Mobility and Modularity:
expressing π-calculus in CCS
(extended abstract)

Richard Banach*

University of Manchester, Department of Computer Science
Oxford Road, Manchester M13 9PL, United Kingdom

banach@cs.man.ac.uk

Franck van Breugel†

McGill University, School of Computer Science
3480 University Street, Montreal, Canada H3A 2A7

franck@cs.mcgill.ca

Abstract

A compositional encoding of the π-calculus into infinitary CCS is given that maps reduction bisimilarity in π-calculus onto bisimilarity in CCS in a fully abstract way.

Introduction

In concurrent programming languages like CML [Rep97], Facile [TLK96] and Pict [PT97], which contain a scoping mechanism for channels and allow the communication of channels names, one can write programs with *mobility*: sending (the name of) a local channel out of its scope and, as a result, enlarging its scope with the receiver. Mobility is an abstraction of a crucial programming idiom. This phenomenon will be discussed in more detail in the next paragraph. Most implementations of concurrent programming languages with mobility contain a phase in which a language with mobility is translated into a language without mobility. For example, CML and Facile are both built on top of Standard ML [MTHM97] and Pict is compiled into C. In this paper, we address the question whether this translation can be done in a *modular* (or compositional) way. For that purpose, we consider the *π-calculus* [MPW92], a basic calculus with mobility as its key feature, and *CCS*, the π-calculus' predecessor without mobility. We investigate whether the π-calculus can be expressed in CCS. In a trivial sense, the π-calculus and CCS have the same expressiveness, since they are Turing-equivalent (cf. [Mil89, Section 6.1]). However, Turing-equivalence fails to capture the relative power of concurrency mechanisms and is therefore of little interest in comparing the expressiveness of concurrent calculi. What is more to the point is the extent to which semantic equivalences are preserved by an encoding and the encoding is compositional. According to Milner [Mil83, page 291], the passing of communication links as values, which is one of the key features of the π-calculus, cannot obviously be expressed in a general form in CCS. Sangiorgi [San96, page 235] even claimed that this phenomenon gives the π-calculus a much greater expressiveness than CCS. This claim was supported by a proof of Palamidessi [Pal97] that there does not exist a compositional and 'uniform' encoding of the π-calculus into CCS preserving a 'reasonable'

*Supported by the British Council, the Consiglio Nazionale delle Ricerche, and the Staff Exchange Scheme of the European Commission's ERASMUS programme.

†Supported by the European Commission's HCM programme and NSERC of Canada.

semantics. Nestmann [Nes97] showed that either the 'uniformity' of the encoding or the 'reasonableness' of the semantics is necessary for Palamidessi's other[1] negative result. In this paper, we present a compositional, yet not 'uniform', encoding of the π-calculus into CCS. We follow the scheme proposed by Sangiorgi in his thesis [San92, page 8 and 9]: we treat the formal definition of the semantics of the two calculi, the definition of the compositional encoding of the one calculus into the other, and a proof of the correctness of the encoding with respect to the semantics given. We show that the encoding maps reduction bisimilarity in π-calculus onto bisimilarity in CCS. Reduction bisimilarity does not satisfy Palamidessi's 'reasonableness' condition. We will discuss Palamidessi's additional conditions, 'uniformity' and 'reasonableness', in some detail later.

Let us discuss mobility further now by means of an example. Consider the CML program below. It implements a request-reply protocol, where one process, the `server`, provides some service to other processes, called the `clients`. Each `client` creates a dedicated channel, the `reply_channel`, sends the `request` together with (the name of) this local `reply_channel` along the global `request_channel` to the `server` and then waits for a reply from the `sender` along its created `reply_channel`. The `server` is structured as a loop in which each iteration corresponds to the handling of a `client`'s `request`: the `server` waits for a `request` and a `reply_channel`, sent along the `request_channel` by a `client`, and spawns a process which sends the `reply_to` the `request`—the function `reply_to` actually implements the service—along the `client`'s `reply_channel`. Every `client` has its own `reply_channel` to ensure that the `server` returns each reply to the right `client` (for more details we refer the reader to [Rep92, Chapter 4]).

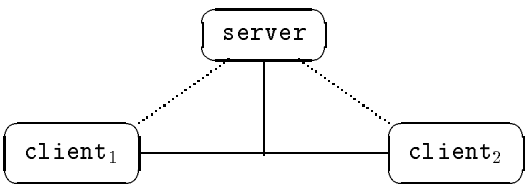
```
val request_channel = channel ()

fun client request = let
      val reply_channel = channel ()
      in
        send (request_channel, (request, reply_channel));
        accept reply_channel
      end

fun reply_to request = ...

fun server () = let
      val (request, reply_channel) = accept request_channel
      in
        spawn (fn () => send (reply_channel, reply_to request));
        server ()
      end
```

In CML, the `let` construct allows the programmer to define the scope of a channel (for example, the `reply_channel` of a client). Furthermore, one can send a channel name along a channel in CML (for example, the `client` sends its `reply_channel` along the `request_channel`). In the above example, mobility arises when a `client`'s local `reply_channel` is communicated, enlarging its scope to the `server`. As a consequence, the communication topology changes. Initially, the three processes can communicate along the global `request_channel`. However, after both `clients` have created their own `reply_channel` and have sent their `reply_channel` to the `server`, the `server` can communicate with each `client` along their `reply_channel`.



[1] Palamidessi also showed that there does not exist a compositional and 'uniform' encoding of the π-calculus into the asynchronous π-calculus preserving a 'reasonable' semantics.

Note that the above communication topology could not have been accomplished if the scoping of channels was static.

