Asynchronous Observations of Processes*

Michele Boreale[1] Rocco De Nicola[2] Rosario Pugliese[2]

[1]Dipartimento di Scienze dell'Informazione, Università di Roma "La Sapienza"
[2]Dipartimento di Sistemi e Informatica, Università di Firenze

Abstract. We study may and must testing–based preorders in an asynchronous setting. In particular, we provide some full abstraction theorems that offer alternative characterizations of these preorders in terms of context closure w.r.t. basic observables and in terms of traces and acceptance sets. These characterizations throw light on the asymmetry between input and output actions in asynchronous interactions and on the difference between synchrony and asynchrony.

1 Introduction

Distributed systems can seldom rely on a global clock, and little assumptions can be made about their relative speed; as a consequence, it is natural to adopt for them an *asynchronous* communication mechanism. This calls for non–blocking sending primitives that do not oblige producers and consumers to synchronize when exchanging messages, but allow the sender of a message to continue with its task while the message travels to destination. Therefore, for describing distributed systems, a model based on a paradigm that imposes a neat distinction between input and output primitives, in the style of [1] and [17], appears to be a natural choice. In spite of these considerations, the most studied concurrency models in the process algebra community (e.g. [18, 3, 14, 20]) are based on *synchronous* communications and model process interaction as the execution of simultaneous "complementary actions".

Only recently, variants of process algebras based on asynchronous communications have been studied. Two main approaches have been followed to this purpose. They differ in the way (non–blocking) output actions are modelled. These actions are rendered either as *state transformers* or as *processes* themselves. The asynchronous variants of ACP [9] and CSP [16] follow the first approach and introduce explicit buffers in correspondence of output channels. This makes outputs non–blocking and immediately executable; their executions make messages available for consumption. The asynchronous variants of π-calculus [15, 6, 12, 2] and CCS [21, 11, 8] follow the second approach and model outputs by creating new concurrent processes. This amounts to modelling an output prefix $\overline{a}.P$ as a parallel composition $\overline{a} \mid P$.

* Work partially supported by EEC: HCM project EXPRESS, and by CNR: project "Specifica ad alto livello e verifica formale di sistemi digitali". The third author has been supported by a scholarship from CNR — Comitato Scienza e Tecnologie dell'Informazione.

The problem of specifying the abstract behaviour of asynchronous processes, i.e. of defining "good" observational semantics, has not yet been investigated in depth. Only few observational semantics have been considered. The maximal congruence induced by completed trace equivalence has been studied in [9] for asynchronous ACP. Bisimulation [18] for asynchronous π-calculus has been investigated in [15, 12, 2].

A natural alternative is represented by the *testing* framework of [10, 13]. Testing offers a uniform mechanism to define sensible behavioural equivalences on different process algebras, as it relies on little more than a notion of reduction relation ($\stackrel{\tau}{\longrightarrow}$). Moreover, testing has the advantage of identifying only those processes that cannot be differentiated by running observers in parallel with them. No new operator is introduced, as both the parallel composition operator and the observers are taken from the process description language under investigation. The testing approach has been partially followed in [22], where *synchronous* processes and observers are connected via input/output queues. This permits asynchronously testing synchronous processes.

In this paper we investigate the testing theory for a variety of asynchronous process algebras. For the sake of simplicity, the basic theory will be developed for an asynchronous version of CCS [18] (ACCS); we will then see how the obtained results can be extended with little effort to an asynchronous variant of π-calculus and to an asynchronous version of CCS with non–injective relabelling. The latter leads to a significantly different theory.

We shall study both the *may* and the *must* testing preorders. While natural, these preorders rely on a universal quantification over the set of all observers that makes reasoning about processes extremely difficult. This calls for alternative, observers–independent characterizations that permit a full appreciation of the impact of an asynchronous semantics over the considered languages. For each preorder, we will offer two characterizations: one in terms of the traces/acceptances of processes, the other in terms of the context–closure w.r.t. some *basic observables*, in the same spirit as [5].

As far as basic observables are concerned, we will see that, differently from the synchronous case, the only important actions are the output ones. In particular, for capturing the may preorder, we will need, as basic observables, tests about the possibility of processes to perform specific output actions. For capturing the must preorder, we will need, as basic observables, tests about the guarantee that processes offer of performing specific output actions.

The other alternative characterizations for the may preorder will be based on sequences of visible actions (*traces*), while that for the must preorder will rely on pairs ⟨trace, acceptance set⟩ in the same spirit as [13] and [7]. However, the usual trace containment for may is not adequate anymore, and the notion of acceptance–set for must is more complicate. We have for both may and must preorders equalities like $a.\overline{a} = \mathbf{0}$. The underlying reason is that, since no behaviour can causally depend upon outputs, observers cannot fully determine the occurrence of process *input* actions. As a consequence, both for may and for must, the set of traces will have to be factored via the preorder induced by the

three laws below, whose intuition is that whenever a trace s performed by some process is "acceptable" for the environment, then any $s' \preceq s$ is acceptable as well:

- (*deletion*) $\epsilon \preceq a$: process inputs cannot be forced;
- (*postponement*) $sa \preceq as$: observations of process inputs can be delayed;
- (*annihilation*) $\epsilon \preceq a\overline{a}$: buffers are not observable.

The extension of the alternative characterizations to the π-calculus is relatively straightforward and vindicates the stability of the approach. The extension to a process description language with non–injective relabelling shows that this operator enables external observers to get more precise information about inputs of asynchronous systems.

The rest of the paper is organized as follows. Section 2 introduces Asynchronous CCS and the testing preorders. Section 3 presents the alternative characterizations based on traces and acceptance–sets, while the next section presents those based on basic observables. The extensions to π-calculus and to CCS with general relabelling are sketched in Section 5. Some concluding remarks are reported in Section 6. Due to space limitations, many proofs will be omitted.

2 Asynchronous CCS

In this section we present syntax, and operational and testing semantics of asynchronous CCS (ACCS, for short). It differs from standard CCS because only guarded choices are used and output guards are not allowed. The absence of output guards "forces" the asynchrony; it is not possible to have processes that causally depends on output actions.

