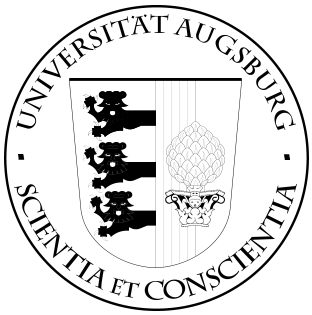
Fairness of components in system computations

Flavio Corradini, Maria Rita di Berardini, Walter Vogler

Angaben zur Veröffentlichung / Publication details:

Corradini, Flavio, Maria Rita di Berardini, and Walter Vogler. 2005. "Fairness of components in system computations." Augsburg: Universität Augsburg.

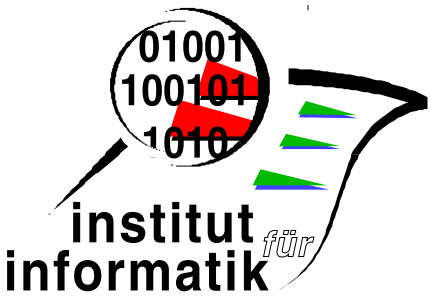
Universität Augsburg



Fairness of Components in System Computations

F. Corradini, M.R. Di Berardini, W. Vogler

Report 2005–3 January 2005



Institut für Informatik

D-86135 Augsburg

Fairness of Components in System Computations*

F. Corradini, M.R. Di Berardini
Dipartimento di Matematica e Informatica
Università di Camerino
{flavio.corradini,mariarita.diberardini}@unicam.it

W. Vogler
Institut für Informatik
Universität Augsburg
vogler@informatik.uni-augsburg.de

Abstract

In this paper we provide a simple characterization of (weak) fairness of *components* as defined by Costa and Stirling in [6]. The study is carried out at system specification level by resorting to a common process description language. This paper follows and exploits similar techniques as those developed in [3] – where fairness of *actions* was taken into account and was contrasted to the PAFAS timed operational semantics – but the characterization of fair executions is based on a new semantics for PAFAS; it makes use of only two copies of each basic action instead of infinitely many as in [6] and allows for a simple and finite representation of fair executions by using regular expressions.

The new semantics can also be understood as describing timed behaviour of systems with upper time bounds. The paper discusses in detail how this new semantics differs from the old one, and why theses changes are necessary to properly capture fairness of components.

1 Introduction

In the theory and practice of parallel systems, fairness plays an important role when describing the system dynamics. It is usually a necessary requirement for proving liveness properties of the system. Several fairness notions applied to different entities in a system have been proposed in the literature.

Costa and Stirling define fairness for CCS without restriction in [5] and for fully fledged CCS in [6] and present very nice characterizations of fair runs. They distinguish between weak fairness and strong fairness as well as between fairness of actions (also called events) and of components; while these notions coincide in [5], they differ in [6], where weak and strong fairness of components are studied. In this paper, we will concentrate on weak fairness, which requires that if a component (an action, resp.) can *almost always* proceed then it must eventually do so, and in fact it must proceed infinitely often. An important and useful result stated in [5, 6] characterizes fair computations as the concatenation of certain finite sequences, called LP-steps in [6]. This characterization permits to think of fairness in terms of a localizable property and not as a property of complete (maximal) executions; but even for a finite-state process, LP-steps usually give rise to a transition system with infinitely many transitions, which is therefore infinitely branching.

In our previous paper [3], we have exhibited a connection between weak *fairness of actions* and a timed operational semantics by resorting to a common, well-known process description language PAFAS (a variant of CCS with TCSP parallel composition). The language is extended with labels allowing to filter out those process executions that are (weakly) fair (as in [5, 6]), and with upper time bounds for the process activities (as in [4]), where these bounds are 1 for simplicity and time is discrete. (Upper time bounds have also been studied in [9] for the area of distributed algorithms,

*This work was supported by MURST project 'Sahara: Software Architectures for Heterogeneous Access Networks infrastructures' and by the Center of Excellence for Research 'DEWS: Architectures and Design Methodologies for Embedded Controllers, Wireless Interconnect and System-on-chip'.

in e.g. [11] for Petri nets, and in [8] for a process algebraic setting with bisimulation.) The paper [3] shows that *fairness* and *timing*, two important features of parallel system computations, are closely related by giving two main results. First, it is shown that each everlasting (or non-Zeno) timed process execution is fair. Second, [3] provides a characterization for fair executions of untimed processes in terms of timed process executions. For finite state processes, it also results in a finite representation of fair executions using regular expressions.

In this paper we concentrate on weak *fairness of components*. It turned out that the PAFAS timed operational semantics is not a suitable abstraction for fairness of components as it is for fairness of actions. But we have found a suitable variation of this semantics which allows us to characterize Costa and Stirling's fairness of components in terms of a much simpler filtering of system executions compared to the label-based fairness definition in [5, 6]. The results of this paper are conceptually analogous to those in [3], but a number of technical changes were needed to define the new semantics and, consequently, the proof details are quite different.

The new operational semantics of processes we have arrived at can again be understood as the behaviour of timed processes with upper time bounds. We assume that for each parallel component this upper time bound is 1; hence, a component will perform some action within time 1 provided it is continually enabled (or live in the terminology of [5, 6]). In other words, when time 1 passes, a live component becomes urgent and, before the next time step, it must perform an action (or get disabled). We will show that the phases between the time steps correspond to the above mentioned LP-steps in [6].

Our characterization of fair executions results in a representation with technical advantages compared to the approach of [5, 6]. In order to keep track of the different instances of system activities along a system execution, Costa and Stirling associate labels to actions (and the operators), and the labels are essential in the definition of fair computations. New labels are created dynamically during the system evolution with the immediate effect of changing the syntax of process terms and of assuming that different instances of the same basic actions exist; if a process has an infinite execution, there will be infinitely many instances of some actions – distinguished by their label. Consequently, because of this dynamic generation of labels, cycles in the transition system of a process are impossible and even finite-state processes (according to the ordinary operational semantics) usually become infinite-state. From the maximal runs of such a transition system, Costa and Stirling filter out the fair computations by a criterion that considers the processes and their labels on a maximal run.

Our alternative operational semantics also provides such a two-level description. We also change the syntax of processes to take note of urgency, but this is much simpler than the labels of [5, 6]; e.g. we only assume two instances of the same basic action corresponding to two different states of the action itself: one in which the action is not forced to be performed, and one in which it has to be performed urgently. An important consequence of this fact is that our operational semantics leaves finite-state processes finite-state. To get the fair runs, we apply a simpler filter, which does not consider the processes: we simply require that infinitely many time steps occur in a run, i.e. we only consider non-Zeno runs. As a small price, we have to project away the time steps in the end.

As mentioned above, Costa and Stirling give a one-level characterization of fair computations with an SOS-semantics defining so-called LP-steps; these are (finite, though usually unbounded) sequences of actions leading from ordinary processes to ordinary processes, with the effect that even finite-state transition systems for LP-steps usually have infinitely many transitions – although they are at least finite-state. In contrast, our operational semantics only refers to unit time steps and single actions, and consequently a finite-state transition system is really finite.

Finally, using standard automata-theoretic techniques, we can get rid of the time steps in such a finite-state transition system by constructing another finite-state transition system with regular expressions as arc labels; maximal runs in this transition system are exactly the fair runs. This way we also arrive at a one-level description, and ours is truly finite. With respect to the similar

result in [3], we have overcome some technical problems with the treatment of recursive processes; as a consequence the transition system in the present paper provides a more faithful description of fair runs because it only contains standard processes (without any marking of urgent components) which can be reached from the initial process according to standard transitions.

We close with a detailed discussion on the changes we needed to the above mentioned PAFAS timed operational semantics and on the difficulties we had to overcome to properly capture fairness of components. This discussion highlights the differences between the two fairness notions of system computations we have studied, i.e. fairness of actions and fairness of components. Most of the proofs have been moved to appendixes.

2 PAFAS - A Process Algebra for Faster Asynchronous Systems

PAFAS is a CCS-like process description language [10] (with *TCSP*-like parallel composition [7]), where basic actions are atomic and instantaneous but have associated a time bound interpreted as a maximal time delay for their execution. As explained in [4], these upper time bounds (which are either 0 or 1, for simplicity) are suitable for evaluating the performance of asynchronous systems. Moreover, time bounds do not influence functionality (i.e. which actions are performed); so compared to CCS, also PAFAS treats the full functionality of asynchronous systems. In the present paper, the time bounds are associated to the parallel components of a term, resulting in slightly different terms and different SOS-rules; this variant of PAFAS will be called PAFAS$^{\mathbf{c}}$ henceforth.

2.1 PAFAS$^{\mathbf{c}}$ Processes

We use standard notation. $\mathbb{A}$ denotes an infinite set of basic actions. τ represents internal activity. Let $\mathbb{A}_\tau = \mathbb{A} \cup \{\tau\}$. Elements of $\mathbb{A}$ are denoted by $a, b, c, \ldots$ and those of $\mathbb{A}_\tau$ are denoted by $\alpha, \beta, \ldots$. Actions in $\mathbb{A}_\tau$ can let time 1 pass before their execution, i.e. 1 is their maximal delay. After that time, they become *urgent* actions written $\underline{a}$ or $\underline{\tau}$; these have maximal delay 0. The set of urgent actions is denoted by $\underline{\mathbb{A}}_\tau = \{\underline{a} \mid a \in \mathbb{A}\} \cup \{\underline{\tau}\}$ and is ranged over by $\underline{\alpha}, \underline{\beta}, \ldots$. Elements of $\mathbb{A}_\tau \cup \underline{\mathbb{A}}_\tau$ are ranged over by μ. $\mathcal{X}$ is the set of process variables, used for recursive definitions. Elements of $\mathcal{X}$ are denoted by $x, y, z, \ldots$. $\Phi : \mathbb{A}_\tau \to \mathbb{A}_\tau$ is a *general relabelling function* if the set $\{\alpha \in \mathbb{A}_\tau \mid \emptyset \neq \Phi^{-1}(\alpha) \neq \{\alpha\}\}$ is finite and $\Phi(\tau) = \tau$. Such a function can also be used to define *hiding*: P/A, where the actions in A are made internal, is the same as $P[\Phi_A]$, where the relabelling function Φ_A is defined by $\Phi_A(\alpha) = \tau$ if $\alpha \in A$ and $\Phi_A(\alpha) = \alpha$ if $\alpha \notin A$.

We assume that time elapses in a discrete way.[1] Thus, an action prefixed process $a.P$ can either do action a and become process P (as usual in CCS) or can let one unit time step pass and become $\underline{a}.P$; $\underline{a}$ is called *urgent* a, and $\underline{a}.P$ cannot let time pass, but can only do a to become P. Since we associate time bounds to components in the present paper, we may also mark the other dynamic operator $+$ as urgent: a process $P + Q$ becomes $P \underline{+} Q$ after a time step.

Definition 2.1 (*timed process terms*)
The set $\tilde{\mathbb{P}}_1$ of *initial timed process terms* is generated by the following grammar:

$$P ::= \mathsf{nil} \mid x \mid \alpha.P \mid P + P \mid P \|_A P \mid P[\Phi] \mid \mathsf{rec}\, x.P$$

where $x \in \mathcal{X}$, $\alpha \in \mathbb{A}_\tau$, Φ is a general relabelling function and $A \subseteq \mathbb{A}$ possibly infinite. Elements in $\tilde{\mathbb{P}}_1$ correspond to ordinary CCS-like process terms.

The set $\tilde{\mathbb{P}}$ of (general) *timed process terms* is generated by the following grammar:

$$Q ::= P \mid \underline{\alpha}.P \mid P \underline{+} P \mid Q \|_A Q \mid Q[\Phi] \mid \mathsf{rec}\, x.Q$$

[1] PAFAS is not time domain dependent, meaning that the choice of discrete or continuous time makes no difference for the testing-based semantics of asynchronous systems studied in [4, 2].

where $P \in \tilde{\mathbb{P}}_1$, $x \in \mathcal{X}$, $\alpha \in \mathbb{A}_\tau$, Φ is a general relabelling function, and $A \subseteq \mathbb{A}$ possibly infinite. We assume that recursion is always *guarded*, i.e. for $\mathsf{rec}\, x.Q$ variable x only appears in Q within the scope of a prefix $\mu.()$ with $\mu \in \mathbb{A}_\tau \cup \underline{\mathbb{A}}_\tau$. A term Q is *guarded* if each occurrence of a variable is guarded in this sense; it is *closed* if every variable x is bound by the corresponding $\mathsf{rec}\, x$-operator.

The set of closed timed process terms in $\tilde{\mathbb{P}}$ and $\tilde{\mathbb{P}}_1$, simply called *processes* and *initial processes* resp., is denoted by $\mathbb{P}$ and $\mathbb{P}_1$ resp.[2]

For studying fairness, we are interested in the initial processes, and these coincide in PAFAS and in PAFAS$^{\mathbf{c}}$; they are actually common CCS/TCSP-like processes. The additional terms of $\tilde{\mathbb{P}}$ turn up in evolutions of terms from $\tilde{\mathbb{P}}_1$ involving time steps, and here PAFAS and PAFAS$^{\mathbf{c}}$ differ.

We define function $\mathcal{A}(\_)$ on process terms, that returns the *active* (or enabled) actions of a process term. Given a process Q, $\mathcal{A}(Q)$ abbreviates $\mathcal{A}(Q, \emptyset)$ and $\mathcal{A}(Q, A)$ denotes the set of actions that process Q can perform when the environment prevents the actions in $A \subseteq \mathbb{A}$.

Definition 2.2 (*activated basic actions*)
Let $Q \in \tilde{\mathbb{P}}$ and $A \subseteq \mathbb{A}$. The set $\mathcal{A}(Q, A)$ is defined by induction on Q as follows:

Var, Nil: $\mathcal{A}(x, A) = \mathcal{A}(\mathsf{nil}, A) = \emptyset$

Pref: $\mathcal{A}(\alpha.P, A) = \mathcal{A}(\underline{\alpha}.P, A) = \begin{cases} \{\alpha\} & \text{if } \alpha \notin A \\ \emptyset & \text{otherwise} \end{cases}$

Sum: $\mathcal{A}(P_1 + P_2, A) = \mathcal{A}(P_1 \pm P_2, A) = \mathcal{A}(P_1, A) \cup \mathcal{A}(P_2, A)$

Par: $\mathcal{A}(Q_1 \parallel_B Q_2, A) = \mathcal{A}(Q_1, A \cup A') \cup \mathcal{A}(Q_2, A \cup A'')$ where
$A' = (\mathcal{A}(Q_1) \backslash \mathcal{A}(Q_2)) \cap B$ and $A'' = (\mathcal{A}(Q_2) \backslash \mathcal{A}(Q_1)) \cap B$

Rel: $\mathcal{A}(Q[\Phi], A) = \Phi(\mathcal{A}(Q, \Phi^{-1}(A)))$

Rec: $\mathcal{A}(\mathsf{rec}\, x.Q, A) = \mathcal{A}(Q, A)$

The set A represents the actions restricted upon; therefore $\mathcal{A}(\alpha.P, A) = \mathcal{A}(\underline{\alpha}.P, A) = \emptyset$ if $\alpha \in A$ and $\mathcal{A}(\alpha.P, A) = \mathcal{A}(\underline{\alpha}.P, A) = \{\alpha\}$, if $\alpha \notin A$. A nondeterministic process can perform all the actions that its alternative components can perform minus the restricted ones. Parallel composition increases the prevented set. $\mathcal{A}(P \parallel_B Q, A)$ includes the actions that P can perform when we prevent all actions in A plus all actions in B that Q cannot perform, and it includes the analogous actions of Q.

2.2 The operational behaviour of PAFAS$^{\mathbf{c}}$ processes

The transitional semantics describing the functional behavior of PAFAS$^{\mathbf{c}}$ processes indicates which basic actions they can perform. Timing can be disregarded: when an action is performed, one cannot see whether it was urgent or not, i.e. $\underline{\alpha}.P \xrightarrow{\alpha} P$; on the other hand, component $\alpha.P$ has to act *within* time 1, i.e. it can also act immediately, giving $\alpha.P \xrightarrow{\alpha} P$. The operational semantics exploits two functions on process terms: $\mathsf{clean}(\_)$ and $\mathsf{unmark}(\_)$. Function $\mathsf{clean}(\_)$ *removes all inactive urgencies* in a process term $Q \in \tilde{\mathbb{P}}$. Indeed, when a process evolves, components may lose their urgency since their actions are no longer enabled due to changes of the context; the corresponding change of markings is performed by clean, where again set A in $\mathsf{clean}(Q, A)$ denotes the set of actions that are not enabled due to restrictions of the environment.

Definition 2.3 (*cleaning inactive urgencies*)
Given a process term $Q \in \tilde{\mathbb{P}}$ we define $\mathsf{clean}(Q)$ as $\mathsf{clean}(Q, \emptyset)$ where, for a set $A \subseteq \mathbb{A}$, $\mathsf{clean}(Q, A)$ is defined as follows:

[2] As shown in [4], $\mathbb{P}_1$ processes do not have time-stops; i.e. every finite process run can be extended such that time grows unboundedly. This result was proven for a different operational semantics than that defined in this paper but a similar proof applies also in the current setting.

Nil, Var: $\mathsf{clean}(\mathsf{nil}, A) = \mathsf{nil}, \qquad \mathsf{clean}(x, A) = x$

Pref: $\mathsf{clean}(\underline{\alpha}.P, A) = \begin{cases} \alpha.P & \text{if } \alpha \in A \\ \underline{\alpha}.P & \text{otherwise} \end{cases} \qquad \mathsf{clean}(\alpha.P, A) = \alpha.P$

Sum: $\mathsf{clean}(P_1 \mathbin{\underline{+}} P_2, A) = \begin{cases} P_1 + P_2 & \text{if } \mathcal{A}(P_1) \cup \mathcal{A}(P_2) \subseteq A \\ P_1 \mathbin{\underline{+}} P_2 & \text{otherwise} \end{cases}$

$\mathsf{clean}(P_1 + P_2, A) = P_1 + P_2$

Par: $\mathsf{clean}(Q_1 \parallel_B Q_2, A) = \mathsf{clean}(Q_1, A \cup A') \parallel_B \mathsf{clean}(Q_2, A \cup A'')$
where $A' = (\mathcal{A}(Q_1) \backslash \mathcal{A}(Q_2)) \cap B$ and $A'' = (\mathcal{A}(Q_2) \backslash \mathcal{A}(Q_1)) \cap B$

Rel: $\mathsf{clean}(Q[\Phi], A) = \mathsf{clean}(Q, \Phi^{-1}(A))[\Phi]$

Rec: $\mathsf{clean}(\mathsf{rec}\, x.Q, A) = \mathsf{rec}\, x.\, \mathsf{clean}(Q, A)$

Function $\mathsf{unmark}(\_)$ simply removes all urgencies (inactive or not) in a process term $Q \in \tilde{\mathbb{P}}$ and can be defined, as expected, by induction on the process structure (see Definition 8.3 in Appendix A).

Definition 2.4 (*Functional operational semantics*) The following SOS-rules define the transition relations $\xrightarrow{\alpha} \subseteq (\tilde{\mathbb{P}} \times \tilde{\mathbb{P}})$ for $\alpha \in \mathbb{A}_\tau$, the *action transitions*. We write $Q \xrightarrow{\alpha} Q'$ if $(Q, Q') \in \xrightarrow{\alpha}$ and $Q \xrightarrow{\alpha}$ if there exists a $Q' \in \tilde{\mathbb{P}}$ such that $(Q, Q') \in \xrightarrow{\alpha}$, and similar conventions will apply later on.

$$\textsc{Pref}_{a1}\frac{}{\alpha.P \xrightarrow{\alpha} P} \qquad \textsc{Pref}_{a2}\frac{}{\underline{\alpha}.P \xrightarrow{\alpha} P}$$

$$\textsc{Sum}_{a1}\frac{P_1 \xrightarrow{\alpha} P_1'}{P_1 + P_2 \xrightarrow{\alpha} P_1'} \qquad \textsc{Sum}_{a2}\frac{P_1 \xrightarrow{\alpha} P_1'}{P_1 \mathbin{\underline{+}} P_2 \xrightarrow{\alpha} P_1'}$$

$$\textsc{Par}_{a1}\frac{\alpha \notin A,\ Q_1 \xrightarrow{\alpha} Q_1'}{Q_1 \|_A Q_2 \xrightarrow{\alpha} \mathsf{clean}(Q_1' \|_A Q_2)} \qquad \textsc{Par}_{a2}\frac{\alpha \in A,\ Q_1 \xrightarrow{\alpha} Q_1',\ Q_2 \xrightarrow{\alpha} Q_2'}{Q_1 \|_A Q_2 \xrightarrow{\alpha} \mathsf{clean}(Q_1' \|_A Q_2')}$$

$$\textsc{Rel}_a\frac{Q \xrightarrow{\alpha} Q'}{Q[\Phi] \xrightarrow{\Phi(\alpha)} Q'[\Phi]} \qquad \textsc{Rec}_a\frac{Q\{\mathsf{rec}\, x.\, \mathsf{unmark}(Q)/x\} \xrightarrow{\alpha} Q'}{\mathsf{rec}\, x.Q \xrightarrow{\alpha} Q'}$$

Additionally, there are symmetric rules for Par_{a1}, Sum_{a1} and Sum_{a2} for actions of P_2.

Observe the following for $\textsc{Sum}_{a2}$: due to our syntax, P_1 in $P_1 \mathbin{\underline{+}} P_2$ is an initial process, i.e. has no components marked as urgent, and the same applies to P_1'. Thus, $P_1 \mathbin{\underline{+}} P_2$ loses its urgency in a transition according to $\textsc{Sum}_{a2}$; this corresponds to our intuition, since this atomic component (i.e. without parallel subcomponents) performs an action, which it had to perform urgently, and can afterwards wait with any further activity for time 1.

When in the rules for parallel composition one component changes, this changes the context for the other component such that some urgent components might get disabled. For the necessary changes to the marking, clean is called upon as announced above. E.g. in $(\underline{a}.\mathsf{nil} \|_\emptyset \underline{b}.\mathsf{nil}) \|_{\{a,b\}} (a.\mathsf{nil} + b.\mathsf{nil} + c.a.b.\mathsf{nil}) \xrightarrow{c} (\underline{a}.\mathsf{nil} \|_\emptyset b.\mathsf{nil}) \|_{\{a,b\}} (a.b.\mathsf{nil})$, the second (but not the first) component loses its urgency.

The use of unmark in rule $\textsc{Rec}_a$ has to be contrasted with the temporal behaviour defined next that marks as urgent recursive processes $\mathsf{rec}x.P$ according to a rule $\mathsf{urgent}(\mathsf{rec}x.P) = \mathsf{rec}x.\mathsf{urgent}(P)$. Since occurrences of x in P are guarded, each x stands for a process which is not enabled yet and cannot be urgent; thus, these recursive calls in $\mathsf{rec}\, x.\, \mathsf{urgent}(P)$ refer to P and not to $\mathsf{urgent}(P)$, which explains the substitution in rule $\textsc{Rec}_a$ of Definition 2.4, which in turn shows the use of unmark; cf. the example at the end of the section.

In addition to the purely functional transitions, we also consider transitions corresponding to the passage of one unit of time. The function urgent we exploit marks the *enabled* parallel components of a process as urgent; such a component can be identified with a dynamic operator (an action or a choice), which gets underlined. This marking occurs when a time step is performed, because afterwards the marked components have to act in zero time – unless they are disabled. If such an urgent component acts, it should lose its urgency; and indeed, the marking vanishes with the dynamic operator. The next time step will only be possible, if no component is marked as urgent.

Definition 2.5 (*time step, execution sequence, timed execution sequence*)

For $P \in \tilde{\mathbb{P}}_1$, we write $P \xrightarrow{1} Q$ when $Q = \mathsf{urgent}(P)$, where $\mathsf{urgent}(P)$ abbreviates $\mathsf{urgent}(P, \emptyset)$ and $\mathsf{urgent}(P, A)$ is defined as follows:

Nil, Var: $\quad \mathsf{urgent}(\mathsf{nil}, A) = \mathsf{nil}, \qquad \mathsf{urgent}(x, A) = x$

Pref: $\quad \mathsf{urgent}(\alpha.P, A) = \begin{cases} \underline{\alpha}.P & \text{if } \alpha \notin A \\ \alpha.P & \text{otherwise} \end{cases}$

Sum: $\quad \mathsf{urgent}(P_1 + P_2, A) = \begin{cases} P_1 \mathbin{\underline{+}} P_2 & \text{if } (\mathcal{A}(P_1) \cup \mathcal{A}(P_2)) \backslash A \neq \emptyset \\ P_1 + P_2 & \text{otherwise} \end{cases}$

Par: $\quad \mathsf{urgent}(P_1 \parallel_B P_2, A) = \mathsf{urgent}(P_1, A \cup A') \parallel_B \mathsf{urgent}(P_2, A \cup A'')$
where $\quad A' = (\mathcal{A}(P_1) \backslash \mathcal{A}(P_2)) \cap B$ and $A'' = (\mathcal{A}(P_2) \backslash \mathcal{A}(P_1)) \cap B$

Rel: $\quad \mathsf{urgent}(P[\Phi, A) = \mathsf{urgent}(P, \Phi^{-1}(A))[\Phi]$

Rec: $\quad \mathsf{urgent}(\mathsf{rec}\, x.P, A) = \mathsf{rec}\, x.\, \mathsf{urgent}(P, A)$

We say that a sequence of transitions $\gamma = Q_0 \xrightarrow{\lambda_0} Q_1 \xrightarrow{\lambda_1} \ldots$ with $\lambda_i \in \mathbb{A}_\tau \cup \{1\}$ is a *timed execution sequence* if it is an infinite sequence of action transitions and time steps; note that a maximal sequence of such transitions/steps starting at some $Q_0 \in \tilde{\mathbb{P}}_1$ is never finite, since for $\gamma = Q_0 \xrightarrow{\lambda_0} Q_1 \xrightarrow{\lambda_1} \ldots \xrightarrow{\lambda_{n-1}} Q_n$, we have $Q_n \xrightarrow{\alpha}$ or $Q_n \xrightarrow{1}$ (see Proposition 8.16 in Appendix A).

