A Broadcast-based Calculus for Communicating Systems

Cristian ENE, Traian MUNTEAN
University of Marseilles
Laboratoire d’Informatique de Marseille; (CNRS)
Parc Scientifique de Luminy - Case 925
F-13288 Marseille, France
telephone: 33 - 491 82 85 32, fax: 33 - 491 82 85 11
cene@esil.univ-mrs.fr, muntean@lim.univ-mrs.fr

Abstract

This paper presents a process calculus for reconfigurable communicating systems which has broadcast as basic communication primitive, and we provide an operational semantics for this calculus. We illustrate the calculus through some examples, and we propose three behavioural equivalences for reasoning about systems of broadcasting processes, namely, barbed equivalence, step-equivalence and labelled bisimilarity. An important result, is that all these relations coincide, providing different ways to study the equivalence/non-equivalence of two systems. Then, we provide a direct characterization for the strong congruence relation induced by these equivalences. Finally, we give a complete axiomatisation for strong congruence.

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 [3] 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 [8] 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-symmetry 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 (9) where outputs are non-blocking and locally controlled, whereas inputs are externally controlled and can not be refused). In [14], Prasad introduces and develops [15] a calculus of broadcasting systems, namely CBS. His calculus, inspired by Milner’s CCS (10), 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 [7], 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. To summarise, it seems that there is not a framework which try to analyse (at least at theoretical level) what it happens if we combine mobility and broadcast (as it is the case for processes which use group communications à la PVM [5], buses-based reconfigurable architectures or Packet Radio Networks).

The aim of this paper is to introduce a new process calculus, whose unique and basic communication primitive is broadcast, and which permits to model reconfigurable group communication systems.

The rest of the paper is as follows. In section 2 we present the \(b\pi\)-calculus (already briefly introduced in [3]) as a variant of a broadcast calculus (inspired from [15]) together with some examples. Section 3 presents three equivalences between processes, and a proof of their similar discriminative power. The section 4 is devoted to the congruence induced by the already defined equivalences. In section 5 we provide for the congruence a complete axiomatisation. Section 6, discusses related works and presents future directions of research. Due to the limited length of this paper, the proofs of presented results have been omitted; they are included in the full version of this paper [4].

2. Preliminaries

2.1. The \(b\pi\)-calculus

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 [15], and the \(\pi\)-calculus proposed by Milner, Parrow and Walker [11]. It differs from the broadcast calculus, in that communications are made on channels or ports (and transmitted values are channels too), and from the \(\pi\)-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.

\[
p ::= n \mid \pi.p \mid \nu x.p \mid p_1 + p_2 \mid \langle x = y \rangle p_1.p_2 \mid X \langle \tilde{x} \rangle \mid \langle \text{rec } X \langle \tilde{x} \rangle.p\rangle/y
\]

Table 1. Processes

where \(\pi\) belongs to the set of prefixes \(\pi ::= \tau \mid x(y) \mid \tilde{x}y\), and \(x, y \in Ch_b\).

Prefixes denote the basic actions of processes: \(\tau\) is a silent action (which corresponds to an internal transition), \(x(y)\) is the input of the name \(y\) on the channel \(x\), and \(\tilde{x}y\) is the output of the name \(y\) on the channel \(x\). \(\text{nil}\) is a process which does nothing. \(\pi p\) is the process which realize the action denoted by \(\pi\) and next behaves like \(p\). \(p_1 + p_2\) denotes choice, it behaves like \(p_1\) or \(p_2\). \(\nu x.p\) 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 \(p_2\). \(X \langle \tilde{x} \rangle\) is a process identifier whose arity is satisfied by \(\langle \tilde{x} \rangle\) and \(\langle \text{rec } X \langle \tilde{x} \rangle.p\rangle/y\) 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 \(\nu x.y(x)\), are \(x - \text{binders}, i.e. in \nu x.p\) and \(y(x)\), \(x\) is 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.

In literature there have been defined relations among processes which relate processes which are “almost the same”. Such a relation is a congruence, and allows to substitute a process by another congruent process, when needed. We shall need to use the congruence to push “the \(\nu\)” to outside, in order to model the scope extrusion.

Definition 1 Structural congruence, denoted \(\equiv\), is the smallest congruence over the set of processes which satisfies the conditions of Table 2. (we omitted the symmetric versions of rules (2), (3), (4) and (5)).

<table>
<thead>
<tr>
<th>Rule</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(p \equiv q) if (p) and (q) are (\alpha)-convertible</td>
</tr>
<tr>
<td>2</td>
<td>(\nu x.y(x)p \equiv \nu y.y(x)p) if (x \neq y)</td>
</tr>
<tr>
<td>3</td>
<td>(\langle x = y \rangle p \equiv \nu x.p) if (x \notin fn(q))</td>
</tr>
<tr>
<td>4</td>
<td>(\nu x.p + q \equiv \nu x.(p + q)) if (x \notin fn(q))</td>
</tr>
</tbody>
</table>
| 5 | \(\langle y = z \rangle q \equiv \nu x.(\langle y = z \rangle p.q)\) if \(x \notin fn(q)\) \cup \{y, z\}\)

Table 2. Structural congruence

Definition 2 Actions, ranged over \(\alpha, \beta\) are defined by the following grammar:

\[
\alpha ::= a(x) \mid \tilde{a}x \mid \nu x.a(x) \mid \tau
\]

where \(a, x \in Ch_b\). An action is either a reception, a (possibly bound) output, or the silent action \(\tau\), denoting an internal transition. In \(\tilde{a}x\), \(a(x)\) and \(\nu x.a(x)\), \(a\) is the subject
of the communication and $x$ is its object. By extension $f n(\alpha)$ also denotes the free names used in the action $\alpha$ i.e. $f n(\alpha(x)) = f n(\bar{\alpha} x) = \{ a, x \}$, $f n(v x a x) = \{ a \}$).

