Testing Theories for Broadcasting Processes

Cristian ENE[1], Traian MUNTEAN[1,2]

[1] University of Marseilles; Parc Scientifique de Luminy - Case 925
F-13288 Marseille, France
tel: 33 - 491 82 85 32, fax: 33 - 491 82 85 11
cene@esil.univ-mrs.fr, muntean@lim.univ-mrs.fr
[2] CNRS

Abstract. This paper presents a theory of testing for processes calculi which have broadcast as basic communication primitive. Firstly, we justify the necessity of an alternative theory to bisimulations for broadcasting calculi. Then, we remind CBS without message passing and $b\pi$-calculus, and we adapt the definitions of *may testing* and *must testing* in the framework of broadcasting calculi. Finally, we give a direct characterization for these pre-orders.

1 Introduction

Communication between processes is the main aspect of concurrency when dealing with distributed and/or parallel computing. One can specify basic communications from several points of view; primitives interactions can be, for instance, synchronous or asynchronous, associated to point-to-point or broadcast (one-to-many) message exchange protocols. The theory behind point-to-point communication is today well-established in process algebra (e.g. started with Milner's CCS and Hoare's CSP pioneering works). On the other hand more complex and higher level communication schemes, like broadcast or multicast are encountered in many applications and programming models, but they remain nevertheless poorly represented in the algebraic theory of distributed systems. We emphasize here that group interactions shall be considered as a more appropriate exchange scheme for modelling and reasoning about many communicating systems and networking applications (e.g. multimedia, data and knowledge mining, mobile computing). Group communications are, in our opinion, a more abstract and higher level concept of interaction in distributed computing than the commonly used point-to-point communications, usually expressed by handshaking message-passing primitives or by remote invocations. Broadcast has been even chosen as a hardware exchange primitive for some local networks, and in this case point-to-point message passing (when needed) is to be implemented on top of it. Primitives for broadcast programming offer several advantages: processes may interact without having explicit knowledge of each other, receivers may be dynamically added or deleted without modifying the emitter, and activity of a process can be monitored without modifying the behaviour of the observed process (this is clearly not the case with the classical rendez-vous communications). Moreover, from a theoretical point of view, it appears difficult [5] to encode broadcast in calculi based on point-to-point communications.

Thus, developing an algebraic theory for models based on broadcast communication has its own interest. Hoare's CSP [12] is based on a multiway synchronisation mechanism, but it does not make any difference between input and output. Or, in a broadcast setting the anti-symmetrie between these two kinds of actions is particularly important (in a broadcast communication there is one sender, and an unbounded or possibly empty set of receivers;

this is well represented in the I/O automata of Lynch and Tuttle ([13]) where outputs are non-blocking and locally controlled, whereas inputs are externally controlled and can not be refused). In [16], Prasad introduces and develops [17] a calculus of broadcasting systems, namely CBS. His calculus, inspired by Milner's CCS ([14]), has as main goal to provide a formal model for packets broadcast in Ethernet-like communication media. It is based on broadcast, but its main limitation is that it does not allow to model reconfigurable finer topologies of networks of processes which communicate by broadcast (as dynamic group communications). It is up to the receiver to use the received value or to discard it. In [11], Hennessy and Rathke present a process calculus based on broadcast with a more restrictive input ($x \in S?p$), but the continuation process p, do not change dynamically his restrictions on further inputs; so it cannot model reconfigurable systems based on broadcast.

In [5], Ene and Muntean introduce $b\pi$-calculus as a framework which combine mobility and broadcast (as it is the case for processes which use group communications *la PVM* [9], buses-based reconfigurable architectures or Packet Radio Networks). In [8], the authors develops for $b\pi$-calculus a theory of co-inductive equivalences and congruences. Bisimulations have been successfully used in processes algebra to compare two systems according to their operational ability to simulate each other. Bisimulations are appropriate for distributed reactive systems, which often have infinite behaviour, and make actions as answers to extern stimuli. In addition, bisimulations seem well adapted for point-to-point systems, since the execution of such a system is entirely controlled by the environment (or by a outside observer). In a broadcast calculus, only the inputs are controlled by the environment, whilst outputs are controlled by the system itself. For exemple, $p = \overline{news}.(\overline{pub} + \overline{movie})$ and $q = \overline{news}.\overline{pub} + \overline{news}.\overline{movie}$ are distinguished by bisimulations. In a point-to-point model, an observer could put them apart: initially, it provide a $news$ action, and then, deppending on the evolution of q, it provide pub or $movie$, thus exhibiting a diffrence between p and q. In a broadcasting model (where outputs are non-blocking), these two processes cannot be distinguished, since the actions of p and q do not deppend, no more, of an extern observer (all what we can, is listening and trying to send messages; the system itself, will be forced to accept a message only if it could not continue to evolve autonomously). Intuitively, if we watch TV, we cannot know (nor influence) before (or during) the $news$, the decision of the editor to send afterwards the pub or the $movie$. This example justifies why bisimulations are too restrictive (at least as they are defined) for broadcasting systems in certain cases.

The aim of this paper is to develop a theory of testing for broadcasting systems. Such studies were made just in the framework of point-to-point processes algebra. Two systems are equivalent whenever they satisfy the same set of observers. Depending on the choice for the universe of observers or for the notion of satisfaction, we obtain several pre-orders induced by tests. Let S be a system defined over a set of actions $\mathcal{A}$; an observer for the system S is a process O defined over the set of actions $\mathcal{A} \cup \{\bar{\omega}\}$, where $\bar{\omega}$ is a new action used by the observer O to report the success of its observation. For an observer O, a system S *must* satisfy it (denoted S $\underline{must}$ O), if any execution in parallel of S and O pass by a state such that O can report a success (can made an output $\bar{\omega}$); a system S *may* satisfy an observer O (S $\underline{may}$ O), if there is at least an execution in parallel of S and O that pass by a state such that O can report a success (can made an output $\bar{\omega}$). $\underline{must}$ and $\underline{may}$ induce two preorders on the set of systems denoted $\ll_{must}$ and $\ll_{may}$. These preorders play a significant role in the methodologies used for the description and verification of systems: they allow to establish the conformance of an implementation with respect to a specification. For a specification *Spec*, the set of behaviours that an implementation must enjoy is modeled by the set of observers that *Spec must* satisfy. The set of erroneous behaviours is modeled by the set of observers that *Spec may* not satisfy. Hence, an implementation *Impl* is correct with respect a specification *Spec* iff $Spec \ll_{must} Impl$ and $Impl \ll_{may} Spec$.

The rest of the paper is as follows. In section 2 we remind the *CBS* without message passing and the $b\pi$-calculus. Section 3 presents the definitions of *may testing* and *must testing* in the framework of broadcasting calculi. In section 4 we give a direct characterization for these pre-orders. We conclude by related works and some continuations.

2 Broadcasting calculi

2.1 $b\pi$-calculus

In this subsection, we briefly remind $b\pi$-calculus (more details and examples which illustrate the expressiveness of the model can be found in [6] and [7]).

The $b\pi$-calculus is a process calculus in which broadcast is the fundamental communication paradigm. It is derived from the broadcast calculus proposed by Prasad [17], and the π-calculus proposed by Milner, Parrow and Walker [15]. It differs from the broadcast calculus, in that communications are made on channels or ports (and transmitted values are channels too), and from the π-calculus in the manner the channels are used: for broadcast communications only. Let Ch_b be a countable set of *channels*. Processes are defined by the grammar of Table 1.

$\mathcal{P}_b \ni p ::= \quad nil \mid \pi.p \mid \nu xp \mid \langle x = y\rangle p, q \mid p_1 + p_2 \mid p_1 \parallel p_2 \mid A\langle\tilde{x}\rangle \mid (rec\ A\langle\tilde{x}\rangle.p)\langle\tilde{y}\rangle$
where π belongs to the set of prefixes $\pi ::= x(\tilde{y}) \mid \bar{x}\tilde{y} \mid \tau$, and $\tilde{x}, \tilde{y} \subseteq Ch_b,\ x, \in Ch_b$.

Table 1. Processes in $b\pi$-calculus

Prefixes denote the basic actions of processes: τ is a silent action (which corresponds to an internal transition), $x(\tilde{y})$ is the *input* of the names $\tilde{y}$ on the channel x, and $\bar{x}\tilde{y}$ is the *output* of the names $\tilde{y}$ on the channel x. nil is a process which does nothing. $\pi.p$ is the process which realize the action denoted by π and next behaves like p. $p_1 + p_2$ denotes choice, it behaves like p_1 or p_2. νxp is the creation of a new local channel x (whose initial scope is the process p). $\langle x = y\rangle p_1, p_2$ is a process which behaves like p_1 or p_2 depending on the relation between x and y. $p_1 \parallel p_2$ is the parallel composition of p_1 and p_2. X is a process identifier whose arity is satisfied by $\langle\tilde{x}\rangle$ and $(recX\langle\tilde{x}\rangle.p)\langle\tilde{y}\rangle$ is a recursive process (this allows to represent processes with infinite behaviour), with $\tilde{x}$ containing all the free names which appear in p. In this article, we assume that X occurs guarded in any recursive definition (underneath a prefix).

The operators νx and $y(\tilde{x})$, are $x - binders$, i.e. in νxp and $y(\tilde{x}).p$, x and $\tilde{x}$ are bound, and $bn(p)$ denotes the set of bound names of p. The free names of p are those that do not occur in the scope of any binder, and are denoted by $fn(p)$. The set of names of p is denoted by $n(p)$. *Alpha-conversion* is defined as usual.

Definition 1. *The set of* **actions** *denoted $\mathcal{A}ct$ and ranged over by α,β is defined by the following grammar:*

$$\alpha ::= a\langle\tilde{x}\rangle \ \mid \ \nu\tilde{y}\bar{a}\tilde{x} \ \mid \ \tau \ \mid \ a :$$

where $a, x \in Ch_b$, $\tilde{x}, \tilde{y} \subseteq Ch_b$. An action is either a reception, a (possibly bound) output, or the silent action τ, denoting an internal transition. In $a\langle\tilde{x}\rangle$ and $\nu\tilde{y}\bar{a}\tilde{x}$, a is the subject of the communication and $\tilde{x}$ is its object. By extension $n(\alpha)$ ($fn(\alpha)$, $bn(\alpha)$) denotes the names (respectively free names, bound names) used in the action α (($fn(\tau) = \emptyset$, $fn(a\langle\tilde{x}\rangle) = \{a\} \cup \tilde{x}$, $fn(\nu\tilde{y}\bar{a}\tilde{x}) = \{a\} \cup \tilde{x} \setminus \tilde{y}$, $fn(a :) = \{a\}$, $bn(\tau) = \emptyset$, $bn(a\langle\tilde{x}\rangle) = \emptyset$, $bn(\nu\tilde{y}\bar{a}\tilde{x}) = \tilde{y}$, $bn(a :) = \emptyset$, $n(\alpha) = fn(\alpha) \cup bn(\alpha)$).

We give an operational semantics for our calculus in terms of transitions over the set $\mathcal{P}_b$ of processes. Before, we define, similarly to [16], a relation $\longrightarrow \subseteq \mathcal{P}_b \times Ch_b$ denoted $p \xrightarrow{a:}$ and which can be read "p discards all outputs made on the channel a " (see Table 2).