For an *initial* process P_0, we say that a sequence of transitions $\gamma = P_0 \xrightarrow{\alpha_0} P_1 \xrightarrow{\alpha_1} \ldots$ with $\alpha_i \in \mathbb{A}_\tau$ is an *execution sequence* if it is a maximal sequence of action transitions; i.e. it is infinite or ends with a process P_n such that $P_n \overset{\alpha}{\not\rightarrow}$ for any action α.

As an example for the use of the various definitions, consider the following behaviour of $P = (a.\mathsf{nil} + b.\mathsf{nil}) \parallel_{\{a,b\}} \mathsf{rec}\, x.\, (a.\mathsf{nil} + c.(b.\mathsf{nil} + d.x))$:

$P \xrightarrow{1} (a.\mathsf{nil} \mathbin{\underline{+}} b.\mathsf{nil}) \parallel_{\{a,b\}} \mathsf{rec}\, x.\, (a.\mathsf{nil} \mathbin{\underline{+}} c.(b.\mathsf{nil} + d.x)) \xrightarrow{c}$
$(a.\mathsf{nil} \mathbin{\underline{+}} b.\mathsf{nil}) \parallel_{\{a,b\}} (b.\mathsf{nil} + d.\mathsf{rec}\, x.\, (a.\mathsf{nil} + c.(b.\mathsf{nil} + d.x))) \xrightarrow{d}$
$(a.\mathsf{nil} \mathbin{\underline{+}} b.\mathsf{nil}) \parallel_{\{a,b\}} \mathsf{rec}\, x.\, (a.\mathsf{nil} + c.(b.\mathsf{nil} + d.x)) \xrightarrow{c}$
$(a.\mathsf{nil} \mathbin{\underline{+}} b.\mathsf{nil}) \parallel_{\{a,b\}} (b.\mathsf{nil} + d.\mathsf{rec}\, x.\, (a.\mathsf{nil} + c.(b.\mathsf{nil} + d.x)))$

After the time step, both components are urgent; in particular, the left hand component can synchronize on a, while b is not possible. Then the right hand component performs c and loses its urgency. Now a is not possible anymore, but the left hand component remains urgent since now it can synchronize on b. Also, observe the application of $\textsc{Rec}_a$. The process reached returns to itself with dc, so this behaviour could be repeated indefinitely. But since the left hand component is urgent all the time, a time step is never possible, matching the intuitive idea that this component has to act within time 1.

3 Fairness and PAFAS$^{\mathbf{C}}$

In this section we briefly describe our theory of fairness. It closely follows Costa and Stirling's theory of (weak) fairness. The main ingredients of the theory are:

- *A labelling for process terms.* This allows to detect during a transition which component actually moves; e.g., for process $P = \mathsf{rec}\, x.\alpha.x$, we need additional information to detect whether the left hand side or the right hand side actually moves in the transition $P\|_\emptyset P \xrightarrow{\alpha} P\|_\emptyset P$.

- *Live components.* A component of a process term is live if it can perform an action. In a term like $a.b.\mathsf{nil}\|_{\{b\}} b.\mathsf{nil}$ only action a can be performed while b cannot, momentarily. Thus the left component of the parallel composition is live and such a component corresponds to a label. Intuitively, the components becoming urgent with a time step should exactly be the live components (see Corollary 9.7-3 in Appendix B).

- *Fair sequences.* A maximal sequence is fair when no component in a process term becomes live and then remains live throughout.

These items sketch the general methodology used by Costa and Stirling to define and isolate fair computations in [5, 6]. Most of the definitions in the rest of this section are taken from [6] with the obvious slight variations due to the different language we are using (the timed process algebra PAFAS$^{\mathrm{c}}$ with TCSP parallel composition instead of CCS). We also take from [6] those results that are language independent. The others are proven in the appendix.

3.1 A labelling for process terms

Costa and Stirling associate labels with all basic actions and operators inside a process, in such a way that no label occurs more than once in an expression. We call this property *unicity of labels*. Also along a computation, labels are unique and, once a label disappears, it will not reappear in the process anymore.

The set of *labels* is $\mathsf{LAB} = \{1, 2\}^*$ with ε as the empty label and $u, v, w, \ldots$ as typical elements. Labels are written as indexes and in case of parallel composition as upper indexes; they are assigned systematically following the structure of PAFAS$^{\mathrm{c}}$ terms. Due to recursion the labelling is dynamic: the rule for rec generates new labels.

Definition 3.1 (*labelled process algebra*)
The labelled process algebra $\mathsf{L}(\tilde{\mathbb{P}})$ (and similarly $\mathsf{L}(\tilde{\mathbb{P}}_1)$ etc.) is defined as $\bigcup_{u\in\mathsf{LAB}} \mathsf{L}_u(\tilde{\mathbb{P}})$, where $\mathsf{L}_u(\tilde{\mathbb{P}}) = \bigcup_{P\in\tilde{\mathbb{P}}} \mathsf{L}_u(P)$ and $\mathsf{L}_u(P)$ is defined inductively as follows:

Nil, Var :	$\mathsf{L}_u(\mathsf{nil}) = \{\mathsf{nil}_u\}$, $\mathsf{L}_u(x) = \{x_u\}$
	In examples, we will often write nil for nil_u, if the label u is not relevant.
Pref:	$\mathsf{L}_u(\mu.P) = \{\mu_u.P' \mid P' \in \mathsf{L}_{u1}(P)\}$
Sum:	$\mathsf{L}_u(P_1 + P_2) = \{P_1' +_u P_2' \mid P_1' \in \mathsf{L}_{u1}(P_1),\ P_2' \in \mathsf{L}_{u2}(P_2)\}$
	$\mathsf{L}_u(P_1 \pm P_2) = \{P_1' \pm_u P_2' \mid P_1' \in \mathsf{L}_{u1}(P_1),\ P_2' \in \mathsf{L}_{u2}(P_2)\}$
Par:	$\mathsf{L}_u(Q_1 \ \|_A Q_2) = \{Q_1' \|_A^u Q_2' \mid Q_1' \in \mathsf{L}_{u1v}(Q_1),\ Q_2' \in \mathsf{L}_{u2v'}(Q_2)$
	where $v, v' \in \mathsf{LAB}\}$
Rel:	$\mathsf{L}_u(Q[\Phi]) = \{Q'[\Phi_u] \mid Q' \in \mathsf{L}_{u1v}(Q)$ where $v \in \mathsf{LAB}\}$
Rec:	$\mathsf{L}_u(\mathsf{rec}\, x.Q) = \{\mathsf{rec}\, x_u.Q' \mid Q' \in \mathsf{L}_{u1}(Q)\}$

We assume that, in $\mathsf{rec}\, x_u.Q$, $\mathsf{rec}\, x_u$ binds all free occurrences of a labelled x; analogously, Φ_u acts on actions as Φ. We let $\mathsf{L}(Q) = \bigcup_{u\in\mathsf{LAB}} \mathsf{L}_u(Q)$ and $\mathsf{LAB}(Q)$ is the set of labels occurring in Q.

The unicity of labels must be preserved under derivation. For this reason, in the rec rule the standard substitution must be replaced by a substitution operation which also changes the labels of the substituted expression.

Definition 3.2 (*a new substitution operator*)
The new substitution operation, denoted by $\{\!|\_|\!\}$, is defined on $\mathsf{L}(\tilde{\mathbb{P}})$ using the following operators:

i. $()^{+v}$ If $Q \in \mathsf{L}_u(\tilde{\mathbb{P}})$, then $(Q)^{+v}$ is the term in $\mathsf{L}_{vu}(\tilde{\mathbb{P}})$ obtained by prefixing v to labels in Q.

ii. $()_\varepsilon$ If $Q \in \mathsf{L}_u(\tilde{\mathbb{P}})$, then $(Q)_\varepsilon$ is the term in $\mathsf{L}_\varepsilon(\tilde{\mathbb{P}})$ obtained by removing the prefix u from all labels in Q. (Note that u is the unique prefix-minimal label in Q.)

Suppose $Q, R \in \mathsf{L}(\tilde{\mathbb{P}})$ and $x_u, \ldots, x_v$ are all free occurrences of a labelled x in Q then $Q\{\!|R/x|\!\} = Q\{((R)_\varepsilon)^{+u}/x_u, \ldots, ((R)_\varepsilon)^{+v}/x_v\}$. The motivation of this definition is that in $Q\{\!|R/x|\!\}$ each substituted R inherits the label of the x it replaces.

Moreover, for $P \in \mathsf{L}(\tilde{\mathbb{P}}_1)$ and $A \subseteq \mathbb{A}$ we can define $\mathsf{urgent}(P, A)$ just as in Definition 2.5. Similarly, we can define $\mathcal{A}(Q, A)$, $\mathsf{clean}(Q, A)$ and $\mathsf{unmark}(Q)$ for labelled terms as above. Now, the behavioural operational semantics of labelled PAFAS$^{\mathrm{c}}$ is obtained by replacing the rules Rec_a in Definition 2.4 with the rule:

$$\mathrm{Rec}_a \frac{Q\{\!|\, \mathsf{rec}\, x_u.\mathsf{unmark}(Q)/x \,|\!\} \xrightarrow{\alpha} Q'}{\mathsf{rec}\, x_u.Q \xrightarrow{\alpha} Q'}$$

and the rules Pref_{a1} and Pref_{a2} in Definition 2.4 with the rules:

$$\mathrm{Pref}_{a1} \frac{}{\alpha_u.P \xrightarrow{\alpha} P} \qquad \mathrm{Pref}_{a2} \frac{}{\underline{\alpha}_u.P \xrightarrow{\alpha} P}$$

because we assume that labels are not observable when actions are performed. The other rules are unchanged.

Easy but important are the relationships between activated actions and transitions of PAFAS$^{\mathrm{c}}$ and labelled PAFAS$^{\mathrm{c}}$ processes. The following proposition shows that labels are just annotations that do not interfere with these notions. Let R be the operation of removing labels from a labelled PAFAS$^{\mathrm{c}}$ term.

Proposition 3.3 Let $Q \in \mathsf{L}_u(\tilde{\mathbb{P}})$ and $A \subseteq \mathbb{A}_\tau$. Then:

i. $Q \xrightarrow{\alpha} R$ implies $\mathsf{R}(Q) \xrightarrow{\alpha} \mathsf{R}(R)$ in unlabelled PAFAS$^{\mathrm{c}}$;

ii. if $Q' \xrightarrow{\alpha} R'$ in unlabelled PAFAS$^{\mathrm{c}}$ and $Q' = \mathsf{R}(Q)$, then $Q \xrightarrow{\alpha} R$ for some R with $R' = \mathsf{R}(R)$;

iii. $\mathcal{A}(Q, A) = \mathcal{A}(\mathsf{R}(Q), A)$.

An immediate consequence of the labelling are the following facts that have been proven in [6]: No label occurs more than once in a given process $P \in \mathsf{L}_u(\tilde{\mathbb{P}})$. Moreover, central to labelling is the persistence and disappearance of labels under derivation. In particular, once a label disappears in a sequence of transitions it can never reappear. It is these features which allow us to recognize when a component contributes to the performance of an action. Throughout the rest of this section we assume the labelled calculus.

3.2 Live components

To capture the fairness constraint for execution sequences, we need to define the *live components*. We now define $\mathsf{LC}(Q, A)$ as the set of live components of Q (when the execution of actions in A are prevented by the environment).

Definition 3.4 (*live components*)
Let $Q \in \mathsf{L}(\tilde{\mathbb{P}})$ and $A \subseteq \mathbb{A}$. The set $\mathsf{LC}(Q, A)$ is defined by induction on Q.

Var, Nil: $\mathsf{LC}(x_u, A) = \mathsf{LC}(\mathsf{nil}_u, A) = \emptyset$

Pref: $\mathsf{LC}(\mu_u.P, A) = \begin{cases} \{u\} & \text{if } \mu = \alpha \text{ or } \mu = \underline{\alpha} \text{ and } \alpha \notin A \\ \emptyset & \text{otherwise} \end{cases}$

Sum: $\mathsf{LC}(P_1 \oplus_u P_2, A) = \begin{cases} \{u\} & \text{if } \mathsf{LC}(P_1, A) \cup \mathsf{LC}(P_2, A) \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$
where $\oplus \in \{+, \underline{+}\}$

Par: $\mathsf{LC}(Q_1 \parallel^u_B Q_1, A) = \mathsf{LC}(Q_1, A \cup A') \cup \mathsf{LC}(Q_2, A \cup A'')$
where $A' = (\mathcal{A}(Q_1) \backslash \mathcal{A}(Q_2)) \cap B$ and $A'' = (\mathcal{A}(Q_2) \backslash \mathcal{A}(Q_1)) \cap B$

Rel: $\mathsf{LC}(Q[\Phi_u], A) = \mathsf{LC}(Q, \Phi^{-1}(A))$

Rec: $\mathsf{LC}(\mathsf{rec}\, x_u.Q, A) = \mathsf{LC}(Q, A)$

The *set of live components* in Q is defined as $\mathsf{LC}(Q, \emptyset)$ which we abbreviate to $\mathsf{LC}(Q)$.

An important subset of the live components of a process Q is the subset of urgent live components. Let $Q \in \mathsf{L}(\tilde{\mathbb{P}})$ and $A \subseteq \mathbb{A}$. The set $\mathsf{UC}(Q, A)$ is defined as in Definition 3.4 when $\mathsf{LC}(\_)$ is replaced by $\mathsf{UC}(\_)$ and rules Pref and Sum are replaced by the following one (again, define $\mathsf{UC}(Q) = \mathsf{UC}(Q, \emptyset)$):

Definition 3.5 (*urgent live components*)
Let $Q \in \mathsf{L}(\tilde{\mathbb{P}})$ and $A \subseteq \mathbb{A}$. The set $\mathsf{UC}(Q, A)$ is defined by induction on Q.

Pref: $\mathsf{UC}(\mu_u.P, A) = \begin{cases} \{u\} & \text{if } \mu = \underline{\alpha} \text{ and } \alpha \notin A \\ \emptyset & \text{otherwise} \end{cases}$

Sum: $\mathsf{UC}(P_1 +_u P_2, A) = \emptyset$

$\mathsf{UC}(P_1 \ \underline{+}_u P_2, A) = \begin{cases} \{u\} & \text{if } \mathsf{LC}(P_1, A) \cup \mathsf{LC}(P_2, A) \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$

Of course, $\mathsf{UC}(Q, A) \subseteq \mathsf{LC}(Q, A)$, for every Q and $A \subseteq \mathbb{A}$.

3.3 Fair execution sequences

Definition 3.6 (*fair execution sequences*)
Let $\gamma = P_0 \xrightarrow{\lambda_0} P_1 \xrightarrow{\lambda_1} \ldots$ be an execution sequence or a timed execution sequence; we will write '(timed) execution sequence' for such a sequence. We say that γ is *fair* if

$$\neg(\exists\, u\, \exists\, i\, .\, \forall\, k \geq i \ :\in \mathsf{LC}(P_k))$$

Following [6], we now present an alternative, more local, definition of fair computations which will be useful to prove our main statements.

Definition 3.7 (*B-step*)
For any process P_0, we say that $P_0 \xrightarrow{\lambda_0} P_1 \xrightarrow{\lambda_1} \ldots \xrightarrow{\lambda_{n-1}} P_n$ with $n > 0$ is a *timed B-step* when

i. B is a finite set of labels,

ii. $B \cap \mathsf{LC}(P_0) \cap \ldots \cap \mathsf{LC}(P_n) = \emptyset$.

If $\lambda_i \in \mathbb{A}_\tau$, $i = 0, \ldots, n-1$, then the sequence is a *B-step*. If $P_0 \xrightarrow{\lambda_0} P_1 \xrightarrow{\lambda_1} \ldots \xrightarrow{\lambda_{n-1}} P_n$ is a (timed) B-step and $v = \lambda_0 \ldots \lambda_{n-1}$ we write $P_0 \xrightarrow{v}_B P_{n+1}$; if $B = \mathsf{LC}(P_0)$, we also speak of a (timed) LC-step.

In particular, a (timed) LC-step from P is "locally" fair: all live events of P lose their liveness at some point in the step.

Definition 3.8 (*fair-step sequences*)
A *(timed) fair-step sequence* from P_0 is any maximal sequence of (timed) LC-steps of the form

$$P_0 \xrightarrow{v_0}_{\mathsf{LC}(P_0)} P_1 \xrightarrow{v_1}_{\mathsf{LC}(P_1)} \cdots$$

A fair-step sequence is simply a concatenation of locally fair steps. If δ is a (timed) fair-step sequence, then its *associated* (timed) execution sequence is the sequence which drops all references to the sets $\mathsf{LC}(P_i)$.

The following theorem shows that fair execution sequences and fair-step sequences are essentially the same and has been proven, as in [5, 6], with yet another, intermediate notion of local fairness (see appendix for more details).

Theorem 3.9 A (timed) execution sequence is fair if and only if it is the sequence associated with a (timed) fair-step sequence.

4 Fairness and Timing

This and the next section form the core of the paper. They relate fairness and timing in a process algebraic setting, and contain the two main contributions:

(i) We provide a characterization of fair execution sequences of *initial* PAFAS$^{\mathrm{C}}$ processes (PAFAS$^{\mathrm{C}}$ processes evolving only via functional operational semantics) in terms of *timed* execution sequences.

(ii) For the case of a finite state process, we derive from this a finite representation of the fair runs with a transition system that has arcs labelled by regular expressions.

The following propositions are key statements for proving our main results. They also provide some intuition on the reasons why fairness and (our notion of) timing are so strictly related.

Proposition 4.1 Let $P_0 \in \mathsf{L}(\mathbb{P}_1)$, $Q_0 = \mathsf{urgent}(P_0)$ and $v = \alpha_1 \ldots \alpha_n \in \mathbb{A}_\tau^*$. Then:

1. $P_0 \xrightarrow{v}_{\mathsf{LC}(P_0)} P_n$ implies $Q_0 \xrightarrow{v} P_n$;
2. $Q_0 \xrightarrow{v} Q_n$ and $\mathsf{UC}(Q_n) = \emptyset$ implies $P_0 \xrightarrow{v}_{\mathsf{LC}(P_0)} Q_n$.

Proposition 4.2 Let $P_0, P_1, P_2 \in \mathsf{L}(\tilde{\mathbb{P}}_1)$, v and $w \in (\mathbb{A}_\tau)^*$. Then:

1. $P_0 \xrightarrow{1} Q \xrightarrow{v} P_1$ implies $P_0 \xrightarrow{v}_{\mathsf{LC}(P_0)} P_1$;
2. $P_0 \xrightarrow{v} P_1 \xrightarrow{1} Q \xrightarrow{w} P_2$ implies $P_0 \xrightarrow{vw}_{\mathsf{LC}(P_0)} P_2$.

Proof:

1. $Q = \mathsf{urgent}(P_0)$, $Q \xrightarrow{v} P_1$ and $\mathsf{UC}(P_1) = \emptyset$ (by Lemma 9.5-3) imply $P_0 \xrightarrow{v}_{\mathsf{LC}(P_0)} P_1$ by Proposition 4.1-2.

2. This follows immediately from 1 and the definition of B-step.

□

The next statement shows that each everlasting timed execution sequence is fair.

Theorem 4.3 Each everlasting timed execution sequence of $P_0 \in \mathsf{L}(\mathbb{P}_1)$, i.e. each timed execution sequence of the form

$$\gamma = P_0 \xrightarrow{v_0} P_1 \xrightarrow{1} Q_1 \xrightarrow{v_1} P_2 \xrightarrow{1} Q_2 \xrightarrow{v_2} P_3 \xrightarrow{1} \dots$$

with infinitely many time steps and $v_0, v_1, v_2 \ldots \in (\mathbb{A}_\tau)^*$ is fair.

Proof: By Proposition 4.2 we have that $P_0 \xrightarrow{v_0 v_1}_{\mathsf{LC}(P_0)} P_2$, $P_2 \xrightarrow{v_2}_{\mathsf{LC}(P_2)} P_3$ and so on. Then γ is a sequence associated with a timed fair-step sequence and is fair by Theorem 3.9. □

4.1 Relating Timed Execution Sequences and Fair Execution

Our characterization results will be presented in two separate theorems where we distinguish between finite and infinite sequences of untimed systems. These results immediately carry over to fair execution sequences by Theorem 3.9. Furthermore, the timed execution sequences ignore labels, so they give indeed the announced characterizations with the simple filtering mechanism of requiring infinitely many time steps.

Theorem 4.4 Let $P_0 \in \mathsf{L}(\mathbb{P}_1)$ and $v_0, v_1, v_2 \ldots \in (\mathbb{A}_\tau)^*$. Then:

1. For any finite fair-step sequence from P_0

$$P_0 \xrightarrow{v_0}_{\mathsf{LC}(P_0)} P_1 \xrightarrow{v_1}_{\mathsf{LC}(P_1)} P_2 \dots P_{n-1} \xrightarrow{v_{n-1}}_{\mathsf{LC}(P_{n-1})} P_n$$

   there exists a timed execution sequence

$$P_0 \xrightarrow{1} Q_0 \xrightarrow{v_0} P_1 \xrightarrow{1} Q_1 \xrightarrow{v_1} P_2 \dots P_{n-1} \xrightarrow{1} Q_{n-1} \xrightarrow{v_{n-1}} P_n \xrightarrow{1} Q_n \xrightarrow{1} Q_n \dots$$

2. For any timed execution sequence from P_0

$$P_0 \xrightarrow{1} Q_0 \xrightarrow{v_0} P_1 \xrightarrow{1} Q_1 \xrightarrow{v_1} P_2 \dots P_{n-1} \xrightarrow{1} Q_{n-1} \xrightarrow{v_{n-1}} P_n \xrightarrow{1} Q_n \xrightarrow{1} Q_n \dots$$

   the following is a finite fair-step sequence:

$$P_0 \xrightarrow{v_0}_{\mathsf{LC}(P_0)} P_1 \xrightarrow{v_1}_{\mathsf{LC}(P_1)} P_2 \dots P_{n-1} \xrightarrow{v_{n-1}}_{\mathsf{LC}(P_{n-1})} P_n$$

Similarly, one can prove our characterization result for infinite sequences of untimed systems.

Theorem 4.5 Let $P_0 \in \mathsf{L}(\mathbb{P}_1)$ and $v_0, v_1, v_2 \ldots \in (\mathbb{A}_\tau)^*$. Then:

1. For any infinite fair-step sequence from P_0

$$P_0 \xrightarrow{v_0}_{\mathsf{LC}(P_0)} P_1 \xrightarrow{v_1}_{\mathsf{LC}(P_1)} P_2 \dots P_i \xrightarrow{v_i}_{\mathsf{LC}(P_i)} P_{i+1} \dots$$

   there exists a timed execution sequence

$$P_0 \xrightarrow{1} Q_0 \xrightarrow{v_0} P_1 \xrightarrow{1} Q_1 \xrightarrow{v_1} P_2 \dots P_i \xrightarrow{1} Q_i \xrightarrow{v_i} P_{i+1} \xrightarrow{1} Q_{i+1} \dots$$

2. For any timed execution sequence from P_0

$$P_0 \xrightarrow{1} Q_0 \xrightarrow{v_0} P_1 \xrightarrow{1} Q_1 \xrightarrow{v_1} P_2 \ldots P_i \xrightarrow{1} Q_i \xrightarrow{v_i} P_{i+1} \xrightarrow{1} Q_{i+1} \ldots$$

with infinitely many time steps, the following is a fair-step sequence:

$$P_0 \xrightarrow{v_0}_{\mathsf{LC}(P_0)} P_1 \xrightarrow{v_1}_{\mathsf{LC}(P_1)} P_2 \ldots P_i \xrightarrow{v_i}_{\mathsf{LC}(P_i)} P_{i+1} \ldots$$

5 Transition systems for fair execution sequences and finite state processes

We say that one process is *action-reachable* from another, if it can be reached according to the standard functional operational semantics, i.e. with transitions $\xrightarrow{\alpha}$. For an unlabelled initial process $P \in \mathbb{P}_1$ (i.e. a standard untimed process), we denote by $\mathcal{AT}(P)$ the set of processes action-reachable from P; we call P *finite state*, if $\mathcal{AT}(P)$ is finite.

For the definition of fair executions, we followed Costa and Stirling and introduced two semantic levels: one level (the positive) prescribes the finite and infinite execution sequences of labelled processes disregarding their fairness, while the other (the negative) filters out the unfair ones. The labels are notationally heavy, and keeping track of them is pretty involved. Since the labels evolve dynamically along computations, the transition system defined for the first level is in general infinite state even if the process without labels were finite state (namely if it has at least one infinite computation). Also the filtering mechanism is rather involved, since we have to check repeatedly what happens to live components along the computation, and for this we have to consider and compare the processes passed in the computation.

With the characterization results of the previous section, we have not only shown a conceptional relationship between timing (which is analogous to the timing as used in the PAFAS approach to the efficiency of asynchronous processes) and fairness. We have also given a much lighter description of the fair execution sequences of a process $P \in \mathbb{P}_1$ via the transition system of processes *time-reachable* (i.e. with transitions $\xrightarrow{\alpha}$ and $\xrightarrow{1}$) from P, which we will denote by $\mathcal{TT}(P)$: the marking of some actions and some choice-operators with underlines is easier than the labelling mechanism, and the filtering simply requires infinitely many time steps, i.e. non-Zeno behaviour; hence, for filtering one does not have to consider the processes passed. Moreover, we will show that the transition system $\mathcal{TT}(P)$ is finite for finite state processes.

