# A Theory of "May" Testing for Asynchronous Languages

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.** Asynchronous communication mechanisms are usually a basic ingredient of distributed systems and protocols. For these systems, asynchronous may-based testing seems to be exactly what is needed to capture safety and certain security properties. We study may testing equivalence focusing on the asynchronous versions of CCS and $\pi$-calculus. We start from an operational testing preorder and provide finitary and fully abstract trace-based interpretations for it, together with complete inequational axiomatizations. The results throw light on the differences between synchronous and asynchronous systems and on the weaker testing power of asynchronous observations.

## 1 Introduction

Distributed systems often rely on asynchronous communication primitives for exchanging information. Many properties of these systems can be conveniently expressed and verified by means of behavioural equivalences. In particular, *may* testing [11] seems to be exactly what is needed for reasoning about safety properties. In this respect, an assumption of asynchrony can often play a crucial role.

As an example, consider a trivial communication protocol with two users $A$ and $B$ sharing a private channel $c$. The protocol requires that $A$ uses $c$ to send a bit of information $m$ to $B$, then $B$ receives two messages on channels $a$ and $b$, finally $B$ sends, on channel $d$, the message received on $a$. The ordering of the inputs on $a$ and $b$ depends on the message received on $c$. In $\pi$-calculus we can formulate this protocol as follows (the meaning of the various operators is the usual one; in particular, $(\nu\, c)$ stands for creation of a local channel $c$):

$$\begin{aligned} A &= \overline{c}m \\ B &= c(x).([x=0]a(y).b(z).\overline{d}y + [x=1]b(z).a(y).\overline{d}y) \\ S &= (\nu\, c)(A \mid B) \end{aligned}$$

*Secrecy*, i.e. the ability to keep a datum secret, is an important property which one might want to check of this protocol: externally, it should not be possible to guess message $m$ from the behaviour of the whole system $S$. Following [2], this property can be formalized by requiring that the behaviour of the protocol should not depend on the bit that $A$ sends to $B$: in other words, processes $S[0/m]$ and $S[1/m]$ should be equivalent. The intended equivalence is here the

W. Thomas (Ed.): FOSSACS'99, LNCS 1578, pp. 165-179, 1999.

one induced by may testing: a process may pass a 'test' performed by an external observer if and only if the other process may. If one interprets 'passing a test' as 'revealing a piece of information', then equivalent processes *may* reveal externally the same information. Now, it is easy to see that an observer could tell $S[0/m]$ and $S[1/m]$ apart via *synchronous* communication on $a$ and $b$ (*traffic analysis*). However, $S[0/m]$ and $S[1/m]$ *are* equivalent in a truly asynchronous scenario, in which no ordering on the arrival of outgoing messages is guaranteed.

It is therefore important to have a full understanding of may-semantics in an asynchronous setting. We shall consider asynchronous variants of CCS and $\pi$-calculus: in these models, the communication medium can be understood as a *bag* of output actions (messages), waiting to be consumed by corresponding input actions. This is reminiscent of the Linda approach [13]. In [7], we have provided an observers-independent characterization of the asynchronous testing preorders. Here, we use this characterization as a starting point for defining a "finitary" trace-based model and a complete axiomatization for the may testing preorder.

When modelling asynchronous processes, the main source of complications is the non-blocking nature of output primitives. It is demanded that processes be *receptive*, i.e. that they be able to receive all messages sent by the environment at any time. A simple approach to this problem leads to models where all possible inputs (i.e. outputs from the environment) at any stage are explicitly described. As a result, infinitary descriptions are obtained even for simple, non–recursive, processes. For example, according to [16], the operational description of the null process $\mathbf{0}$ is the same as that of $recX.a.(\overline{a} \mid X)$, where $a$ stands for any input action, $\overline{a}$ is its complementary output and $rec$ is the recursion operator. Similarly, [5] presents a trace-based model that permits arbitrary "gaps" in traces to take into account any external influence on processes behaviour.

Differently from [16], we build on the usual operational semantics of the language, which just describes what the process intentions are at any stage, and we take advantage of a preorder, $\preceq$, between sequences of actions (traces). The intuition behind $\preceq$ is that whenever a trace $s$ may lead to a successful interaction with the environment and $s' \preceq s$, then $s'$ may lead to success as well. It turns out that, when comparing two processes, only their "minimal" traces need to be taken into account. This leads to a model that assigns finite denotations to finite processes. More precisely, the interpretation of the may preorder ($\underset{m}{\sqsubseteq}$) suggested by the model is as follows: $P \underset{m}{\sqsubseteq} Q$ if, consuming the same messages, $Q$ can produce at least the same messages as $P$.

Building on the above mentioned preorder over traces, we provide a complete (in-)equational axiomatization for asynchronous CCS that relies on the laws:

$$(\mathtt{A1}) \quad a.b.P \sqsubseteq b.a.P \qquad \text{and} \qquad (\mathtt{A2}) \quad a.(\overline{a} \mid P) \sqsubseteq P\,.$$

These two laws are specific to asynchronous testing and are not sound for the synchronous may preorder [11]. The completeness proof relies on the existence of canonical forms directly inspired by the finitary trace–based model.

We develop both the model and the axiomatization first for asynchronous CCS, and then for asynchronous $\pi$-calculus. The simpler calculus is sufficient to isolate the key issues of asynchrony. Indeed, both the trace interpretation and the axiomatization for $\pi$-calculus are dictated by those for CCS.

The rest of the paper is organized as follows. Section 2 introduces asynchronous CCS and the may–testing preorder. Section 3 and 4 present a fully abstract trace–based interpretation of processes and a complete proof system for finite processes, respectively. In Section 5 the results of the previous sections are extended to $\pi$-calculus. The final section contains a few concluding remarks and a brief discussion of related work.

## 2 Asynchronous CCS

In this section we present syntax, operational and testing semantics of asynchronous CCS (ACCS, for short) [7]. It differs from standard CCS because only guarded choices are used and output guards are not allowed. The absence of output guards "forces" asynchrony; it is not possible to define processes that causally depend on output actions.

**Syntax** We let $\mathcal{N}$, ranged over by $a, b, \ldots$, be an infinite set of *names* used to model input actions 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* that model outputs. $\mathcal{N}$ and $\overline{\mathcal{N}}$ are disjoint and are in bijection via the *complementation* function $(\overline{\cdot})$; 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 $\mathcal{L}_\tau = \mathcal{L} \cup \{\tau\}$, where $\tau$ is a distinct action, for the set of all *actions* or *labels*, ranged over by $\mu$. We shall use $A, B, L, \ldots$, to range over subsets of $\mathcal{L}$. 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:

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

where $g_i \in \mathcal{N} \cup \{\tau\}$, 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 $f(\overline{a}) = \overline{f(a)}$. 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).

