $\mathcal{R}(\alpha, P) = \max\{\mathcal{R}(\alpha, P_1), \mathcal{R}(\alpha, P_2)\} < 1$. Hence $\mathcal{R}(\alpha, P_1) < 1$ and $\mathcal{R}(\alpha, P_2) < 1$. By induction hypothesis there are time-guarded Q_1 and Q_2 in $\tilde{\mathbb{P}}_c$ such that $P_1 \xrightarrow{\alpha} Q_1$, $P_2 \xrightarrow{\alpha} Q_2$, $\text{bound}(P_1) > \text{bound}(Q_1)$ and $\text{bound}(P_2) > \text{bound}(Q_2)$. Then $Q_1 \|_A Q_2$ is a time-guarded $\tilde{\mathbb{P}}_c$-term and $P_1 \|_A P_2 \xrightarrow{\alpha} Q_1 \|_A Q_2$. Moreover, $\text{bound}(P_1 \|_A P_2) = \text{bound}(P_1) + \text{bound}(P_2) > \text{bound}(Q_1) + \text{bound}(Q_2) = \text{bound}(Q_1 \|_A Q_2)$.
- $\alpha \notin A$. $\mathcal{R}(\alpha, P) = \min\{\mathcal{R}(\alpha, P_1), \mathcal{R}(\alpha, P_2)\} < 1$. Then $\mathcal{R}(\alpha, P_1) < 1$ or $\mathcal{R}(\alpha, P_2) < 1$. If $\mathcal{R}(\alpha, P_1) < 1$ (symmetrically if $\mathcal{R}(\alpha, P_2) < 1$), then by induction there is $Q \in \tilde{\mathbb{P}}_c$ such that $P_1 \xrightarrow{\alpha} Q$ and $\text{bound}(P_1) > \text{bound}(Q)$. Hence, $Q \|_A P_2$ is a time-guarded $\tilde{\mathbb{P}}_c$-term and $P_1 \|_A P_2 \xrightarrow{\alpha} Q \|_A P_2$. Moreover $\text{bound}(P_1 \|_A P_2) = \text{bound}(P_1) + \text{bound}(P_2) > \text{bound}(Q) + \text{bound}(P_2) = \text{bound}(Q \|_A P_2)$.

Rel: $P \equiv P_1[\Phi]$. $\mathcal{R}(\alpha, P) < 1$ means there exists $\beta \in Act$ such that $\Phi(\beta) = \alpha$ and $\mathcal{R}(\beta, P_1) < 1$. By induction hypothesis there exists a time-guarded $Q \in \tilde{\mathbb{P}}_c$ such that $P_1 \xrightarrow{\beta} Q$ and $\text{bound}(P_1) > \text{bound}(Q)$. Then $P_1[\Phi] \xrightarrow{\alpha} Q[\Phi]$ and $\text{bound}(P_1[\Phi]) > \text{bound}(Q[\Phi])$.

Rec: $P \equiv \mu x.P_1$. $\mathcal{R}(\alpha, P) < 1$ means $\mathcal{R}(\alpha, P_1) < 1$. Of course P_1 is time-guarded. Thus, by induction hypothesis, there exists a time-guarded $Q \in \tilde{\mathbb{P}}_c$ such that $P_1 \xrightarrow{\alpha} Q$ and $\text{bound}(P_1) > \text{bound}(Q)$. By Proposition 3.4 $Q\{\mu x.P_1/x\}$ is a time-guarded $\tilde{\mathbb{P}}_c$-term and $\mu x.P_1 \xrightarrow{\alpha} Q\{\mu x.P_1/x\}$ and $\text{bound}(\mu x.P_1) = \text{bound}(P_1) > \text{bound}(Q) = \text{bound}(Q\{\mu x.P_1/x\})$ by Proposition 3.4.

□

Corollary 3.6 Let P be a time-guarded $\mathbb{P}_c$-term. If $\mathcal{R}(\alpha, P) < 1$ then there exists a time-guarded $Q \in \mathbb{P}_c$ such that $P \xrightarrow{\alpha} Q$ and $\text{bound}(Q) < \text{bound}(P)$.

Proof: By 2.6.5, the Q from 3.5 is in $\mathbb{P}_c$. □

Lemma 3.7 Let P be a time-guarded $\tilde{\mathbb{P}}_c$-($\mathbb{P}_c$-)term such that $\mathcal{R}(P) < 1$. Then there exists a time-guarded $\tilde{\mathbb{P}}_c$-($\mathbb{P}_c$-)term Q and a sequence $w \in (\mathbb{A}_\tau)^*$ such that $P \xrightarrow{w}_c Q$ and $\mathcal{R}(Q) = 1$.

Proof: Assume P to be a time-guarded $\tilde{\mathbb{P}}_c$-($\mathbb{P}_c$-)term. By Proposition 3.5 (Corollary 3.6), there exists a computation $P \stackrel{\alpha_1}{\rightarrow}_c Q_1 \stackrel{\alpha_2}{\rightarrow}_c ... \stackrel{\alpha_i}{\rightarrow}_c Q_i \stackrel{\alpha_{i+1}}{\rightarrow}_c ...$ such that $\mathcal{R}(Q_i) < 1$ and $\text{bound}(P) > \text{bound}(Q_1) > ... > \text{bound}(Q_{i+1}) > ...$ By Lemma 3.3 such a computation is finite, ending at some $Q \equiv Q_n$ with $\mathcal{R}(Q) = 1$; hence, we choose $w = \alpha_1\alpha_2...\alpha_n$. □

Proposition 3.8 Let P be an Ω-free $\mathbb{P}_c$-process. Then there exists $w \in (\mathbb{A}_\tau)^*$ such that $P \stackrel{w1}{\rightarrow}_c Q$ and Q is an Ω-free $\mathbb{P}$-term.

Proof: If $\mathcal{R}(P) = 1$ then $P \stackrel{1}{\rightarrow}_c Q$. Q is, of course, Ω-free and by Lemma 2.6.6, in $\mathbb{P}$. Otherwise, $\mathcal{R}(P) < 1$ and, by Lemma 3.7, there exists $w \in (\mathbb{A}_\tau)^*$ and $R \in \mathbb{P}_c$ such that $P \stackrel{w}{\rightarrow}_c R$ and $\mathcal{R}(R) = 1$. Thus, $R \stackrel{1}{\rightarrow}_c Q$ and by Lemma 2.6.6, $Q \in \mathbb{P}$. Again Q is Ω-free. □

Proposition 3.9 No Ω-free $\mathbb{P}_c$-process P is a time-stop.

Proof: By Proposition 3.8, there exists $w \in (\mathbb{A}_\tau)^*$ such that $P \stackrel{w1}{\rightarrow}_c Q$ and Q is an Ω-free $\mathbb{P}$-process. Iterative applications of Proposition 3.8 prove that P cannot be a time-stop. □

Corollary 3.10 No Ω-free $\mathbb{P}_c$-process P has a time-stop.

Proof: Any derivative of an Ω-free $\mathbb{P}_c$-process is an Ω-free $\mathbb{P}_c$-process and, by Proposition 3.9, cannot be a time-stop. □

As discussed at the end of the last section, this result is important for our testing scenario to make sense intuitively. In our definition of test satisfaction, it might seem that we completely ignore runs with duration less than the test duration R. Now we know that this is not the case: such runs can be extended to runs with duration $> R$, since Ω-free processes like $\tau.P \| O$ do not have time-stops. Compare this e.g. to approaches that only consider infinite runs and therefore ignore runs that lead to deadlocks; such approaches can lead to counter-intuitive results. (It is not clear how such finite runs can be avoided operationally.)

4 Discretization

Intuitively, satisfaction of a timed test essentially depends on the 'slowest' sequences in $\mathsf{CL}(\tau.P \| O)$; in this section, we will show that these are already captured if we consider only discrete behaviour, i.e. traces where all time steps have duration 1. This will yield a simple theory.

Definition 4.1 (*Discrete language of processes*) Let $P, P' \in \tilde{\mathbb{L}}$ be general process terms. We write $P \stackrel{\varepsilon}{\rightarrow}_d P'$ if either $\varepsilon \in \mathbb{A}_\tau$ and $P \stackrel{\varepsilon}{\rightarrow} P'$, or $\varepsilon = 1$ and $P \stackrel{\varepsilon}{\rightarrow}_c P'$; in the latter case we say that P performs a *unit time step*. For sequences $w \in (\mathbb{A}_\tau \cup \{1\})^*$, we define $\stackrel{w}{\rightarrow}_d$ and $\stackrel{w/\tau}{\Rightarrow}_d$ analogously to Definition 2.13.

For a general process $P \in \mathbb{L}$ we define $\mathsf{DL}_\tau(P) = \{w \mid P \stackrel{w}{\rightarrow}_d\}$ to be the *discrete(ly timed) τ-language*, containing the *discrete τ-traces* of P, and $\mathsf{DL}(P) = \{w/\tau \mid w \in \mathsf{DL}_\tau(P)\}$ to be the *discrete language*, containing the *discrete traces* of P.

Observe that by definition $\mathsf{DL}(P) \subseteq \mathsf{CL}(P)$ and $\mathsf{DL}_\tau(P) \subseteq \mathsf{CL}_\tau(P)$, and that $\mathsf{DL}_\tau(P) = \mathsf{DL}(P) = \{\lambda\}$ for P not guarded. Furthermore, the set of Ω-free $\mathbb{P}$- and $\tilde{\mathbb{P}}$-terms and the set of $\tilde{\mathbb{P}}$-terms are closed under $\stackrel{w}{\rightarrow}_d$, i.e. $P \stackrel{w}{\rightarrow}_d P'$ for some $P \in \mathbb{P}$ implies $P' \in \mathbb{P}$ etc. (These results follow or are easy adaptations of the closure properties in Lemma 2.6.)

The following proposition states some useful properties of discrete processes.

Proposition 4.2 Let $P \in \tilde{\mathbb{P}}$. Then:

1. $\mathcal{R}(\alpha, P) \in \{0, 1\}$, thus $\mathcal{R}(P) \in \{0, 1\}$ for every $P \in \tilde{\mathbb{P}}$.
2. Let P be a time-guarded $\tilde{\mathbb{P}}$-term. Then there exists a time-guarded $P' \in \tilde{\mathbb{P}}$ and $w \in \{\alpha\}^*$, such that $P \xrightarrow{w}_d P'$ and $\mathcal{R}(\alpha, P') = 1$.

Proof:
1. from the definition of $\tilde{\mathbb{P}}$ and $\mathcal{R}$ with structural induction
2. The proof is similar to that of Lemma 3.7. By Proposition 3.5 there exists a finite maximal sequence $P \xrightarrow{\alpha}_c Q_1 \xrightarrow{\alpha}_c ... \xrightarrow{\alpha}_c P'$ such that $\mathcal{R}(\alpha, Q_i) < 1$, $\text{bound}(P) > \text{bound}(Q_1) > ...$ and $\mathcal{R}(\alpha, P') = 1$. □

So far, we only know that discrete behaviour of a discrete process is part of its continuous behaviour, viz $\mathsf{DL}(P) \subseteq \mathsf{CL}(P)$. We now aim to show that discrete behaviour already contains enough information for checking $P \sqsupseteq_c Q$ for testable P and Q. For this purpose, we will map each continuous trace of a discrete process to a discrete trace of the same process. Related traces will exhibit the same behaviour, but at different points in time. We first relate the intermediate general processes reached when performing such traces.

Definition 4.3 (*Progress preorder of general process terms*)
The *progress preorder* $\succ_\delta$ for $\delta \in \mathbb{T}$ is a relation on $\tilde{\mathbb{L}}$ satisfying:

1.	Nil, Stop, Var:	$\mathbf{0} \succ_\delta \mathbf{0}$, $\Omega \succ_\delta \Omega$ and $x \succ_\delta x$
2.	Pref:	$\langle \alpha, r_1 \rangle.P \succ_\delta \langle \alpha, r_2 \rangle.P$ if $r_2 - r_1 \leq \delta$
3.	Sum:	$P_1 + P_2 \succ_\delta Q_1 + Q_2$ if $\forall_{i=1,2}\ P_i \succ_\delta Q_i$
4.	Par:	$P_1 \|\|_A P_2 \succ_\delta Q_1 \|\|_A Q_2$ if $\forall_{i=1,2}\ P_i \succ_\delta Q_i$
5.	Rel:	$P[\Phi] \succ_\delta Q[\Phi]$ if $P \succ_\delta Q$
6.	Rec:	a) $\mu x.P \succ_\delta \mu x.P$
		b) $P'\{\mu x.P/x\} \succ_\delta \mu x.P$ if $P' \succ_\delta P$
		c) $\mu x.P \succ_\delta P'\{\mu x.P/x\}$ if $P \succ_\delta P'$

Intuitively, $P \succ_\delta Q$ means that P and Q are essentially identical up to the values of timers, and if P is ahead of Q, then for at most time δ. However, Q may be ahead of P for an arbitrary amount of time; compare the Pref-case, where we allow $r_2 < r_1$.

Cases Rec b) and c) say that $\mu x.P$ and $P'\{\mu x.P/x\}$ can be identified in two specific situations; this is necessary to make Proposition 4.4.8.a) and b) below true: if $P \equiv \mu x.R \succ_\delta \mu x.R \equiv Q$ and P makes a time step, then only for P recursion is unfolded by rule Rec_c. An alternative would have been permitting 0 time steps in the discrete behaviour, which we have reprobated for the sake of compactness, since they only alter recursive terms and expand discrete traces unnecessarily.

Proposition 4.4 For general process terms $P, Q, R \in \tilde{\mathbb{L}}$ and $\delta, \delta' \in \mathbb{T}$ let $P \succ_\delta Q$. Furthermore, let $\alpha \in \mathbb{A}_\tau$, $x \in \mathcal{X}$ and $\rho, \rho_1, \rho_2 \in \mathbb{T}$.

1. $P \succ_0 P$ for all $P \in \tilde{\mathbb{L}}$.
2. x is guarded in P iff x is guarded in Q.
3. $\mathcal{A}(P) = \mathcal{A}(Q)$.

4. $\mathcal{R}(\alpha, Q) - \mathcal{R}(\alpha, P) \leq \delta$, in particular $\mathcal{R}(Q) - \mathcal{R}(P) \leq \delta$.

5. If $\delta \leq \delta'$, then $P \succ_{\delta'} Q$; for all δ', $P \succ_{\delta'} P$.

6. $P\{R/x\} \succ_\delta Q\{R/x\}$.

7. If $P \xrightarrow{\alpha} P'$, then there exists Q' such that $Q \xrightarrow{\alpha} Q'$ and $P' \succ_\delta Q'$, and vice versa.

8. a) If $P \xrightarrow{\rho}_c P'$ and $\delta + \rho \leq 1$, then $P' \succ_{\delta+\rho} Q$.

   b) If $Q \xrightarrow{\rho}_c Q'$ and $0 \leq \delta - \rho$, then $P \succ_{\delta-\rho} Q'$.

   c) If $P \xrightarrow{\rho_1}_c P'$, $Q \xrightarrow{\rho_2}_c Q'$ and $0 \leq \delta + \rho_1 - \rho_2 \leq 1$, then $P' \succ_{\delta+\rho_1-\rho_2} Q'$.

9. a), b) and c) of 8. hold with $\rightarrow_c$ replaced by $\rightsquigarrow_c$.

10. $P \overset{1}{\rightsquigarrow}_c P'$ and $Q \overset{1}{\rightsquigarrow}_c Q'$ imply $P' \equiv Q'$.

Proof: 1. Induction on the structure of P.

2. Induction on the inference of $P \succ_\delta Q$; only the Rec-cases are non-trivial:

a) clear

b) x guarded in $\mu y.P$ iff x guarded in P iff (ind.) x guarded in P' iff x guarded in $P'\{\mu y.P/y\}$. (For the last 'iff', observe for '$\Rightarrow$' that y is guarded in P, hence in P' by ind., so all occurrences of x in $\mu y.P$ are guarded in $P'\{\mu y.P/y\}$. Observe for '$\Leftarrow$' and $x \equiv y$ that again y is guarded in P, hence in P' by ind.)

c) analogous

3. Induction on the inference of $P \succ_\delta Q$; in the Rec-cases observe that x is guarded in P and apply 2. and Proposition 2.4.2.

4. For the following, compare the properties listed at the beginning of the proof of 2.10. We perform induction on the inference of $P \succ_\delta Q$, showing only some cases:

Sum: $P_1 + P_2 \succ_\delta Q_1 + Q_2 \Rightarrow \forall_{i=1,2}\ P_i \succ_\delta Q_i \Rightarrow \forall_{\alpha\in\mathbb{A}_\tau} \forall_{i=1,2}\ \mathcal{R}(\alpha, Q_i) - \mathcal{R}(\alpha, P_i) \leq \delta \Rightarrow \forall_{\alpha\in\mathbb{A}_\tau}\ \mathcal{R}(\alpha, Q_1 + Q_2) - \mathcal{R}(\alpha, P_1 + P_2) = \min_{i=1,2} \mathcal{R}(\alpha, Q_i) - \min_{i=1,2} \mathcal{R}(\alpha, P_i) \leq \delta$.

Par: $P_1 \| _A P_2 \succ_\delta Q_1 \| _A Q_2 \Rightarrow \forall_{i=1,2}\ P_i \succ_\delta Q_i \Rightarrow \forall_{\alpha\in\mathbb{A}_\tau} \forall_{i=1,2}\ \mathcal{R}(\alpha, Q_i) - \mathcal{R}(\alpha, P_i) \leq \delta \Rightarrow \forall_{\alpha\in\mathbb{A}_\tau} \max_{i=1,2} \mathcal{R}(\alpha, Q_i) - \max_{i=1,2} \mathcal{R}(\alpha, P_i) \leq \delta \wedge \min_{i=1,2} \mathcal{R}(\alpha, Q_i) - \min_{i=1,2} \mathcal{R}(\alpha, P_i) \leq \delta \Rightarrow \forall_{\alpha\in\mathbb{A}_\tau} \mathcal{R}(\alpha, Q_1 \| _A Q_2) - \mathcal{R}(\alpha, P_1 \| _A P_2) \leq \delta$.

Rel: $P[\Phi] \succ_\delta Q[\Phi] \Rightarrow P \succ_\delta Q \Rightarrow \forall_{\alpha\in\mathbb{A}_\tau}\ \mathcal{R}(\alpha, Q) - \mathcal{R}(\alpha, P) \leq \delta$
$\Rightarrow \forall_{\alpha\in\mathbb{A}_\tau}\ \min_{\beta\in\Phi^{-1}(\alpha)} \mathcal{R}(\beta, Q) - \min_{\beta\in\Phi^{-1}(\alpha)} \mathcal{R}(\beta, P) \leq \delta$
$\Rightarrow \forall_{\alpha\in\mathbb{A}_\tau}\ \mathcal{R}(\alpha, Q[\Phi]) - \mathcal{R}(\alpha, P[\Phi]) \leq \delta$.

Rec: a) $\mathcal{R}(\mu x.P) - \mathcal{R}(\mu x.P) = 0 \leq \delta$.

b) Since x is guarded in P, it is guarded in P' by 2., hence $\mathcal{R}(\alpha, P'\{\mu x.P/x\}) = \mathcal{R}(\alpha, P')$ by Proposition 2.9.2. Now $\mathcal{R}(\alpha, \mu x.P) - \mathcal{R}(\alpha, P'\{\mu x.P/x\}) = \mathcal{R}(\alpha, P) - \mathcal{R}(\alpha, P') \leq \delta$ by ind.

c) analogous

For the additional property we can either choose $\alpha \in \mathcal{A}(P) = \mathcal{A}(Q)$ with $\mathcal{R}(P) = \mathcal{R}(\alpha, P)$ and get $\mathcal{R}(Q) - \mathcal{R}(P) \leq \mathcal{R}(\alpha, Q) - \mathcal{R}(\alpha, P) \leq \delta$, or we have $\mathcal{A}(P) = \mathcal{A}(Q) = \emptyset$ and by Proposition 2.9.1: $\mathcal{R}(Q) - \mathcal{R}(P) = 1 - 1 \leq \delta$.

5. Induction on the inference of $P \succ_\delta Q$; in particular, $r_2 - r_1 \leq \delta \leq \delta'$ in Definition 4.3.2. Then apply 1. for the second part.

6. Induction on the inference of $P \succ_\delta Q$; the case $P \equiv Q$ is covered by 5., and covers Nil, Stop, Var and Rec a).