Theorem 5.1 If $P \in \mathbb{P}_1$ is finite state, then $\mathcal{TT}(P)$ is finite.

Proof: It is easy to prove that for any process $P' \in \mathbb{P}_1$, there are only finitely many processes $Q \in \mathbb{P}$ with $\mathsf{unmark}(Q) = P'$; the intuitive reason is that Q only differs from P', since some prefixes and some choice operators are marked as urgent.

We will argue that, if Q is time-reachable from P, we have $\mathsf{unmark}(Q) = P'$ for some P' action-reachable from P. Then we are done, since by assumption of the theorem, there are only finitely many such P'. There are also only finitely many arcs in $\mathcal{TT}(P)$; note that, in particular due to the restriction on relabelling functions, our processes are sort-finite.

Assume that Q is time-reachable from P. Then we have to consider two possible cases:

- $P \xrightarrow{v} Q$ with $v \in (\mathbb{A}_\tau)^*$. In this case $Q \in \mathbb{P}_1$ action-reachable from P such that $\mathsf{unmark}(Q) = Q$ (by Proposition 8.6-2). We can choose $P' = Q$.

- $P \xrightarrow{v} P_1 \xrightarrow{1} Q_1 \xrightarrow{v_1} P_2 \ldots \xrightarrow{v_{k-1}} P_k \xrightarrow{1} Q_k \xrightarrow{v} Q$, where $v_i \in \mathbb{A}_\tau^*$, $Q_i = \mathsf{urgent}(P_i)$, for all $1 \leq i \leq k$, and $k \geq 1$ (here k is the number of time steps). Then $P_i = \mathsf{unmark}(Q_i)$ (by Lemma

8.7-1) and $Q_i \xrightarrow{v_i} P_{i+1}$ imply by Lemma 10.1-2 $P_i \xrightarrow{v_i} P_{i+1}$ for every $i \in [1, k-1]$ and, hence, P_k is action-reachable from P. Similarly, again by Lemma 10.1-2, $Q_k \xrightarrow{v} Q$ implies $P_k \xrightarrow{v} P'$ with $P' = \mathsf{unmark}(Q)$. Thus, P' is action-reachable from P_k and, hence, action-reachable from P.

□

The main result in [5, 6] is a characterization of fair execution sequences with only one (positive) level: SOS-rules are given that describe all transitions $P \xrightarrow{v} Q$ with $v \in (\mathbb{A}_\tau)^*$ such that $P \xrightarrow{v}_{\mathsf{LC}(P)} Q$. This is conceptionally very simple, since there is only one level and there is no labelling or marking of processes: the corresponding transition system for P only contains processes reachable from P. In particular, the transition system is finite-state if P is finite-state. The drawback is that, in general, P has infinitely many $\mathsf{LC}(P)$-steps (namely, if it has an infinite computation), and therefore the transition system has infinitely many arcs and is infinitely branching. (Observe that this drawback is not shared by our transition system of timed-reachable processes.)

As a second main result, we will now derive from $\mathcal{TT}(P)$ for a finite-state process P a finite transition system with finitely many arcs that describes the fair execution sequences in one level: the essential idea is that the arcs are inscribed with regular expressions (and not just with sequences as in [5, 6]); this idea has already been used for the analogous fairness of actions in [3], but only the construction here has a nice feature as explained below.

The states of the new transition system are the initial processes in $\mathcal{TT}(P)$, i.e. the states Q with $Q \xrightarrow{1} Q'$; if R is another such state, we have an arc from Q to R labelled with a regular expression e. This expression is obtained by taking $\mathcal{TT}(P)$ with Q' as initial state and R as the only final state, deleting all transitions $\xrightarrow{1}$ and applying the well-known Kleene construction to get an (equivalent) regular expression from a nondeterministic automaton. (The arc can be omitted, if e describes the empty set.) By Proposition 4.2.1, such an arc corresponds to a set of LC-steps which are also present in the one-level characterization of Costa and Stirling; vice versa, each LC-step is represented by such an arc by Proposition 4.1.1. There is one exception: if $Q' \xrightarrow{1}$, then $Q = Q'$ (by Proposition 8.11 in the appendix) and Q cannot perform any action; hence, there will only be an ε-labelled arc from Q to itself. With these loops, fair executions correspond to infinite paths in the new transition system, where we replace each e-labelled arc on the path by some v in the language of e. If we omit the loops, we can take maximal paths instead.

Note that, by definition of time step, the new transition system has only arcs $P \xrightarrow{e} Q$ such that P and Q are initial processes and for each v belonging to e one has $P \xrightarrow{v} Q$. This is a nice property that is not shared by the analogous construction in [3], which considers also states that are not initial. The property is achieved in particular by our specific treatment of recursion, where components in the body of a recursion can be urgent. (In [3] this is not the case; instead, function urgent unfolds a recursion.)

6 A Comparison with Fairness of Actions

We now give a formal comparison with our previous paper on fairness of actions. As shown in [3], fairness of actions is strictly related to a notion of timing that is used to evaluate the worst-case efficiency of processes e.g. in [4]. We have shown in [3] that each everlasting (or non-Zeno) timed process execution is action-fair and we have provided a characterization for action-fair executions of untimed processes in terms of timed process executions. This also results in a finite representation of action-fair executions using regular expressions, but the one we have presented in the preceding section gives a more faithful representation; we will discuss this in greater detail later on (see Observation 6.4).

The following discussion will explain the design decisions taken in the current work, the changes needed to the above mentioned timed operational semantics in order to capture fairness of components and the difficulties we had to overcome. Only with hindsight, we have been able to see that we found another rather natural timed semantics which intuitively attaches upper time bounds to components. The discussion will highlight the differences in the treatment for fairness of actions and fairness of components.

Before going into the details of the comparison we recall some basic definitions from [3]: the set of general process terms in PAFAS, their functional and timed operational semantics and the notion of live actions. Then we discuss the observations that lead to the characterization of fairness of components starting from the timed operational semantics that characterizes fairness of actions.

The set of initial timed process terms in PAFAS is the same as in PAFAS$^{\mathrm{c}}$ as given in Definition 2.1, while the set of general timed terms is given by the following grammar:

$$Q ::= \mathsf{nil} \mid x \mid \alpha.P \mid \underline{\alpha}.P \mid Q + Q \mid Q \|_A Q \mid Q[\Phi] \mid \mathsf{rec}\, x.P$$

where P denotes a generic initial process term. Note that within the scope of a prefix operator, we can only have an initial process term as in PAFAS$^{\mathrm{c}}$; different from PAFAS$^{\mathrm{c}}$, within the body of a recursive definition only initial process terms can be present, while choice allows general terms as arguments – and there is no urgent choice. The following subsections recall the functional and timed transitional semantics of process terms.

6.1 The functional behaviour of PAFAS process

The transitional semantics describing the functional behaviour of PAFAS processes indicates which basic actions they can perform; when performing an action, timing information can be disregarded, since we only have *upper* time bounds.

Definition 6.1 (*Functional operational semantics*) The following SOS-rules define the transition relations $\xrightarrow{\alpha}_a \subseteq (\tilde{\mathbb{P}} \times \tilde{\mathbb{P}})$ for $\alpha \in \mathbb{A}_\tau$, the *action transitions*; the index a indicates transitions for the original action-oriented PAFAS.

As usual, we write $Q \xrightarrow{\alpha}_a Q'$ if $(Q, Q') \in \xrightarrow{\alpha}_a$ and $Q \xrightarrow{\alpha}_a$ if there exists a $Q' \in \tilde{\mathbb{P}}$ such that $(Q, Q') \in \xrightarrow{\alpha}_a$, and similar conventions will apply later on.

$$\textsc{Pref}_{a1}\frac{}{\alpha.P \xrightarrow{\alpha}_a P} \qquad \textsc{Pref}_{a2}\frac{}{\underline{\alpha}.P \xrightarrow{\alpha}_a P}$$

$$\textsc{Sum}_a\frac{Q_1 \xrightarrow{\alpha}_a Q_1'}{Q_1 + Q_2 \xrightarrow{\alpha}_a Q_1'}$$

$$\textsc{Par}_{a1}\frac{\alpha \notin A,\ Q_1 \xrightarrow{\alpha}_a Q_1'}{Q_1 \|_A Q_2 \xrightarrow{\alpha}_a Q_1' \|_A Q_2} \qquad \textsc{Par}_{a2}\frac{\alpha \in A,\ Q_1 \xrightarrow{\alpha}_a Q_1',\ Q_2 \xrightarrow{\alpha}_a Q_2'}{Q_1 \|_A Q_2 \xrightarrow{\alpha}_a Q_1' \|_A Q_2'}$$

$$\textsc{Rel}_a\frac{Q \xrightarrow{\alpha}_a Q'}{Q[\Phi] \xrightarrow{\Phi(\alpha)}_a Q'[\Phi]}$$

$$\textsc{Rec}_a\frac{P\{\mathsf{rec}\, x.P/x\} \xrightarrow{\alpha}_a Q'}{\mathsf{rec}\, x.P \xrightarrow{\alpha}_a Q'}$$

Additionally, there are symmetric rules for Par_{a1} and Sum_a for actions of P_2.

6.2 The temporal behaviour of PAFAS process

In contrast to PAFAS$^{\mathrm{c}}$, where time steps are defined on initial process terms only using a function to determine the target, time steps in PAFAS are built up with SOS-rules for general terms, defining transitions like $Q \xrightarrow{X}_r Q'$ called *partial time steps*. The actions listed in X are not urgent; hence Q is justified in not performing them, but performing a time step instead. This time step is partial because it can occur only in contexts that can refuse the actions not in X. If $X = \mathbb{A}$, then Q is fully justified in performing this time step; i.e., Q can perform it independently of the environment. If $Q \xrightarrow{\mathbb{A}}_r Q'$ we write $Q \xrightarrow{1}_a Q'$ and say that Q performs a *(full) time step*. In [4], it is shown that refusal traces (arising from action transitions and partial time steps) characterize an efficiency preorder, which is intuitively justified by a testing scenario. For our comparison, we need partial time steps only to set up the following SOS-semantics; our real interest is in runs where all time steps are full.

Definition 6.2 (*Refusal transitional semantics*)
The following inference rules define $\xrightarrow{X}_r \subseteq (\tilde{\mathbb{P}} \times \tilde{\mathbb{P}})$, where $X \subseteq \mathbb{A}$.

$$\textsc{Nil}_r \frac{}{\mathsf{nil} \xrightarrow{X}_r \mathsf{nil}}$$

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

$$\textsc{Par}_r \frac{Q_i \xrightarrow{X_i}_r Q_i' \text{ for } i = 1,2,\ X \subseteq (A \cap (X_1 \cup X_2)) \cup (X_1 \cap X_2) \backslash A}{Q_1 \|_A Q_2 \xrightarrow{X}_r Q_1' \|_A Q_2'}$$

$$\textsc{Sum}_r \frac{\forall\, i = 1,2\ \ Q_i \xrightarrow{X}_r Q_i'}{Q_1 + Q_2 \xrightarrow{X}_r Q_1' + Q_2'}$$

$$\textsc{Rel}_r \frac{Q \xrightarrow{\Phi^{-1}(X \cup \{\tau\}) \backslash \{\tau\}}_r Q'}{Q[\Phi] \xrightarrow{X}_r Q'[\Phi]} \qquad \textsc{Rec}_r \frac{P\{\mathsf{rec}\ x.P/x\} \xrightarrow{X}_r Q'}{\mathsf{rec}\ x.P \xrightarrow{X}_r Q'}$$

The rules in Definition 6.2 explain the refusal operational semantics of a PAFAS term. Rule $\textsc{Pref}_{r1}$ says that a process $\alpha.P$ can let time pass and refuse to perform any action while rule $\textsc{Pref}_{r2}$ says that a process P prefixed by an urgent action $\underline{\alpha}$, can let time pass but action α cannot be refused. Process $\underline{\tau}.P$ cannot let time pass and cannot refuse any action; also in any context, $\underline{\tau}.P$ has to perform τ as explained by Rule $\textsc{Pref}_{a2}$ in Definition 6.1 before time can pass further.

Another rule worth noting is $\textsc{Par}_r$ which defines which actions a parallel composition can refuse during a time step. The intuition is that $Q_1 \|_A Q_2$ can refuse an action α if either $\alpha \notin A$ (Q_1 and Q_2 can perform α independently) and both Q_1 and Q_2 can refuse α, or $\alpha \in A$ (Q_1 and Q_2 are forced to synchronize on α) and at least one of Q_1 and Q_2 can refuse α, i.e. can delay it.

Thus, an action in a parallel composition is urgent (cannot be further delayed) only when all synchronizing 'local' actions are urgent. The other rules are as expected.

It is worth noting that a full time step in fairness of actions (action 1) plays the same role as a time step in fairness of components as presented in the previous sections.

The definitions of labelling of process terms, B-steps (actually based on LE instead of LC in that setting), and fair computations are the same as those presented in the previous sections. We report here the definition of live events (actions) that is quite different from the corresponding definition of live components in Definition 3.4.

The set of *labels* is still $\mathsf{LAB} = \{1,2\}^*$ with ε as the empty label but a performed action might correspond to a pair or more generally to a tuple of labels, namely if it is a synchronization of several action occurrences whose labels we collect. We use the following notation:

- (Set of tuples) $\mathcal{N} = \{\langle v_1, \ldots, v_n\rangle \mid n \geq 1,\ v_1, \ldots, v_n \in \mathsf{LAB}\}$;
- (Composition of tuples) $s_1 \times s_2 = \langle v_1, \ldots, v_n, w_1, \ldots, w_m\rangle$, where $s_1, s_2 \in \mathcal{N}$ and $s_1 = \langle v_1, \ldots, v_n\rangle$, $s_2 = \langle w_1, \ldots, w_m\rangle$;
- (Composition of sets of tuples) $N \times M = \{s_1 \times s_2 \mid s_1 \in N \text{ and } s_2 \in M\}$, where $N, M \subseteq \mathcal{N}$. Note that $N = \emptyset$ or $M = \emptyset$ implies $N \times M = \emptyset$.

We now define $\mathsf{LE}(P, A)$ as the set of live events of P when the execution of actions in A are prevented by the environment. For technical convenience we allow A to be a subset of $\mathbb{A}_\tau$.

Definition 6.3 (*live events*)
Let $P \in \mathsf{L}(\tilde{\mathbb{P}})$ and $A \subseteq \mathbb{A}_\tau$. The set $\mathsf{LE}(P, A)$ is defined by induction on P.

Var, Nil: $\mathsf{LE}(x_u, A) = \mathsf{LE}(\mathsf{nil}_u, A) = \emptyset$

Pref: $$\mathsf{LE}(\mu_u.P, A) = \begin{cases} \{\langle u\rangle\} & \text{if } \mu = \alpha \text{ or } \mu = \underline{\alpha} \text{ and } \alpha \notin A \\ \emptyset & \text{otherwise} \end{cases}$$

Sum: $\mathsf{LE}(P +_u Q, A) = \mathsf{LE}(P, A) \cup \mathsf{LE}(Q, A)$

Par: $\mathsf{LE}(P \parallel^u_B Q, A) = \mathsf{LE}(P, A \cup B) \cup \mathsf{LE}(Q, A \cup B) \cup$
$\bigcup_{\alpha \in B \setminus A}(\mathsf{LE}(P, \mathbb{A}_\tau \setminus \{\alpha\}) \times (\mathsf{LE}(Q, \mathbb{A}_\tau \setminus \{\alpha\})$

Rel: $\mathsf{LE}(P[\Phi_u], A) = \mathsf{LE}(P, \Phi^{-1}(A))$

Rec: $\mathsf{LE}(\mathsf{rec}\, x_u.P, A) = \mathsf{LE}(P, A)$

The *set of live events* in P is defined as $\mathsf{LE}(P, \emptyset)$ which we abbreviate to $\mathsf{LE}(P)$.

The intuition behind the rules is similar to that in Definition 3.4. Only note that to properly deal with synchronization, for all $\alpha \in B \setminus A$ we combine each live event of P corresponding to α with each live event of Q corresponding to α, getting *tuples of labels*. To filter out the appropriate events, we use a second parameter for LE which contains τ.

6.3 From Fairness of Actions to Fairness of Components

In this section, we will address the differences between the timed operational semantics for PAFAS (also suitable for capturing fair runs w.r.t. fairness of actions) and the timed operational semantics for PAFAS$^{\mathsf{C}}$ (also suitable for capturing fair runs w.r.t. fairness of components). We observe a number of differences, and each observation is followed by a discussion why this difference was necessary or at least appropriate.

We start with a fairly simple point regarding the treatment of recursive processes; in contrast to the differences we observe later, this difference has not been introduced for capturing fair runs in a component-oriented timed operational semantics, but for the representation of fair runs from the preceding section.

Observation 6.4 *(on unfoldings of recursive processes)*
If a recursive process performs a time step according to rule Rec_r in Definition 6.2, then recursion is unfolded once in the target state. This is not the case, when the target state is determined with urgent.

We opted for the new treatment of recursive processes because it allows a more faithful correspondence between timed executions and LC-steps. Proposition 4.2 shows: if an initial process performs a time step followed by a sequence of actions leading to a process that can perform an action 1, then the action sequence corresponds to an LC-step and vice versa. Note that the target states are exactly the same; this only works because of our treatment of the rec-operator that unfolds terms only when strictly needed (namely, when functional transitions are performed) and not also when time passes as in rule Rec_r. Indeed, in Proposition 4.7 in [3] (the counterpart of Proposition 4.2 for fairness of actions) the target states are not equal but only equal up to unfoldings of recursive processes.

As a consequence of the stronger result in Proposition 4.2 we also have a more faithful correspondence between timed executions and fair executions (see Theorems 4.4 and 4.5) and a more faithful finite representation in Section 5; as we already observed, in this representation there are only arcs $P \xrightarrow{e} Q$ such that P and Q are initial processes and for each v belonging to e one has $P \xrightarrow{v} Q$ – and this does not hold for the corresponding representation in [3].

In the rest of this section, we will discuss PAFAS-transitions as if time steps in PAFAS would treat recursion as in PAFAS$^{\mathrm{C}}$; e.g. we pretend that also in PAFAS $\mathsf{rec}x.a.x+b.\mathsf{nil} \xrightarrow{1} \mathsf{rec}x.\underline{a}.x+\underline{b}.\mathsf{nil}$,[3] and we hope that this is most suitable for readers of the present paper.

Observation 6.5 *(on urgent synchronization)*
Rule PAR_r *in Definition 6.2 defines which actions a parallel composition can refuse during a time step. We already stated that, according to this rule, a synchronizing action in a parallel composition is urgent (cannot be further delayed) only when all synchronizing 'local' actions (i.e. action occurrences in the term) are urgent. Consequently,* $\mathsf{rec}\, x.(a.x + b.\mathsf{nil}) \parallel_{\{b\}} \underline{b}.\mathsf{nil}$ *can perform a time step in PAFAS, while it cannot in PAFAS*$^{\mathrm{C}}$.

To understand this difference, consider process $(\mathsf{rec}\ x_1.a_{111}.x_{1111} +_{11} b_{112}.\mathsf{nil}) \parallel^{\varepsilon}_{\{b\}} b_2.\mathsf{nil}$ and computation:

$(\mathsf{rec}\, x_1.a_{111}.x_{1111} +_{11} b_{112}.\mathsf{nil}) \parallel^{\varepsilon}_{\{b\}} b_2.\mathsf{nil} \xrightarrow{a}$
$(\mathsf{rec}\, x_v.a_{v11}.x_{v111} +_{v1} b_{v12}.\mathsf{nil}) \parallel^{\varepsilon}_{\{b\}} b_2.\mathsf{nil} \xrightarrow{a}$
$(\mathsf{rec}\, x_u.a_{u11}.x_{u111} +_{u1} b_{u12}.\mathsf{nil}) \parallel^{\varepsilon}_{\{b\}} b_2.\mathsf{nil} \xrightarrow{a} \ldots$

where $v = 1111$ and $u = v111$. In PAFAS, the label for a synchronized *action* b, is a pair of labels, each stemming from one of the components. Since the first component offers a different local b at each step, these pairs change: we have $< 112, 2 >$ for the first process, $< v12, 2 >$ for the second process and $< u12, 2 >$ for the third one etc. Each such pair is a live event (according to Definition 6.3), but since no event remains live throughout the computation, the above is a fair execution sequence in PAFAS.

From the point of view of components, the second component remains live throughout the computation, so the latter is not fair in PAFAS$^{\mathrm{C}}$.

In the timed operational semantics of PAFAS (where only actions can be urgent), we have the corresponding run:

$\mathsf{rec}\ x.\ (a.x + b.\mathsf{nil}) \parallel_{\{b\}} b.\mathsf{nil} \xrightarrow{1}$
$\mathsf{rec}\ x.\ (\underline{a}.x + \underline{b}.\mathsf{nil}) \parallel_{\{b\}} \underline{b}.\mathsf{nil} \xrightarrow{a}$
$\mathsf{rec}\ x.\ (a.x + b.\mathsf{nil}) \parallel_{\{b\}} \underline{b}.\mathsf{nil} \xrightarrow{1}$
$\mathsf{rec}\ x.\ (\underline{a}.x + \underline{b}.\mathsf{nil}) \parallel_{\{b\}} \underline{b}.\mathsf{nil} \xrightarrow{a} \ldots$

Since this timed run has infinitely many time steps, we can see from it that the above computation is fair in labelled PAFAS. Since fairness does not hold in labelled PAFAS$^{\mathrm{C}}$, we must have different rules for time steps; crucial is the second time step we already mentioned above: in

[3] and not $\mathsf{rec}\ x.a.x + b.\mathsf{nil} \xrightarrow{1} \underline{a}.(\mathsf{rec}\ x.a.x + b.\mathsf{nil}) + \underline{b}.\mathsf{nil}$

PAFAS$^{\mathsf{c}}$, we have to consider the second component as urgent, in order to disallow the second time step; i.e. the synchronized b is urgent just because one local b is urgent.

In our presentation of PAFAS$^{\mathsf{c}}$, this idea is realized in Definition 2.5 that allows 1 only when the source process is an initial one and hence does not have urgencies in its body. We explain next why this seems most appropriate.

Observation 6.6 *(on the disappearance of urgent components)*
If an action is marked as urgent in PAFAS, it remains urgent until it is performed; this is not true in PAFAS$^{\mathsf{c}}$; cf. the application of clean *in rules Par_{a1} and Par_{a2} of Definition 2.4, which unmarks a component that was activated and declared as urgent and has now become inactive in the evolution of the system via functional transitions.*

Consider, for instance, the following sequence of transitions, which we would get according to the ideas developed so far:

$\mathsf{rec}\ x.\ (a.c.x + b.\mathsf{nil}) \parallel_{\{b\}} b.\mathsf{nil} \xrightarrow{1}$
$\mathsf{rec}\ x.\ (\underline{a}.c.x + \underline{b}.\mathsf{nil}) \parallel_{\{b\}} \underline{b}.\mathsf{nil} \xrightarrow{a}$
$c.(\mathsf{rec}\ x.\ (a.c.x + b.\mathsf{nil})) \parallel_{\{b\}} \underline{b}.\mathsf{nil} = Q \xrightarrow{c}$
$\mathsf{rec}\ x.\ (a.c.x + b.\mathsf{nil}) \parallel_{\{b\}} \underline{b}.\mathsf{nil} = R \xrightarrow{a}$
$c.(\mathsf{rec}\ x.\ (a.c.x + b.\mathsf{nil})) \parallel_{\{b\}} \underline{b}.\mathsf{nil} \xrightarrow{c} \ldots$

After the first a-transition, the first component has performed an action, while the second is disabled; repeating this observation, it should be clear that the action transitions in this timed run give a fair computation (– in labelled PAFAS$^{\mathsf{c}}$, but we will again leave labels implicit in the rest of the section). Also, this observation shows that reaching Q we have made as much local progress w.r.t. to component-fairness as one can expect, so a time step should be possible for Q and – since no time step actually is performed – for every succeeding process as e.g. R.

According to Definition 2.5, Q cannot perform a time step; a possibility would be to change this definition and allow a time step if no urgent local action can be performed. With this change, we would have $Q \xrightarrow{1}$, but we still would not have $R \xrightarrow{1}$. This should make it plausible that, when a functional transition is performed, all inactive urgencies must be removed in the target state, as it is done in PAFAS$^{\mathsf{c}}$ with clean.

To argue more forcefully for this approach, consider the modified rule for time steps of the previous paragraph and the more complicated example, where $P = \mathsf{rec}\ x.(a.(c.x + d.\mathsf{nil}) + b.\mathsf{nil})$:

$(P \parallel_{\{b\}} b.\mathsf{nil}) \parallel_{\{d\}} d.\mathsf{nil} \xrightarrow{1}$
$((\mathsf{rec}\ x.(\underline{a}.(c.x + d.\mathsf{nil}) + \underline{b}.\mathsf{nil})) \parallel_{\{b\}} \underline{b}.\mathsf{nil}) \parallel_{\{d\}} \underline{d}.\mathsf{nil} \xrightarrow{a}$
$((c.P + d.\mathsf{nil}) \parallel_{\{b\}} \underline{b}.\mathsf{nil}) \parallel_{\{d\}} \underline{d}.\mathsf{nil} \xrightarrow{c}$
$(P \parallel_{\{b\}} \underline{b}.\mathsf{nil}) \parallel_{\{d\}} \underline{d}.\mathsf{nil} \xrightarrow{a}$
$((c.P + d.\mathsf{nil}) \parallel_{\{b\}} \underline{b}.\mathsf{nil}) \parallel_{\{d\}} \underline{d}.\mathsf{nil} \xrightarrow{c} \ldots$

Again, the action transitions in this run give a component-fair computation, since the first component acts repeatedly while the other two are disabled repeatedly. But no state after the first time step would allow another time step, even with the modification under discussion; so cleaning is a necessity. It is also very intuitive that a disabled component cannot be urgent.