2.1 Syntax

We let $\mathcal{N}$, ranged over by $a, b, \ldots$, be an infinite set of *names* and $\overline{\mathcal{N}} = \{\overline{a} \mid a \in \mathcal{N}\}$, ranged over by $\overline{a}, \overline{b}, \ldots$, be the set of *co–names*. $\mathcal{N}$ and $\overline{\mathcal{N}}$ are disjoint and are in bijection via the *complementation* function $(\bar{\cdot})$; we define: $\overline{(\overline{a})} = a$. We let $\mathcal{L} = \mathcal{N} \cup \overline{\mathcal{N}}$ be the set of *visible actions*, and let $l, l', \ldots$ range over it. We let $\mathcal{L}_\tau = \mathcal{L} \cup \{\tau\}$ for a distinct action τ, be the set of all *actions* or *labels*, ranged over by μ. We shall use $A, B, L, \ldots$, to range over subsets of $\mathcal{L}$, M to range over multisets of $\mathcal{L}$ and s to range over $\mathcal{L}^*$. We define $\overline{L} = \{\overline{l} \mid l \in L\}$ and similarly for M and s. We let $\mathcal{X}$, ranged over by $X, Y, \ldots$, be a countable set of *process variables*.

Definition 1. The set of *ACCS terms* is generated by the grammar:

$$E ::= \overline{a} \mid \textstyle\sum_{i \in I} g_i.E_i \mid E_1 \mid E_2 \mid E\backslash L \mid E\{f\} \mid X \mid recX.E$$

where $g_i \in \mathcal{N} \cup \{\tau\}$, I is finite and $f : \mathcal{N} \to \mathcal{N}$, called *relabelling function*, is injective and such that $\{l \mid f(l) \neq l\}$ is finite. We extend f to $\mathcal{L}$ by letting $\forall \overline{a} \in \overline{\mathcal{N}} : f(\overline{a}) = \overline{f(a)}$. We let $\mathcal{P}$, ranged over by P, Q, etc., denote the set of *closed* and *guarded* terms or *processes* (i.e. those terms where every occurrence of any agent variable X lies within the scope of some $recX.\_$ and $\sum$ operators).

Notation. In the sequel, $\sum_{i\in\{1,2\}} g_i.E_i$ will be abbreviated as $g_1.E_1 + g_2.E_2$, $\sum_{i\in\emptyset} g_i.E_i$ will be abbreviated as $\mathbf{0}$; we will also write g for $g.\mathbf{0}$. $\Pi_{i\in I}E_i$ represents the parallel composition of the terms E_i. We write $\_\{l'_1/l_1,\ldots,l'_n/l_n\}$ for the relabelling operator $\_\{f\}$ where $f(l) = l'_i$ if $l = l_i$, $i \in \{1,\ldots,n\}$, and $f(l) = l$ otherwise. As usual, we write $E[F/X]$ for the term obtained by replacing each occurrence of X in E by F (with possibly renaming of bound process variables).

Throughout the paper, we will use the *structural congruence* relation over ACCS processes, $\equiv$, as defined in, e.g., [19] (the unique change with respect to [19] is the addition of some obvious distribution laws for injective relabelling).

2.2 Operational Semantics

The labelled transition system $(\mathcal{P}, \mathcal{L}_\tau, \xrightarrow{\mu})$, which characterizes the operational semantics of the language, is given by the rules in Figure 1.

$$\textbf{AR1}\ \sum_{i\in I} g_i.P_i \xrightarrow{g_j} P_j \quad j\in I \qquad \textbf{AR2}\ \overline{a} \xrightarrow{\overline{a}} \mathbf{0}$$

$$\textbf{AR3}\ \frac{P \xrightarrow{\mu} P'}{P\{f\} \xrightarrow{f(\mu)} P'\{f\}} \qquad \textbf{AR4}\ \frac{P \xrightarrow{\mu} P'}{P\backslash L \xrightarrow{\mu} P'\backslash L}\ \text{if } \mu \notin L\cup\overline{L}$$

$$\textbf{AR5}\ \frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q} \qquad \textbf{AR6}\ \frac{P[recX.P/X] \xrightarrow{\mu} P'}{recX.P \xrightarrow{\mu} P'}$$

$$\textbf{AR7}\ \frac{P \xrightarrow{l} P',\ Q \xrightarrow{\overline{l}} Q}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

Fig. 1. Operational semantics of ACCS (symmetric of rule AR5 omitted)

As usual, we use $\Longrightarrow$ or $\overset{\epsilon}{\Longrightarrow}$ to denote the reflexive and transitive closure of $\xrightarrow{\tau}$ and use $\overset{s}{\Longrightarrow}$ (resp. $\xrightarrow{s}$) for $\Longrightarrow \xrightarrow{l} \overset{s'}{\Longrightarrow}$ (resp. $\xrightarrow{l} \xrightarrow{s'}$) when $s = ls'$. Moreover, we write $P \overset{s}{\Longrightarrow}$ for $\exists P' : P \overset{s}{\Longrightarrow} P'$ ($P \xrightarrow{s}$ and $P \xrightarrow{\tau}$ will be used similarly). We will call *sort* of P the set $sort(P) = \{l \in \mathcal{L} \mid \exists s \in \mathcal{L}^* : P \overset{sl}{\Longrightarrow}\}$, *input* (resp. *output*) *successors* of P the set $In(P) = \{l \in \mathcal{N} \mid P \overset{l}{\Longrightarrow}\}$ ($Out(P) = \{l \in \overline{\mathcal{N}} \mid P \overset{l}{\Longrightarrow}\}$), *successors* of P the set $S(P) = In(P) \cup Out(P)$ and *language* generated by P the set $L(P) = \{s \in \mathcal{L}^* \mid P \overset{s}{\Longrightarrow}\}$. We say that a process P is *stable* if $P \not\xrightarrow{\tau}$.

From now onward, we adopt the following convention: an action declared *fresh* in a statement is assumed different from any other name and co–name mentioned in the statement. Note that, since for all relabelling operators f we have that $\{l \mid f(l) \neq l\}$ is finite, every ACCS process has a finite sort.

The following lemma implies that behaviours do not causally depend on the execution of output actions.

Lemma 2. For any process P and $\overline{a} \in \overline{\mathcal{N}}$, $P \xrightarrow{\overline{a}} Q$ implies $P \equiv Q \mid \overline{a}$.

2.3 Testing Semantics

We are now ready to instantiate the general framework of testing equivalences [10, 13] on ACCS.

Definition 3. *Observers* are ACCS processes that can also perform a distinct *success action* ω. $\mathcal{O}$ denotes the set of all the ACCS observers. A *computation* from a process P and an observer O is sequence of transitions

$$P \mid O = P_0 \mid O_0 \xrightarrow{\tau} P_1 \mid O_1 \xrightarrow{\tau} P_2 \mid O_2 \cdots P_k \mid O_k \xrightarrow{\tau} \cdots$$

which is either infinite or such that the last $P_k \mid O_k$ is stable. The computation is *successful* iff there exists some $n \geq 0$ such that $O_n \xrightarrow{\omega}$.