(1) $\dfrac{}{nil \xrightarrow{a:}}$	(2) $\dfrac{}{\tau.p \xrightarrow{a:}}$	(3) $\dfrac{}{\bar{b}\tilde{y}.p \xrightarrow{a:}}$
(4) $\dfrac{b \neq a}{b(\tilde{x}).p \xrightarrow{a:}}$	(5) $\dfrac{p \xrightarrow{a:} \ \vee \ x = a}{\nu x p \xrightarrow{a:}}$	(6) $\dfrac{p_1 \xrightarrow{a:} \wedge p_2 \xrightarrow{a:}}{p_1 + p_2 \xrightarrow{a:}}$
(7) $\dfrac{p_1 \xrightarrow{a:}}{\langle x = x\rangle p_1, p_2 \xrightarrow{a:}}$	(8) $\dfrac{x \neq y \wedge \ p_2 \xrightarrow{a:}}{\langle x = y\rangle p_1, p_2 \xrightarrow{a:}}$	
(9) $\dfrac{p_1 \xrightarrow{a:} \ \wedge \ p_2 \xrightarrow{a:}}{p_1 \| p_2 \xrightarrow{a:}}$	(10) $\dfrac{p[(rec\ X\langle\tilde{x}\rangle.p)/X, \tilde{y}/\tilde{x}] \xrightarrow{a:}}{(rec\ X\langle\tilde{x}\rangle.p)\langle\tilde{y}\rangle \xrightarrow{a:}}$	

Table 2. The "discard" relation

Intuitively, a process ignores all the communications made on the channels it is not listening. nil, $\tau.p$ or $\bar{b}\tilde{y}.p$ discard any communication. A process waiting for a message on a channel b, discards actions on the other channels a with $a \neq b$. In rule (5) a the condition $x = a$ expres the possibility that a does not occur free in p. Rules (6) to (10) follow the structure of the term.

To simplify the presentation, we extend sub to τ and we denote $sub(\tau) = obj(\tau) = \tau$, and $p \xrightarrow{\tau:}$ for any process p. Also, we denote by $Ch_b : \stackrel{def}{=} \{a : \mid a \in Ch_b\}$.

Definition 2. Transition system *The operational semantics of $b\pi$-calculus is defined as a labeled transition system defined over the set $\mathcal{P}_b$ of processes. The judgement $p \xrightarrow{\alpha} p'$ means that the process p is able to perform action α and to evolve next to p'. The operational semantics is given in Table 3 (we omitted the symmetric versions of rules* (7)*,* (12) *and* (13)*).*

A communication between processes is performed through unbuffered broadcast. Compared to π-calculus, outputs are non-blocking, i.e. there is no need of a receiving process. One of the processes broadcasts an output and the remaining processes either receive or ignore the sending, according to whether they are "listening" or not on the channel which serves as support for the output. A process which "listens" to a channel a, cannot ignore any value sent on this channel.

$$(1)\ \frac{}{\tau.p \xrightarrow{\tau} p} \qquad (2)\ \frac{}{a(\tilde{x}).p \xrightarrow{a\langle\tilde{z}\rangle} p[\tilde{z}/\tilde{x}]} \qquad (3)\ \frac{}{\bar{a}\tilde{x}.p \xrightarrow{\bar{a}\tilde{x}} p}$$

$$(4)\ \frac{p \xrightarrow{\nu\tilde{y}\bar{a}\tilde{x}} p' \ \wedge\ z \in \tilde{x} \setminus \{a, \tilde{y}\} \ \wedge\ w \notin fn(\nu z p')}{\nu z p \xrightarrow{\nu w \nu \tilde{y} \bar{a}(\tilde{x}[w/z])} p'[w/z]} \qquad (5)\ \frac{p \xrightarrow{\nu\tilde{y}\bar{a}\tilde{x}} p'}{\nu a p \xrightarrow{\tau} \nu a \nu \tilde{y} p'} \qquad (6)\ \frac{p \xrightarrow{\alpha} p' \ \wedge\ x \notin n(\alpha)}{\nu x p \xrightarrow{\alpha} \nu x p'}$$

$$(7)\ \frac{p_1 \xrightarrow{\alpha} p' \ \wedge\ bn(\alpha) \cap fn(p_2) = \emptyset}{p_1 + p_2 \xrightarrow{\alpha} p'} \qquad (8)\ \frac{p_1 \xrightarrow{\alpha} p' \ \wedge\ bn(\alpha) \cap fn(p_2) = \emptyset}{\langle x = x \rangle p_1, p_2 \xrightarrow{\alpha} p'}$$

$$(9)\ \frac{x \neq y \ \wedge\ p_2 \xrightarrow{\alpha} p' \ \wedge\ bn(\alpha) \cap fn(p_2) = \emptyset}{\langle x = y \rangle p_1, p_2 \xrightarrow{\alpha} p'} \qquad (10)\ \frac{p[(rec\ X\langle\tilde{x}\rangle.p)/X, \tilde{y}/\tilde{x}] \xrightarrow{\alpha} p' \ \wedge\ \alpha \notin Ch_b:}{(rec\ X\langle\tilde{x}\rangle.p)\langle\tilde{y}\rangle \xrightarrow{\alpha} p'} \qquad (11)\ \frac{p_1 \xrightarrow{a\langle\tilde{x}\rangle} p_1' \ \wedge\ p_2 \xrightarrow{a\langle\tilde{x}\rangle} p_2'}{p_1 \| p_2 \xrightarrow{a\langle\tilde{x}\rangle} p_1' \| p_2'}$$

$$(12)\ \frac{p_1 \xrightarrow{\nu\tilde{y}\bar{a}\tilde{x}} p_1' \ \wedge\ p_2 \xrightarrow{a\langle\tilde{x}\rangle} p_2' \ \wedge\ \tilde{y} \cap fn(p_2) = \emptyset}{p_1 \| p_2 \xrightarrow{\nu\tilde{y}\bar{a}\tilde{x}} p_1' \| p_2'} \qquad (13)\ \frac{p_1 \xrightarrow{\alpha} p_1' \ \wedge\ bn(\alpha) \cap fn(p_2) = \emptyset \ \wedge\ p_2 \xrightarrow{sub(\alpha):}}{p_1 \| p_2 \xrightarrow{\alpha} p_1' \| p_2}$$

Table 3. Operational semantics of $b\pi$-calculus

The operational semantics is an early one, i.e. the bound names of an input are instantiated as soon as possible, in the rule for input. Rule (1) allows to identify process which are alpha - convertible. Rules (1) to (3) are straightforward and they have the same signification as in π-calculus: describe the initial action an process can do. Rule (4) states that when a local channel name is emitted, the related output has to be bound. Rule (5) does not exist in π -calculus; if a is a channel local to p, then all communications made on a are hidden for the environment; in addition, this rule establish the scope of the names exported on the channel a: the new scope of $\tilde{y}$ will be extended by the processes which were listening on a. Rules (6) to (10) have the same meaning as in π -calculus; condition $bn(\alpha) \cap fn(p_2) = \emptyset$ assure that a process cannot send a free name as a new local name. In rule (10), condition $\alpha \notin Ch_b$: assure that no processes can change of state as the result of a discard. Rules (11) and (12) are specific to broadcast; the same message can be received by more processes in a single communication. In rule (13), a process which does not listen on a channel a, remains unchanged during a communication made on this channel.

As usual we shall use the following notations:

- $\stackrel{\epsilon}{\Longrightarrow} \stackrel{def}{=} (\stackrel{\tau}{\longrightarrow})^*, \quad \stackrel{\alpha}{\Longrightarrow} \stackrel{def}{=} \stackrel{\epsilon}{\Longrightarrow} \stackrel{\alpha}{\longrightarrow} \stackrel{\epsilon}{\Longrightarrow}$, if $\alpha \neq \tau$,
- $\stackrel{\oslash}{\longrightarrow} \stackrel{def}{=} \bigcup \big\{ \stackrel{\alpha}{\longrightarrow} \big| \alpha\ is\ an\ output\ or\ \alpha = \tau \big\}, \quad \stackrel{\oslash}{\Longrightarrow} \stackrel{def}{=} (\stackrel{\oslash}{\longrightarrow})^*$.

We shall omit the trail nil and we shall also use the notation $\bar{a}(x)$ to stand for the bound output $\nu x \bar{a} x$.

2.2 CBS without message passing

CBS without message passing [16], is for $b\pi$-calculus what CCS is for π-calculus: broadcast remains the basic communications primitive, but this time processes synchronize by signals, i.e. values exchanged do not change the communication topology and are ignored. As CBS without message passing is similar and simpler than $b\pi$-calculus, we defer the syntax of processes and the rules for the operational semantics without any explanation to the appendix (for more details, the reader can consult [6] or [16]). All the notations are the same as for $b\pi$-calculus.

3 Preorders induced by tests

We adapt in this section the definitions of the preorders induced by tests from [4] in the framework of broadcasting processes: CBS and $b\pi$-calculus. From now on, as in [11], we shall use $p \xrightarrow{\alpha:} p$ instead of $p \xrightarrow{a:}$, if $sub(\alpha) = a$.

Definition 3. **Observers** *are processes that can make a distinguished output* $\bar{\omega}$ *with* $\omega \notin Ch_b$.

Action $\bar{\omega}$ can be interpreted as an action which allow to observers to announce their success. Observers interact with tested processes by exchanging messages.

Definition 4. *An* **execution** *of* p *is a sequence of transitions* $p = p_0 \xrightarrow{\oslash} p_1 \xrightarrow{\oslash} p_2 \xrightarrow{\oslash} \ldots \xrightarrow{\oslash} p_k \xrightarrow{\oslash} \ldots$ *which is either infinite, or* p_k *is deadlocked. The execution is successful if there exists* $n \geq 0$ *such that* $p_n \xrightarrow{\bar{\omega}}$.

In calculi where rendez-vous is the basic communication (as is the case with CCS or π-calculus), in definition 4, $\xrightarrow{\oslash}$ is replaced by $\xrightarrow{\tau}$. But in broadcasting calculi, a system (or process) evolves autonomously (without implicating the environment) using the relation $\xrightarrow{\oslash}$ instead of $\xrightarrow{\tau}$ (since outputs are non-blocking).

For any process p and any observer O, we define p $\underline{may}$ O, if and only if there exists a successful execution of $p \parallel O$.

Definition 5. **May testing**

Given two processes p *and* q*, we define* $p \ll_{may} q$ *if and only if for any observer* O*,* p $\underline{may}$ O *implies* q $\underline{may}$ O.

For any process p and any observer O, we define p $\underline{must}$ O, if and only if all executions of $p \parallel O$ are successful.

Definition 6. **May testing**

Given two processes p *and* q*, we define* $p \ll_{must} q$ *if and only if for any observer* O*,* p $\underline{must}$ O *implies* q $\underline{must}$ O.