With this intuition, it is actually not so much surprising that we use clean in PAFAS$^{\mathsf{c}}$ – but it *is* surprising that nothing like that is needed in PAFAS.

Observation 6.7 *(urgent components after a time step)*
According to the timed operational semantics in Definition 6.2, all initial actions in a parallel composition are made urgent when a full time step is performed, for instance:

$$a.b.\mathsf{nil} \parallel_{\{b\}} b.\mathsf{nil} \xrightarrow{1} P = \underline{a}.b.\mathsf{nil} \parallel_{\{b\}} \underline{b}.\mathsf{nil}$$

This is not the case in PAFAS^c, where according to the definition of urgent *we have that* $a.b.\mathsf{nil} \parallel_{\{b\}} b.\mathsf{nil} \xrightarrow{1} P' = \underline{a}.b.\mathsf{nil} \parallel_{\{b\}} b.\mathsf{nil}$.

This is just a uniform application of the intuition we have just discussed: since component $b.\mathsf{nil}$ is not enabled, it should not be urgent in PAFAS$^{\mathsf{c}}$. Even if this consequence may not be strictly necessary to capture fair runs, it certainly is a clean way to proceed. It is also of advantage for the technical proof, which relies on a match between action sequences between two time steps and LC-steps: we can continue the above PAFAS transition sequence with

$$P \xrightarrow{a} Q = b.\mathsf{nil} \parallel_{\{b\}} \underline{b}.\mathsf{nil}.$$

Since in P only the first component is live, the only live component acts in this transition, which therefore is an LC-step; but Q does not allow a time step according to PAFAS$^{\mathsf{c}}$. In contrast, with $P' \xrightarrow{a} b.\mathsf{nil} \parallel_{\{b\}} b.\mathsf{nil} \xrightarrow{1}$ things fit together nicely.

Observation 6.8 *(on urgent nondeterministic choice)*
In PAFAS processes, only actions are marked as urgent, while also the choice-operator can be marked as urgent in PAFAS^c; e.g. $b.\mathsf{nil} + d.\mathsf{nil} \xrightarrow{1} \underline{b}.\mathsf{nil} + \underline{d}.\mathsf{nil}$ *in PAFAS, while in PAFAS^c we have* $b.\mathsf{nil} + d.\mathsf{nil} \xrightarrow{1} b.\mathsf{nil} \underline{+} d.\mathsf{nil}$.

So far, the discussion concentrated on the urgency of parallel components that are prefix terms; such a component loses its urgency together with the action in its prefix when this action is not enabled. Thus, according to our considerations so far, we would have the following transition sequence in PAFAS$^{\mathsf{c}}$:

$(\mathsf{rec}\, x.(a.(c.x + d.\mathsf{nil}) + b.\mathsf{nil})) \parallel_{\{b,d\}} (b.\mathsf{nil} + d.\mathsf{nil}) \xrightarrow{1}$
$(\mathsf{rec}\, x.(\underline{a}.(c.x + d.\mathsf{nil}) + \underline{b}.\mathsf{nil})) \parallel_{\{b,d\}} (\underline{b}.\mathsf{nil} + d.\mathsf{nil}) \xrightarrow{a}$
$((c.(\mathsf{rec}\, x.a.(c.x + d.\mathsf{nil}) + b.\mathsf{nil}) + d.\mathsf{nil}) \parallel_{\{b,d\}} (b.\mathsf{nil} + d.\mathsf{nil}) \xrightarrow{c}$
$((\mathsf{rec}\, x.a.(c.x + d.\mathsf{nil}) + b.\mathsf{nil}) \parallel_{\{b,d\}} (b.\mathsf{nil} + d.\mathsf{nil}) \xrightarrow{1} \ldots$

After the first time step, action d is not urgent in the second component since it is not enabled. For the same reason, action b in the second component loses its urgency after the a-transition. Hence, according to the characterization result we wanted to achieve, we would have to consider an infinite repetition of ac as a fair run. But actually, the second component remains live all through such a run, since in each state one of the actions it offers is live; this run is certainly not fair for components.

The most appropriate notational consequence is to mark the whole component as urgent – and to stick with the intuition that a component loses its urgency if the whole component is disabled. Now the infinite transition sequence from above never allows a second time step since it looks like this:

$(\mathsf{rec}\, x.(a.(c.x + d.\mathsf{nil}) + b.\mathsf{nil})) \parallel_{\{b,d\}} (b.\mathsf{nil} + d.\mathsf{nil}) \xrightarrow{1}$
$(\mathsf{rec}\, x.(a.(c.x + d.\mathsf{nil}) \underline{+} b.\mathsf{nil})) \parallel_{\{b,d\}} (b.\mathsf{nil} \underline{+} d.\mathsf{nil}) \xrightarrow{a}$
$(c.(\mathsf{rec}\, x.(a.(c.x + d.\mathsf{nil}) + b.\mathsf{nil})) + d.\mathsf{nil}) \parallel_{\{b,d\}} (b.\mathsf{nil} \underline{+} d.\mathsf{nil}) \xrightarrow{c}$
$(\mathsf{rec}\, x.(a.(c.x + d.\mathsf{nil}) + b.\mathsf{nil})) \parallel_{\{b,d\}} (b.\mathsf{nil} \underline{+} d.\mathsf{nil}) \xrightarrow{a}$
$(c.(\mathsf{rec}\, x.(a.(c.x + d.\mathsf{nil}) + b.\mathsf{nil})) + d.\mathsf{nil}) \parallel_{\{b,d\}} (b.\mathsf{nil} \underline{+} d.\mathsf{nil}) \xrightarrow{c} \ldots$

According to this treatment of urgency, the PAFAS-definition of live events sets is $\mathsf{LE}(P +_u Q) = \mathsf{LE}(P) \cup \mathsf{LE}(Q)$, while for PAFAS$^{\mathsf{c}}$ we find in the definition of live components that

$$\mathsf{LC}(P_1 +_u P_2) = \begin{cases} \{u\} & \text{if } \mathsf{LC}(P_1) \cup \mathsf{LC}(P_2) \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$$

So only the complete choice is considered as live.

Having developed descriptions of action- and of component-fair runs in process algebra that are much easier to use, we are currently studying how to apply our approaches to prove e.g. liveness of MUTEX-solutions. First considerations have shown that action-fairness can easily be too weak for this purpose, while component-fairness looks promising.

A definition of fair behavior different from those in [5, 6] can be found in [1]; it would be interesting to find a similar characterization for this definition.

References

[1] S. Brookes. Traces, Pomsets, Fairness and Full Abstractions for Communicating Processes. Proc. of *Concur'02*, Lect. Notes Comp. Sci. 2421, pp. 466-482, Springer 2002.

[2] F. Corradini, M.R. Di Berardini and W. Vogler. PAFAS at work: Comparing the Worst-Case Efficiency of Three Buffer Implementations. Proc. of 2nd Asia-Pacific Conference on Quality Software, APAQS 2001, pp. 231-240, IEEE, 2001.

[3] F. Corradini, M.R. Di Berardini and W. Vogler. Relating Fairness and Timing in Process Algebras. Proc. of *Concur'03*, Lect. Notes Comp. Sci. 2761, pp. 446-460, Springer 2003.

[4] F. Corradini, W. Vogler and L. Jenner. Comparing the Worst-Case Efficiency of Asynchronous Systems with PAFAS. *Acta Informatica* **38**, pp. 735-792, 2002.

[5] G. Costa, C. Stirling. A Fair Calculus of Communicating Systems. *Acta Informatica* **21**, pp. 417-441, 1984.

[6] G. Costa, C. Stirling. Weak and Strong Fairness in CCS. *Information and Computation* **73**, pp. 207-244, 1987.

[7] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[8] G. Lüttgen and W. Vogler. Bisimulation on speed: Worst–case efficiency. *Information and Computation* **191**, pp. 105-144, 2004.

[9] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, 1996.

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

[11] W. Vogler. Efficiency of asynchronous systems, read arcs, and the MUTEX-problem. *Theoret. Comput. Sci.* 275, pp. 589-631, 2002.

7 Appendix: An intermediate notion of local fairness

As in [6], we now present an itermediate, more local definition of fair computations that allows us to think of fairness in term of localizable property and not just as a property of (timed) execution sequences as a whole. In the following, we use $|\gamma|$ to denote the length – i.e. the number of processes – of a (timed) execution sequence γ, which is ∞ if γ is an infinite computation.

Definition 7.1 Let $\gamma = P_0 \xrightarrow{\lambda_0} P_1 \xrightarrow{\lambda_1} \dots$ be a (timed) execution sequence. We say that

i. γ is *l-fair* at i if there exists $j \geq i$ such that

$$\mathsf{LC}(P_i) \cap \mathsf{LC}(P_{i+1}) \cap \ldots \cap \mathsf{LC}(P_j) = \emptyset$$

ii. γ is *l-fair* if for all $i < |\gamma|$ we have that γ is l-fair at i.

γ is *l-fair* at i when every live event in P_i loses its liveness. Starting from P_0 we can generate a derivation $P_0 \xrightarrow{\lambda_0} P_1 \ldots \xrightarrow{\lambda_{n-1}} P_n$ which satisfies l-fairness at 0, i.e. such that $\mathsf{LC}(P_0) \cap \mathsf{LC}(P_1) \cap \ldots \cap \mathsf{LC}(P_n) = \emptyset$. One then continues by generating a derivation $P_n \xrightarrow{\lambda_n} P_{n+1} \ldots \xrightarrow{\lambda_{m-1}} P_m$ which satisfies l-fairness at n. The concatenation of these two derivations guarantees l-fairness at any $j \leq n$. In this way we can generate only fair sequences. The following theorem formalizes this strategy.

Theorem 7.2 A (timed) execution sequence $\gamma = P_0 \xrightarrow{\lambda_0} P_1 \xrightarrow{\lambda_1} \dots$ is l-fair if and only if it is fair.

Proof: We prove that γ is not l-fair if and only if it is not fair.

1. Assume γ not l-fair. There exists $i < |\gamma|$ such that, for all $j \geq i$, $\mathsf{LC}(P_i) \cap \mathsf{LC}(P_{i+1}) \cap \ldots \cap \mathsf{LC}(P_j) \neq \emptyset$. Since $\mathsf{LC}(P_i)$ is finite, there is some u such that for all $j \geq i$, $u \in \mathsf{LC}(P_j)$.
2. Vice versa, if γ is not fair, then there are i and u such that, for all $k \geq i$, $u \in \mathsf{LC}(P_k)$, i.e. γ is not l-fair at i and, hence, γ is not l-fair.

□

The following theorem relates l-fair execution sequences and fair-step sequences.

Theorem 7.3 A (timed) execution sequence is l-fair if and only if it is the sequence associated with a (timed) fair-step sequence.

Proof: Assume $\gamma = P_0 \xrightarrow{\lambda_0} P_1 \xrightarrow{\lambda_1} \dots$ l-fair. By definition, γ is l-fair at 0 and, hence, there exists $j \geq 0$ such that $\mathsf{LC}(P_0) \cap \ldots \cap \mathsf{LC}(P_j) = \emptyset$. Then, for $v_0 = \lambda_0 \ldots \lambda_{j-1}$, we have that $P_0 \xrightarrow{v_0}_{\mathsf{LC}(P_0)} P_j$. Since γ is l-fair at j for any $j < |\gamma|$, we can iterate this strategy and generate a fair-step sequence $\gamma' = P_0 \xrightarrow{v_0}_{\mathsf{LC}(P_0)} P_j \xrightarrow{v_1}_{\mathsf{LC}(P_j)} \dots$ Clearly, γ is the (timed) execution sequence associated with γ'.

Vice versa, assume that $\gamma' = P_0 \xrightarrow{v_0}_{\mathsf{LC}(P_0)} P_1 \xrightarrow{v_1}_{\mathsf{LC}(P_1)} \dots$ is a fair-step sequence. Let $v_0 = \lambda_0 \ldots \lambda_{j-1}$, $v_1 = \lambda_j \ldots \lambda_{k-1}$ and so on. The (timed) execution sequence associated with γ' is
$\gamma = P_0 \xrightarrow{\lambda_0} P'_1 \xrightarrow{\lambda_1} \dots \xrightarrow{\lambda_{j-1}} P'_j = P_1 \xrightarrow{\lambda_j} P'_{j+1} \xrightarrow{\lambda_{j+1}} \dots \xrightarrow{\lambda_{k-1}} P'_k = P_2 \dots$

The definition of a B-step implies that γ is l-fair at 0, j and k etc. and, thus, fair at any $i \leq j$ and $i \leq k$ etc.; therefore, γ is l-fair for any $i < |\gamma|$ and, hence, it is l-fair. □

Theorem 3.9 A (timed) execution sequence is fair if and only if it is the sequence associated with a (timed) fair-step sequence.

Proof: It follows directly from Theorems 7.2 and 7.3. □

8 Appendix A: Some Useful Properties

In this appendix section we state and prove some useful properties relating some of the notions in the main body of the paper. They are not related to each other but are useful to prove the main statements.

8.1 Closure properties of $\tilde{\mathbb{P}}_1$

Proposition 8.1 Let $P, P' \in \tilde{\mathbb{P}}_1$. Then

1. $P\{P'/x\} \in \tilde{\mathbb{P}}_1$;
2. $P \xrightarrow{\alpha} Q$ implies $Q \in \tilde{\mathbb{P}}_1$.

Proof: Both statements can be easily proved by induction on $P \in \tilde{\mathbb{P}}_1$. □

8.2 Properties of $\mathcal{A}(Q, A)$

Lemma 8.2 Let $Q \in \tilde{\mathbb{P}}$, A and $A' \subseteq \mathbb{A}$. Then:

1. $\beta \in \mathcal{A}(Q, A)$ implies $\beta \notin A$;
2. $\beta \in \mathcal{A}(Q, A)$ and $\beta \notin A'$ implies $\beta \in \mathcal{A}(Q, A')$;
3. $A \subseteq A'$ implies $\mathcal{A}(Q, A') \subseteq \mathcal{A}(Q, A)$;
4. $\mathcal{A}(Q, A) = \mathcal{A}(Q) \backslash A$.

Proof: First, we prove Item 1 and Item 2 together by induction on $Q \in \tilde{\mathbb{P}}$.

Nil, Var: $Q = \mathsf{nil}$, $Q = x$. These cases are not possible since $\mathcal{A}(Q, A) = \emptyset$ for all A.

Pref: $Q = \alpha.P_1$ or $Q = \underline{\alpha}.P_1$. In both cases $\beta \in \mathcal{A}(Q, A)$ implies $\beta = \alpha \notin A$. Then:

1. clearly $\beta \notin A$.
2. $\beta \notin A'$ implies $\alpha \notin A'$ and, hence, $\beta \in \mathcal{A}(Q, A') = \{\alpha\}$.

Sum: $Q = P_1 + P_2$ or $Q = P_1 \underline{+} P_2$. In both cases $\beta \in \mathcal{A}(Q, A)$ implies either (i) $\beta \in \mathcal{A}(P_1, A)$ or (ii) $\beta \in \mathcal{A}(P_1, A)$. Consider case (i) – case (ii) can be proved similarly. By induction hypothesis:

1. $\beta \in \mathcal{A}(P_1, A)$ implies $\beta \notin A$.
2. $\beta \in \mathcal{A}(P_1, A)$ and $\beta \notin A'$ implies $\beta \in \mathcal{A}(P_1, A') \subseteq \mathcal{A}(Q, A')$.

Par: $Q = Q_1 \parallel_B Q_2$. Let $A_1 = (\mathcal{A}(Q_1) \backslash \mathcal{A}(Q_2)) \cap B$ and $A_2 = (\mathcal{A}(Q_2) \backslash \mathcal{A}(Q_1)) \cap B$. In this case $\beta \in \mathcal{A}(Q, A)$ implies either (i) $\beta \in \mathcal{A}(Q_1, A \cup A_1)$ or (ii) $\beta \in \mathcal{A}(Q_2, A \cup A_2)$. Consider the (i)-case (the (ii)-case is symmetric).

1. $\beta \in \mathcal{A}(Q_1, A \cup A_1)$ implies, by induction hypothesis, $\beta \notin A \cup A_1$ and, hence, $\beta \notin A$.
2. $\beta \in \mathcal{A}(Q_1, A \cup A_1)$ implies, by Item 1, $\beta \notin A \cup A_1$ and, hence, $\beta \notin A_1$. Assume, now, $\beta \notin A'$. Then $\beta \in \mathcal{A}(Q_1, A \cup A_1)$ and $\beta \notin A' \cup A_1$ implies, by induction hypothesis $\beta \in \mathcal{A}(Q_1, A' \cup A_1) \subseteq \mathcal{A}(Q, A')$.

Rel: $Q = Q_1[\Phi]$. In this case we have that $\beta \in \mathcal{A}(Q, A)$ if there exists $\beta' \in \Phi^{-1}(\beta)$ such that $\beta' \in \mathcal{A}(Q_1, \Phi^{-1}(A))$.

1. $\beta' \in \mathcal{A}(Q_1, \Phi^{-1}(A))$ implies, by induction hypothesis, $\beta' \notin \Phi^{-1}(A)$ and, hence $\beta = \Phi(\beta') \notin A$.
2. Assume $\beta \notin A'$ and, hence, $\beta' \notin \Phi^{-1}(A')$. Again by induction hypothesis $\beta' \in \mathcal{A}(Q_1, \Phi^{-1}(A))$ and $\beta' \notin \Phi^{-1}(A')$ implies $\beta' \in \mathcal{A}(Q_1, \Phi^{-1}(A'))$ and, hence, $\beta = \Phi(\beta') \in \Phi(\mathcal{A}(Q_1, \Phi^{-1}(A'))) = \mathcal{A}(Q, A')$.

Rec: $Q = \mathsf{rec}\, x.Q_1$. We have that $\beta \in \mathcal{A}(Q, A)$ implies $\beta \in \mathcal{A}(Q_1, A)$.

1. by induction hypothesis $\beta \notin A$.
2. Assume $\beta \notin A'$. By induction hypothesis $\beta \in \mathcal{A}(Q_1, A)$ and $\beta \notin A'$ implies $\beta \in \mathcal{A}(Q_1, A') = \mathcal{A}(Q, A')$.

Items 3. and 4. can be proved as follows.

3. Assume $A \subseteq A'$ and $\alpha \in \mathcal{A}(Q, A')$. Item 1 and $\alpha \in \mathcal{A}(Q, A')$ implies $\alpha \notin A'$ and, hence, $\alpha \notin A$. Now, by Item 2, $\alpha \in \mathcal{A}(Q, A')$ and $\alpha \notin A$ implies $\alpha \in \mathcal{A}(Q, A)$.

4. Assume $\alpha \in \mathcal{A}(Q, A)$. Item 1 implies $\alpha \notin A$. Moreover, since trivially $\alpha \notin \emptyset$, by Item 2 we have $\alpha \in \mathcal{A}(Q)$ and thus $\alpha \in \mathcal{A}(Q) \backslash A$.

   Now let $\alpha \in \mathcal{A}(Q) \backslash A$. Again by Item 2, $\alpha \in \mathcal{A}(Q)$ and $\alpha \notin A$ implies $\alpha \in \mathcal{A}(Q, A)$.

□

8.3 Properties of urgent, clean and unmark

First we report the formal definition of function $\mathsf{unmark}(Q)$.

Definition 8.3 (*cleaning all urgencies*)
Let Q be a $\tilde{\mathbb{P}}$ term. Then $\mathsf{unmark}(Q)$ is defined by induction on Q as follows:

Nil, Var: $\mathsf{unmark}(\mathsf{nil}) = \mathsf{nil}, \qquad \mathsf{unmark}(x) = x$

Pref: $\mathsf{unmark}(\alpha.P) = \mathsf{unmark}(\underline{\alpha}.P) = \alpha.P$

Sum: $\mathsf{unmark}(P_1 + P_2) = \mathsf{unmark}(P_1 \underline{+} P_2) = P_1 + P_2$

Par: $\mathsf{unmark}(Q_1 \parallel_B Q_2) = \mathsf{unmark}(Q_1) \parallel_B \mathsf{unmark}(Q_2)$

Rel: $\mathsf{unmark}(Q[\Phi]) = \mathsf{unmark}(Q)[\Phi]$

Rec: $\mathsf{unmark}(\mathsf{rec}\, x.Q) = \mathsf{rec}\, x.\, \mathsf{unmark}(Q)$

A significant subset of activated actions is the set of urgent ones. These are activated actions that cannot let time pass. They are used to understand if processes time-reachable from terms in $\tilde{\mathbb{P}}_1$ are initial or not.

Definition 8.4 (*urgent activated basic actions*)
Let $Q \in \tilde{\mathbb{P}}$ and $A \subseteq \mathbb{A}$. The set $\mathcal{U}(Q, A)$ is defined as in Definition 2.2 when $\mathcal{A}(\_)$ is replaced by $\mathcal{U}(\_)$ and the Pref and Sum rules are replaced as follows:

Pref: $\mathcal{U}(\alpha.P, A) = \emptyset$

$$\mathcal{U}(\underline{\alpha}.P, A) = \begin{cases} \{\alpha\} & \text{if } \alpha \notin A \\ \emptyset & \text{otherwise} \end{cases}$$

Sum: $\mathcal{U}(P_1 + P_2, A) = \emptyset$

$$\mathcal{U}(P_1 \underline{+} P_2, A) = \mathcal{A}(P_1, A) + \mathcal{A}(P_2, A)$$

$\mathcal{U}(Q)$ abbreviates $\mathcal{U}(Q, \emptyset)$ and is called the set of *urgent activated actions* of Q.

We can state the following properties:

Lemma 8.5 Let $Q \in \tilde{\mathbb{P}}$ and $A \subseteq \mathbb{A}$. Then:

1. $\mathcal{U}(Q, A) \subseteq \mathcal{A}(Q, A)$;
2. $Q \in \tilde{\mathbb{P}}_1$ implies $\mathcal{U}(Q, A) = \emptyset$.

Proof: Both Items can be easily proved by induction on Q. □

Now we prove some useful properties of functions clean and unmark. Most of them are stated for terms in $\tilde{\mathbb{P}}$ but, since the "action" of removing urgencies does not depend from labels we can easily prove that they also hold for terms in $\mathsf{L}(\tilde{\mathbb{P}})$.

Lemma 8.6 Let $Q \in \tilde{\mathbb{P}}$ and $A, A' \subseteq \mathbb{A}$. Then:

1. $\mathsf{unmark}(Q) \in \tilde{\mathbb{P}}_1$;
2. $Q \in \tilde{\mathbb{P}}_1$ implies $\mathsf{unmark}(Q) = Q$ and $\mathsf{clean}(Q, A) = Q$;
3. $\mathsf{unmark}(\mathsf{clean}(Q, A)) = \mathsf{unmark}(Q)$;
4. $\mathcal{A}(\mathsf{unmark}(Q), A') = \mathcal{A}(\mathsf{clean}(Q, A), A') = \mathcal{A}(Q, A')$.

Proof: All Items follow directly from Definitions 2.3 and 8.3. □

Lemma 8.7 Let $P \in \tilde{\mathbb{P}}_1$, $Q \in \tilde{\mathbb{P}}$ and $A \subseteq \mathbb{A}$ such that $Q = \mathsf{urgent}(P, A)$. Then:

1. $\mathsf{unmark}(Q) = P$;
2. $\mathcal{A}(P, A') = \mathcal{A}(Q, A')$ for all $A' \subseteq \mathbb{A}$;
3. $Q \in \tilde{\mathbb{P}}_1$ implies $Q = P$;
4. $Q = P$ iff $\mathcal{A}(P, A) = \emptyset$;
5. $\mathsf{clean}(Q, A) = Q$;
6. $\mathcal{U}(Q, A) = \emptyset$ implies $Q = P$.

Proof: We prove Items 3, 4, 5 and 6 together by induction on $P \in \tilde{\mathbb{P}}_1$. Item 1 follows directly from Definitions 2.5 and 8.3, and this implies Item 2 with Lemma 8.6-4.

Nil, Var: $P = \mathsf{nil}$, $P = x$. By Definition 2.5 $Q = \mathsf{urgent}(P, A)$ implies $Q = P$. In these cases

3. $Q \in \tilde{\mathbb{P}}_1$ and $Q = P$.
4. $Q = P$ and $\mathcal{A}(P, A) = \emptyset$.
5. $\mathsf{clean}(Q, A) = Q$.
6. Trivially $\mathcal{U}(Q, A) = \emptyset$ and $Q = P$.

Pref: $P = \alpha.P_1$.

3. By Definition 2.5 $Q \in \tilde{\mathbb{P}}_1$ implies $\alpha \in A$ and $Q = \alpha.P_1 = P$.
4. Similarly to the previuos case $Q = P$ iff $\alpha \in A$ iff $\mathcal{A}(P, A) = \emptyset$.
5. Consider the following cases:
   - $\alpha \notin A$. Then $Q = \underline{\alpha}.P_1$ and $\mathsf{clean}(Q, A) = \underline{\alpha}.P_1 = Q$.
   - $\alpha \in A$. In this case $Q = \alpha.P_1$ and $\mathsf{clean}(Q, A) = \alpha.P_1 = Q$.
6. Consider the following cases:
   - $\alpha \notin A$. By Definitions 2.3 and 8.4, $Q = \underline{\alpha}.P_1$ and $\mathcal{U}(Q, A) = \{\alpha\}$, and we are done.
   - $\alpha \in A$. In this case $Q = \alpha.P_1 = P$.

