On the Expressiveness of Chi, Update, and Fusion calculi [1]

Massimo Merro [2]

INRIA, 2004 Rue des Lucioles, B.P. 93, 06902 Sophia-Antipolis, France.

Abstract

Chi and *Update* calculi [9,17] have been independently introduced in order to model *mobile systems*. The two calculi are very close to each other and represent an evolution of π-*calculus* [15]. More recently a (non-straightforward) polyadic version of the Update calculus, the *Fusion* calculus, has been proposed [18].

In the paper we give a *fully abstract encoding* from an asynchronous variant of Chi and Update calculi to asynchronous π-calculus [11,4]. This proves that, at least for their asynchronous variants, Chi and Update calculi are not more expressive than π-calculus. A similar result can be proved for the Fusion calculus.

1 Introduction

Chi and *Update* calculi [9,17] have been independently introduced in order to model *mobile systems*, that is, systems with a dynamically changing communication topology. They represent an evolution of π-*calculus* [15], the standard calculus of mobile processes where linkage changing is achieved by using the *name-passing* paradigm. More recently a (non-straightforward) polyadic version of the Update calculus, the *Fusion* calculus, has been proposed [18].

The Chi calculus has essentially two motivations: (i) to overcome the limits of the input operator in π-calculus, due to its blocking nature; (ii) to model communication-as-cut-elimination techniques in concurrency theory. The Update calculus shares the first motivation but has a quite different second one, that is, to model assignment to shared structures in concurrent constraint languages [24].

[1] This research has been supported by INRIA Sophia-Antipolis and France Télécom, CTI-CNET 95-1B-182 Modélisation de Systémes Mobiles.

[2] E-mail: mmerro@sophia.inria.fr

This is a preliminary version. The final version can be accessed at
URL: http://www.elsevier.nl/locate/entcs/volume16.html

Chi and Update calculi are very close to each other and in the remainder of the paper we shall suppose to deal with the same calculus, in short CUP. The principal novelties in CUP w.r.t. π-calculus are that: (i) the input primitive does not bind names and (ii) there exists only one binding operator, written $(x)P$, to mean that name x in P is local. A pleasant property of CUP is the full symmetry between input and output constructs like in Sangiorgi's πI-calculus [21].

Communication in π-calculus affects only its participants while in CUP-calculus it may involve other processes. To better understand what this means, let us compare the basic interaction in π and CUP calculi, respectively:

$$\begin{array}{ll} \pi\text{-calculus:} & \overline{a}b.\, P \mid a(x).\, Q \mid R \longrightarrow P \mid Q\{b/x\} \mid R \\ \text{CUP:} & (x)(\overline{a}b.\, P \mid ax.\, Q \mid R) \longrightarrow (P \mid Q \mid R)\{b/x\} \end{array}$$

In the first case the effect of the communication (i.e. the substitution) is confined to the receiver process Q; in CUP, on the contrary, all the processes in the scope of (x) are updated.

An obvious encoding from π into Chi has been suggested in [9]. So, in some sense, the claim that Chi and Update calculi inherit all π-calculus' expressive power is reasonable. It remains to understand whether Chi and Update calculi are more expressive than π-calculus. In [17,18] Parrow and Victor give a couples of examples, which have no direct counterpart in π-calculus, to argue that Update and Fusion calculi have a greater expressive power than π-calculus and its polyadic version, respectively. The examples are essentially the following:

(i) In π-calculus there is no a direct way to encode non-blocking message reception, that is, a construct $a(x)P$ which allows the continuation P to evolve underneath the input guard (except for observable actions along the placeholder x). Similar forms of *delayed input* have been studied in [3,5,6,25,13]. Such construct would allow to model "strong" notions of reduction (where computations may be performed within any context) like the *strong* reduction strategy in *lazy* lambda calculus. The delayed input can be easily encoded in CUP calculus as:

$$a(x)P \stackrel{\text{def}}{=} (x)(ax \mid P).$$

(ii) The Fusion calculus allows the same name to be received twice. This implies the "fusion" of the received names, as shown by the following example:

$$(b)((x)axx \mid \overline{a}bc \mid \overline{b}h) \longrightarrow \overline{c}h.$$

Also in this case there is no a direct counterpart in polyadic π-calculus.

In the paper we define a fully abstract encoding from an asynchronous variant of CUP-calculus to asynchronous π-calculus [4,11]. A straightforward polyadic variant of such encoding can be used to encode asynchronous Fusion calculus into asynchronous polyadic π-calculus [14]. So we prove that asynchronous Chi, Update, and Fusion calculi are not more expressive than asynchronous π-calculus.