In the sequel, $\sum_{i\in\{1,2\}} g_i.E_i$ will be abbreviated as $g_1.E_1 + g_2.E_2$, $\sum_{i\in\{1\}} g_i.E_i$ as $g_1.E_1$ and $\sum_{i\in\emptyset} g_i.E_i$ as $\mathbf{0}$; we will also write $g$ for $g.\mathbf{0}$. As usual, we write $E[F/X]$ for the term obtained by replacing each free occurrence of $X$ in $E$ by $F$ (with possible renaming of bound process variables). We write $\mathrm{n}(P)$ to denote the set of visible actions occurring in $P$.

**Operational Semantics** The labelled transition system $(\mathcal{P}, \mathcal{L}_\tau, \xrightarrow{\mu})$ in Figure 1 defines the operational semantics of the language.

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 *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 $P \not\xrightarrow{\tau}$.

$$\begin{array}{llll}
\texttt{AR1}\ \sum_{i\in I} g_i.P_i \xrightarrow{g_j} P_j \quad j \in I & & \texttt{AR2}\ \overline{a} \xrightarrow{\overline{a}} \mathbf{0} & \\
\texttt{AR3}\ \dfrac{P \xrightarrow{\mu} P'}{P\{f\} \xrightarrow{f(\mu)} P'\{f\}} & & \texttt{AR4}\ \dfrac{P \xrightarrow{\mu} P'}{P\backslash L \xrightarrow{\mu} P'\backslash L} & \text{if } \mu \notin L \cup \overline{L} \\
\texttt{AR5}\ \dfrac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q} & & \texttt{AR6}\ \dfrac{P[recX.P/X] \xrightarrow{\mu} P'}{recX.P \xrightarrow{\mu} P'} & \\
\texttt{AR7}\ \dfrac{P \xrightarrow{l} P', \quad Q \xrightarrow{\overline{l}} Q}{P \mid Q \xrightarrow{\tau} P' \mid Q'} & & &
\end{array}$$

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

**May Semantics** We are now ready to instantiate on ACCS the general framework of [11, 15] to obtain the may preorder and equivalence. In the sequel, *observers*, ranged over by $O$, are ACCS processes that can additionally perform a distinct *success* action $\omega$.

**Definition 2.** $P \underset{m}{\underset{\sim}{\sqsubset}} Q$ iff for every observer $O$, $P \mid O \overset{\omega}{\Longrightarrow}$ implies $Q \mid O \overset{\omega}{\Longrightarrow}$ .

We will use $\simeq_m$ to denote the equivalence obtained as the kernel of the preorder $\underset{m}{\underset{\sim}{\sqsubset}}$ (i.e. $\simeq_m \; = \; \underset{m}{\underset{\sim}{\sqsubset}} \cap \underset{m}{\underset{\sim}{\sqsubset}}^{-1}$).

Universal quantification on observers makes it difficult to work with the operational definition of the may preorder; an alternative characterization is on demand. In the synchronous case, this characterization is simply *trace inclusion* (see, e.g., [11, 15]). In [7], by taking advantage of a preorder over single traces, we proved that in case of asynchronous communication a weaker condition is required; we summarize these results below.

**Definition 3.** Let $\preceq$ be the least preorder over $\mathcal{L}^*$ preserved under trace composition and satisfying the following laws

`TO1` $\quad \epsilon \preceq a \qquad$ `TO2` $\quad la \preceq al \qquad$ `TO3` $\quad \epsilon \preceq a\overline{a}$

The intuition behind the the laws in Definition 3 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 `TO1` (*deletion*) says that process inputs cannot be enforced. 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 `TO2` (*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 $\overline{a}\overline{b}\overline{c}$ then it also offers $\overline{b}\overline{a}\overline{c}$. Finally, law `TO3` (*annihilation*) allows the environment to internally consume 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 4.** For processes $P$ and $Q$, we write $P \ll_m Q$ iff whenever $P \overset{s}{\Longrightarrow}$ then there exists $s'$ such that $s' \preceq s$ and $Q \overset{s'}{\Longrightarrow}$ .

**Theorem 5.** For all processes $P$ and $Q$, $P \underset{m}{\underset{\sim}{\sqsubset}} Q$ iff $P \ll_m Q$.

One can easily prove that $\underset{m}{\underset{\sim}{\sqsubset}}$ is a pre–congruence; the proof relies on the coincidence between $\underset{m}{\underset{\sim}{\sqsubset}}$ and $\ll_m$ (however, the case of parallel composition is best dealt with by relying on the definition of $\underset{m}{\underset{\sim}{\sqsubset}}$ ).

## 3 A Finitary Trace-based Model

A fully abstract set-theoretic interpretation for $\underset{m}{\underset{\sim}{\sqsubset}}$ can be obtained by interpreting each $P$ as the set of traces $[\![P]\!]_m = \{s \mid \text{there is } s' \in L(P) : s' \preceq s\}$ and then ordering interpretations by set inclusion. However, this naive interpretation is not satisfactory, because it includes infinitely many traces even for finite processes; for instance, $[\![\mathbf{0}]\!]_m = \{\epsilon, a, a\overline{a}, a\overline{a}a, \ldots, b, b\overline{b}, \ldots\}$.

To obtain a non–redundant interpretation, we shall "minimize" the language of a process $P$, $L(P)$, w.r.t. the trace preorder $\preceq$. In the sequel, we use $[s]$ to denote the $\preceq$-equivalence class of $s$, i.e. the set $\{s' : s' \preceq s \text{ and } s \preceq s'\}$.

**Definition 6.**

- Consider a set $D$ of $\preceq$-equivalence classes. We say that $D$ is a *denotation* if whenever $[s], [s'] \in D$ and $s \preceq s'$ then $[s] = [s']$. We call $\mathcal{D}$ the set of all denotations.
- $\mathcal{D}$ is ordered by setting: $D_1 \leq D_2$ iff for each $[s] \in D_1$ there is $[s'] \in D_2$ such that $s' \preceq s$.

In words, a denotation $D$ is a set of $\preceq$-equivalence classes which are *minimal* elements of $D$.