We give an operational semantics for our calculus in terms of transitions over the set $P_b$ of processes. Before, we define, similarly to [14], a relation $\rightarrow \subseteq P_b \times A$ denoted $p \rightarrow \alpha$ and which can be read “$p$ discards the action $\alpha$” (see Table 3).

| (1) $n\bar{a} \rightarrow \bar{a}$ | (2) $\tau, p \rightarrow \bar{a}$ | (3) $b_{b} \cdot p \rightarrow p$ | (4) $p, a \rightarrow p'$ |
| (5) $b_{b} \cdot p \rightarrow p$ | (6) $b_{b} \cdot p \rightarrow p$ | (7) $p, a \rightarrow p$ | (8) $p, a \rightarrow p'$ |
| (9) $\bar{a} \cdot p \rightarrow p$ | (10) $\bar{a} \cdot p \rightarrow p'$ | (11) $\bar{a} \cdot p \rightarrow p$ | (12) $\bar{a} \cdot p \rightarrow p'$ |

Table 3. The “discard” relation

Intuitively, a process discards actions which do not concern it (process is ready to output or to make a $\tau$ action, or actions have as subject other channels that those on which the process is waiting). $n\bar{a}$, $\tau, p$ or $b_{b} \cdot p$ discard any action. $\tau$ is ignored by all processes. A process waiting for a message on a channel, $b$, discards actions on the other channels $a$ with $a \neq b$. Rules (7) to (12) follow the structure of the term.

Definition 3 Transition system The operational semantics of $\mathit{br}$-calculus is defined as a labelled transition system defined over the set $P_b$ of processes. The judgement $p \rightarrow \alpha \rightarrow p'$ means that the process $p$ is able to perform action $\alpha$ and to evolve next to $p'$. The operational semantics is given in Table 4 (we omitted the symmetric versions of rules (11) and (12)).

A communication between processes is performed through unbuffered broadcast. Compared to $\pi$-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.

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.

Rules (1) to (4) are straightforward. Rule (5) states that when a local channel name is emitted, the related output has to be bound. The other rules follow the structure of the term.

At first glance, rule (5) seems counter-intuitive as opposed to the related rule in $\pi$-calculus, but in our calculus, scope extrusion is obtained firstly using the congruences - permitted by rule (14) - to push “the $\nu^\prime$ outside, and the others rules to obtain a transition which matches the antecedent of rule (5) (since a fresh name can be acquired in a single step by more than one observer).

Lemma 1 If $p \overset{\nu x a}{\rightarrow} q$, then $3 \chi$ such that $q \equiv v x \chi^\prime$.

As usual we shall use the following notations:

- $\overset{\alpha}{\rightarrow} \overset{\beta}{\rightarrow} \alpha \beta$
- $\overset{\alpha}{\rightarrow} \overset{\beta}{\rightarrow} \alpha \beta$

Sometimes, we shall use, as in [7], $p \overset{a}{\rightarrow} p$ instead of $p \overset{a}{\rightarrow} p$ and $p \overset{a}{\rightarrow} p'$ to stand for either $p \overset{a}{\rightarrow} p'$ or $p \overset{a}{\rightarrow} p'$. Also, we shall omit the trail $n\bar{a}$ for processes, and we shall use $a, \bar{a}$ instead of $a(x)$ and $\bar{a} x$ whenever the communication over the channel $a$ is just used for synchronization, and the carried name $x$ does not matter. We shall also use the notation $\bar{a}(x)$ to stand for the bound output $v x a x$.

Lemma 2 If $p \overset{\overline{\alpha}}{\rightarrow} p'$, then $f n(p') \subseteq f n(p)$.

Corollary 1 If $p \overset{\overline{\alpha}}{\rightarrow} p'$, then $f n(p') \subseteq f n(p)$.

2.2. Examples

Example 1 A distributed algorithm for cycle-detection
We present a distributed algorithm for cycle-detection in a directed graph. *Detector* is a process which listens to new edges of the graph on a channel \(i\), and spawns for each received pair of names (source and destination of edge) a new “edge manager” *Edge manager*. An edge manager, broadcasts a personal token \(v\) (using the mechanism of name-generation). Next, for each received token \(w\), if it receives the own token \(v\), a cycle is detected and a signal is sent on \(o\), otherwise, it propagates the token further (along the paths of the graph). This is done by using the channels \(a\) and \(b\), which denote an edge \((a, b)\) in the graph. Hence, any token received on \(a\), and different from \(v\) is transmitted on \(b\).

\[
Detector(i, o) \overset{def}{=} 
i(x), i(y). (Detector(i, o) \parallel Edge\_manager(o, x, y))
\]

\[
Edge\_manager(o, a, b) \overset{def}{=} 
\nu v(\nu w. nil \parallel (\mu x. a(w) \parallel X(o, a, b, v)))(a, a, b, v))
\]

**Example 2 Detecting inconsistencies for transactions systems**

We extend the previous example to give an implementation in \(br\)-calculus of a fully distributed algorithm for detecting inconsistencies in partitioned distributed databases (our implementation is inspired from [11]).

In a replicated database, there exist several copies of each data item, with copies located at distinct sites in the system. We suppose that the database becomes partitioned (partitions \(p_j\) with \(j = 1, \ldots, n\)).