Bisimilarity [Mil80, Par81] is the most commonly used semantic equivalence on CCS-processes. For π-processes, a large variety of different notions of bisimilarity have been introduced (see, e.g., [BS96]). *Reduction bisimilarity* focuses on the key feature of concurrency: interaction. Although this semantic equivalence is rather weak—in general, reduction bisimilarity is not preserved by composition—it allows us to give a compositional encoding. Reduction (bisimilarity) is often used in more practical applications (cf. the abstract machine implementation of the π-calculus [Tur95, Chapter 7] and the modelling of stores as processes in CML [BMT92]). In the conclusion, we will sketch how we expect to change our encoding to deal with *early bisimilarity*. This semantic equivalence is the one that corresponds to barbed bisimilarity [MS92]—a canonical semantic equivalence for calculi like CCS and the π-calculus.

In our encoding, the π-calculus is mapped into infinitary CCS. The latter calculus contains two infinite constructs, namely countably infinite summations and restrictions, and countably infinitely many recursive definitions. As we will see, we use these infinite constructs in a controlled way in our encoding. To smooth the encoding of replication, we extend the π-calculus with two infinite constructs: countably infinite restrictions and compositions. These infinite restrictions and compositions show some resemblance with the infinite existential quantifications and conjunctions in infinitary logic [Kar64] as we will point out later.

Let us briefly discuss the basic ideas of our encoding. It is based on Milner's encoding of value passing CCS into pure CCS [Mil89]. To overcome some complications caused by π-calculus' alpha-conversion, we focus on *clash-free* π-processes. Clash-freeness is a minor variation on Barendregt's variable convention for the λ-calculus [Bar84]. In the encoding, we exploit the fact that all restrictions in a clash-free π-process can be pulled out—the phenomenon is also known as *scope extrusion*—preserving reduction bisimilarity. Here it is essential that we have countably infinite restrictions and compositions to handle replication, since restrictions under a replication cannot be pulled out in general.

The main contributions of the present paper are the following. First of all, the encoding of the π-calculus into CCS. This result does not contradict Palamidessi's negative result. It shows that if we drop the 'uniformity' and 'reasonableness' conditions, then we can give a compositional encoding. Second, the study of clash-freeness. Several papers are rather vague on this point. As far as we are aware, the only other papers which consider this notion are [FMQ96, NS97]. Finally, the investigation of countably infinite restrictions and compositions in the π-calculus. As far as we know, these infinite constructs have not been considered. Several algebraic laws for these constructs are exploited, extending laws of, e.g., [PS95] to our infinite setting.

In the rest of this paper we do the following. Section 1 describes the π-calculus and reduction bisimilarity. For more details on the π-calculus we refer the reader to Milner's tutorial [Mil91]. A good introduction to CCS is Milner's book [Mil89]. Section 2 introduces the clash-free fragment of the π-calculus. Section 3 gives the encoding of the restriction-free part of the π-calculus. Section 4 focuses on scope extrusion. Section 5 considers the incorporation of restriction in the encoding. The final section concludes and discusses related and future work.

1 π-Calculus

We present the coinfinite fragment of the monadic[2] π-calculus extended with countable restrictions and countably infinite compositions. Furthermore, we introduce reduction bisimilarity. Our presentation is based on Sangiorgi's thesis [San92, Subsection 2.2.2 and Section 3.2].

DEFINITION 1.1 Let $\mathcal{N} = \{x, y, z, \ldots\}$ be a (countably infinite) set of *names*. The set of *π-processes* is defined by

$$P ::= 0 \mid \bar{x}y.P \mid x(y).P \mid (\nu X)P \mid P + Q \mid P \mid Q \mid \prod_{n \in \mathbb{N}} P_n,$$

where X is a set of names. ⌟

[2] We are confident that the results of the present paper can be extended straightforwardly to deal with the polyadic π-calculus.

To simplify some of the subsequent proofs (e.g., the one of Lemma 2.2) we restrict our attention to coinfinite π-processes.

DEFINITION 1.2 A π-processes P is *coinfinite* if the set $\mathcal{N} \setminus \mathrm{n}(P)$ is infinite. ⌟

In the rest of this paper, all π-processes are assumed to be coinfinite. Note that coinfiniteness is not preserved by alpha-conversion in general.

DEFINITION 1.3 The set of *π-actions* is given by

$$a ::= \bar{x}y \mid \bar{x}(y) \mid xy \mid \tau.$$

The *labelled transition relation*[3] $\longrightarrow$ is defined by the following axioms and rules.

1. $\dfrac{P' \xrightarrow{a} Q'}{P \xrightarrow{a} Q}$ $\quad P$ and Q are alpha-variants of P' and Q'

2. $\bar{x}y.P \xrightarrow{\bar{x}y} P$

3. $x(y).P \xrightarrow{xz} P\{{}^{z}/_{y}\}$

4. $\dfrac{P \xrightarrow{a} P'}{(\nu X)P \xrightarrow{a} (\nu X)P'}$ $\quad X \cap \mathrm{n}(a) \neq \emptyset$

5. $\dfrac{P \xrightarrow{\bar{x}y} P'}{(\nu X)P \xrightarrow{\bar{x}(y)} (\nu X \setminus \{y\})P'}$ $\quad y \in X$ and $x \notin X$

6. $\dfrac{P \xrightarrow{a} P'}{P + Q \xrightarrow{a} P'}$

7. $\dfrac{P \xrightarrow{a} P'}{P \mid Q \xrightarrow{a} P' \mid Q}$ $\quad \mathrm{bn}(a) \cap \mathrm{fn}(Q) = \emptyset$

8. $\dfrac{P \xrightarrow{\bar{x}y} P' \qquad Q \xrightarrow{xy} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$

9. $\dfrac{P \xrightarrow{\bar{x}(y)} P' \qquad Q \xrightarrow{xy} Q'}{P \mid Q \xrightarrow{\tau} (\nu\{y\})(P' \mid Q')}$ $\quad y \notin \mathrm{fn}(Q)$

10. $\dfrac{P_1 \mid \prod_{n \in \mathbb{N}} P_{n+1} \xrightarrow{a} P'}{\prod_{n \in \mathbb{N}} P_n \xrightarrow{a} P'}$

⌟

We have omitted the symmetric versions of the rules 6, 7, 8, and 9. The only differences from the axioms and rules given in [San92] are that rule 1 is a little more general (like in, e.g., [ACS96]), the rules 4, 5, and 9 have been changed to deal with sets of names, and rule 10 has been added to handle countably infinite compositions

[3] Since we will focus on reduction bisimilarity (see Definition 1.4), the early and late instantiation schemes give rise to the same τ-transitions (straightforward modification of [MPW93, Lemma 2.5]), and some results are easier to prove for the early scheme, we have chosen the former.