As to synchronous Chi, Update, and Fusion calculi, Honda and Tokoro [11], and independently Boudol [4], have shown that asynchronous π-calculus is expressive enough to encode synchronous π-calculus by providing a sound (but not complete) encoding. We believe that a similar encoding can be given for Chi, Update, and Fusion calculi. So we conjecture that asynchronous π-calculus is expressive enough to encode synchronous Chi, Update and Fusion calculi.

2 The asynchronous π-calculus without matching

The *asynchronous π-calculus* is a variant of π-calculus where message emission is non-blocking. Formally, the output prefix $\overline{a}b.\,P$ of π-calculus is replaced by the simpler output particle $\overline{a}b$, which has no continuation. The asynchronous π-calculus has been independently introduced by Honda and Tokoro [11] and Boudol [4], who showed that it is expressive enough to encode the (synchronous) π-calculus.

Asynchronous communications are interesting from the point of view of concurrent and distributed programming languages, because they are easier to implement and they are closer to the communication primitives offered by available distributed systems.

In this section we present a sub-calculus of the asynchronous π-calculus without match nor summation operators. This sub-calculus, in short π_{a}, is a very expressive fragment of π-calculus. At the practical level, this expressiveness gives rise to the useful high-level concurrent programming language Pict [19], which is basically built on the polyadic version of this calculus with a strong typing system.

2.1 Syntax and semantics

The grammar of π_{a} has operators of inaction, input prefix, asynchronous output, parallel composition, restriction and replicated input:

$$P ::= \mathbf{0} \;\Big|\; a(x).\,P \;\Big|\; \overline{a}b \;\Big|\; P \mid P \;\Big|\; (a)P \;\Big|\; !a(x).\,P$$

We use small letters $a, b, \ldots, x, y$ for *names*; capital letters P, Q, R for *processes*; σ for substitutions; $P\sigma$ is the result of applying σ to P, with the usual renaming convention to avoid captures. We write $(\widetilde{x})P$ for $(x_1)\ldots(x_n)P$. Parallel composition has the lowest precedence among the operators. Free and bound names (fn, bn) of processes are defined as usual. In processes $a(b).\,P$ and $\overline{a}b$ we say that a appears in subject position and b in object position.

We focus on bisimulation-based behavioural equivalences, precisely on *barbed congruence* [16] (studies of bisimulation-based behavioural equivalences for asynchronous mobile calculi are [11,12,10,1]). It is well-known that barbed bisimilarity represents a uniform mechanism for defining behavioural equivalences in any process calculus possessing (i) a *reduction relation* and (ii) an *observability predicate* which simply detects the possibility of performing some observable action. We define the *reduction semantics* of π_{a} in two steps. First, we identify several processes by giving a *structural congruence* over processes. Second, we define the computation steps of processes in terms of a reduction relation.

Definition 2.1 The *structural congruence* $\equiv$ is defined as the smallest congruence relation over processes satisfying:

(i) if P and Q are alpha-convertible then $P \equiv Q$

(ii) $P\,|\,(Q\,|\,R) \equiv (P\,|\,Q)\,|\,R,\ P\,|\,Q \equiv Q\,|\,P,\ P\,|\,\mathbf{0} \equiv P$

(iii) $(x)\mathbf{0} \equiv \mathbf{0},\ (x)(y)P \equiv (y)(x)P$

(iv) $(x)(P\,|\,Q) \equiv P\,|\,(x)Q$ if x not free in P.

Definition 2.2 The *reduction relation* $\longrightarrow$ is defined as the smallest relation over processes satisfying:

(i) $\texttt{(comm)}: a(x).\,P \mid \overline{a}y \longrightarrow P\{y/x\}$

(ii) $\texttt{(repl)}: !a(x).\,P \mid \overline{a}y \longrightarrow\ !a(x).\,P \mid P\{y/x\}$

(iii) $\texttt{(par)}: \dfrac{P \longrightarrow P'}{P|Q \longrightarrow P'|Q}$

(iv) $\texttt{(res)}\dfrac{P \longrightarrow P'}{(x)P \longrightarrow (x)P'}$

(v) $\texttt{(s-cong)}: \dfrac{P \equiv Q \qquad Q \longrightarrow Q' \qquad Q' \equiv P'}{P \longrightarrow P'}$

We write $\Longrightarrow$ for the reflexive and transitive closure of $\longrightarrow$.

The definition of observability is as usual for asynchronous calculi [1].

Definition 2.3

(i) $\overline{a}b \downarrow_a$

(ii) $P\,|\,Q \downarrow_a$ if $P \downarrow_a$ or $Q \downarrow_a$

(iii) $(x)P \downarrow_a$ if $P \downarrow_a$ and $a \neq x$.