Pref: depends on r_1, r_2 and δ only

Sum, Par, Rel: straightforward induction by 'distributivity' of substitution.

Rec: b) y is not free in $P'\{\mu y.P/y\}$ and $\mu y.P$, hence assume $x \not\equiv y$; then by Barendregt convention $P'\{\mu y.P/y\}\{R/x\} \equiv P'\{R/x\}\{\mu y.P\{R/x\}/y\}$. Now $P' \succ_\delta P$ implies $P'\{R/x\} \succ_\delta P\{R/x\}$ by ind., hence $(P'\{R/x\})\{\mu y.P\{R/x\}/y\} \succ_\delta \mu y.(P\{R/x\}) \equiv (\mu y.P)\{R/x\}$.

c) analogous

7. Induction on the inference of $P \succ_\delta Q$:

Nil,Stop,Var: No α-transition is possible

Pref: since $P \succ_\delta P$ by 5.

Sum,Par, Rel: straightforward induction.

Rec: a) clear.

b) x is guarded in P and also in P' by 2.

On the one hand, $P'\{\mu x.P/x\} \xrightarrow{\alpha} R$ only if (by Proposition 2.4.2)
$\exists P'' : P' \xrightarrow{\alpha} P'' \wedge R \equiv P''\{\mu x.P/x\}$ only if (by ind.)
$\exists P'', P''' : P \xrightarrow{\alpha} P''' \wedge P'' \succ_\delta P''' \wedge R \equiv P''\{\mu x.P/x\}$
only if (by Rule Rec_a and 6.)
$\exists P'', P''' : \mu x.P \xrightarrow{\alpha} P'''\{\mu x.P/x\} \wedge R \equiv P''\{\mu x.P/x\} \succ_\delta P'''\{\mu x.P/x\}$.

On the other hand, $\mu x.P \xrightarrow{\alpha} R$ only if (by rule Rec_a)
$\exists P''' : P \xrightarrow{\alpha} P''' \wedge R \equiv P'''\{\mu x.P/x\}$ only if (by ind.)
$\exists P'', P''' : P' \xrightarrow{\alpha} P'' \wedge P'' \succ_\delta P''' \wedge R \equiv P'''\{\mu x.P/x\}$
only if (by Proposition 2.4.2 and 6.)
$\exists P'', P''' : P'\{\mu x.P/x\} \xrightarrow{\alpha} P''\{\mu x.P/x\} \succ_\delta P'''\{\mu x.P/x\} \equiv R$.

c) analogous

8/9.a) By Definition 2.11, it suffices to show $P \overset{\rho}{\rightsquigarrow}_c P' \wedge \delta + \rho \leq 1 \Rightarrow P' \succ_{\delta+\rho} Q$ by induction on the overall size of P *and* Q (where the size is the number of operators, also counting μx):
Clear for $\mathbf{0}$, Ω and x.

Pref: $\langle \alpha, r_1 \rangle.P \succ_\delta \langle \alpha, r_2 \rangle.P \Rightarrow r_2 - r_1 \leq \delta$. We distinguish two cases:

i) $\rho \leq r_1$:
then $\langle \alpha, r_1 \rangle.P \overset{\rho}{\rightsquigarrow}_c \langle \alpha, r_1 - \rho \rangle.P$ and
$r_2 - r_1 \leq \delta \Rightarrow r_2 - (r_1 - \rho) = r_2 - r_1 + \rho \leq \delta + \rho \Rightarrow \langle \alpha, r_1 - \rho \rangle.P \succ_{\delta+\rho} \langle \alpha, r_2 \rangle.P$.

ii) $\rho > r_1$:
then $\langle \alpha, r_1 \rangle.P \overset{\rho}{\rightsquigarrow}_c \langle \alpha, 0 \rangle.P$ and
$r_2 - r_1 \leq \delta \Rightarrow r_2 \leq \delta + r_1 < \delta + \rho \Rightarrow r_2 - 0 \leq \delta + \rho \Rightarrow \langle \alpha, 0 \rangle.P \succ_{\delta+\rho} \langle \alpha, r_2 \rangle.P$.

Sum,Par,Rel: straightforward induction.

Rec: a) $\mu x.P \overset{\rho}{\rightsquigarrow}_c P''\{\mu x.P/x\}$ and $P \overset{\rho}{\rightsquigarrow}_c P''$ for some P'' by rule Rec_c. Hence $P \succ_\delta P$ implies $P'' \succ_{\rho+\delta} P$ by ind. and thus $P''\{\mu x.P/x\} \succ_{\rho+\delta} \mu x.P$ by Definition 4.3.6.b).

b) $P'\{\mu x.P/x\} \overset{\rho}{\rightsquigarrow}_c R$ implies by 2. and Lemma 2.7.3 that $\exists P'' : P' \overset{\rho}{\rightsquigarrow}_c P''$ and $R \equiv P''\{\mu x.P/x\}$. Hence by ind. $P'' \succ_{\delta+\rho} P$, such that $R \succ_{\delta+\rho} \mu x.P$ by Definition 4.3.6.b). Note that P' has at most the size of $P'\{\mu x.P/x\}$, and the sizes might be equal if P' does not contain a free x; but in any case, P has a smaller size than $\mu x.P$ and, thus, induction is applicable.

c) Similar to a) with induction and 6.

8/9.b) Similar to 8/9.a). We only consider the Pref-case:
$\langle \alpha, r_1 \rangle.P \succ_\delta \langle \alpha, r_2 \rangle.P \Rightarrow r_2 - r_1 \leq \delta$. We distinguish two cases:

i) $\rho \leq r_2$:
then $\langle \alpha, r_2 \rangle.P \overset{\rho}{\rightsquigarrow}_c \langle \alpha, r_2 - \rho \rangle.P$ and
$r_2 - r_1 \leq \delta \Rightarrow (r_2 - \rho) - r_1 = r_2 - r_1 - \rho \leq \delta - \rho \Rightarrow \langle \alpha, r_1 \rangle.P \succ_{\delta-\rho} \langle \alpha, r_2 - \rho \rangle.P$.

ii) $\rho > r_2$:
then $\langle \alpha, r_2 \rangle.P \overset{\rho}{\rightsquigarrow}_c \langle \alpha, 0 \rangle.P$ and
$0 - r_1 \leq 0 \leq \delta - \rho \Rightarrow \langle \alpha, r_1 \rangle.P \succ_{\delta-\rho} \langle \alpha, 0 \rangle.P$.

8/9.c) If $\delta + \rho_1 \leq 1$, we get $P' \succ_{\delta+\rho_1} Q$ by 8/9.a) and $P' \succ_{\delta+\rho_1-\rho_2} Q'$ by $0 \leq \delta + \rho_1 - \rho_2$ and 8/9.b). Otherwise, $P \overset{1-\delta}{\rightsquigarrow}_c P'' \overset{\rho}{\rightsquigarrow}_c P'$ with $\rho = \rho_1 + \delta - 1$ by Lemma 2.7.5. Now $P'' \succ_1 Q$ by 8/9.a), $P'' \succ_{1-\rho_2} Q'$ by 8/9.b) and $P' \succ_{\delta+\rho_1-\rho_2} Q'$ by 8/9.a) again.

10. Induction on the inference of $P \succ_\delta Q$; for Rec b) and c) use 2. and 2.7.3. □

Proposition 4.4 provides the elements for emulating each continuous trace of an initial process by a discrete trace that exhibits the same behaviour but consumes more time:

Lemma 4.5 Let $P \in \mathbb{P}$ be a discrete process; then for each $w \in \mathsf{CL}(P)$ there is a $v \in \mathsf{DL}(P)$, such that $act(v) = act(w)$ and $\zeta(v) \geq \zeta(w)$.

Proof: We will construct for each $w \in \mathsf{CL}_\tau(P)$ a $v \in \mathsf{DL}_\tau(P)$, such that $act(v) = act(w)$ and $\zeta(v) \geq \zeta(w)$; furthermore, we will show that for $P_w \in \mathbb{P}$ and $P_v \in \mathbb{P}$ reached after w and v we have $P_v \succ_{\zeta(v)-\zeta(w)} P_w$. Then $w/\tau \in \mathsf{CL}(P)$, $v/\tau \in \mathsf{DL}(P)$, $act(v/\tau) = act(w/\tau)$ and $\zeta(v/\tau) = \zeta(v) \geq \zeta(w) = \zeta(w/\tau)$.

The proof is by induction on $|w|$, where for $w = \lambda$ we can choose $v = \lambda$; then $P \succ_0 P$ by Proposition 4.4.1, hence $P_v \succ_{0-0} P_w$.

Hence, assume that for $w \in \mathsf{CL}_\tau(P)$ we have constructed $v \in \mathsf{DL}_\tau(P)$ as desired and consider $w' = w\varepsilon \in \mathsf{CL}_\tau(P)$. We denote the processes reached after w' and the corresponding v' by $P_{w'}$ and $P_{v'}$.

If $\varepsilon = \alpha \in \mathbb{A}$, then $v' = v\alpha$ with $act(v') = act(w')$ and $\zeta(v') = \zeta(v) \geq \zeta(w) = \zeta(w')$. We have $P_w \xrightarrow{\alpha} P_{w'}$ and by Proposition 4.4.7, there is a $P_{v'}$ such that $P_v \xrightarrow{\alpha} P_{v'}$ and $P_{v'} \succ_{\zeta(v)-\zeta(w)} P_{w'}$, i.e. $P_{v'} \succ_{\zeta(v')-\zeta(w')} P_{w'}$.

Now let $\varepsilon = \rho \in \mathbb{T}$. If $\rho \leq \zeta(v) - \zeta(w)$ we choose $v' = v$; obviously, $act(v') = act(w')$, $\zeta(v') = \zeta(v) \geq \rho + \zeta(w) = \zeta(w')$. Furthermore, $\zeta(v') - \zeta(w') = \zeta(v) - \zeta(w) - \rho \geq 0$, hence $P_v \succ_{\zeta(v)-\zeta(w)} P_w$ and Proposition 4.4.8.b) yield $P_{v'} \equiv P_v \succ_{\zeta(v')-\zeta(w')} P_{w'}$.

If on the other hand $\rho > \zeta(v) - \zeta(w)$, we choose $v' = v1$. With Proposition 4.4.4, from $P_v \succ_{\zeta(v)-\zeta(w)} P_w$ we conclude $\mathcal{R}(P_v) + \zeta(v) - \zeta(w) \geq \mathcal{R}(P_w)$ and $\mathcal{R}(P_w) \geq \rho > \zeta(v) - \zeta(w)$ by Definition 2.11, i.e. $\mathcal{R}(P_v) > 0$ and $\mathcal{R}(P_v) = 1$ by Proposition 4.2.1. Now by Proposition 4.4.2, P_v is guarded iff P_w is guarded, and P_w is guarded by Definition 2.11 and Lemma 2.7.1; hence by Proposition 2.12.1, the time step 1 is allowed after v and $v' = v1 \in \mathsf{DL}_\tau(P)$ with $act(v') = act(w')$. Furthermore, $\zeta(v') = \zeta(v) + 1 \geq \zeta(w) + \rho = \zeta(w')$, and finally, $\rho \leq 1$ and $0 \leq \zeta(v) - \zeta(w)$ give $0 \leq \zeta(v) - \zeta(w) + 1 - \rho$, and $\zeta(v) - \zeta(w) < \rho$ gives $\zeta(v) - \zeta(w) + 1 - \rho \leq 1$; so with Proposition 4.4.8.c) we conclude $P_{v'} \succ_{\zeta(v)-\zeta(w)+1-\rho} P_{w'}$, i.e. $P_{v'} \succ_{\zeta(v')-\zeta(w')} P_{w'}$. □

With this emulation result we can restrict attention to discretely timed testing based on discrete behaviour and discrete time bounds:

Definition 4.6 (*Discretely timed tests*) For a testable process $P \in \mathbb{P}$, an observer $O \in \mathbb{P}_1$ and $D \in \mathbb{N}_0$ define P *must*$_d$ (O, D), if each $w \in \mathsf{DL}(\tau.P \| O)$, with $\zeta(w) > D$ contains some ω. The relation $\sqsupseteq_d$ is defined accordingly.

Again, one could wonder whether this definition might ignore some relevant behaviour due to discrete time-stops, where we could define being or having a discrete time-stop, replacing $\xrightarrow{w}_c$ by $\xrightarrow{w}_d$ in Definition 3.1. Since $\xrightarrow{w}_d$ is a subset of $\xrightarrow{w}_c$, Proposition 3.9 and Corollary 3.10 imply immediately that Ω-free $\mathbb{P}$-processes like $\tau.P \| O$ neither are nor have such discrete time-stops.

We now give our first main result: although $\sqsupseteq_d$ is based on fewer tests and much more restricted behaviour than $\sqsupseteq_c$, it turns out that both relations define the same efficiency preorder. By this, we have also reached simplicity: we can now work with a *CCS*-like untimed algebra, extended syntactically by urgent actions and semantically by 1-time-steps.

Theorem 4.7 *The relations $\sqsupseteq_c$ and $\sqsupseteq_d$ coincide.*

Proof: Let P and Q be testable processes, O an observer and $R \in \mathbb{R}_0^+$. We first show

$$P \; must_c \; (O, R) \;\Leftrightarrow\; P \; must_d \; (O, \lfloor R \rfloor)$$

Assume $P \not\; must_c \; (O, R)$; then there is a $w \in \mathsf{CL}(\tau.P \| O)$ without ω and $\zeta(w) > R$; now by Lemma 4.5, there is a $v \in \mathsf{DL}(\tau.P \| O)$ without ω and $\zeta(v) \geq \zeta(w) > \lfloor R \rfloor$, hence $P \not\; must_d$ $(O, \lfloor R \rfloor)$. Now assume $P \not\; must_d \; (O, \lfloor R \rfloor)$; then there is a $w \in \mathsf{DL}(\tau.P \| O)$ without ω and $\zeta(w) > \lfloor R \rfloor$, hence $\zeta(w) \geq \lfloor R \rfloor + 1 > R$; since $\mathsf{DL}(\tau.P \| O) \subseteq \mathsf{CL}(\tau.P \| O)$, the same w causes $P \not\; must_c \; (O, R)$.

With this result we conclude $\forall\, (O, R) : Q \; must_c \; (O, R) \Rightarrow P \; must_c \; (O, R)$ iff $\forall\, (O, R) : Q \; must_d \; (O, \lfloor R \rfloor) \Rightarrow P \; must_d \; (O, \lfloor R \rfloor)$, hence $P \sqsupseteq_c Q$ iff $P \sqsupseteq_d Q$. □

Checking $P \sqsupseteq_c Q$ now reduces to checking $P \sqsupseteq_d Q$. But as for testing in general, using the definition of $\sqsupseteq_d$ directly is hardly feasible, since there are still infinitely many discretely timed tests to apply. And as indicated in Section 2, we cannot decide $P \sqsupseteq_d Q$ from $\mathsf{DL}(P)$

and $\mathsf{DL}(Q)$ only, since $\mathsf{DL}(\tau.P\|O)$ generally cannot be determined from $\mathsf{DL}(P)$ and $\mathsf{DL}(O)$ alone: e.g. synchronization allows activated actions in one component to wait for a partner in the other one, which is not the case in stand-alone behaviour of a single component, recorded in $\mathsf{DL}(P)$, $\mathsf{DL}(O)$ resp.

Technically, DL-inclusion is not a precongruence for parallel composition. As an example consider $P = (a+b)\|_{\{b\}}c.b$ and $Q = (a+b)\|c$, where $\mathsf{DL}(P) \subseteq \mathsf{DL}(Q)$ but $1c1 \in \mathsf{DL}(P\|_{\{a\}}\mathbf{0})$ and $1c1 \notin \mathsf{DL}(Q\|_{\{a\}}\mathbf{0})$. Thus, in the next section we will refine the discrete language to a kind of refusal traces, fulfilling the precongruence criterion. Refusal traces of a testable process will allow us to characterize the preorder $\sqsupseteq_d$ denotationally. There, we will also need the following result, stating that the number of different actions ever performable by a process is finite.

Definition 4.8 (*Semantic sort of a process*) For a general process term $P \in \tilde{\mathbb{L}}$ let $\ell_c(P) = \{a \in \mathbb{A} \,|\, \exists w \in \mathsf{CL}_\tau(P),\, P' \in \mathbb{L} : P \xrightarrow{w}_c P' \xrightarrow{a}_c\}$ be the *continuous semantic sort* of P, and $\ell_d(P) = \{a \in \mathbb{A} \,|\, \exists w \in \mathsf{DL}_\tau(P),\, P' \in \mathbb{L} : P \xrightarrow{w}_d P' \xrightarrow{a}_d\}$ be the *discrete semantic sort* of P.

For a general relabelling function Φ let $ib(\Phi) = \{a \in \mathbb{A} \,|\, \emptyset \neq \Phi^{-1}(a) \neq \{a\}\}$ (image base of Φ); by definition, $ib(\Phi)$ is finite. The *syntactic sort* of P is $\mathcal{L}(P) = \{a \in \mathbb{A} \,|\, a \text{ occurs in } P\} \cup \bigcup_{\Phi \text{ occurs in } P} ib(\Phi)$, where occurrence means being part of the syntactic structure of P.

Proposition 4.9 Let $P \in \tilde{\mathbb{L}}$.

1. $\mathcal{A}(P) \subseteq \ell_d(P) \cup \{\tau\}$ and $\ell_d(P) \subseteq \ell_c(P)$
2. For each $w \in \mathsf{CL}_\tau(P)$ there is a $v \in \mathsf{DL}_\tau(P)$ with $act(v) = act(w)$ and no time steps.
3. $\ell_c(P)$ and $\ell_d(P)$ coincide and will both be denoted by $\ell(P)$ (*semantic sort of* P).
4. $\ell(P)$ is finite.

Proof: 1. clear.

2. We construct an adequate v performing induction on $|w|$ and show additionally that for P_w and P_v reached after w and v resp. we have $P_v \succ_1 P_w$.

By Proposition 4.4.1, $P \succ_0 P$, and by Proposition 4.4.5 $P \succ_1 P$, hence we are done for $w = \lambda$. Thus assume there is an adequate v for a given w and $P_v \succ_1 P_w$.

If $w' = w\alpha$ for $\alpha \in \mathbb{A}_\tau$, then by Proposition 4.4.7, $P_w \xrightarrow{\alpha} P_{w'}$ and $P_v \xrightarrow{\alpha} P_{v'}$ for some $P_{v'}$ such that $P_{w'} \succ_1 P_{v'}$, hence we have $v\alpha \in \mathsf{DL}_\tau(P)$, and there is no time step in $v\alpha$ since there is none in v.

If $w' = w\rho$ for $\rho \in \mathbb{T}$, then $P_w \xrightarrow{\rho}_c P_{w'}$ and by Proposition 4.4.8.b) and $0 \leq \rho \leq 1$, $P_v \succ_{1-\rho} P_{w'}$, hence by Proposition 4.4.5 $P_v \succ_1 P_{w'}$, thus we may choose $v' = v$.

3. from 1. and 2.

4. Obviously, $\mathcal{L}(P\{Q/x\}) \subseteq \mathcal{L}(P) \cup \mathcal{L}(Q)$, which will be used in the Rec-case below, and $\mathcal{L}(P)$ is finite; we show $\ell(P) \subseteq \mathcal{L}(P)$ and are done.

By Proposition 2.4.2, $P \xrightarrow{a}$ iff $a \in \mathcal{A}(P)$ for $a \in \mathbb{A}$, and by 1. and 2. it suffices to show by induction on the structure of P that $\mathcal{A}(P) \setminus \{\tau\} \subseteq \mathcal{L}(P)$ and that $P \xrightarrow{\alpha} P'$ implies $\mathcal{L}(P') \subseteq \mathcal{L}(P)$:

Clear for $\mathbf{0}$, Ω and $x \in \mathcal{X}$.

Pref: $\mathcal{A}(\langle\alpha, r\rangle.P) \setminus \{\tau\} = \{\alpha\} \setminus \{\tau\} \subseteq \mathcal{L}(\langle\alpha, r\rangle.P)$; furthermore, $\langle\alpha, r\rangle.P \xrightarrow{\alpha} P$ and $\mathcal{L}(P) \subseteq \mathcal{L}(\langle\alpha, r\rangle.P)$.