Sum: $P = P_1 + P_2$. Assume $Q = \mathsf{urgent}(P, A)$.

3. $Q \in \tilde{\mathbb{P}}_1$ implies $(\mathcal{A}(P_1) \cup \mathcal{A}(P_2))\backslash A = \emptyset$ and $Q = P_1 + P_2 = P$.
4. By Lemma 8.2-4, $\mathcal{A}(P, A) = \mathcal{A}(P)\backslash A = (\mathcal{A}(P_1) \cup \mathcal{A}(P_2))\backslash A$. Thus, again by Definition 2.5, $Q = P$ iff $(\mathcal{A}(P_1 \cup \mathcal{A}(P_2))\backslash A = \mathcal{A}(P, A) = \emptyset$.
5. Consider the following cases:
   - $(\mathcal{A}(P_1) \cup \mathcal{A}(P_2))\backslash A \neq \emptyset$ (and hence $\mathcal{A}(P_1) \cup \mathcal{A}(P_2) \not\subseteq A$). In this case $Q = P_1 \underline{+} P_2$ and, by Definition 2.3, $\mathsf{clean}(Q, A) = P_1 \underline{+} P_2 = Q$.
   - $(\mathcal{A}(P_1) \cup \mathcal{A}(P_2))\backslash A = \emptyset$. In this case $Q = P_1 + P_2$ and, again by Definition 2.3, $\mathsf{clean}(Q, A) = P_1 + P_2 = Q$.
6. Consider the following cases:
   - $(\mathcal{A}(P_1) \cup \mathcal{A}(P_2))\backslash A \neq \emptyset$. This case is not possible since, by Lemma 8.2-4, $\mathcal{U}(Q, A) = \mathcal{A}(P_1, A) \cup \mathcal{A}(P_2, A) = (\mathcal{A}(P_1)\backslash A) \cup (\mathcal{A}(P_2)\backslash A) = (\mathcal{A}(P_1) \cup \mathcal{A}(P_2))\backslash A \neq \emptyset$.
   - $(\mathcal{A}(P_1) \cup \mathcal{A}(P_2))\backslash A = \emptyset$. Trivially, $\mathcal{U}(Q, A) = (\mathcal{A}(P_1) \cup \mathcal{A}(P_2))\backslash A = \emptyset$ and $Q = P_1 + P_2 = P$.

Par: $P = P_1 \parallel_B P_2$. Let $A_1 = (\mathcal{A}(P_1)\backslash\mathcal{A}(P_2)) \cap B$ and $A_2 = (\mathcal{A}(P_2)\backslash\mathcal{A}(P_1)) \cap B$. In this case $Q = \mathsf{urgent}(P, A)$ implies $Q = Q_1 \parallel_B Q_2$ with $Q_i = \mathsf{urgent}(P_i, A \cup A_i)$ for $i = 1, 2$.

3. $Q \in \tilde{\mathbb{P}}_1$ implies $Q_1, Q_2 \in \tilde{\mathbb{P}}_1$. By induction hypothesis we have $Q_1 = P_1$ and $Q_2 = P_2$ and, hence, $Q = P$.
4. $Q = P$ iff $Q_i = P_i$ for $i = 1, 2$ iff by induction hypothesis $\mathcal{A}(P_i, A \cup A_i) = \emptyset$ for $i = 1, 2$ iff, by Definition 2.2, $\mathcal{A}(P, A) = \emptyset$.
5. Since, by Item 2, $(\mathcal{A}(Q_1)\backslash\mathcal{A}(Q_2)) \cap B = A_1$ and $(\mathcal{A}(Q_2)\backslash\mathcal{A}(Q_1)) \cap B = A_2$, we have that $\mathsf{clean}(Q, A) = \mathsf{clean}(Q_1, A \cup A_1) \parallel_B \mathsf{clean}(Q_2, A \cup A_2) = Q_1 \parallel_B Q_2 = Q$, by induction hypothesis.
6. Since, as in the previous case, $(\mathcal{A}(Q_1)\backslash\mathcal{A}(Q_2)) \cap B = A_1$ and $(\mathcal{A}(Q_2)\backslash\mathcal{A}(Q_1)) \cap B = A_2$, $\mathcal{U}(Q, A) = \emptyset$ implies $\mathcal{U}(Q_1, A \cup A_1) = \mathcal{U}(Q_2, A \cup A_2) = \emptyset$. By induction hypothesis $Q_1 = P_1$, $Q_2 = P_2$ and, hence, $Q = P$.

Rel: $P = P_1[\Phi]$. $Q = \mathsf{urgent}(P, A)$ implies $Q = Q_1[\Phi]$ with $Q_1 = \mathsf{urgent}(P_1, \Phi^{-1}(A))$.

3. $Q \in \tilde{\mathbb{P}}_1$ implies $Q_1 \in \tilde{\mathbb{P}}_1$. By induction hypothesis we have $Q_1 = P_1$ and, hence, $Q = Q_1[\Phi] = P_1[\Phi] = P$.
4. $Q = P$ iff $Q_1 = P_1$ iff, by induction hypothesis, $\mathcal{A}(P_1, \Phi^{-1}(A)) = \emptyset$ iff $\mathcal{A}(P, A) = \Phi(\mathcal{A}(P_1, \Phi^{-1}(A))) = \emptyset$.
5. By induction hypothesis $\mathsf{clean}(Q, A) = \mathsf{clean}(Q_1, \Phi^{-1}(A))[\Phi] = Q_1[\Phi] = Q$
6. In this case $\mathcal{U}(Q, A) = \Phi(\mathcal{U}(Q_1, \Phi^{-1}(A))) = \emptyset$ implies $\mathcal{U}(Q_1, \Phi^{-1}(A)) = \emptyset$. By induction hypothesis we have that $Q_1 = P_1$ and, hence, $Q = P$.

Rec: $P = \mathsf{rec}\, x.P_1$. $Q = \mathsf{urgent}(P, A)$ implies $Q = \mathsf{rec}\, x.Q_1$ with $Q_1 = \mathsf{urgent}(P_1, A)$.

3. $Q \in \tilde{\mathbb{P}}_1$ implies $Q_1 \in \tilde{\mathbb{P}}_1$. By induction hypothesis we have $Q_1 = P_1$ and, hence, $Q = P$.
4. $Q = P$ iff $Q_1 = P_1$ iff, by induction hypothesis, $\mathcal{A}(P_1, A) = \mathcal{A}(P, A) = \emptyset$.
5. By induction hypothesis $\mathsf{clean}(Q, A) = \mathsf{rec}\, x.\mathsf{clean}(Q_1, A) = \mathsf{rec}\, x.Q_1 = Q$.
6. $\mathcal{U}(Q, A) = \mathcal{U}(Q_1, A) = \emptyset$ implies, by induction hypothesis, $Q_1 = P_1$ and hence $Q = P$.

□

We collect the most important properties shown so far in the following proposition, and then carry on with further technical results:

Proposition 8.8 Let $P \in \tilde{\mathbb{P}}_1$ and $Q = \mathsf{urgent}(P)$. Then:

1. $Q \in \tilde{\mathbb{P}}_1$ implies $Q = P$;
2. $\mathsf{unmark}(P) = P = \mathsf{unmark}(Q)$;
3. $Q = P$ iff $\mathcal{A}(P) = \emptyset$.

Proof: Part 1 and 3 follow from Lemmas 8.7-3 and 8.7-4, while Part 2 follows from Lemmas 8.6-2 and 8.7-1. □

Lemma 8.9 Let $P \in \tilde{\mathbb{P}}_1$, $Q \in \tilde{\mathbb{P}}$ and $x \in \mathcal{X}$ such that x is guarded in Q. Then:

1. $\mathcal{A}(Q\{P/x\}, A) = \mathcal{A}(Q, A)$;
2. $\mathsf{clean}(Q\{P/x\}, A) = \mathsf{clean}(Q, A)\{P/x\}$;
3. $\mathsf{unmark}(Q\{P/x\}) = \mathsf{unmark}(Q)\{P/x\}$;
4. $Q = P' \in \tilde{\mathbb{P}}_1$ implies $\mathsf{urgent}(P'\{P/x\}, A) = \mathsf{urgent}(P', A)\{P/x\}$.

Proof: We prove Items 1, 2 and 4 together by induction on $Q \in \tilde{\mathbb{P}}$. Item 3 can be proved similarly to Item 2 and the proof is omitted.

Nil: $Q = \mathsf{nil}$. In this case x is guarded in Q and $Q\{P/x\} = \mathsf{nil}$. Moreover

1. $\mathcal{A}(Q\{P/x\}, A) = \mathcal{A}(Q, A) = \mathcal{A}(\mathsf{nil}, A) = \emptyset$.
2. $\mathsf{clean}(Q\{P/x\}, A) = \mathsf{clean}(Q, A)\{P/x\} = \mathsf{nil}$.
4. $Q \in \tilde{\mathbb{P}}_1$ and $\mathsf{urgent}(\mathsf{nil}\{P/x\}, A) = \mathsf{urgent}(\mathsf{nil}, A) = \mathsf{nil} = \mathsf{urgent}(\mathsf{nil}, A)\{P/x\}$.

Var: $Q = y$. x guarded in Q implies $y \neq x$ and $Q\{P/x\} = y \in \tilde{\mathbb{P}}_1$. Similar to the Nil-case.

Pref: $Q = \alpha.P_1$ or $Q = \underline{\alpha}.P_1$. In this case x is guarded in Q.

1. Assume $Q = \underline{\alpha}.P_1$ (if $Q = \alpha.P_1$ the item can be proved similarly) and, hence, $Q\{P/x\} = \underline{\alpha}.(P_1\{P/x\})$. Then $\alpha \notin A$ implies $\mathcal{A}(Q, A) = \mathcal{A}(Q\{P/x\}, A) = \{\alpha\}$; otherwise $\mathcal{A}(Q, A) = \mathcal{A}(Q\{P/x\}, A) = \emptyset$. Thus $\mathcal{A}(Q, A) = \mathcal{A}(Q\{P/x\}, A)$.
2. Again we can assume $Q = \underline{\alpha}.P_1$ and $Q\{P/x\} = \underline{\alpha}.(P_1\{P/x\})$. Then $\alpha \in A$ implies $\mathsf{clean}(Q\{P/x\}, A) = \alpha.(P_1\{P/x\}) = (\alpha.P_1)\{P/x\} = \mathsf{clean}(Q, A)\{P/x\}$, otherwise $\mathsf{clean}(Q\{P/x\}, A) = \underline{\alpha}.(P_1\{P/x\}) = (\underline{\alpha}.P_1)\{P/x\} = \mathsf{clean}(Q, A)\{P/x\}$.
4. $Q = P' \in \tilde{\mathbb{P}}_1$ implies $P' = \alpha.P_1$ and, hence, $P'\{P/x\} = \alpha.(P_1\{P/x\})$. By definition 2.5 $\alpha \notin A$ implies $\mathsf{urgent}(P'\{P/x\}) = \underline{\alpha}.(P_1\{P/x\}) = (\underline{\alpha}.P_1)\{P/x\} = \mathsf{urgent}(P', A)\{P/x\})$. Similarly $\alpha \in A$ implies $\mathsf{urgent}(P'\{P/x\}) = \alpha.(P_1\{P/x\}) = (\alpha.P_1)\{P/x\} = \mathsf{urgent}(P', A)\{P/x\})$.

Sum: $Q = P_1 \oplus P_2$ with $\oplus \in \{+, \underline{+}\}$. In this case x guarded in Q implies x guarded in P_1 and in P_2. Moreover $Q\{P/x\} = P_1\{P/x\} \oplus P_2\{P/x\}$. Let $A' = \mathcal{A}(P_1\{Q/x\}) \cup \mathcal{A}(P_2\{P/x\}) = \mathcal{A}(P_1) \cup \mathcal{A}(P_2)$ by Item 1.

1. By induction hypothesis $\mathcal{A}(Q\{P/x\}, A) = \mathcal{A}(P_1\{Q/x\}, A) \cup \mathcal{A}(P_2\{P/x\}, A) = \mathcal{A}(P_1, A) \cup \mathcal{A}(P_2, A) = \mathcal{A}(Q, A)$.
2. We prove only the case $\oplus = \underline{+}$ (the other case is simplier). By Definition 2.3 we have to consider two possible cases. If $A' \subseteq A$ then $\mathsf{clean}(Q\{P/x\}, A) = P_1\{P/x\} + P_2\{P/x\} = (P_1 + P_2)\{P/x\} = \mathsf{clean}(Q, A)\{P/x\}$. Otherwise, if $A' \not\subseteq A$, $\mathsf{clean}(Q\{P/x\}, A) = P_1\{P/x\} \underline{+} P_2\{P/x\} = (P_1 \underline{+} P_2)\{P/x\} = \mathsf{clean}(Q, A)\{P/x\}$.
4. In this case $Q = P' \in \tilde{\mathbb{P}}_1$ implies $P' = P_1 + P_2$ and $P'\{P/x\} = P_1\{P/x\} + P_2\{P/x\}$. If $A' \neq \emptyset$ then $\mathsf{urgent}(P'\{P/x\}, A) = P_1\{P/x\} \underline{+} P_2\{P/x\} = (P_1 \underline{+} P_2)\{P/x\} = \mathsf{urgent}(P', A)\{P/x\}$. Otherwise, if $A' = \emptyset$, $\mathsf{urgent}(P'\{P/x\}, A) = P_1\{P/x\} + P_2\{P/x\} = (P_1 + P_2)\{P/x\} = \mathsf{urgent}(P', A)\{P/x\}$.

Par: $Q = Q_1 \parallel_B Q_2$. As in the previous case x guarded in Q implies x guarded in Q_1 and in Q_2. Moreover $Q\{P/x\} = Q_1\{P/x\} \parallel_B Q_2\{P/x\}$. Let $A_1 = (\mathcal{A}(Q_1\{P/x\}) \backslash \mathcal{A}(Q_2\{P/x\})) \cap B$ and $A_2 = (\mathcal{A}(Q_2\{P/x\}) \backslash \mathcal{A}(Q_1\{P/x\})) \cap B$. Item 1 implies $A_1 = (\mathcal{A}(Q_1) \backslash \mathcal{A}(Q_2)) \cap B$ and $A_2 = (\mathcal{A}(Q_2) \backslash \mathcal{A}(Q_1)) \cap B$. By induction hypothesis we have that:

1. $\mathcal{A}(Q\{P/x\}, A) = \mathcal{A}(Q_1\{P/x\}, A \cup A_1) \cup \mathcal{A}(Q_2\{P/x\}, A \cup A_2) = \mathcal{A}(Q_1, A \cup A_1) \cup \mathcal{A}(Q_2, A \cup A_2) = \mathcal{A}(Q, A)$.
2. $\mathsf{clean}(Q\{P/x\}, A) = \mathsf{clean}(Q_1\{P/x\}, A \cup A_1) \parallel_B \mathsf{clean}(Q_2\{P/x\}, A \cup A_2) = (\mathsf{clean}(Q_1, A \cup A_1)\{P/x\}) \parallel_B (\mathsf{clean}(Q_2, A \cup A_2)\{P/x\}) = (\mathsf{clean}(Q_1, A \cup A_1) \parallel_B \mathsf{clean}(Q_2, A \cup A_2))\{P/x\} = \mathsf{clean}(Q, A)\{P/x\}$.
4. $Q = P' \in \tilde{\mathbb{P}}_1$ implies $P' = P'_1 \parallel_B P'_2$ with $P'_i = Q_i$ and $P'_i \in \tilde{\mathbb{P}}_1$ for $i = 1, 2$. Then $\mathsf{urgent}(P'\{P/x\}, A) = \mathsf{urgent}(P'_1\{P/x\}, A \cup A_1) \parallel_B \mathsf{urgent}(P'_2\{P/x\}, A \cup A_2) = (\mathsf{urgent}(P'_1, A \cup A_1)\{P/x\}) \parallel_B (\mathsf{urgent}(P'_2, A \cup A_2)\{P/x\}) = (\mathsf{urgent}(P'_1, A \cup A_1) \parallel_B \mathsf{urgent}(P'_2, A \cup A_2))\{P/x\} = \mathsf{urgent}(P', A)\{P/x\}$.

Rel: $Q = Q_1[\Phi]$. In this case x guarded in Q implies x guarded in Q_1. Moreover $Q\{P/x\} = (Q_1[\Phi])\{P/x\} = (Q_1\{P/x\})[\Phi]$. By induction hypothesis we have that:

1. $\mathcal{A}(Q\{P/x\}, A) = \mathcal{A}((Q_1\{P/x\})[\Phi], A) = \Phi(\mathcal{A}(Q_1\{P/x\}, \Phi^{-1}(A))) = \Phi(\mathcal{A}(Q_1, \Phi^{-1}(A))) = \mathcal{A}(Q, A)$.

2. $\mathsf{clean}(Q\{P/x\}, A) = \mathsf{clean}(Q_1\{P/x\}, \Phi^{-1}(A))[\Phi] =$
$(\mathsf{clean}(Q_1, \Phi^{-1}(A))\{P/x\})[\Phi] = (\mathsf{clean}(Q_1, \Phi^{-1}(A))[\Phi])\{P/x\} =$
$\mathsf{clean}(Q, A)\{P/x\}$.
4. $Q = P' \in \tilde{\mathbb{P}}_1$ implies $Q_1 = P'_1 \in \tilde{\mathbb{P}}_1$ and $P' = P'_1[\Phi]$. Then $\mathsf{urgent}(P'\{P/x\}, A) =$
$\mathsf{urgent}(P'_1\{P/x\}, \Phi^{-1}(A))[\Phi] = (\mathsf{urgent}(P'_1, \Phi^{-1}(A))\{P/x\})[\Phi] =$
$(\mathsf{urgent}(P'_1, \Phi^{-1}(A))[\Phi])\{P/x\} = \mathsf{urgent}(P', A)\{P/x\}$.

Rec: $Q = \mathsf{rec}\, y.Q_1$. We have to consider two cases:

- $x = y$. $Q\{P/x\} = Q$ and the items can be proved trivially.
- $x \neq y$. x guarded in Q implies x guarded in Q_1 and $Q\{P/x\} = \mathsf{rec}\, y.(Q_1\{P/x\})$. By induction hypothesis:
  1. $\mathcal{A}(Q\{P/x\}, A) = \mathcal{A}(Q_1\{P/x\}, A) = \mathcal{A}(Q_1, A) = \mathcal{A}(Q, A)$.
  2. $\mathsf{clean}(Q\{P/x\}, A) = \mathsf{rec}\, y.\mathsf{clean}(Q_1\{P/x\}, A) =$
  $\mathsf{rec}\, y.(\mathsf{clean}(Q_1, A)\{P/x\}) = (\mathsf{rec}\, y.\mathsf{clean}(Q_1, A))\{P/x\} =$
  $\mathsf{clean}(Q, A)\{P/x\}$.
  4. $Q = P' \in \tilde{\mathbb{P}}_1$ implies $Q_1 = P'_1$ and $P' = \mathsf{rec}\, y.P'_1$. Then $\mathsf{urgent}(P'\{P/x\}, A) =$
  $\mathsf{rec}\, y.\mathsf{urgent}(P'_1\{P/x\}, A) = \mathsf{rec}\, y.(\mathsf{urgent}(P'_1, A)\{P/x\}) =$
  $(\mathsf{rec}\, y.\mathsf{urgent}(P'_1, A))\{P/x\} = \mathsf{urgent}(P', A)\{P/x\}$.

□

Lemma 8.10 Let $Q, Q' \in \tilde{\mathbb{P}}$ and $A \subseteq \mathbb{A}$ such that $Q' = \mathsf{clean}(Q, A)$. Then:

1. $\mathsf{clean}(Q', A) = Q'$;
2. $\mathcal{U}(Q', A) = \emptyset$ implies $Q' \in \tilde{\mathbb{P}}_1$.

Proof: We prove both items by induction on $Q \in \tilde{\mathbb{P}}$.

Nil, Var: $Q = \mathsf{nil}$, $Q = x$. In these cases $Q' = \mathsf{clean}(Q, A)$ implies $Q' = Q$. Moreover

1. $\mathsf{clean}(Q', A) = Q'$.
2. $\mathcal{U}(Q', A) = \emptyset$ and $Q' \in \tilde{\mathbb{P}}_1$.

Pref: $Q = \alpha.P_1$ or $Q = \underline{\alpha}.P_1$. We prove only the latter case, the former one is simpler. Consider the following cases:

- $\alpha \in A$. Then $Q' = \alpha.P_1$ and clearly
  1. $\mathsf{clean}(Q', A) = \alpha.P_1 = Q'$.
  2. $\mathcal{U}(Q', A) = \emptyset$ and $Q' \in \tilde{\mathbb{P}}_1$.
- $\alpha \notin A$. In this case $Q' = \underline{\alpha}.P_1$. Moreover $\alpha \notin A$ implies
  1. $\mathsf{clean}(Q', A) = \underline{\alpha}.P_1 = Q'$.
  2. $\mathcal{U}(Q', A) = \{\alpha\}$.

Sum: $Q = P_1 + P_2$ or $Q = P_1 \underline{+} P_2$. Again we prove only the latter case. Consider the following cases:

- $\mathcal{A}(P_1) \cup \mathcal{A}(P_2) \subseteq A$. In this case $Q' = P_1 + P_2$ and
  1. $\mathsf{clean}(Q', A) = P_1 + P_2 = Q'$.

2. $\mathcal{U}(Q', A) = \emptyset$ and $Q' \in \tilde{\mathbb{P}}_1$.

- $\mathcal{A}(P_1) \cup \mathcal{A}(P_2) \not\subseteq A$. In this case $Q' = P_1 \underline{+} P_2$. Moreover $\mathcal{A}(P_1) \cup \mathcal{A}(P_2) \not\subseteq A$ implies
  1. $\mathsf{clean}(Q', A) = P_1 \underline{+} P_2 = Q'$.
  2. $\mathcal{U}(Q', A) = \mathcal{A}(P_1, A) \cup \mathcal{A}(P_2, A) = (\mathcal{A}(P_1) \cup \mathcal{A}(P_2)) \backslash A \neq \emptyset$ (cf. 8.2-2)

Par: $Q = Q_1 \parallel_B Q_2$. Let $A_1 = (\mathcal{A}(Q_1) \backslash \mathcal{A}(Q_2)) \cap B$ and $A_2 = (\mathcal{A}(Q_2) \backslash \mathcal{A}(Q_1)) \cap B$. In this case $Q' = \mathsf{clean}(Q, A)$ implies $Q' = Q'_1 \parallel_B Q'_2$ with $Q'_i = \mathsf{clean}(Q_i, A \cup A_i)$ for $i = 1, 2$. Moreover, by Lemma 8.6-4, $A_1 = (\mathcal{A}(Q'_1) \backslash \mathcal{A}(Q'_2)) \cap B$ and $A_2 = (\mathcal{A}(Q'_2) \backslash \mathcal{A}(Q'_1)) \cap B$. Thus:

1. $\mathsf{clean}(Q', A) = \mathsf{clean}(Q'_1, A \cup A_1) \parallel_B \mathsf{clean}(Q'_2, A \cup A_2) = Q'_1 \parallel_B Q'_2 = Q'$, by induction hypothesis.
2. $\mathcal{U}(Q', A) = \mathcal{U}(Q'_1, A \cup A_1) \cup \mathcal{U}(Q'_2, A \cup A_2) = \emptyset$ implies $\mathcal{U}(Q'_1, A \cup A_1) = \mathcal{U}(Q'_2, A \cup A_2) = \emptyset$. By induction hypothesis $Q'_1 \in \tilde{\mathbb{P}}_1$, $Q'_2 \in \tilde{\mathbb{P}}_1$ and, hence, $Q' \in \tilde{\mathbb{P}}_1$.

Rel: $Q = Q_1[\Phi]$. $Q' = Q'_1[\Phi]$ where $Q'_1 = \mathsf{clean}(Q_1, \Phi^{-1}(A))$.

1. By induction hypothesis $\mathsf{clean}(Q', A) = \mathsf{clean}(Q'_1, \Phi^{-1}(A))[\Phi] = Q'_1[\Phi] = Q'$.
2. $\mathcal{U}(Q', A) = \Phi(\mathcal{U}(Q'_1, \Phi^{-1}(A))) = \emptyset$ implies $\mathcal{U}(Q'_1, \Phi^{-1}(A)) = \emptyset$. Thus, by induction hypothesis, $Q'_1 \in \tilde{\mathbb{P}}_1$ and, hence, $Q' \in \tilde{\mathbb{P}}_1$.

Rec: $Q = \mathsf{rec}\, x.Q_1$. $Q' = \mathsf{rec}\, x.Q'_1$ where $Q'_1 = \mathsf{clean}(Q_1, A)$.

1. By induction hypothesis $\mathsf{clean}(Q', A) = \mathsf{rec}\, x.\mathsf{clean}(Q'_1, A) = \mathsf{rec}\, x.Q'_1 = Q'$.
2. $\mathcal{U}(Q', A) = \mathcal{U}(Q'_1, A) = \emptyset$ implies, by induction hypothesis, $Q'_1 \in \tilde{\mathbb{P}}_1$ and, hence, $Q' \in \tilde{\mathbb{P}}_1$.

□

8.4 Properties of $\xrightarrow{\alpha}$ and $\xrightarrow{1}$

The following two propositions relate $\mathcal{A}(\_)$ to the operational semantics; they are needed in our main proofs.

Proposition 8.11 $Q \xrightarrow{1} Q' \xrightarrow{1}$ implies $Q = Q' \in \tilde{\mathbb{P}}_1$ and $\mathcal{A}(Q) = \emptyset$.