$P \Downarrow_a$ if there exists P' s.t. $P \Longrightarrow P'$ and $P' \downarrow_a$.

We repeat the standard definition of barbed bisimulation from [16].

Definition 2.4 (barbed bisimulation) Barbed bisimulation, written $\dot{\approx}$, is the largest symmetric relation on π_{a}-processes s.t. $P \dot{\approx} Q$ implies:

(i) If $P \longrightarrow P'$ then $\exists\, Q'$ s.t. $Q \Longrightarrow Q'$ and $P' \dot{\approx} Q'$.

(ii) If $P \downarrow_a$ then $Q \Downarrow_a$.

Barbed bisimulation equips a global observer with a minimal ability to observe actions and/or process states but it is not preserved by the operators of the calculus. By closing barbed bisimulation under contexts we obtain a finer relation (a *context* $C[\cdot]$ is a process with a single hole $[\cdot]$ in it).

Definition 2.5 (barbed congruence) Two π_{a}-processes P and Q are *barbed congruent*, written $\cong$, if for each π_{a}-context $C[\cdot]$ it holds that $C[P] \stackrel{\bullet}{\approx} C[Q]$.

In the technical part of the paper we shall need a tool to count the number of reductions performed by a process in order to use up-to techniques [23]. We write $P \lesssim Q$, if $C[P] \stackrel{\bullet}{\approx} C[Q]$ holds for each context $C[\cdot]$, and also $C[Q]$ has at least as many reductions as $C[P]$ (the relation $\lesssim$ is related to $\cong$ in the same way as *expansion* is related to *weak bisimilarity* [2,23]).

2.2 Equators

Equators are special processes, first defined by Honda and Yoshida [12] as follows:

$$\mathcal{E}(a,b) \stackrel{\mathrm{def}}{=} !a(x).\,\overline{b}x \mid !b(x).\,\overline{a}x$$

Equators induce a sort of equivalence relation over names.

Lemma 2.6

(i) $\mathcal{E}(a,a) \ \gtrsim\ \mathbf{0}$

(ii) $\mathcal{E}(a,b) \ \equiv\ \mathcal{E}(b,a)$

(iii) $(c)(\mathcal{E}(a,c) \mid \mathcal{E}(c,b)) \ \gtrsim\ \mathcal{E}(a,b)$.

In the asynchronous π-calculus without matching, a further interesting law involving equators holds:

$$(a)(P \mid \mathcal{E}(a,b)) \ \gtrsim\ P\{{}^{b}\!/\!{}_{a}\} \tag{1}$$

This law can be proved by extending the theory developed in [13]. In that paper we give a labeled characterisation of barbed congruence in local asynchronous π-calculus Lπ, a variant of π_{a} where only the output capability of names may be transmitted, that is, the recipient of a name may only use it in output actions. In such theory a central role is played by the *link* process $!a(x).\,\overline{b}x$ through which outputs along a are redirected onto b. This directionality reflects the output capability hypothesis. The theory developed in [13] can be extended to (full) π_{a} by replacing the directional link process $!a(x).\,\overline{b}x$ with the bidirectional equator process $\mathcal{E}(a,b)$.

Law 1 has already been proved w.r.t. the *max sound theory*, a relation similar to barbed congruence [12]. More recently, Fournet and Gonthier have proved that, in asynchronous π-calculus, max sound theory and barbed congruence coincide [8]. Notice that the presence of either output prefix or matching would invalidate Law 1.

3 The asynchronous CUP-calculus without matching

Chi and Update calculi are very close to each other and, up-to some minor features, we can suppose to deal with the same calculus, in short CUP. In this section we define an asynchronous variant of CUP without match nor summation operators, in short $\mathrm{CUP_a}$. As π_a such variant is a powerful subcalculus.

3.1 Syntax and semantics

The grammar of $\mathrm{CUP_a}$ has operators of inaction, *unbound input prefix*, asynchronous output, parallel composition, restriction and replicated bound input:

$$P ::= \mathbf{0} \;\Big|\; ax.\,P \;\Big|\; \overline{a}b \;\Big|\; P \mid P \;\Big|\; (a)P \;\Big|\; !(x)ax.\,P$$

Conventions about names are as in π_a, except for the unbound input prefix for which we have

$$\mathrm{fn}(ax.\,P) \stackrel{\mathrm{def}}{=} \{a, x\} \cup \mathrm{fn}(P) \;\text{ and }\; \mathrm{bn}(ax.\,P) \stackrel{\mathrm{def}}{=} \mathrm{bn}(P).$$

We write $(\widetilde{x})P$ for $(x_1)\ldots(x_n)P$. Conventions about processes and substitutions are as in π_a.