Definition 4. For every process P and observer O, we say

- $P \underline{may}\, O$ iff there exists a successful computation from $P \mid O$;
- $P \underline{must}\, O$ iff each computation from $P \mid O$ is successful.

Definition 5. We define the following preorders over processes:

- $P \sqsubseteq_{\approx} Q$ iff for every observer $O \in \mathcal{O}$, $P \underline{may}\, O$ implies $Q \underline{may}\, O$;
- $P \sqsubseteq^{m}_{\approx_M} Q$ iff for every observer $O \in \mathcal{O}$, $P \underline{must} O$ implies $Q \underline{must} O$.

We will use $\simeq$ to denote the equivalence obtained as the kernel of a preorder $\sqsubseteq_{\approx}$ (i.e. $\simeq = \sqsubseteq_{\approx} \cap \sqsubseteq_{\approx}^{-1}$).

3 Alternative Characterizations of Testing Semantics

The adaptation of the testing framework to an asynchronous setting discussed in the previous section is straightforward, but, like in the synchronous case, universal quantification on observers makes it difficult to work with the operational definitions of the two preorders. This calls for alternative characterizations that will make it easier to reason about processes. These characterizations will be given in terms of the traces and of the acceptance sets of processes.

3.1 A trace ordering

The following ordering over sequences of actions will be used for defining the alternative characterizations of the testing preorders.

Definition 6. Let $\preceq$ be the least preorder over $\mathcal{L}^*$ preserved under trace composition and satisfying the laws in Figure 2.

TO1 $\epsilon \preceq a$	TO2 $la \preceq al$	TO3 $\epsilon \preceq a\bar{a}$

Fig. 2. Trace Ordering Laws

The intuition behind the three laws in Figure 2 is that, whenever a process interacts with its environment by performing a sequence of actions s, an interaction is possible also if the process performs any $s' \preceq s$. To put it differently, if the environment offers $\overline{s}$, then it also offers any $\overline{s'}$ s.t. $s' \preceq s$.

More specifically, law T01 (*deletion*) says that process inputs cannot be forced to take place. For example, we have $\overline{b}c \preceq a\overline{b}c$: if the environment offers the sequence $\overline{a}b\overline{c}$, then it also offers $b\overline{c}$, as there can be no causal dependence of $b\overline{c}$ upon the output $\overline{a}$. Law T02 (*postponement*) says that observations of process inputs can be delayed. For example, we have that $\overline{b}ac \preceq a\overline{b}c$. Indeed, if the environment offers $\overline{a}b\overline{c}$ then it also offers $b\overline{a}c$. Finally, law T03 (*annihilation*) allows the environment to internally consume pairs of complementary actions, e.g. $\overline{b} \preceq a\overline{a}\overline{b}$. Indeed, if the environment offers $\overline{a}ab$ it can internally consume $\overline{a}$ and a and offer b.

Definition 7. Given $s \in \mathcal{L}^*$, we let $\{\!|\ s\ |\!\}$ denote the multiset of actions occurring in s, and $\{\!|\ s\ |\!\}_i$ (resp. $\{\!|\ s\ |\!\}_o$) denote the multiset of input (resp. output) actions in s. We let $s \ominus s'$ denote the multiset of input actions $(\{\!|\ s\ |\!\}_i \setminus \{\!|\ s'\ |\!\}_i) \setminus \overline{(\{\!|\ s\ |\!\}_o \setminus \{\!|\ s'\ |\!\}_o)}$, where $\setminus$ denotes difference between multisets.

Intuitively, if $s' \preceq s$ then $s \ominus s'$ is the multiset of input actions of s which have actually been deleted (law T01), and not annihilated (law T03), in s'. For instance, if $s = ab\overline{a}c$ and $s' = b$ then $s \ominus s' = \{\!|\ c\ |\!\}$.

Notation. If M is a multiset of actions, we will write $\Pi\, M$ for denoting $\Pi_{l \in M} l$, the parallel composition of all actions in M. We shall write "$P \stackrel{M}{\Longrightarrow} P'$" if $P \stackrel{s}{\Longrightarrow} P'$ for some sequentialization s of the actions in M. When M is a multiset of input actions, with a slight abuse of notation, we will sometimes denote by M also the trace obtained by arbitrarily ordering the elements of M (remember that we work modulo law T02). We shall write "$P \stackrel{s}{\Longrightarrow} P'$ l–free" if there exists a sequence of transitions $P = P_0 \stackrel{\mu_1}{\longrightarrow} P_1 \stackrel{\mu_2}{\longrightarrow} \cdots \stackrel{\mu_n}{\longrightarrow} P_n = P'$ such that $P_i \stackrel{l}{\not\longrightarrow}$ for $0 \leq i \leq n$ and s is obtained from $\mu_1 \cdots \mu_n$ by erasing the τ's.

The following is the crucial lemma for the preorder $\preceq$. Its proof relies on Lemma 2 and proceeds by induction on the number of times the laws in Figure 2 are used.

Lemma 8. Let P be a process and l an action and assume $s' \preceq s$. If $P \stackrel{\overline{s}}{\Longrightarrow} P'$ l–free then there exists P'' such that $P \stackrel{\overline{s'}}{\Longrightarrow} P''$ l–free and $P'' \equiv P' \mid \Pi\, \overline{s \ominus s'}$.[2]

3.2 The may case

By relying on the trace ordering $\preceq$, we can now define a new preorder that will be proved to be an alternative characterization of the may preorder $\sqsubseteq_m$.

[2] We remind the reader that $\equiv$ denotes structural congruence.

Definition 9. For processes P and Q, we write $P \ll_m Q$ iff whenever $P \stackrel{s}{\Longrightarrow}$ then there exists s' such that $s' \preceq s$ and $Q \stackrel{s'}{\Longrightarrow}$.

The difference with respect to the synchronous case (see, e.g., [10, 13]) is that we require a weaker condition than trace inclusion by taking advantage of a preorder over single traces. We define below a special class of observers.

Definition 10. Let $s \in \mathcal{L}^*$. The observers $t(s)$ are defined inductively as follows: $t(\epsilon) \stackrel{\text{def}}{=} \omega$, $t(\overline{a}s') \stackrel{\text{def}}{=} a.t(s')$ and $t(as') \stackrel{\text{def}}{=} \overline{a} \mid t(s')$.

The following property can be easily proved relying on Lemma 8.

Proposition 11. For every process P and $s \in \mathcal{L}^*$, $P \underline{may}\, t(s)$ iff there exists $s' \in L(P)$ such that $s' \preceq s$.

Theorem 12. For all processes P and Q, $P \sqsubseteq_{\lesssim_m} Q$ iff $P \ll_m Q$.