The rule for replication is

$$\frac{P \mid !P \xrightarrow{a} P'}{!P \xrightarrow{a} P'}$$

Obviously, one can view $!P$ as syntactic sugar for $\prod_{n \in \mathbb{N}} P$. Note that a π-process without countably infinite restrictions and compositions, possibly containing replications, is coinfinite.

We conclude this section by introducing reduction bisimilarity.

DEFINITION 1.4 A relation $\mathcal{R}$ on π-processes is a *reduction bisimulation* if $P \mathcal{R} Q$ implies that

* if $P \xrightarrow{\tau} P'$ then $Q \xrightarrow{\tau} Q'$ for some Q' such that $P' \mathcal{R} Q'$, and
* if $Q \xrightarrow{\tau} Q'$ then $P \xrightarrow{\tau} P'$ for some P' such that $P' \mathcal{R} Q'$.

Processes P and Q are *reduction bisimilar*, written $P \underset{\pi}{\sim} Q$, if $P \mathcal{R} Q$ for some reduction bisimulation $\mathcal{R}$. ⌟

Palamidessi calls a semantics *reasonable* if it distinguishes processes P and Q whenever in some transition sequence of P the names sent or received along certain intended names are different from those of any transition sequence of Q. Since reduction bisimilarity only takes τ-transitions into account, it does not satisfy Palamidessi's 'reasonableness' condition

PROPOSITION 1.5 *Alpha convertibility is a reduction bisimulation.*

PROOF Immediate consequence of rule 1. □

It is important to notice the following. Although we restrict our attention to reductions, i.e. τ-transitions, we still have to deal with inputs xy, free outputs $\bar{x}y$ and bound outputs $\bar{x}(y)$ in our compositional encoding. For example, by compositionality the encoding of $(\nu\{x\})\bar{y}x.P \mid y(z).Q$ should be defined in terms of the encodings of $(\nu\{x\})\bar{y}x.P$ and $y(z).Q$, and we have the following transition proof

$$\frac{(\nu\{x\})\bar{y}x.P \xrightarrow{\bar{y}(x)} (\nu\emptyset)P \qquad y(z).Q \xrightarrow{yx} Q\{^x/_z\}}{(\nu\{x\})\bar{y}x.P \mid y(z).Q \xrightarrow{\tau} (\nu\{x\})((\nu\emptyset)P \mid Q\{^x/_z\})}$$

provided that $x \neq y$ and $x \notin \mathrm{fn}(Q) \setminus \{z\}$.

2 Clash-freeness

We introduce the clash-free fragment of the π-calculus. Clash-freeness is a minor variation on Barendregt's variable convention for the λ-calculus [Bar84, page 26] and coincides with the non-homonymy condition of [FMQ96, page 57] and α-freeness of [NS97, Definition 3]. This notion will turn out to be essential in Proposition 3.3 and Lemma 4.2. A π-process is clash-free if each binder in the process is named distinctly from any other binder in the process, and each binder also differs from any free name in the process. Clash-freeness is formalised as follows.

DEFINITION 2.1

* 0 is clash-free.
* $\bar{x}y.P$ is clash-free if P is clash-free and $x, y \notin \mathrm{bn}(P)$.
* $x(y).P$ is clash-free if P is clash-free and $x, y \notin \mathrm{bn}(P)$ and $x \neq y$.
* $(\nu X)P$ is clash-free if P is clash-free and $X \cap \mathrm{bn}(P) = \emptyset$.
* $P + Q$ is clash-free if P and Q are clash-free and $\mathrm{bn}(P) \cap \mathrm{n}(Q) = \emptyset$ and $\mathrm{bn}(Q) \cap \mathrm{n}(P) = \emptyset$.

* $P \mid Q$ is clash-free if P and Q are clash-free and $\mathrm{bn}(P) \cap \mathrm{n}(Q) = \emptyset$ and $\mathrm{bn}(Q) \cap \mathrm{n}(P) = \emptyset$.
* $\prod_{n \in \mathbb{N}} P_n$ is clash-free if all P_n's are clash-free and $\mathrm{bn}(P_m) \cap \mathrm{n}(P_n) = \emptyset$ for all $m \neq n$.

⌟

LEMMA 2.2 *Every π-process has a clash-free alpha-variant.*

PROOF We define the relation $\Downarrow$ by the following axiom and rules.

* $0 \Downarrow 0$
* $\dfrac{P \Downarrow Q}{\bar{x}y.P \Downarrow \bar{x}y.Q} \qquad x, y \notin \mathrm{bn}(Q)$
* $\dfrac{P \Downarrow Q}{x(y).P \Downarrow x(z).(Q\{^z/_y\})} \qquad x \notin \mathrm{bn}(Q) \text{ and } z \notin \mathrm{n}(Q) \cup \{x\}$
* $\dfrac{P \Downarrow Q}{(\nu X)P \Downarrow (\nu Y)Q\{^Y/_X\}} \qquad Y \cap \mathrm{n}(Q) = \emptyset$
* $\dfrac{P_1 \Downarrow Q_1 \quad P_2 \Downarrow Q_2}{P_1 + P_2 \Downarrow Q_1 + Q_2} \qquad \mathrm{bn}(Q_1) \cap \mathrm{n}(Q_2) = \emptyset \text{ and } \mathrm{bn}(Q_2) \cap \mathrm{n}(Q_1) = \emptyset$
* $\dfrac{P_1 \Downarrow Q_1 \quad P_2 \Downarrow Q_2}{P_1 \mid P_2 \Downarrow Q_1 \mid Q_2} \qquad \mathrm{bn}(Q_1) \cap \mathrm{n}(Q_2) = \emptyset \text{ and } \mathrm{bn}(Q_2) \cap \mathrm{n}(Q_1) = \emptyset$
* $\dfrac{P_n \Downarrow Q_n}{\prod_{n \in \mathbb{N}} P_n \Downarrow \prod_{n \in \mathbb{N}} Q_n} \qquad \mathrm{bn}(Q_m) \cap \mathrm{n}(Q_n) = \emptyset \text{ for all } m \neq n$