In order to relate two processes using the definitions 5 and 6 we need to use a quantification on the universe of observers. This push us to search for a direct characterization of $\ll_{may}$ and $\ll_{must}$. Sections 4 and 5 are devoted to direct characterizations for $\ll_{may}$ and $\ll_{must}$ in CBS without message passing and $b\pi$-calculus.

4 Trace-based characterizations for CBS without message passing

A trace of a process is a sequence of actions that the process can do. An observer it is not sensible to intern changes of state. Hence, the occurrences of τ in traces are not significant. For a given trace t, all traces which differ only by some occurrences of τ satisfy the same set of observers. For this reason we consider a restricted set of actions $\mathcal{R}Act \stackrel{def}{=} \mathcal{A}ct \backslash \{\tau, \tau :\}$. Then, the set of traces Tr, is defined by the grammar: $t ::= \epsilon \mid \alpha.t$, where $\alpha \in \mathcal{R}Act$; ϵ represents the empty trace; in $\alpha.t$, the prefix α denotes the first action made by the process, while the suffix t is the trace that the process can make afterwards. We shall omit often the trail ϵ ($a_1.\ldots.a_n.\epsilon$ will be written $a_1.\ldots.a_n$). The length of a trace $t \in Tr$ is denoted by $|t|$. The set of traces of a process p, denoted by $tr(p)$ is: $tr(p) \stackrel{def}{=} \{w \in \mathcal{R}Act^* | \exists p' \text{ such that } p \stackrel{w}{\Longrightarrow} p'\}$. The set of prefixes $pre(s)$ of a trace $s = a_1.\ldots.a_n \in \mathcal{R}Act^*$, is defined by $pre(s) \stackrel{def}{=} \{\epsilon, a_1, a_1.a_2, \ldots, a_1.\ldots.a_n\}$.

If t is a trace of an observer O which leads it to a successful state, the set of "complementary traces" of t are the traces that allow to p to satisfy O. For a trace t, $Comp(t)$ is the set of traces defined by: $Comp : Tr \longrightarrow 2^{Tr}$

$$Comp(t) = \begin{cases} \{\epsilon\} & \text{if } t = \epsilon, \\ \{\bar{a}.s_1 \mid s_1 \in Comp(s)\} & \text{if } t = a : .s \text{ or } t = a.s,\ a \in Ch, \\ \{a : .s_1,\ a.s_1 \mid s_1 \in Comp(s)\} & \text{if } t = \bar{a}.s,\ a \in Ch. \end{cases} \tag{1}$$

$Comp$ will be extended to set of traces: if $M \subseteq Tr$, then $Comp(M) = \bigcup_{s \in M} Comp(s)$.

The following two results are very useful to prove the main theorems of this paper. Intuitively, if $s_i \in tr(p_i)$, then the "composition" of traces s_1 and s_2 provide an execution of $p_1 \parallel p_2$.

Lemma 1.
- *Let $p_1, p_2 \in \mathcal{P}_b$ be two processes such that $p_1 \parallel p_2 \stackrel{\oslash}{\Longrightarrow} r$. Then, there exist two processes q_1, q_2 and two traces $s_i \in tr(p_i), i = 1, 2$, such that $r = q_1 \parallel q_2$, $p_i \stackrel{s_i}{\Longrightarrow} q_i$, $i = 1, 2$ and $s_1 \in Comp(s_2)$.*
- *Conversely, if $s_i \in tr(p_i), i = 1, 2$, such that $r = q_1 \parallel q_2$, $p_i \stackrel{s_i}{\Longrightarrow} q_i$, $i = 1, 2$ and $s_1 \in Comp(s_2)$, then $p_1 \parallel p_2 \stackrel{\oslash}{\Longrightarrow} r$.*

Proof

The first implication is proved by induction on the length of the derivation $p_1 \parallel p_2 \stackrel{\oslash}{\Longrightarrow} r$. The converse is proved by induction on $|s_2|$. □ *Lemma* 1

Lemma 2. *For any traces s_0, s_1 and s_2, if $s_0 \in Comp(Comp(s_1))$ and $s_2 \in Comp(s_1)$, then $s_2 \in Comp(s_0)$.*

Proof

By induction on $|s_1|$. □ *Lemma* 2

4.1 May testing

For an observer O, p <u>may</u> O if there is a trace s of p that "satisfy" O (that is complementary to a trace t of O). Intuitively, we should have $p \ll_{may} q$ if for any trace s of p, q can exhibit a trace s' which is equivalent from an observational point of view. This intuition is reflected by the following trace-based characterization of $\ll_{may}$.

Theorem 1. *For all processes p and q,* $\quad p \ll_{may} q \quad$ *iff* $\quad tr(p) \subseteq Comp(Comp(tr(q)))$.

Proof

" $\Longrightarrow$ " Firstly, we define a special set of observers. If $M \subseteq Ch_b$ is a finite set of channels, we shall denote by

$$in(M) \stackrel{not}{=} \begin{cases} nil & \text{if } M = \emptyset, \\ a.nil + in(M \setminus \{a\}) & \text{otherwise, with } a \in M. \end{cases} \tag{2}$$

Let t be a trace and let $M \subseteq Ch_b$ be a set of channels. Let $o_M(t)$ be an observer defined by

$$o_M(t) = \begin{cases} \bar{\omega}.nil & \text{if } t = \epsilon, \\ \bar{a}.o_M(s) + in(M) & \text{if } t = a : .s \text{ or } t = a.s,\ a \in Ch, \\ a.o_M(s) + in(M) & \text{if } t = \bar{a}.s,\ \ a \in Ch. \end{cases} \tag{3}$$

Let $s \in tr(p)$ and let $M = fn(p, q)$. Then, using the definition of $o_M(s)$, there exists $s_2 \in tr(o_M(s))$ such that $p \stackrel{s}{\Longrightarrow} p'$, $o_M(s) \stackrel{s_2}{\Longrightarrow} \bar{\omega}.nil$, and $s \in Comp(s_2)$. $p \ll_{may} q$ implies $q\ \underline{may}\ o_M(s)$, i.e. $q \parallel o_M(s) \stackrel{\oslash}{\Longrightarrow} q' \parallel \bar{\omega}.nil$. Lemma 1, assure the existence of $s_0" \in tr(q)$ and $s_2" \in tr(o_M(s))$ such that $s_2" \in Comp(s_0")$ and $o_M(s) \stackrel{s_2"}{\Longrightarrow} \bar{\omega}.nil$. By construction, $s_2 \in pre(s_2")$ and there exists a prefix $s_0' \in pre(s_0")$ such that $s_2 \in Comp(s_0')$ (the expression $in(M)$ in the definition of $o_M(s)$ assure that the observed process do not make "wrong outputs"). $s \in Comp(s_2)$ and $s_2 \in Comp(s_0')$ imply $s \in Comp(Comp(s_0'))$. Since $s_0" \in tr(q)$ and $s_0' \in pre(s_0")$ we get $s_0' \in tr(q)$. Hence $s \in Comp(Comp(s_0')) \subseteq Comp(Comp(tr(q)))$.

" $\Longleftarrow$ " Let be $p\ \underline{may}\ o$. Then, there exist s_1, s_2 with $s_1 \in Comp(s_2)$ such that $p \stackrel{s_1}{\Longrightarrow} p'$, $o \stackrel{s_2}{\Longrightarrow} o'$ and $o' \stackrel{\bar{\omega}}{\longrightarrow}$. $tr(p) \subseteq Comp(Comp(tr(q)))$ implies that there exists $s_0 \in tr(q)$ such that $s_1 \in Comp(Comp(s_0))$, and as $Comp$ is symmetrical, we obtain $s_0 \in Comp(Comp(s_1))$. Since $s_0 \in Comp(Comp(s_1))$ and $s_1 \in Comp(s_2)$, using Lemma 2, we get $s_0 \in Comp(s_2)$ (using that $Comp$ is symmetrical). From $o \stackrel{s_2}{\Longrightarrow} o'$, $s_0 \in Comp(s_2)$, and $s_0 \in tr(q)$, using Lemma 1 we obtain that there exists q' such that $q \parallel o \stackrel{\oslash}{\Longrightarrow} q' \parallel o'$. Since $o' \stackrel{\bar{\omega}}{\longrightarrow}$, we get $q\ \underline{may}\ o$. $\Box$ *Theorem* 1

4.2 Must testing

The characterization of "may testing" is intuitive and looks like the characterization for CCS ([4]); the difference is that for $p \ll_{may} q$ we do not require $tr(p) \subseteq tr(q)$, but that any trace s of p must be simulate by a trace $s' \in Comp(Comp(s))$ that is equivalent from the visibility point of view (remind that in CBS we cannot see directly whenever a process receive or discards a message). On the contrary, the trace-based characterization of "must testing" is much less evident; more, it is completely different w.r.t. to point-to-point calculi (CCS or other versions). The fact that outputs are autonomous, changes a lot the results for CBS.

We say that p "diverges" (denoted $p \Uparrow^{\oslash}$) if there exists an infinite execution $p = p_0 \stackrel{\oslash}{\longrightarrow} p_1 \stackrel{\oslash}{\longrightarrow} \dots p_k \stackrel{\oslash}{\longrightarrow} \dots$ such that $p_k \stackrel{\bar{\omega}}{\not\longrightarrow}$ for any $k \in \mathbb{N}$. On the contrary, p "converges" (denoted $p \Downarrow^{\oslash}$) iff $\neg p \Uparrow^{\oslash}$. Following [4], we shall extend the convergence predicate to traces: for any $s \in \mathcal{R}Act$, $p \Downarrow^{\oslash}_s$ iff

- $p \Downarrow^{\oslash}_{\epsilon}$ if $p \Downarrow^{\oslash}$;
- $p \Downarrow^{\oslash}_{\alpha.s}$ if $p \Downarrow^{\oslash}$ and whenever $p \stackrel{\alpha}{\Longrightarrow} p'$, then $p' \Downarrow^{\oslash}_s$.

We can prove that $p \Downarrow_{s}^{\oslash}$ iff for any prefix $s_1 \in pre(s)$ it holds $p \Downarrow_{s_1}^{\oslash}$. We denote $Conv(p, s)$ iff $\forall s' \in Comp(Comp(s))$, $p \Downarrow_{s'}^{\oslash}$.

$\mathcal{S}tab(p, s)$ denote the set of processes p' that are accessible from p by a trace s' "equivalent" with s and which can not evolve autonomously afterwards.

$$\mathcal{S}tab(p, s) \stackrel{def}{=} \{p' \mid p \stackrel{s'}{\Longrightarrow} p', s' \in Comp(Comp(s)), p \stackrel{\oslash}{\not\longrightarrow}\}.$$

The trace-based characterization of "must testing" is given by the relation $\leq$.

Definition 7. (alternate characterization of $\ll_{must}$) *$p \leq q$ iff $\forall s \in \mathcal{R}Act^*$, $Conv(p, s)$ implies*

- *$Conv(q, s)$;*
- *$\mathcal{S}tab(q, s) \neq \emptyset$ implies $\mathcal{S}tab(p, s) \neq \emptyset$.*

Before to prove the main theorem of this subsection, we need two auxiliary results.