PROOF: 'Only if' part. Suppose that $P \sqsubseteq_{\lesssim_m} Q$ and that $s \in L(P)$. We must show that there exists $s' \in L(Q)$ such that $s' \preceq s$. The hypothesis $s \in L(P)$ implies that $P \underline{may}\, t(s)$. Since $P \sqsubseteq_{\lesssim_m} Q$, we infer that $Q \underline{may}\, t(s)$. The thesis follows from Proposition 11.

'If' part. Suppose that $P \ll_m Q$ and that $P \underline{may}\, O$ for an observer O. Then there exists a successful computation with an initial sequence of transitions $P \mid O \Longrightarrow P' \mid O'$ where $O' \stackrel{\omega}{\longrightarrow}$. This sequence of transitions may be unzipped into two sequences $P \stackrel{s}{\Longrightarrow} P'$ and $O \stackrel{\overline{s}}{\Longrightarrow} O'$. The hypothesis $P \ll_m Q$ implies that there exist s' and Q' such that $s' \preceq s$ and $Q \stackrel{s'}{\longrightarrow} Q'$. By Lemma 8, there exists an observer O'' such that $O \stackrel{\overline{s'}}{\Longrightarrow} O''$ and $O'' \equiv O' \mid \Pi\, \overline{s \ominus s'}$. Now, $O' \stackrel{\omega}{\longrightarrow}$ implies $O'' \stackrel{\omega}{\longrightarrow}$. Hence, the sequence of transitions $Q \mid O \Longrightarrow Q' \mid O''$ can be extended to a successful computation and the thesis is proved. □

By relying on the alternative characterization $\ll_m$ one can easily prove that $\sqsubseteq_{\lesssim_m}$ is a pre–congruence.

Examples. We show some examples of pairs of processes related by the preorder. All of the relationships can be proven by using the alternative characterization of the preorder $\ll_m$.

- Since $L(P) \subseteq L(Q)$ implies $P \sqsubseteq_{\lesssim_m} Q$, all of the relationships for the synchronous may preorder do hold in our setting.
- Since $\epsilon \in L(P)$ for each process P, from T01 and T03 in Figure 2, we get $a \simeq_m \mathbf{0}$ and $a.\overline{a} \simeq_m \mathbf{0}$. In particular, from $a \simeq_m \mathbf{0}$ we get $a \simeq_m b$ and $a.b \simeq_m b.a$ which imply that all processes containing only input actions are equivalent to $\mathbf{0}$.
- An interesting law is the $a.(\overline{a}|b) \simeq_m b$. More generally, we have $a.(\overline{a}|G) \sqsubseteq_{\lesssim_m} G$, where G is an input guarded summation $\sum_{i \in I} a_i.P_i$ (in fact, $a.\overline{a} \simeq_m \mathbf{0}$ is just a consequence of this law). Guardedness of G is essential: $\overline{b} \sqsubseteq_{\lesssim_m} a.(\overline{a} \mid \overline{b})$ does not hold (consider the observer $b.\omega$).

3.3 The must case

Definition 13.

- Let P be a process and $s \in \mathcal{L}^*$. We write $P \downarrow$, and say that P *converges*, if and only if there is no infinite sequence of internal transitions $P \xrightarrow{\tau} P_1 \xrightarrow{\tau} P_2 \xrightarrow{\tau} \cdots$ starting from P. We write $P \downarrow s$, and say that P *converges along* s if and only if whenever s' is a prefix of s and $P \stackrel{s'}{\Longrightarrow} P'$ then P' converges. We write $P \uparrow s$, and say that P *diverges along* s if it is not the case that $P \downarrow s$.
- Let P be a process and $s \in \mathcal{L}^*$. The set of processes $P \underline{after}\, s$ is defined by:
$$P \underline{after}\, s \stackrel{\text{def}}{=} \{(P' \mid \Pi\, \overline{s \ominus s'}) \,:\, s' \preceq s \text{ and } P \stackrel{s'}{\Longrightarrow} P'\}.$$
- Let X be a set of processes and $L \subseteq_{\text{fin}} \overline{\mathcal{N}}$. We write X $\underline{must}$ L if and only if for each $P \in X$ there exists $\overline{a} \in L$ s.t. $P \stackrel{\overline{a}}{\Longrightarrow}$.

In the sequel, given a set of traces $T \subseteq \mathcal{L}^*$, we will let $P \downarrow T$ stand for $P \downarrow s$ for each $s \in T$. Furthermore, we define $\widehat{s} \stackrel{\text{def}}{=} \{s' : s' \preceq s\}$.

Definition 14. We set $P \ll_M Q$ iff for each $s \in \mathcal{L}^*$ s.t. $P \downarrow \widehat{s}$ it holds that:

- $Q \downarrow \widehat{s}$, and
- for each $L \subseteq_{\text{fin}} \overline{\mathcal{N}}$: $(P \underline{after}\, s)$ $\underline{must}$ L implies $(Q \underline{after}\, s)$ $\underline{must}$ L.

Note that the above definition is formally similar to that for the synchronous case [10, 13]. The difference lies in the definition of the set $P \underline{after}\, s$: the latter can be seen as the set of possible states that P can reach after an interaction triggered by the environment offering $\overline{s}$. In an asynchronous setting, output actions can be freely performed by the environment, without any involvement of the process under consideration. In the definition of $P \underline{after}\, s$, these particular output actions represent the "difference" between the behaviour of the environment, $\overline{s}$, and the actual behaviour of the process, s', that is, $\Pi\, \overline{s \ominus s'}$.

Lemma 15. Let P be any process.

1. If P is stable then $In(P) \cap Out(P) = \emptyset$.
2. If P is stable then there exist P' and a unique multiset $M \subseteq_{\text{fin}} \overline{\mathcal{N}}$ s.t. $P \equiv P' \mid \Pi\, M$ and $Out(P') = \emptyset$.
3. If $P \stackrel{\overline{a}}{\Longrightarrow} P'$ then $S(P') \cup \{\overline{a}\} \subseteq S(P)$.

When P is stable, we will use $\mathsf{O}(P)$ to denote the unique multiset M implicitly defined by part 2 of the above lemma.

Theorem 16. If $P \ll_M Q$ then $P \sqsubseteq_{\approx_M} Q$.

PROOF: Let O be any observer and suppose that Q $\underline{must}$ O: we show that P $\underline{must}$ O as well. We make a case analysis on why Q $\underline{must}$ O. All cases can be easily reduced to the case of a finite unsuccessful computation, i.e. a sequence

of transitions $Q \mid O \Longrightarrow Q' \mid O'$ such that, for some s: $Q \stackrel{s}{\Longrightarrow} Q'$, $O \stackrel{\overline{s}}{\Longrightarrow} O'$ ω–free and $Q' \mid O'$ is stable. Furthermore, we suppose that $P \downarrow \widehat{s}$ and $Q \downarrow \widehat{s}$.