Sum,Par: induction

Rel: Consider $a \in \mathbb{A}$ with $a \in \mathcal{A}(P[\Phi]) = \Phi(\mathcal{A}(P))$; if $a \in ib(\Phi)$, then $a \in \mathcal{L}(P[\Phi])$, otherwise $\Phi^{-1}(a) = \{a\}$ and $a \in \mathcal{A}(P) \setminus \{\tau\} \subseteq \mathcal{L}(P) \subseteq \mathcal{L}(P[\Phi])$ by induction and definition. Furthermore, $P[\Phi] \xrightarrow{\alpha} P'[\Phi]$ implies $P \xrightarrow{\beta} P'$ for some $\beta \in \mathbb{A}_\tau$ with $\Phi(\beta) = \alpha$, hence by induction $\mathcal{L}(P') \subseteq \mathcal{L}(P)$, thus $\mathcal{L}(P'[\Phi]) = \mathcal{L}(P') \cup ib(\Phi) \subseteq \mathcal{L}(P) \cup ib(\Phi) = \mathcal{L}(P[\Phi])$.

Rec: By induction $\mathcal{A}(\mu x.P) \setminus \{\tau\} = \mathcal{A}(P) \setminus \{\tau\} \subseteq \mathcal{L}(P) = \mathcal{L}(\mu x.P)$.
Furthermore, $\mu x.P \xrightarrow{\alpha} P'\{\mu x.P/x\}$ implies $P \xrightarrow{\alpha} P'$, hence by ind. $\mathcal{L}(P') \subseteq \mathcal{L}(P)$, thus $\mathcal{L}(P'\{\mu x.P/x\}) \subseteq \mathcal{L}(P') \cup \mathcal{L}(\mu x.P) = \mathcal{L}(P)$.

□

5 Characterization

As a consequence of the last section, from now on we let $\sqsupseteq$ denote the (coinciding) preorders $\sqsupseteq_c$ and $\sqsupseteq_d$. Furthermore, we will henceforth only deal with *discrete* processes and their discrete behaviour.

We first modify the SOS-rules for wait-time as follows: we only allow unit time steps and record at each time step a so-called *refusal set* X of actions which are *not* waiting; i.e. these actions are *not* urgent, they do not have to be performed and can be refused at this moment. Note that in contrast to wait-time we now prohibit passage of time if an urgent τ can be performed. Just as wait-time is a very generous relaxation of idle-time which does not care whether any action is urgent or not, also the new time semantics is a relaxation of (discrete) idle time: when a time step occurs, all actions in $X \cup \{\tau\}$ are treated correctly w.r.t. passage of idle time, but the other actions might be urgent.

Definition 5.1 (*SOS-rules for refusal of actions, refusal traces*) The following SOS-rules define a relation $\xrightarrow{X}_r \subseteq (\tilde{\mathbb{P}} \times \tilde{\mathbb{P}})$, for each $X \subseteq \mathbb{A}$:

$$\text{Nil}_r \ \frac{}{\mathbf{0} \xrightarrow{X}_r \mathbf{0}} \qquad \text{Pref}_{r1} \ \frac{}{\alpha.P \xrightarrow{X}_r \underline{\alpha}.P} \qquad \text{Pref}_{r2} \ \frac{\alpha \notin X \cup \{\tau\}}{\underline{\alpha}.P \xrightarrow{X}_r \underline{\alpha}.P}$$

$$\text{Par}_r \ \frac{\forall_{i=1,2}\ P_i \xrightarrow{X_i}_r P_i',\ X \subseteq (A \cap \bigcup_{i=1,2} X_i) \cup ((\bigcap_{i=1,2} X_i) \setminus A)}{P_1 \|_A P_2 \xrightarrow{X}_r P_1' \|_A P_2'}$$

$$\text{Sum}_r \ \frac{\forall_{i=1,2}\ P_i \xrightarrow{X}_r P_i'}{P_1 + P_2 \xrightarrow{X}_r P_1' + P_2'} \qquad \text{Rel}_r \ \frac{P \xrightarrow{\Phi^{-1}(X \cup \{\tau\}) \setminus \{\tau\}}_r P'}{P[\Phi] \xrightarrow{X}_r P'[\Phi]} \qquad \text{Rec}_r \ \frac{P \xrightarrow{X}_r P'}{\mu x.P \xrightarrow{X}_r P'\{\mu x.P/x\}}$$

When $P \xrightarrow{X}_r P'$, we call this a *time step*.

For process terms $P, P' \in \tilde{\mathbb{P}}$, we write $P \xrightarrow{\varepsilon}_r P'$, if either $\varepsilon = \alpha \in \mathbb{A}_\tau$ and $P \xrightarrow{\alpha} P'$, or $\varepsilon = X \subseteq \mathbb{A}$ and $P \xrightarrow{X}_r P'$. For sequences w, we define $P \xrightarrow{w}_r P'$ and $P \stackrel{w}{\Longrightarrow}_r P'$ analogously to Definition 2.13.

$\mathsf{RT}_\tau(P) = \{w \mid P \xrightarrow{w}_r\}$ is the set of *τ-refusal traces* of P, and $\mathsf{RT}(P) = \{w \mid P \stackrel{w}{\Longrightarrow}_r\}$ is the set of *refusal traces* of P. $act(w)$ and $\zeta(w)$ are extended to elements from $\mathsf{RT}_\tau(P)$ and $\mathsf{RT}(P)$, i.e. $\zeta(w)$ is the number of time steps (sets) in w.

As for discrete traces, we note that the sets of processes, process terms resp., are closed under performance of refusal traces, i.e. $P \in \mathbb{P}$ and $P \xrightarrow{w}_r P'$ for some $w \in \mathsf{RT}_\tau(P)$ implies $P' \in \mathbb{P}$ again etc.

By Proposition 5.2.1 below, the set of possible refusal sets at a time step is downward closed w.r.t. set inclusion, and by .3, not activated actions can always be refused. Proposition 5.2.4 links time steps to unit-time-waiting, unit-time-idling resp. Finally, Proposition 5.2.5 is an element needed in the treatment of recursion (Section 6), stating that guarded subterms of a process term are not affected by or involved in time steps or occurrence of actions.

Proposition 5.2 Let $P, Q, R \in \tilde{\mathbb{P}}$ be process terms, let $X, X' \subseteq \mathbb{A}$, let $x \in \mathcal{X}$ and let $\varepsilon \in (\mathbb{A}_\tau \cup 2^{\mathbb{A}})$.

1. If $P \xrightarrow{X}_r Q$ and $X' \subseteq X$, then $P \xrightarrow{X'}_r Q$.
2. If $P \xrightarrow{X}_r Q$ and $P \xrightarrow{X'}_r R$, then $Q \equiv R$.
3. If $P \xrightarrow{X}_r Q$ and $X' \cap \mathcal{A}(P) = \emptyset$, then $P \xrightarrow{X \cup X'}_r Q$.
4. $P \xrightarrow{X}_r Q$ if and only if $P \overset{1}{\rightsquigarrow}_c Q$ and $\forall_{\alpha \in X \cup \{\tau\}}\ \mathcal{R}(\alpha, P) = 1$,
   in particular $P \xrightarrow{\mathbb{A}}_r Q$ if and only if $P \xrightarrow{1}_d Q$.
5. Let x be guarded in P. Then $P\{Q/x\} \xrightarrow{\varepsilon}_r R$ if and only if there exists $P' \in \tilde{\mathbb{P}}$ with $P \xrightarrow{\varepsilon}_r P'$ and $R \equiv P'\{Q/x\}$.
6. If P is not guarded, then $\mathsf{RT}_\tau(P) = \mathsf{RT}(P) = \{\lambda\}$.

Proof: For **1.** to **4.** use structural induction on P. For **3.**, also use 2.4.3; in the Par-case, this shows that $X' \cap A$ can be added to X_1 or X_2 by induction, while $X' \setminus A$ can be added to X_1 and X_2.

The additional property of **4.** follows with Definition 2.8 of $\mathcal{R}(P)$, $\mathcal{A}(P) \subseteq \mathbb{A}_\tau$, 2.9.1 and Definition 2.11.

4. If $\varepsilon = \alpha \in \mathbb{A}_\tau$, we are done by Proposition 2.4.2, hence let $\varepsilon = X \subseteq \mathbb{A}$. Then $P\{Q/x\} \xrightarrow{X}_r R$ iff $P\{Q/x\} \overset{1}{\rightsquigarrow}_c R$ and $\forall_{\alpha \in X \cup \{\tau\}} \mathcal{R}(\alpha, P\{Q/x\}) = 1$ by 4. iff $P \overset{1}{\rightsquigarrow}_c P'$ for some P' such that $R \equiv P'\{Q/x\}$ and $\forall_{\alpha \in X \cup \{\tau\}} \mathcal{R}(\alpha, P) = 1$ by Lemma 2.7.3 and Proposition 2.9.2 iff $P \xrightarrow{X}_r P'$ and $R \equiv P'\{Q/x\}$ by 4. again.

6. from Definition 2.3 and with induction from Definition 5.1. □

We now state that refusal traces refine the discrete language, which is due to Proposition 5.2.4 (part 2).

Theorem 5.3 Let $P, Q \in \tilde{\mathbb{P}}$ be process terms; then $\mathsf{RT}(P) \subseteq \mathsf{RT}(Q)$ implies $\mathsf{DL}(P) \subseteq \mathsf{DL}(Q)$.

Proof: By Proposition 5.2.4, $P \xrightarrow{1}_d P'$ iff $P \xrightarrow{\mathbb{A}}_r P'$, hence $\mathsf{DL}(P)$ can be derived from those $w \in \mathsf{RT}(P)$ where $X = \mathbb{A}$ for all refusal sets X in w, replacing $\mathbb{A}$ by 1. □

From now on we denote refusal-trace-inclusion and -equivalence of *processes* by $\leq_r$, $=_r$ resp., and lift this relation to process terms as usual via closed substitutions:

Definition 5.4 Let $P, Q \in \tilde{\mathbb{P}}$ be process terms. We write $P \leq_r Q$ if for all closed substitutions $\mathcal{S} : \mathcal{X} \mapsto \mathbb{P}$ we have $\mathsf{RT}([P]_\mathcal{S}) \subseteq \mathsf{RT}([Q]_\mathcal{S})$. We write $P =_r Q$ if $P \leq_r Q$ and $Q \leq_r P$.

The information on temporal and nondeterministic behaviour of a process provided by refusal traces is very similar to the one e.g. contained in the 'barbs' of TPL (see [14]). But astonishingly, we will be able to observe this with asynchronous and initial – i.e. 'temporally weak' – test processes.

For technical reasons, in the following we do not only consider the RT-semantics but also the RT_τ-semantics: it will play an important rôle when deriving the precongruence property of RT- and RT_τ-inclusion w.r.t. the recursion operator in Section 6. Note that $\mathsf{RT}_\tau(P)$ does not only treat τ's like visible actions: additionally, by Definition 5.1, all refusal sets X in a $w \in \mathsf{RT}_\tau(P)$ implicitly contain τ, i.e. in w after a time step an activated τ must either occur or be disabled before the next time step X.

Before characterizing the testing preorder in Theorem 5.13, the following developments are concerned with (pre)congruence properties of refusal-trace-equivalence (-inclusion). As indicated in Section 4, DL-inclusion is not a precongruence for parallel composition: it does not record runs of a component in which actions are delayed beyond idle time, which in general is necessary in a parallel composition when waiting for a communication partner. We first show that (τ-) refusal traces serve this purpose:

Definition 5.5 (*Shuffle of refusal traces w.r.t A*) Let $u, v \in (\mathbb{A}_\tau \cup 2^\mathbb{A})^*$ and $A \subseteq \mathbb{A}$; then $u \|_A v$ is the set of all $w \in (\mathbb{A}_\tau \cup 2^\mathbb{A})^*$ such that for some n $u = u_1 \ldots u_n$, $v = v_1 \ldots v_n$, $w = w_1 \ldots w_n$ and for all $k = 1, \ldots, n$ one of the following cases applies:

1. $u_k = v_k = w_k = a \in A$
2. $u_k = w_k = \alpha \in \mathbb{A}_\tau \setminus A$ and $v_k = \lambda$
3. $v_k = w_k = \alpha \in \mathbb{A}_\tau \setminus A$ and $u_k = \lambda$
4. $u_k = X_u \subseteq \mathbb{A}$, $v_k = X_v \subseteq \mathbb{A}$, $w_k = X \subseteq \mathbb{A}$ and $X \subseteq (A \cap (X_u \cup X_v)) \cup ((X_u \cap X_v) \setminus A)$

For sets $R_1, R_2 \subseteq (\mathbb{A}_\tau \cup 2^\mathbb{A})^*$, we define $R_1 \|_A R_2 = \bigcup \{u \|_A v \mid u \in R_1,\, v \in R_2\}$.

Observe that if $(v \|_A u) \neq \emptyset$, then by 1. for all $a \in A$ the number of a's is equal in u, v and all w, and by 4. the number of time steps is equal in u, v and all w.

Theorem 5.6 For processes $P_1, P_2 \in \mathbb{P}$, we have $\mathsf{RT}(P_1 \|_A P_2) = \mathsf{RT}(P_1) \|_A \mathsf{RT}(P_2)$ and $\mathsf{RT}_\tau(P_1 \|_A P_2) = \mathsf{RT}_\tau(P_1) \|_A \mathsf{RT}_\tau(P_2)$. In particular, both RT-inclusion and RT_τ-inclusion are precongruences for parallel composition on $\mathbb{P}$.

Proof: First observe that $P_1 \|_A P_2$ is not guarded if and only if at least one of P_1 or P_2 is not guarded; in this case, we are done by 5.2.6. Hence, assume P_1 and P_2 to be guarded.

It suffices to show the claim for RT-semantics; the same technique then applies for RT_τ-semantics, where τ's are treated like visible actions. Let $P \equiv P_1 \|_A P_2$.

'$\subseteq$':
Let $v \in \mathsf{RT}(P)$. Then there is a $w \in \mathsf{RT}_\tau(P)$ such that $v = w/\tau$. With induction on the length of w one shows that if $P \xrightarrow{w}_r P'$, then there are $w_1 \in \mathsf{RT}_\tau(P_1)$ and $w_2 \in \mathsf{RT}_\tau(P_2)$

such that $w/\tau \in ((w_1/\tau)\|_A(w_2/\tau)) \subseteq (\mathsf{RT}(P_1)\|_A\mathsf{RT}(P_2))$, $P_1 \xrightarrow{w_1}_r P_1'$, $P_2 \xrightarrow{w_2}_r P_2'$ and $P' \equiv P_1'\|_A P_2'$. (For a detailed proof, see [16].)

'$\supseteq$':

Let $v \in (\mathsf{RT}(P_1)\|_A\mathsf{RT}(P_2))$. Then there are $w_1 \in \mathsf{RT}_\tau(P_1)$ and $w_2 \in \mathsf{RT}_\tau(P_2)$ such that $v \in ((w_1/\tau)\|_A(w_2/\tau))$, $P_1 \xrightarrow{w_1}_r P_1'$ and $P_2 \xrightarrow{w_2}_r P_2'$. We show for all such w_1 and w_2 that there is a $w \in \mathsf{RT}_\tau(P)$ with $w/\tau = v \in \mathsf{RT}(P)$ and $P \xrightarrow{w}_r P_1'\|_A P_2'$. We perform induction on $|w_1| + |w_2|$. For $|w_1| + |w_2| = 0$ we choose $w = \lambda$. We now distinguish several cases:

1. $w_1 = w_1'\alpha$ with $\alpha \in \mathbb{A}_\tau \setminus A$; then we apply induction to w_1' and w_2 to get w' and choose $w = w'\alpha$.

2. $w_2 = w_2'\alpha$ with $\alpha \in \mathbb{A}_\tau \setminus A$; analogous.

3. Neither 1. nor 2.: then either $w_1 = w_1'a$ and $w_2 = w_2'a$ with $a \in A$ or $w_1 = w_1'X$ and $w_2 = w_2'Y$ with sets X, Y; we apply induction to w_1' and w_2' to get w' and choose $w = w'a$ or $w = w'Z$ with some $Z \subseteq (A \cap (X \cup Y)) \cup ((X \cap Y) \setminus A)$.

The additional property follows since $\|_A$ on sets is monotonic, i.e. $R_1 \subseteq R_2$ implies $R\|_A R_1 \subseteq R\|_A R_2$. □

We now show that (τ-)refusal-trace-inclusion is also a precongruence for prefix:

Definition 5.7 (*Prefix of refusal traces*) For $R \subseteq (\mathbb{A}_\tau \cup 2^{\mathbb{A}})^*$ and $a \in \mathbb{A}$ we define

1. $a.R$ to be the set of all prefixes of
   $\{X_1 \ldots X_n a \,|\, n \in \mathbb{N}_0,\ X_1 \subseteq \mathbb{A},\ X_2, \ldots, X_n \subseteq \mathbb{A} \setminus \{a\}\} \circ R$,

2. $\underline{a}.R$ to be the set of all prefixes of
   $\{X_1 \ldots X_n a \,|\, n \in \mathbb{N}_0,\ X_1, \ldots, X_n \subseteq \mathbb{A} \setminus \{a\}\} \circ R$,

3. $\tau.R = \{X \,|\, X \subseteq \mathbb{A}\} \cup \{\lambda\} \circ R$,

4. $\underline{\tau}.R = R$.

Theorem 5.8 Let $P \in \mathbb{P}$ be a process; for $\alpha \in \mathbb{A}_\tau$, $\mathsf{RT}(\alpha.P) = \alpha.\mathsf{RT}(P)$ and $\mathsf{RT}(\underline{\alpha}.P) = \underline{\alpha}.\mathsf{RT}(P)$; for $a \in \mathbb{A}$, $\mathsf{RT}_\tau(a.P) = a.\mathsf{RT}_\tau(P)$ and $\mathsf{RT}_\tau(\underline{a}.P) = \underline{a}.\mathsf{RT}_\tau(P)$. Finally, $\mathsf{RT}_\tau(\tau.P)$ is the set of all prefixes of $\{\tau, X\tau \,|\, X \subseteq \mathbb{A}\} \circ \mathsf{RT}_\tau(P)$ and $\mathsf{RT}_\tau(\underline{\tau}.P)$ is the set of all prefixes of $\{\tau\} \circ \mathsf{RT}_\tau(P)$. In particular, both RT-inclusion and RT_τ-inclusion are precongruences for prefixing of (initial) processes.

Proof: From the definitions. □

We allow τ's as guards for recursion, and they actually have some potential visibility in refusal traces due to time steps, but this is not enough for making fixed points unique modulo RT-equivalence: consider $P \equiv \mu x.\tau.x$ and $Q \equiv \mu x.(\tau.x + a.\mathbf{0})$; we have $\tau.x\{P/x\} =_r P$ and $\tau.x\{Q/x\} =_r Q$, but $P \neq_r Q$. This observation will rule out application of Banach's fixed point theorem when treating recursion in Section 6.

For the characterization we will also use the precongruence property of (τ-)refusal-trace-inclusion w.r.t. hiding and relabelling:

Definition 5.9 (*Relabelling of refusal traces*) Let Φ be a general relabelling function, $\alpha \in \mathbb{A}_\tau$, $X \subseteq \mathbb{A}$ and define $\alpha[\Phi]_\tau^{-1} = \Phi^{-1}(\alpha)$ and $X[\Phi]_\tau^{-1} = \{\Phi^{-1}(X \cup \{\tau\}) \setminus \{\tau\}\}$; we extend $[\Phi]_\tau^{-1}$ to sequences $w \in (\mathbb{A}_\tau \cup 2^{\mathbb{A}})^*$ via concatenation $\circ$. We define $[\Phi]^{-1}$ identically, but additionally $\lambda[\Phi]^{-1} = (\Phi^{-1}(\tau) \setminus \{\tau\})^*$. $[\Phi]^{-1}$ is again extended to sequences.

Theorem 5.10 For a process $P \in \mathbb{P}$ and a general relabelling function Φ we have

1. $\mathsf{RT}(P[\Phi]) = \{w \in (\mathbb{A} \cup 2^{\mathbb{A}})^* \mid w[\Phi]^{-1} \cap \mathsf{RT}(P) \neq \emptyset\}$
2. $\mathsf{RT}_\tau(P[\Phi]) = \{w \in (\mathbb{A}_\tau \cup 2^{\mathbb{A}})^* \mid w[\Phi]_\tau^{-1} \cap \mathsf{RT}_\tau(P) \neq \emptyset\}$

Furthermore, both RT-inclusion and RT_τ-inclusion are precongruences for general relabelling $P[\Phi]$ of processes, in particular for relabelling $P[f]$ and hiding P/A.