Following [24,9] we give the *reduction semantics* of $\mathrm{CUP_a}$. As in π-calculus, we have a notion of structural congruence (essentially the same as Definition 2.1); and a reduction relation similar to Definition 2.2 except for rule `(comm)` which is replaced by the following two rules:

$$\texttt{(comm1)}:\; (x)(P \mid ax.\,Q \mid \overline{a}y) \longrightarrow (x)((P \mid Q)\{^{y}\!/\!_{x}\})$$

$$\texttt{(comm2)}:\; (y)(P \mid ax.\,Q \mid \overline{a}y) \longrightarrow (y)((P \mid Q)\{^{x}\!/\!_{y}\})$$

Notice that the restrictions (x) and (y) in the derivatives make sense only when $x = y$, otherwise, up-to structural congruence, they disappear.

The definition of observability and barbed bisimulation are essentially the same as Definitions 2.3 and 2.4.

Remark 3.1 For avoiding heavy notations we use symbols $\longrightarrow$, $\Longrightarrow$, $\downarrow_a$, $\Downarrow_a$, $\dot{\approx}$ both for π_a and $\mathrm{CUP_a}$ processes.

In Chi and Update calculi, communications can arise only in presence of a scoping construct delimiting their effects (see reduction rules `(comm1)` and `(comm2)`). So, to observe all potential communications (and their effects) we consider a notion of barbed congruence obtained by closing barbed bisimulation under contexts that bind all free names of the tested processes. Similar *closing contexts* have been used in typed calculi [22]. As in the definition of *testing equivalences* [7], our contexts signals success by emitting along names that do not appear in the tested processes.

Definition 3.2 (closed barbed congruence) Two $\mathrm{CUP_a}$-processes P and Q are *closed barbed congruent*, written $\cong_\mathrm{c}$, if for each $\mathrm{CUP_a}$-context $C[\cdot]$ s.t.

(i) $\mathrm{fn}(P) \cap \mathrm{fn}(C[P]) = \emptyset$ and $\mathrm{fn}(Q) \cap \mathrm{fn}(C[Q]) = \emptyset$

(ii) if $a \in \mathrm{fn}(C[\cdot])$ then a appears in $C[\cdot]$ only in output subject position;

it holds that $C[P] \;\dot{\approx}\; C[Q]$.

It is immediate to adapt this definition to other calculi, such as π-calculus. We add clause (ii) because in asynchronous semantics only output actions are

Table 1
Encoding from $\mathrm{CUP_a}$ to π_a.

$\langle\!\vert\, P_1 \mid P_2 \,\vert\!\rangle \stackrel{\mathrm{def}}{=} \langle\!\vert\, P_1 \,\vert\!\rangle \mid \langle\!\vert\, P_2 \,\vert\!\rangle$	$\langle\!\vert\, (a)P \,\vert\!\rangle \stackrel{\mathrm{def}}{=} (a)\langle\!\vert\, P \,\vert\!\rangle$
$\langle\!\vert\, !(x)ax.P \,\vert\!\rangle \stackrel{\mathrm{def}}{=} !a(x).\, \langle\!\vert\, P \,\vert\!\rangle$	$\langle\!\vert\, \overline{a}b \,\vert\!\rangle \stackrel{\mathrm{def}}{=} \overline{a}b$
$\langle\!\vert\, ax.P \,\vert\!\rangle \stackrel{\mathrm{def}}{=} a(z).\, (\mathcal{E}(z,x) \mid \langle\!\vert\, P \,\vert\!\rangle) \quad z \notin \mathrm{fn}(P)$	$\langle\!\vert\, \mathbf{0} \,\vert\!\rangle \stackrel{\mathrm{def}}{=} \mathbf{0}$

observable [1]. The names in clause (ii) appear only in output subject position so that they cannot be acquired by the tested processes.

In π-calculus and CCS, closed barbed congruence coincides with barbed congruence. Actually, one can prove that closed barbed congruence coincide with the closure under substitutions of early bisimulation, by adapting the proofs in [20,1]. Since the closure under substitutions of early bisimulation is known to coincide with barbed congruence, the two definitions of barbed congruence coincide.

4 Encoding $\mathrm{CUP_a}$ into π_a

In this section we provide a fully abstract encoding (w.r.t. closed barbed congruence) from $\mathrm{CUP_a}$ to π_a. In this way we prove that asynchronous π-calculus has at least the same expressive power as the asynchronous variant of Chi and Update calculi. For simplicity our encoding deals with monadic calculi; the extension to polyadic calculi as Fusion calculus is straightforward. The encoding $\langle\!\vert\;\vert\!\rangle$ from $\mathrm{CUP_a}$ to π_a is formally defined in Table 1. Lemma 4.1 gives some trivial (but useful) properties of $\langle\!\vert\;\vert\!\rangle$.