We can show that

* if $P \Downarrow Q$ then Q is an alpha-variant of P,
* if $P \Downarrow Q$ then Q is clash-free, and
* for every π-process P and infinite set of names X satisfying $\mathrm{n}(P) \cap X = \emptyset$ there exists a π-process Q such that $P \Downarrow Q$ and $\mathrm{bn}(Q) \subseteq X$

by induction on the proof of $P \Downarrow Q$. □

An alternative approach to distinguish free and bound names in the π-calculus is presented in [BDP96].

Because we can construct for every π-process a clash-free alpha-variant in a compositional way (proof of Lemma 2.2) and alpha-convertibility is a reduction bisimulation (Proposition 1.5), we will restrict our attention in the sequel to clash-free π-processes. Furthermore, we can also confine ourselves to clash-free proofs, i.e. proofs in which all π-processes are clash-free. This is a consequence of

PROPOSITION 2.3 *If $P \xrightarrow{a} Q$ and P and Q are clash-free then there exists a clash-free proof of $P \xrightarrow{a} Q$.*

PROOF We can show that if $P \xrightarrow{a} Q$ then there exists a clash-free proof of $P' \xrightarrow{a} Q'$, with P' and Q' clash-free alpha-variants of P and Q, by transition induction. An application of rule 1 of Definition 1.3 completes the proof. □

3 Encoding

We focus on the restriction-free fragment of the π-calculus. How to encode restriction we will consider later. The π-calculus has inaction, summation and composition and so does CCS, so it's clear how we might want to deal with those. To handle the output and input prefixes of the π-calculus, we emulate the trick used for encoding value passing CCS into pure CCS [Mil89, Section 2.8]. We design the set of CCS channels as pairs $\langle x, y \rangle$, with the first entry of the pair holding the π-calculus name along which is communicated, and the

second entry containing the π-calculus name being transmitted. In the encoding $\mathcal{E}$, the π-calculus output $\bar{x}y$ becomes $\overline{\langle x, y\rangle}$ in CCS:

$$\mathcal{E}(\bar{x}y.P) = \overline{\langle x, y\rangle}.\mathcal{E}(P).$$

To reflect the fact that the π-calculus input $x(y)$ can receive any name z, we encode it by a summation $\sum_{z\in\mathcal{N}}\langle x, z\rangle$:

$$\mathcal{E}(x(y).P) = \sum_{z\in\mathcal{N}} \langle x, z\rangle.\mathcal{E}(P\{{}^{z}/_{y}\}).$$

Note that the encoding of $x(y).P$ is not fully compositional, since it is defined in terms of the encoding of $P\{{}^{z}/_{y}\}$, rather than P. We cannot simply pull the substitution $\{{}^{z}/_{y}\}$ out of the encoding, resulting in

$$\mathcal{E}(x(y).P) = \sum_{z\in\mathcal{N}} \langle x, z\rangle.(\mathcal{E}(P)\{{}^{z}/_{y}\}),$$

since it leads to an unsound encoding[4]. To obtain full compositionality, we define the following operation on CCS-processes. Let σ be a substitution with finite support, i.e. a function from names to names for which the set $\mathrm{dom}(\sigma) = \{\, x \mid \sigma(x) \neq x \,\}$ is finite. The two essential clauses of the definition of $[\sigma]$ on CCS-processes are

$$(\overline{\langle x, y\rangle}.M)[\sigma] = \overline{\langle \sigma(x), \sigma(y)\rangle}.(M[\sigma])$$
$$(\langle x, y\rangle.M)[\sigma] = \langle \sigma(x), y\rangle.(M[\sigma\{{}^{y}/_{y}\}]).$$

Notice that $[\sigma]$ treats the input prefix $\langle x, y\rangle$ like it is a binder (binding y) whose bindings should be preserved (but not necessarily reflected). This operation is exploited in the encoding of the input prefix as follows:

$$\mathcal{E}(x(y).P) = \sum_{z\in\mathcal{N}} \langle x, z\rangle.(\mathcal{E}(P)[\{{}^{z}/_{y}\}]).$$

The use of $[\{{}^{z}/_{y}\}]$ in our encoding is reminiscent to the explicit substitutions in the $\pi\xi$-calculus [FMQ96]. In the infinite summation, we only encounter (guarded or normal) CCS-processes of the form $\langle x, z\rangle.M$ (cf. [Mil91, page 6] and [San92, page 19]).

By analogy with the encoding of replication in terms of constants, we can encode countably infinite compositions by

$$\mathcal{E}(\prod_{n\in\mathbb{N}} P_n) = A_1,$$

where the constants A_1, A_2, ... are fresh and

$$A_n \stackrel{\mathrm{def}}{=} \mathcal{E}(P_n) \mid A_{n+1}.$$

However, to deal correctly with $[\sigma]$ applied to constants, we consider constants of the form $A_{[\sigma],n}$[5] and define the application of $[\sigma]$ to such a constant by

$$A_{[\sigma_1],n}[\sigma_2] = A_{[\sigma_2\circ\sigma_1],n}.$$

Infinite compositions are now treated as follows.

$$\mathcal{E}(\prod_{n\in\mathbb{N}} P_n) = A_{[id],1},$$

where for every substitution σ, the constants $A_{[\sigma],1}$, $A_{[\sigma],2}$, ... are fresh and

$$A_{[\sigma],n} \stackrel{\mathrm{def}}{=} \mathcal{E}(P_n)[\sigma] \mid A_{[\sigma],n+1}.$$

Combining the above leads us to

[4] We leave it to the reader to verify that in this setting the encoding of the π-process $x(y_1).x(y_2).0$ cannot make an $\langle x, y_1\rangle$-transition after having done an $\langle x, y_2\rangle$-transition.