Proof: Assume that $Q \xrightarrow{1} Q' \xrightarrow{1}$. Then, by Definition 2.5, $Q, Q' \in \tilde{\mathbb{P}}_1$ and $Q' = \mathsf{urgent}(Q)$. Now, $Q' = \mathsf{urgent}(Q) \in \tilde{\mathbb{P}}_1$ and Proposition 8.8-1, $Q' = Q$. Moreover $\mathsf{urgent}(Q) = Q$ implies, by Proposition 8.8-3, $\mathcal{A}(Q) = \emptyset$. □

The following proposition will follow directly from Lemmas 8.13 and 8.14 stated below.

Proposition 8.12 Let $Q \in \tilde{\mathbb{P}}$. Then $Q \xrightarrow{\alpha}$ if and only if $\alpha \in \mathcal{A}(Q)$.

Lemma 8.13 Let $P \in \tilde{\mathbb{P}}_1$ and $Q \in \tilde{\mathbb{P}}$ such that $Q \xrightarrow{\alpha}$. Then:

1. $\alpha \in \mathcal{A}(Q)$;
2. x guarded in Q implies $Q\{P/x\} \xrightarrow{\alpha}$.

Proof: The proof is by induction on the depth of derivation $Q \xrightarrow{\alpha}$. We proceed by case analysis on the structure of Q.

Nil, Var: $Q = \mathsf{nil}$, $Q = x$. These cases are not possible since $Q \not\xrightarrow{\alpha}$ for any α.

Pref: $Q = \alpha.P_1$ or $Q = \underline{\alpha}.P_1$. In both case $Q \xrightarrow{\alpha}$. Moreover:

1. $\alpha \in \mathcal{A}(Q) = \{\alpha\}$.
2. x is guarded in Q and either $Q\{P/x\} = \alpha.P_1\{P/x\}$ or $Q\{P/x\} = \underline{\alpha}.P_1\{P/x\}$. In both cases $Q\{P/x\} \xrightarrow{\alpha}$.

Sum: $Q = P_1 \oplus P_2$ (where $\oplus \in \{+, \underline{+}\}$). $Q \xrightarrow{\alpha}$ implies either (i) $P_1 \xrightarrow{\alpha}$ or (ii) $P_2 \xrightarrow{\alpha}$. Consider the (i)-case (the latter case can be proved similarly). By induction hypothesis:

1. $\alpha \in \mathcal{A}(P_1) \subseteq \mathcal{A}(Q)$.
2. x guarded in Q and, hence, in P_1 implies $P_1\{P/x\} \xrightarrow{\alpha}$. By operational semantics $Q\{P/x\} = P_1\{P/x\} \oplus P_2\{P/x\} \xrightarrow{\alpha}$.

Par: $Q = Q_1 \parallel_B Q_2$. Let $A_1 = (\mathcal{A}(Q_1) \backslash \mathcal{A}(Q_2)) \cap B$ and $A_2 = (\mathcal{A}(Q_2) \backslash \mathcal{A}(Q_1)) \cap B$. Assume $Q \xrightarrow{\alpha}$ and consider the following cases:

- $\alpha \notin B$ and $Q_1 \xrightarrow{\alpha}$. By induction hypothesis:
  1. $\alpha \in \mathcal{A}(Q_1)$. Moreover, since $\alpha \notin B$ clearly implies $\alpha \notin A_1$, by Lemma 8.2-2 we have that $\alpha \in \mathcal{A}(Q_1, A_1) \subseteq \mathcal{A}(Q)$.
  2. x guarded in Q and, hence, in Q_1 implies $Q_1\{P/x\} \xrightarrow{\alpha}$. Then, by operational semantics, $Q\{P/x\} = Q_1\{P/x\} \parallel_B Q_2\{P/x\} \xrightarrow{\alpha}$.
- $\alpha \notin B$ and $Q_2 \xrightarrow{\alpha}$. Similar to the previous case.
- $\alpha \in B$ and $Q_i \xrightarrow{\alpha}$ for $i = 1, 2$. By induction hypothesis:
  1. $\alpha \in \mathcal{A}(Q_1)$ and $\alpha \in \mathcal{A}(Q_2)$. Thus, clearly, $\alpha \notin A_1$ and, as in the previous case, $\alpha \in \mathcal{A}(Q_1, A_1) \subseteq \mathcal{A}(Q)$.
  2. x guarded in Q and, hence, in Q_i implies $Q_i\{P/x\} \xrightarrow{\alpha}$ for $i = 1, 2$. Thus, again by operational semantics, $Q\{P/x\} = Q_1\{P/x\} \parallel_B Q_2\{P/x\} \xrightarrow{\alpha}$.

Rel: $Q = Q_1[\Phi]$. $Q \xrightarrow{\alpha}$ if there exists $\beta \in \Phi^{-1}(\alpha)$ such that $Q_1 \xrightarrow{\beta}$. By induction hypothesis:

1. $\beta \in \mathcal{A}(Q_1)$ and, hence, $\alpha = \Phi(\beta) \in \Phi(\mathcal{A}(Q_1)) = \mathcal{A}(Q)$.
2. x guarded in Q and, hence, in Q_1 implies $Q_1\{P/x\} \xrightarrow{\beta}$. By operational semantics, $Q\{P/x\} = (Q_1\{P/x\})[\Phi] \xrightarrow{\alpha}$.

Rec: $Q = \mathsf{rec}\ y.Q_1$. Let $P_1 = \mathsf{unmark}(Q_1)$ and $R = Q_1\{\mathsf{rec}\ y.P_1/y\}$. By operational semantics we have that $Q \xrightarrow{\alpha}$ only if $R \xrightarrow{\alpha}$.

1. By induction hypothesis we have that $\alpha \in \mathcal{A}(R)$. Moreover y guarded in Q_1 and Lemma 8.9-1 implies $\mathcal{A}(R) = \mathcal{A}(Q_1\{\mathsf{rec}\ y.P_1/y\}) = \mathcal{A}(Q_1) = \mathcal{A}(Q)$. Thus we can conclude that $\alpha \in \mathcal{A}(Q)$.
2. We have to consider to possible subcases. If $x = y$ then x is guarded in Q, $Q\{P/x\} = Q$ and the statement follows easily. Assume $x \neq y$, x guarded in Q and hence in Q_1, and $Q\{P/x\} = \mathsf{rec}\ y.(Q_1\{P/x\}) = \mathsf{rec}\ y.S_1$, where $S_1 = Q_1\{P/x\}$. In this case, by operational semantics, we have to prove that $S = S_1\{\mathsf{rec}\ y.\mathsf{unmark}(S_1)/y\} \xrightarrow{\alpha}$.

   By Lemma 8.9-3 $\mathsf{unmark}(S_1) = \mathsf{unmark}(Q_1\{P/x\}) = \mathsf{unmark}(Q_1)\{P/x\} = P_1\{P/x\}$. Thus $S = S_1\{\mathsf{rec}\ y.\ \mathsf{unmark}(S_1)/y\} = S_1\{\mathsf{rec}\ y.\ (P_1\{P/x\})/y\} =$
   $(Q_1\{P/x\})\{\mathsf{rec}\ y.\ (P_1\{P/x\})/y\} = (Q_1\{\mathsf{rec}\ y.\ P_1/y\})\{P/x\} = R\{P/x\}$. Moreover x guarded in Q_1 implies x guarded in $P_1 = \mathsf{unmark}(Q_1)$ and in $R = Q_1\{\mathsf{rec}\ y.P_1/y\}$. Finally, by induction hypothesis, $R \xrightarrow{\alpha}$ and x guarded in R implies that $S = R\{P/x\} \xrightarrow{\alpha}$.

□

Lemma 8.14 Let $Q \in \tilde{\mathbb{P}}$ and $\alpha \in \mathbb{A}_\tau$. If there exists $A \subseteq \mathbb{A}$ such that $\alpha \in \mathcal{A}(Q, A)$ then $Q \xrightarrow{\alpha}$.

Proof: By induction on $Q \in \tilde{\mathbb{P}}$.

Nil, Var: $Q = \mathsf{nil}$, $Q = x$. In these cases $\mathcal{A}(Q, A) = \emptyset$ for all A and $Q \not\xrightarrow{\alpha}$.

Pref: $Q = \alpha.P_1$ or $Q = \underline{\alpha}.P_1$. Let $A \subseteq \mathbb{A}$ such that $\alpha \notin A$. Then $\alpha \in \mathcal{A}(Q, A)$ and $Q \xrightarrow{\alpha}$.

Sum: $Q = P_1 \oplus P_2$ (where $\oplus \in \{+, \underline{+}\}$). Assume that there exists $A \subseteq \mathbb{A}$ such that $\alpha \in \mathcal{A}(Q, A) = \mathcal{A}(P_1, A) \cup \mathcal{A}(P_2, A)$. We have either $\alpha \in \mathcal{A}(P_1, A)$ or $\alpha \in \mathcal{A}(P_2, A)$. Then, by induction hypothesis, either $P_1 \xrightarrow{\alpha}$ or $P_2 \xrightarrow{\alpha}$. In both cases $Q \xrightarrow{\alpha}$.

Par: $Q = Q_1 \parallel_B Q_2$. Let $A_1 = (\mathcal{A}(Q_1) \backslash \mathcal{A}(Q_2)) \cap B$ and $A_2 = (\mathcal{A}(Q_2) \backslash \mathcal{A}(Q_1)) \cap B$ and assume that there exists $A \subseteq \mathbb{A}$ such that $\alpha \in \mathcal{A}(Q, A) = \mathcal{A}(Q_1, A \cup A_1) \cup \mathcal{A}(Q_2, A \cup A_2)$. Assume $\alpha \in \mathcal{A}(Q_1, A \cup A_1)$. By induction hypothesis $Q_1 \xrightarrow{\alpha}$, and we have to consider the following possible cases:

- $\alpha \notin B$. Then, by operational semantics, $Q \xrightarrow{\alpha}$.
- $\alpha \in B$. In this case $\alpha \in \mathcal{A}(Q_1, A \cup A_1)$ and Lemmas 8.2-3 and 8.2-1 imply that $\alpha \in \mathcal{A}(Q_1)$ and $\alpha \notin A_1$. Thus $\alpha \in \mathcal{A}(Q_1) \cap B$ such that $\alpha \notin A_1$. We can conclude that $\alpha \in \mathcal{A}(Q_2)$ and, again by induction hypothesis, $Q_2 \xrightarrow{\alpha}$. Finally, by operational semantics, $Q \xrightarrow{\alpha}$.

Similarly we can prove that $\alpha \in \mathcal{A}(Q_2, A \cup A_2)$ implies $Q \xrightarrow{\alpha}$.

Rel: $Q = Q_1[\Phi]$. Assume that there exists $A \subseteq \mathbb{A}$ such that $\alpha \in \mathcal{A}(Q, A) = \Phi(\mathcal{A}(Q_1, \Phi^{-1}(A)))$. Then there exists $\beta \in \Phi^{-1}(\alpha)$ such that $\beta \in \mathcal{A}(Q_1, \Phi^{-1}(A)))$. By induction hypothesis $Q_1 \xrightarrow{\beta}$ and, by operational semantics, $Q \xrightarrow{\alpha}$.

Rec: $Q = \mathsf{rec}\, x.Q_1$. Let $P_1 = \mathsf{unmark}(Q_1)$. Assume that there exists $A \subseteq \mathbb{A}$ such that $\alpha \in \mathcal{A}(Q, A) = \mathcal{A}(Q_1, A)$. By induction hypothesis we have that $Q_1 \xrightarrow{\alpha}$. Moreover x guarded in Q_1 and Lemma 8.13-2 implies $Q_1\{\mathsf{rec}\, x.P_1/x\} \xrightarrow{\alpha}$. Finally, by operational semantics, $Q \xrightarrow{\alpha}$.

□

In the rest of this section, we prove a claim made in Section 2.

Lemma 8.15 Let $Q \in \tilde{\mathbb{P}}$ such that $Q \xrightarrow{\alpha} Q'$. Then:

1. $\mathsf{clean}(Q') = Q'$;
2. $\mathcal{U}(Q') = \emptyset$ implies $Q' \in \tilde{\mathbb{P}}_1$.

Proof: Item 2 follows from Item 1 and Lemma 8.10-2. The proof of Item 1 is by induction on the depth of derivation $Q \xrightarrow{\alpha} Q'$. We proceed by case analysis on the structure of Q.

Nil, Var: $Q = \mathsf{nil}$, $Q = x$. These cases are not possible since $Q \not\xrightarrow{\alpha}$ for all α.

Pref: $Q = \alpha.P_1$ or $Q = \underline{\alpha}.P_1$. In both cases $Q \xrightarrow{\alpha} P_1 \in \tilde{\mathbb{P}}_1$ and, by Lemma 8.6-2, $\mathsf{clean}(P_1) = P_1$

Sum: $Q = P_1 \oplus P_2$ (where $\oplus \in \{+, \underline{+}\}$). $Q \xrightarrow{\alpha} Q'$ implies either $P_1 \xrightarrow{\alpha} Q'$ or $P_2 \xrightarrow{\alpha} Q'$. In both cases, by induction hypothesis, $\mathsf{clean}(Q') = Q'$.

Par: $Q = Q_1 \parallel_B Q_2$. By operational semantics $Q \xrightarrow{\alpha} Q'$ implies $Q' = \mathsf{clean}(R)$ for some R. Then, by Lemma 8.10-1, $\mathsf{clean}(Q') = Q'$.

Rel: $Q = Q_1[\Phi]$. $Q \xrightarrow{\alpha} Q'$ if there exists $\beta \in \Phi^{-1}(\alpha)$ such that $Q_1 \xrightarrow{\beta} Q_1'$ and $Q' = Q_1'[\Phi]$. By induction hypothesis $\mathsf{clean}(Q_1') = Q_1'$. Thus $\mathsf{clean}(Q') = \mathsf{clean}(Q_1')[\Phi] = Q_1'[\Phi] = Q'$.

Rec: $Q = \mathsf{rec}\ y.Q_1$. Let $P_1 = \mathsf{unmark}(Q_1)$ and $R = Q_1\{\mathsf{rec}\ y.P_1/y\}$. By operational semantics $Q \xrightarrow{\alpha} Q'$ only if $R \xrightarrow{\alpha} Q'$. By induction hypothesis we have that $\mathsf{clean}(Q') = Q'$.

□

The final proposition of this section in combination with 8.1-2 implies that maximal sequences of action- and time-transitions starting at some initial process are never finite.

Proposition 8.16 Let $P_0 \in \tilde{\mathbb{P}}_1$ and $Q_0 = \mathsf{urgent}(P_0) \in \tilde{\mathbb{P}}$. Then $Q_0 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} Q_n$ and $Q_n \not\xrightarrow{\sigma}$ implies $Q_n \in \tilde{\mathbb{P}}_1$.

Proof: By Proposition 8.12 $Q_n \not\xrightarrow{\sigma}$ implies $\mathcal{A}(Q_n) = \emptyset$ and hence, by Lemma 8.5-1, $\mathcal{U}(Q_n) = \emptyset$. Now consider the following possible subcases:

- $n = 0$. Then $Q_n = Q_0 = \mathsf{urgent}(P_0)$ and $\mathcal{U}(Q_n) = \emptyset$ implies, by Lemma 8.7-6, $Q_n = P_0$ and, clearly, $Q_n \in \tilde{\mathbb{P}}_1$.
- $n \geq 1$. Then $Q_{n-1} \xrightarrow{\alpha_n} Q_n$ and $\mathcal{U}(Q_n) = \emptyset$ implies, by Lemma 8.15-2, $Q_n \in \tilde{\mathbb{P}}_1$.

□

9 Appendix B: A key property

This section is devoted to prove the following proposition that states a key property for our main correspondence results.

Proposition 9.1 Let $P_0 \in \mathsf{L}(\tilde{\mathbb{P}}_1)$ and $Q_0 \in \mathsf{L}(\tilde{\mathbb{P}})$ such that $Q_0 = \mathsf{urgent}(P_0)$. Then $Q_0 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} Q_n$ implies:

1. $\mathsf{UC}(Q_{i+1}) \subseteq \mathsf{UC}(Q_i)$ for every $i \in [0, n-1]$. Moreover, $\mathsf{UC}(Q_n) = \emptyset$ implies $Q_n \in \mathsf{L}(\tilde{\mathbb{P}}_1)$;
2. $(\mathsf{LC}(Q_0) \cap \ldots \cap \mathsf{LC}(Q_n)) \backslash \mathsf{UC}(Q_n) = \emptyset$.

A series of results will lead to a detailed proof that can be found at the end of this section. First of all, the following facts (that have been proved in [6]) are an immediate consequence of the labelling.

Fact 9.2 Let $P \in \mathsf{L}_u(\tilde{\mathbb{P}})$. Then

1. no label occurs more than once in P,
2. $w \in \mathsf{LAB}(P)$ implies $u \leq w$.

Fact 9.3 Let $P \in \mathsf{L}_u(\tilde{\mathbb{P}})$ and $\alpha, \alpha_1, \ldots \alpha_n \in \mathbb{A}_\tau$.

1. $P \xrightarrow{\alpha} Q$ implies $Q \in \mathsf{L}_v(\tilde{\mathbb{P}})$ with $u \leq v$.
2. $P \xrightarrow{\alpha_1} P_1 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_n} P_n$ implies $P_i \in \mathsf{L}_{v_i}(\tilde{\mathbb{P}})$ with $u \leq v_i$. Moreover, if $w \in \mathsf{LAB}$ such that $w < u$ then $w \notin \mathsf{LAB}(P_i)$.

The first proposition we state relates live components and activated actions in a process term.

Lemma 9.4 Let $Q \in \mathsf{L}(\tilde{\mathbb{P}})$ and $A \subseteq \mathbb{A}$. Then:

1. $\mathsf{LC}(Q, A) = \emptyset$ iff $\mathcal{A}(Q, A) = \emptyset$;
2. $\mathsf{UC}(Q, A) = \emptyset$ iff $\mathcal{U}(Q, A) = \emptyset$.

Proof: We prove, by induction on the structure of Q, only Item 1 (Item 2 can be proved similarly).

Nil, Var: $Q = \mathsf{nil}_u$, $Q = x_u$. In these cases, we have that $\mathsf{LC}(Q, A) = \mathcal{A}(Q, A) = \emptyset$ for all A.

Pref: $Q = \alpha_u.P_1$ or $Q = \underline{\alpha}_u.P_1$. In both cases $\mathsf{LC}(Q, A) = \emptyset$ iff $\alpha \in A$ iff $\mathcal{A}(Q, A) = \emptyset$.

Sum: $Q = P_1 \oplus_u P_2$ (where $\oplus \in \{+, \underline{+}\}$). $\mathsf{LC}(Q, A) = \emptyset$ iff $\mathsf{LC}(P_1, A) = \mathsf{LC}(P_2, A) = \emptyset$ iff, by induction hypothesis, $\mathcal{A}(P_1, A) = \mathcal{A}(P_2, A) = \emptyset$ iff, by Definition 2.2, $\mathcal{A}(Q, A) = \emptyset$.

Par: $Q = Q_1 \|^u_B Q_2$. Let $A_1 = (\mathcal{A}(Q_1) \backslash \mathcal{A}(Q_2)) \cap B$ and $A_2 = (\mathcal{A}(Q_2) \backslash \mathcal{A}(Q_1)) \cap B$. $\mathsf{LC}(Q, A) = \emptyset$ iff $\mathsf{LC}(Q_1, A \cup A_1) = \mathsf{LC}(Q_2, A \cup A_2) = \emptyset$ iff, by induction hypothesis, $\mathcal{A}(Q_1, A \cup A_1) = \mathcal{A}(Q_2, A \cup A_2) = \emptyset$ iff, by Definition 2.2, $\mathcal{A}(Q, A) = \emptyset$.

Rel: $Q = Q_1[\Phi_u]$. In this case we have that $\mathsf{LC}(Q, A) = \mathsf{LC}(Q_1, \Phi^{-1}(A)) = \emptyset$ iff, by induction hypothesis, $\mathcal{A}(Q_1, \Phi^{-1}(A)) = \emptyset$ iff $\mathcal{A}(Q, A) = \Phi(\mathcal{A}(Q_1, \Phi^{-1}(A))) = \emptyset$.

Rec: $Q = \mathsf{rec}\, x.Q_1$. $\mathsf{LC}(Q, A) = \mathsf{LC}(Q_1, A) = \emptyset$ iff, by induction hypothesis, $\mathcal{A}(Q, A) = \mathcal{A}(Q_1, A) = \emptyset$.

□

An easy, but useful lemma, is the following.

Lemma 9.5 Let Q be a labelled process term. Then:

1. $\mathsf{UC}(Q, A) \subseteq \mathsf{LC}(Q, A)$, for every $A \subseteq \mathbb{A}$;
2. $v \in \mathsf{LC}(Q)$ implies $v \in \mathsf{LAB}(Q)$;
3. $Q \in \mathsf{L}(\tilde{\mathbb{P}}_1)$ implies $\mathsf{UC}(Q, A) = \emptyset$, for every $A \subseteq \mathbb{A}$.

Lemma 9.6 Let $P \in \mathsf{L}(\tilde{\mathbb{P}}_1)$, $A, A' \subseteq \mathbb{A}$ and $Q = \mathsf{urgent}(P, A) \in \mathsf{L}(\tilde{\mathbb{P}})$. Then $A \subseteq A'$ implies $\mathsf{UC}(Q, A') = \mathsf{LC}(Q, A')$.

Proof: By induction on the structure of P.

Nil, Var: $P = \mathsf{nil}_u$, $P = x_u$. In these cases $Q = P$ and $\mathsf{UC}(Q, A') = \mathsf{LC}(Q, A') = \emptyset$ for all $A' \subseteq \mathbb{A}$.

Pref: $P = \alpha_u.P_1$ with $P_1 \in \mathsf{L}_{u1}(\tilde{\mathbb{P}}_1)$. Consider the following cases:

- $\alpha \notin A$. In this case $Q = \underline{\alpha}_u.P_1$. If $\alpha \notin A'$ then $\mathsf{UC}(Q, A') = \mathsf{LC}(Q, A') = \{u\}$, otherwise $\mathsf{UC}(Q, A') = \mathsf{LC}(Q, A') = \emptyset$.
- $\alpha \in A$. In this case $Q = \alpha_u.P_1$ and $\mathsf{UC}(Q, A') = \emptyset$ for all A'.For $A \subseteq A'$, we have $\alpha \in A'$ and, by Definition 3.4 $\mathsf{LC}(Q, A') = \emptyset$.

Sum: $P = P_1 +_u P_2$ with $P_i \in \mathsf{L}_{ui}(\tilde{\mathbb{P}}_1)$ for $i = 1, 2$. Consider the following cases:

- $(\mathcal{A}(P_1) \cup \mathcal{A}(P_2)) \backslash A \neq \emptyset$. In this case $Q = P_1 \underline{+}_u P_2$. Moreover $\mathsf{LC}(P_1, A') \cup \mathsf{LC}(P_2, A') \neq \emptyset$ implies $\mathsf{UC}(Q, A') = \mathsf{LC}(Q, A') = \{u\}$, otherwise $\mathsf{UC}(Q, A') = \mathsf{LC}(Q, A') = \emptyset$.
- $(\mathcal{A}(P_1) \cup \mathcal{A}(P_2)) \backslash A = \emptyset$. In this case $Q = P_1 +_u P_2$ and, by Definition 3.5, $\mathsf{UC}(Q, A') = \emptyset$ for all A'. For $A \subseteq A'$, we have $(\mathcal{A}(P_1) \cup \mathcal{A}(P_2)) \backslash A' \subseteq (\mathcal{A}(P_1) \cup \mathcal{A}(P_2)) \backslash A = \emptyset$. Moreover, by Lemma 8.2-4, $(\mathcal{A}(P_1) \cup \mathcal{A}(P_2)) \backslash A' = \mathcal{A}(Q) \backslash A' = \mathcal{A}(Q, A')$. Finally, $\mathcal{A}(Q, A') = \emptyset$ and Lemma 9.4-1 imply $\mathsf{LC}(Q, A') = \emptyset$.

Par: $P = P_1 \parallel_B^u P_2$. Let $A_1 = (\mathcal{A}(P_1) \backslash \mathcal{A}(P_2) \cap B$ and $A_2 = (\mathcal{A}(P_2) \backslash \mathcal{A}(P_1) \cap B$. In this case $Q = Q_1 \parallel_B^u Q_2$ with $Q_i = \mathsf{urgent}(P_i, A \cup A_i)$ for $i = 1, 2$. Since $A \cup A_i \subseteq A' \cup A_i$ for $i = 1, 2$, we get by induction hypothesis: $\mathsf{UC}(Q, A') = \mathsf{UC}(Q_1, A' \cup A_1) \cup \mathsf{UC}(Q_2, A' \cup A_2) = \mathsf{LC}(Q_1, A' \cup A_1) \cup \mathsf{LC}(Q_2, A' \cup A_2) = \mathsf{LC}(Q, A')$.

Rel: $P = P_1[\Phi_u]$. Here $Q = Q_1[\Phi_u]$ where $Q_1 = \mathsf{urgent}(P_1, \Phi^{-1}(A))$. Since $\Phi^{-1}(A) \subseteq \Phi^{-1}(A')$, we get by induction hypothesis: $\mathsf{UC}(Q, A') = \mathsf{UC}(Q_1, \Phi^{-1}(A')) = \mathsf{LC}(Q_1, \Phi^{-1}(A')) = \mathsf{LC}(Q, A')$.

Rec: $P = \mathsf{rec}\, x_u.P_1$. In this case $Q = \mathsf{rec}\, x_u.Q_1$ where $Q_1 = \mathsf{urgent}(P_1, A)$. By induction hypothesis $\mathsf{UC}(Q, A') = \mathsf{UC}(Q_1, A') = \mathsf{LC}(Q_1, A') = \mathsf{LC}(Q, A')$.