We allow transactions to continue to execute, but when the network is reconnected (simulated in our implementation by a broadcast on the channel “*uni f*”) , we have to check for inconsistencies. The idea is to construct a precedence graph that captures the temporal partial order between transactions. Then, the database is consistent iff the precedence graph contains no cycle. The vertices of the graph are all the transactions. An edge \(<t, p>\rightarrow<t_1, p_1>\) indicates that transaction \(t\) occurred before transaction \(t_1\), where \(p, p_1\) indicate the partition in which the transactions were executed.

Such an edge exists iff one of the following holds:

1. \(t\) read a data item \(i\) that was later written by \(t_1\) and \(p = p_1\)
2. \(t\) write a data item \(i\) that was later read or written by \(t_1\) and \(p = p_1\)
3. \(t\) read a data item \(i\) that was written by \(t_1\) and \(p \neq p_1\)

The database can be simulated by:

\[
\prod_{j=1}^{k} \prod_{l=1}^{n(l)} Item(data^j_i, data^l_j, p_j, uni f, Val)]
\]

where \(data^j_i\) with \(j = 1, k\) are the data items, each data item \(data^j_i\) having \(n(j)\) copies, and \(p_j \in \{p_1, \ldots, p_n\}\) is the corresponding partition.

An item manager waits for transactions; for each new transaction, it forks a new transaction manager, and serves the user which was making the request. A transaction for a data item \(i\), is an output on the channel \(i_1\), and contains the transaction identifier \(t_1\), the type (read or write), the partition affected, the return channel, and a value (which make sense for a write transaction).

\[
Item(i_1, i_2, p, ch\_part, uni f, Val) \overset{def}{=} 
\begin{cases}
uni f(p_1). Item(i_1, i_2, p, ch\_part, uni f, Val) +
i_1(t_1, type, p, req, V). (p_1 = p) \\
(type = w)
\end{cases}
\]

\[
Tr\_Man_w(i_1, i_2, p, uni f, t_1) \parallel Tr\_Man_r(i_1, i_2, p, uni f, t_1) \parallel req\_Val + req\_V)
\]

A transaction manager generates a new edge manager for each new ongoing transaction which affects the same data item on the same partition (cases 1. or 2.).

\[
Tr\_Man_w(i_1, i_2, p, uni f, t) \overset{def}{=} 
i_1(t_1, type, p, req, V). (p_1 = p) [Tr\_Man_w(i_1, i_2, p, uni f, t_1)] +
uni f(p). Edge\_manager(error, t, t_1)
\]

\[
Tr\_Man_r(i_1, i_2, p, uni f, t) \overset{def}{=} 
i_1(t_1, type, p, req, V). (p_1 = p) [Tr\_Man_r(i_1, i_2, p, uni f, t_1)] +
uni f(p). Edge\_manager(error, t, t_1)
\]

Once the network reconnected, transaction managers change their behaviour, trying to detect edges of the third kind. In addition, if there are two transactions which have written the same data item in two distinct partitions, then an error is detected (two contrary edges between two vertices).

\[
STr\_Man_w(i_2, p, t) \overset{def}{=} 
i_2(t_1, type, p_1). (p_1 = p) nil,
\]

\[
(type = w) error, [STr\_Man_w(i_2, p, t) \parallel Edge\_manager(error, t, t_1)] +
\]

\[
STr\_Man_r(i_2, p, t) \overset{def}{=} 
i_2(t_1, type, p_1). (p_1 = p) nil,
\]

0-7695-0990-8/01/$10.00 (C) 2001 IEEE
STR_{Man,w}(i_2, p, t) \mid \langle \text{type} =\r_nil, Edge_{\text{manager}}(\text{error}, t, t_1) \rangle +
\gamma_2[t, r, p]. STR_{Man,r}(i_2, p, t)

We note that this example uses the entire expressiveness power of our calculus. The same data item can be replicated (for reliability or efficiency reasons) and a transaction \( t \) can affect several data items; in this case broadcast is a quite natural communication primitive. In the same time, the ability to send and receive channel names across channels is used by item managers to fork new transaction managers corresponding to the received identifier.

**Example 3 Semantics of group communication primitives**

\(\pi\)-calculus provide a framework to specify and analyze systems which interact by a broadcast (or multicast) mechanism combined with mobility of processes (names, addresses). We take here, as an example programs which use communication primitives of PVM-like libraries ([5]). PVM is a software system that permits a network of heterogeneous computers to be used as a single parallel computer (the virtual machine). Thus large computational problems may be solved using the power of many computers. PVM supplies functions to automatically start up tasks on the virtual machine, and allows task to communicate (by point-to-point communications) and synchronize with each other. The interesting part is the simple simulation of group communication primitives, which seems difficult to express in a process algebra based on point-to-point communications ([3]). Moreover, even in CBS ([15]) it is unclear how one can implement the primitives which permit to have dynamic groups (processes can freely join or leave a group, once they have knowledge of the name of the group).

We present just a few communication primitives specific to concurrent applications (for the interpretation of imperative features in process algebra, see for example [16]):

\[
I := \text{send}(a, m) \mid \text{bcast}(g, m) \mid x = \text{receive}() \mid x = \text{spawn}(Q) \mid \text{join}(g) \mid \text{leave}(g)
\]

\[
P := I \mid I; P
\]

A process (or a task) is a sequence of actions. An action is either an output of a message \( m \) to another process \( a \) or to a group \( g \) of processes, or an input (from the own buffer) of a message (stored after in variable \( x \)), or a creation of a new group \( g \), or a joining to a group \( g \), or a leaving of a group \( g \) or a spawning of a child \( Q \). Once a process become member of a group \( g \), it receives any message sent to that group. Communications are asynchronous, in that outputs are non-blocking (messages being stored in the buffers of receivers). For the sake of simplicity, we suppose that there is no guarantee in what concerns the order of messages’ arrival.