Lemma 3. *If $p \ll_{must} q$, then $\forall s \in \mathcal{R}Act^*$, $Conv(p, s)$ implies*

1. *$Conv(q, s)$;*
2. *$Comp(Comp(s)) \cap tr(q) \neq \emptyset$ implies $Comp(Comp(s)) \cap tr(p) \neq \emptyset$.*

Lemma 4. *If $p \leq q$, and $Conv(p, s)$ then*
$Comp(Comp(s)) \cap tr(q) \neq \emptyset$ implies $Comp(Comp(s)) \cap tr(p) \neq \emptyset$.

Now, we prove the equivalence of $\ll_{must}$ and $\leq$.

Theorem 2. *For all processes p and q, $\quad p \ll_{must} q \quad$ iff $\quad p \leq q$.*

Proof

" $\Longrightarrow$ " We prove that $p \not\leq q$ implies $p \not\ll_{must} q$. Let suppose $p \not\leq q$ and $p \ll_{must} q$.
$p \not\leq q$ implies that there exists $s \in \mathcal{R}Act^*$, such that $Conv(p, s)$ and $[\neg Conv(q, s)$ or $(\mathcal{S}tab(q, s) \neq \emptyset$ and $\mathcal{S}tab(p, s) = \emptyset)]$.

- Let suppose that $Conv(p, s)$ and $\neg Conv(q, s)$.
  Since $p \ll_{must} q$ and $Conv(p, s)$, using Lemma 3 we obtain $Conv(q, s)$, contradiction.
- Let suppose that $Conv(p, s)$, $Conv(q, s)$, $\mathcal{S}tab(q, s) \neq \emptyset$ and $\mathcal{S}tab(p, s) = \emptyset$.
  For a finite set of channels $X \in Ch_b$, let $in'(X)$ and $o_3(t, X)$ be two processes defined as follows:

$$in'(X) \stackrel{not}{=} \begin{cases} nil & \text{if } X = \emptyset, \\ a.\bar{\omega} + in'(X \setminus \{a\}) & \text{otherwise, with } a \in X. \end{cases} \tag{4}$$

$$o_3(t, X) = \begin{cases} in'(X) & \text{if } t = \epsilon, \\ \tau.\bar{\omega} + \bar{a}.o_3(s, X) + in'(X) & \text{if } t = a:.s \text{ or } t = a.s,\ a \in Ch_b, \\ \tau.\bar{\omega} + a.o_3(s, X) + in'(X) & \text{if } t = \bar{a}.s,\ a \in Ch_b. \end{cases} \tag{5}$$

  $\mathcal{S}tab(q, s) \neq \emptyset$ implies that there exist q_1 and $s_1 \in Comp(Comp(s)) \cap tr(q)$ such that $q \stackrel{s_1}{\Longrightarrow} q_1$ and $q_1 \stackrel{\oslash}{\not\longrightarrow}$.
  By construction (and using Lemma 1) we have $q \parallel o_3(s, fn(p, q)) \stackrel{\oslash}{\Longrightarrow} q_1 \parallel in'(X)$ (without emitting on the channel ω) and as $q_1 \stackrel{\oslash}{\not\longrightarrow}$ we obtain $q\ \underline{m\!\!/ust}\ o_3(s, fn(p, q))$.
  Since $\mathcal{S}tab(p, s) = \emptyset$, we can prove $p\ \underline{must}\ o_3(s, fn(p, q))$, and hence $p \not\ll_{must} q$.

" $\Longleftarrow$ " We shall prove that $p \not\ll_{must} q$ implies $p \not\leq q$. Let suppose that $p \not\ll_{must} q$ and $p \leq q$. $p \not\ll_{must} q$ implies that there exists o such that p $\underline{must}$ o and q $\underline{m\!\!\!/ust}$ o.
Since q $\underline{m\!\!\!/ust}$ o, one of following cases must hold.

- $q \parallel o = q_0 \parallel o_0 \xrightarrow{\emptyset} q_1 \parallel o_1 \xrightarrow{\emptyset} \ldots \xrightarrow{\emptyset} q_n \parallel o_n$ and $q_n \parallel o_n \stackrel{\emptyset}{\not\longrightarrow}$, and for any $i = 0, n$, $o_i \stackrel{\bar{\omega}}{\not\longrightarrow}$.
  If we consider the contributions of q and o to this execution, by Lemme 1 we obtain that there exist $s \in \mathcal{R}Act^*$ and $t \in Comp(s)$ such that $q \stackrel{s}{\Longrightarrow} q_n$ and $o \stackrel{t}{\Longrightarrow} o_n$.
  In addition, since $q_n \parallel o_n \stackrel{\emptyset}{\not\longrightarrow}$ we obtain $o_n \stackrel{\emptyset}{\not\longrightarrow}$ and $q_n \stackrel{\emptyset}{\not\longrightarrow}$ and thus, $\mathcal{S}tab(q, s) \neq \emptyset$.
  p $\underline{must}$ o, $o \stackrel{t}{\Longrightarrow} o_n$, for any $i = 0, n$, $o_i \stackrel{\bar{\omega}}{\not\longrightarrow}$ and $t \in Comp(s)$ imply $Conv(p, s)$.
  From the definition 7, $\mathcal{S}tab(q, s) \neq \emptyset$, $p \leq q$, and $Conv(p, s)$ involve $\mathcal{S}tab(p, s) \neq \emptyset$, and so there exist p_n and $u \in Comp(Comp(s))$ such that $p \stackrel{u}{\Longrightarrow} p_n$ and $p_n \stackrel{\emptyset}{\not\longrightarrow}$.
  From $u \in Comp(Comp(s))$ and $t \in Comp(s)$, using the Lemma 2 we get $u \in Comp(t)$, and by the Lemma 1 we can build the following execution
  $p \parallel o = p_0 \parallel o_0 \xrightarrow{\emptyset} p_1 \parallel o_1 \xrightarrow{\emptyset} \ldots \xrightarrow{\emptyset} p_n \parallel o_n$ and $p_n \parallel o_n \stackrel{\emptyset}{\not\longrightarrow}$, so p $\underline{m\!\!\!/ust}$ o, contradiction.
- $q \parallel o = q_0 \parallel o_0 \xrightarrow{\emptyset} q_1 \parallel o_1 \xrightarrow{\emptyset} \ldots \xrightarrow{\emptyset} q_n \parallel o_n$, for any $i = 0, n$, $o_i \stackrel{\bar{\omega}}{\not\longrightarrow}$ and
  $$(q_n \Uparrow^{\emptyset} \text{ or } o_n \Uparrow^{\emptyset}). \tag{6}$$
  If we consider the contributions of q and o to this execution, using the Lemma 1 we obtain that there exist $s \in \mathcal{R}Act^*$ and $t \in Comp(s)$ such that $q \stackrel{s}{\Longrightarrow} q_n$ and $o \stackrel{t}{\Longrightarrow} o_n$.
  p $\underline{must}$ o, $o \stackrel{t}{\Longrightarrow} o_n$, for any $i = 0, n$, $o_i \stackrel{\bar{\omega}}{\not\longrightarrow}$ and $t \in Comp(s)$ imply $Conv(p, s)$.
  From the definition 7, $p \leq q$, and $Conv(p, s)$ involve $Conv(q, s)$, and so
  $$q_n \Downarrow^{\emptyset}. \tag{7}$$
  $q \stackrel{s}{\Longrightarrow} q_n$ implies $s \in tr(q)$, and since $p \leq q$, and $Conv(p, s)$, using the Lemma 4 we obtain that there exist p_n and $u \in Comp(Comp(s))$ such that $p \stackrel{u}{\Longrightarrow} p_n$.
  From $u \in Comp(Comp(s))$ and $t \in Comp(s)$, using the Lemma 2 we obtain $u \in Comp(t)$, and using the Lemma 1 we can build the following execution
  $p \parallel o = p_0 \parallel o_0 \xrightarrow{\emptyset} p_1 \parallel o_1 \xrightarrow{\emptyset} \ldots \xrightarrow{\emptyset} p_n \parallel o_n$, such that for any $i = 0, n$, $o_i \stackrel{\bar{\omega}}{\not\longrightarrow}$.
  Since p $\underline{must}$ o, we obtain $o_n \Downarrow^{\emptyset}$, contradiction with 6 and 7.
- $q \parallel o = q_0 \parallel o_0 \xrightarrow{\emptyset} q_1 \parallel o_1 \xrightarrow{\emptyset} \ldots \xrightarrow{\emptyset} q_n \parallel o_n \xrightarrow{\emptyset} \ldots$, and for any $i \in \mathbb{N}$, $o_i \stackrel{\bar{\omega}}{\not\longrightarrow}$.
  If we consider the contributions of q and o to this execution, using the Lemma 1 we obtain that there exist "infinite traces" s and t such that for any $k \in \mathbb{N}$, $s(k) \in \mathcal{R}Act^*$, $t(k) \in Comp(s(k))$, $q \stackrel{s(k)}{\Longrightarrow} q_k$ and $o \stackrel{t(k)}{\Longrightarrow} o_k$.
  Since p $\underline{must}$ o, using the Lemma 1 we infer that there exists $n \in \mathbb{N}$ such that
  $$tr(p) \cap Comp(Comp(s(n))) = \emptyset. \tag{8}$$
  In addition, since $o \stackrel{t(n)}{\Longrightarrow}$, $t(n) \in Comp(s(n))$ and $i \in \mathbb{N}$, $o_i \stackrel{\bar{\omega}}{\not\longrightarrow}$ we obtain $Conv(p, s(n))$ (otherwise, we could build a "divergent" execution from $p \parallel o$).
  $q \stackrel{s(n)}{\Longrightarrow} q_n$ implies $tr(q) \cap Comp(Comp(s(n))) \neq \emptyset$. In addition we have $p \leq q$, and $Conv(p, s(n))$, thus, using the Lemma 4 we obtain $tr(p) \cap Comp(Comp(s(n))) \neq \emptyset$, contradiction with 8. $\square$ *Theorem* 2

5 Trace-based characterizations for $b\pi$-calculus

In this section, we shall extend to monadic $b\pi$-calculus the results presented in the previous section. Most of the notations remain unchanged; we shall give explicitly only the differences.

Let $\xrightarrow{\alpha}_1$ be the relation defined in the Table 4.

(1) $\frac{p \xrightarrow{\alpha} p'}{p \xrightarrow{\alpha}_1 p'}$	(2) $\frac{p \xrightarrow{a\langle x\rangle} p' \ \wedge\ x \notin fn(p)}{p \xrightarrow{a(x)}_1 p'}$	(3) $\frac{p \xrightarrow{a:} \ \wedge\ x \notin fn(p)}{p \xrightarrow{a(x):}_1 p}$

Table 4. The relation $\xrightarrow{\alpha}_1$

The set of traces Tr is defined by the grammar: $t ::= \epsilon \ \mid \ \alpha.t$, where $\alpha \in \mathcal{E}Act = \{\bar{a}x, \bar{a}(x), a\langle x\rangle, a(x), a:, a(x): | a, x \in Ch_b\}$. The set of traces of a process p, denoted by $tr(p)$ is: $tr(p) \stackrel{def}{=} \{w \in \mathcal{E}Act^* | \exists p' \text{ such that } p \stackrel{w}{\Longrightarrow}_1 p'\}$.