**Lemma 7.** $(\mathcal{D}, \leq)$ is a partial order.

**Definition 8.** For each $P$, we interpret $P$ as the denotation

$$[\![P]\!]_m \overset{\text{def}}{=} \{[s] \,:\, s \in L(P) \text{ and for each } s' \in L(P) :\; s' \preceq s \text{ implies } [s] = [s']\}\,.$$

*Example 1.*

1. If $P \overset{\text{def}}{=} a.(\overline{a} \,|\, b)$, we have $L(P) = \{\epsilon, a, a\overline{a}, ab, a\overline{a}b, ab\overline{a}\}$ and that $\epsilon$ is minimal in $L(P)$ (by `TO1`–`TO3`), hence $[\![a.(\overline{a} \mid b)]\!]_m = [\![\mathbf{0}]\!]_m = \{\,[\epsilon]\,\}$.

2. If $P \stackrel{\mathrm{def}}{=} \overline{a} \mid b.\overline{c}$, then $L(P) = \{\epsilon, \overline{a}, \overline{a}b, \overline{a}b\overline{c}, b, b\overline{a}, b\overline{ac}, b\overline{c}, b\overline{ca}\}$. The set of $\preceq$-minimal traces of $L(P)$ is $\{\epsilon, \overline{a}, b\overline{c}, \overline{a}b\overline{c}, b\overline{ca}\}$ and $[\![P]\!]_m = \{[\epsilon], [\overline{a}], [b\overline{c}], [\overline{a}b\overline{c}], [b\overline{ca}]\}$.
3. If $P \stackrel{\mathrm{def}}{=} \overline{a} \mid b.\overline{c}$ and $Q \stackrel{\mathrm{def}}{=} a.P$, then $[\![P]\!]_m = \{[\epsilon], [\overline{a}], [b\overline{c}], [\overline{a}b\overline{c}], [b\overline{ca}]\}$ and $[\![Q]\!]_m = \{[\epsilon], [ab\overline{c}], [a\overline{a}b\overline{c}], [ab\overline{ca}]\}$; hence $[\![Q]\!]_m \leq [\![P]\!]_m$.

**Lemma 9.** Let $C$ be a non–empty set of $\preceq$-equivalence classes. Then $C$ has minimal elements (w.r.t. the obvious ordering $[s'] \leq [s]$ iff $s' \preceq s$).

**Theorem 10.** $P \underset{m}{\underset{\sim}{\sqsubset}} Q$ if and only if $[\![P]\!]_m \leq [\![Q]\!]_m$ in $\mathcal{D}$.

Proof: We use the alternative characterization $\ll_m$ of $\underset{m}{\underset{\sim}{\sqsubset}}$. Suppose that $P \ll_m Q$; we show that $[\![P]\!]_m \leq [\![Q]\!]_m$ in $\mathcal{D}$. Let $[s] \in [\![P]\!]_m$, with $P \stackrel{s}{\Longrightarrow}$. Then there is $s'$ s.t. $Q \stackrel{s'}{\Longrightarrow}$ and $s' \preceq s$. Choose now $[s_0]$ which is minimal for the set $\{[s''] : s'' \in L(Q)$ and $s'' \preceq s'\}$, and which exists by virtue of Lemma 9. By definition of $[\![\cdot]\!]_m$, $[s_0] \in [\![Q]\!]_m$, and moreover $s_0 \preceq s$. The converse implication can be proven similarly. $\square$

It is possible to give a "concrete" representation of equivalence classes.

**Proposition 11.** Let $s_1 = m_1 M_1 \cdots m_n M_n$, $n \geq 0$, be any trace, where, for $1 \leq i \leq n$, $m_i$ (resp $M_i$) is a trace containing only inputs (resp. outputs). Suppose that $s_2 \preceq s_1$ and $s_1 \preceq s_2$. Then $s_2$ is of the form $m'_1 M_1 \cdots m'_n M_n$, where, for $1 \leq i \leq n$, $m'_i$ is a permutation of $m_i$.

The above proposition allows one to consider equivalence classes of traces as sequences where *multisets* of input actions alternate with sequences of output actions. This model can be further optimized. For example, when defining $[\![\cdot]\!]_m$ it is possible to enrich the theory of $\preceq$ with a commutativity law for outputs ($\overline{a}\overline{b} \preceq \overline{b}\overline{a}$); this permits viewing sequences of outputs as multisets and yields smaller denotations of processes. For instance, the denotation of process $P$ in Example 1 would reduce to $\{[\epsilon], [\overline{a}], [b\overline{c}], [\overline{a}b\overline{c}]\}$. A similar optimization will be used in the definition of canonical traces, in the next section.

## 4 A Proof System for ACCS

In this section we define a proof system for ACCS and prove that it is sound and complete with respect to $\underset{m}{\underset{\sim}{\sqsubset}}$ for finite (without recursion) processes.

The proof system, that we call $\mathcal{A}$, is based on the in-equational laws in Table 1 plus the usual inference rules for reflexivity, transitivity and substitutivity in any context. We use $G$ to range over guarded sums. Given two guarded sums $G = \sum_{i \in I} g_i.P_i$ and $G' = \sum_{j \in J} g_j.P_j$, we define $G + G'$ as $\sum_{k \in I \cup J} g_k.P_k$. Each equation $P = Q$ is an abbreviation for the pair of inequations $P \sqsubseteq Q$ and $Q \sqsubseteq P$. We write $P \sqsubseteq_{\mathcal{A}} Q$ ($P =_{\mathcal{A}} Q$) to indicate that $P \sqsubseteq Q$ ($P = Q$) can be derived within the proof system $\mathcal{A}$.

| | | |
|---|---|---|
| `C1` | $G + G = G$ | |
| `P1` | $P \mid \mathbf{0} = P$ | |
| `P2` | $P \mid Q = Q \mid P$ | |
| `P3` | $P \mid (Q \mid R) = (P \mid Q) \mid R$ | |
| `EXP` | Let $G = \sum_{i\in I} g_i.P_i$ and $G' = \sum_{j\in J} g'_j.P'_j$; then:<br>$G \mid G' = \sum_{i\in I} g_i.(P_i \mid G') + \sum_{j\in J} g'_j.(G \mid P'_j)$ | |
| `R1` | $(\sum_{i\in I} g_i.P_i)\{f\} = \sum_{i\in I} f(g_i).P_i\{f\}$ | |
| `R2` | $(P \mid Q)\{f\} = P\{f\} \mid Q\{f\}$ | |
| `R3` | $\overline{a}\{f\} = f(\overline{a})$ | |
| `H1` | $(\sum_{i\in I} g_i.P_i)\backslash L = \sum_{i\in I \wedge g_i \notin L\cup\overline{L}} g_i.P_i\backslash L$ | |
| `H2` | $(P \mid Q)\backslash L = P \mid Q\backslash L$ | $L \cap \mathrm{n}(P) = \emptyset$ |
| `H3` | $(P\backslash L_1)\backslash L_2 = P\backslash L_1 \cup L_2$ | |
| `H4` | $(\overline{a} \mid g.P)\backslash a = g.(\overline{a} \mid P)\backslash a$ | $g \neq a$ |
| `H5` | $(\overline{a} \mid g.P)\backslash a = P\backslash a$ | $g = a$ |
| `T1` | $\overline{a} \mid \sum_{i\in I} g_i.P_i = \sum_{i\in I} \tau.(\overline{a} \mid g_i.P_i)$ | |
| `T2` | $g.\sum_{i\in I} g_i.P_i = \sum_{i\in I} g.g_i.P_i$ | |
| `T3` | $P = \tau.P$ | |
| `T4` | $G \sqsubseteq G + G'$ | |
| `T5` | $a.(\overline{b} \mid P) \sqsubseteq \overline{b} \mid a.P$ | |
| `T6` | $P \sqsubseteq \overline{a} \mid a.P$ | |
| `A1` | $a.b.P \sqsubseteq b.a.P$ | |
| `A2` | $a.(\overline{a} \mid P) \sqsubseteq P$ | |