Then, a possible encoding \( \left\{ P \right\}_a \) of a process \( P \) of address (pid) \( a \) is given below:

\[
\left\{ P \right\}_a \overset{def}{=} \nu r. v k(a, P \text{pool}(a, r, k, a) \mid \left\{ P \right\}_r)
\]

\[
P \text{pool}(a, r, k, f) \overset{def}{=} k + a(x). (P \text{pool}(a, r, k) \mid \text{cell}(r, x))
\]

\[
\text{cell}(r, x) \overset{def}{=} r(x) \cdot (\exists x + c(x). \text{cell}(r, x))
\]

\[
\text{send}(a, m); P \overset{def}{=} \nu t. (\exists t(x). P \mid t)
\]

\[
\text{bcast}(g, m); P \overset{def}{=} \nu t. (\exists gm \cdot t \cdot P \mid t)
\]

\[
\text{join}(g); P \overset{def}{=} \nu \pi k(g). P \mid t(k_g) \cdot P \mid t
\]

\[
\text{leave}(g); P \overset{def}{=} \nu \pi k(g). P \mid t(k_g) \cdot P \mid t
\]

\[
\text{spawn}(Q); P \overset{def}{=} \nu a. (\exists Q \cdot a \cdot \exists t(x) \cdot P \mid t)
\]

The translations of “send” and “bcast” primitives are similar, but actually, for a channel \( a \) which appears as the argument of the “send” primitive, there is exactly one receiver, whereas for a channel \( g \) which appears as argument for the “bcast” primitive, there can be zero or many receivers (depending on the actual number of members in group \( g \)).

### 3. Bisimulations

We define now appropriate tools allowing to reason about processes. Bisimulations have been successfully used in processes algebra to compare two systems according to their operational ability to simulate each other. For the rest of the paper, we shall use the term process to stand for a closed process (which does not contain free identifiers), and we shall specify explicitly open processes.

#### 3.1. Barbed Equivalences

Barbed bisimulation was firstly introduced by Sangiorgi and Milner in ([12], [17]) for the \( \tau \)-calculus. It is a natural relation which is easy definable in various calculi (or rewriting systems): it suffices to define a observability predicate and then to distinguish between two processes whenever they are not similarly observable, or their similar observability is not preserved by reduction (or rewriting). Barbed bisimulations were already used for calculi based on broadcast ([7]), but in a calculus where communications are made on a global ether rather than explicit channels.
Barbed bisimulation describes a relation between processes which can make the same “visible” actions, and whenever one can silently progress, so can the other, evolving to a process preserving the relation. It remains to precise some parts of this informal definition. In a broadcast calculus, outputs are visible (if we are listening a process on a channel, we receive value it sends on it), while inputs are not (as sending is not blocking, we do not know if the observed process was receiving or discarding the value). We write $p \xrightarrow{\alpha} (p \Downarrow_a)$ if $p \xrightarrow{\alpha} p'$ (and respectively $p \xrightarrow{\alpha} p'$) for some output $\alpha$ of subject $a$.

**Definition 4 Barbed Bisimulation**

A symmetric relation $S$ over the set $P_b$ (of processes) is a weak barbed bisimulation if whenever $(p, q) \in S$, then

- if $p \xrightarrow{\alpha} p'$, then $\exists q'$ such that $q \xrightarrow{\alpha} q'$ and $(p', q') \in S$.
- $\forall a \in Ch_b$, if $p \Downarrow_a$ then $q \Downarrow_a$.

Two processes $p$ and $q$ are weak barbed bisimilar, noted $p \approx_b q$, if $(p, q) \in S$ for some barbed bisimulation $S$. The strong barbed bisimilarity $\approx_b$ is defined similar by replacing $\xrightarrow{\alpha}$ by $\xrightarrow{\alpha}$ and $\Downarrow$ by $\Downarrow$.

Because barbed bisimilarity is a quite weak relation (it is not preserved by parallel composition and cannot make any distinction between $\alpha, p$ and $\alpha, q$ for any $p, q, \alpha \neq \tau$), it is necessary to close it over various classes of contexts. A context is a term containing a single “hole”, such that placing another term in the hole defines a valid term. A static context is a context which is build using just “the hole”, constants, parallelism and name creation.

**Definition 5 Barbed Equivalence**

- Two processes $p$ and $q$ are weak barbed equivalent, shortly $p \approx_b q$, if for every static context $C$, $C[p] \approx_b C[q]$.
- Strong barbed equivalence ($\approx_b$) is obtained similarly from strong barbed bisimilarity over processes.

**Remark 1** Contrary to the $\pi$-calculus, in the $b\pi$-calculus it is no more true that $p \approx_b q$ ($p \approx_b q$), implies $\nu \alpha p \approx_b \nu \alpha q$ ($\nu \alpha p \approx_b \nu \alpha q$) for any channel $\alpha$. For example, take $p_0 \xrightarrow{d, f} \tilde{a}b$ and $q_0 \xrightarrow{d, f} \tilde{a}b, \tilde{c}d$ which are barbed bisimilar, but $\nu a p_0$ and $\nu a q_0$ are not.

Because of remark 1, we cannot define barbed equivalence as the closure of barbed bisimilarity with respect to arbitrary testers as in the $\pi$-calculus, and this push us to search for another characterization of barbed bisimulation, still based on observations, but which is preserved by restriction.

We denote $p \Downarrow_a^\phi$ if $p \xrightarrow{\alpha} \xrightarrow{\alpha} p'$ for some output $\alpha$ of subject $a$.