$a(x)$, $a(x):$ and $\bar{a}(x)$ bind the occurrences of x in the traces. We will call an action or a trace "harmless" with respect to a particular statement, if the bound names (channels) are different than any of the name appearing in the rest of the statement.

Next lemma admits a similar proof to Proposition 2.7 from [10].

Lemma 5. *If $p \equiv_\alpha q$ and $p \stackrel{s}{\Longrightarrow}_1 p'$, for any harmless trace s' such that $s \equiv_\alpha s'$, there exists q' such that $q \stackrel{s'}{\Longrightarrow}_1 q'$, $p' \equiv_\alpha q'[bn(s)/bn(s')]$ and $q' \equiv_\alpha p'[bn(s')/bn(s)]$.*

For a trace t, the set of complementary traces $Comp(t)$ is defined as follows: $Comp : Tr \longrightarrow 2^{Tr}$

$$Comp(t) = \begin{cases} \{\epsilon\} & \text{if } t = \epsilon, \\ \{\bar{a}x.s_1 \mid s_1 \in Comp(s)\} & \text{if } t = a:.s \text{ or } t = a\langle x\rangle.s,\ a, x \in Ch, \\ \{\bar{a}(x).s_1 \mid s_1 \in Comp(s)\} & \text{if } t = a(x):.s \text{ or } t = a(x).s,\ a, x \in Ch, \\ \{a:.s_1,\ a\langle x\rangle.s_1 \mid s_1 \in Comp(s)\} & \text{if } t = \bar{a}x.s,\ \ a \in Ch, \\ \{a(x):.s_1,\ a(x).s_1 \mid s_1 \in Comp(s)\} & \text{if } t = \bar{a}(x).s,\ \ a \in Ch. \end{cases} \tag{9}$$

Lemma 1 and Lemma 2 remain true for $b\pi$-calculus, and they allow us to restrict in the Theorem 3 just to harmless traces (for any successful execution (or unsuccessful execution) of $p \parallel o$, there exists a successful execution (respectively a unsuccessful execution) of $p \parallel o$ such that the contributions s of p and t of o are harmless).

The trace-based characterization of $\ll_{may}$ for $b\pi$-calculus looks like that one from CBS without message passing:

Theorem 3. *For all processes p and q, $\quad p \ll_{may} q \quad$ iff $\quad tr(p) \subseteq Comp(Comp(tr(q)))$.*

The alternate characterization of "must testing" for $b\pi$-calculus is a natural extension of that one from CBS. The definitions of $p \Uparrow^{\oslash}$ and $p \Downarrow^{\oslash}_s$ remain the same. In the definition of $Conv(p, s)$ we take also into account the alpha conversion. $Conv(p, s)$ iff $p \Downarrow^{\oslash}_{s'}$, $\forall s'$ such that $s' \equiv_\alpha s$" and $s" \in Comp(Comp(s))$.

Next lemma admits a similar proof to Lemma 3.4 from [10].

Lemma 6. *If* $bn(s) \cap fn(p) = \emptyset$ *then* $Conv(p, s)$ *iff* $p \Downarrow^{\oslash}_{s'}$, $\forall s'$ *such that* $s' \in Comp(Comp(s))$.

Lemmas 1, 5 and 6 allow us to restrict everywhere in the sequel just to harmless traces.

We will denote by $\mathcal{S}tab(p, s)$ the set of processes p', which are accessible from p through a trace s' ("equivalent" of s) and that cannot evolve autonomously afterwards.

$$\mathcal{S}tab(p, s) \overset{def}{=} \{p' \mid p \overset{s'}{\Longrightarrow} p', s' \equiv_\alpha s", s" \in Comp(Comp(s)), p \overset{\oslash}{\not\longrightarrow}\}.$$

The trace-based characterization of $\ll_{must}$ for $b\pi$-calculus is given by the relation $\leq$.

Definition 8. (alternate characterization of $\ll_{must}$) $p \leq q$ *iff* $\forall s \in \mathcal{E}Act^*$, $Conv(p, s)$ *implies*

- $Conv(q, s)$;
- $\mathcal{S}tab(q, s) \neq \emptyset$ *implies* $\mathcal{S}tab(p, s) \neq \emptyset$.

As in the Section 4.2, we can prove the next theorem.

Theorem 4. *For all processes* p *and* q, $\quad p \ll_{must} q \quad$ *iff* $\quad p \leq q$.

6 Related work and conclusions

In this paper, we presented trace-based characterizations for "may testing" and "must testing" for CBS without message passing and $b\pi$-calculus. These characterizations do not depend of observers. As far as the authors are aware, all testing theories in the literature are only for point-to-point process calculi.

The first works for synchronous point-to-point models, is the seminal paper [4] where Hennessy and De Nicola introduce the definitions of "may testing" and "must testing" and give characterizations for these preorders together with complete proof systems. For mobile point-to-point systems (π-calculus or other versions), characterizations are presented in [1] and [10]. These characterizations are based on inclusions between traces sets (for "may testing"), and on the so-called "acceptance sets" (for "must testing").

Recently, these results were extended to asynchronous models.

The preorders "may testing" and "must testing" have been studied for an asynchronous version of CCS (enriched with intern and external choice operators) called $TACCS$ [3]. The authors give afterwards an equational characterization of "must testing" for the finite fragment of $TACCS$ (without recursion).

In [2], the authors present characterizations of "may testing" and "must testing" for another asynchronous version of CCS called $ACCS$ (a more restrictive calculus w.r.t. $TACCS$ - the external choice can be applied only to prefixed processes). The characterization for "may testing" is afterwards generalized to asynchronous π-calculus. We emphasize that all these characterizations (given in point-to-point calculi) are very different w.r.t. characterizations given in this paper for broadcasting calculi.

As a possible continuation of our study, it remains to provide equational characterizations for the congruences induced by $\ll_{must}$ and $\ll_{may}$.

References

[1] M. Boreale and R. De Nicola. Testing Equivalence for Mobile Systems. In *Information and Computation*, volume 120, pages 279–303, 1995.

[2] M. Boreale and R. De Nicola and R. Pugliese. Trace and Testing Equivalence on Asynchronous Processes. In *Lecture Notes in Computer Science*, volume 1578, pages 165–179. Springer Verlag, 1999.

[3] I. Castellani and M. Hennessy, Testing Theories for Asynchronous Processes. In *Lecture Notes in Computer Science*, volume 1530, pages 90–108. Springer Verlag, 1998.

[4] R. de Nicola and M. Hennessy. Testing Equivalences for Processes. In *Theoretical Computer Science*, volume 34, pages 83–133, 1984.

[5] C. Ene and T. Muntean. Expressiveness of point-to-point versus broadcast communications. In *Fundamentals of Computation Theory, 12th International Symposium, Lecture Notes in Computer Science*, volume 1684. Springer Verlag, 1999.

[6] C. Ene. A formal model for broadcasting mobile systems. PhD thesis, Laboratoire d'Informatique de Marseille, 2001 (in french). Available at http://www.esil.univ-mrs.fr/ cene/teza.ps.gz

[7] C. Ene and T. Muntean. A broadcast-based calculus for communicating systems. Research Report. Extended version of [8], 2001. Available at http://www.esil.univ-mrs.fr/ cene/FMPPTA.ps.gz

[8] C. Ene and T. Muntean. A broadcast-based calculus for communicating systems. In *6th International Workshop on Formal Methods for Parallel Programming: Theory and Applications, San Francisco*, 2001.

[9] A. Geist, A. Beguelin, J. Dongarra, R. Manchek, W. Jiang, and V. Sunderam. *PVM: A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press, 1994.

[10] M. Hennessy. A model for the π-calculus. Report 8/91, School of Cognitive and Computer Science, University of Sussex, 1991.

[11] M. Hennessy and J. Rathke. Bisimulations for a calculus of broadcasting systems. In *CONCUR 95, Lecture Notes in Computer Science*, volume 962. Springer Verlag, 1995.

[12] C. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[13] N. Lynch and M. Tuttle. An introduction to input/output automata. Technical report, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands. Also, Technical Memo MIT/LCS/TM-373, Laboratory for Computer Science, Massachusetts Institute ofTechnology, 2000.

[14] R. Milner. *Communication and concurrency*. Prentice-Hall, 1989.

[15] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part I/II. *Journal of Information and Computation*, 100:1–77, 1992.

[16] K. Prasad. A calculus of broadcasting systems. In *In TAPSOFT'91, Volume 1: CAAP, Lecture Notes in Computer Science*, volume 493. Springer Verlag, 1991.

[17] K. V. S. Prasad. A calculus of broadcasting systems. *Science of Computer Programming*, 25, 1995.

7 Syntax and semantics of *CBS* without message passing

$p \quad ::= \qquad nil \;\mid\; \pi.p \;\mid\; \nu x p \;\mid\; p_1 + p_2 \;\mid\; p_1 \parallel p_2 \;\mid\; X \;\mid\; rec\ X.p$	
where π belongs to the set of prefixes $\pi ::= \tau \;\mid\; x \;\mid\; \bar{x}$, with $x, y \in Ch_b$.	

Table 5. Processes in CBS without message passing

(1) $\dfrac{}{nil \xrightarrow{a:}}$	(2) $\dfrac{}{\tau.p \xrightarrow{a:}}$	(3) $\dfrac{}{\bar{b}.p \xrightarrow{a:}}$	(4) $\dfrac{b \neq a}{b.p \xrightarrow{a:}}$
(5) $\dfrac{p \xrightarrow{a:} \vee x = a}{\nu x p \xrightarrow{a:}}$	(6) $\dfrac{p_1 \xrightarrow{a:} \wedge p_2 \xrightarrow{a:}}{p_1 + p_2 \xrightarrow{a:}}$	(7) $\dfrac{p_1 \xrightarrow{a:} \wedge\ p_2 \xrightarrow{a:}}{p_1 \parallel p_2 \xrightarrow{a:}}$	(8) $\dfrac{p[(rec\ X.p)/X] \xrightarrow{a:}}{rec\ X.p \xrightarrow{a:}}$

Table 6. The "discard" relation

(1) $\dfrac{}{\tau.p \xrightarrow{\tau} p}$	(2) $\dfrac{}{a.p \xrightarrow{a} p}$	(3) $\dfrac{}{\bar{a}.p \xrightarrow{\bar{a}} p}$	(4) $\dfrac{p \xrightarrow{\bar{a}} p'}{\nu a p \xrightarrow{\tau} \nu a p'}$
(5) $\dfrac{p \xrightarrow{\alpha} p' \ \wedge\ x \notin n(\alpha)}{\nu x p \xrightarrow{\alpha} \nu x p'}$	(6) $\dfrac{p_1 \xrightarrow{\alpha} p' \ \vee\ p_2 \xrightarrow{\alpha} p'}{p_1 + p_2 \xrightarrow{\alpha} p'}$	(7) $\dfrac{p_1 \xrightarrow{a} p_1' \ \wedge\ p_2 \xrightarrow{a} p_2'}{p_1 \parallel p_2 \xrightarrow{a} p_1' \parallel p_2'}$	
(8) $\dfrac{p_1 \xrightarrow{\bar{a}} p_1' \ \wedge\ p_2 \xrightarrow{a} p_2'}{p_1 \parallel p_2 \xrightarrow{\bar{a}} p_1' \parallel p_2'}$	(9) $\dfrac{p_1 \xrightarrow{\alpha} p_1' \ \wedge\ p_2 \xrightarrow{sub(\alpha):}}{p_1 \parallel p_2 \xrightarrow{\alpha} p_1' \parallel p_2}$	(10) $\dfrac{p[(rec\ X.p)/X] \xrightarrow{\alpha} p' \ \wedge\ \alpha \notin Ch_b:}{(rec\ X.p) \xrightarrow{\alpha} p'}$	

Table 7. Operational semantics of CBS without message passing

8 Proofs relative to Section 4

Remark 1.