**Table 1.** Laws for ACCS

Laws `A1` and `A2` differentiate asynchronous from synchronous may testing: they are not sound for the synchronous may preorder [11]. In particular, law `A1` states that processes are insensitive to the arrival ordering of messages from the environment, while law `A2` states that any execution of $P$ that depends on the availability of a message $\overline{a}$ is worse than $P$ itself, even if $\overline{a}$ is immediately re-issued. The other laws in Table 1 are sound also for the synchronous may testing [11]. The laws in Table 1 can be easily proven sound by taking advantage of the preorder $\ll_m$ .

Let us now consider some derived laws, among which (`D1`) $a.P \sqsubseteq_{\mathcal{A}} P$ and (`D2`) $\mathbf{0} \sqsubseteq_{\mathcal{A}} \overline{a}$. Law `D2` follows immediately from law `T4`. The inequality `D1` can be derived by first noting that from `D2` it follows $P \sqsubseteq_{\mathcal{A}} \overline{a} \mid P$, which implies $a.P \sqsubseteq_{\mathcal{A}} a.(\overline{a} \mid P)$; now apply `A2`. In particular, we have that $a \sqsubseteq_{\mathcal{A}} \mathbf{0}$. From $\mathbf{0} \sqsubseteq_{\mathcal{A}} P$, for any $P$ (a consequence of `T4`), and $a.\overline{a} =_{\mathcal{A}} a.(\overline{a} \mid \mathbf{0}) \sqsubseteq_{\mathcal{A}} \mathbf{0}$ (law `A2`), we get $a.\overline{a} =_{\mathcal{A}} \mathbf{0}$.

For proving completeness of the proof system, we shall rely on the existence of canonical forms for processes, which are *unique* up to associativity and commutativity of summation and parallel composition and up to permutation of consecutive input actions. Uniqueness is a result of independent interest, because it leads to unique (and rather compact) representatives for equivalence classes of processes. The canonical form of a process will be obtained by minimizing its set of traces via a trace preorder, that extends $\preceq$ with a commutativity law for output actions.

**Definition 12.** Let $\preceq_{|}$ be the least preorder over traces induced by the laws `T01`–`T03` plus law: (`T04`) $\overline{a}\overline{b} \preceq \overline{b}\overline{a}$.

Of course, $\preceq$ is included in $\preceq_{|}$.

**Definition 13 (canonical forms).**

- Given $s \in Act^*$, the process $t(s)$ is defined by induction on $s$ as follows: $t(\epsilon) \overset{\mathrm{def}}{=} \mathbf{0}$, $t(as') \overset{\mathrm{def}}{=} a.t(s')$ and $t(\overline{a}s') \overset{\mathrm{def}}{=} \overline{a} \,|\, t(s')$.
- Consider $A \subseteq_{\mathrm{fin}} \mathcal{L}^*$. We say that $A$ is:
  - *complete* if whenever $t(r) \overset{s}{\Longrightarrow}$, for $r \in A$, then there is $s' \in A$ s.t. $s' \preceq_{|} s$;
  - *minimal* if whenever $s, s' \in A$ and $s' \preceq_{|} s$ then $s' = s$.
- A *canonical form* is a process of the form $\sum_{s \in A - \{\epsilon\}} \tau.t(s)$, for some $A \subseteq_{\mathrm{fin}} \mathcal{L}^*$ which is both complete and minimal.

Note that a complete set of traces always contains the empty trace $\epsilon$. The proof of uniqueness of canonical forms can be decomposed into three simple lemmata.

**Lemma 14.** If $t(s) \overset{s'}{\Longrightarrow}$ then $t(s') \sqsubseteq_{\mathcal{A}} t(s)$.

Proof: The proof proceeds by induction on the length of $s$. The most interesting case is when $s = \overline{a}s_0$, for some $s_0$; hence $t(s) = \overline{a} \,|\, t(s_0)$. Then there are two cases for $s'$: either $t(s_0) \overset{s'}{\Longrightarrow}$, and then the thesis follows from $P \sqsubseteq_{\mathcal{A}} \overline{a} \,|\, P$ (by `D2`) and induction hypothesis, or $s' = \sigma\overline{a}\rho$, with $t(s_0) \overset{\sigma\rho}{\Longrightarrow}$, for some traces $\sigma$ and $\rho$. In the latter case, we get from the induction hypothesis that $t(\sigma\rho) \sqsubseteq_{\mathcal{A}} t(s_0)$; hence $\overline{a} \,|\, t(\sigma\rho) \sqsubseteq_{\mathcal{A}} \overline{a} \,|\, t(s_0) = t(s)$; from repeated applications of `P2`) and `T5`, we get $t(s') = t(\sigma\overline{a}\rho) \sqsubseteq_{\mathcal{A}} \overline{a} \,|\, t(\sigma\rho)$, and hence the thesis. □

**Lemma 15.** Let $C_1 \overset{\mathrm{def}}{=} \sum_{s \in A - \{\epsilon\}} \tau.t(s)$ and $C_2 \overset{\mathrm{def}}{=} \sum_{r \in B - \{\epsilon\}} \tau.t(r)$ be canonical forms such that $C_1 \underset{\sim}{\sqsubset}_m C_2$. Then for each $s \in A$ there is $r \in B$ such that $r \preceq_{|} s$.