**Definition 6 Step-Bisimulation**

A symmetric relation $S$ over the set $P_b$ (of processes) is a weak step-bisimulation if whenever $(p, q) \in S$, then

- if $p \xrightarrow{\alpha} p'$, then $\exists q'$ such that $q \xrightarrow{\alpha} q'$ and $(p', q') \in S$.
- $\forall a \in Ch_b$, if $p \Downarrow_a$ then $q \Downarrow_a$.

Two processes $p$ and $q$ are weak step-bisimilar, noted $p \approx_b q$, if $(p, q) \in S$ for some step-bisimulation $S$. The strong step-bisimilarity $\approx_b$ is defined similar, but replacing $\xrightarrow{\alpha}$ by $\xrightarrow{\alpha}$ and $\Downarrow$ by $\Downarrow$.

Intuitively, step-bisimilarity identifies processes which can broadcast messages on the same set of channels (immediately or after some autonomous transitions), and whenever one can progress without implicating the environment, so can the other, evolving to a process preserving the relation. In fact, for our calculus, step-bisimulation is more natural than barbed bisimulation since the real reduction is $\xrightarrow{\alpha}$ and not $\xrightarrow{\tau}$ (or $\xrightarrow{\tau}$) as in $\pi$-calculus. Like barbed bisimilarity, step-bisimilarity is a very weak relation (it is not preserved by parallel composition), so it is necessary to close it with respect to observers.

**Definition 7 Step-Equivalence**

- Two processes $p$ and $q$ are weak step-equivalent, shortly $p \approx_\phi q$, if for each process $r$, $p \parallel r \approx_\phi q \parallel r$;
- Strong step-equivalence ($\approx_\phi$) is obtained similarly from strong step-bisimilarity over processes.

Using some convenient contexts, we can prove that barbed equivalence and step-equivalence coincide, that is to say:

**Theorem 1**

1. $\approx_\phi = \approx_b$
2. $\approx_\phi = \approx_\phi$

### 3.2. Labelled Bisimulations

In order to prove that two processes $p$ and $q$ are not (weak) step-equivalent (or weak barbed equivalent), it is enough to find a suitable $r$, such that $p \parallel r$ and $q \parallel r$ are not weak step-bisimilar (respectively suitable static context $C$, such that $C[p]$ and $C[q]$ are not weak barbed bisimilar). But the converse (i.e. proving the step-equivalence (or the barbed equivalence) is much harder to prove in general just using the given definitions; indeed it requires a quantification over all the contexts. Thus, it is interesting to have another way to directly prove that two systems are weak barbed equivalent.
Definition 8. Labelled bisimulations. A symmetric relation $S$ over the set $P_b$ of processes is a (weak) bisimulation if whenever $(p, q) \in S$, then

1) if $p \xrightarrow{\alpha} p'$, then $\exists q'$ such that $q \xrightarrow{\epsilon} q'$ and $(p', q') \in S$,

2) if $p \xrightarrow{\beta} p'$, then $\exists q'$ such that $q \xrightarrow{\beta} q'$ and $(p', q') \in S$,

3) if $p \xrightarrow{\nu \beta} q$, then $\exists q'$ such that $q \xrightarrow{\nu \beta} q'$ and $(p', q') \in S$,

4) if $p \xrightarrow{\alpha(b)} q'$ then $q \xrightarrow{\epsilon(b)} q'$.

Two processes $p$ and $q$ are weak bisimilar, noted $p \approx q$, if $(p, q) \in S$ for some bisimulation $S$.

The strong bisimilarity $\approx$ is defined similar by substituting $\rightarrow$ to $\Rightarrow$ in conditions 1), 2) and 3), replacing condition 4) by “if $p \xrightarrow{\alpha(b)} q'$ then $\exists q'$ such that $q \xrightarrow{\epsilon(b)} q'$ and $(p', q') \in S$”.

We cannot collapse clauses 2) and 3) in Definition 8, since we have to deal with scope extrusion of new names in bound outputs, and this is no more reflected in rule (5) of the operational semantics. Note that in clause 3) we required $(p', q') \in S$ instead of $(\nu \beta p', \nu \beta q') \in S$.

Using similar arguments as in [17], we can prove that labelled bisimulation and labelled bisimilarity coincide for a large class of processes, namely those which are image finite. A process $(P, Act \xrightarrow{\rightarrow})$ is image finite if for any process $p$ and action $\alpha$, the set $\{q \mid p \xrightarrow{\alpha} q\}$ is finite (in our case $\rightarrow$ can stand either for the strong or for the weak reduction). We say that a process $p$ is image finite if the its generated by $p$ is image finite. Remark that with respect to strong reduction, all processes are image finite (up to alpha-conversion).

Theorem 2. For all image finite processes $p$ and $q$.

1) $p \approx_b q$ iff $p \approx_\phi q$ iff $p \approx q$.

2) $p \approx_b q$ iff $p \approx_\phi q$ iff $p \approx q$.

4. Congruences

In this section we focus on the congruence induced by labelled bisimilarity. Our goal will be to define a relation $R$ over the set $P_b$ of processes such that if $p R q$ then $p$ and $q$ are observable on the same set of channels, observability is preserved by reduction, and moreover, when placed in an arbitrary context $C$, $p$ and $q$ cannot be distinguished ($C[p] R C[q]$). In this paper we shall concentrate on the strong congruence, for the weak case (which abstracts for internal steps), we shall defer to future work.

Definition 9. Barbed Congruence. Two processes $p$ and $q$ are barbed congruent, shortly $p \sim_b q$, if for any context $C$, $C[p] \sim_b C[q]$.

Obviously, $\sim_b$ is by definition a congruence. But as for barbed equivalence, this definition is more appropriate to prove the non-congruence rather than the congruence of two processes. So we are interested in finding a direct characterisation of this congruence based on labelled transitions.