Proof: From the definitions. □

Another property needed for the above mentioned test construction is that $\mathbf{0}$ is a zero element for both choice and parallel composition without synchronisation:

Proposition 5.11 Let $P \in \tilde{\mathbb{P}}$ be a process term; then $P \parallel_\emptyset \mathbf{0} =_r P$ and $P + \mathbf{0} =_r P$.

Proof: We can assume $P \in \mathbb{P}$ and P is guarded. First, $\mathsf{RT}(P \parallel_\emptyset \mathbf{0}) = \mathsf{RT}(P) \parallel_\emptyset \mathsf{RT}(\mathbf{0})$ by Theorem 5.6. Now since by Definition 5.1 $\mathsf{RT}(\mathbf{0}) = \{X_1 \dots X_n \mid n \in \mathbb{N},\, X_1, \dots, X_n \subseteq \mathbb{A}\}$, Proposition 5.2.1 and Definition 5.5 (where only cases 2 and 4 apply, since $A = \emptyset$) yield $\mathsf{RT}(P) \parallel_\emptyset \mathsf{RT}(\mathbf{0}) = \mathsf{RT}(P)$.

Second, one shows for all $v \in (\mathbb{A}_\tau \cup 2^{\mathbb{A}})^*$ by induction on $|v|$: $P + \mathbf{0} \xrightarrow{v}_r Q$ if and only if $P \xrightarrow{v}_r Q$ or $P \xrightarrow{v}_r R$ for some R with $Q \equiv R + \mathbf{0}$; use that $\mathbf{0}$ can do arbitrary time steps, but no action that would decide the choice. □

Finally, we state that refusal traces of Ω-free processes can always be extended by a time step (after performing all urgent internal activity) and that time steps can be omitted. The latter demonstrates how tightly the timed behaviour of an initial process P according to the RT-semantics is related to the usual 'CCS-behaviour' of the CCS-like process P: a usual run of P is just an RT-run of P with only functional behaviour, i.e. according to Definition 2.3; if we delete all time steps in an RT-run of P, we obtain an RT-run of P with only functional behaviour, hence a usual run of P.

Proposition 5.12 Let P, P', P'' be Ω-free $\mathbb{P}$ processes, $w, w' \in (\mathbb{A} \cup 2^{\mathbb{A}})^*$ and $X \subseteq \mathbb{A}$.

1. $w \in \mathsf{RT}(P)$ if and only if $w\emptyset \in \mathsf{RT}(P)$.
2. $wXw' \in \mathsf{RT}(P)$ implies $ww' \in \mathsf{RT}(P)$.

Proof: 1. 'if': clear
'only-if': assume $P \stackrel{w}{\Longrightarrow}_r P'$ for some $P' \in \mathbb{P}$; then by Proposition 4.2.2 there is a $t \in \{\tau\}^*$ such that $P' \xrightarrow{t}_r P''$ for some $P'' \in \mathbb{P}$ with $\mathcal{R}(\tau, P'') = 1$; now $P'' \xrightarrow{\emptyset}_r$ by Proposition 5.2.4 and Lemma 2.7.1 since P'' is closed, hence guarded.

2. $wXw' \in \mathsf{RT}(P)$ implies $wXw' = (uXv)/\tau$ for some $uXv \in \mathsf{RT}_\tau(P)$. Now it suffices to show by induction on $|v|$ that $uXv \in \mathsf{RT}_\tau(P)$ implies $uv \in \mathsf{RT}_\tau(P)$, where we additionally show that for P_1 reached after uv and P_2 reached after uXv we have $P_1 \succ_0 P_2$.

The base case is $v = \lambda$, hence $P \xrightarrow{u}_r P_1$ and $P_1 \succ_1 P_1$ by Proposition 4.4.5. Then by Proposition 5.2.4 and Proposition 4.4.9 b), $P_1 \xrightarrow{X}_r P_2$ implies $P_1 \succ_0 P_2$. Now assume the property to hold for v.

If $v' = v\alpha$ for $\alpha \in \mathbb{A}_\tau$, then $P_2 \xrightarrow{\alpha} P_2'$ for some P_2', and since $P_1 \succ_0 P_2$ by assumption, Proposition 4.4.7 implies $P_1 \xrightarrow{\alpha} P_1'$ for some P_1', such that $P_1' \succ_0 P_2'$ again.

If $v' = vX'$ for $X' \subseteq \mathbb{A}$, then $P_2 \xrightarrow{X'}_r P_2'$ for some P_2' implies $P_2 \overset{1}{\rightsquigarrow}_c P_2'$ and for all $\alpha \in X' \cup \{\tau\}$ we have $\mathcal{R}(\alpha, P_2) = 1$ by Proposition 5.2.4. Hence, for $P_1 \overset{1}{\rightsquigarrow}_c P_1'$, $P_1 \succ_0 P_2$ and Proposition 4.4.9 c) imply that $P_1' \succ_0 P_2'$; furthermore by Proposition 4.4.4, for all $\alpha \in X' \cup \{\tau\}$ we have $\mathcal{R}(\alpha, P_1) \geq \mathcal{R}(\alpha, P_2) - 0 = 1$, thus finally $P_1 \xrightarrow{X'}_r P_1'$ by Proposition 5.2.4. □

We now have collected all elements for characterising the efficiency preorder via refusal-trace-inclusion, which is our second main result:

Theorem 5.13 (*Characterization of the testing preorder*) Let P_1, P_2 be testable processes. Then $P_1 \sqsupseteq P_2$ if and only if $P_1 \leq_r P_2$.

Proof: 'if':
Let (O, D) be a timed test. Then $\mathsf{RT}(P_1) \subseteq \mathsf{RT}(P_2)$ implies $\mathsf{DL}(\tau.P_1 \| O) \subseteq \mathsf{DL}(\tau.P_2 \| O)$ by Theorem 5.8, Theorem 5.6 and Theorem 5.3. Thus, if P_1 fails the test due to some $w_1 \in \mathsf{DL}(\tau.P_1 \| O)$, then so does P_2.

'only if':
We assume $P_1 \sqsupseteq P_2$ and take some $w_1 \in \mathsf{RT}(P_1)$. By Definition 5.1, Proposition 5.2.4 and Definition 4.8, all actions in w_1 are in $\ell(P_1) \cup \ell(P_2)$. Furthermore, by Proposition 5.2.3 and .1, we may assume that for all refusal sets X in w_1 we have $X \subseteq \ell(P_1) \cup \ell(P_2)$, which is finite due to Proposition 4.9.4.

Now let $w = w_1$ if $w_1 = \lambda$ and $w = w_1\emptyset$ otherwise; by Proposition 5.12.1, $w \in \mathsf{RT}(P_1)$, too. Furthermore $Xw \in \mathsf{RT}(\tau.P_1)$ for each $X \subseteq \ell(P_1) \cup \ell(P_2)$ by Theorem 5.8 and Definition 5.7.3.

We will construct a timed test $(O_{Xw}, \zeta(w))$ that is failed by a testable process $P \in \mathbb{P}$ if and only if $Xw \in \mathsf{RT}(\tau.P)$. Hence, P_1 fails $(O_{Xw}, \zeta(w))$, thus by assumption P_2 fails $(O_{Xw}, \zeta(w))$, too, and we conclude $Xw \in \mathsf{RT}(\tau.P_2)$. But then $Xw_1 \in \mathsf{RT}(\tau.P_2)$ by Proposition 5.12.1 and $w_1 \in \mathsf{RT}(P_2)$ or $Xw_1 \in \mathsf{RT}(P_2)$ by Theorem 5.8 and Definition 5.7.2, i.e. $w_1 \in \mathsf{RT}(P_2)$ by Proposition 5.12.2, and we are done.

Note that it actually suffices to consider the case where $X = \emptyset$, which implies that Xw ends with $\emptyset$. To make induction work, we will treat arbitrary sequences Xw that end with $\emptyset$, but at the end of the proof we will come back to the observation that additionally we can assume $X = \emptyset$.

The proof is structured as follows: We first give the construction of O_{Xw} (1), then we show that P fails the test $(O_{Xw}, \zeta(w))$ if $\tau.P$ performs Xw (2), and finally we show that P fails the test $(O_{Xw}, \zeta(w))$ *only if* $\tau.P$ is able to perform Xw (3). All three parts are inductive w.r.t. the structure of w.

(1)
We define O_{Xw} for sequences Xw that end with $\emptyset$. Furthermore, all actions of Xw are in $\ell(P_1) \cup \ell(P_2)$ and all refusal sets are subsets of $\ell(P_1) \cup \ell(P_2)$.

O_{Xw} will consist of several components that communicate via synchronized actions which must not occur in the sort of P_1 or P_2. Hence, let $H = \{b_0, c_0, b_1, c_1, \dots\} \subseteq \mathbb{A}$ be an infinite

set such that $H \cap (\ell(P_1) \cup \ell(P_2)) = \emptyset$; H exists since $\ell(P_1)$ and $\ell(P_2)$ are finite and $\mathbb{A}$ is infinite.

The components Q_{Xw}, R'_{Xw}, S_{Xw} and R_{Xw} of O_{Xw} are defined inductively as follows:

The base case is $Xw = \emptyset$:

$Q_\emptyset \equiv \omega.\mathbf{0}$

$S_\emptyset \equiv \mathbf{0}$

$R_\emptyset \equiv \mathbf{0}$

Now let the general case be $Xw = Xa_1 \ldots a_n X'w'$, where $X'w'$ ends with $\emptyset$. We define:

$Q_{Xw} \equiv (b_{\zeta(w)}.Q_{X'w'}) \parallel_\emptyset (c_{\zeta(w)}.\mathbf{0} + \omega.\mathbf{0})$

$S_{Xw} \equiv b_{\zeta(w)}.a_1 \ldots a_n.c_{\zeta(w)}.S_{X'w'}$

$R_{Xw} \equiv (b_{\zeta(w)}.R_{X'w'}) \parallel_\emptyset (c_{\zeta(w)-1}.\mathbf{0} + \sum_{a \in X'} a.\mathbf{0})$

In both cases let

$$R'_{Xw} \equiv (R_{Xw} \parallel_\emptyset c_{\zeta(w)}.\mathbf{0})$$

and finally $O_{Xw} \equiv T_{Xw}/H$ where

$$T_{Xw} \equiv Q_{Xw} \parallel_H S_{Xw} \parallel_H R'_{Xw}$$

Before detailed formal reasoning, the function and the interplay of the parts are shortly and informally described in the following:

The part $Xa_1 \ldots a_n$ of $Xw = Xa_1 \ldots a_n X'w'$ is called the $\zeta(w)$-th *round* of Xw, started by occurrence of X, whereas occurrence of X' marks the begin of the $(\zeta(w)-1)$-th round.

Q_{Xw} is the 'clock'-part of the test, which for each round i of Xw enables an ω that is urgent after the time step starting round i and can only be deactivated by performing the auxiliary action c_i (*completion* of round i) before the next time step.

The 'action-sequence'-part S_{Xw} will ensure that c_i can only occur after the action sequence $a_1 \ldots a_n$, which itself must be preceded by the auxiliary action b_i (*begin* of round i). Furthermore, occurrence of b_i triggers the activation of the ω for the next round by enabling $Q_{X'w'}$. This must not happen too early, i.e. b_i and hence c_i will be performed after the time step starting round i and before the next one.

At the beginning of the present round, the 'refusal-set'-part R_{Xw} enables all actions a from the refusal set X' of the following round in conflict with the auxiliary action c_{i-1} which has to occur only at completion of the following round. After the time-step of the present round, all a from X' have become urgent, but may not occur – i.e. must be refusable by the tested process at the time-step starting the following round.

Finally, R_{Xw} is augmented to R'_{Xw} for proof-technical reasons, T_{Xw} puts all three parts via synchronisation together, and O_{Xw} hides the auxiliary actions away. Otherwise, they would have to synchronise with the tested process, which is of course impossible by the definition of H.

(2)

On the one hand, by Definition 4.6, P fails the test $(O_{Xw}, \zeta(w))$ if and only if there is a $u \in \mathsf{DL}(\tau.P \| O_{Xw})$ without ω and with $\zeta(u) > \zeta(w)$. By Proposition 5.2.4, this is case if and only if there is a $v \in \mathsf{RT}(\tau.P \| O_{Xw})$ without ω and with $\zeta(v) > \zeta(w)$ and all refusal sets in v are $\mathbb{A}$. By Theorem 5.6, Proposition 5.2.1 and Definition 5.5, such a v exists if and only if $v \in (v_1 \| v_2)$ for some $v_1 \in \mathsf{RT}(\tau.P)$ and $v_2 \in \mathsf{RT}(O_{Xw})$ satisfying the following: $\zeta(v_1) = \zeta(v_2) > \zeta(w)$, both v_1 and v_2 are without ω, all refusal sets in both v_1 and v_2 contain ω, and $match(v_1) = v_2$, where *match* is defined inductively as follows :

1. $match(\lambda) = \lambda$.
2. $match(av') = a\, match(v')$ for $a \in \mathbb{A}$.
3. $match(Xv') = \overline{X}\, match(v')$ for $X \subseteq \mathbb{A}$, where $\overline{X}$ denotes $\{\omega\} \cup \mathbb{A} \setminus X$.

On the other hand, for any testable process P we have $\omega \notin \ell(P)$, hence by Proposition 4.9.1 and Proposition 5.2.3 (and .1) we have $Xw \in \mathsf{RT}(\tau.P)$ (if and) only if $v_1 \in \mathsf{RT}(\tau.P)$, where v_1 is Xw with each refusal set augmented by ω; also, $match(Xw) = match(v_1)$. Hence, in order show that P fails the test $(O_{Xw}, \zeta(w))$ if $Xw \in \mathsf{RT}(\tau.P)$, with the above it suffices to show that $match(Xw) = match(v_1) \in \mathsf{RT}(O_{Xw})$.

In order to apply inductive reasoning, we consider an intermediate state that is reached when O_{Xw} performs $match(Xw)$. Let

$$R^+_{Xw} \equiv R_{Xw} \parallel_\emptyset (\underline{c}_{\zeta(w)}.\mathbf{0} + \sum\nolimits_{a \in X} \underline{a}.\mathbf{0})$$

and let $O^+_{Xw} \equiv T^+_{Xw}/H$ where

$$T^+_{Xw} \equiv Q_{Xw} \parallel_H S_{Xw} \parallel_H R^+_{Xw}$$

We first observe that

(2.1) $O_\emptyset \stackrel{\mathbb{A}}{\Longrightarrow}_r$
(2.2) $O^+_\emptyset \stackrel{\mathbb{A}}{\Longrightarrow}_r$

since (2.1) $O_\emptyset$ is initial and (2.2) $\sum_{a\in\emptyset} \underline{a}.\mathbf{0} \equiv \mathbf{0} \in \mathbb{P}_1$ by definition, $Q_\emptyset, S_\emptyset \in \mathbb{P}_1$, and $\underline{c}_0$ is the only urgent action in $R^+_\emptyset$, but has no synchronization partner in $Q_\emptyset$ and $S_\emptyset$, thus $\mathcal{R}(O^+_\emptyset) = 1$, and we are done by Proposition 5.2.4.

Now let $Xw = Xa_1 \ldots a_n X'w'$. We show the following properties:

(2.3) $O_{Xw} \stackrel{\overline{X}a_1\ldots a_n}{\Longrightarrow}_r O'_{Xw} =_r O^+_{X'w'}$
(2.4) $O^+_{Xw} \stackrel{\overline{X}a_1\ldots a_n}{\Longrightarrow}_r O^{+'}_{Xw} =_r O^+_{X'w'}$

I.e. from both O_{Xw} and O^+_{Xw} by performing a sequence matching the $\zeta(w)$-th round of w, we reach a process that is RT-equivalent to $O^+_{X'w'}$. For the proof of (2.3) consider (using Proposition 5.11)

$$\begin{array}{lllll}
Q_{Xw} & \stackrel{\mathbb{A}}{\rightarrow}_r & \underline{b}_{\zeta(w)}.Q_{X'w'} \parallel_\emptyset (\underline{c}_{\zeta(w)}.\mathbf{0} + \underline{\omega}.\mathbf{0}) & \stackrel{b_{\zeta(w)}c_{\zeta(w)}}{\longrightarrow}_r & Q_{X'w'} \parallel_\emptyset \mathbf{0} \;=_r\; Q_{X'w'} \\
S_{Xw} & \stackrel{\mathbb{A}}{\rightarrow}_r & \underline{b}_{\zeta(w)}.a_1 \ldots a_n.c_{\zeta(w)}.S_{X'w'} & \stackrel{b_{\zeta(w)}a_1\ldots a_n c_{\zeta(w)}}{\longrightarrow}_r & S_{X'w'}
\end{array}$$

and

$$\begin{array}{rlll}
R'_{Xw} \equiv R_{Xw} & \parallel_\emptyset & c_{\zeta(w)}.\mathbf{0} & \stackrel{\mathbb{A}}{\rightarrow}_r \\
(\underline{b}_{\zeta(w)}.R_{X'w'} \parallel_\emptyset (\underline{c}_{\zeta(w)-1}.\mathbf{0} + \sum_{a\in X'} \underline{a}.\mathbf{0})) & \parallel_\emptyset & \underline{c}_{\zeta(w)}.\mathbf{0} & \stackrel{b_{\zeta(w)}c_{\zeta(w)}}{\longrightarrow}_r \\
(R_{X'w'} \parallel_\emptyset (\underline{c}_{\zeta(w)-1}.\mathbf{0} + \sum_{a\in X'} \underline{a}.\mathbf{0})) & \parallel_\emptyset & \mathbf{0} & =_r \\
(R_{X'w'} \parallel_\emptyset (\underline{c}_{\zeta(w)-1}.\mathbf{0} + \sum_{a\in X'} \underline{a}.\mathbf{0})) & & & \equiv \quad R^+_{X'w'}
\end{array}$$

Hence with synchronisation over H by Definition 5.5 and Theorem 5.6 we get

$$T_{Xw} \stackrel{\mathbb{A}b_{\zeta(w)}a_1\ldots a_n c_{\zeta(w)}}{\longrightarrow}_r \underline{T}'_{Xw} =_r (Q_{X'w'} \parallel_H S_{X'w'}) \parallel_H R^+_{X'w'} \equiv T^+_{X'w'}$$

and hiding H by Definition 5.9 and Theorem 5.10 and finally applying Proposition 5.2.1:

$$O_{Xw} \stackrel{\overline{X}a_1 \ldots a_n}{\Longrightarrow}_r O'_{Xw} =_r O^+_{X'w'}$$

For (2.4), we consider R^+_{Xw} (using Corollary 5.11 again):

$$\begin{array}{rcll} R^+_{Xw} \equiv R_{Xw} & \|_\emptyset & (\underline{c}_{\zeta(w)}.\mathbf{0} + \sum_{a\in X} \underline{a}.\mathbf{0}) & \stackrel{\overline{X}\setminus\{c_{\zeta(w)}\}}{\longrightarrow}_r \\ (\underline{b}_{\zeta(w)}.R_{X'w'} \|_\emptyset (\underline{c}_{\zeta(w)-1}.\mathbf{0} + \sum_{a\in X'} \underline{a}.\mathbf{0})) & \|_\emptyset & (\underline{c}_{\zeta(w)}.\mathbf{0} + \sum_{a\in X} \underline{a}.\mathbf{0}) & \stackrel{b_{\zeta(w)}c_{\zeta(w)}}{\longrightarrow}_r \\ (R_{X'w'} \|_\emptyset (\underline{c}_{\zeta(w)-1}.\mathbf{0} + \sum_{a\in X'} \underline{a}.\mathbf{0})) & \|_\emptyset & \mathbf{0} & =_r \\ (R_{X'w'} \|_\emptyset (\underline{c}_{\zeta(w)-1}.\mathbf{0} + \sum_{a\in X'} \underline{a}.\mathbf{0})) & & & \equiv \quad R^+_{X'w'} \end{array}$$

Hence, as above:

$$O^+_{Xw} \stackrel{\overline{X}a_1 \ldots a_n}{\Longrightarrow}_r O^{+'}_{Xw} =_r O^+_{X'w'}$$

Using these properties, we now perform induction on the length of Xw to show that $match(Xw) \in \mathsf{RT}(O^+_{Xw})$:

For $Xw = \emptyset$, by (2.2) we have $match(\emptyset) = \mathbb{A} \in \mathsf{RT}(O^+_{Xw})$. For $Xw = Xa_1 \ldots a_n X'w'$ by (2.4) we have $\overline{X}a_1 \ldots a_n\, match(X'w') \in \mathsf{RT}(O^+_{Xw})$ by induction.