[5] Since we restrict ourselves to substitutions with finite support, the set of constants is countable.

DEFINITION 3.1 The *encoding* $\mathcal{E}$ of a restriction-free π-process is defined by

$$\begin{aligned}
\mathcal{E}(0) &= 0\\
\mathcal{E}(\bar{x}y.P) &= \overline{\langle x,y\rangle}.\mathcal{E}(P)\\
\mathcal{E}(x(y).P) &= \textstyle\sum_{z\in\mathcal{N}}\langle x,z\rangle.(\mathcal{E}(P)[\{^{z}\!/\!_{y}\}])\\
\mathcal{E}(P+Q) &= \mathcal{E}(P)+\mathcal{E}(Q)\\
\mathcal{E}(P\mid Q) &= \mathcal{E}(P)\mid\mathcal{E}(Q)\\
\mathcal{E}(\textstyle\prod_{n\in\mathbb{N}}P_n) &= A_{[id],1},
\end{aligned}$$

where for every substitution σ, the constants $A_{[\sigma],1}$, $A_{[\sigma],2}$, ... are fresh and

$$A_{[\sigma],n} \stackrel{\text{def}}{=} \mathcal{E}(P_n)[\sigma] \mid A_{[\sigma],n+1}.$$

The *encoding* $\mathcal{E}$ of a free π-action is defined by

$$\begin{aligned}
\mathcal{E}(\bar{x}y) &= \overline{\langle x,y\rangle}\\
\mathcal{E}(xy) &= \langle x,y\rangle\\
\mathcal{E}(\tau) &= \tau.
\end{aligned}$$

⌟

Although the encoding of a countably infinite composition defines infinitely many constants, only one definition schema is introduced, where the substitution σ and the natural number n can be viewed as parameters. Because different choices for the constants $A_{[\sigma],1}$, $A_{[\sigma],2}$, ... are allowed, the above introduced encoding is not a function but a relation.

DEFINITION 3.2 CCS-processes are *chi-variants* if they only differ in the choice of constants. ⌟

According to [Mil89, Proposition 4.12], chi-variants are bisimilar. Different choices for the constants in the final clause of the encoding give rise to chi-variants, and hence to bisimilar CCS-processes.

The encoding maps alpha-variants to chi-variants. This is a consequence of

PROPOSITION 3.3 *Let P_1 and P_2 be restriction-free π-processes and let σ_1 and σ_2 be substitutions with $\mathrm{bn}(P_1)\cap\mathrm{dom}(\sigma_1)=\emptyset$ and $\mathrm{bn}(P_2)\cap\mathrm{dom}(\sigma_2)=\emptyset$. If $P_1\sigma_1$ and $P_2\sigma_2$ are alpha-variants then $\mathcal{E}(P_1)[\sigma_1]$ and $\mathcal{E}(P_2)[\sigma_2]$ are chi-variants.*

PROOF Structural induction on P_1 and P_2, exploiting the fact that if $\mathrm{dom}(\sigma_1)\cap\mathrm{dom}(\sigma_2)=\emptyset$ then $(M[\sigma_1])[\sigma_2]=M[\sigma_2\circ\sigma_1]$. □

According to the above proposition, $\mathcal{E}(P\sigma)$ and $\mathcal{E}(P)[\sigma]$ are chi-variants, and hence bisimilar. Note that the equality of $\mathcal{E}(P\sigma)$ and $\mathcal{E}(P)\sigma$, one of the two conditions a *uniform*[6] encoding has to satisfy, does not hold in general.

The proposition is also useful for proving the following operational correspondence between a restriction-free π-process and its encoding.

PROPOSITION 3.4

* *If $P \stackrel{a}{\longrightarrow} Q$ then $\mathcal{E}(P) \stackrel{\mathcal{E}(a)}{\longrightarrow} M$ for some M such that $M \underset{\text{ccs}}{\sim} \mathcal{E}(Q)$.*
* *If $\mathcal{E}(P) \stackrel{c}{\longrightarrow} N$ then $P \stackrel{a}{\longrightarrow} Q$ for some a and Q such that $\mathcal{E}(a)=c$ and $\mathcal{E}(Q) \underset{\text{ccs}}{\sim} N$.*

PROOF Transition induction exploiting Proposition 3.3 and [Mil89, Proposition 4.10, 4.11, and 4.12]. □

Reduction bisimilarity on restriction-free π-processes and bisimilarity on their encodings are linked as follows.

[6] Instead of requiring $\mathcal{E}(P\sigma)$ and $\mathcal{E}(P)\sigma$ to be equal, it seems more natural to ask that $\mathcal{E}(P\sigma)$ and $\mathcal{E}(P)[\sigma]$, where $[-]$ is some operation on substitutions, are semantically equivalent.

LEMMA 3.5 *For all restriction-free π-processes P and Q,*

$$P \underset{\pi}{\simeq} Q \text{ if and only if } \mathcal{E}(P) \setminus \mathcal{N}^2 \underset{\text{ccs}}{\simeq} \mathcal{E}(Q) \setminus \mathcal{N}^2.$$

PROOF From Proposition 3.4 and [Mil89, Proposition 4.10] we can deduce that $\{ \langle \mathcal{E}(P) \setminus \mathcal{N}^2, \mathcal{E}(Q) \setminus \mathcal{N}^2 \rangle \mid P \underset{\pi}{\simeq} Q \}$ is a bisimulation and that $\{ \langle P, Q \rangle \mid \mathcal{E}(P) \setminus \mathcal{N}^2 \underset{\text{ccs}}{\simeq} \mathcal{E}(Q) \setminus \mathcal{N}^2 \}$ is a reduction bisimulation up to alpha-conversion. □

4 Scope extrusion

We introduce the normal form of a π-process. This normal form is obtained by pulling all the restrictions out of the π-process—this is also known as scope extrusion. As we will see in Lemma 5.2, the encoding $\mathcal{E}'$ of an arbitrary π-process can be expressed in terms of the encoding $\mathcal{E}$ of its normal form stripped of its restriction.