Unfortunately, $\sim$ is not the target characterisation, as it follows from the above remark.

Remark 2.

- $\sim$ is not preserved by choice. We have that $a \sim b$, but $a + \overline{c} \not\sim b + \overline{c}$.

- $\sim$ is not preserved by substitution. Let $p \overset{\text{def}}{=} x. y. \overline{c} + y. \overline{c}$ and $q \overset{\text{def}}{=} x. y. \overline{c}$. Then $p \sim q$, but $p[x/y] \not\sim q[x/y]$.

- $\sim$ is not preserved by prefixing. It is a direct consequence of the previous item.

Unlike strong bisimilarity from the $\pi$-calculus, $\sim$ is not preserved by choice. We borrow ideas from [7] and [13] to obtain a congruence relation which do not require closure with respect to contexts.

Definition 10. Let $\sim_+$ be given by

1) if $p \xrightarrow{\alpha} p'$, then $\exists q'$ such that $q \xrightarrow{\epsilon} q'$ and $p' \sim q'$.

2) if $p \xrightarrow{\beta} p'$, then $\exists q'$ such that $q \xrightarrow{\beta} q'$ and $p' \sim q'$.

3) if $p \xrightarrow{\nu \beta} p'$, then $\exists q'$ such that $q \xrightarrow{\nu \beta} q'$ and $p' \sim q'$.

4) if $p \xrightarrow{\alpha} p'$ then $\exists q'$ such that $q \xrightarrow{\epsilon} q'$ and $p' \sim q'$.

Let $\sim$ be given by $p \sim q$ if $p \sigma \sim_+ q \sigma$ for all substitutions $\sigma$.

Remark 3.

- $\sim_+ \subseteq \sim$.

- $\sim_+$ is preserved by choice.

Lemma 3. $\sim_+$ is preserved by prefix, restriction, summation, matching and parallelism.

For a process $E$ which contains a free identifier $X$, and a process $p$, we denote $E(p)$ for the process obtained from $E$ by replacing $X$ by $p$. For example, if $p \overset{\text{def}}{=} x_1, x_2)(x_1, x_2 || x_2)$ and $E \overset{\text{def}}{=} ab. X(a, b) + \nu c. a. X(c, b)$, then $E(p) = ab. (\overline{a} b || \overline{b}) + \nu c. a. (\overline{c} b || \overline{b})$.
Definition 11 Let $E$ and $F$ be two processes which contain a free identifier $X$. Then $E \sim F$ (respectively $E \rightarrow F$, $E \equiv F$ ) means that $E(p) \sim F(p)$ (respectively $E(p) \rightarrow F(p)$, $E(p) \equiv F(p)$ ) for any process $p$.

Lemma 4 Let $E$ and $F$ be open processes which contain $X$ as free identifier. If $E \sim F$, then $\langle rec X \langle \hat{x} \rangle, E \rangle \sim \langle \hat{x} \rangle$ (rec $X \langle \hat{x} \rangle, F \rangle \langle \hat{x} \rangle$).

Corollary 2 $\sim_c$ is preserved by recursion.

Using Lemma 3 and Corollary 2, we obtain that indeed $\sim_c$ is a congruence.

Theorem 3 $\sim_c$ is a congruence.

Using some convenient contexts, we can prove that $\sim_c$ and $\sim^c_\phi$ coincide.

Theorem 4 $\sim_c = \sim^c_\phi$

If we denote by $\sim^c_\phi$ the congruence induced by $\sim_\phi$, then it is easy to prove that $\sim_c = \sim^c_\phi = \sim_\phi$.

5. Axiomatization of strong congruence

In this section we give a complete axiomatisation of strong congruence for finite processes (without recursion). Our axiomatisation is derived from those given for the π-calculus by Parrow and Sangiorgi in [13]. But we need to take care of the fact that strong congruence $\sim_c$ is not directly obtained form strong bisimilarity $\sim_b$ by closure with respect to all substitutions (as for the π-calculus), but from a strictly stronger relation $\sim_\phi$. The gap between $\sim$ and $\sim_\phi$ is filled by the new axiom $(H)$ (which does not hold for strong congruence in π-calculus), which corresponds to the axiom $P = Noisy$ given for CBS in [7].

5.1. Characterising strong congruence over simple processes

In this subsection we restrict our attention to processes given by

\[
p ::= \text{nil} \mid \pi.p \mid p_1 + p_2 \mid \phi p_1.p_2
\]

where $\pi$ belongs to the set of prefixes $\pi ::= \tau \mid x(y) \mid \bar{x}y$, $x, y \in Ch_0$.

Following [13] we use the more general form $\phi p_1.p_2$ with

\[
\phi ::= \langle x = y \rangle \mid \neg \phi \mid \phi_1 \land \phi_2
\]

Table 5. Axiom system $\mathcal{A}$ for strong congruence.