It remains to show $match(Xw) \in \mathsf{RT}(O_{Xw})$: for $Xw = \emptyset$ we are done by (2.1); for $Xw = Xa_1 \ldots a_n X'w'$, we have $O_{Xw} \stackrel{\overline{X}a_1 \ldots a_n}{\Longrightarrow}_r O'_{Xw} =_r O^+_{X'w'}$ by (2.3) and $match(X'w') \in \mathsf{RT}(O^+_{X'w'})$ by the above, hence we are done.

(3)

We now show that P fails the test $(O_{Xw}, \zeta(w))$ only if $\tau.P$ is able to perform Xw.

We say that a refusal trace $v \in \mathsf{RT}(O_{Xw})$ refuses ω if ω does not occur in v but in all refusal sets of v. Now by Theorem 5.6, Definition 5.5 and analogous arguments as in the beginning of part (2), P can fail the test $(O_{Xw}, \zeta(w))$ only if there is a $v \in \mathsf{RT}(O_{Xw})$ that refuses ω with $\zeta(v) > \zeta(w)$ and $match(v) \in \mathsf{RT}(\tau.P)$. We will show that this implies $Xw \in \mathsf{RT}(\tau.P)$ and are done.

By $V(O_{Xw})$ we denote the set of all $v \in \mathsf{RT}(O_{Xw})$ that refuse ω and satisfy $\zeta(v) > \zeta(w)$, and similarly for Q_{Xw} etc. We will determine $V(O_{Xw})$ by induction on the length of Xw, where we first state the following properties:

The base case is $w = \lambda$ and $X = \emptyset$:

(3.1) $(Q_\emptyset \|_H S_\emptyset) \stackrel{v}{\Longrightarrow}_r$ for a v refusing ω with $\zeta(v) > \zeta(w) = 0$ if and only if $v = X_v$ for some $X_v \subseteq \mathbb{A}$ with $\omega \in X_v$, hence $\zeta(v) = \zeta(w) + 1 = 1$.

(3.2) There is *no* v refusing ω with $\zeta(v) > 1$ such that $((Q_\emptyset \|_\emptyset (c_1.\mathbf{0} + \omega.\mathbf{0})) \|_H (a'_1 \ldots a'_m.c_1.S_\emptyset)) \stackrel{v}{\Longrightarrow}_r$.

Now let $Xw = Xa_1 \ldots a_n X'w'$, where $X'w'$ ends with $\emptyset$:

(3.3) $(Q_{Xw} \|_H S_{Xw}) \stackrel{v}{\Longrightarrow}_r$ for a v refusing ω with $\zeta(v) > \zeta(w)$ if and only if $v = X_v b_{\zeta(w)} a_1 \ldots a_n c_{\zeta(w)} v'$ for some $X_v \subseteq \mathbb{A}$ with $\omega \in X_v$ and v' refuses ω and $\zeta(v) = \zeta(w) + 1$, such that
$(Q_{Xw} \|_H S_{Xw}) \stackrel{X_v b_{\zeta(w)} a_1 \ldots a_n c_{\zeta(w)}}{\Longrightarrow}_r Q'_{X'w'} =_r (Q_{X'w'} \|_H S_{X'w'}) \stackrel{v'}{\Longrightarrow}_r$.

(3.4) There is *no* v refusing ω with $\zeta(v) > \zeta(w) + 1$ such that $(Q_{Xw} \|_\emptyset (c_{\zeta(w)+1}.\mathbf{0} + \omega.\mathbf{0})) \|_H (a'_1 \ldots a'_m.c_{\zeta(w)+1}.S_{Xw}) \stackrel{v}{\Longrightarrow}_r$.

Whereas (3.1) and (3.2) can be checked directly, we show (3.3) and (3.4) by induction using Corollary 5.11 and Theorem 5.6:

(3.3) The if-case is clear. $(Q_{Xw} \parallel_H S_{Xw})$ can perform $b_{\zeta(w)}$, ω or a time step X_v.
Performance of $b_{\zeta(w)}$ yields $(Q_{X'w'} \parallel_\emptyset (c_{\zeta(w)}.\mathbf{0} + \omega.\mathbf{0})) \parallel_H (a_1 \ldots a_n.c_{\zeta(w)}.S_{X'w'})$, and since $\zeta(w') = \zeta(w) - 1$, by ind. and (3.4) or (3.2), no v refusing ω with $\zeta(v) > \zeta(w) = \zeta(w') + 1$ is possible any more.
Hence, v starts with some $X_v \subseteq \mathbb{A}$ with $\omega \in X_v$; afterwards, only $b_{\zeta(w)}a_1 \ldots a_n c_{\zeta(w)}$ is possible, since ω is urgent, hence no time step may occur before its deactivation by $c_{\zeta(w)}$; now a process RT-equivalent to $(Q_{X'w'} \parallel_H S_{X'w'})$ is reached, and $v = X_v b_{\zeta(w)} a_1 \ldots a_n c_{\zeta(w)} v'$. By ind. or (3.1), $\zeta(v') = \zeta(w') + 1$, hence $\zeta(v) = \zeta(w) + 1$.

(3.4) There are two possibilities for an appropriate v:
i) v starts $a'_1 \ldots a'_m c_{\zeta(w)+1}$, reaching a *unique* process, which is RT-equivalent to the process $Q_{Xw} \parallel_H S_{Xw}$; but then (3.3) yields $\zeta(v) = \zeta(w) + 1$ only.
ii) v starts $a'_1 \ldots a'_i X_v a'_{i+1} \ldots a'_m c_{\zeta(w)+1}$ with $0 \leq i \leq m$ and $\omega \in X_v \subseteq \mathbb{A}$, yielding a *unique* process RT-equivalent to $((\underline{b}_{\zeta(w)}.Q_{X'w'}) \parallel_\emptyset (\underline{c}_{\zeta(w)}.\mathbf{0} + \underline{\omega}.\mathbf{0})) \parallel_H S_{Xw}$; now due to the urgent ω, from here only $b_{\zeta(w)}a_1 \ldots a_n c_{\zeta(w)}$ is possible, reaching a *unique* process that is RT-equivalent to $(Q_{X'w'} \parallel_H S_{X'w'})$; but then (3.3) or (3.1) yields only $\zeta(v) = 1 + \zeta(w') + 1 = \zeta(w) + 1$ again.

We are now able to determine the set $V(Q_{Xw} \parallel_H S_{Xw})$: by (3.1), we have $V(Q_\emptyset \parallel_H S_\emptyset) = \{X_v \mid \omega \in X_v \subseteq \mathbb{A}\}$, and by (3.3), for $Xw = Xa_1 \ldots a_n X'w'$ we get

$$V(Q_{Xw} \parallel_H S_{Xw}) = \{X_v \mid \omega \in X_v \subseteq \mathbb{A}\} \circ \{b_{\zeta(w)}a_1 \ldots a_n c_{\zeta(w)}\} \circ V(Q_{X'w'} \parallel_H S_{X'w'})$$

For the following let $l = \zeta(w)$ and Xw be of the form

$$Xw = \; X^l \, a_1^l \ldots a_{n_l}^l \; X^{l-1} \; \ldots \; X^0$$

Hence, by induction, $v_1 \in V(Q_{Xw} \parallel_H S_{Xw})$ is of the form

$$v_1 = \; \Gamma^l \, b_l \, a_1^l \ldots a_{n_l}^l \, c_l \, \Gamma^{l-1} \, b_{l-1} \; \ldots \; c_1 \, \Gamma^0,$$

where $\omega \in \Gamma^i \subseteq \mathbb{A}$ for all $i = 0, \ldots, l$. Now by Theorem 5.6, $v_3 \in V(T_{Xw})$ implies $v_3 \in (v_1 \parallel_H v_2)$, where $v_1 \in V(Q_{Xw} \parallel_H S_{Xw})$ and $v_2 \in \mathsf{RT}(R'_{Xw})$. By the above and Definition 5.5, $(v_1 \parallel_H v_2) \neq \emptyset$ only if v_2 is of the form

$$v_2 = \; u_1^l \, \Upsilon^l \, u_2^l \, b_l \, u_3^l \, c_l \, u_1^{l-1} \, \Upsilon^{l-1} \, u_2^{l-1} \, b_{l-1} \; \ldots \; u_3^1 \, c_1 \, u_1^0 \, \Upsilon^0 \, u_2^0 \, u_3^0,$$

where $\omega \in \Upsilon^i \subseteq \mathbb{A}$ for all $i = 0, \ldots, l$ and $u_j^i \in (\mathbb{A} \setminus (H \cup \{\omega\}))^*$ for $i = 0, \ldots, l$ and $j = 1, \ldots, 3$. If $u_j^i = au$ for some $i = 0, \ldots, l$ and $j = 1, \ldots, 3$ and $a \in \mathbb{A}$, then a must stem from some sum-part of R_{Xw}, hence the respective c_k could not occur any more; observe that the sum-part for c_0 is empty. We conclude $u_j^i = \lambda$ for all $i = 1, \ldots, l$ and all $j = 1, \ldots, 3$.

Furthermore, $R'_{Xw} \stackrel{v_2}{\Longrightarrow}_r R'$ if and only if $R'_{Xw} \stackrel{v_2}{\longrightarrow}_r R'$. The derivations of (2) show that, for v_2 of the form just determined, $R'_{Xw} \stackrel{v_2}{\longrightarrow}_r$ if and only if $\Upsilon^i \subseteq \overline{X}^i \setminus \{c_i\}$ for all $i = 0, \ldots, l-1$. As said in the very beginning of this proof, it suffices to consider the case where $X = \emptyset$ in Xw, hence since $\overline{\emptyset} = \mathbb{A}$, by Definition 5.5 and the above, we determine $v_3 \in V(T_{Xw})$ to be of the form:

$$v_3 = \; \Gamma^l \, b_l \, a_1^l \ldots a_{n_l}^l \, c_l \, \Gamma^{l-1} \, b_{l-1} \; \ldots \; c_1 \, \Gamma^0,$$

where $\Gamma^i \subseteq \overline{X}^i$ for all $i = 0, \ldots, l$. Finally, with Theorem 5.10 and Definition 5.9 we calculate for the form of a $v \in V(O_{Xw})$:

$$v = \ \Gamma^l \ a_1^l \ldots a_{n_l}^l \ \Gamma^{l-1} \ \ldots \ \Gamma^0,$$

where $\Gamma^i \subseteq \overline{X}^i$ for all $i = 0, \ldots, l$, hence

$$match(v) = \ \overline{\Gamma}^l \ a_1^l \ldots a_{n_l}^l \ \overline{\Gamma}^{l-1} \ \ldots \overline{\Gamma}^0,$$

such that $\overline{\Gamma}^i \supseteq X^i$ for $i = 0, \ldots, l$, thus by Proposition 5.2.1, $match(v) \in \mathsf{RT}(\tau.P)$ implies $Xw \in \mathsf{RT}(\tau.P)$, and we are done. □

In our testing scenario, we have restricted ourselves to test processes $O \in \mathbb{P}_1$. We can now argue that this makes our approach stronger, since test processes $O \in \mathbb{P}$ cannot distinguish testable processes any better: if $P_1 \sqsupseteq P_2$, i.e. $P_1 \leq_r P_2$ by the above theorem, then for all $O \in \mathbb{P}$ we have $\tau.P_1 \| O \leq_r \tau.P_2 \| O$ by our precongruence results, hence $\mathsf{DL}(\tau.P_1 \| O) \subseteq \mathsf{DL}(\tau.P_2 \| O)$ by Theorem 5.3; this implies that whenever P_2 satisfies a timed test (O, R) with $O \in \mathbb{P}$, then so does P_1.

It should be remarked though that this observation relies on our definition of timed testing where we embed a process P in a test environment as $\tau.P \| O$; in fact, test processes from $\mathbb{P}$ are stronger in the following sense. One could use test processes O from $\mathbb{P}$ and the usual embedding $P \| O$ in the definition of timed testing, and get the same testing preorder that we have in the present paper. As already remarked above, [15] shows in a Petri-net setting that test processes from $\mathbb{P}_1$ together with the embedding $P \| O$ give a weaker preorder; the latter only becomes the preorder of the present paper if we refine it to a precongruence for prefixing. We have chosen to work out the presented version of timed testing, since it generalizes the preliminary version with its focus on CCS-like processes and since we considered it as less 'traditional' and hence more interesting.

6 Full Abstractness

Refusal-trace-inclusion not only characterizes the efficiency preorder, but also makes just the necessary refinements to discrete behaviour of (initial) processes in order to get a precongruence for parallel composition and prefix:

Corollary 6.1 For Ω-free $\mathbb{P}$-processes, RT-equivalence (-inclusion) is *fully abstract* w.r.t. DL-equivalence (-inclusion) and parallel composition and prefixing, i.e. it gives the coarsest (pre)congruence for these operators that respects DL-equivalence (-inclusion). For process terms, i.e. on $\tilde{\mathbb{P}}$, $\leq_r$ is a precongruence for these operators, hiding and relabelling.

Proof: Theorem 5.6, Theorem 5.8, Theorem 5.10 and Theorem 5.3 show that RT-equivalence on $\mathbb{P}$ is a congruence and RT-inclusion is a precongruence for parallel composition, prefixing, hiding and relabelling of processes that respects DL-equivalence, -inclusion resp. By Definition 5.4, the result for RT-inclusion carries over to process terms related by $\leq_r$.

If for Ω-free processes P_1, P_2 we have $\neg\ \mathsf{RT}(P_1) \subseteq \mathsf{RT}(P_2)$, then the proof of Theorem 5.13 exhibits a test process O such that $\neg\ \mathsf{DL}(\tau.P_1 \| O) \subseteq \mathsf{DL}(\tau.P_2 \| O)$. (If P_1 or P_2 contains the special action ω, then its rôle in O must be played by some other action $a \notin \ell(P_1) \cup \ell(P_2)$; consider $\mathsf{DL}(\tau.P_i \|_{\mathbb{A}-\{a\}} O)$ in this case.). Hence, RT-equivalence (-inclusion) is the coarsest

relation that refines DL-equivalence (-inclusion) to a (pre)congruence for parallel composition and prefixing. □

As usual, the testing preorder alone is not a precongruence for choice: e.g. for the three processes $\underline{a}$, $\underline{\tau}.\underline{a}$ and $\underline{a}+\tau.\underline{a}$, we have $\underline{a} =_r \underline{\tau}.\underline{a} =_r \underline{a}+\tau.\underline{a}$, but: $\emptyset\emptyset b$ is in $\mathsf{RT}(\underline{a}+b)$, but not in $\mathsf{RT}(\underline{\tau}.\underline{a}+b)$ or $\mathsf{RT}(\underline{a}+\tau.\underline{a}+b)$; $\{b\}\{b\}$ is in $\mathsf{RT}(\underline{\tau}.\underline{a}+b)$ and $\mathsf{RT}(\underline{a}+\tau.\underline{a}+b)$, but not in $\mathsf{RT}(\underline{a}+b)$; we conclude that, in a precongruence, $\underline{a}$ cannot be in any order with the other two processes, which can do a τ, i.e. are *instable*. Hence, stable and instable processes will be incomparable, which is a little surprising, since we are only dealing with a preorder. But note that Ω should be a least element.

Furthermore, $\emptyset b \in \mathsf{RT}(\underline{a}+\tau.\underline{a}+b) \setminus \mathsf{RT}(\underline{\tau}.\underline{a}+b)$, hence $\underline{\tau}.\underline{a}$ can only be smaller than $\underline{a}+\tau.\underline{a}$ due to the urgent τ.

As another example, consider $P \equiv a+\mu x.(\tau.x+a)$ and $Q \equiv \underline{a}+\mu x.(\tau.x+a)$, which have the same refusal traces, are instable and cannot do an urgent τ; we have $\mathbb{A}b \in \mathsf{RT}(P+b) \setminus \mathsf{RT}(Q+b)$. This example shows that we need some information about behaviour unobservably starting with τ. Recall that in the case of bisimulation complete information of this kind is needed. In our setting, we only need very limited 'negative' information, namely which maximal refusal set is initially possible without a preceding τ – this is $\mathbb{A}$ for P, but not for Q in our example. On the 'positive' side, we need more, but again not complete information:

$a\|_\emptyset\tau$ and $\tau.\underline{a}+a$ are RT-equivalent with the same initial refusal set, and are both instable and cannot do an urgent τ; but $\mathbb{A}a \in \mathsf{RT}(a\|_\emptyset\tau+\underline{\tau}) \setminus \mathsf{RT}(\tau.\underline{a}+a+\underline{\tau})$; this indicates that we need to know which refusal traces beginning with a refusal set unobservably start with τ. We add this 'positive' information to RT, and also add τ to indicate that a process is instable.

Definition 6.2 A process $P \in \mathbb{P}$ is *stable*, if $\tau \notin \mathcal{A}(P)$, *instable* otherwise.

$\tau\mathsf{RT}(P) = \mathsf{RT}(P) \cup \{\tau w \mid w \text{ does not start with an action and } \exists P' :\ P \xrightarrow{\tau} P' \stackrel{w}{\Longrightarrow}_r\}$.

The (maximal) *initial refusal set* $\mathsf{IR}(P)$ is the maximal X with $P \xrightarrow{X}_r$; if $\neg P \xrightarrow{\emptyset}_r$, then we put $\mathsf{IR}(P) = \bot$ and define $\bot \subseteq X$ for all $X \subseteq \mathcal{A}$. (Uniqueness, i.e. well-definedness of $\mathsf{IR}(P)$ follows by an easy structural induction.)

To prepare the definition of our precongruence, we give a lemma relating guardedness and stability to the τRT-semantics. Recall that $P \in \mathbb{P}$ is not guarded iff there is an unguarded Ω as in Ω or $\Omega + a$. Recall further that such processes cannot perform any action or time step; in particular, they are stable.

Lemma 6.3 Let $P \in \mathbb{P}$.

1. $\tau \notin \tau\mathsf{RT}(P)$ iff P is stable iff $\tau\mathsf{RT}(P) = \mathsf{RT}(P)$.
2. P is not guarded iff $\tau\mathsf{RT}(P) = \{\lambda\}$ iff $\mathsf{RT}_\tau(P) = \{\lambda\}$; in this case, $\mathsf{IR}(P) = \bot$.
3. If P is guarded and $\mathsf{IR}(P) = \bot$, then P is instable and $\neg P \xrightarrow{X}_r$ for all $X \subseteq \mathbb{A}$.

Proof: 1. For the first 'iff', consider $w = \lambda$ in Definition 6.2.

2. P not guarded implies $\mathsf{RT}_\tau(P) = \{\lambda\}$ by 5.2.6, which implies $\tau\mathsf{RT}(P) = \{\lambda\}$. The latter implies that either P is not guarded, which implies the equivalences, or otherwise P is guarded and stable by 1. In the latter case, we have $\mathcal{R}(\tau, P) = 1$ by Proposition 2.9, hence $P \xrightarrow{\emptyset}_r$ by 5.2.4, which shows $\emptyset \in \tau\mathsf{RT}(P)$ – a contradiction. The rest is clear.

3. We have just seen that, in case P is guarded and stable, we would have $P \xrightarrow{\emptyset}_r$, a contradiction. The rest follows from 5.2.1. □

We now define our precongruence based on the pair $(\tau\mathsf{RT}(P), \mathsf{IR}(P))$. The examples above have shown that neither $\mathsf{IR}(P)$ nor the additions to $\mathsf{RT}(P)$ are superfluous. (Note that, in particular, P and Q above satisfy $\tau\mathsf{RT}(P) = \tau\mathsf{RT}(Q)$.) We will also show a coarsest precongruence result below – but still, there is some redundancy. Firstly, we can by 6.3 determine from $\tau\mathsf{RT}(P)$ whether P is not guarded, in which case $\mathsf{IR}(P) = \bot$; secondly, we can also determine from $\tau\mathsf{RT}(P)$ whether P is stable and guarded, in which case $\mathsf{IR}(P)$ is the maximal $X \subseteq \mathbb{A}$ with $X \in \mathsf{RT}(P)$. Thirdly, if P is guarded and $\mathsf{IR}(P) = \bot$, then $\neg P \xrightarrow{X}_r$ for any X; thus, if w starts with a set, then $\tau w \in \tau\mathsf{RT}(P)$ iff $w \in \mathsf{RT}(P)$, and these additional τw in $\tau\mathsf{RT}(P)$ can be determined from $\mathsf{RT}(P)$. There might be more redundancies.