Proof: Let $s \in A$. Then $C_1 \overset{s}{\Longrightarrow}$, thus, since $C_1 \ll_m C_2$, there is $s'$ s.t. $C_2 \overset{s'}{\Longrightarrow}$ and $s' \preceq s$. This implies, by completeness of $B$, that there is $r \in B$ such that $r \preceq_{|} s'$. Since $s' \preceq s$, we obtain that $r \preceq_{|} s$. □

We write $P_1 =_{AC} P_2$ if $P_1 =_{\mathcal{A}} P_2$ can be derived using only the laws `C2`–`C3`, `P2`–`P3` and `A1`. For the proof of the following lemma, just note that whenever $s_1$ and $s_2$ are $\preceq_{|}$-equivalent, then only laws `T02` and `T04` can be used to derive $s_1 \preceq_{|} s_2$ and $s_2 \preceq_{|} s_1$.

**Lemma 16.** If $s_1 \preceq_{|} s_2$ and $s_2 \preceq_{|} s_1$ then $t(s_1) =_{AC} t(s_2)$.

**Theorem 17 (uniqueness).** Let $C_1$ and $C_2$ be canonical forms such that $C_1 \simeq_m C_2$. Then $C_1 =_{AC} C_2$.

Proof: Suppose $C_1 = \sum_{s \in A - \{\epsilon\}} \tau.t(s)$ and $C_2 = \sum_{r \in B - \{\epsilon\}} \tau.t(r)$. We prove that for each $s \in A$ there is $r \in B$ s.t. $s \preceq_{|} r$ and $r \preceq_{|} s$, by which the result will follow by Lemma 16 and by symmetry. Suppose that $s \in A$. Since $C_1 \underset{m}{\underset{\sim}{\sqsubset}} C_2$, by Lemma 15, we deduce that there is $r \in B$ s.t. $r \preceq_{|} s$. But since $C_2 \underset{m}{\underset{\sim}{\sqsubset}} C_1$ as well, we deduce the existence of $s' \in A$ with $s' \preceq_{|} r$, hence $s' \preceq_{|} r \preceq_{|} s$. By minimality of $A$ we deduce that $s = s' \preceq_{|} r$. □

*Example 2.* Consider $P \stackrel{\text{def}}{=} \tau.(\overline{a} \mid b.\overline{b}) + \tau.b.(\overline{a} \mid \overline{b})$. To get the canonical form of $P$, we first compute the language of $P$ and obtain the complete set $\{\epsilon, \overline{a}, b, \overline{a}b, b\overline{a}, b\overline{b}, ab\overline{b}, b\overline{a}\overline{b}, b\overline{b}\overline{a}\}$. Then we minimize, thus finding the minimal set $\{\epsilon, \overline{a}\}$, which is also complete. Thus $\tau.\overline{a}$ is the canonical form of $P$.

We proceed now to prove completeness of the proof system.

**Lemma 18 (absorption).** If $s' \preceq_{|} s$ then $t(s) \sqsubseteq_{\mathcal{A}} t(s')$.

Proof: We prove the thesis by induction on the number $n$ of times the laws `TO1`–`TO4` are used to derive $s' \preceq_{|} s$. The proof relies on the laws `D1`, `D2`, `A2` and `P2`. As an example, we analyze the base case ($n = 1$), when $s' \preceq_{|} s$ is derived with one application of `TO3`. This means that $s' = \sigma a \overline{a} \rho$ and $s = \sigma\rho$, for some $a$ and some traces $\sigma$ and $\rho$. Now, note that whenever $s = s_1 s_2$ then $t(s) = t(s_1)[t(s_2)]$, where the latter term is obtained by replacing the single occurrence of $\mathbf{0}$ in $t(s_1)$ with $t(s_2)$. Therefore, by congruence of $\sqsubseteq_{\mathcal{A}}$ and law `A2`, we get:

$$t(s) = t(\sigma)[a.(\overline{a} \mid t(\rho))] \sqsubseteq_{\mathcal{A}} t(\sigma)[t(\rho)] = t(s')\,.$$

□

**Lemma 19.** For each $P$ there exists a canonical form $C$ s.t. $P =_{\mathcal{A}} C$.

Proof: By induction on $P$ and using the laws in Table 1 it is easy to show that $P$ is provably equivalent to some process $C_1 = \sum_{s \in A_1 - \{\epsilon\}} \tau.t(s)$, for some set $A_1$. Consider now the following two facts:

1. Whenever $t(s) \stackrel{s'}{\Longrightarrow}$ then $t(s) =_{\mathcal{A}} \tau.t(s) + \tau.t(s')$.
2. Let $A$ be a complete set. Suppose that there are $s, s' \in A$ s.t. $s \preceq_{|} s'$ and $s \neq s'$. Then: (a) $A - \{s'\}$ is complete, and (b) $\sum_{r \in A - \{\epsilon\}} \tau.t(r) =_{\mathcal{A}} \sum_{r \in A - \{\epsilon, s'\}} \tau.t(r)$.

(1 is a consequence of Lemma 14; 2 derives from the definition of complete set and, for part (b), of Lemma 18 and law `C1`).

By repeatedly applying 1, we can 'saturate' $A_1$, thus proving $C_1$ equivalent to a summation $C_2$ over a complete set $A_2$. Then, by repeatedly applying 2, we can remove redundant traces in $A_2$, thus proving $C_2$ equivalent to a summation over a complete and minimal set of traces. □

**Theorem 20 (completeness).** For finite ACCS processes $P$ and $Q$, $P \underset{\sim}{\sqsubset}_{m} Q$ implies $P \sqsubseteq_{\mathcal{A}} Q$.

Proof: Lemma 19 allows us to assume that both $P$ and $Q$ are in canonical form: $P \stackrel{\mathrm{def}}{=} \sum_{s \in A-\{\epsilon\}} \tau.t(s)$ and $Q \stackrel{\mathrm{def}}{=} \sum_{r \in B-\{\epsilon\}} \tau.t(r)$. It is sufficient to show that for each $s \in A$ there is $r \in B$ s.t. $t(s) \sqsubseteq_{\mathcal{A}} t(r)$, by which the thesis will follows thanks to the law T4. But this fact follows by Lemmata 15 and 18. $\Box$

## 5 The $\pi$-calculus

In this section we discuss the extensions of our theory to the asynchronous variant of $\pi$-calculus [16, 8, 14, 1].