Lemma 4.1 *Let P be a process in* $\mathrm{CUP_a}$. *Then*

(i) $\mathrm{fn}(P) = \mathrm{fn}(\langle\!\vert\, P \,\vert\!\rangle)$
(ii) $\langle\!\vert\, P\{y/x\} \,\vert\!\rangle = \langle\!\vert\, P \,\vert\!\rangle\{y/x\}$
(iii) $P \equiv Q$ *implies* $\langle\!\vert\, P \,\vert\!\rangle \equiv \langle\!\vert\, Q \,\vert\!\rangle$.

Proof. Parts 1 and 2 are proved by structural induction. Part 3 is proved by case analysis on the rules defining structural congruence.

Lemma 4.2 (Operational correspondence) *Let P be a process in* $\mathrm{CUP_a}$ *s.t. all its free names appear only in output subject position. Then*

(i) *If $P \longrightarrow P'$ then there exists $P_1 \in \pi_\mathrm{a}$ s.t. $\langle\!\vert\, P \,\vert\!\rangle \longrightarrow P_1 \gtrsim \langle\!\vert\, P' \,\vert\!\rangle$.*
(ii) *If $\langle\!\vert\, P \,\vert\!\rangle \longrightarrow P_1$ then there exists $P' \in$* $\mathrm{CUP_a}$ *s.t. $P \longrightarrow P'$ and $P_1 \gtrsim \langle\!\vert\, P' \,\vert\!\rangle$*
(iii) $P \downarrow_a$ iff $\langle\!\vert\, P \,\vert\!\rangle \downarrow_a$.

Proof.

(i) Recall that all free names in P appear only in output subject position. We have four cases:

(a) $P \equiv (\widetilde{w})(y)(x)(Q \,|\, ax.\, R \,|\, \overline{a}y)$ where $x \neq y$, $a \in \widetilde{w}$ and by rule `(comm1)` $P' \equiv (\widetilde{w})(y)((Q \mid R)\{y/x\})$. Then by Lemma 4.1(iii)

$$\begin{array}{lll} \langle\!| P |\!\rangle \equiv & (\widetilde{w})(y)(x)(\langle\!| Q |\!\rangle \mid a(z).\,(\mathcal{E}(z,x) \mid \langle\!| R |\!\rangle) \mid \overline{a}y) & (z \text{ fresh}) \\ \longrightarrow & (\widetilde{w})(y)(x)(\langle\!| Q |\!\rangle \mid \mathcal{E}(y,x) \mid \langle\!| R |\!\rangle) & \\ \gtrsim & (\widetilde{w})(y)((\langle\!| Q |\!\rangle \mid \langle\!| R |\!\rangle)\{y/x\}) & (\text{Law } 1) \\ \equiv & \langle\!| P' |\!\rangle. & (\text{Lemmas 4.1(ii) and 4.1(iii)}) \end{array}$$

(b) $P \equiv (\widetilde{w})(x)(y)(Q \,|\, ax.\, R \,|\, \overline{a}y)$ where $x \neq y$, $a \in \widetilde{w}$ and by rule `(comm2)` $P' \equiv (\widetilde{w})(x)((Q \mid R)\{x/y\})$. This case is similar to the previous one.

(c) $P \equiv (\widetilde{w})(x)(Q \mid ax.\, R \mid \overline{a}x)$ where $a \in \widetilde{w}$ and by rule `(comm1)` (or `(comm2)`) $P' \equiv (\widetilde{w})(x)(Q \mid R)$. Then by Lemma 4.1(iii)

$$\begin{array}{lll} \langle\!| P |\!\rangle \equiv & (\widetilde{w})(x)(\langle\!| Q |\!\rangle \mid a(z).\,(\mathcal{E}(z,x) \mid \langle\!| R |\!\rangle) \mid \overline{a}x) & \\ \longrightarrow & (\widetilde{w})(x)(\langle\!| Q |\!\rangle \mid \mathcal{E}(x,x) \mid \langle\!| R |\!\rangle) & \\ \gtrsim & (\widetilde{w})(x)(\langle\!| Q |\!\rangle \mid \langle\!| R |\!\rangle) & (\text{Lemma 2.6(i)}) \\ \equiv & \langle\!| P' |\!\rangle. & \end{array}$$

(d) $P \equiv (\widetilde{w})(y)(Q \mid !(x)ax.\, R \mid \overline{a}y)$ where $a \in \widetilde{w}$ and, by rule `(repl)` $P' \equiv (\widetilde{w})(y)(Q \,|\, !(x)ax.\, R \,|\, R\{y/x\})$. Then by Lemma 4.1(iii)

$$\begin{array}{lll} \langle\!| P |\!\rangle \equiv & (\widetilde{w})(y)(\langle\!| Q |\!\rangle \mid !a(x).\,\langle\!| R |\!\rangle \mid \overline{a}y) & \\ \longrightarrow & (\widetilde{w})(y)(Q \mid !a(x).\,\langle\!| R |\!\rangle \mid \langle\!| R |\!\rangle\{y/x\}) & (\text{Lemma 4.1(ii)}) \\ \equiv & \langle\!| P' |\!\rangle. & \end{array}$$

(ii) It is similar to part 1 but we need the hypothesis on P for applying reduction rules `(comm1)` and `(comm2)`.