From the fact that $Q' \mid O'$ is stable and from Lemma 15(1), we deduce that:

$$(i)\ Out(Q') \cap \overline{In(O')} = \emptyset$$
$$(ii)\ In(Q') \cap \overline{Out(O')} = \emptyset$$
$$(iii)\ In(O') \cap \overline{Out(O')} = \emptyset\ .$$

We show now how to build an unsuccessful computation for $P \mid O$. Let us define the set of output actions $L \stackrel{\text{def}}{=} \overline{In(O')}$ and the multiset of input actions $M \stackrel{\text{def}}{=} \overline{O(O')}$ (note that, since O' is stable, this multiset is well defined in virtue of Lemma 15(2)). First, we show that

$$(Q\ \underline{after}\ sM)\ \underline{must}\ L\,. \tag{1}$$

Indeed, since $s \preceq sM$ and $Q \stackrel{s}{\Longrightarrow} Q'$, we have that $Q' \mid \Pi\ \overline{M} \in (Q\ \underline{after}\ sM)$; furthermore, we have that $Q' \mid \Pi\ \overline{M} \stackrel{\tau}{\not\longrightarrow}$ (from (ii) and $Q' \stackrel{\tau}{\not\longrightarrow}$), that $Out(Q') \cap L = \emptyset$ (from (i)) and that $\overline{M} \cap L = \emptyset$ (from (iii)). From these facts, it follows that $Out(Q' \mid \Pi\ \overline{M}) \cap L = \emptyset$. This proves (1).

Now, from (1) and definition of $\ll_M$ it follows that $(P\ \underline{after}\ sM)\ \underline{must}\ L$, which means that there are P' and $s' \preceq sM$ such that:

$$P \stackrel{s'}{\Longrightarrow} P' \text{ and } Out(P' \mid \Pi\ \overline{sM \ominus s'}) \cap L = \emptyset\ . \tag{2}$$

Now, since O' is stable, from Lemma 15(2), it follows that there exists O'' such that $O' \equiv O'' \mid \Pi\ \overline{M}$ and $Out(O'') = \emptyset$. Hence $O' \stackrel{\overline{M}}{\longrightarrow} \equiv O''$ and therefore $O \stackrel{\overline{sM}}{\Longrightarrow} \equiv O''$ ω–free. Since $s' \preceq sM$, from Lemma 8 it then follows that there is O_1 such that $O \stackrel{\overline{s'}}{\Longrightarrow} O_1 \equiv O'' \mid \Pi\ \overline{sM \ominus s'}$ ω–free. Combining these transitions of O with $P \stackrel{s'}{\Longrightarrow} P'$ in (2), we get:

$$P \mid O \Longrightarrow P' \mid O_1 \equiv P' \mid O'' \mid \Pi\ \overline{sM \ominus s'} \qquad \omega\text{–free.} \tag{3}$$

To prove that (3) leads to an unsuccessful computation, it suffices to show that $P' \mid O'' \mid \Pi\ \overline{sM \ominus s'} \stackrel{\omega}{\not\Longrightarrow}$. The latter is a consequence of the following three facts:

1. $Out(P' \mid \Pi\ \overline{sM \ominus s'}) \cap \overline{In(O'')} = \emptyset$. This derives from (2) and from $In(O'') \subseteq In(O') = \overline{L}$ (Lemma 15(3) applied to $O' \stackrel{\overline{M}}{\longrightarrow} \equiv O''$);
2. $Out(O'') = \emptyset$;
3. $O'' \stackrel{\omega}{\not\Longrightarrow}$ (Lemma 15(3) applied to $O' \stackrel{\overline{M}}{\longrightarrow} \equiv O''$). □

For proving the converse of the above theorem, we will use two families of observers: the first can be used to test for convergence along sequences of a given set $\widehat{s}$, and the second to test that a given pair (s, L) is an "acceptance" pair.

Definition 17. Let $s \in \mathcal{L}^*$ and $L \subseteq_{\text{fin}} \overline{\mathcal{N}}$. The observers $c(s)$ and $a(s, L)$ are defined by induction on s as follows:

$$\begin{array}{lll} c(s): & c(\epsilon) = \tau.\omega & a(s,L): \quad a(\epsilon, L) = \sum_{\overline{a} \in L} a.\omega \\ & c(bs') = \overline{b} \mid c(s') & \quad\ a(bs', L) = \overline{b} \mid a(s', L) \\ & c(\overline{b}s') = \tau.\omega + b.c(s') & \quad\ a(\overline{b}s', L) = \tau.\omega + b.a(s', L)\,. \end{array}$$

Lemma 18. Let P be a process, $s \in \mathcal{L}^*$ and $L \subseteq_{\text{fin}} \overline{N}$. We have:
1. $P \underline{must}\, c(s)$ if and only if $P \downarrow \widehat{s}$.
2. Suppose that $P \downarrow \widehat{s}$. Then $P \underline{must}\, a(s, L)$ if and only if $(P \underline{after}\, s) \underline{must}\, L$.

PROOF: An easy application of Lemma 8. □

Theorem 19. $P \sqsubseteq_{\approx_M} Q$ implies $P \ll_M Q$.

PROOF: An easy consequence of Lemma 18. □

By relying on $\ll_M$, it is straightforward to show that $\sqsubseteq_{\approx_M}$ is a pre–congruence.

Examples. We give below some meaningful examples of processes that are related (or unrelated) according to the preorder. All the examples are checked relying on the alternative characterization provided by $\ll_M$. In the examples, we shall also refer to the asynchronous bisimilarity[3] of [2].

- The process $\mathbf{0}$ represents the top element for the family of terms built using only input actions: $a \sqsubseteq_{\approx_M} \mathbf{0}$, but $\mathbf{0} \not\sqsubseteq_{\approx_M} a$; thus $a{+}b \sqsubseteq_{\approx_M} a$, but $a \not\sqsubseteq_{\approx_M} a{+}b$.
- Input prefixes can be distributed over summation, i.e. $a.(b{+}c) \simeq_M a.b{+}a.c$. This is in sharp contrast with the asynchronous bisimilarity.
- Sequences of inputs can absorb their own prefixes, as in $a.b{+}a \simeq_M a.b$ This law was also present in [9], but is not valid for asynchronous bisimilarity.
- Like in [2], we have $a.\overline{a} \simeq_M \mathbf{0}$. This is an instance of the more general law $a.(\overline{a} \mid G){+}G \simeq_M G$, where G is any guarded summation $\sum_{i \in I} g_i.P_i$. Unlike [2], however, the law does not hold for infinite behaviours: $recX.(a.(\overline{a} \mid X)) \not\simeq_M \mathbf{0}$. This is due to the sensitivity of must to divergence: when put in parallel with $\overline{a}$, $recX.(a.(\overline{a} \mid X))$ diverges, while $\mathbf{0}$ does not.