The following definition of our precongruence refers to stability and guardedness, which as just noted can be derived from the $\tau\mathsf{RT}$-semantics.

Definition 6.4 For $P, Q \in \mathbb{P}$, we write $P \leq Q$ if

1. $\tau\mathsf{RT}(P) \subseteq \tau\mathsf{RT}(Q)$ (hence $\mathsf{RT}(P) \subseteq \mathsf{RT}(Q)$)
2. If P is stable and guarded, then Q is stable.
3. $\mathsf{IR}(P) \subseteq \mathsf{IR}(Q)$

The *efficiency precongruence* $\leq$ is extended to $\tilde{\mathbb{P}}$ via substitution as above and usual. We write $P = Q$ (*efficiency congruence*) if $P \leq Q$ and $Q \leq P$.

We write $P \leq_\tau Q$ if $\mathsf{RT}_\tau(P) \subseteq \mathsf{RT}_\tau(Q)$ and we write $P =_\tau Q$ if $P \leq_\tau Q$ and $Q \leq_\tau P$.

We note some properties, in particular that the unguarded processes are efficiency congruent and minimal w.r.t. $\leq$, and that $\leq$ indeed refines RT-inclusion.

Proposition 6.5
1. Let $P, Q \in \mathbb{P}$; if $P \leq Q$ and P is guarded, then Q is guarded as well and P stable iff Q stable.
2. Let $P, Q \in \mathbb{P}$; if P is not guarded, then $P \leq Q$.
3. Let $P, Q \in \tilde{\mathbb{P}}$. Then $P \leq_\tau Q$ implies $P \leq Q$, and $P \leq Q$ implies $P \leq_r Q$, but none of the reverse implications holds – not even on the set of Ω-free $\mathbb{P}_1$-processes.

Proof: 1. Q is guarded by 6.4.1 and 6.3.2; the iff-statement follows from 6.4.2 and .1, using 6.3.1.

2. $\tau\mathsf{RT}(P) = \{\lambda\}$ is minimal, as is $\mathsf{IR}(P) = \bot$, and 6.4.2 holds vacuously.

3. It suffices to prove the implications for $P, Q \in \mathbb{P}$, where the second holds directly by Definition 6.4.

Let $P \leq_\tau Q$, i.e. $\mathsf{RT}_\tau(P) \subseteq \mathsf{RT}_\tau(Q)$, hence $\tau\mathsf{RT}(P) \subseteq \tau\mathsf{RT}(Q)$. If P is stable and guarded, then $\tau \notin \mathcal{A}(P)$, hence $\mathcal{R}(\tau, P) = 1$ by Proposition 2.9.1, thus $P \xrightarrow{\emptyset}_r P'$ for some P' by Proposition 5.2.4 and Lemma 2.7.1. Furthermore, $\tau \notin \mathcal{A}(P')$ by Lemma 2.7.4 and $\mathcal{R}(\tau, P') = 1$ by Proposition 2.9.1 again, hence $P' \xrightarrow{\emptyset}_r$ again. Now by assumption also $Q \xrightarrow{\emptyset}_r Q' \xrightarrow{\emptyset}_r$ for some Q', hence $\mathcal{R}(\tau, Q) = \mathcal{R}(\tau, Q') = 1$ by Proposition 5.2.4, thus $\tau \notin \mathcal{A}(Q)$ by Proposition 5.2.4 and Lemma 2.10. We conclude that Q is stable. Finally, $\mathsf{IR}(P) \in \tau\mathsf{RT}(P) \subseteq \tau\mathsf{RT}(Q)$, thus $\mathsf{IR}(P) \subseteq \mathsf{IR}(Q)$.

For the reverse implications consider $a.\tau \leq a$ and $a \leq_r \tau.a$. □

Remark 6.6 Let $P, Q \in \mathbb{P}$. Assume that P and Q are guarded and either both stable or both not stable with $\mathcal{R}(\tau, P) = 0$. Then $P \leq Q$ if and only if $P \leq_r Q$.

Theorem 6.7 (*Refusal traces of a sum*) Let $P \equiv \sum_{i \in I} P_i \in \mathbb{P}$.

Put $\tau\mathsf{RT}^{\cap}(P) = \{X_1 \ldots X_n w \in \bigcup_{i \in I} \tau\mathsf{RT}(P_i) \mid n \geq 0, X_i \subseteq \mathbb{A}, w$ does not start with a set and $X_1 \ldots X_n \in \bigcap_{i \in I} \tau\mathsf{RT}(P_i)\}$, and define $\mathsf{RT}_\tau^{\cap}(P)$ analogously, using RT_τ instead of $\tau\mathsf{RT}$.

Put $\tau\mathsf{RT}^{\alpha}(P) = \{w \mid w \in \tau\mathsf{RT}(P_i)$ does not start with a set or $\tau w \in \tau\mathsf{RT}(P_i)$ for some $i \in I\}$.

Then: $\mathsf{IR}(P) = \bigcap_{i \in I} \mathsf{IR}(P_i)$ (which is $\perp$ if some $\mathsf{IR}(P_i) = \perp$).

If some P_i is not guarded, then neither is P, i.e. $\tau\mathsf{RT}(P) = \mathsf{RT}_\tau(P) = \{\lambda\}$ and $\mathsf{IR}(P) = \perp$.

Now assume that all P_i are guarded; let $I = S \dot\cup T \dot\cup U$ such that $i \in S$ iff P_i stable and $i \in U$ iff $\mathsf{IR}(P_i) = \perp$.

1. $\mathsf{RT}_\tau(P) = \mathsf{RT}_\tau^{\cap}(P)$
2. If $S = I$, then $\tau\mathsf{RT}(P) = \tau\mathsf{RT}^{\cap}(P)$.
3. If $S \neq I$ and $U = \emptyset$, then $\tau\mathsf{RT}(P) = \tau\mathsf{RT}^{\alpha}(P) \cup \{Xw \mid X \subseteq \mathsf{IR}(P)$ and $Xw \in \tau\mathsf{RT}(P_i)$ such that w does not start with a set or $i \in T\}$.
4. If $U \neq \emptyset$, then $\tau\mathsf{RT}(P) = \tau\mathsf{RT}^{\alpha}(P)$.

Proof: The equation for $\mathsf{IR}(P)$ follows from the definitions, as does the unguarded case with the above. 1. and 2. follow from the operational semantics, where for 2. stability implies that the τ-refusal-trace underlying some $X_1 \ldots X_n w \in \tau\mathsf{RT}^{\cap}(P)$ does not contain a τ up to the first a in w (if it exists).

In cases 3. and 4., note that $i \in U$ implies by Lemma 6.3.3 that P_i is instable and $\neg P_i \xrightarrow{X}_r$ for all $X \subseteq \mathbb{A}$; in particular, I is indeed a disjoint union of S, T and U.

We first consider $\tau\mathsf{RT}$-behaviour w where the underlying τ-refusal-trace starts with a set; since there is some instable P_i, from the operational semantics for P we see that the P_i together might refuse at most one set X; this can only happen in case 3., where X can be any subset of $\mathsf{IR}(P)$, and the succeeding behaviour w (if any) belongs to some P_i; this P_i must do an action after X, thus w can only start with a visible action for $i \in S$. For $i \in T$, on the other hand, we know that P_i can do an urgent τ after X, hence cannot directly refuse a set; thus, any $Xw \in \tau\mathsf{RT}(P_i)$ contributes to $\tau\mathsf{RT}(P)$, since P_i certainly performs τ before w, should w start with a set.

It remains to consider for cases 3. and 4. $\tau\mathsf{RT}$-behaviour w of some P_i, where the underlying τ-refusal-trace starts with an action; this is the case if and only if w starts with an action (possibly τ) or $\tau w \in \tau\mathsf{RT}(P_i)$. □

Theorem 6.8 1. On $\tilde{\mathbb{P}}$, $\leq_\tau$ and $\leq$ are precongruences for choice.

2. On Ω-free $\tilde{\mathbb{P}}$-terms, $\leq$ is fully abstract w.r.t. $\leq_r$ and choice.

Proof: Since substitution distributes over $+$, it suffices to consider $\mathbb{P}$, its Ω-free part resp.

1. The unguarded case is clear, using 6.5.2 for $\leq$; consider guarded processes only. Precongruence for $\leq_\tau$ is clear from 6.7.1, since $\mathsf{RT}_\tau^{\cap}(P)$ is monotonic in its arguments $\mathsf{RT}_\tau(P_i)$.

For $\leq$, take $P \leq Q$ and some R. Computation of $\mathsf{IR}(P+R)$ is monotonic in $\mathsf{IR}(P)$, hence $\mathsf{IR}(P+R) \subseteq \mathsf{IR}(Q+R)$, i.e. 6.4.3 holds. P stable iff Q stable by 6.5.1, hence $P+R$ stable

iff $Q+R$ stable, i.e. 6.4.2 holds; and in this case, $\tau\mathsf{RT}(P+R)$ and $\tau\mathsf{RT}(Q+R)$ can be determined with the monotonic $\tau\mathsf{RT}^{\cap}(P)$ according to 6.7.2, hence 6.4.1 holds.

Thus, assume P and Q to be instable; we note that $\mathsf{IR}(P) \neq \bot$ implies $\mathsf{IR}(Q) \neq \bot$, i.e. if P belongs to T in 6.7, then so does Q. We conclude that $\tau\mathsf{RT}(P+R)$ and $\tau\mathsf{RT}(Q+R)$ can be determined with the same monotonic construction according to 6.7.3 or .4 – unless $\mathsf{IR}(P) = \bot \neq \mathsf{IR}(Q)$, in which case $\tau\mathsf{RT}(Q+R)$ might have additional elements as described in 6.7.3 in contrast to 6.7.4. In any case, 6.4.1 holds; thus, $P+R \leq Q+R$.

2. Take Ω-free $P, Q \in \mathbb{P}$ with $\neg\, P \leq Q$; we have to show that $\neg\, P+R \leq_r Q+R$ for some Ω-free $R \in \mathbb{P}$. Let $c \in \mathbb{A} \setminus (\ell(P) \cup \ell(Q))$.

If $X = \mathsf{IR}(P) \subseteq \mathsf{IR}(Q)$ fails (in particular, $X \neq \bot$), then $Xc \in \mathsf{RT}(P+c) \setminus \mathsf{RT}(Q+c)$.

If P is stable, then $P \xrightarrow{\emptyset}_r \xrightarrow{\emptyset}_r$ and $\emptyset\emptyset c \in \mathsf{RT}(P+c)$; should Q be instable, then $\emptyset\emptyset c \notin \tau\mathsf{RT}(Q+c) \supseteq RT(Q+c)$ by 6.7. For the remaining part we need the following. If P is instable (i.e. $\tau \in \tau\mathsf{RT}(P)$), then P can do τ and refuse $\{c\}$ (observe 5.12.1 and 5.2.3), hence $\{c\} \in \mathsf{RT}(P+\underline{c})$; should Q be stable, then $\{c\} \notin RT(Q+\underline{c})$. [2]

If $w \in \mathsf{RT}(P) \setminus \mathsf{RT}(Q)$, then choose $R \equiv \mathbf{0}$ with 5.11. Thus, it remains to consider $\tau w \in \tau\mathsf{RT}(P) \setminus \tau\mathsf{RT}(Q)$ where by the above $w \neq \lambda$, i.e. w starts with a set; by 5.2.1 and .3, we can assume that this set contains c. In this case, $w \in \mathsf{RT}(P+\underline{\tau}.\underline{c}) \setminus \mathsf{RT}(Q+\underline{\tau}.\underline{c})$, and we are done. □

Theorem 6.9 1. On $\tilde{\mathbb{P}}$, $\leq_\tau$ and $\leq$ are precongruences for parallel composition, prefixing, hiding and relabelling; this holds analogously for $=$.

2. On the Ω-free part of $\tilde{\mathbb{P}}$, $\leq$ is fully abstract w.r.t DL-inclusion and parallel composition, prefixing and choice.

Proof: As in the last proof, it suffices to consider $\mathbb{P}$, its Ω-free part resp.

1. By Theorems 5.6, 5.8 and 5.10, $\leq_\tau$ is a precongruence as desired, and so is $\leq_r$. The constructions easily carry over to show that $\tau\mathsf{RT}$-inclusion is such a precongruence as well. Stability and the IR-set of $\alpha.P$ depend on α only, thus, $\leq$ is a precongruence for prefixing. In the unguarded case, 6.4.2 and .3 are easy also for the other operators.

(In)stability of a parallel composition depends on the (in)stability (and (un)guardedness) of its operands, and $\mathsf{IR}(P\|_A Q)$ is $\bot$ if $\mathsf{IR}(P) = \bot$ or $\mathsf{IR}(Q) = \bot$ and $\mathsf{IR}(P)\|_A\mathsf{IR}(Q))$ (with $\mathsf{IR}(P)$ and $\mathsf{IR}(Q)$ considered as strings of length 1) otherwise. Thus, $\leq$ is a precongruence for parallel composition.

Finally, consider a general relabelling function Φ and guarded P and Q with $P \leq Q$. Since $\mathsf{IR}(P[\Phi])$ is the maximal X with $X[\Phi]^{-1} \subseteq \mathsf{IR}(P)$ (possibly $\bot$), $P[\Phi]$ and $Q[\Phi]$ satisfy 6.4.3. Now assume $P[\Phi]$ and thus P (by $\Phi(\tau) = \tau$) and Q (by 6.4.2) to be stable. We will show $\mathcal{A}(Q) \subseteq \mathcal{A}(P)$ and conclude that $Q[\Phi]$ is stable and $\leq$ a precongruence for relabelling (and hiding).

Let $\tau \neq a \notin \mathcal{A}(P)$; by 2.9.1 and 5.2.4, $P \xrightarrow{\{a\}}_r P'$ for some guarded P' with $\mathcal{A}(P') = \mathcal{A}(P)$ by Lemma 2.7.4. Thus, $P' \xrightarrow{\{a\}}_r$ and $\{a\}\{a\} \in \tau\mathsf{RT}(P) \subseteq \tau\mathsf{RT}(Q)$. Since Q is stable, this shows $Q \xrightarrow{\{a\}}_r Q' \xrightarrow{\{a\}}_r$ for some Q' with $\mathcal{R}(a, Q') = 1$ by 5.2.4, and this gives $a \notin \mathcal{A}(Q)$ by 5.2.4 and 2.10; hence $\mathcal{A}(Q) \subseteq \mathcal{A}(P)$.

2. follows from 1., 6.8 and 6.1. □

[2] For $P \in \mathbb{P}$ containing Ω, we would have a problem in this part: $\neg\, \underline{\tau}.\Omega \leq \mathbf{0}$, but $\mathsf{RT}(\underline{\tau}.\Omega + R) \subseteq \mathsf{RT}(R) = \mathsf{RT}(\mathbf{0}+R)$ for all R.

We finally aim to show that $\leq_\tau$ and $\leq$ are also precongruences for (guarded) recursion. A sort of iteration of processes is needed.

Definition 6.10 Let $P \in \tilde{\mathbb{P}}$ be such that at most $x \in \mathcal{X}$ is free in P, and assume x to be guarded in P. Define, for each $n \in \mathbb{N}$, $P^1 \equiv P$ and $P^{n+1} \equiv P\{P^n/x\}$.

The following proposition states a basic property of recursive processes; namely, that any $\mu x.P$ process is $=$- or $=_\tau$-related to its unfolding.

Proposition 6.11 Assume that at most $x \in \mathcal{X}$ is free in $P \in \tilde{\mathbb{P}}$ and that x is guarded in P. Then:

1. $\mu x.P =_\tau P\{\mu x.P/x\}$;
2. $\mu x.P = P\{\mu x.P/x\}$.

Proof: We just have to prove 1., since this implies 2. by Proposition 6.5.3. In this proof let $\varepsilon \in (\mathbb{A}_\tau \cup 2^{\mathbb{A}_\tau})$ and $v \in (\mathbb{A}_\tau \cup 2^{\mathbb{A}_\tau})^*$.

By rule Rec_a or rule Rec_r we have $\mu x.P \xrightarrow{\varepsilon}_r R \xrightarrow{v}_r$ iff $P \xrightarrow{\varepsilon}_r Q$ for some Q such that $R \equiv Q\{\mu x.P/x\} \xrightarrow{v}_r$ iff $P\{\mu x.P/x\} \xrightarrow{\varepsilon}_r Q\{\mu x.P/x\} \xrightarrow{v}_r$ by Proposition 5.2.5 since x guarded in P. □

Lemma 6.12 Assume that at most $x \in \mathcal{X}$ is free in $P \in \tilde{\mathbb{P}}$ and in $Q \in \tilde{\mathbb{P}}$ and that x is guarded in P and Q. Let $R, S \in \mathbb{P}$ and consider a sequence w such that $|w| = n$. Then:

1. $Q\{P^n/x\}\{R/x\} \xrightarrow{w}_r$ if and only if $Q\{P^n/x\}\{S/x\} \xrightarrow{w}_r$;
2. $P^{n+1}\{R/x\} \xrightarrow{w}_r$ if and only if $P^{n+1}\{S/x\} \xrightarrow{w}_r$.

Proof: Prove Item 1 by induction on n. Assume $n = 1$ and $w = \varepsilon \in (\mathbb{A}_\tau \cup 2^{\mathbb{A}_\tau})$. Then $Q\{P/x\}\{R/x\} \xrightarrow{\varepsilon}_r T$ if and only if $Q \xrightarrow{\varepsilon}_r Q'$ and $T \equiv Q'\{P/x\}\{R/x\}$ if and only if $Q \xrightarrow{\varepsilon}_r Q'$ and $T' \equiv Q'\{P/x\}\{S/x\}$ if and only if $Q\{P/x\}\{S/x\} \xrightarrow{\varepsilon}_r T'$.

Now let $|w| = n + 1$ and $w = \varepsilon v$ with $|v| = n$. Assume the statement for sequences of length less or equal than n. Similarly to the base case we have:

$Q\{P^{n+1}/x\}\{R/x\} \xrightarrow{\varepsilon}_r T$ if and only if $Q \xrightarrow{\varepsilon}_r Q'$ and $T \equiv Q'\{P^{n+1}/x\}\{R/x\}$ if and only if $Q \xrightarrow{\varepsilon}_r Q'$ and $T' \equiv Q'\{P^{n+1}/x\}\{S/x\}$ if and only if $Q\{P^{n+1}/x\}\{S/x\} \xrightarrow{\varepsilon}_r T'$.

Now $Q'\{P^{n+1}/x\}\{R/x\} \equiv Q'\{P\{P^n/x\}/x\}\{R/x\} \equiv Q'\{P/x\}\{P^n/x\}\{R/x\}$ and x is guarded in $Q'\{P/x\}$. By induction hypothesis $(Q'\{P/x\})\{P^n/x\}\{R/x\} \xrightarrow{v}_r$ if and only if $(Q'\{P/x\})\{P^n/x\}\{S/x\} \xrightarrow{v}_r$ and, hence, we have $Q'\{P^{n+1}/x\}\{R/x\} \xrightarrow{v}_r$ if and only if $Q'\{P^{n+1}/x\}\{S/x\} \xrightarrow{v}_r$. Thus the main statement holds.

Item 2 immediately follows from Item 1 by taking $Q \equiv P$.

□

The following sort of depth over $\tilde{\mathbb{P}}$-processes is now needed.

Definition 6.13 The *depth* of a process $P \in \tilde{\mathbb{P}}$, $d(P)$, is defined by:

Nil, Stop, Var:	$d(\mathbf{0}) = d(\Omega) = d(x) = 0$
Pref, Rel:	$d(\langle \alpha, r \rangle.P) = d(P[\Phi]) = d(P)$, $r \in \{0, 1\}$
Sum, Par:	$d(P_1 + P_2) = d(P_1 \|_A P_2) = \max(d(P_1), d(P_2))$
Rec:	$d(\mu x.P) = \begin{cases} 0 & \text{if } \mu x.P \text{ is closed} \\ d(P) + 1 & \text{otherwise} \end{cases}$

Example 6.14 An interesting example for depth d is

$$d(\mu y.(\mu x.(a.x + b.y)) + c.z) = 2$$

Lemma 6.15 Let P and R be $\tilde{\mathbb{P}}$ processes. Then:

1. P closed implies $d(P) = 0$;
2. R closed implies $d(P\{R/x\}) \leq d(P)$.

Proof: Item 1 follows by simple inductive reasoning, so let us concentrate on Item 2. The proof proceeds by induction on the structure of P.