- $s_1 \in Comp(s_2)$ implies $|s_1| = |s_2|$.
- The relation $Comp$ is symmetrical: $\forall s_1, s_2 \in Tr$ it holds ($s_1 \in Comp(s_2)$ if and only if $s_2 \in Comp(s_1)$).

  Proof By induction on $|s_2|$. $\square$ *Remark* 1

  Lemma 1

- Let $p_1, p_2 \in \mathcal{P}_b$ be two processes such that $p_1 \parallel p_2 \stackrel{\oslash}{\Longrightarrow} r$. Then, there exist two processes q_1, q_2 and two traces $s_i \in tr(p_i), i = 1, 2$, such that $r = q_1 \parallel q_2$, $p_i \stackrel{s_i}{\Longrightarrow} q_i,\ i = 1, 2$ and $s_1 \in Comp(s_2)$.

– Conversely, if $s_i \in tr(p_i), i = 1,2$, such that $r = q_1 \parallel q_2,\ p_i \stackrel{s_i}{\Longrightarrow} q_i,\ i = 1,2$ and $s_1 \in Comp(s_2)$, then $p_1 \parallel p_2 \stackrel{\oslash}{\Longrightarrow} r$.

Proof

We prove the first implication by induction on the length of the derivation $p_1 \parallel p_2 \stackrel{\oslash}{\Longrightarrow} r$ using the fact the rule applied lastly must be (8) or (9) (or their symmetrical) from the Table 7

- if the length of the derivation is 0, then there exist processes $q_1 = p_1$ and $q_2 = p_2$, and the traces $\epsilon \in tr(p_i)$, $i = 1,2$, such that $r = p_1 \parallel p_2$. Evidently $\epsilon \in Comp(\epsilon)$.

- the hypothesis is true for any $p_1, p_2, r \in \mathcal{P}_b$ such that $p_1 \parallel p_2 \stackrel{\oslash}{\Longrightarrow} r$, by a derivation of length at most n. Let suppose that the length of the derivation is $n+1$. Then it must look like: $p_1 \parallel p_2 \stackrel{\alpha}{\longrightarrow} r_1 \stackrel{\oslash}{\Longrightarrow} r$, where α is either τ or $\bar{a}$, with $a \in Ch$. For the first step $p_1 \parallel p_2 \stackrel{\alpha}{\longrightarrow} r_1$ we have a transition obtained either by the rule (8) or by the rule (9) (or by one of their symmetrical).

- if $\alpha = \tau$, then the transition is obtained by the rule (9) and we obtain $p_1 \stackrel{\tau}{\longrightarrow} p_1'$, $p_2 \stackrel{\tau:}{\longrightarrow} p_2$ and $p_1 \parallel p_2 \stackrel{\tau}{\longrightarrow} p_1' \parallel p_2$. Then $r_1 = p_1' \parallel p_2'$, and $p_1' \parallel p_2' \stackrel{\oslash}{\Longrightarrow} r$ by a derivation of length at most n, where $p_2' = p_2$. By hypothesis of induction, there exists processes $q_1,\ q_2$ and traces $s_i \in tr(p_i), i = 1,2$, such that $r = q_1 \parallel q_2,\ p_i' \stackrel{s_i}{\Longrightarrow} q_i,\ i = 1,2$ and $s_1 \in Comp(s_2)$. Since $p_1 \stackrel{\tau}{\longrightarrow} p_1'$, we infer $p_1 \stackrel{s_1}{\Longrightarrow} q_1$ and $p_2 \stackrel{s_2}{\Longrightarrow} q_2$.

- if α is $\bar{a}$ and the transition is obtained by the rule (8), then we obtain $p_1 \stackrel{\bar{a}}{\longrightarrow} p_1'$, $p_2 \stackrel{a}{\longrightarrow} p_2$ and $p_1 \parallel p_2 \stackrel{\bar{a}}{\longrightarrow} p_1' \parallel p_2$. Then $r_1 = p_1' \parallel p_2'$, and $p_1' \parallel p_2' \stackrel{\oslash}{\Longrightarrow} r$ by a derivation of length at most n, where $p_2' = p_2$. By hypothesis of induction, there exist processes $q_1,\ q_2$ and traces $s_i' \in tr(p_i'), i = 1,2$, such that $r = q_1 \parallel q_2,\ p_i' \stackrel{s_i'}{\Longrightarrow} q_i,\ i = 1,2$ and $s_1' \in Comp(s_2')$. Since $p_1 \stackrel{\tau}{\longrightarrow} p_1'$, we infer $p_1 \stackrel{s_1}{\Longrightarrow} q_1$, where $s_1 = \bar{a}.s_1'$. Also, we obtain $p_2 \stackrel{s_2}{\longrightarrow} p_2'$, where $s_2 = a.s_2'$. It remains to prove $s_1 \in Comp(s_2)$. Using the definition, $Comp(s_2) = Comp(a.s_2') = \{\bar{a}.s \mid s \in Comp(s_2')\}$. But, by hypothesis of induction we have that $s_1' \in Comp(s_2')$, i.e. $s_1 = \bar{a}.s_1' \in Comp(s_2)$.

The other cases are similar.

We prove the converse by induction on $|s_2|$.

- $s_2 = \epsilon$.

$s_1 \in Comp(s_2) = \{\epsilon\}$ implies $s_1 = \epsilon$, thus $p_i = q_i, i = 1,2$, and since $r = q_1 \parallel q_2$ we obtain $p_1 \parallel p_2 \stackrel{\oslash}{\Longrightarrow} r$.

$s_2 = \alpha.s_2'$, where $\alpha \in \{\bar{a}, a\}$.

- $s_2 = a.s_2'$.

$s_1 \in Comp(s_2) = \{\bar{a}.s \mid s \in Comp(s_2')\}$, then $s_1 = \bar{a}.s_1', s_1' \in Comp(s_2')$. By hypothesis of induction, $p_i \stackrel{s_i}{\Longrightarrow} q_i,\ i = 1,2$, i.e. $p_1 \stackrel{\bar{a}}{\Longrightarrow} p_1' \stackrel{s_1'}{\Longrightarrow} q_1$ and $p_2 \stackrel{a}{\Longrightarrow} p_2' \stackrel{s_2'}{\Longrightarrow} q_2$. We also have $r = q_1 \parallel q_2$. Using the hypothesis of induction, we obtain $p_1' \parallel p_2' \stackrel{\oslash}{\Longrightarrow} r$. Applying once the rule (8) and zero or several times the rule (9) (depending on the number of τ contained in $p_1 \stackrel{\bar{a}}{\Longrightarrow} p_1'$ and $p_2 \stackrel{a}{\Longrightarrow} p_2'$), we obtain $p_1 \parallel p_2 \stackrel{\bar{a}}{\Longrightarrow} p_1' \parallel p_2' \stackrel{\oslash}{\Longrightarrow} r$, and thus $p_1 \parallel p_2 \stackrel{\oslash}{\Longrightarrow} r$.

- $s_2 = \bar{a}.s_2'$.

$s_1 \in Comp(s_2) = \{a.s,\ a:.s \mid s \in Comp(s_2')\}$, then $s_1 = a.s_1'$ or $s_1 = a:.s_1'$, such that $s_1' \in Comp(s_2')$. By hypothesis, $p_i \stackrel{s_i}{\Longrightarrow} q_i,\ i = 1,2$, i.e. $p_1 \stackrel{a}{\Longrightarrow} p_1' \stackrel{s_1'}{\Longrightarrow} q_1$ or $p_1 \stackrel{a:}{\Longrightarrow} p_1' \stackrel{s_1'}{\Longrightarrow} q_1$

and $p_2 \stackrel{\bar{a}}{\Longrightarrow} p'_2 \stackrel{s'_2}{\Longrightarrow} q_2$. We also have $r = q_1 \parallel q_2$. Using the hypothesis of induction we have $p'_1 \parallel p'_2 \stackrel{\oslash}{\Longrightarrow} r$. By applying the symmetrical of the rule (8) or of the rule (9), and zero or several times the rule (9) (depending on the number of τ contained in $p_1 \stackrel{\bar{a}}{\Longrightarrow} p'_1$ and $p_2 \stackrel{a}{\Longrightarrow} p'_2$),we obtain $p_1 \parallel p_2 \stackrel{\bar{a}}{\Longrightarrow} p'_1 \parallel p'_2 \stackrel{\oslash}{\Longrightarrow} r$, and hence $p_1 \parallel p_2 \stackrel{\oslash}{\Longrightarrow} r$.

The case $s_2 = a:.s'_2$ is similar as the case $s_2 = a.s'_2$. $\Box$ *Lemma* 1

Lemma 2

For any traces s_0, s_1 and s_2, if $s_0 \in Comp(Comp(s_1))$ and $s_2 \in Comp(s_1)$, then $s_2 \in Comp(s_0)$.

Proof

By induction on $|s_1|$.

- $s_1 = \epsilon$ $Comp(\epsilon) = \{\epsilon\}$. $Comp(Comp(\epsilon)) = \{\epsilon\}$.

$s_0 \in Comp(Comp(s_1))$ implies $s_0 = \epsilon$. $s_2 \in Comp(s_1)$ implies $s_2 = \epsilon$. Thus $s_2 \in Comp(s_0)$.

- $s_1 = a.s'_1$

$Comp(a.s'_1) = \{\bar{a}.s'_2 \mid s'_2 \in Comp(s'_1)\}$. $Comp(Comp(a.s'_1)) = Comp(\{\bar{a}.s'_2 \mid s'_2 \in Comp(s'_1)\}) = \{a:.s'_0, a.s'_0 \mid s'_0 \in Comp(Comp(s'_1))\}$. $s_0 \in Comp(Comp(a.s'_1))$ implies $s_0 = a:.s'_0$ or $s_0 = a.s'_0$ with $s'_0 \in Comp(Comp(s'_1))\}$.

$s_2 \in Comp(s_1)$ implies $s_2 = \bar{a}.s'_2$ with $s'_2 \in Comp(s'_1)\}$.

Thus we obtain $s'_2 \in Comp(s'_1)$, $s'_0 \in Comp(Comp(s'_1))$, and by the hypothesis of induction we have, $s'_2 \in Comp(s'_0)$.

Hence $s_2 = \bar{a}.s'_2 \in Comp(s_0)$.