**Syntax and semantics** We assume existence of a countable set $\mathcal{N}$ of *names* ranged over by $a, b, \ldots, x, \ldots$. Processes are ranged over by $P$, $Q$ and $R$. The syntax of asynchronous $\pi$–calculus contains the operators of inaction, output action, guarded summation, restriction, parallel composition, matching and replication:

$$P ::= \overline{a}b \;\mid\; \textstyle\sum_{i \in I} \alpha_i.P_i \;\mid\; \nu\, a\, P \;\mid\; P_1 \,|\, P_2 \;\mid\; [a=b]P \;\mid\; !\,P$$

where $\alpha$ is an *input action* $a(b)$ or a *silent action* $\tau$. We adopt for the sum operator the same shorthands as for ACCS. *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 $\mathrm{fn}(P) \cup \mathrm{bn}(P)$. We shall consider processes up to $\alpha$-equivalence. Thus $\alpha$-equivalent processes have the same transitions and all bound names are always assumed to be different from each other and from the free names. The tilde $\tilde{\cdot}$ will be used to denote tuples of names; when convenient, we shall regard a tuple simply as a set. We omit the definition of operational semantics (see e.g. [1]), but remind that labels on transitions (*actions*), ranged over by $\mu$, can be of four forms: $\tau$ (interaction), $ab$ (input at $a$ of $b$), $\overline{a}b$ (output at $a$ of $b$) or $\overline{a}(b)$ (bound output at $a$ of $b$). 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.

The definition of the may preorder over the $\pi$-calculus, $\underset{\sim}{\sqsubset}_{m}$, is formally the same as for ACCS. Due to the presence of matching (see e.g. [1]), $\underset{\sim}{\sqsubset}_{m}$ is not preserved by input prefix.

**The trace preorder** We extend the operational semantics of the $\pi$-calculus with the following rule: if $P \xrightarrow{ab} P'$ and $b \notin \mathrm{fn}(P)$ then $P \xrightarrow{a(b)} P'$. 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 (non-$\tau$) actions, including bound inputs, and let $\theta$ 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 $s'$, $\theta$ and $s''$, then $\mathrm{bn}(\theta)$ does not occur in $s'$ and $\mathrm{bn}(\theta)$ is different from any other bound name occurring in $s''$. Functions

$\mathrm{bn}(\cdot)$ and $\mathrm{fn}(\cdot)$ are extended to normal traces as expected. We consider normal traces up to $\alpha$-equivalence. 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.* 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 $\overline{\overline{a}(b)} \stackrel{\mathrm{def}}{=} a(b)$; note that $\overline{\overline{s}} = s$.

| | | |
|---|---|---|
| `P1` | $s.s' \preceq s.\theta.s'$ | if $\theta$ is an input action and $\mathrm{bn}(\theta) \cap \mathrm{n}(s') = \emptyset$ |
| `P2` | $s.\theta'.\theta.s' \preceq s.\theta.\theta' s'$ | if $\theta$ is an input action and $\mathrm{bn}(\theta) \cap \mathrm{n}(\theta') = \emptyset$ |
| `P3` | $s.s' \preceq s.\theta.\overline{a}b.s'$ | if $\theta = ab$ or ($\theta = a(b)$ and $b \notin \mathrm{n}(s')$) |
| `P4` | $s.\overline{a}c.(s'\{c/b\}) \preceq s.\overline{a}(b).s'$ | |

**Fig. 2.** Trace ordering laws over $\mathcal{T}$.

The presence of bound names requires a slightly different definition of the trace preorder $\preceq$, which is given below.

**Definition 21.** Let $\preceq_0$ the least binary relation induced by the laws in Figure 2: $\preceq$ is the reflexive and transitive closure of $\preceq_0$.

Rules `P1`, `P2`, `P3` are the natural extensions to asynchronous $\pi$-calculus of the rules for ACCS. Here, some extra attention has to be paid to bound names: an output action declaring a new name (bound output) cannot be postponed after those actions that use that name. As an example, action $\overline{a}(b)$ cannot be postponed after $b(c)$, in any execution of the observer $\nu\, b\, (\overline{a}b \mid b(c).O)$. Accordingly, in the observed process, an input action receiving the new name, $a(b)$, cannot be postponed after output actions at $b$.

Rule `P4` is specific to $\pi$-calculus, and is linked to the impossibility for observers to fully discriminate between free and bound outputs. Informally, rule `P4` states that if a bound (hence new) name is "acceptable" for an observer, then any public name is acceptable as well. Rule `P4` would disappear if we extended the language with the *mismatch* ($[a \neq b]P$) operator, considered e.g. in [6], which permits a full discrimination between free and bound outputs.

The definition of $\ll_m$ for the $\pi$-calculus relies on the trace preorder $\preceq$ and remains formally unchanged w.r.t. ACCS. In [7], we prove that $\ll_m$ and $\underset{m}{\underset{\sim}{\sqsubset}}$ coincide for the $\pi$-calculus. All the results obtained for ACCS about the trace–based model carry over smoothly to the $\pi$-calculus.

**The proof system** A sound and complete proof system for $\underset{m}{\underset{\sim}{\sqsubset}}$ over the finite (without replication) part of the language can be obtained by "translating" the proof system for ACCS into $\pi$-calculus, and then adding four new laws, as done in Table 2. `I1` replaces the substitutivity rule for input prefix, `M1` and `M2` are concerned with matching, and `S1` is related to the law `P4` for $\preceq$.

We write $P \sqsubseteq_\pi Q$ if the inequality $P \sqsubseteq Q$ is derivable within the system of Table 2. Soundness of the system is straightforward. Completeness requires an