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A)</td>
<td>if $p$ and $q$ are alpha-equivalent, then $p = q$</td>
</tr>
<tr>
<td>(IP)</td>
<td>if $p = q$ then $\alpha.p = \alpha.q$</td>
</tr>
<tr>
<td>(IC)</td>
<td>if $p = q$ then $\phi p = \phi q$</td>
</tr>
<tr>
<td>(IS)</td>
<td>if $p = q$ then $p + r = q + r$</td>
</tr>
<tr>
<td>(H)</td>
<td>if $x \notin fn(p)$ and $\forall \nu \in I_n(p) \phi \Rightarrow \langle a \neq b \rangle$ then $\alpha.p = \alpha.(p + \alpha(a).p)$</td>
</tr>
<tr>
<td>(S1)</td>
<td>$p + nil = p$</td>
</tr>
<tr>
<td>(S2)</td>
<td>$p + p = p$</td>
</tr>
<tr>
<td>(S3)</td>
<td>$p + q = q + p$</td>
</tr>
<tr>
<td>(S4)</td>
<td>$(p + q) + r = p + (q + r)$</td>
</tr>
<tr>
<td>(C3)</td>
<td>if $\phi \iff \psi$ then $\phi p = \psi p$</td>
</tr>
<tr>
<td>(C4)</td>
<td>False $p = False q$</td>
</tr>
<tr>
<td>(C5)</td>
<td>$\phi p, q = \neg \phi q, p$</td>
</tr>
<tr>
<td>(CC1)</td>
<td>$\phi(\psi p) = [\phi \land \psi] p$</td>
</tr>
<tr>
<td>(SC1)</td>
<td>$\phi(p_1 + p_2), (q_1 + q_2) = \phi p_1, q_1 + \phi p_2, q_2$</td>
</tr>
<tr>
<td>(CP1)</td>
<td>$\langle \alpha \rangle n(\alpha) \cap n(\phi) = \emptyset$ then $\phi(\alpha.p) = \phi(\alpha.p)$</td>
</tr>
<tr>
<td>(CP2)</td>
<td>$\langle x = y \rangle \alpha.p = \langle x = y \rangle (\alpha(\langle x/y \rangle)).p$</td>
</tr>
<tr>
<td>(SP)</td>
<td>$\alpha(x).p + \alpha(a).q = \alpha(x).p + \alpha(a).q + \alpha(x).(\langle x = y \rangle.p, q)$</td>
</tr>
</tbody>
</table>

with $x, y \in Ch_0$, and we use the shortcut $\phi p$ to stand for $\phi p, nil$ (respectively $\langle x \neq y \rangle.p, q$ for $\neg(\langle x = y \rangle.p, q)$). We denote by $I_n(p)$ the set of all input ports of $p$ (the set of names $a$ such that $p \xrightarrow{a,x} p'$ for some $p'$).

The axiom system $\mathcal{A}$ for strong congruence $\sim_c$ is given in Table 5.

We write $\mathcal{A} \vdash p = q$ whenever $p = q$ can be proved using the rules of the Table 5. The following theorem is easy to prove:

Theorem 5 (soundness of $\mathcal{A}$ for $\sim_c$) If $\mathcal{A} \vdash p = q$ then $p \sim_c q$.

As in [13], it can be proved that for every process, there exists one equivalent process (in the system of axioms $\mathcal{A}$) which is in “normal form”, and that congruent processes in normal form can be proved equal in our system of axioms.

Definition 12 [13] Let $V$ be a set of names; a condition $\phi$ is complete on $V$ if for some equivalence relation $\mathcal{R}$ on $V$, it holds that $\phi \Rightarrow \langle x = y \rangle$ if $x \mathcal{R} y$, and $\phi \Rightarrow \langle x \neq y \rangle$ if $\neg(\langle x = y \rangle)$.

Definition 13 (head normal form) Let $V$ be a set of names. $p$ is in head normal form on $V$, if it is of the form

\[
\Sigma_{i \in I} \phi_i \alpha_i \Phi_i p_i, \text{ where for all } i,
\]

1. $\forall \alpha_i \notin V$;
(1R) if \( p = q \) then \( \nu x p = \nu x q \)

(\( R \)) \( \nu x \) nil = nil

(\( RR \)) \( \nu v \nu y p = \nu y \nu v p \)

(\( RS \)) \( \nu x (p + q) = \nu x p + \nu x q \)

(\( RP1 \)) if \( x \notin n(\alpha) \) then \( \nu x \alpha p = \alpha . \nu x p \)

(\( RP2 \)) \( \nu x . y . p = \tau . \nu x p \)

(\( RP3 \)) \( \nu x \; x(y).p = \nu y.p \)

(\( RC1 \)) if \( x \neq y \) then \( \nu x (x = y).p = \nu y.p \)

(\( RC2 \)) if \( x \neq y, z \) then \( \nu x (z = y).p = (z = y) \nu x p \)

**Table 6. The axioms for restriction.**

2 \( \phi_i \) is complete on \( V \).

**Lemma 5** For each process \( p \), and for each finite set of names \( V \) with \( \text{fn}(p) \subseteq V \), there is a process \( h \) of no greater depth than \( p \) and in \( \text{hnf} \) on \( V \), such that \( \mathcal{A} \vdash p = h \).

Using a similar reasoning as in \[13\] for the proof of the Theorems 4.9 and 4.11, and using the axiom \( (H) \) when needed, we can prove the following result:

**Theorem 6** (completeness of \( \mathcal{A} \) for \( \sim_c \)) If \( p \sim_c q \) then \( \mathcal{A} \vdash p = q \).

Moreover, our axioms are independent (this follows from the fact that in \[13\] it is proved that all axioms, but \( (H) \), are independent, and that \( (H) \) cannot be proved from the other).

### 5.2. Adding restriction operator

To the grammar given in the previous section, we add the restriction operator:

\[ p ::= \ldots \mid \nu x p \]

The axioms to deal with restriction are given in Table 6.

The only axiom which is new (and which does not hold in the \( \pi \)-calculus) is \( (RP2) \). The soundness of all axioms is easy to prove. For the completeness, the axioms from Table 6. are used to push a restriction inside a term until either it disappears or it gives rise to a bound output. The definition for the normal form changes slightly: \( \Sigma_{i \in I} \phi_i \alpha_i . \phi'_i p_i \), where for all \( i \),

1. \( b n(\alpha_i) \notin V \),
2. \( \phi_i \) is complete on \( V \),
3. \( \phi_i = \phi'_i \) if \( \alpha_i \) is \( \tau \), an input or a free output,
4. \( \phi'_i = \phi_i \land (\forall z \in V (x \neq z)) \).