□

The following corollary states some properties of the urgent versions of initial processes that can be derived directly from Lemmas 8.7 and 9.6.

Corollary 9.7 Let $P \in \mathsf{L}(\tilde{\mathbb{P}}_1)$, $Q = \mathsf{urgent}(P) \in \mathsf{L}(\tilde{\mathbb{P}})$ and $A \subseteq \mathbb{A}$. Then:

1. $\mathcal{A}(P, A) = \mathcal{A}(Q, A)$;

2. $P = \mathsf{unmark}(Q)$ and $\mathsf{clean}(Q) = Q$;

3. $\mathsf{UC}(Q, A) = \mathsf{LC}(Q, A)$.

4. $Q = P$ iff $\mathcal{A}(P) = \mathcal{A}(Q) = \emptyset$

Lemma 9.8 Let $Q \in \mathsf{L}(\tilde{\mathbb{P}})$ and $A \subseteq A' \subseteq \mathbb{A}$. Then $\mathsf{UC}(\mathsf{clean}(Q, A), A') = \mathsf{UC}(Q, A')$.

Proof: By induction on $Q \in \mathsf{L}(\tilde{\mathbb{P}})$.

Nil, Var: $Q = \mathsf{nil}_u$, $Q = x_u$. In this case, for all $A, A' \subseteq \mathbb{A}$, $\mathsf{UC}(\mathsf{clean}(Q, A), A') = \mathsf{UC}(Q, A') = \emptyset$.

Pref: $Q = \alpha_u.P_1$ or $P = \underline{\alpha}_u.P_1$. The former case is simple; let us prove the latter one. Consider the following subcases:

- $\alpha \in A$. Then $A \subseteq A'$ implies $\alpha \in A'$ and $\mathsf{UC}(\mathsf{clean}(Q, A), A') = \mathsf{UC}(\alpha_u.P_1, A) = \emptyset = \mathsf{UC}(Q, A')$.
- $\alpha \notin A$. In this case $\mathsf{clean}(Q, A) = Q$ and the statement easily follows.

Sum: $Q = P_1 +_u P_2$ or $Q = P_1 \underline{+}_u P_2$. Again we only prove the latter case. Consider the following subcases:

- $\mathcal{A}(P_1) \cup \mathcal{A}(P_2) \subseteq A$. In this case $\mathcal{A}(Q) = \mathcal{A}(P_1) \cup \mathcal{A}(P_2)$ and $A \subseteq A'$ implies $\mathcal{A}(Q) \subseteq A'$ and, by Lemma 8.2-4, $\mathcal{A}(Q, A') = \mathcal{A}(Q) \backslash A' = \emptyset$. Thus, by Lemma 9.4-1, $\mathsf{LC}(Q, A') = \emptyset$ and $\mathsf{UC}(Q, A') \subseteq \mathsf{LC}(Q, A')$ (see Lemma 9.5) implies also $\mathsf{UC}(Q, A') = \emptyset$. Morever, by Definition 2.3, $\mathsf{UC}(\mathsf{clean}(Q, A), A') = \mathsf{UC}(P_1 +_u P_2, A') = \emptyset$.
- $\mathcal{A}(P_1) \cup \mathcal{A}(P_2) \not\subseteq A$. In this case $\mathsf{clean}(Q, A) = Q$ and the statement easily follows.

Par: $Q = Q_1 \parallel^u_B Q_2$. Let $A_1 = (\mathcal{A}(Q_1) \backslash \mathcal{A}(Q_2)) \cap B$, $A_2 = (\mathcal{A}(Q_2) \backslash \mathcal{A}(Q_1)) \cap B$. By Definition 2.3, $\mathsf{clean}(Q, A) = \mathsf{clean}(Q_1, A \cup A_1) \parallel^u_B \mathsf{clean}(Q_2, A \cup A_2) = Q'_1 \parallel^u_B Q'_2$. Moreover, by Lemma 8.6-4, $A_1 = (\mathcal{A}(Q'_1) \backslash \mathcal{A}(Q'_2)) \cap B$ and $A_2 = (\mathcal{A}(Q'_2) \backslash \mathcal{A}(Q'_1)) \cap B$. Thus $\mathsf{UC}(\mathsf{clean}(Q, A), A') = \mathsf{UC}(Q'_1 \parallel^u_B Q'_2, A') = \mathsf{UC}(Q'_1, A' \cup A_1) \cup \mathsf{UC}(Q'_2, A' \cup A_2)$. Finally, $A \cup A_i \subseteq A' \cup A_i$ implies, by induction hypothesis, $\mathsf{UC}(Q'_i, A' \cup A_i) = \mathsf{UC}(\mathsf{clean}(Q_i, A \cup A_i), A' \cup A_i) = \mathsf{UC}(Q_i, A' \cup A_i)$ and, hence, $\mathsf{UC}(\mathsf{clean}(Q, A), A') = \mathsf{UC}(Q_1, A' \cup A_1) \cup \mathsf{UC}(Q_2, A' \cup A_2) = \mathsf{UC}(Q, A')$.

Rel: $Q = Q_1[\Phi_u]$. Since $\Phi^{-1}(A) \subseteq \Phi^{-1}(A')$, we get by induction hypothesis: $\mathsf{UC}(\mathsf{clean}(Q, A), A') = \mathsf{UC}(\mathsf{clean}(Q_1, \Phi^{-1}(A))[\Phi_u], A') = \mathsf{UC}(\mathsf{clean}(Q_1, \Phi^{-1}(A)), \Phi^{-1}(A')) = \mathsf{UC}(Q_1, \Phi^{-1}(A')) = \mathsf{UC}(Q, A')$.

Rec: $Q = \mathsf{rec}\, x_u.Q_1$. By induction hypothesis we have

$\mathsf{UC}(\mathsf{clean}(Q, A), A') = \mathsf{UC}(\mathsf{rec}\ x_u.\mathsf{clean}(Q_1, A), A') = \mathsf{UC}(\mathsf{clean}(Q_1, A), A') = \mathsf{UC}(Q_1, A') = \mathsf{UC}(Q, A')$.

□

Corollary 9.9 Let $Q \in \mathsf{L}(\tilde{\mathbb{P}})$ and $A \subseteq \mathbb{A}$. Then $\mathsf{UC}(\mathsf{clean}(Q), A) = \mathsf{UC}(Q, A)$.

Lemma 9.10 Let $P \in \mathsf{L}(\tilde{\mathbb{P}}_1)$, $Q \in \mathsf{L}(\tilde{\mathbb{P}})$, $x \in \mathcal{X}$ guarded in Q. Then $\mathsf{UC}(Q\{\!|P/x|\!\}, A) = \mathsf{UC}(Q, A)$.

Proof: This statement can be proved similarly to Lemma 8.9 and its proof is omitted. □

Lemma 9.11 Let $Q, R \in \mathsf{L}(\tilde{\mathbb{P}})$ and $A \subseteq \mathbb{A}$ such that $Q = \mathsf{clean}(R, A)$. Then:

1. $\mathsf{UC}(Q, A) = \emptyset$ implies $Q \in \mathsf{L}(\tilde{\mathbb{P}}_1)$;
2. $\mathsf{UC}(Q, A') \subseteq \mathsf{UC}(Q, A)$ for all $A' \subseteq \mathbb{A}$;
3. $Q \xrightarrow{\alpha} Q'$ implies $\mathsf{UC}(Q', A') \subseteq \mathsf{UC}(Q, A)$ for all $A' \subseteq \mathbb{A}$.

Proof: We prove Items 1 and 2 by induction on R and Item 3 by induction on the depth of derivation $Q \xrightarrow{\alpha} Q'$. We proceed by case analysis on structure of R.

Nil, Var: $R = \mathsf{nil}_u$, $R = x_u$. In these cases $Q = \mathsf{clean}(R, A)$ implies $Q = R$.

1. $\mathsf{UC}(Q, A) = \emptyset$ and $Q \in \mathsf{L}(\tilde{\mathbb{P}}_1)$.
2. For all $A' \subseteq \mathbb{A}$ we have $\mathsf{UC}(Q, A') = \emptyset = \mathsf{UC}(Q, A)$.
3. This case is not possible since $Q \not\xrightarrow{\alpha}$.

Pref: $R = \alpha_u.P_1$ or $R = \underline{\alpha}_u.P_1$. We prove only the latter case (the former one is simpler). Assume $Q = \mathsf{clean}(R, A)$ and consider the following subcases:

- $\alpha \notin A$. $Q = \underline{\alpha}_u.P_1$ and $\mathsf{UC}(Q, A) = \{u\}$.
  1. This case is not possible since $\mathsf{UC}(Q, A) \neq \emptyset$.
  2. For all $A' \subseteq \mathbb{A}$ we have $\mathsf{UC}(Q, A') \subseteq \{u\} = \mathsf{UC}(Q, A)$ (see Definition 3.5).
  3. $Q \xrightarrow{\alpha} P_1 \in \mathsf{L}(\tilde{\mathbb{P}}_1)$. Then, by Lemma 9.5, $\mathsf{UC}(P_1, A') = \emptyset$ and, hence $\mathsf{UC}(P_1, A') \subseteq \mathsf{UC}(Q, A)$ for all $A' \subseteq \mathbb{A}$.
- $\alpha \in A$. $Q = \alpha_u.P_1$ and $\mathsf{UC}(Q, A) = \emptyset$.
  1. $\mathsf{UC}(Q, A) = \emptyset$ and $Q \in \mathsf{L}(\tilde{\mathbb{P}}_1)$.
  2. For all $A' \subseteq \mathbb{A}$ we have that $\mathsf{UC}(Q, A') = \emptyset = \mathsf{UC}(Q, A)$.
  3. similar to case 3. above

Sum: $R = P_1 +_u P_2$ or $R = P_1 \underline{+}_u P_2$. Again we prove only the latter case. Assume $Q = \mathsf{clean}(R, A)$ and consider the following subcases:

- $\mathcal{A}(P_1) \cup \mathcal{A}(P_2) \not\subseteq A$. In this case $Q = P_1 \underline{+}_u P_2$ and there exists $\alpha \in \mathbb{A}_\tau$ such that either $\alpha \in \mathcal{A}(P_1)$ and $\alpha \notin A$ or $\alpha \in \mathcal{A}(P_2)$ and $\alpha \notin A$. Consider the former case (the latter case is similar). $\alpha \in \mathcal{A}(P_1)$ and $\alpha \notin A$ implies, by Lemma 8.2-2, $\alpha \in \mathcal{A}(P_1, A) \neq \emptyset$ and hence, by Lemma 9.4-1, $\mathsf{LC}(P_1, A) \neq \emptyset$. Finally, by Definition 3.5, $\mathsf{UC}(Q, A) = \{u\}$.
  1. This case is not possible since $\mathsf{UC}(Q, A) \neq \emptyset$.
  2. Again by Definition 3.5, for all $A' \subseteq \mathbb{A}$, $\mathsf{UC}(Q, A') \subseteq \{u\} = \mathsf{UC}(Q, A)$.
  3. $Q \xrightarrow{\alpha} Q'$ implies either $P_1 \xrightarrow{\alpha} Q'$ or $P_2 \xrightarrow{\alpha} Q'$. In both case $P_i \in \mathsf{L}(\tilde{\mathbb{P}}_1)$ and Proposition 8.1 imply $Q' \in \mathsf{L}(\tilde{\mathbb{P}}_1)$. Again by Lemma 9.5 we have that $\mathsf{UC}(Q', A') = \emptyset$ and, clearly, $\mathsf{UC}(Q', A') \subseteq \mathsf{UC}(Q, A)$ for all $A' \subseteq \mathbb{A}$.
- $\mathcal{A}(P_1) \cup \mathcal{A}(P_2) \subseteq A$. In this case $Q = P_1 +_u P_2$ and $\mathsf{UC}(Q, A) = \emptyset$.
  1. $\mathsf{UC}(Q, A) = \emptyset$ and $Q \in \mathsf{L}(\tilde{\mathbb{P}}_1)$.
  2. As above $\mathsf{UC}(Q, A') = \emptyset = \mathsf{UC}(Q, A)$ for all $A' \subseteq \mathbb{A}$.
  3. similar to case 3. above

Par: $R = R_1 \|^u_B R_2$. Let $A_1 = (\mathcal{A}(R_1) \backslash \mathcal{A}(R_2)) \cap B$ and $A_2 = (\mathcal{A}(R_2) \backslash \mathcal{A}(R_1)) \cap B$. In this case $Q = \mathsf{clean}(R, A)$ implies $Q = Q_1 \|^u_B Q_2$ where $Q_i = \mathsf{clean}(R_i, A \cup A_i)$ for $i = 1, 2$. Moreover, by Lemma 8.6-4, we have that $A_1 = (\mathcal{A}(Q_1) \backslash \mathcal{A}(Q_2)) \cap B$ and $A_2 = (\mathcal{A}(Q_2) \backslash \mathcal{A}(Q_1)) \cap B$.

1. $\mathsf{UC}(Q, A) = \emptyset$ implies $\mathsf{UC}(Q_1, A \cup A_1) = \mathsf{UC}(Q_2, A \cup A_2) = \emptyset$. By induction hypothesis $Q_1, Q_2 \in \mathsf{L}(\tilde{\mathbb{P}}_1)$ and, hence, $Q \in \mathsf{L}(\tilde{\mathbb{P}}_1)$.

2. Let $A' \subseteq \mathbb{A}$. By induction hypothesis we have that $\mathsf{UC}(Q_1, A' \cup A_1) \subseteq \mathsf{UC}(Q_1, A \cup A_1)$ and $\mathsf{UC}(Q_2, A' \cup A_2) \subseteq \mathsf{UC}(Q_2, A \cup A_2)$. Thus $\mathsf{UC}(Q, A') = \mathsf{UC}(Q_1, A' \cup A_1) \cup \mathsf{UC}(Q_2, A' \cup A_2) \subseteq \mathsf{UC}(Q_1, A \cup A_1) \cup \mathsf{UC}(Q_2, A \cup A_2) = \mathsf{UC}(Q, A)$.
3. Assume that $Q \xrightarrow{\alpha} Q'$ and consider the following cases:
   - $\alpha \notin B$, $Q_1 \xrightarrow{\alpha} Q_1'$ and $Q' = \mathsf{clean}(Q_1' \parallel_B^u Q_2)$. Let $A_1' = (\mathcal{A}(Q_1') \backslash \mathcal{A}(Q_2)) \cap B$ and $A_2' = (\mathcal{A}(Q_2) \backslash \mathcal{A}(Q_1')) \cap B$. Let, moreover, $A' \subseteq \mathbb{A}$. By Corollary 9.9 we have that $\mathsf{UC}(Q', A') = \mathsf{UC}(Q_1' \parallel_B^u Q_2, A') = \mathsf{UC}(Q_1', A' \cup A_1') \cup \mathsf{UC}(Q_2, A' \cup A_2')$. Moreover:
     - by induction hypothesis $\mathsf{UC}(Q_1', A' \cup A_1') \subseteq \mathsf{UC}(Q_1, A \cup A_1)$, and
     - by Item 2 $\mathsf{UC}(Q_2, A' \cup A_2') \subseteq \mathsf{UC}(Q_2, A \cup A_2)$.

     Thus $\mathsf{UC}(Q', A') \subseteq \mathsf{UC}(Q_1, A \cup A_1) \cup \mathsf{UC}(Q_2, A \cup A_2) = \mathsf{UC}(Q, A)$.
   - $\alpha \notin B$, $Q_2 \xrightarrow{\alpha} Q_2'$ and $Q' = \mathsf{clean}(Q_1 \parallel_B^u Q_2')$. Similar to the previous case.
   - $\alpha \in B$, $Q_i \xrightarrow{\alpha} Q_i'$, for $i = 1, 2$ and $Q' = \mathsf{clean}(Q_1' \parallel_B^u Q_2')$. Let $A' \subseteq \mathbb{A}$. As above $\mathsf{UC}(Q', A') = \mathsf{UC}(Q_1' \parallel_B^u Q_2', A') = \mathsf{UC}(Q_1', A' \cup A_1') \cup \mathsf{UC}(Q_2', A' \cup A_2')$ where $A_1' = (\mathcal{A}(Q_1') \backslash \mathcal{A}(Q_2')) \cap B$ and $A_2' = (\mathcal{A}(Q_2') \backslash \mathcal{A}(Q_1')) \cap B$. Moreover, by induction hypothesis, $\mathsf{UC}(Q_1', A' \cup A_1') \subseteq \mathsf{UC}(Q_1, A \cup A_1)$ and $\mathsf{UC}(Q_2', A' \cup A_2') \subseteq \mathsf{UC}(Q_2, A \cup A_2)$. Thus $\mathsf{UC}(Q', A') \subseteq \mathsf{UC}(Q_1, A \cup A_1) \cup \mathsf{UC}(Q_2, A \cup A_2) = \mathsf{UC}(Q, A)$.

Rel: $R = R_1[\Phi_u]$. In this case $Q = \mathsf{clean}(R, A)$ implies $Q = Q_1[\Phi_u]$ where $Q_1 = \mathsf{clean}(R_1, \Phi^{-1}(A))$.

1. $\mathsf{UC}(Q, A) = \mathsf{UC}(Q_1, \Phi^{-1}(A)) = \emptyset$ implies, by induction hypothesis, $Q_1 \in \mathsf{L}(\tilde{\mathbb{P}}_1)$ and, hence, $Q \in \mathsf{L}(\tilde{\mathbb{P}}_1)$.
2. By induction hypothesis $\mathsf{UC}(Q, A') = \mathsf{UC}(Q_1, \Phi^{-1}(A')) \subseteq \mathsf{UC}(Q_1, \Phi^{-1}(A)) = \mathsf{UC}(Q, A)$ for any $A' \subseteq \mathbb{A}$.
3. In this case $Q \xrightarrow{\alpha} Q'$ if there exists $\beta \in \Phi^{-1}(\alpha)$ such that $Q_1 \xrightarrow{\alpha} Q_1'$ and $Q' = Q_1'[\Phi_u]$. Let $A' \subseteq \mathbb{A}$. Then, by induction hypothesis, $\mathsf{UC}(Q', A') = \mathsf{UC}(Q_1', \Phi^{-1}(A')) \subseteq \mathsf{UC}(Q_1, \Phi^{-1}(A)) = \mathsf{UC}(Q, A)$.

Rec: $R = \mathsf{rec}\, x_u.R_1$. In this case $Q = \mathsf{clean}(R, A)$ implies $Q = \mathsf{rec}\, x_u.Q_1$ where $Q_1 = \mathsf{clean}(R_1, A)$.

1. $\mathsf{UC}(Q, A) = \mathsf{UC}(Q_1, A) = \emptyset$ implies, by induction hypothesis, $Q_1 \in \mathsf{L}(\tilde{\mathbb{P}}_1)$ and, hence, $Q \in \mathsf{L}(\tilde{\mathbb{P}}_1)$.
2. Let $A' \subseteq \mathbb{A}$. By induction hypothesis $\mathsf{UC}(Q, A') = \mathsf{UC}(Q_1, A') \subseteq \mathsf{UC}(Q_1, A) = \mathsf{UC}(Q, A)$.
3. Let $P = \mathsf{unmark}(Q_1)$ and $S = Q_1\{\!|\mathsf{rec}\, x_u.P/x|\!\} \xrightarrow{\alpha} Q'$; x guarded in R_1 implies x guarded in $Q_1 = \mathsf{clean}(R_1, A)$ and, by Lemma 9.10, $\mathsf{UC}(S, A) = \mathsf{UC}(Q_1, A) = \mathsf{UC}(Q, A)$. Moreover x guarded in R_1 and Lemma 8.9-2 imply $S = Q_1\{\!|\mathsf{rec}\, x_u.P/x|\!\} = \mathsf{clean}(R_1, A)\{\!|\mathsf{rec}\, x_u.P/x|\!\} = \mathsf{clean}(R_1\{\!|\mathsf{rec}\, x_u.P/x|\!\}, A)$. Now assume $Q \xrightarrow{\alpha} Q'$. Then, by operational semantics, we also have $S \xrightarrow{\alpha} Q'$. By induction hypothesis $\mathsf{UC}(Q', A') \subseteq \mathsf{UC}(S, A) = \mathsf{UC}(Q, A)$ for any $A' \subseteq \mathbb{A}$.

□

Lemma 9.12 Let $Q \in \mathsf{L}(\tilde{\mathbb{P}})$ and $A, A' \subseteq \mathbb{A}$. Then $v \in \mathsf{UC}(Q, A)$ and $v \notin \mathsf{UC}(Q, A')$ implies $v \notin \mathsf{LC}(Q, A')$.

Proof: By induction on the structure of Q.

Nil, Var: $Q = \mathsf{nil}_u$, $Q = x_u$. These cases are not possible since $\mathsf{UC}(Q, A) = \emptyset$ for all A.

Pref: $Q = \alpha.P_1$ or $Q = \underline{\alpha}_u.P_1$. Clearly only the latter case is possible. In this case $v \in \mathsf{UC}(Q, A)$ implies $\alpha \notin A$ and $v = u$. Thus $v = u \notin \mathsf{UC}(Q, A')$ implies $\alpha \in A'$ and, hence, $v \notin \mathsf{LC}(Q, A') = \emptyset$.

Sum: $Q = P_1 +_u P_2$ or $Q = P_1 \underline{+}_u P_2$. Similar to the Pref-case.

Par: $Q = Q_1 \|^u_B Q_2$. Let $A_1 = (\mathcal{A}(Q_1) \backslash \mathcal{A}(Q_2)) \cap B$ and $A_2 = (\mathcal{A}(Q_2) \backslash \mathcal{A}(Q_1)) \cap B$. Assume $v \in \mathsf{UC}(Q, A)$, $v \notin \mathsf{UC}(Q, A')$ and consider the following cases:

1. $v \in \mathsf{UC}(Q_1, A \cup A_1)$. In this case $v \notin \mathsf{UC}(Q, A')$ implies $v \notin \mathsf{UC}(Q_1, A' \cup A_1)$ and, by induction hypothesis $v \notin \mathsf{LC}(Q_1, A' \cup A_1)$. Moreover $v \in \mathsf{UC}(Q_1, A \cup A_1)$ and Fact 9.2-1 imply that $v \notin \mathsf{LAB}(Q_2)$ and, by Lemma 9.5-2, $v \notin \mathsf{LC}(Q_2, A' \cup A_2)$. Thus we can conclude $v \notin \mathsf{LC}(Q, A')$.
2. $v \in \mathsf{UC}(Q_2, A \cup A_2)$. Similar to the previous case.

Rel: $Q = Q_1[\Phi_u]$. By induction hypothesis $v \in \mathsf{UC}(Q, A) = \mathsf{UC}(Q_1, \Phi^{-1}(A))$ and $v \notin \mathsf{UC}(Q, A') = \mathsf{UC}(Q_1, \Phi^{-1}(A'))$ implies $v \notin \mathsf{LC}(Q_1, \Phi^{-1}(A')) = \mathsf{LC}(Q, A')$.

Rec: $Q = \mathsf{rec}\, x_u.Q_1$. By induction hypothesis $v \in \mathsf{UC}(Q, A) = \mathsf{UC}(Q_1, A)$ and $v \notin \mathsf{UC}(Q, A') = \mathsf{UC}(Q_1, A')$ implies $v \notin \mathsf{LC}(Q_1, A') = \mathsf{LC}(Q, A')$.

□

Lemma 9.13 Let $Q \in \mathsf{L}(\tilde{\mathbb{P}})$ and $A, A' \subseteq \mathbb{A}$. Assume that $Q \xrightarrow{\alpha} Q'$. Then $v \in \mathsf{UC}(Q, A)$ and $v \notin \mathsf{UC}(Q', A')$ implies $v \notin \mathsf{LC}(Q', A')$.

Proof: The proof is by induction on the depth of derivation $Q \xrightarrow{\alpha} Q'$. We proceed by case analysis on the structure of Q.

Nil, Var: $Q = \mathsf{nil}_u$, $Q = x_u$. These cases are not possible since $Q \not\xrightarrow{\alpha}$.

Pref: $Q = \alpha.P_1$ or $Q = \underline{\alpha}_u.P_1$. In this case $Q \xrightarrow{\alpha} Q' = P_1 \in \mathsf{L}_{u1}(\tilde{\mathbb{P}}_1)$. Clearly, only the latter case is possible. Moreover $v \in \mathsf{UC}(Q, A)$ implies $\alpha \notin A$ and $v = u$. Thus $u < u1$ and Fact 9.2-2 give $v \notin \mathsf{LAB}(P_1)$ and, hence, $v \notin \mathsf{LC}(Q', A')$.

Sum: $Q = P_1 +_u P_2$ or $Q = P_1 \underline{+}_u P_2$. Similar to the Pref-case.