| | | |
|---|---|---|
| I1 | if for each $b \in \mathrm{fn}(P,Q)$ $P\{b/x\} \sqsubseteq Q\{b/x\}$ then $a(x).P \sqsubseteq a(x).Q$ | |
| M1 | $[a=b]P = \mathbf{0}$ | $a \neq b$ |
| M2 | $[a=a]P = P$ | |
| C1 | $G+G=G$ | |
| P1 | $P \mid \mathbf{0} = P$ | |
| P2 | $P \mid Q = Q \mid P$ | |
| P3 | $P \mid (Q \mid R) = (P \mid Q) \mid R$ | |
| EXP | Let $G = \sum_{i\in I} \alpha_i.P_i$ and $G' = \sum_{j\in J} \alpha'_j.P'_j$, where each $\alpha_i$ (resp. $\alpha'_j$) does not bind free names of $G'$ (resp. $G$). Then: $G \mid G' = \sum_{i\in I} \alpha_i.(P_i \mid G') + \sum_{j\in J} \alpha'_j.(G \mid P'_j)$ | |
| H1 | $(\nu\widetilde{b})(\sum_{i\in I} \alpha_i.P_i) = \sum_{i\in I \wedge \mathrm{n}(\alpha_i)\cap\widetilde{b}=\emptyset} \alpha_i.(\nu\widetilde{b})P_i$ | |
| H2 | $(\nu\widetilde{b})(P \mid Q) = P \mid (\nu\widetilde{b})Q$ | $\widetilde{b} \cap \mathrm{n}(P) = \emptyset$ |
| H3 | $(\nu a)(\overline{a}b \mid \alpha.P) = \alpha.(\nu a)(\overline{a}b \mid P)$ | $a \notin \mathrm{n}(\alpha)$ |
| H4 | $(\nu a)(\overline{a}b \mid a(c).P) = (\nu a)(P\{b/c\})$ | |
| T1 | $\overline{a}b \mid \sum_{i\in I} \alpha_i.P_i = \sum_{i\in I} \tau.(\overline{a}b \mid \alpha_i.P_i)$ | |
| T2 | $\alpha.\sum_{i\in I} \alpha_i.P_i = \sum_{i\in I} \alpha.\alpha_i.P_i$ | |
| T3 | $P = \tau.P$ | |
| T4 | $G \sqsubseteq G + G'$ | |
| T5 | $a(c).(\overline{b}d \mid P) \sqsubseteq \overline{b}d \mid a(c).P$ | $c \neq b$, $c \neq d$ |
| T6 | $P\{b/c\} \sqsubseteq \overline{a}b \mid a(c).P$ | |
| A1 | $a(c).b(d).P \sqsubseteq b(d).a(c).P$ | $c \neq b$, $c \neq d$ |
| A2 | $a(c).(\overline{a}c \mid P) \sqsubseteq P$ | $c \notin \mathrm{n}(P)$ |
| S1 | $(\nu c)P \sqsubseteq P\{b/c\}$ | |

**Table 2.** Laws for the asynchronous $\pi$-calculus

appropriate definition of canonical form. This implies extending $\preceq$ via commutativity for output actions.

**Definition 22.** Let $\preceq_{|}$ be the trace preorder over $\mathcal{T}$ induced by laws P1–P4 *plus* the laws:

- (P5) $s.\theta.\theta'.s' \preceq s.\theta'.\theta.s'$ if $\mathrm{bn}(\theta) \cap \mathrm{fn}(\theta') = \emptyset$ and $\mathrm{bn}(\theta') \cap \mathrm{fn}(\theta) = \emptyset$ ;
- (P6) $s.\overline{a}(b).\overline{c}b.s' \preceq s'.\overline{c}(b).\overline{a}b.s$ if $c \neq b$.

**Definition 23 (canonical forms).** Let $s$ be a normal trace. The process $t(s)$ is defined by induction on $s$ as follows: $t(\epsilon) \overset{\mathrm{def}}{=} \mathbf{0}$, $t(\overline{a}(b).s') \overset{\mathrm{def}}{=} \nu\, b\,(\overline{a}b \mid t(s'))$, $t(\overline{a}b.s') \overset{\mathrm{def}}{=} \overline{a}b \mid t(s')$, $t(a(c).s') \overset{\mathrm{def}}{=} a(c).t(s')$ and $t(ab.s') \overset{\mathrm{def}}{=} a(x).[x=b]t(s')$ ($x$ fresh).

Modulo the new definitions of $t(s)$ and of $\preceq_{|}$, the definitions of *complete* set, of *minimal* set and of *canonical form* remain formally as in Definition 13.

**Lemma 24.** If $t(s) \stackrel{s'}{\Longrightarrow}$ then $t(s') \sqsubseteq_{\pi} t(s)$.

Proof: The proof parallels that of Lemma 14. We analyze only the case when $s = \overline{a}(b).s_0$, hence $t(s) = \nu\, b\,(\overline{a}b \mid t(s_0))$. There are four possible cases for $s'$ depending on how the execution of actions in $t(s_0)$ and action $\overline{a}b$ are interleaved.

1. $t(s_0) \stackrel{s'}{\Longrightarrow}$ (action $\overline{a}b$ is not fired at all);
2. $s' = \sigma.\overline{a'}(b).\rho$ and $t(s_0) \stackrel{\sigma.\overline{a'}b.\rho}{\Longrightarrow}$ ;
3. $s' = \sigma.\overline{a}(b).\rho$ and $t(s_0) \stackrel{\sigma.\rho}{\Longrightarrow}$ ;
4. $s' = \sigma_1.\overline{a'}(b).\sigma_2.\overline{a}b.\rho$ and $t(s_0) \stackrel{\sigma_1.\overline{a'}b.\sigma_2.\rho}{\Longrightarrow}$ .

For case 1, the thesis follows from induction hypothesis. We analyze now case 4, because 2 and 3 are easier. By induction hypothesis, $t(\sigma_1.\overline{a'}b.\sigma_2.\rho) \sqsubseteq_{\mathcal{A}} t(s_0)$, hence

$$T \stackrel{\mathrm{def}}{=} \nu\, b\,(\overline{a}b \mid t(\sigma_1.\overline{a'}b.\sigma_2.\rho)) \sqsubseteq_{\pi} \nu\, b\,(\overline{a}b \mid t(s_0)) = t(s).$$

On the other hand, by repeatedly applying `T5` and `P2`, we can push $\overline{a}b$ rightward inside $T$ and get $\nu\, b\, t(\sigma_1.\overline{a'}b.\sigma_2.\overline{a}b.\rho) \sqsubseteq_{\pi} T$. Finally, since $b \notin \mathrm{n}(\sigma_1)$, we can push $\nu\, b$ rightward (using `H1` and `H2`) until it reaches $\overline{a'}b$, to get $t(s') \sqsubseteq_{\pi} T$, and the thesis. □

**Lemma 25.** If $s' \preceq_{|} s$ then $t(s) \sqsubseteq_{\mathcal{A}} t(s')$.

Proof: The thesis is proven by induction on the number $n$ of times the laws `P1`–`P6` are used to derive $s' \preceq_{|} s$. As an example, we analyze the base case ($n = 1$), when $s' \preceq_{|} s$ is derived with one application of `P3`. In particular, consider the case $s' = \sigma.ab.\overline{a}b.\rho$ and $s = \sigma\rho$, for some $a$, $b$ and some traces $\sigma$ and $\rho$. For any $P$ and fresh $x$, we have that $a(x).[x = b](\overline{a}b \mid P) \sqsubseteq_{\mathcal{A}} a(x).(\overline{a}x \mid P)$ (use rule `I1` and laws `M1` and `M2`). This inequality can be proven under any substitution $\sigma$ for the names in $\mathrm{fn}(P) \cup \{a, b\}$, hence under any context. From this and `A2`, we get:

$$t(s) = t(\sigma)[a(x).[x = b](\overline{a}b \mid t(\rho))] \sqsubseteq_{\mathcal{A}} t(\sigma)[a(x).(\overline{a}x \mid t(\rho))] \sqsubseteq_{\mathcal{A}} t(\sigma)[t(\rho)] = t(s')\,.$$

□

The proof of uniqueness of canonical forms remains essentially unchanged. The proof of existence of provably equivalent canonical forms requires the following derived laws:

**(1)** $a(y).[b = c]P =_{\pi} \tau.[b = c]a(y).P + \tau.a(y)$ if $y \notin \{b, c\}$, and
**(2)** $a(b).[b = c]P =_{\pi} a(b).[b = c]P\{c/b\}$.