Nil,Stop: If $P \equiv \mathbf{0}$ then $P\{R/x\} \equiv \mathbf{0}$ and hence $d(P\{R/x\}) = d(P) = 0$; $P \equiv \Omega$ is analogous.

Var: $P \equiv y$. We have two cases to distinguish: $y = x$ and $y \neq x$. In the former case, $P\{R/x\} \equiv R$ and, since R is closed, we have $d(P) = d(R) = 0$ by 1. In the latter one, $y \neq x$, we have $d(P\{R/x\}) = d(P)$.

Pref: $P \equiv \langle \alpha, r \rangle.P_1$. In such a case we have $(\langle \alpha, r \rangle.P_1)\{R/x\} \equiv \langle \alpha, r \rangle.(P_1\{R/x\})$. By induction hypothesis $d(P_1\{R/x\}) \leq d(P_1)$. Thus, $d((\langle \alpha, r \rangle.P_1)\{R/x\}) = d(P_1\{R/x\}) \leq d(P_1) = d(\langle \alpha, r \rangle.P_1)$.

Sum: $P \equiv P_1 + P_2$. By induction hypothesis $d(P_1\{R/x\}) \leq d(P_1)$ and $d(P_2\{R/x\}) \leq d(P_2)$. Thus also $d((P_1 + P_2)\{R/x\}) = d((P_1\{R/x\}) + (P_2\{R/x\})) =$
$\max(d(P_1\{R/x\}), d(P_2\{R/x\})) \leq \max(d(P_1), d(P_2)) = d(P)$.

Par, Rel: analogously to Sum and Pref.

Rec: $P \equiv \mu y.P_1$. We distinguish two cases: $y = x$ and $y \neq x$. The former case is simple since $(\mu x.P_1)\{R/x\} \equiv \mu x.P_1$ and hence $d((\mu x.P_1)\{R/x\}) = d(\mu x.P_1)$. Assume $y \neq x$. If $(\mu y.P_1)\{R/x\}$ is closed then $d((\mu y.P_1)\{R/x\}) = 0 \leq d(\mu y.P_1)$. Otherwise, assume $d(P_1\{R/x\}) \leq d(P_1)$ by induction hypothesis. Hence, $d((\mu y.P_1)\{R/x\}) = d(\mu y.(P_1\{R/x\}) = d(P_1\{R/x\}) + 1 \leq d(P_1) + 1 = d(\mu y.P_1)$, since $\mu y.P_1$ cannot be closed if $(\mu y.P_1)\{R/x\}$ is not.

□

The following theorem states the precongruence of our preorders for recursion.

Theorem 6.16 Let $P, Q \in \tilde{\mathbb{P}}$ and $x \in \mathcal{X}$ be guarded in P and Q. Then:

1. $P \leq_\tau Q$ implies $\mu x.P \leq_\tau \mu x.Q$;
2. $P \leq Q$ implies $\mu x.P \leq \mu x.Q$.

Proof: By well-founded induction on $d = \max(d(P), d(Q))$. One can prove Item 1 first and then Item 2, but the proofs are so similar that we do them together. Actually, we will only show the most involved parts of the two statements.

We have to show that for every closed substitution $\mathcal{S}$ that is defined for all free variables in P and Q except x, we have $[\mu x.P]_\mathcal{S} \leq_\tau [\mu x.Q]_\mathcal{S}$ or $[\mu x.P]_\mathcal{S} \leq [\mu x.Q]_\mathcal{S}$, i.e. $\mu x.([P]_\mathcal{S}) \leq_\tau \mu x.([Q]_\mathcal{S})$ or $\mu x.([P]_\mathcal{S}) \leq \mu x.([Q]_\mathcal{S})$.

By hypothesis, we have $[P]_\mathcal{S} \leq_\tau [Q]_\mathcal{S}$ or $[P]_\mathcal{S} \leq [Q]_\mathcal{S}$ and by Lemma 6.15 we have $d([P]_\mathcal{S}\}) \leq d(P)$ and $d([Q]_\mathcal{S}\}) \leq d(Q)$.

If $\max(d([P]_\mathcal{S}), d([Q]_\mathcal{S})) < d$, then the statement immediately follows by induction. Otherwise, $\max(d([P]_\mathcal{S}), d([Q]_\mathcal{S})) = d$ by Lemma 6.15, and we can concentrate on processes P and Q that have at most x as free variable.

Thus, assume at most x free in P and Q and x guarded in P and Q. We need three preliminary results:

Statement A: Let $T \in \tilde{\mathbb{P}}$, $R, S \in \mathbb{P}$ be such that $d(T) \leq d$. Then:

(a) $R \leq_\tau S$ implies $T\{R/x\} \leq_\tau T\{S/x\}$;

(b) $R \leq S$ implies $T\{R/x\} \leq T\{S/x\}$.

Proof: Items (a) and (b) have similar proofs, thus we concentrate on the first one. The proof proceeds by structural induction on T.

Nil,Stop,Var: For $T = \mathbf{0}$, we have $T\{R/x\} \equiv T\{S/x\} \equiv \mathbf{0}$, and similarly for $T \equiv \Omega$. If $T \equiv y$ then we have two cases: $y = x$ and $y \neq x$. The latter case is similar to the previous ones, so assume $y = x$. Then $T\{R/x\} \equiv R$ and $T\{S/x\} \equiv S$, and $T\{R/x\} \leq_\tau T\{S/x\}$ follows by hypothesis $R \leq_\tau S$.

Pref: $T = \langle \alpha, r\rangle.T_1$. Then $T\{R/x\} \equiv \langle \alpha, r\rangle.(T_1\{R/x\})$ and $T\{S/x\} \equiv \langle \alpha, r\rangle.(T_1\{S/x\})$. Since $d(T_1) = d(T) \leq d$, the statement follows by induction and Theorem 6.9.

Sum: $T \equiv T_1 + T_2$. By definition $d(T_1), d(T_2) \leq d(T) \leq d$. By structural induction $T_1\{R/x\} \leq_\tau T_1\{S/x\}$ and $T_2\{R/x\} \leq_\tau T_2\{S/x\}$. Hence, by the fact that $\leq_\tau$ is precongruence for $+$, $T\{R/x\} \equiv (T_1 + T_2)\{R/x\} \equiv (T_1\{R/x\}) + (T_2\{R/x\}) \leq_\tau (T_1\{S/x\}) + (T_2\{S/x\}) \equiv T\{S/x\}$.

Par, Rel: analogously to Sum.

Rec: $T \equiv \mu y.T_1$, and we may assume that x is free in T, hence that x is free in T_1 and $x \neq y$. Therefore, $d(T_1) = d(T) - 1 < d$ and $T_1\{R/x\} \leq_\tau T_1\{S/x\}$ by structural induction. Moreover, by Lemma 6.15.2, $d(T_1\{R/x\}), d(T_1\{S/x\}) \leq d(T_1) < d$ and thus, by outer induction (the main statement), we also have $(\mu y.T_1)\{R/x\} \equiv \mu y.(T_1\{R/x\}) \leq_\tau (\mu y.T_1)\{S/x\}$.

Statement B: For all $i \in \mathbb{N}$,

(a) $P^i\{\Omega/x\} \leq_\tau \mu x.Q$;

(b) $P^i\{\Omega/x\} \leq \mu x.Q$.

Proof: Again concentrate on Item (a), since (b) is similar. The proof is by well-founded induction on $i \geq 1$, and we denote Ω with P^0. For arbitrary $i \geq 1$, we have $P^i\{\Omega/x\} \equiv P\{P^{i-1}/x\}\{\Omega/x\} \equiv P\{P^{i-1}\{\Omega/x\}/x\} \leq_\tau Q\{P^{i-1}\{\Omega/x\}/x\}$, since by hypothesis $P \leq_\tau Q$. By induction hypothesis or by $P^0\{\Omega/x\} \equiv \Omega \leq_\tau \mu x.Q$, we have $P^{i-1}\{\Omega/x\} \leq_\tau \mu x.Q$, and since $P^{i-1}\{\Omega/x\}$ and $\mu x.Q$ are closed, *Statement A* implies $Q\{P^{i-1}\{\Omega/x\}/x\} \leq_\tau Q\{\mu x.Q/x\}$. Finally, $Q\{\mu x.Q/x\} =_\tau \mu x.Q$ by Proposition 6.11.

Statement C: For all $i \in \mathbb{N}$,

(a) $\mu x.P \leq_\tau P^i\{\mu x.P/x\}$;

(b) $\mu x.P \leq P^i\{\mu x.P/x\}$.

Proof: Again, we only prove (a) by induction on i. Case $i = 1$ follows by Proposition 6.11, hence consider $i + 1$. By Proposition 6.11, *Statement A* and the induction hypothesis, we get $\mu x.P \leq_\tau P\{\mu x.P/x\} \leq_\tau P\{P^i\{\mu x.P/x\}/x\}$, and finally $P\{P^i\{\mu x.P/x\}/x\} \equiv P^{i+1}\{\mu x.P/x\}$.

Let us now come back to the main theorem. We prove Item 2, since 1 is easier. Since x is guarded in P and Q and at most x is free in P and Q, all variables are guarded in $\mu x.P$ and $\mu x.Q$. If there is an unguarded Ω in Q or equivalently in $\mu x.Q$, then there is one in P or equivalently in $\mu x.P$ by Proposition 6.5.1, and in the latter case we are done by Proposition 6.5.2. Hence, we may assume $\mu x.P$ and $\mu x.Q$ to be guarded.

We have to prove that, if $\mu x.P$ is stable, then $\mu x.Q$ is stable. Assume $\mu x.P$ stable (and guarded). Then $\mu x.P \stackrel{\tau}{\not\rightarrow}_r$, hence $P \stackrel{\tau}{\not\rightarrow}_r$ and (closed) $P\{\Omega/x\} \stackrel{\tau}{\not\rightarrow}_r$ by Proposition 5.2.5. Since $P \leq Q$, also $Q\{\Omega/x\} \stackrel{\tau}{\not\rightarrow}_r$, thus $Q \stackrel{\tau}{\not\rightarrow}_r$ and hence also $\mu x.Q \stackrel{\tau}{\not\rightarrow}_r$, that is $\mu x.Q$ is stable.

For $X \in \mathsf{IR}(\mu x.P)$, we have $\mu x.P \stackrel{X}{\rightarrow}_r$ if and only if $P \stackrel{X}{\rightarrow}_r$ if and only if $P\{\Omega/x\} \stackrel{X}{\rightarrow}_r$, again by Proposition 5.2.5. By $P \leq Q$ also $Q\{\Omega/x\} \stackrel{X}{\rightarrow}_r$, $Q \stackrel{X}{\rightarrow}_r$ and $\mu x.Q \stackrel{X}{\rightarrow}_r$.

Now it only remains to show that $\tau\mathsf{RT}(\mu x.P) \subseteq \tau\mathsf{RT}(\mu x.Q)$. Let $v \in \tau\mathsf{RT}(\mu x.P)$ and let w be the underlying τ-refusal trace with $|w| = n$. By *Statement C*, we have $w \in \mathsf{RT}_\tau(P^{n+1}\{\mu x.P/x\})$, hence $w \in \mathsf{RT}_\tau(P^{n+1}\{\Omega/x\})$ by Lemma 6.12 and $v \in \tau\mathsf{RT}(P^{n+1}\{\Omega/x\})$. Since $P^{n+1}\{\Omega/x\} \leq \mu x.Q$ by *Statement B*, we get $v \in \tau\mathsf{RT}(\mu x.Q)$. □

7 An Example

In this section we show an application of our theory. Consider a simple server that is able to manage at most two requests from the external environment. (Of course, the example extends to any number of requests.) The server receives requests via a channel *in* and sends replies via a channel *out*.

We first provide a purely sequential implementation of such a system and then a parallel one. We compare the efficiency of these two according to our preorder.

The first implementation *Seq* of the simple server can be described in our language by:

$$Seq \equiv \mu x.\underline{in}.(\mu y.(out.x + \underline{in}.out.y))$$

An automaton describing the refusal-based transitional semantics (Definition 5.1) of this system is given in Fig. 1, where the state inscriptions are:

$$
\begin{aligned}
S_1 &\equiv \mu y.(out.Seq + \underline{in}.out.y) \\
S_2 &\equiv out.S_1 \\
S_3 &\equiv \underline{out}.Seq + \underline{in}.out.S_1 \\
S_4 &\equiv \underline{out}.S_1
\end{aligned}
$$

In the refusal sets, we only list actions from $\{in, out\}$, since all other actions can always be refused additionally, see Proposition 5.2.3. Also, we only give maximal refusal sets, compare 5.2.1; hence, there are also e.g. transitions $\xrightarrow{\emptyset}_r$, $\xrightarrow{\{in\}}_r$ and $\xrightarrow{\{out\}}_r$ from S_2 to S_4. Furthermore, Seq has the form $\mu x.P$ and the process reached after a time step is $P\{\mu x.P/x\}$; we identify Seq with this unfolding, since the two processes are equivalent.

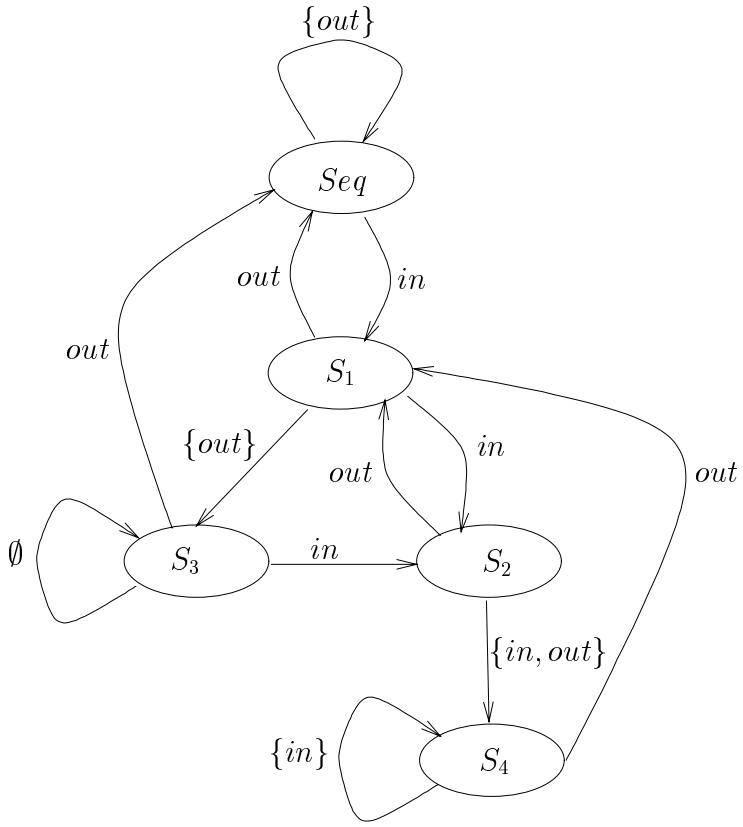


Figure 1: The (Refusal) Transitional Semantics of *Spec*

After receiving an input from the external environment, the server reaches a state in which it is able either to provide the corresponding output or to receive another input. In the former case the system reaches its initial state, in the latter one it must provide an output before continuing. This ensures that the system receives no more than two inputs and provides an output for each input.

Regarding efficiency, *Seq* is always ready to receive an input immediately if it can handle an input at all; after an input, it might take one unit of time before it can provide the respective output.

The second implementation *Par* is the parallel composition of two servers; each of them is able to manage at most one request in the same way and with the same efficiency as just described. In our language:

$$Par \equiv (\mu x.\underline{in}.out.x) \parallel_\emptyset (\mu x.\underline{in}.out.x)$$

An automaton describing the refusal-based transitional semantics of Par is given in Fig. 2, where the state inscriptions are:

$$\begin{array}{rcl} P & \equiv & \mu x.\underline{in}.out.x \\ P_1 & \equiv & out.P \\ P_2 & \equiv & \underline{out}.P \end{array}$$

In addition to the remarks given for Fig. 1, we have also used another sound equality, namely commutativity of parallel composition, in order to reduce the size of the automaton; furthermore, we have abbreviated $\parallel_\emptyset$ to $\parallel$.

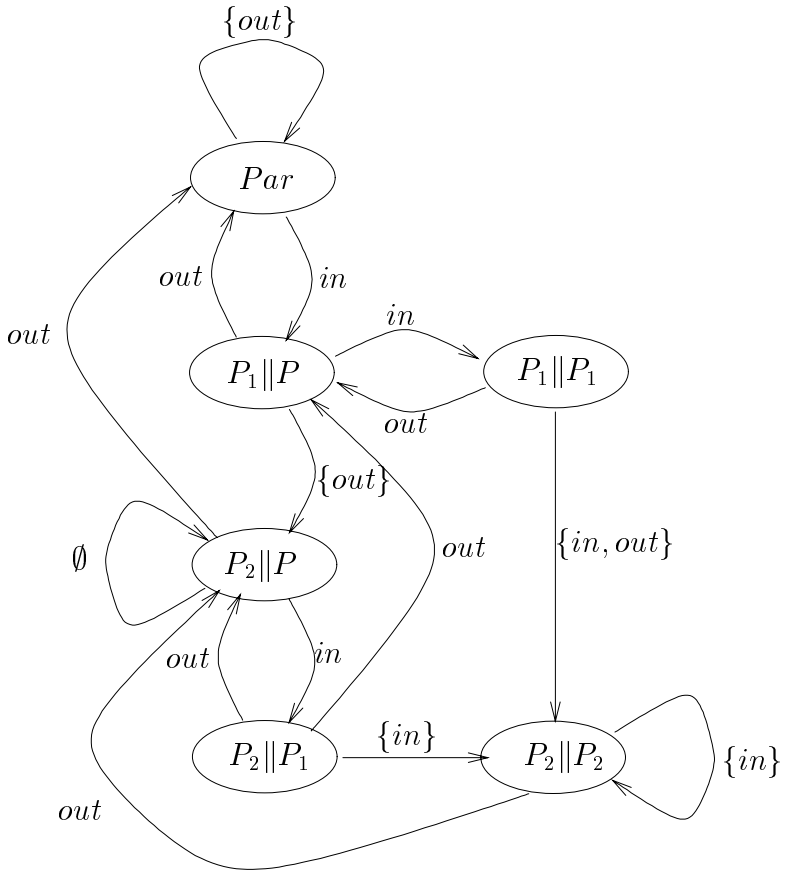


Figure 2: The (Refusal) Transitional Semantics of Par

Our main aim is to show that the parallel view of the server gives a faster implementation than the sequential one; namely, $Par \sqsupseteq Seq$. Of course, for this proof, we will exploit the alternative characterization of our faster implementation preorder in terms of refusal traces (Theorem 5.13). It is well-known that, for proving an inclusion of languages like $\mathsf{RT}(Par) \subseteq \mathsf{RT}(Seq)$, it is sufficient to find a simulation relation between the states of the respective automata, which here describe the refusal-based transitional semantics of Par and Seq. For the automata in Fig. 1 and Fig. 2 such a simulation relation is the following:

$$\begin{aligned}\Re \;=\; \{&(Par, Seq), (P_1 \parallel_\emptyset P, S_1), (P_1 \parallel_\emptyset P_1, S_2), (P_2 \parallel_\emptyset P, S_3),\\ &(P_2 \parallel_\emptyset P_1, S_2), (P_2 \parallel_\emptyset P_2, S_4), (P_2 \parallel_\emptyset P, S_1)\}\end{aligned}$$

Relation $\Re$ is a simulation relation; this means that it relates the initial states, i.e. Par and Seq, and whenever $(P, Q) \in \Re$ and $P \xrightarrow{\varepsilon}_r P'$, where $\varepsilon = \alpha \in \mathbb{A}_\tau$ or $\varepsilon = X \subseteq \mathbb{A}$, then $Q \stackrel{\varepsilon}{\Longrightarrow}_r Q'$ (or $Q \Longrightarrow_r Q'$ if $\varepsilon = \tau$) and $(P', Q') \in \Re$. The existence of the simulation $\Re$ implies that $\mathsf{RT}(Par) \subseteq \mathsf{RT}(Seq)$, i.e. $Par \sqsupseteq Seq$.