As shown in the examples above, must equivalence and asynchronous bisimilarity are in general incomparable, due to the sensitivity of must to divergence. They are comparable if we consider only *strongly convergent* processes, i.e. those processes P such that $P \downarrow s$ for each s. The crux is given by the following characterization of $\approx$:

Proposition 20. $P \approx Q$ if and only if whenever $P \stackrel{s}{\Longrightarrow} P'$ then there is $s' \preceq s$ s.t. $Q \stackrel{s'}{\Longrightarrow} Q'$ and $P' \approx Q' \mid \Pi\, \overline{s \ominus s'}$, and vice–versa for Q and P.

Corollary 21. Let P and Q be strongly convergent processes. Then $P \approx Q$ implies $P \sqsubseteq_{\approx_M} Q$.

[3] We remind the reader that asynchronous bisimilarity is defined as the maximal equivalence relation $\approx$ s.t. whenever $P \approx Q$ and $P \stackrel{\mu}{\longrightarrow} P'$ then:
(a) if $\mu = \tau$ then there is Q' such that $Q \Longrightarrow Q'$ and $P' \approx Q'$,
(b) if $\mu = \overline{a}$ then there is Q' such that $Q \stackrel{\overline{a}}{\Longrightarrow} Q'$ and $P' \approx Q'$, and
(c) if $\mu = a$ then there is Q' such that either (i) $Q \stackrel{a}{\Longrightarrow} Q'$ and $P' \approx Q'$, or (ii) $Q \Longrightarrow Q'$ and $P' \approx Q' \mid \overline{a}$.

4 Basic Observables for Asynchronous Processes

Following [5], we introduce a characterization of the asynchronous may and must preorders in terms of the pre–congruence induced by *basic observables*. The difference with the synchronous case is that here only output actions are important.

Definition 22. A *context* is a term C with one free occurrence of a process variable, usually denoted by $\_$. We write $C[P]$ instead of $C[P/\_]$.

The context *closure* $\mathcal{R}^c$ of a given binary relation $\mathcal{R}$ over processes, is defined as: $P\,\mathcal{R}^c\,Q$ iff for each context C, $C[P]\,\mathcal{R}\,C[Q]$. $\mathcal{R}^c$ enjoys two important properties: (a) $(\mathcal{R}^c)^c = \mathcal{R}^c$, and (b) $\mathcal{R} \subseteq \mathcal{R}'$ implies $\mathcal{R}^c \subseteq \mathcal{R}'^c$. In the following, we will write $\not\mathcal{R}$ for the complement of $\mathcal{R}$.

4.1 The may case

Definition 23. Let P be a process and $\overline{a} \in \overline{\mathcal{N}}$. We define the following *observation predicate* over processes: $P\surd\overline{a}$ (P *offers* $\overline{a}$) iff $P \stackrel{\overline{a}}{\Longrightarrow}$.

The observation preorder induced by $\surd$ is defined as follows: $P \preceq_{\surd} Q$ iff for each $\overline{a} \in \overline{\mathcal{N}}$: $P\surd\overline{a}$ implies $Q\surd\overline{a}$.

Of course, the observation preorder is very coarse; a more refined relation can be obtained by closing it under all ACCS contexts. The *contextual preorder* of $\preceq_{\surd}$ is just its context closure $\preceq^c_{\surd}$; the latter is another characterization of $\sqsubseteq_{\approx_m}$.

Theorem 24. For all processes P and Q, $P \sqsubseteq_{\approx_m} Q$ iff $P \preceq^c_{\surd} Q$.

PROOF: We use the alternative characterization $\ll_m$ of $\sqsubseteq_{\approx_m}$.

'Only if' part. From the definition, it is easily seen that $\ll_m$ is contained in $\preceq_{\surd}$ (note that for each $\overline{a} \in \overline{\mathcal{N}}$, $s \preceq \overline{a}$ implies $s = \overline{a}$). From this fact, by closing under contexts and recalling that $\preceq^c_{\surd}$ is a pre–congruence the thesis follows.

'If' part. Here, we show that $\preceq^c_{\surd}$ is contained in $\ll_m$. From this fact and recalling that $\preceq^c_{\surd}$ is a pre–congruence the thesis will follow. Assume that $P \preceq^c_{\surd} Q$ and that $s \in L(P)$, for some $s \in \mathcal{L}^*$. We have to show that there exists $s' \in L(Q)$ such that $s' \preceq s$. Now, let $t'(s)$ be the process defined like the observer $t(s)$ in Definition 10, but with a fresh, standard action $\overline{c}$ in place of ω. The following fact, where R is any process where neither c nor $\overline{c}$ occur, is straightforward to prove by relying on Lemma 8: $(t'(s) \mid R)\surd\overline{c}$ iff there exists $s' \in L(R)$ such that $s' \preceq s$. The thesis is an immediate consequence of this fact. □

4.2 The must case

We introduce below the *guarantee* predicate, $P!l$; informally, this predicate checks whether P will always be able to offer a communication on l; however, differently from [5], we here only consider output actions.

Definition 25. Let P be a process and $\overline{a} \in \overline{\mathcal{N}}$. We write $P!\overline{a}$ (P *guarantees* $\overline{a}$) if and only if whenever $P \Longrightarrow P'$ then $P' \stackrel{\overline{a}}{\Longrightarrow}$.

The *observation preorder* induced by $\downarrow$ and $!$ is defined as: $P \,{}_{\downarrow}\preceq_{!}\, Q$ if and only if for each $\overline{a}$: ($P \downarrow$ and $P!\overline{a}$) implies ($Q \downarrow$ and $Q!\overline{a}$).

Theorem 26. $P \sqsubseteq_{\approx_M} Q$ if and only if $P \,{}_{\downarrow}\preceq^{c}_{!}\, Q$.

PROOF: We use the characterization of the must preorder in terms of $\ll_M$.

'If' part. First, note that $P!\overline{a}$ if and only if $(P \underline{after}\, \epsilon)$ $\underline{must}$ $\{\overline{a}\}$. Hence, by definition, $\ll_M$ is included in ${}_{\downarrow}\preceq_{!}$. The thesis then follows by closing under contexts and recalling that $\sqsubseteq_{\approx_M}$ is a pre–congruence.