These are used to accommodate matching, when initially proving that $P$ is equivalent to a summation of $t(s)$'s; then, the proof proceeds formally unchanged. Given the existence and the uniqueness of canonical forms, the actual proof of completeness remains essentially unchanged.

**Theorem 26 (completeness).** For finite $\pi$-calculus processes $P$ and $Q$, $P \underset{\sim}{\sqsubset}_{m} Q$ implies $P \sqsubseteq_{\pi} Q$.

## 6 Conclusions and Related Works

In this paper, we have studied a may testing semantics for two asynchronous variants of CCS and $\pi$-calculus. For both calculi we have proposed a finitary trace–based interpretation of processes and a complete inequational proof system.

Recently, there have been various proposals of models of asynchronous processes. Two main approaches have been followed to this purpose. They differ in the way (non–blocking) output actions are modelled. The asynchronous variants of ACP [4], CSP [17] and LOTOS [21] introduce external buffers in correspondence of output channels. This makes outputs non–blocking and immediately executable, while preserving the orderings between different output actions. Within the same group we can place the work on the actors foundation [3]. Differently, the asynchronous variants of $\pi$-calculus [16, 8, 14, 1] and CCS [20, 12, 9] model output prefix $\overline{a}.P$ as a parallel composition $\overline{a}\,|\,P$, i.e. output actions are independent processes. The communication medium is rendered as a bag of messages, which is directly represented within the syntax as a parallel composition of output actions.

In the past, all these formalisms have been equipped with observational semantics based on bisimulation or failures, but very few denotational or equational characterizations have been studied. A notable exception is the work by de Boer, Palamidessi and their collaborators. On one hand, in [5], they propose a trace-based model for a variant of failure semantics, on the other, in [4], they provide axiomatizations that rely on state operators and explicitly model evolution of buffers. Other studies deal with languages that fall in the first group of asynchronous formalisms and propose set of laws that help to understand the proposed semantics, but do not offer complete axiomatizations [21, 3]. For those languages that model outputs by means of processes creation, the only paper that presents an axiomatization is [1]. There, a complete axiomatization of strong bisimilarity for asynchronous $\pi$-calculus is proposed, but the problem of axiomatizing weak ($\tau$-forgetful) variants of the equivalence is left open.

A paper closely related to ours is the recent [10]. There, for a variant of asynchronous CCS, the authors present a complete axiomatization of *must* testing semantics, which is more appropriate for reasoning about liveness properties. No finitary model is presented and the problem of extending the results to the asynchronous $\pi$-calculus is left open.

**Acknowledgments.** Five anonymous referees provided valuable suggestions; Istituto di Elaborazione dell'Informazione in Pisa made our collaboration possible.

## References

1. R.M. Amadio, I. Castellani, D. Sangiorgi. On Bisimulations for the Asynchronous $\pi$–calculus. *CONCUR'96*, *LNCS* 1119, pp.147-162, Springer, 1996.

2. M. Abadi, A.D. Gordon: A calculus for cryptographic protocols: The Spi calculus. Proc. *4th ACM Confeence on Computer and Communication Security*, ACM Press, 1997.
3. G.A. Agha, I.A. Mason, S.F. Smith, C.L. Talcott. A foundation for actor computation. *Journal of Functional Programming*, 7:1-72, 1997.
4. F.S. de Boer, J.W. Klop, C. Palamidessi. Asynchronous Communication in Process Algebra. *LICS'92*, IEEE Computer Society Press, pp. 137-147, 1992.
5. F.S. de Boer, J.N. Kok, C. Palamidessi, J.J.M.M. Rutten. The Failure of Failures in a Paradigm for Asynchronous Communication. *CONCUR'91*, *LNCS* 527, pages 111-126, Springer, 1991.
6. M. Boreale, R. De Nicola. Testing Equivalence for Mobile Systems. *Information and Computation*, 120: 279-303, 1995.
7. M. Boreale, R. De Nicola, R. Pugliese. Asynchronous Observations of Processes. *FoSSaCS'98*, *LNCS* , Springer, 1998.
8. G. Boudol. Asynchrony in the $\pi$–calculus (note). Rapport de Recherche 1702, INRIA Sophia–Antipolis, 1992.
9. N. Busi, R. Gorrieri, G-L. Zavattaro. A process algebraic view of Linda coordination primitives. Technical Report UBLCS-97-05, University of Bologna, 1997.
10. I. Castellani, M. Hennesy. Testing Theories for Asynchronous Languages. Proc. *FSTTCS*, *LNCS* , to appear Dec. 1998.
11. R. De Nicola, M.C.B. Hennessy. Testing Equivalence for Processes. *Theoretical Computers Science*, 34:83-133, 1984.
12. R. De Nicola, R. Pugliese. A Process Algebra based on Linda. *COORDINATION'96*, *LNCS* 1061, pp.160-178, Springer, 1996.
13. D. Gelernter. Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80-112, 1985.
14. M. Hansen, H. Huttel, J. Kleist. Bisimulations for Asynchronous Mobile Processes. *In Proc. of the Tblisi Symposium on Language, Logic, and Computation*, 1995.
15. M.C.B. Hennessy. *Algebraic Theory of Processes.* The MIT Press, 1988.
16. K. Honda, M. Tokoro. An Object Calculus for Asynchronous Communication. *ECOOP'91*, *LNCS* 512, pp.133-147, Springer, 1991.
17. 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.
18. R. Milner. The Polyadic $\pi$-calculus: A Tutorial. Technical Report, University of Edinburgh, 1991.
19. J. Parrow, D. Sangiorgi. Algebraic theories for name-passing calculi. *Information and Computation*, 120(2):174–197, 1995.
20. 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.
21. J. Tretmans. A formal approach to conformance testing. Ph.D. Thesis, University of Twente, 1992.