(iii) It can be easily proved by structural induction.

We need a weak version of the previous lemma.

Lemma 4.3 *Let P be a process in* $\mathrm{CUP_a}$ *s.t. all its free names appear only in output subject position. Then*

(i) *If $P \Longrightarrow P'$ then $\exists\, P_1$ s.t. $\langle\!| P |\!\rangle \Longrightarrow P_1 \gtrsim \langle\!| P' |\!\rangle$*

(ii) *If $\langle\!| P |\!\rangle \Longrightarrow P_1$ then $\exists\, P'$ s.t. $P \Longrightarrow P'$ and $P_1 \gtrsim \langle\!| P' |\!\rangle$*

(iii) $P \Downarrow_a$ iff $\langle\!| P |\!\rangle \Downarrow_a$

Proof. By Lemma 4.2 and transitivity of $\gtrsim$.

Lemma 4.4 *Let P and Q be two processes in* $\mathrm{CUP_a}$ *s.t. all their free names appear only in output subject position. Then*

$$P \stackrel{\bullet}{\approx} Q \text{ iff } \langle\!| P |\!\rangle \stackrel{\bullet}{\approx} \langle\!| Q |\!\rangle$$

Proof. In the "if" part we use Lemma 4.3 to prove that the relation $\mathcal{R} = \{(\langle\!| P |\!\rangle, \langle\!| Q |\!\rangle) : P \stackrel{\bullet}{\approx} Q\}$ is a barbed bisimulation up-to $\gtrsim$ [23]. In the "only if" part we use Lemma 4.3 and the fact that $\lesssim \stackrel{\bullet}{\approx} \gtrsim\ \subset\ \stackrel{\bullet}{\approx}$ to prove that the relation $\mathcal{R} = \{(P, Q) : \langle\!| P |\!\rangle \stackrel{\bullet}{\approx} \langle\!| Q |\!\rangle\}$ is a barbed bisimulation.

The following lemma is crucial for proving the *completeness* of our encoding.

Lemma 4.5 *For each process $P \in \pi_a$ there exists $\widehat{P} \in \mathrm{CUP}_a$ s.t. $\langle\!|\, \widehat{P} \,|\!\rangle \gtrsim P$.*

Proof. By structural induction. The most interesting case is when $P \equiv a(x).\, P_1$. We choose $\widehat{P} \stackrel{\text{def}}{=} (x)ax.\, \widehat{P_1}$ where $\langle\!|\, \widehat{P_1} \,|\!\rangle \gtrsim P_1$ by inductive hypothesis. Then

$$
\begin{array}{rll}
\langle\!|\, \widehat{P} \,|\!\rangle & = (x)(a(z).\, (\mathcal{E}(z,x) \mid \langle\!|\, \widehat{P_1} \,|\!\rangle)) & (\text{with } z \notin \mathrm{fn}(\widehat{P_1})) \\
& \equiv (z)(a(x).\, (\mathcal{E}(z,x) \mid \langle\!|\, \widehat{P_1} \,|\!\rangle\{z/x\})) & (\text{alpha-conversion}) \\
& \gtrsim a(x).\, (z)(\mathcal{E}(z,x) \mid \langle\!|\, \widehat{P_1} \,|\!\rangle\{z/x\}) & (\text{easy law for } \gtrsim) \\
& \gtrsim a(x).\, \langle\!|\, \widehat{P_1} \,|\!\rangle & (\text{Law 1}) \\
& \gtrsim a(x).\, P_1 & (\text{inductive hypothesis}) \\
& \equiv P. &
\end{array}
$$

Theorem 4.6 (Full-abstraction of $\langle\!|\ |\!\rangle$) *Let P and Q be two processes in CUP_a. Then*

$$P \cong_c Q \ \text{ iff } \ \langle\!|\, P \,|\!\rangle \cong_c \langle\!|\, Q \,|\!\rangle.$$