The other cases are similar. $\Box$ *Lemma* 2

Lemma 3 If $p \ll_{must} q$, then $\forall s \in \mathcal{R}Act^*$, $Conv(p,s)$ implies

1. $Conv(q,s)$;
2. $Comp(Comp(s)) \cap tr(q) \neq \emptyset$ implies $Comp(Comp(s)) \cap tr(p) \neq \emptyset$.

Proof Let $p \ll_{must} q$.

1. Let suppose that there exists $s \in \mathcal{R}Act^*$ such that $Conv(p,s)$ and $\neg Conv(q,s)$.
   $\neg Conv(q,s)$ implies that there exist a prefix s_1 of s, a trace $t_1 \in Comp(Comp(s_1))$ and a processes q_1 such that $q \stackrel{t_1}{\Longrightarrow} q_1$ and $q_1 \Uparrow^{\oslash}$.
   For a finite set of channels $X \in Ch_b$, let $in'(X)$ and $o_1(t,X)$ be two processes defined as follows:
   $$in'(X) \stackrel{not}{=} \begin{cases} nil & \text{if } X = \emptyset, \\ a.\bar{\omega} + in'(X \setminus \{a\}) & \text{otherwise, with } a \in X. \end{cases} \quad (10)$$
   $$o_1(t,X) = \begin{cases} \tau.\bar{\omega} & \text{if } t = \epsilon, \\ \tau.\bar{\omega} + \bar{a}.o_1(s,X) + in'(X) & \text{if } t = a:.s \text{ or } t = a.s,\ a \in Ch_b, \\ \tau.\bar{\omega} + a.o_1(s,X) + in'(X) & \text{if } t = \bar{a}.s,\ a \in Ch_b. \end{cases} \quad (11)$$
   By construction (and using Lemma 1) we obtain $q \parallel o_1(s_1, fn(p,q)) \stackrel{\oslash}{\Longrightarrow} q_1 \parallel \tau.\bar{\omega}$ and since $q_1 \Uparrow^{\oslash}$ we have q $\underline{m\!\!\!/ust}$ $o_1(s_1, fn(p,q))$.
   Since $Conv(p,s)$ and $s_1 \in pre(s)$ we obtain $Conv(p,s_1)$ and hence p $\underline{must}$ $o_1(s_1, fn(p,q))$, contradiction with $p \ll_{must} q$.
   $in'(X)$ assure that any deviation from the trace s leads to a successful execution.

2. Let suppose that there exists $s \in \mathcal{R}Act^*$ such that $Conv(p, s)$, $Comp(Comp(s)) \cap tr(q) \neq \emptyset$ and $Comp(Comp(s)) \cap tr(p) = \emptyset$.
   For a finite set of channels $X \in Ch_b$, let $o_2(t, X)$ be the observer defined as follows:

$$o_2(t,X) = \begin{cases} nil & \text{if } t = \epsilon, \\ \tau.\bar{\omega} + \bar{a}.o_2(s,X) + in'(X) & \text{if } t = a : .s \text{ or } t = a.s,\ a \in Ch_b, \\ \tau.\bar{\omega} + a.o_2(s,X) + in'(X) & \text{if } t = \bar{a}.s,\ \ a \in Ch_b. \end{cases} \tag{12}$$

   By construction (and using the Lemma 1) we have $q \parallel o_2(s, fn(p, q)) \stackrel{\oslash}{\Longrightarrow} q_1 \parallel nil$ (without emitting on the channel ω) and we obtain $q \; \underline{\not{m}ust} \; o_2(s, fn(p, q))$.
   Since $Conv(p, s)$ and $Comp(Comp(s)) \cap tr(p) = \emptyset$ we obtain $p \; \underline{must} \; o_2(s, fn(p, q))$, contradiction with $p \ll_{must} q$.
   $in'(X)$ assure that any deviation from the trace s leads to a successful execution.

$\square$ *Lemma* 3

Lemma 4 If $p \leq q$, and $Conv(p, s)$ then
$Comp(Comp(s)) \cap tr(q) \neq \emptyset$ implies $Comp(Comp(s)) \cap tr(p) \neq \emptyset$.

Proof

Let suppose that there exists $s \in \mathcal{R}Act^*$ such that $Conv(p, s)$, $Comp(Comp(s)) \cap tr(q) \neq \emptyset$.

Since $p \leq q$, and $Conv(p, s)$, from the definition 7 we obtain $Conv(q, s)$.

$Comp(Comp(s)) \cap tr(q) \neq \emptyset$ implies that there exists q_1 and $s_1 \in Comp(Comp(s)) \cap tr(q)$ such that $q \stackrel{s_1}{\Longrightarrow} q_1$.

$Conv(q, s)$ implies $q_1 \Downarrow^{\oslash}$.

$q \stackrel{s_1}{\Longrightarrow} q_1$ and $q_1 \Downarrow^{\oslash}$ imply that there exists an extension t_1 of the trace s_1 (obtained from s_1 by adding a maximal finite sequence of autonomous actions that q_1 can make) such that $t_1 \in tr(q)$, $Conv(q, t_1)$, and $\mathcal{S}tab(q, t_1) \neq \emptyset$.

Since $Conv(p, s)$ and $s_1 \in pre(t_1) \cap Comp(Comp(s))$ and t_1 is obtained from s_1 by adding to the end only autonomous, we obtain $Conv(p, t_1)$.

From the definition 7, $p \leq q$, and $Conv(p, t_1)$ and $\mathcal{S}tab(q, t_1) \neq \emptyset$ imply $\mathcal{S}tab(p, t_1) \neq \emptyset$, and thus $tr(p) \cap Comp(Comp(t_1)) \neq \emptyset$, which involve $Comp(Comp(s)) \cap tr(p) \neq \emptyset$.

$\square$ *Lemma* 4

9 Proofs relative to Section 5

Lemma 7.

- *Let $p_1, p_2 \in \mathcal{P}_b$ be two processes such that $p_1 \parallel p_2 \stackrel{\oslash}{\Longrightarrow} r$. Then, there exist two processes q_1, q_2 and two traces $s_i \in tr(p_i), i = 1, 2$, such that $r = q_1 \parallel q_2$, $p_i \stackrel{s_i}{\Longrightarrow} q_i$, $i = 1, 2$ and $s_1 \in Comp(s_2)$.*
- *Conversely, if $s_i \in tr(p_i), i = 1, 2$, such that $r = q_1 \parallel q_2$, $p_i \stackrel{s_i}{\Longrightarrow} q_i$, $i = 1, 2$ and $s_1 \in Comp(s_2)$, then $p_1 \parallel p_2 \stackrel{\oslash}{\Longrightarrow} r$.*

Proof

We prove the first implication by induction on the length of the derivation $p_1 \parallel p_2 \stackrel{\oslash}{\Longrightarrow} r$ using the fact that the last applied rule is (12) or (13) (or their symmetrical) from Table 3

- if the length of the derivation is 0, then there are processes $q_1 = p_1$ and $q_2 = p_2$, and traces $\epsilon \in tr(p_i), i = 1, 2)$, such that $r = p_1 \parallel p_1$. Evidently $\epsilon \in Comp(\epsilon)$.

- let suppose that the assertion is true for any $p_1, p_2, r \in \mathcal{P}_b$ such that $p_1 \parallel p_2 \stackrel{\oslash}{\Longrightarrow} r$, by a derivation of length at most n. Let suppose that the length of the derivation is $n+1$. Then we have: $p_1 \parallel p_2 \stackrel{\alpha}{\longrightarrow} r_1 \stackrel{\oslash}{\Longrightarrow} r$, where α is τ or $\bar{a}x$ or $\bar{a}(x)$, with $a, x \in Ch$. For the first step $p_1 \parallel p_2 \stackrel{\alpha}{\longrightarrow} r_1$ we have a transition obtained either by the rule (12) or by the rule (13) (or by one or their symmetrical).

- if α is τ the transition is obtained by the rule (14) and we have $p_1 \stackrel{\tau}{\longrightarrow} p_1'$, $p_2 \stackrel{\tau:}{\longrightarrow} p_2$ and $p_1 \parallel p_2 \stackrel{\tau}{\longrightarrow} p_1' \parallel p_2$. Then $r_1 = p_1' \parallel p_2'$, and $p_1' \parallel p_2' \stackrel{\oslash}{\Longrightarrow} r$ by a derivation of length at most n, where $p_2' = p_2$.

By the hypothesis of induction, there are processes q_1, q_2 and traces $s_i \in tr(p_i'), i = 1, 2$, such that $r = q_1 \parallel q_2$, $p_i' \stackrel{s_i}{\Longrightarrow}_1 q_i,\ i = 1, 2$ and $s_1 \in Comp(s_2)$. Since $p_1 \stackrel{\tau}{\longrightarrow} p_1'$, we obtain that $p_1 \stackrel{s_1}{\Longrightarrow} q_1$ and $p_2 \stackrel{s_2}{\Longrightarrow} q_2$.

- if α is $\bar{a}x$ the transition $p_1 \parallel p_2 \stackrel{\bar{a}x}{\longrightarrow} p_1' \parallel p_2$ is obtained by one of the rules (12) or (13). We have $p_1 \stackrel{\bar{a}x}{\longrightarrow} p_1'$ and either $p_2 \stackrel{a\langle x\rangle}{\longrightarrow} p_2$, or $p_2 \stackrel{a:}{\longrightarrow} p_2$.

Then $r_1 = p_1' \parallel p_2'$, and $p_1' \parallel p_2' \stackrel{\oslash}{\Longrightarrow} r$ by a derivation of length at most n, where $p_2' = p_2$. By the hypothesis of induction, there are processes q_1, q_2 and traces $s_i' \in tr(p_i'), i = 1, 2$, such that $r = q_1 \parallel q_2$, $p_i' \stackrel{s_i}{\Longrightarrow}_1 q_i,\ i = 1, 2$ and $s_1' \in Comp(s_2')$. Since $p_1 \stackrel{\bar{a}x}{\longrightarrow} p_1'$, we have that $p_1 \stackrel{s_1}{\Longrightarrow}_1 q_1$, where $s_1 = \bar{a}x.s_1'$. We also have $p_2 \stackrel{s_2}{\Longrightarrow}_1 p_2'$, where $s_2 = a : .s_2'$ or $s_2 = a\langle x\rangle.s_2'$. It remains to prove that $s_1 \in Comp(s_2)$. From the definition, $Comp(s_2) = Comp(a : .s_2') = Comp(a\langle x\rangle.s_2') = \{\bar{a}x.s \mid s \in Comp(s_2')\}$. But using the hypothesis of induction, we have that $s_1' \in Comp(s_2')$, i.e. $s_1 = \bar{a}\langle x\rangle.s_1' \in Comp(s_2)$.

- if α is $\bar{a}(x)$ the transition $p_1 \parallel p_2 \stackrel{\bar{a}(x)}{\longrightarrow} p_1' \parallel p_2$ is obtained by one of the rules (12) or (13). We have $p_1 \stackrel{\bar{a}(x)}{\longrightarrow} p_1'$, $x \notin fn(q)$ and either $p_2 \stackrel{a\langle x\rangle}{\longrightarrow} p_2$, either $p_2 \stackrel{a:}{\longrightarrow} p_2$.