The proof of the completeness is then similar as for Theorem 6.

Assume

\[ p = \sum_{i \in M_1} \phi_i \; x_i^1[v] . p_{i_1} + \sum_{i_2 \in M_2} \phi_{i_2} . x_{i_2}^2(v) . p_{i_2} \]

and

\[ q = \sum_{j_1 \in N_1} \phi_{j_1} . x_{j_1}^1[v] . q_{j_1} + \sum_{j_2 \in N_2} \phi_{j_2} . x_{j_2}^2(v) . q_{j_2} \]

where \([v] \) stand for \( v \) (free output) or \((v) \) (bound output). Let \( S = \{ x_i \mid i \in M_2 \} \) and \( T = \{ x_i \mid i \in N_2 \} \). Then:

\[ p \parallel q = \sum_{(i_2, j_2) \in S \times T} \phi_{i_2} \land \phi_{j_2} \land (x_{i_2} = x_{j_2}) . x_{i_2}^1(v) . (p_{i_2} \parallel q_{j_2}) + \sum_{(i_1, j_2) \in S \times T} \phi_{i_1} \land \phi_{j_2} \land (x_{i_1} = x_{j_2}) . x_{i_1}^1[v] . (p_{i_1} \parallel q_{j_2}) + \sum_{(i_1, j_1) \in T \times T} \phi_{i_1} \land \phi_{j_1} \land (x_{i_1} = x_{j_1}) . x_{i_1}^1[v] . (p_{i_1} \parallel q_{j_1}) + \sum_{i_1 \in T \mid (x_i \notin T)} \phi_{i_1} \land \phi_{j_1} \land (x_{i_1} \notin T) . x_{i_1}^1[v] . (p_{i_1} \parallel q_{j_1}) + \sum_{j_2 \in T \mid (x_j \notin T)} \phi_{i_2} \land \phi_{j_2} \land (x_{i_2} \notin T) . x_{i_2}^2(v) . (p_{i_2} \parallel q_{j_2}) + \sum_{j_2 \in T \mid (x_j \notin T)} \phi_{j_2} \land \phi_{j_2} \land (x_{i_2} \notin T) . x_{i_2}^2(v) . (p_{i_2} \parallel q_{j_2}) \]

**Table 7. The expansion axiom.**

### 5.3. Adding parallelism

To the grammar given in the previous section, we add the parallel operator:

\[ p ::= \ldots \mid p_1 \parallel p_2 \]

The axioms needed to deal with parallelism are the expansion axiom given in Table 7. plus the axiom \( (P1) \) \( p \parallel \text{nul} = p \).

In the Table 7., the first summand corresponds to the situation where both processes makes an input. The second and the third summands to the situation where one process makes an output, and the other an input. The fourth and the fifth summands to the situation where one process makes an output and the other a discard. And finally, the sixth and the seventh to the situation where one process makes an input and the other a discard. To prove the completeness it suffices to eliminate the operator \( \parallel \) using the expansion axiom and the axiom \( (P1) \).

### 6. Related work and conclusions

Closest related work to this paper concerns the work on CBS by Prasad \[14\], \[15\], and the work by Hennessy and Rathke \[7\]. In \[7\], the authors present a process calculus based on broadcast, together with an operational semantics. They also provide simpler characterisations of the congruence induced by barbed bisimilarity, together with complete
axiomatisation for congruences (for finite processes). Our bisimilarities are following ideas borrowed from their work. However, our calculus focus mainly on the influence of received values (names) by a process on his further possible communications by using a syntax closer to the $\pi$-calculus. Our axiomatisation is thus closer to the one given by Parrow and Sangiorgi [13]. The main difference with existing broadcast calculus is the presence of dynamic scoping (versus static scoping of $CBS$). It is common in concurrent programming to have several groups of processes participating in the same protocol concurrently (using different “channels”). It is then essential that communications be kept separate so that there is no risk of interference between the multiple instances of a protocol executed simultaneously. This is achieved by lexical scoping. Dynamic scoping is then obtained by the combination of local scoping and the ability to send channels along channels.

Concerning the expressiveness of our calculus, it is easy ( [4]) to give an implementation (very similar to those given in [2] for a process algebraic approach of Linda) of a Random Access Machine. Also, it is interesting to compare the $b\pi$-calculus with the $\pi$-calculus. In [3], we have already proved that “there is no uniform encoding of the $b\pi$-calculus into the $\pi$-calculus”. The existence of a “good” (compositional) encoding of the $b\pi$-calculus into the $\pi$-calculus remains an open question. Conversely, we can give an “uniform” encoding adequate with respect to barbed equivalence of the $\pi$-calculus into the $b\pi$-calculus.

Also, even if bisimulations provide a nice method to prove the relation which holds between two equivalents systems (just looking at their states, without building the whole traces set), we can ask if they are not too restrictive? For example, $\bar{a} \cdot (\bar{b} + \bar{c})$ and $\bar{a} \cdot \bar{b} + \bar{a} \cdot \bar{c}$ are not barbed equivalents. This seems surprising, as in our calculus an observer cannot influence the behaviour of the two processes, nor it can distinguish them; indeed, this is the case in processes algebra based on point-to-point communications ($CCS$, $\pi$-calculus), where an observer provides to tested process the necessary “co-actions”. In a forthcoming paper we analyse the preorders induced by “may testing” in calculi based on broadcast.

Acknowledgements. We have gratefully appreciated the useful comments received from the referees.

References