Proof. The soundness part follows by Lemma 4.4 and compositionality of $\langle\!|\ |\!\rangle$. As to the completeness part, let $C[\cdot]$ be a closing context for $\langle\!|\, P \,|\!\rangle$ and $\langle\!|\, Q \,|\!\rangle$ in π_a. We want to prove that $C[\langle\!|\, P \,|\!\rangle] \stackrel{\bullet}{\approx} C[\langle\!|\, Q \,|\!\rangle]$. By Lemma 4.5 there exists a closing context $\widehat{C}[\cdot]$ for P and Q in CUP_a s.t. $\langle\!|\, \widehat{C}[\cdot] \,|\!\rangle \gtrsim C[\cdot]$. By hypothesis, $\widehat{C}[P] \stackrel{\bullet}{\approx} \widehat{C}[Q]$. By Lemma 4.4 this implies $\langle\!|\, \widehat{C}[P] \,|\!\rangle \stackrel{\bullet}{\approx} \langle\!|\, \widehat{C}[Q] \,|\!\rangle$. So, by compositionality of $\langle\!|\ |\!\rangle$, we have $C[\langle\!|\, P \,|\!\rangle] \lesssim \langle\!|\, \widehat{C}[P] \,|\!\rangle \stackrel{\bullet}{\approx} \langle\!|\, \widehat{C}[Q] \,|\!\rangle \gtrsim C[\langle\!|\, Q \,|\!\rangle]$. Since $\lesssim \stackrel{\bullet}{\approx} \gtrsim \ \subset \ \stackrel{\bullet}{\approx}$ we have the thesis.

5 Conclusions

We have introduced an encoding from asynchronous Chi and Update calculi to asynchronous π-calculus. A straightforward polyadic variant of such encoding can be used to encode asynchronous Fusion calculus into asynchronous polyadic π-calculus. Our encoding is fully abstract w.r.t. closed barbed congruence, a notion of barbed congruence that allows us to observe all potentially communications in Chi, Update and Fusion calculi. So we have proved that asynchronous Chi and Update calculi are not more expressive than asynchronous π-calculus, and similarly that asynchronous Fusion calculus is not more expressive than asynchronous polyadic π-calculus.

As to synchronous Chi, Update and Fusion calculi, Honda and Tokoro [11], and independently Boudol [4], have shown that asynchronous π-calculus is expressive enough to encode synchronous π-calculus by providing a sound (but not complete) encoding. We believe that a similar encoding can be given for Chi, Update and Fusion calculi. So we conjecture that asynchronous π-calculus is expressive enough to encode synchronous Chi, Update and Fusion calculi.

Finally, by using our encoding, we give the π-calculus counterpart of the two examples discussed in the introduction.

(i) By applying our encoding to $(x)(ax \mid P)$ we obtain the following definition for the *delayed input* in π-calculus:

$$a(x)P \stackrel{\text{def}}{=} (x)(a(z).\,\mathcal{E}(z,x) \mid P) \qquad \text{where } z \notin \text{fn}(P)$$

A similar encoding has already been suggested in [25]. A variant of this encoding, where the equator process $\mathcal{E}(z,x)$ is replaced by the *link* process $!x(h).\,\overline{z}h$, has been proved fully abstract w.r.t. barbed congruence in local asynchronous π-calculus Lπ, a sub-calculus of π_{a} where only the output capability of names may be transmitted, that is, the recipient of a name may only use it in output actions [13].

(ii) The "fusion" of different names showed by means of the example

$$(b)((x)axx \mid \overline{a}bc \mid \overline{b}h) \longrightarrow \overline{c}h.$$

can be expressed in polyadic π-calculus by using the straightforward extension of our encoding to polyadic Fusion:

$$\begin{array}{rcll} \langle\!|\, (b)((x)axx \mid \overline{a}bc \mid \overline{b}h)\,|\!\rangle & = & (b)((x)a(z,y).\,(\mathcal{E}(z,x) \mid \mathcal{E}(y,x)) \mid \overline{a}bc \mid \overline{b}h) & \\ & \longrightarrow & (b)((x)(\mathcal{E}(b,x) \mid \mathcal{E}(c,x)) \mid \overline{b}h) & \\ & \gtrsim & (b)(\mathcal{E}(b,c) \mid \overline{b}h) & \text{(Lemma 2.6(iii))} \\ & \gtrsim & \overline{c}h & \text{(Law 1)} \end{array}$$

Acknowledgement

Many thanks to Gérard Boudol, Franck van Breugel, Ilaria Castellani, Davide Sangiorgi, Nobuko Yoshida, and Björn Victor for stimulating and insightful discussions. The anonymous referees provided useful suggestions.

References

[1] R. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous π-calculus. In *Proc. CONCUR '96*, volume 1119 of *Lecture Notes in Computer Science*. Springer Verlag, 1996.