DEFINITION 4.1 The *normal form* of a π-process is defined by

$$\begin{aligned}
\mathrm{nf}(0) &= (\nu\emptyset)0 \\
\mathrm{nf}(\bar{x}y.P) &= (\nu X)\bar{x}y.P' \\
\mathrm{nf}(x(y).P) &= (\nu X)x(y).P' \\
\mathrm{nf}((\nu Z)P) &= (\nu X \cup Z)P' \\
\mathrm{nf}(P + Q) &= (\nu X \cup Y)(P' + Q') \\
\mathrm{nf}(P \mid Q) &= (\nu X \cup Y)(P' \mid Q') \\
\mathrm{nf}(\textstyle\prod_{n\in\mathbb{N}} P_n) &= (\nu \bigcup_{n\in\mathbb{N}} X_n) \prod_{n\in\mathbb{N}} P'_n,
\end{aligned}$$

where

$$\begin{aligned}
\mathrm{nf}(P) &= (\nu X)P' \\
\mathrm{nf}(Q) &= (\nu Y)Q' \\
\mathrm{nf}(P_n) &= (\nu X_n)P'_n.
\end{aligned}$$

⌟

LEMMA 4.2 *For all π-processes P, $P \underset{\pi}{\simeq} \mathrm{nf}(P)$.*

PROOF We can prove that

* $\mathrm{bn}(\mathrm{nf}(P)) = \mathrm{bn}(P)$,
* $\mathrm{n}(\mathrm{nf}(P)) = \mathrm{n}(P)$, and
* $\mathrm{nf}(P)$ is clash-free

by structural induction on P. Furthermore, we can show that P is early congruent to $\mathrm{nf}(P)$, again by structural induction on P. The definition of early congruence is given in, e.g., [PS95, Definition 2.5]. Besides the fact that early congruence is also a congruence for our extended calculus, we use the fact that it satisfies several laws including

$$\text{If } x, y \notin X \text{ then } \bar{x}y.(\nu X)P = (\nu X)\bar{x}y.P \tag{1}$$

$$\text{If } x \notin X \text{ then } x(y).(\nu X)P = (\nu X)x(y).P \tag{2}$$

$$\text{If } \mathrm{fn}(Q) \cap X = \emptyset \text{ then } ((\nu X)P) + Q = (\nu X)(P + Q) \tag{3}$$

$$\text{If } \mathrm{fn}(Q) \cap X = \emptyset \text{ then } ((\nu X)P) \mid Q = (\nu X)(P \mid Q) \tag{4}$$

$$\textstyle\prod_{n\in\mathbb{N}}(\nu X_n)P_n = (\nu \bigcup_{n\in\mathbb{N}} X_n) \prod_{n\in\mathbb{N}} P_n. \tag{5}$$

The observation that early congruence is a reduction bisimulation concludes the proof. □

Law (5) seems closely related to the law of independent choice in infinitary logic (see, e.g., [Kar64, Subsection 11.1.1]). Further study is needed to make this correspondence precise.

Observe that we cannot pull a restriction out of a replication, i.e. $!(\nu X)P \underset{\pi}{\simeq} (\nu X)!P$ does not hold in general. This is the reason why we added countably infinite restrictions and compositions to the calculus.

Since restriction preserves and reflects (up to alpha-conversion) τ-transitions, we have

LEMMA 4.3 *For all restriction-free π-processes P, $(\nu X)P \underset{\pi}{\sim} P$.*

PROOF From rule 4 of Definition 1.3 we can conclude that restriction preserves τ-transitions. We can also show that restriction reflects τ-transitions up to alpha-conversion exploiting straightforward modifications of [MPW92, Lemma 1 and 3]. From these facts we can easily deduce that the corresponding relation is a reduction bisimulation up to alpha-conversion. □

5 Restriction

We present the encoding of the full calculus.

DEFINITION 5.1 The *encoding* $\mathcal{E}'$ of a π-process is defined by

$$\begin{aligned}
\mathcal{E}'(0) &= 0 \setminus \mathcal{N}^2 \\
\mathcal{E}'(\bar{x}y.P) &= (\overline{\langle x, y\rangle}.M) \setminus \mathcal{N}^2 \\
\mathcal{E}'(x(y).P) &= (\textstyle\sum_{z \in \mathcal{N}} \langle x, z\rangle.(M[\{z/y\}])) \setminus \mathcal{N}^2 \\
\mathcal{E}'((\nu X)P) &= M \setminus \mathcal{N}^2 \\
\mathcal{E}'(P + Q) &= (M + N) \setminus \mathcal{N}^2 \\
\mathcal{E}'(P \mid Q) &= (M \mid N) \setminus \mathcal{N}^2 \\
\mathcal{E}'(\textstyle\prod_{n \in \mathbb{N}} P_n) &= A_{[id],1} \setminus \mathcal{N}^2,
\end{aligned}$$

where

$$\begin{aligned}
\mathcal{E}'(P) &= M \setminus \mathcal{N}^2 \\
\mathcal{E}'(Q) &= N \setminus \mathcal{N}^2 \\
\mathcal{E}'(P_n) &= M_n \setminus \mathcal{N}^2,
\end{aligned}$$

where for every substitution σ, the constants $A_{[\sigma],1}$, $A_{[\sigma],2}$, ... are fresh and

$$A_{[\sigma],n} \stackrel{\text{def}}{=} M_n[\sigma] \mid A_{[\sigma],n+1}.$$

⌟

The only way we use infinite restrictions in CCS is by internalising all CCS channels of the form $\langle x, y\rangle$.