Applying the rules of the Table 4 we obtain $p_2 \stackrel{a(x)}{\longrightarrow}_1 p_2$, or $p_2 \stackrel{a(x):}{\longrightarrow} p_2$.

Then $r_1 = p_1' \parallel p_2'$, and $p_1' \parallel p_2' \stackrel{\oslash}{\Longrightarrow} r$ by a derivation of length at most n, where $p_2' = p_2$. By the hypothesis of induction, there are processes q_1, q_2 and traces $s_i' \in tr(p_i'), i = 1, 2$, such that $r = q_1 \parallel q_2$, $p_i' \stackrel{s_i}{\Longrightarrow}_1 q_i,\ i = 1, 2$ and $s_1' \in Comp(s_2')$. Since $p_1 \stackrel{\bar{a}(x)}{\longrightarrow}_1 p_1'$, we have that $p_1 \stackrel{s_1}{\Longrightarrow}_1 q_1$, where $s_1 = \bar{a}(x).s_1'$. We also have $p_2 \stackrel{s_2}{\Longrightarrow}_1 p_2'$, where $s_2 = a(x) : .s_2'$ or $s_2 = a(x).s_2'$. It remains to prove that $s_1 \in Comp(s_2)$. From the definition, $Comp(s_2) = Comp(a(x) : .s_2') = Comp(a(x).s_2') = \{\bar{a}(x).s \mid s \in Comp(s_2')\}$. But using the hypothesis of induction, we have that $s_1' \in Comp(s_2')$, i.e. $s_1 = \bar{a}(x).s_1' \in Comp(s_2)$.

The other cases are similar.

We prove the converse by induction on $|s_2|$.

- $s_2 = \epsilon$.

$s_1 \in Comp(s_2) = \{\epsilon\}$ implies $s_1 = \epsilon$, then $p_i = q_i, i = (1, 2)$, and since $r = q_1 \parallel q_2$ we obtain $p_1 \parallel p_2 \stackrel{\oslash}{\Longrightarrow} r$.

$s_2 = \alpha.s_2'$, where $\alpha \in \{\bar{a}x, \bar{a}(x), a\langle x\rangle, a(x), a :, a(x) :\}$.

- $s_2 = a\langle x\rangle.s'_2$.

$s_1 \in Comp(s_2) = \{\bar{a}x.s \mid s \in Comp(s'_2)\}$, then $s_1 = \bar{a}x.s'_1, s'_1 \in Comp(s'_2)$. By the hypothesis, $p_i \stackrel{s_i}{\Longrightarrow}_1 q_i,\ i = 1,2$, i.e. $p_1 \xrightarrow{\bar{a}x}_1 p'_1 \stackrel{s'_1}{\Longrightarrow}_1 q_1$ and $p_2 \stackrel{a\langle x\rangle}{\Longrightarrow}_1 p'_2 \stackrel{s'_2}{\Longrightarrow} q_2$. We also have $r = q_1 \parallel q_2$.

$p_1 \xrightarrow{\bar{a}x}_1 p'_1$ and $p_2 \stackrel{a\langle x\rangle}{\Longrightarrow}_1 p'_2$ implies $p_1 \xrightarrow{\bar{a}x} p'_1$ and $p_2 \stackrel{a\langle x\rangle}{\Longrightarrow} p'_2$

Using the hypothesis of induction, we have that $p'_1 \parallel p'_2 \stackrel{\oslash}{\Longrightarrow} r$. By applying the rule (12), $p_1 \parallel p_2 \xrightarrow{\bar{a}x} p'_1 \parallel p'_2 \stackrel{\oslash}{\Longrightarrow} r$, thus $p_1 \parallel p_2 \stackrel{\oslash}{\Longrightarrow} r$

- $s_2 = \bar{a}(x).s'_2$.

$s_1 \in Comp(s_2) = \{a(x).s,\ a(x) : .s \mid s \in Comp(s'_2)\}$, then $s_1 = a(x).s'_1$ or $s_1 = a(x) : .s'_1$, such that $s'_1 \in Comp(s'_2)$.

By the hypothesis, $p_i \stackrel{s_i}{\Longrightarrow}_1 q_i,\ i = 1,2$, i.e. $p_1 \xrightarrow{a(x)}_1 p'_1 \stackrel{s'_1}{\Longrightarrow}_1 q_1$ or $p_1 \xrightarrow{a(x):}_1 p'_1 \stackrel{s'_1}{\Longrightarrow}_1 q_1$ and $p_2 \stackrel{\bar{a}(x)}{\Longrightarrow}_1 p'_2 \stackrel{s'_2}{\Longrightarrow}_1 q_2$. We also have $r = q_1 \parallel q_2$.

$p_1 \xrightarrow{a(x)}_1 p'_1$ (respectively $p_1 \xrightarrow{a(x):}_1 p'_1$, $p_2 \stackrel{\bar{a}(x)}{\Longrightarrow}_1 p'_2$) implies $p_1 \xrightarrow{ax} p'_1$ (respectively $p_1 \xrightarrow{a:} p'_1$, $p_2 \stackrel{\bar{a}(x)}{\Longrightarrow} p'_2$) and $x \notin fn(p_1)$.

Using the hypothesis of induction, we have that $p'_1 \parallel p'_2 \stackrel{\oslash}{\Longrightarrow} r$. By applying the symmetric of the rule (12) or of the rule (13), we obtain $p_1 \parallel p_2 \xrightarrow{\bar{a}} p'_1 \parallel p'_2 \stackrel{\oslash}{\Longrightarrow} r$, and hence $p_1 \parallel p_2 \stackrel{\oslash}{\Longrightarrow} r$.

The other cases are similar. □ *Lemma* 7

Theorem 3 For all processes p and q, $p \ll_{may} q$ iff $tr(p) \subseteq Comp(Comp(tr(q)))$.

Proof

Firstly, we will define a special set of observers. For a finite set of channels $M \subseteq Ch_b$, we shall define:

$$in(M) \stackrel{not}{=} \begin{cases} nil & \text{if } M = \emptyset, \\ a.nil + in(M \setminus \{a\}) & \text{otherwise, with } a \in M. \end{cases} \tag{13}$$

$$\langle x \notin M\rangle p \stackrel{not}{=} \begin{cases} p & \text{if } M = \emptyset, \\ \langle x = a\rangle nil, \langle x \notin M \setminus \{a\}\rangle p & \text{otherwise, with } a \in M. \end{cases} \tag{14}$$

Let t be a trace and let $M \subseteq Ch_b$ be a finite set of channels. Let $o_M(t)$ be an observer defined by

$$o_M(t) = \begin{cases} \bar{\omega}.nil + in(M) & \text{if } t = \epsilon, \\ \bar{a}x.o_{M\cup\{x\}}(s) + in(M) & \text{if } t = a : .s \text{ or } t = ax.s,\ a, x \in Ch, \\ \bar{a}(x).o_{M\cup\{x\}}(s) + in(M) & \text{if } t = a(x) : .s \text{ or } t = a(x).s,\ a, x \in Ch, \\ a(y).\langle x = y\rangle o_M(s) + in(M) & \text{if } t = \bar{a}x.s,\ a, x \in Ch, \\ a(y).\langle y \notin M\rangle o_{M\cup\{y\}}(s) + in(M) & \text{if } t = \bar{a}(x).s,\ a, x \in Ch. \end{cases} \tag{15}$$

Now the proof follows as in the proof of the Theorem 1. Disposing in $b\pi$-calculus of the operator "if then else" in its complete form (and not only in the form "if then" as in π-calculus) allow us to distinguish a bound output from a free one. □ *Theorem* 3

Theorem 4 For all processes p and q, $\quad p \ll_{must} q \quad$ iff $\quad p \leq q$.

Proof

The proof follows as for the Theorem 2; the observers used this time are defined below.

For a finite set of channels $X \in Ch_b$, let $o_1(t,X)$ $o_2(t,X)$ $o_3(t,X)$ be the processes defined as follows:

$$o_1(t,X) = \begin{cases} \tau.\bar{\omega} & \text{if } t = \epsilon, \\ \tau.\bar{\omega} + \bar{a}x.o_1(s,X) + in'(X) & \text{if } t = a:.s \text{ or } t = a\langle x\rangle.s,\ a, x \in Ch_b, \\ \tau.\bar{\omega} + \bar{a}(x).o_1(s, X \cup \{x\}) + in'(X) & \text{if } t = a(x):.s \text{ or } t = a(x).s,\ a, x \in Ch_b, \\ \tau.\bar{\omega} + a(y).\langle x = y\rangle o_1(s,X), \bar{\omega} + in'(X) & \text{if } t = \bar{a}x.s,\ a, x \in Ch_b, \\ \tau.\bar{\omega} + a(y).\langle y \notin X\rangle o_1(s, X \cup \{y\}) + in'(X) & \text{if } t = \bar{a}(x).s,\ a, x \in Ch_b. \end{cases} \tag{16}$$

$$o_2(t,X) = \begin{cases} nil & \text{if } t = \epsilon, \\ \tau.\bar{\omega} + \bar{a}x.o_2(s,X) + in'(X) & \text{if } t = a:.s \text{ or } t = a\langle x\rangle.s,\ a \in Ch_b, \\ \tau.\bar{\omega} + \bar{a}(x).o_2(s, X \cup \{x\}) + in'(X) & \text{if } t = a(x):.s \text{ or } t = a(x).s,\ a, x \in Ch_b, \\ \tau.\bar{\omega} + a(y).\langle x = y\rangle o_2(s,X), \bar{\omega} + in'(X) & \text{if } t = \bar{a}x.s,\ a, x \in Ch_b, \\ \tau.\bar{\omega} + a(y).\langle y \notin X\rangle o_2(s, X \cup \{y\}) + in'(X) & \text{if } t = \bar{a}(x).s,\ a, x \in Ch_b. \end{cases} \tag{17}$$

$$o_3(t,X) = \begin{cases} in'(X) & \text{if } t = \epsilon, \\ \tau.\bar{\omega} + \bar{a}x.o_3(s,X) + in'(X) & \text{if } t = a:.s \text{ or } t = a\langle x\rangle.s,\ a \in Ch_b, \\ \tau.\bar{\omega} + \bar{a}(x).o_3(s, X \cup \{x\}) + in'(X) & \text{if } t = a(x):.s \text{ or } t = a(x).s,\ a \in Ch_b, \\ \tau.\bar{\omega} + a(y).\langle x = y\rangle o_3(s,X), \bar{\omega} + in'(X) & \text{if } t = \bar{a}x.s,\ a, x \in Ch_b, \\ \tau.\bar{\omega} + a(y).\langle y \notin X\rangle o_3(s, X \cup \{y\}) + in'(X) & \text{if } t = \bar{a}(x).s,\ a, x \in Ch_b. \end{cases} \tag{18}$$

where

$$\langle x \notin X\rangle p \stackrel{not}{=} \begin{cases} p & \text{if } X = \emptyset, \\ \langle x = a\rangle\bar{\omega}, \langle x \notin X \setminus \{a\}\rangle p & \text{otherwise, with } a \in X. \end{cases} \tag{19}$$

$\Box$ *Theorem* 4