Actually, Par is strictly faster than Seq since we can find a refusal trace of Seq that is not one of Par: $in\ in\ \{out\}\ out\ \{out\}$. This is a witness of slow behaviour that Seq can show, but Par cannot: if you quickly provide Par with two inputs, you do not have to wait another unit of time for the second output. This result can be seen as a validation of our approach, since one would expect intuitively that a parallel implementation gives a speed-up.

In the rest of this section, we have a short look at two variants of Seq. Consider the following alternative system:

$$Seq_1 \equiv \mu x.(\underline{in}.out.x \;+\; \underline{in}.(\mu y.(out.x + \underline{in}.out.y)$$

This server behaves nondeterministically as a "one request" server (choosing the left in) that has to produce an output before accepting another input or as a "two requests" server (choosing the middle in) just as Seq. As above, one can prove that $Seq \sqsupseteq Seq_1$. Thus, our preorder is not only a faster-than relation, but also an implementation relation that may reduce nondeterminism e.g. in a stepwise refinement.

As another example, consider:

$$Seq_2 \equiv \mu x.in.(\mu y.(out.x + in.out.y))$$

This specification is similar to the original one, but this time the input actions are not offered immediately. Again $Seq \sqsupseteq Seq_2$, and in fact it is not hard to see that generally $\underline{\alpha}.P \leq \alpha.P$. Again, our preorder behaves as one should expect from a "faster than" preorder.

8 Conclusion and Related Work

Building on a similar approach in a Petri net setting [32, 17, 15, 33, 7], we have defined a timed process algebra PAFAS for asynchronous systems and developed a timed testing scenario to compare the worst-case performance of such systems. PAFAS is very much like an ordinary CCS-like process algebra, but actions have an upper time bound 0 or 1; by our notation, ordinary CCS-like processes (called initial here) are identified with PAFAS-processes where all these bounds are 1. ¿From experience in a Petri net setting [6], we expect that a generalization to processes where initially timer-values are arbitrary natural numbers is feasible; but this would make the present paper unduly lengthy and deviate too much from our original interest in *CCS*-like processes. Essential in the testing scenario is that each test has a duration and success only counts when it occurs within this duration; also, we only use initial processes as test processes. The testing preorder that results from must-testing in our scenario can naturally be interpreted as a faster-than relation or efficiency preorder that relates processes according to their worst-case performance.

Since we only have upper time bounds, arbitrarily many actions may happen within a given finite time; thus, one component of a system may perform arbitrarily many actions

while another performs just one action within its given time bound, hence PAFAS-processes are truly asynchronous systems. One might wonder at this stage whether we could have any problems with Zeno-runs, where infinitely many actions happen in a finite time. The technical answer is that we do not deal with infinite runs; a more convincing answer might be that, intuitively speaking, runs with 'very many' actions in a short time are irrelevant in our setting, since they do not represent some worst-case performance. We also show that the processes we are really interested in do not have time-stops in a certain sense.

The first main result is that the testing preorder based on continuous time is the same as that based on discrete time; thus, one can use the simpler discrete version to reason about the more realistic continuous version. The same result was obtained in a Petri net setting in [17] (for all time bounds being 1), but our process algebraic proof is technically quite different. A related discretization result is given in [27], which considers Petri nets where transitions are associated with intervals and each enabled transition fires after a delay in the associated interval; it is shown that the reachable markings are the same for continuous and discrete time. In the proof, each continuous firing sequence is translated to a shorter discrete one; for our result, it is important that we translate a continuous trace into a *longer* discrete one. Another, well-known discretization result can be found in [2]: the state space of a timed automaton is infinite due to continuous time, but it can be made finite by a discretization that replaces states by classes of states; this discretization has nothing to do with replacing continuous by discrete time.

The second main result (again as in [17]) is a characterization of our efficiency preorder with some sort of timed refusal traces; the construction of test processes in its proof is quite involved since, intuitively speaking, asynchronous test processes are not very good in controlling or finding out when exactly things happen in the tested system.

Since the efficiency preorder is not a precongruence for choice, we refine it to the coarsest such precongruence and call this the efficiency precongruence. We close with a small example. Work on more elaborate examples is in progress; in a Petri net setting, such examples can be found in particular in [32, 7]. In the present paper, we have mentioned some algebraic laws, and we are working on a complete axiomatization. Such an axiomatization for a fragment of the algebra of initial processes can be found in [34], but a convincing treatment of parallel composition in our setting of asynchronous processes is missing; observe that a straightforward expansion law is not valid, since e.g. $a.\mathbf{0} \parallel_\emptyset b.\mathbf{0}$ is strictly faster than $a.b.\mathbf{0} + b.a.\mathbf{0}$ due to the refusal trace $\mathbb{A}a\mathbb{A}$.

There is a large number of papers about timed process algebras. Usually, they model *synchronous* systems where components can decide what to do when according to a common global clock; in particular, *timeouts* can be modelled where, at a certain time, enabled actions are withdrawn or other actions are offered. In contrast, we have modelled *asynchronous* systems where actions are not enabled or disabled because time passes; we have added time to the process algebra simply to evaluate the *performance*, but not to change the functionality.

A common feature in timed process algebras is a *waiting construct* written *wait* t or (t) or σ that requires the process to let *exactly* time t or time 1 pass. In contrast, in PAFAS such timing information is associated with actions as in [1, 13]; also, an action a is not associated with an exact time, but with the *interval* $[0, 1]$. Timing intervals (already used in [19]) can be modelled e.g. in the algebra of [5] and are associated with actions in the algebra of [11]. In most cases, time is taken to be continuous, but sometimes also discrete time is considered,

e.g. in [14].

Although time is associated to actions, we have given an operational semantics in the most usual style, where some transitions are *time steps*, while others correspond to the *instantaneous* occurrence of actions. Naturally, a time step describes *relative* time, i.e. how much time passes since the last time step. Given a run according to this operational semantics, one can of course determine the *absolute* time of action occurrences, i.e. their time since the beginning of the run. Alternatively to our sort of semantics, one can consider *timed actions*, i.e. actions paired with the time of occurrence; [5] gives an operational semantics with time steps and with timed actions and absolute time, [1, 13] present operational semantics with only such timed actions; [29] gives a denotational semantics in this style, while the corresponding operational semantics in [30] considers time steps and timed actions with relative time. [11] gives a translation from a semantics with absolute time to an equivalent one with relative time.

Another feature common to most timed process algebras is a mechanism to enforce the timely occurrence of actions; very often, this is a combination of *patience* and *maximal progress* as e.g. in [29, 35, 14]. Patience means that components in a composition are willing to wait for a synchronization partner if necessary; in general, this is modelled by allowing all visible actions to idle since they are available for synchronization. Maximal progress means that actions are performed if waiting is certainly not necessary; in general, this applies exactly to internal actions. We have interpreted these concepts slightly differently by requiring maximal progress for all actions in stand-alone processes. In particular, this may force ω to occur and therefore, in the testing definition, success is signalled by the *performance* of ω; we regard this as more natural than the usual definition where success is signalled by reaching a state that enables ω.

Alternatives to patience and maximal progress are insistent actions and soft time. An insistent action as in [22, 5, 26] has to be performed at a certain time; if this is not possible since a synchronization partner is required and not available, then time cannot go on. This can mean that neither time steps nor actions can occur anymore. In soft time, used for multimedia systems, actions can also be performed only at a certain time, but they may also simply be left out, see e.g. [11] and the references there.

At least in some cases, timed process algebras can also be categorized as timed extensions of classical process algebras. E.g. [29] (or the more modern version [28]) is an extension of TCSP and, true to that approach, it gives a denotational semantics in the failures style. This is in fact a refusal trace semantics, in spirit very similar to ours; presumably due to the use of continuous time, the actions of a run and the refusal information are given separately. Also, a so-called stability time is given for each run, and it is assumed that a component has to wait some time δ between any two actions, while actions from different components can occur at the same time. Both these features are missing in the corresponding operational semantics of [30]. In particular for our discretization result, it is very important that we do not have such a δ and are not restricted in performing actions at the same time.

A timed extension of ACP is given in [5]; an operational semantics (where no two actions can occur at the same time) and bisimulation is considered, but the stress lies on studying suitable axioms. Timed extensions of CCS can be found in [22, 35] and the stress lies on studying bisimulation; [22] gives a complete axiomatization of strong bisimilarity for finite terms. We have taken the choice operation from CCS and the parallel composition from TCSP. Hence, in our paper, choice is decided by the first action, which if visible can be influenced by the environment; in contrast, TCSP has a choice that is decided completely

internally and another choice that is decided by the first visible action. As e.g. in [35, 26, 14], time does not decide choices in our paper; it does in [5], while [22] offers two different choice operators for the two different interpretations.

The testing approach also belongs in the first place to the CCS-world, and it leads naturally to considering preorders – instead of equivalences that most papers above look at. Before discussing papers about preorders, we must mention another group of timed process algebras: in this approach, starting with [1, 13], components attach local time stamps to actions (similar to logical time in distributed algorithms, see e.g. [18]) and actions occur as in an untimed algebra, i.e. in a sequence of actions from different components local time stamps might decrease – which is termed 'ill-timed but well-caused'. This way, the timed algebra stays very close to the untimed one, and in particular the operational semantics of parallel composition can be simply given as arbitrary interleaving; as an equivalence, bisimilarity is used. [1] only allows the synchronization of actions that 'by chance' have the same time stamp attached, and a component might block, if it does not find a suitable synchronization partner. [13] instead allows some form of busy-waiting, and the synchronizing components put their local clocks to the maximum of their clock values; this seems more realistic, but as a consequence bisimilarity is not a congruence for parallel composition anymore. Also e.g. [10] (see below) belongs to this approach; local clocks are also used in [11], but there, ill-timed computations are eliminated.

The literature about timed process algebras mostly deals with equivalences, possibly because most often some form of bisimulation is used to compare processes. Another reason is that most often synchronous systems are modelled and, as [23] argues, it seems unlikely that for synchronous systems a faster-than preorder with a few reasonable properties exists that is a precongruence for parallel composition. Nevertheless, there are several approaches to define preorders which we discuss now.

[14] presents a testing approach for an algebra TPL extending CCS that has discrete time steps σ in its syntax and in its operational semantics. The resulting must-preorder and its characterization coincide in essence with ours; it just looks different in [14], since it is formulated in terms of acceptance traces instead of refusal traces and since time steps and instantaneous actions are separated in the algebra. Also, the precongruence problem for choice is solved in a different, 'less semantical' way: process P is precongruent to Q if - for some action a not appearing in P or Q - $P + a$ is less than $Q + a$ in the must-preorder. We could show the same result. Another difference is that in [14] there is just one operational semantics, and the acceptance traces can be read off from the respective transition system. In our approach, already for the treatment of discrete time behaviour, we have one transition system for stand-alone behaviour and another one for the refusal traces; even if we changed the first one and made all visible actions patient, it would not be possible to read off the refusal traces from it – at least not in the way it is done in [14].

An essential difference is that TPL models synchronous systems and σ is exactly one unit of time; it is not clear, how we could translate a PAFAS-process with its upper time bounds to a TPL-process – while the reverse translation is clearly impossible, since synchronous systems are built upon a much stronger assumption about the global time. As a result, the test environment has much more direct control over the temporal behaviour in TPL than in our setting; as a consequence, no time bounds are needed for tests and the tests needed for the characterization are much, much simpler than ours. The complexity of our tests shows how much more difficult it is for an asynchronous test process to 'see' as much as a

synchronous one; that this is possible at all should be quite astonishing.

Since TPL contains a timeout construct, we should regard it as stronger as PAFAS; in this stronger algebra, [14] presents a complete axiomatization, something we do not have as yet. It seems most likely that we will have to strengthen our algebra to obtain such an axiomatization.

In view of the above observation of [23], it is not surprising that the testing preorder is not seen as a faster-than relation in [14]. Instead, [25] shows that it can be considered as relating systems w.r.t. their temporal and functional 'predictability' rather than their efficiency.

In the discretely timed algebra $\ell TCCS$ of [23], asynchronous systems are modelled without any progress assumption; instead, they can idle arbitrarily long and additional fixed delays can be specified. An efficiency preorder is defined based on a sort of strong bisimulation; bisimulation looks at the branching behaviour in detail, and as a consequence, an interpretation in terms of worst-case behaviour is not obvious. An axiomatization for finite sequential terms is given, though it seems not to fit the operational preorder completely; also, the operational preorder does not completely match the intuition, as also the authors have found out in the meantime [21]. The problems with axiomatizing parallel composition are discussed.

Again, it is not clear how to translate between PAFAS and $\ell TCCS$. In [23], for basic actions a and b (which are actually written with an underbar in [23]), we have $a \parallel_\emptyset b = a.b + b.a$ (using TCSP-parallel-composition although [23] really uses CCS-parallel-composition), which is wrong in PAFAS; hence, we should possibly relate our $\underline{a}$ to their basic a and our a to their $(1)a$. Then, in PAFAS, $\underline{a}.b$ is faster than $\underline{a}.\underline{b} + a.b$, since we only look at the worst-case behaviour of the second term, whereas this is wrong in $\ell TCCS$, where also the performance of $\underline{a}$ in the second term has to be matched. Even worse, $a \parallel_\emptyset b + \underline{a}.\underline{b}$ is faster than $\underline{a} \parallel_\emptyset b$ in $\ell TCCS$ and even strictly faster due to the summand $\underline{a}.\underline{b}$; in PAFAS, this summand is ignored as irrelevant for the worst-case behaviour and due to the refusal trace $b\mathbb{A}\ \underline{a} \parallel_\emptyset b$ is strictly faster than $a \parallel_\emptyset b + \underline{a}.\underline{b}$. Observe that the two processes cannot immediately perform b in $\ell TCCS$.

In the 'ill-timed but well-caused' approach, [10] also gives a bisimulation based preorder, where component speeds are fixed with respect to local clocks (modulo patience for communication in [9]). As already mentioned above, this local passage of time leads to ill-timed traces, hence this idea is hard to compare to our approach or any other of the preorders.

Finally, we want to mention an approach to compare the efficiency of untimed CCS-like terms, based on counting internal actions. Efficiency preorders on this basis have been investigated in [8] and [24] with a testing scenario, and in [3] and [4] with a bisimulation-like definition; in all these approaches, efficiency is measured by counting internal actions, where runs of a parallel composition are seen to be the interleaved runs of the components; consequently, in all cases (except [8], which does not consider parallel composition), $\tau.a \|_{\{a\}} \tau.a$ is as efficient as $\tau.\tau.a$, whereas in our setting $\tau.a \|_{\{a\}} \tau.a$ is strictly faster than $\tau.\tau.a$.

References

[1] L. Aceto and D. Murphy. Timing and causality in process algebra. *Acta Informatica*, 33:317–350, 1996.

[2] R. Alur and D. Dill. A theory of timed automata. *Theoret. Comput. Sci.*, 126:183–235, 1994.

[3] S. Arun-Kumar and M. Hennessy. An efficiency preorder for processes. *Acta Informatica*, 29:737–760, 1992.

[4] S. Arun-Kumar and V. Natarajan. Conformance: a precongruence close to bisimilarity. In J. Desel, editor, *Structures in Concurrency Theory*, Worksh. in Computing, 55–68. Springer, 1995.

[5] J. Baeten and J. Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3:142–188, 1991.

[6] E. Bihler. Effizienzvergleich bei verteilten Systemen – eine Theorie und ein Werkzeug. Diplomarbeit an der Uni. Augsburg, 1998.

[7] E. Bihler and W. Vogler. Efficiency of token-passing MUTEX-solutions – some experiments. In J. Desel et al., editors, *Applications and Theory of Petri Nets 1998*, Lect. Notes Comp. Sci. 1420, 185–204. Springer, 1998.

[8] R. Cleaveland and A. Zwarico. A theory of testing for real-time. In *Proc. 6th Symp. on Logic in Computer Science*, pages 110–119. IEEE Computer Society Press, 1991.

[9] F. Corradini. On the coarsest congruence within global-clock-bounded equivalence. *Theoret. Comput. Sci.*, 198:225–237, 1998.

[10] F. Corradini, R. Gorrieri, and M. Roccetti. Performance preorder and competitive equivalence. *Acta Informatica*, 34:805–835, 1997.

[11] F. Corradini and M. Pistore. Closed interval process algebra versus interval process algebra. *Acta Informatica*, 2000. To appear.

[12] R. De Nicola and M.C.B. Hennessy. Testing equivalence for processes. *Theoret. Comput. Sci.*, 34:83–133, 1984.

[13] R. Gorrieri, M. Roccetti, and E. Stancampiano. A theory of processes with durational actions. *Theoret. Comput. Sci.*, 140:73–294, 1995.

[14] M. Hennessy and T. Regan. A process algebra for timed systems. *Information and Computation*, 117:221–239, 1995.

[15] L. Jenner. Modular construction of fast asynchronous systems. Technical Report 1996-2, Inst. f. Informatik, Univ. Augsburg, http://www.Informatik.Uni-Augsburg.DE/techreport/, 1996.

[16] L. Jenner and W. Vogler. Comparing the efficiency of asynchronous systems. In J.-P. Katoen, editor, *AMAST Workshop on Real-Time and Probabilistic Systems*, Lect. Notes Comp. Sci. 1601, 172–191. Springer, 1999. Full version as Techn. Rep 1998-3 on http://www.informatik.uni-augsburg.de/skripts/techreports.

[17] L. Jenner and W. Vogler. Fast asynchronous systems in dense time. In F. Meyer auf der Heide and B. Monien, editors, *Automata, Languages and Programming ICALP'96*, Lect. Notes Comp. Sci. 1099, 75–86. Springer, 1996. To appear in Theoret. Comput. Sci.

[18] N. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers, San Francisco, 1996.

[19] P. Merlin and D.J. Farber. Recoverability of communication protocols. *IEEE Trans. Communications*, COM-24:1036–1043, 1976.

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

[21] F. Moller. Private communication, 2000.

[22] F. Moller and C. Tofts. A temporal calculus of communicating systems. In *CONCUR '90*, Lect. Notes Comp. Sci. 458, 401–415. Springer, 1990.

[23] F. Moller and C. Tofts. Relating processes with respect to speed. In J. Baeten and J. Groote, editors, *CONCUR '91*, Lect. Notes Comp. Sci. 527, 424–438. Springer, 1991.

[24] V. Natarajan and R. Cleaveland. An algebraic theory of process efficiency. In *11th Ann. Symp. Logic in Computer Science (LICS '96)*, 63–72. IEEE, 1996.

[25] V. Natarajan and R. Cleaveland. Predictability of real-time systems: A process-algebraic approach. In *11th IEEE Real-Time Systems Symp. (RTSS '96)*, IEEE, 1996.

[26] X. Nicollin, J. Sifakis, and S. Yovine. From ATP to timed graphs and hybrid systems. *Acta Informatica*, 30:181–202, 1993.

[27] L. Popova. On time Petri nets. *J. Inform. Process. Cybern. EIK*, 27:227–244, 1991.

[28] G.M. Reed and A.W. Roscoe. The timed failures-stability model for CSP. *Theoret. Comput. Sci.*, 211:85–127, 1999.

[29] G.M. Reed and A.W. Roscoe. Metric spaces as models for real-time concurrency. In *Mathematical Foundations of Programming Language Semantics*, Lect. Notes Comp. Sci. 298, 331–343. Springer, 1987.

[30] S. Schneider. An operational semantics for timed CSP. *Information and Computation*, 116(2):193–213, 1995.

[31] W. Vogler. Timed testing of concurrent systems. *Information and Computation*, 121:149–171, 1995.

[32] W. Vogler. Faster asynchronous systems. In I. Lee and S. Smolka, editors, *CONCUR 95*, Lect. Notes Comp. Sci. 962, 299–312. Springer, 1995. Full version as Report Nr. 317, Inst. f. Mathematik, Univ. Augsburg, 1995.

[33] W. Vogler. Efficiency of asynchronous systems and read arcs in Petri nets. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *ICALP 97*, Lect. Notes Comp. Sci. 1256, 538–548. Springer, 1997. Full version at http://www.informatik.uni-augsburg.de/∼vogler/ under the title 'Efficiency of Asynchronous Systems, Read Arcs, and the MUTEX-Problem'.

[34] W. Vogler and L. Jenner. Axiomatizing a fragment of PAFAS. Electronic Notes in Theoretical Computer Science, 39, 2000. To appear. See Techn. Report 2000-1 at http://www.informatik.uni-augsburg.de/skripts/techreports.

[35] Yi Wang. Real-time behaviour of asynchronous agents. In *CONCUR '90*, Lect. Notes Comp. Sci. 458, 502–520. Springer, 1990.