'Only if' part. Fix any s and L and suppose that $P \downarrow \widehat{s}$ and $(P \underline{after}\, s)$ $\underline{must}$ L. We have to show that $Q \downarrow \widehat{s}$ and $(Q \underline{after}\, s)$ $\underline{must}$ L. Now, let $c'(s)$ and $a'(s, L)$ be the observers defined like in Definition 17, but with a fresh, standard action $\overline{c}$ in place of ω. The following two facts, where R is any process where neither c nor $\overline{c}$ occur, are straightforward to prove relying on Lemma 8:

- $R \downarrow \widehat{s}$ if and only if $R \mid c'(s) \downarrow$.
- Suppose that $R \downarrow \widehat{s}$. Then $R \mid a'(s, L) \downarrow$ and furthermore $(R \underline{after}\, s)$ $\underline{must}$ L if and only if $R \mid a'(s, L)!\overline{c}$.

Then $Q \downarrow \widehat{s}$ and $(Q \underline{after}\, s)$ $\underline{must}$ L follow from the definition of ${}_{\downarrow}\preceq^{c}_{!}$ and from the above two facts. □

5 Dealing with Richer Languages

In this section we discuss the extensions of our theory to the asynchronous variant of π-calculus [15, 6, 12, 2] and to a version of asynchronous CCS of Section 2 with possibly non–injective relabelling.

5.1 π-calculus

For the sake of simplicity, we confine ourselves to the may preorder. The must preorder requires a more complex notational machinery but also leads to results similar to those for ACCS.

A countable set $\mathcal{N}$ of *names* is ranged over by $a, b, \ldots$. Processes are ranged over by P, Q and R. The syntax of asynchronous π–calculus contains the operators for output action, input–guarded summation, restriction, parallel composition, matching and replication:

$$P ::= \overline{a}b \mid \sum_{i\in I} a_i(b).P_i \mid \nu a P \mid P_1 \mid P_2 \mid [a=b]P \mid !P.$$

Free names and *bound names* of a process P, written $\mathrm{fn}(P)$ and $\mathrm{bn}(P)$ respectively, arise as expected; the *names* of P, written $\mathrm{n}(P)$ are $\mathrm{fn}(P) \cup \mathrm{bn}(P)$. Due to lack of space, we omit the definition of operational semantics (see, e.g., [2]). Recall that transition labels (actions), ranged over by μ, can be of four forms: τ (interaction), ab (input), $\overline{a}b$ (output) or $\overline{a}(b)$ (bound output). Functions $\mathrm{bn}(\cdot)$, $\mathrm{fn}(\cdot)$ and $\mathrm{n}(\cdot)$ are extended to actions as expected: in particular, $\mathrm{bn}(\mu) = b$ if $\mu = \overline{a}(b)$ and $\mathrm{bn}(\mu) = \emptyset$ otherwise.

In the sequel, we will write $P \xrightarrow{a(b)} P'$ if $P \xrightarrow{ab} P'$ and $b \notin \mathrm{fn}(P)$. The new kind of action $a(b)$ is called *bound input*; we extend $\mathrm{bn}(\cdot)$ to bound inputs by letting $\mathrm{bn}(a(b)) = \{b\}$. Below, we shall use $\mathcal{L}_\pi$ to denote the set of all visible π-calculus actions, including bound inputs, and let θ range over it. Given a trace $s \in \mathcal{L}_\pi^*$, we say that s is *normal* if, whenever $s = s'.\theta.s''$ (the dot . stands for trace composition), for some s', θ and s'', then $\mathrm{bn}(\theta)$ does not occur in s' and $\mathrm{bn}(\theta)$ is different from any other bound name occurring in s' and s''. The set of normal traces over $\mathcal{L}_\pi$ is denoted by $\mathcal{T}$ and ranged over by s. From now on, we shall work with normal traces only. Functions $\mathrm{bn}(\cdot)$ and $\mathrm{fn}(\cdot)$ are extended to $\mathcal{T}$ as expected. A complementation function on $\mathcal{T}$ is defined by setting $\overline{a(b)} \stackrel{\mathrm{def}}{=} \overline{a}(b)$, $\overline{ab} \stackrel{\mathrm{def}}{=} \overline{a}b$, $\overline{\overline{a}b} \stackrel{\mathrm{def}}{=} ab$ and $\overline{\overline{a}(b)} \stackrel{\mathrm{def}}{=} a(b)$; please notice that $\overline{\overline{s}} = s$.

P1	$\epsilon \preceq \theta$	if θ is an input action
P2	$s.\theta \preceq \theta.s$	if θ is an input action and $\mathrm{bn}(\theta) \cap \mathrm{bn}(s) = \emptyset$
P3	$\epsilon \preceq \theta.\overline{a}b$	if $\theta = ab$ or $\theta = a(b)$
P4	$\overline{a}c.(s\{^c/_b\}) \preceq \overline{a}(b).s$	

Fig. 3. Rules for the preorder $\preceq$ over $\mathcal{T}$

The definition of $\ll_m$ remains formally unchanged, but the relation $\preceq$ is now the least preorder over $\mathcal{T}$ closed under composition and generated by the rules in Figure 3. Rules P1, P2, P3 are the natural extensions to asynchronous π-calculus of the rules for ACCS. Here, some extra attention has to be paid to bound names: in the environment, an output declaring a new name (bound output) cannot be postponed after those actions which use the new name (side condition of P2). For an example, consider actions $\overline{a}(b)$ and $b(c)$ of $\nu b\,(\overline{a}b \mid b(c).P)$. Rule P4 is specific to π-calculus; it is due to the impossibility for observers to fully discriminate between free and bound outputs. Informally, rule P4 states that if $\overline{a}(c).s$ is "acceptable" for an observer (i.e. leads to success), then $\overline{a}b.(s\{^b/_y\})$ would be acceptable as well. Rule P4 would not hold if we extended the language with the *mismatching* operator $[a \neq b]P$, considered e.g. in [4]. It is worthwhile to note that ruling out matching from the language would not change the discriminating power of observers. The effect of the test $[a = b]O$ can be simulated by the parallel composition $\overline{a} \mid b.O$.