Note that our encoding $\mathcal{E}'$ does not satisfy $\mathcal{E}'(P \mid Q) = \mathcal{E}'(P) \mid \mathcal{E}'(Q)$, the second condition a *uniform* encoding has to satisfy. It might be reasonable strengthen the notion of compositionality in this way for a distributed implementation (but not for a centralised one). Observe that our encoding preserves the amount of concurrency.

The encoding $\mathcal{E}'$ can be expressed in terms of the encoding $\mathcal{E}$ as follows (cf. the two-level encoding in [Nes97, Section 4.1]).

LEMMA 5.2 *If* $\mathrm{nf}(P) = (\nu X)P'$ *then* $\mathcal{E}'(P) = \mathcal{E}(P') \setminus \mathcal{N}^2$.

PROOF Structural induction on P. □

Combining the above results, we arrive at

THEOREM 5.3 *For all π-processes P and Q,*

$$P \underset{\pi}{\sim} Q \text{ if and only if } \mathcal{E}'(P) \underset{\text{ccs}}{\sim} \mathcal{E}'(Q).$$

PROOF Let $\mathrm{nf}(P) = (\nu X)P'$ and $\mathrm{nf}(Q) = (\nu Y)Q'$. Then

$$
\begin{aligned}
& P \underset{\pi}{\simeq} Q \\
\iff\ & (\nu X)P' \underset{\pi}{\simeq} (\nu Y)Q' \quad \text{[Lemma 4.2]} \\
\iff\ & P' \underset{\pi}{\simeq} Q' \quad \text{[Lemma 4.3]} \\
\iff\ & \mathcal{E}(P') \setminus \mathcal{N}^2 \underset{ccs}{\simeq} \mathcal{E}(Q') \setminus \mathcal{N}^2 \quad \text{[Lemma 3.5]} \\
\iff\ & \mathcal{E}'(P) \underset{ccs}{\simeq} \mathcal{E}'(Q). \quad \text{[Lemma 5.2]}
\end{aligned}
$$

□

Conclusion

Ever since its introduction, the expressiveness of the π-calculus and related calculi has attracted a considerable amount of attention (see, e.g., [Bou92, Bor96, HT91, NP96, San93]). Here, we have shown that the π-calculus can be encoded in CCS is a compositional way. From this result we should *not* conclude that we can therefore focus on CCS. In contrast to CCS, the π-calculus is a calculus with mobility. Our study has shown that we can exploit this mobility without having to give up modularity.

The work of Palamidessi [Pal97], which is closely related to ours, has already been discussed in detail. Another paper that addresses a related problem is [FMQ96]. Ferrari, Montanari and Quaglia present an alternative formulation of the π-calculus in which substitution is handled explicitly via the introduction of a suitable operator. This explicit handling of substitution gives rise to a labelled transition system for the π-calculus which is much closer to the ordinary labelled transition system for CCS. Hence, their study provides another way to map the gap between the π-calculus and CCS. In [FO91], Fredlund and Orava study how mobility can be specified in LOTOS. Although LOTOS differs from CCS, it also does not support mobility.

As the semantics for the π-calculus we chose reduction bisimilarity. In the introduction we already motivated this choice. We are confident that we can exploit the techniques developed in this paper to define a compositional encoding of the π-calculus into CCS that maps early bisimilarity in π-calculus into bisimilarity in CCS. Handling free outputs is easy: we just have to change the (outermost) restriction. To deal with free inputs, we introduce for each name two copies: one which is restricted (by the outermost CCS restriction) and one which is not. Bound outputs are handled similarly, although we have to be careful to always pick a fresh name.

The results of Section 4 might be exploited in register allocation. The smaller the scope of a channel, the less pressure it puts on the registers. Note that the construction of normal forms does exactly the opposite.

Whether our encoding of the π-calculus into CCS has any impact on the existing denotational models for the π-calculus, like e.g. [FMS96, Sta96, CSW97], needs further study. These models are much more complex than the ones for CCS and the encoding might give us some insight how the models for the π-calculus can be simplified.

Acknowledgements

The early inspiration for this work originated during a visit by Juraj Balázs to the first author in Manchester. Both authors would like to express their thanks to Juraj for his ideas at that time. Furthermore, we are thankful to Michele Boreale, Pierpaolo Degano, Gianluigi Ferrari, Laurie Hendren, Claudio Hermida, Michael Makkai, Uwe Nestmann, Vincent van Oostrom, Prakash Panangaden, Paola Quaglia, and Davide Sangiorgi for discussion.

References

[ACS96] R.M. Amadio, I. Castellani, and D. Sangiorgi. On Bisimulation for the Asynchronous π-Calculus. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *Lecture Notes in Computer Science*, pages 147–162, Pisa, August 1996. Springer-Verlag. To appear in *Theoretical Computer Science*.

[Bar84] H.P. Barendregt. *The Lambda Calculus, its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, revised edition, 1984.

[BDP96] C. Bodei, P. Degano, and C. Priami. Handling Locally Names of Mobile Agents. Draft, University of Pisa, Pisa, November 1996.

[BMT92] D. Berry, R. Milner, and D.N. Turner. A Semantics for ML Concurrency Primitives. In *Proceedings of the 19th Annual ACM Symposium on Principes of Programming Languages*, pages 119–129, Albuquerque, January 1992. ACM.

[Bor96] M. Boreale. On the Expressiveness of Internal Mobility in Name-Passing Calculi. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *Lecture Notes in Computer Science*, pages 163–178, Pisa, August 1996. Springer-Verlag. To appear in *Theoretical Computer Science*.

[Bou92] G. Boudol. Asynchrony and the π-Calculus (note). Report RR-1702, INRIA, Sophia Antipolis, May 1992.