[2] S. Arun-Kumar and M. Hennessy. An efficiency preorder for processes. *Acta Informatica*, 29:737–760, 1992.

[3] G. Bellin and P. Scott. On the π-calculus and Linear Logic. *Theoretical Computer Science*, 135(1):11–65, December 1994.

[4] G. Boudol. Asynchrony and the π-calculus. Technical Report RR-1702, INRIA-Sophia Antipolis, 1992.

[5] G. Boudol. Some Chemical Abstract Machines. In *Proc. Rex School/Symposium 1993 "A Decade of Concurrency — Reflexions and Perspectives"*, volume 803 of *Lecture Notes in Computer Science*, pages 92–123. Springer Verlag, 1994.

[6] F. van Breugel. A Labelled Transition System for the $\pi\epsilon$-calculus. In *Proc. of the 7th International Joint Conference on the Theory and Practice of Software Development*, volume 1214 of *Lecture Notes in Computer Science*, pages 312–336. Springer Verlag, April 1997.

[7] R. De Nicola and R. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[8] C. Fournet and G. Gonthier. A Hierarchy of Equivalences for Asynchronous Calculi. In the Proceedings of ICALP'98, volume 1443 of *Lecture Notes in Computer Science*, Springer Verlag, 1998.

[9] Y. Fu. A proof theoretical approach to communication. In *24th ICALP*, volume 1256 of *Lecture Notes in Computer Science*. Springer Verlag, 1997.

[10] M. Hansen, H. Hüttel, and J. Kleist. Bisimulations for asynchronous mobile processes. In *Proc. Tbilisi Symposium on Language, Logic, and Computation*, 1996. Also available as BRICS Report No. EP-95-HHK, BRICS, Aalborg University, Denmark 1996.

[11] K. Honda and M. Tokoro. An Object Calculus for Asynchronous Communications. In M. Tokoro, O. Nierstrasz, P. Wegner, and A. Yonezawa, editors, *ECOOP '91 Workshop on Object Based Concurrent Programming,* Geneva, Switzerland, 1991 , volume 512 of *Lecture Notes in Computer Science*, pages 133–147. Springer Verlag, 1991.

[12] K. Honda and N. Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 152(2):437–486, 1995.

[13] M. Merro and D. Sangiorgi. On asynchrony in name-passing calculi. In the Proceedings of ICALP'98, volume 1443 of *Lecture Notes in Computer Science*, Springer Verlag, 1998.

[14] R. Milner. The polyadic π-calculus: a tutorial. Technical Report ECS–LFCS–91–180, LFCS, Dept. of Comp. Sci., Edinburgh Univ., October 1991. Also in *Logic and Algebra of Specification*, ed. F.L. Bauer, W. Brauer and H. Schwichtenberg, Springer Verlag, 1993.

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

[16] R. Milner and D. Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *19th ICALP*, volume 623 of *Lecture Notes in Computer Science*, pages 685–695. Springer Verlag, 1992.

[17] J. Parrow and B. Victor. The Update calculus. In *Proc. AMAST '97*, volume 1349 of *Lecture Notes in Computer Science*. Springer Verlag, 1997.

[18] J. Parrow and B. Victor. The Fusion calculus: Expressiveness and Symmetry in Mobile Processes. To appear in *Proc. LICS'98*, IEEE Computer Society Press., 1998.

[19] B. C. Pierce and D. N. Turner. Pict: A programming language based on the pi-calculus. Technical Report CSCI 476, Indiana University, 1997. To appear in *Proof, Language and Interaction: Essays in Honour of Robin Milner*, Gordon Plotkin, Colin Stirling, and Mads Tofte, editors, MIT Press.

[20] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms.* PhD thesis CST–99–93, Department of Computer Science, University of Edinburgh, 1992.

[21] D. Sangiorgi. π-calculus, internal mobility and agent-passing calculi. *Theoretical Computer Science*, 167(2):235–274, 1996.

[22] D. Sangiorgi. The name discipline of receptiveness. In *24th ICALP*, LNCS 1256, Springer Verlag, 1997.

[23] D. Sangiorgi and R. Milner. The problem of "Weak Bisimulation up to". In W.R. Cleveland, editor, *Proc. CONCUR '92*, volume 630 of *Lecture Notes in Computer Science*, pages 32–46. Springer Verlag, 1992.

[24] B. Victor and J. Parrow. Concurrent Constrains in the Fusion Calculus. In the Proceedings of ICALP'98, volume 1443 of *Lecture Notes in Computer Science*, Springer Verlag, 1998.

[25] N. Yoshida. Minimality and Separation Results on Asynchronous Mobile Processes: representability theorem by concurrent combinators. To appear in the Proceedings of CONCUR'98.