Par: $Q = Q_1 \|^u_B Q_2$. Let $A_1 = (\mathcal{A}(Q_1) \backslash \mathcal{A}(Q_2)) \cap B$ and $A_2 = (\mathcal{A}(Q_2) \backslash \mathcal{A}(Q_1)) \cap B$. Assume $Q \xrightarrow{\alpha} Q'$ and consider the following case:

1. $\alpha \notin B$, $Q_1 \xrightarrow{\alpha} Q'_1$ and $Q' = \mathsf{clean}(Q'_1 \|_B Q_2)$. Let $A'_1 = (\mathcal{A}(Q'_1) \backslash \mathcal{A}(Q_2)) \cap B$ and $A'_2 = (\mathcal{A}(Q_2) \backslash \mathcal{A}(Q'_1)) \cap B$. Now, assume $v \in \mathsf{UC}(Q, A)$, $v \notin \mathsf{UC}(Q', A') = \mathsf{UC}(Q'_1 \|_B Q_2, A')$ (by Corollary 9.9) and consider the following subcases:
   1.1 $v \in \mathsf{UC}(Q_1, A \cup A_1)$. In this case $v \notin \mathsf{UC}(Q', A')$ implies $v \notin \mathsf{UC}(Q'_1, A' \cup A'_1)$. By induction hypothesis $v \notin \mathsf{LC}(Q'_1, A' \cup A'_1)$. Moreover $v \in \mathsf{UC}(Q_1, A \cup A_1)$ and Fact 9.2-1 imply that $v \notin \mathsf{LAB}(Q_2)$ and, by Lemma 9.5-2, $v \notin \mathsf{LC}(Q_2, A' \cup A'_2)$. Thus we can conclude $v \notin \mathsf{LC}(Q', A')$.
   1.2 $v \in \mathsf{UC}(Q_2, A \cup A_2)$. In this case $v \notin \mathsf{UC}(Q', A')$ implies $v \notin \mathsf{UC}(Q_2, A' \cup A'_2)$ and, by Lemma 9.12, $v \notin \mathsf{LC}(Q_2, A' \cup A'_2)$. Similar to the previous case we can prove that $v \in \mathsf{UC}(Q_2, A \cup A_2)$ implies $v \notin \mathsf{LC}(Q'_1, A' \cup A'_1)$ and, hence, $v \notin \mathsf{LC}(Q', A')$.
2. $\alpha \notin B$, $Q_2 \xrightarrow{\alpha} Q'_2$ and $Q' = \mathsf{clean}(Q_1 \|_B Q'_2)$. Similar to the previous case.

3. $\alpha \in B$, $Q_i \xrightarrow{\alpha} Q_i'$, for $i = 1, 2$ and $Q' = \mathsf{clean}(Q_1' \parallel_B Q_2')$.
   Let $A_1' = (\mathcal{A}(Q_1') \backslash \mathcal{A}(Q_2')) \cap B$ and $A_2' = (\mathcal{A}(Q_2') \backslash \mathcal{A}(Q_1')) \cap B$. Now, assume $v \in \mathsf{UC}(Q, A)$, $v \notin \mathsf{UC}(Q', A') = \mathsf{UC}(Q_1' \|_B Q_2', A')$, by Corollary 9.9, and consider the following subcases:
   3.1 $v \in \mathsf{UC}(Q_1, A \cup A_1)$. In this case $v \notin \mathsf{UC}(Q', A')$ implies $v \notin \mathsf{UC}(Q_1', A' \cup A_1')$. By induction hypothesis $v \notin \mathsf{LC}(Q_1', A' \cup A_1')$. Moreover, since $Q_1 \in \mathsf{L}_w(\tilde{\mathbb{P}})$ with $u1 \leq w$ by the labelling definition, we have that $v \in \mathsf{UC}(Q_1, A \cup A_1)$ implies, by Lemma 9.5-2 and Fact 9.2-2, $u1 \leq w \leq v$. On the other hand, since $Q_2 \in \mathsf{L}_{w'}(\tilde{\mathbb{P}})$ with $u2 \leq w'$ (again by the labelling definition), $Q_2 \xrightarrow{\alpha} Q_2'$ and by Fact 9.3 imply $Q_2' \in \mathsf{L}_{w''}(\tilde{\mathbb{P}})$ with $u2 \leq w' \leq w''$. Thus, again by Fact 9.2-2, $v \notin \mathsf{LAB}(Q_2)$ and, as in the previous cases $v \notin \mathsf{LC}(Q_2, A' \cup A_2')$. Thus we can conclude $v \notin \mathsf{LC}(Q', A')$.
   3.2 $v \in \mathsf{UC}(Q_2, A \cup A_2)$. Similar to the previous case

Rel: $Q = Q_1[\Phi_u]$. In this case $Q \xrightarrow{\alpha} Q'$ if there exists $\beta \in \Phi^{-1}(\alpha)$ such that $Q_1 \xrightarrow{\alpha} Q_1'$ and $Q' = Q_1'[\Phi_u]$. By induction hypothesis $v \in \mathsf{UC}(Q, A) = \mathsf{UC}(Q_1, \Phi^{-1}(A))$ and $v \notin \mathsf{UC}(Q', A') = \mathsf{UC}(Q_1', \Phi^{-1}(A'))$ implies $v \notin \mathsf{LC}(Q_1', \Phi^{-1}(A')) = \mathsf{LC}(Q', A')$

Rec: $Q = \mathsf{rec}\, x_u.Q_1$. In this case $Q \xrightarrow{\alpha} Q'$ if $S = Q_1\{\!|\mathsf{rec}\, x_u.\mathsf{unmark}(Q_1)/x|\!\} \xrightarrow{\alpha} Q'$. Now assume $v \in \mathsf{UC}(Q, A)$ and $v \notin \mathsf{UC}(Q', A')$. Since $\mathsf{UC}(Q, A) = \mathsf{UC}(Q_1, A)$ and, by Lemma 9.10, x guarded in Q_1 implies $\mathsf{UC}(Q_1, A) = \mathsf{UC}(S, A)$, we also have $v \in \mathsf{UC}(S, A)$ and $v \notin \mathsf{UC}(Q', A')$. Hence, by induction hypothesis, $v \notin \mathsf{LC}(Q', A')$.

□

Now we are ready to prove the main result of this section.

Proposition 9.1 Let $P_0 \in \mathsf{L}(\tilde{\mathbb{P}}_1), Q_0 \in \mathsf{L}(\tilde{\mathbb{P}})$ such that $Q_0 = \mathsf{urgent}(P_0)$. Then $Q_0 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} Q_n$ implies

1. $\mathsf{UC}(Q_{i+1}) \subseteq \mathsf{UC}(Q_i)$ for every $i \in [0, n-1]$. Moreover, $\mathsf{UC}(Q_n) = \emptyset$ implies $Q_n \in \mathsf{L}(\tilde{\mathbb{P}}_1)$;
2. $(\mathsf{LC}(Q_0) \cap \ldots \cap \mathsf{LC}(Q_n)) \backslash \mathsf{UC}(Q_n) = \emptyset$.

Proof: We prove both items by induction on the length of derivation $Q_0 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} Q_n$:

1. We distinguish two cases:
   - $n = 0$. Assume $\mathsf{UC}(Q_n) = \mathsf{UC}(Q_0) = \emptyset$. Then, since $\mathsf{UC}(Q_0) = \mathsf{LC}(Q_0)$ (see Corollary 9.7-3), we also have $\mathsf{LC}(Q_0) = \emptyset$ and, by Corollary 9.7-1 and Lemma 9.4-1, $\mathcal{A}(P_0) = \mathcal{A}(Q_0) = \emptyset$. Finally, by Corollary 9.7-4, we can conclude that $Q_0 = P_0 \in \mathsf{L}(\tilde{\mathbb{P}}_1)$
   - $n \geq 1$. $Q_0 = \mathsf{urgent}(P_0)$ and Corollary 9.7-2 imply that $\mathsf{clean}(Q_0) = Q_0$. Moreover $Q_{j-1} \xrightarrow{\alpha_j} Q_j$ and Lemma 8.15-1 imply $Q_j = \mathsf{clean}(Q_j)$ for every $j \in [1, n]$. Now, let $i \in [0, n-1]$. By Lemma 9.11-3, $Q_i = \mathsf{clean}(Q_i)$ and $Q_i \xrightarrow{\alpha_{i+1}} Q_{i+1}$ implies $\mathsf{UC}(Q_{i+1}) \subseteq \mathsf{UC}(Q_i)$. Moreover $Q_n = \mathsf{clean}(Q_n)$ and $\mathsf{UC}(Q_n) = \emptyset$ implies $Q_n \in \mathsf{L}(\tilde{\mathbb{P}}_1)$ (by 9.11-1).
2. As in the previous item we have two cases to consider:
   - $n = 0$. Again $Q_n = Q_0$. Thus, $(\mathsf{LC}(Q_0) \cap \ldots \cap \mathsf{LC}(Q_n)) \backslash \mathsf{UC}(Q_n) = \mathsf{LC}(Q_0) \backslash \mathsf{UC}(Q_0) = \emptyset$, by Corollary 9.7-3.
   - $n \geq 1$. Assume $Q_0 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} Q_n$, and, by contradiction, $v \in \mathsf{LC}(Q_0) \cap \ldots \cap \mathsf{LC}(Q_n)$ but $v \notin \mathsf{UC}(Q_n)$. By Corollary 9.7-3, $\mathsf{LC}(Q_0) = \mathsf{UC}(Q_0)$. Thus we have that $v \in \mathsf{UC}(Q_0)$ and $v \notin \mathsf{UC}(Q_n)$. Now, let i be the smallest index in $[1, n]$ such that $v \notin \mathsf{UC}(Q_i)$. By Lemma 9.13, $Q_{i-1} \xrightarrow{\alpha_i} Q_i$, $v \in \mathsf{UC}(Q_{i-1})$ and $v \notin \mathsf{UC}(Q_i)$ imply $v \notin \mathsf{LC}(Q_i)$ and, hence, $v \notin \mathsf{LC}(Q_0) \cap \ldots \cap \mathsf{LC}(Q_n)$, contradicting the hypothesis.

□

10 Appendix C: Proofs of Statements in Section 4

There is one more proposition needed for the main proofs that we have postponed:

Lemma 10.1 Let $Q \in \mathsf{L}(\tilde{\mathbb{P}})$ and $P = \mathsf{unmark}(Q) \in \mathsf{L}(\tilde{\mathbb{P}}_1)$. Then

1. $\mathsf{LC}(Q, A) = \mathsf{LC}(P, A)$;
2. $Q \xrightarrow{\alpha} Q'$ implies $P \xrightarrow{\alpha} P'$ and $P' = \mathsf{unmark}(Q')$;
3. $P \xrightarrow{\alpha} P'$ implies $Q \xrightarrow{\alpha} Q'$ and $P' = \mathsf{unmark}(Q')$.

Proof: We prove Item 1 by induction on $Q \in \mathsf{L}(\tilde{\mathbb{P}})$ and Item 2 by induction on derivation $Q \xrightarrow{\alpha} Q'$. The proof of Item 3 is omitted since it is similar to the proof of Item 2. We proceed by case analysis on the structure of Q

Nil, Var: $Q = \mathsf{nil}_u$, $Q = x_u$. In these case $P = \mathsf{unmark}(Q)$ implies $P = Q$.

1. $\mathsf{LC}(Q, A) = \mathsf{LC}(P, A) = \emptyset$.
2. This case is not possible since $Q \not\xrightarrow{\alpha}$.

Pref: $Q = \alpha_u.P_1$ or $Q = \underline{\alpha}_u.P_1$. In both cases $P = \mathsf{unmark}(Q) = \alpha_u.P_1$

1. $\alpha \notin A$ implies $\mathsf{LC}(Q, A) = \mathsf{LC}(P, A) = \{u\}$; otherwise $\mathsf{LC}(Q, A) = \mathsf{LC}(P, A) = \emptyset$.
2. $Q \xrightarrow{\alpha} P_1$, $P \xrightarrow{\alpha} P_1$ and, by Lemma 8.6-2, $P_1 \in \mathsf{L}(\tilde{\mathbb{P}}_1)$ implies $P_1 = \mathsf{unmark}(P_1)$.

Sum: $Q = P_1 +_u P_2$ or $Q = P_1 \underline{+}_u P_2$. In both cases $P = \mathsf{unmark}(Q) = P_1 +_u P_2$.

1. $\mathsf{LC}(P_1, A) \cup \mathsf{LC}(P_2, A) \neq \emptyset$ implies $\mathsf{LC}(Q, A) = \mathsf{LC}(P, A) = \{u\}$; otherwise $\mathsf{LC}(Q, A) = \mathsf{LC}(P, A) = \emptyset$.
2. $Q \xrightarrow{\alpha} Q'$ if either $P_1 \xrightarrow{\alpha} Q'$ or $P_2 \xrightarrow{\alpha} Q'$. In both cases $P_i \in \mathsf{L}(\tilde{\mathbb{P}}_1)$ and Proposition 8.1 implies $Q' \in \mathsf{L}(\tilde{\mathbb{P}}_1)$. As in the Pref-case $Q' = \mathsf{unmark}(Q')$.

Par: $Q = Q_1 \|^u_B Q_2$. In this case $P = \mathsf{unmark}(Q)$ implies $P = P_1 \|^u_B P_2$ where $P_i = \mathsf{unmark}(Q_i)$ for $i = 1, 2$.

1. Let $A_1 = (\mathcal{A}(P_1) \backslash \mathcal{A}(P_2)) \cap B = (\mathcal{A}(Q_1) \backslash \mathcal{A}(Q_2)) \cap B$ and $A_2 = (\mathcal{A}(P_2) \backslash \mathcal{A}(P_1)) \cap B = (\mathcal{A}(Q_2) \backslash \mathcal{A}(Q_1)) \cap B$ (see Lemma 8.6-4). By induction hypothesis $\mathsf{LC}(Q, A) = \mathsf{LC}(Q_1, A \cup A_1) \cup \mathsf{LC}(Q_2, A \cup A_2) = \mathsf{LC}(P_1, A \cup A_1) \cup \mathsf{LC}(P_2, A \cup A_2) = \mathsf{LC}(P, A)$.
2. Assume that $Q \xrightarrow{\alpha} Q'$ and consider the following cases:
   - $\alpha \notin B$, $Q_1 \xrightarrow{\alpha} Q_1'$ and $Q' = \mathsf{clean}(Q_1' \|^u_B Q_2)$. By induction hypothesis $P_1 \xrightarrow{\alpha} P_1'$ and $P_1' = \mathsf{unmark}(Q_1')$. Moreover, $P_1 \in \mathsf{L}(\tilde{\mathbb{P}}_1)$ and $P_1 \xrightarrow{\alpha} P_1'$ imply (by Proposition 8.1-2) $P_1' \in \mathsf{L}(\tilde{\mathbb{P}}_1)$ and, hence, $P_1' \|^u_B P_2 \in \mathsf{L}(\tilde{\mathbb{P}}_1)$. Thus $P \xrightarrow{\alpha} \mathsf{clean}(P_1' \|^u_B P_2) = P_1' \|^u_B P_2 = P'$ and $P' = \mathsf{unmark}(Q_1') \|^u_B \mathsf{unmark}(Q_2) = \mathsf{unmark}(Q_1' \|^u_B Q_2) = \mathsf{unmark}(Q')$ by Lemmas 8.6-2 and 8.6-3.
   - $\alpha \notin B$, $Q_2 \xrightarrow{\alpha} Q_2'$ and $Q' = \mathsf{clean}(Q_1 \|^u_B Q_2')$. Similar to the previous case.
   - $\alpha \in B$, $Q_i \xrightarrow{\alpha} Q_i'$ for $i = 1, 2$, and $Q' = \mathsf{clean}(Q_1' \|^u_B Q_2')$. By induction hypothesis $P_i \xrightarrow{\alpha} P_i'$ and $P_i' = \mathsf{unmark}(Q_i')$ for $i = 1, 2$. Moreover, as in the previous cases, $P_1', P_2' \in \mathsf{L}(\tilde{\mathbb{P}}_1)$ and, hence, $P_1' \|^u_B P_2' \in \mathsf{L}(\tilde{\mathbb{P}}_1)$. Thus $P \xrightarrow{\alpha} \mathsf{clean}(P_1' \|^u_B P_2') = P_1' \|^u_B P_2' = P'$ and $P' = P_1' \|^u_B P_2' = \mathsf{unmark}(Q_1') \|^u_B \mathsf{unmark}(Q_2') = \mathsf{unmark}(Q_1' \|^u_B Q_2') = \mathsf{unmark}(Q')$ again by Lemmas 8.6-2 and 8.6-3.

Rel: $Q = Q_1[\Phi]$. In this case $P = \mathsf{unmark}(Q)$ implies $P = P_1[\Phi]$ where $P_1 = \mathsf{unmark}(Q_1)$.

1. By induction hypothesis we have that $\mathsf{LC}(Q, A) = \mathsf{LC}(Q_1, \Phi^{-1}(A)) = \mathsf{LC}(P_1, \Phi^{-1}(A)) = \mathsf{LC}(P, A)$.
2. In this case $Q \xrightarrow{\alpha} Q'$ only if there exists $\beta \in \Phi^{-1}(\alpha)$ such that $Q_1 \xrightarrow{\beta} Q_1'$ and $Q' = Q_1'[\Phi_u]$. By induction hypothesis $P_1 \xrightarrow{\beta} P_1'$ and $P_1' = \mathsf{unmark}(Q_1')$. Thus $P \xrightarrow{\alpha} P_1'[\Phi_u] = P'$ and $P' = P_1'[\Phi_u] = \mathsf{unmark}(Q_1')[\Phi_u] = \mathsf{unmark}(Q_1'[\Phi_u]) = \mathsf{unmark}(Q')$.

Rec: $Q = \mathsf{rec}\, x.Q_1$. In this case $P = \mathsf{unmark}(Q)$ implies $P = \mathsf{rec}\, x.P_1$ where $P_1 = \mathsf{unmark}(Q_1)$.

1. By induction hypothesis we have that $\mathsf{LC}(Q, A) = \mathsf{LC}(Q_1, A) = \mathsf{LC}(P_1, A) = \mathsf{LC}(P, A)$.
2. Let $S = Q_1\{\!|\mathsf{rec}\, x_u.\mathsf{unmark}(Q_1)/x|\!\} = Q_1\{\!|\mathsf{rec}\, x_u.P_1/x|\!\}$ and $R = \mathsf{unmark}(S)$. Then x guarded in Q_1 implies $R = \mathsf{unmark}(Q_1\{\!|\mathsf{rec}\, x_u.P_1/x|\!\}) = \mathsf{unmark}(Q_1)\{\!|\mathsf{rec}\, x_u.P_1/x|\!\} = P_1\{\!|\mathsf{rec}\, x_u.P_1/x|\!\}$ (see Lemma 8.9-3).
   Now assume that $Q \xrightarrow{\alpha} Q'$ and, by operational rules, $S \xrightarrow{\alpha} Q'$. By induction hypothesis we have that $R \xrightarrow{\alpha} P'$ and $P' = \mathsf{unmark}(Q')$. Finally $R \xrightarrow{\alpha} P'$ implies, again by operational rules, $P \xrightarrow{\alpha} P'$.

□

Proposition 4.1 Let $P_0 \in \mathsf{L}(\mathbb{P}_1)$, $Q_0 = \mathsf{urgent}(P_0)$ and $v = \alpha_1 \ldots \alpha_n \in \mathbb{A}_\tau^*$. Then:

1. $P_0 \xrightarrow{v}_{\mathsf{LC}(P_0)} P_n$ implies $Q_0 \xrightarrow{v} P_n$;
2. $Q_0 \xrightarrow{v} Q_n$ and $\mathsf{UC}(Q_n) = \emptyset$ implies $P_0 \xrightarrow{v}_{\mathsf{LC}(P_0)} Q_n$.

Proof:

1. Assume that $P_0 \xrightarrow{v}_{\mathsf{LC}(P_0)} P_n$. Then, by definition of an LC-step, $P_0 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} P_n$ and $\mathsf{LC}(P_0) \cap \ldots \cap \mathsf{LC}(P_n) = \emptyset$. Now:
   - By Corollary 9.7-2 $P_0 = \mathsf{unmark}(Q_0)$. Thus Lemmas 10.1-3 and 10.1-1 imply that $Q_0 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} Q_n$, $\mathsf{LC}(Q_0) \cap \ldots \cap \mathsf{LC}(Q_n) = \emptyset$ and $P_n = \mathsf{unmark}(Q_n)$. Moreover, since $\mathsf{UC}(S) \subseteq \mathsf{LC}(S)$ for a generic S (see Lemma 9.5), we also have $\mathsf{UC}(Q_0) \cap \ldots \cap \mathsf{UC}(Q_n) = \emptyset$.
   - $Q_0 = \mathsf{urgent}(P_0)$ and $Q_0 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} Q_n$ imply, by Proposition 9.1-1, $\mathsf{UC}(Q_n) \subseteq \mathsf{UC}(Q_{n-1}) \subseteq \ldots \subseteq \mathsf{UC}(Q_0)$ and, hence, $\mathsf{UC}(Q_n) = \mathsf{UC}(Q_0) \cap \ldots \cap \mathsf{UC}(Q_n) = \emptyset$. Finally, Proposition 9.1-1 and $\mathsf{UC}(Q_n) = \emptyset$ imply $Q_n \in \mathsf{L}(\tilde{\mathbb{P}}_1)$ and $P_n = \mathsf{unmark}(Q_n) = Q_n$ (by Proposition 8.8-2).

   We can conclude that $Q_0 \xrightarrow{v} P_n$.
2. Assume that $Q_0 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} Q_n$ and $\mathsf{UC}(Q_n) = \emptyset$. Then:
   - by Propositions 9.1-1 and 9.1-2, $\mathsf{UC}(Q_n) = \emptyset$ implies $Q_n \in \mathsf{L}(\tilde{\mathbb{P}}_1)$ and $\mathsf{LC}(Q_0) \cap \ldots \cap \mathsf{LC}(Q_n) = (\mathsf{LC}(Q_0) \cap \ldots \cap \mathsf{LC}(Q_n)) \backslash \mathsf{UC}(Q_0) = \emptyset$.
   - as in the previous item, $P_0 = \mathsf{unmark}(Q_0)$ and by Lemmas 10.1-2 and 10.1-1, we also have that $P_0 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} P_n$, $P_n = \mathsf{unmark}(Q_n)$ and $\mathsf{LC}(P_0) \cap \ldots \cap \mathsf{LC}(P_n) = \emptyset$.

   Then $P_0 \xrightarrow{v}_{\mathsf{LC}(P_0)} P_n$ and $P_n = \mathsf{unmark}(Q_n) = Q_n$ (again by Proposition 8.8-2).

□

Theorem 4.4 Let $P_0 \in \mathsf{L}(\mathbb{P}_1)$ and $v_0, v_1, v_2 \ldots \in (\mathbb{A}_\tau)^*$. Then:

1. For any finite fair-step sequence from P_0

$$P_0 \xrightarrow{v_0}_{\mathsf{LC}(P_0)} P_1 \xrightarrow{v_1}_{\mathsf{LC}(P_1)} P_2 \ldots P_{n-1} \xrightarrow{v_{n-1}}_{\mathsf{LC}(P_{n-1})} P_n$$

   there exists a timed execution sequence

$$P_0 \xrightarrow{1} Q_0 \xrightarrow{v_0} P_1 \xrightarrow{1} Q_1 \xrightarrow{v_1} P_2 \ldots P_{n-1} \xrightarrow{1} Q_{n-1} \xrightarrow{v_{n-1}} P_n \xrightarrow{1} Q_n \xrightarrow{1} Q_n \ldots$$

2. For any timed execution sequence from P_0

$$P_0 \xrightarrow{1} Q_0 \xrightarrow{v_0} P_1 \xrightarrow{1} Q_1 \xrightarrow{v_1} P_2 \ldots P_{n-1} \xrightarrow{1} Q_{n-1} \xrightarrow{v_{n-1}} P_n \xrightarrow{1} Q_n \xrightarrow{1} Q_n \ldots$$

   the following is a finite fair-step sequence:

$$P_0 \xrightarrow{v_0}_{\mathsf{LC}(P_0)} P_1 \xrightarrow{v_1}_{\mathsf{LC}(P_1)} P_2 \ldots P_{n-1} \xrightarrow{v_{n-1}}_{\mathsf{LC}(P_{n-1})} P_n$$

Proof:

1. Assume $P_0 \xrightarrow{v_0}_{\mathsf{LC}(P_0)} P_1 \xrightarrow{v_1}_{\mathsf{LC}(P_1)} P_2 \ldots P_{n-1} \xrightarrow{v_{n-1}}_{\mathsf{LC}(P_{n-1})} P_n$ and $P_n \overset{\alpha}{\not\rightarrow}$ for any $\alpha \in \mathbb{A}_\tau$. Let $i \in [0, n-1]$. Then $P_i \xrightarrow{v_i}_{\mathsf{LC}(P_i)} P_{i+1}$ implies for $Q_i = \mathsf{urgent}(P_i)$ that $Q_i \xrightarrow{v_i} P_{i+1}$ by Proposition 4.1-2; thus, $P_i \xrightarrow{1} Q_i \xrightarrow{v_i} P_{i+1}$ by Definition 2.5. Moreover $P_n \overset{\alpha}{\not\rightarrow}$ for any $\alpha \in \mathbb{A}_\tau$ implies, by Proposition 8.12, $\mathcal{A}(P_n) = \emptyset$. Finally $\mathcal{A}(P_n) = \emptyset$ and Proposition 8.8-3 imply for $Q_n = \mathsf{urgent}(P_n)$ that $Q_n = P_n \xrightarrow{1} Q_n$.

2. Assume that $P_0 \xrightarrow{1} Q_0 \xrightarrow{v_0} P_1 \xrightarrow{1} Q_1 \xrightarrow{v_1} P_2 \ldots P_{n-1} \xrightarrow{1} Q_{n-1} \xrightarrow{v_{n-1}} P_n \xrightarrow{1} Q_n \xrightarrow{1} Q_n \ldots$
   Let $i \in [0, n-1]$. Then $Q_i \xrightarrow{v_i} P_{i+1}$ implies $P_i \xrightarrow{v_i}_{\mathsf{LC}(P_i)} P_{i+1}$ by Proposition 4.2-1. Moreover, $P_n \xrightarrow{1} Q_n \xrightarrow{1} Q_n \ldots$ implies $P_n = Q_n$ and $\mathcal{A}(P_n) = \emptyset$ by Proposition 8.11. Thus, by Proposition 8.12, we have that that $P_n \overset{\alpha}{\not\rightarrow}$ for any $\alpha \in \mathbb{A}_\tau$.

□