[BS96] M. Boreale and D. Sangiorgi. Some Congruence Properties for π-Calculus Bisimilarities. Report RR-2870, INRIA, Sophia Antipolis, 1996. To appear in *Theoretical Computer Science*.

[CSW97] G.L. Cattani, I. Stark, and G. Winskel. Presheaf Models for the Pi-Calculus. In *Proceedings of the 7th International Conference on Category Theory and Computer Science*, volume 1290 of *Lecture Notes in Computer Science*, pages 106–126, Santa Margherita Ligure, September 1997. Springer-Verlag.

[FMQ96] G. Ferrari, U. Montanari, and P. Quaglia. A π-Calculus with Explicit Substitutions. *Theoretical Computer Science*, 168(1):53–103, November 1996.

[FMS96] M. Fiore, E. Moggi, and D. Sangiorgi. A Fully Abstract Model for the π-Calculus. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*, pages 43–54, New Brunswick, July 1996. IEEE Computer Society Press.

[FO91] L. Fredlund and F. Orava. Modelling Dynamic Communication Structures in LOTOS. In K.R. Parker and G.A. Rose, editors, *Proceedings of the 4th International Conference on Formal Description Techniques for Distributed Systems and ' Communications Protocols*, pages 185–200, Sydney, November 1991. North-Holland.

[HT91] K. Honda and M. Tokoro. An Object Calculus for Asynchronous Communication. In P. America, editor, *Proceedings of the European Conference on Object-Oriented Programming*, volume 512 of *Lecture Notes in Computer Science*, pages 133–147, Geneva, July 1991. Springer-Verlag.

[Kar64] C.R. Karp. *Languages with Expressions of Infinite Length*. Studies in Logic and the Foundations of Mathematics. North-Holland, Amsterdam, 1964.

[Mil80] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1980.

[Mil83] R. Milner. Calculi for Synchrony and Asynchrony. *Theoretical Computer Science*, 25(3):267–310, July 1983.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall International Series in Computer Science. Prentice Hall International, New York, 1989.

[Mil91] R. Milner. The Polyadic π-Calculus: a tutorial. Report ECS-LFCS-91-180, University of Edinburgh, Edinburgh, October 1991.

[MPW92] R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes, I and II. *Information and Computation*, 100(1):1–40 and 41–77, September 1992.

[MPW93] R. Milner, J. Parrow, and D. Walker. Modal Logics for Mobile Processes. *Theoretical Computer Science*, 114(1):149–171, June 1993.

[MS92] R. Milner and D. Sangiorgi. Barbed Bisimulation. In W. Kuich, editor, *Proceedings of the 19th International Colloquium on Automata, Languages and Programming*, volume 623 of *Lecture Notes in Computer Science*, pages 685–695, Vienna, July 1992. Springer-Verlag.

[MTHM97] R. Milner, M. Tofte, R. Harper, and D. MacQueen. *The Definition of Standard ML – Revised*. The MIT Press, Cambridge, 1997.

[Nes97] U. Nestmann. What is a 'Good' Encoding of Guarded Choice? In C. Palamidessi and J. Parrow, editors, *Proceedings of EXPRESS'97*, volume 7 of *Electronic Notes in Theoretical Computer Science*, Santa Margherita Ligure, September 1997. Elsevier.

[NP96] U. Nestmann and B.C. Pierce. Decoding Choice Encodings. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *Lecture Notes in Computer Science*, pages 179–194, Pisa, August 1996. Springer-Verlag.

[NS97] U. Nestmann and M. Steffen. Typing Confluence. Report ERCIM-10/97-R052, ERCIM, October 1997.

[Pal97] C. Palamidessi. Comparing the Expressive Power of the Synchronous and the Asynchronous π-Calculus. In *Proceedings of the 24th Annual SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 256–265, Paris, January 1997. ACM.

[Par81] D. Park. Concurrency and Automata on Infinite Sequences. In P. Deussen, editor, *Proceedings of 5th GI-Conference on Theoretical Compute Science*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183, Karlsruhe, March 1981. Springer-Verlag.

[PS95] J. Parrow and D. Sangiorgi. Algebraic Theories for Name-Passing Calculi. *Information and Computation*, 120(2):174–197, August 1995.

[PT97] B.C. Pierce and D.N. Turner. Pict: A Programming Language Based on the Pi-Calculus. Report CSCI 476, Indiana University, Bloomington, March 1997.

[Rep92] J.H. Reppy. *Higher-Order Concurrency.* PhD thesis, Cornell University, Ithaca, January 1992.

[Rep97] J.H. Reppy. *Concurrent Programming in ML.* Cambridge University Pres, Cambridge, 1997.

[San92] D. Sangiorgi. *Expressing Mobility in Process Algebras: first-order and higher-order paradigms.* PhD thesis, University of Edinburgh, Edinburgh, 1992.

[San93] D. Sangiorgi. From π-Calculus to Higher-Order π-Calculus—and back. In M.-C. Gaudel and J.-P. Jouannaud, editors, *Proceedings of the 5th International Conference on Theory and Practice of Software Development*, volume 668 of *Lecture Notes in Computer Science*, pages 151–166, Orsay, April 1993. Springer-Verlag.

[San96] D. Sangiorgi. π-Calculus, Internal Mobility, and Agent-Passing Calculi. *Theoretical Computer Science*, 167(1/2):235–274, October 1996.

[Sta96] I. Stark. A Fully Abstract Domain Model for the π-Calculus. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*, pages 36–42, New Brunswick, July 1996. IEEE Computer Society Press.

[TLK96] B. Thomsen, L. Leth, and T.-M. Kuo. A Facile Tutorial. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *Lecture Notes in Computer Science*, pages 278–298, Pisa, August 1996. Springer-Verlag.

[Tur95] D.N. Turner. *The Polymorphic Pi-Calculus: Theory and Implementation.* PhD thesis, University of Edinburgh, Edinburgh, 1995.