5.2 ACCS with General Relabelling

A consequence of the presence of non–injective relabelling functions, is that observers and contexts become more discriminating. For instance, they lead to $a.\overline{a} \not\lesssim_M \mathbf{0}$ and $a.\overline{a} \not\lesssim_m \mathbf{0}$. These can be proved by considering the observer $(\overline{b} \mid a.\omega)\{a/b\}$. We also have $\mathbf{0} \not\lesssim_M a.\overline{a}$, that can be proved by considering the observer $(\overline{b} \mid (\tau.\omega + a))\{a/b\}$. Therefore, the general laws $a.(\overline{a} \mid G_1) \simeq_m G_1$, where $G_1 = \sum_{i \in I} a_i.P_i$, and $a.(\overline{a} \mid G_2) + G_2 \simeq_M G_2$, where $G_2 = \sum_{i \in I} g_i.P_i$, are not sound anymore. By means of general relabelling, observers are able to distinguish between the messages they emit and those emitted by the observed processes.

The trace preorder is now defined as the least preorder over $\mathcal{L}^*$ closed under trace composition and satisfying the laws T01 and T02 in Figure 2. Notice that if $s' \preceq s$ then $\{\!|\, s \,|\!\}_o = \{\!|\, s' \,|\!\}_o$, therefore now we have $s \ominus s' = \{\!|\, s \,|\!\}_i \setminus \{\!|\, s' \,|\!\}_i$. The definition of $\ll_m$ remains formally unchanged.

Let us now consider the must preorder. In the following we shall write $s' \sim s$ iff $s' \preceq s$ and $\{\!|\, s' \,|\!\} = \{\!|\, s \,|\!\}$, and for M finite multiset of $\mathcal{L}$ and $L \subseteq_{\text{fin}} \mathcal{L}$ we shall write $M \setminus L$ for the multiset $\{\!|\, l \in M \mid l \notin L \,|\!\}$. The alternative characterization of the $\sqsubseteq_M$ preorder is now the following.

Definition 27. We set $P \ll_M Q$ iff for each $s \in \mathcal{L}^*$ s.t. $P \downarrow \widehat{s}$ it holds that:

a) $Q \downarrow \widehat{s}$, and
b) for each $s' \in \widehat{s}$, for each $L \subseteq_{\text{fin}} \overline{\mathcal{N}}$:
$$(P \underline{after}\, s'(s \ominus s'))\, \underline{must}\, L \text{ implies } (Q \underline{after}\, s'(s \ominus s'))\, \underline{must}\, L,$$

where for any process R, $s \in \mathcal{L}^*$ and M multiset of $\mathcal{N}$, we define $R \underline{after}\, sM$ as

$$\{P' : R \stackrel{s'}{\Longrightarrow} P', s' \preceq sM, s' \sim s(s' \ominus s), In(P') \cap (M \setminus (s' \ominus s)) = \emptyset\}.$$

6 Conclusions

We have examined the impact of the testing framework as proposed in [10, 13] on asynchronous CCS. In particular, we have given three equivalent characterizations of asynchronous testing observational semantics. The first one is given in terms of observers and successful computations, the second relies on sets of traces and acceptances, the third one is defined in terms of basic observables and context closures. We have discussed generalizations of the results to asynchronous π-calculus and to ACCS with non–injective relabelling.

The above mentioned characterizations provide a good starting point for understanding asynchronous semantics and for relating testing semantics to other approaches. The picture would have been more complete with an equational characterization of our semantics; this will be the topic of a forthcoming paper.

Acknowledgments. Three anonymous referees provided valuable suggestions. We are grateful to the Dipartimento di Scienze dell'Informazione of Università di Roma "La Sapienza" and to Istituto di Elaborazione dell'Informazione in Pisa for making our collaboration possible.

References

1. G.Agha. *Actors: a model of concurrent computation in Distributed Systems.* Mit–Press, Boston, 1986.
2. R.M. Amadio, I. Castellani, D. Sangiorgi. On Bisimulations for the Asynchronous π–calculus. *CONCUR'96, LNCS* 1119, pp.147-162, Springer, 1996.
3. J. Bergstra, J.W. Klop. Process Algebra for Synchronous Communication. *Information and Control*, 60:109-137, 1984.
4. M. Boreale, R. De Nicola. Testing Equivalence for Mobile Systems. *Information and Computation*, 120: 279-303, 1995.
5. M. Boreale, R. De Nicola, R. Pugliese. Basic Observables for Processes. *ICALP'97, LNCS* 1256, pp.482-492, Springer, 1997.
6. G. Boudol. Asynchrony in the π–calculus (note). Rapport de Recherche 1702, INRIA Sophia–Antipolis, 1992.
7. S.D. Brookes, C.A.R. Hoare, A.W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(3):560-599, 1984.
8. N. Busi, R. Gorrieri, G-L. Zavattaro. A process algebraic view of Linda coordination primitives. Technical Report UBLCS-97-05, University of Bologna, 1997.
9. F.S. de Boer, J.W. Klop, C. Palamidessi. Asynchronous Communication in Process Algebra. *LICS'92*, IEEE Computer Society Press, pp. 137-147, 1992.
10. R. De Nicola, M.C.B. Hennessy. Testing Equivalence for Processes. *Theoretical Computers Science*, 34:83-133, 1984.
11. R. De Nicola, R. Pugliese. A Process Algebra based on Linda. *COORDINATION'96, LNCS* 1061, pp.160-178, Springer, 1996.
12. M. Hansen, H. Huttel, J. Kleist. Bisimulations for Asynchronous Mobile Processes. In Proc. of the Tblisi Symposium on Language, Logic, and Computation, 1995.
13. M.C.B. Hennessy. *Algebraic Theory of Processes.* The MIT Press, 1988.
14. C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall Int., 1985.
15. K. Honda, M. Tokoro. An Object Calculus for Asynchronous Communication. *ECOOP'91, LNCS* 512, pp.133-147, Springer, 1991.
16. H. Jifeng, M.B. Josephs, C.A.R. Hoare. A Theory of Synchrony and Asynchrony. *Proc. of the IFIP Working Conf. on Programming Concepts and Methods*, pp.446-465, 1990.
17. N.A. Lynch, M.R. Tuttle. Hierarchical correctness proofs for distributed algorithms. *In* 6^{th} *ACM Symposium on Principles of Distributed Computing*, pp.137–151, 1987.
18. R. Milner. *Communication and Concurrency.* Prentice Hall International, 1989.
19. R. Milner. The Polyadic π-calculus: A Tutorial. Technical Report, University of Edinburgh, 1991.
20. R. Milner, J. Parrow, D. Walker. A calculus of mobile processes, (Part I and II). *Information and Computation*, 100:1-77, 1992.
21. R. Pugliese. A Process Calculus with Asynchronous Communications. 5th Italian Conference on Theoretical Computer Science, (A. De Santis, ed.), pp.295-310, World Scientific, 1996.
22. J. Tretmans. A formal approach to conformance testing. Ph.D. Thesis, University of Twente, 1992.