Contents

Concurrent Pattern Calculus

Thomas Given-Wilson[1,2] Daniele Gorla[3] Barry Jay[1]

[1] Centre for Quantum Computation and Intelligent Systems &
School of Software, University of Technology, Sydney

[2] National ICT Australia

[3] Dip. di Informatica, "Sapienza" Università di Roma

February 11, 2013

Abstract

Concurrent pattern calculus (CPC) drives interaction between processes by comparing data structures, just as sequential pattern calculus drives computation. By generalising from pattern matching to pattern unification, interaction becomes symmetrical, with information flowing in both directions. CPC provides a natural language to express trade where information exchange is pivotal to interaction. The unification allows some patterns to be more discriminating than others; hence, the behavioural theory must take this aspect into account, leading to a bisimulation subject to compatibility of patterns. Many popular process calculi can be encoded in CPC; this allows for a gain in expressiveness, formalised through encodings.

1 Introduction

The π-calculus [28] holds an honoured position among process calculi as it is the simplest that is able to support computation as represented by λ-calculus [5]. However, *pattern calculus* [24, 22] supports even more computations than λ-calculus since pattern-matching functions may be intensional with respect to their arguments [23]. For example, the pattern $x\ y$ can decompose any compound data structure $u\ v$ into its components u and v. Hence it is natural to explore how a concurrent pattern calculus that exploits factorisation might appear. In fact it turns out rather well.

This paper adapts the pattern-matching mechanism of the pure pattern calculus [24, 22] to concurrent processes, that are built up by exploiting parallel composition, name restriction and replication. This yields a *concurrent pattern calculus* (CPC), where prefixes for input and output are generalised to patterns whose *unification* triggers a two-way, or symmetric, flow of information, as represented by the sole interaction rule

$$(p \to P \mid q \to Q) \quad \longmapsto \quad \sigma P \mid \rho Q$$

where σ and ρ are the substitutions on names resulting from the unification of p and q.

The flexibility of the pattern unification and the symmetry of exchange in CPC align closely with the world of trade. Here the support for discovering a compatible process and exchanging information mirrors the behaviour of trading systems such as the stock market. To this end, an example of stock trading is developed to showcase the capabilities of CPC as a model and specification language.

The definition of CPC also includes a behavioural theory that defines when two processes are behaviourally equivalent. This is done using a standard path in concurrency. First, define an intuitive notion of equivalence that equates processes with the same interactional behaviour, in any context and along any reduction sequence, to yield a notion of *barbed congruence*. Second, provide a more effective characterisation of such equivalence by means of a *labelled transition system* (LTS) and a *bisimulation-based* equivalence. Although this path is familiar, some delicacy is required for each definition. For example, as unification of patterns may require testing of names for equality, the *barbs* of CPC (i.e. the predicate describing the interactional behaviour of a CPC process) must account for names that *might* be matched, not just those that *must* be matched. This is different from the standard barbs of, say, the π-calculus. Further, as some patterns are more discriminating than others, the bisimulation defined here will rely on a notion of *compatibility* of patterns, yielding a bisimulation game in which a challenge can be replied to with a different, though compatible, reply. This is reminiscent of the asynchronous bisimulation for the asynchronous π-calculus [4].

CPC's support for interaction that is both structured and symmetrical makes it more expressive than most approaches to interaction in the literature. For example, checking equality of channel names, as in π-calculus [28], can be viewed as a trivial form of pattern matching. This can be generalized to matching tuples of names, as in Linda [12], or fusing names, as in Fusion [30]. Spi calculus [3] adds patterns for numbers (zero and successors) and encryptions.

More formally, π-calculus, Linda and Spi calculus can all be encoded into CPC but CPC cannot be encoded into any of them. By contrast, the way in which name fusion is modeled in fusion calculus makes the latter not encodable into CPC; conversely, the richness of CPC's pattern matching also makes the converse encoding impossible.

A natural objection to CPC is that the unification is too complex to be an atomic operation. In particular, any limit to the size of communicated messages could be violated by some match. Also, one cannot, in practice, implement a simultaneous exchange of information, so that pattern unification must be implemented in terms of simpler primitives. This objection applies to many other calculi. For example, neither substitution in λ-calculus nor Linda's pattern matching are atomic, but both underpin many existing programming environments [2, 7, 31]. The same comments apply to several other process calculi [25, 32, 34].

The structure of the paper is as follows. Section 2 introduces symmetric matching through a concurrent pattern calculus and an illustrative example. Section 3 defines the behavioural theory of the language: its barbed congruence, LTS and the alternative characterization via a bisimulation-based equivalence. Section 4 formalises the relation between CPC and other process calculi. Section 5 concludes by also considering future work.

2 Concurrent Pattern Calculus

This section presents a *concurrent pattern calculus* (CPC) that uses symmetric pattern matching as the basis of communication. Both symmetry and pattern matching appear in existing models of concurrency, but in more limited ways. For example, π-calculus requires a sender and receiver to share a channel, so that the presence of the channel is symmetric but information flow is in one direction only. Fusion calculus achieves symmetry by fusing names together but has no intensional patterns. Shifting towards intensional patterns is Linda that can test equality of an arbitrary number of names, and the number of names to be communicated in an atomic interaction. Spi calculus has even more intensional patterns, e.g. for natural numbers, and can check equality of terms (i.e. patterns), but does not perform matching in general. Neither Linda or Spi calculus support much symmetry beyond that of the π-calculus.

The expressiveness of CPC comes from extending the class of communicable objects from raw names to a class of *patterns* that can be unified. This merges equality testing and bi-directional communication in a single step.

2.1 Patterns

Suppose given a countable set of *names* $\mathcal{N}$ (meta-variables $n, m, x, y, z, \ldots$. The *patterns* (meta-variables $p, p', p_1, q, q', q_1, \ldots$) are built using names and have the following forms:

$$
\begin{array}{llll}
\textit{Patterns} & p ::= & \lambda x & \text{binding name} \\
 & & x & \text{variable name} \\
 & & \ulcorner x \urcorner & \text{protected name} \\
 & & p \bullet p & \text{compound.}
\end{array}
$$

A binding names λx denotes information sought by a trader; variable names x represent such information. Protected names $\ulcorner x \urcorner$ represent recognised information that cannot be traded. A compound combines two patterns p and q, its *components*, into a pattern $p \bullet q$. Compounding is left associative, similar to application in λ-calculus, pure pattern calculus, and combinatory logics. The *atoms* are patterns that are not compounds. The atoms x and $\ulcorner x \urcorner$ *know* x.

Binding, variable and protected names are all taken from well established concepts in the literature. Indeed, there is a correspondence between patterns and prefixes of more familiar process calculi, such as π-calculus: binding names correspond to input arguments and variable names to output arguments. Moreover, protected names somehow appeared in Linda. There is some subtlety in their relationship to variable names. As protected names specify a requirement, it is natural that they unify with the variable form of the name. Similarly, as protected names in CPC can be used to support channel-based communication, it is also natural that protected names unify with themselves.

The subtleties can be clarified by considering a simple example of trading shares in a stock market. A stock jobber is offering to sell shares in company ABC for one dollar, as represented by a pattern of the form

$$\texttt{ABC} \bullet \texttt{\$1}\ .$$

A stock raider is offering to buy shares in `ABC` but does not want anyone to know this, unless they are offering to sell, as represented by the pattern

$$\ulcorner \mathtt{ABC} \urcorner \bullet \lambda x\ .$$

Finally, a bottom feeder is interested in buying any cheap shares and so may use a pattern of the form

$$\lambda s \bullet \mathtt{\$1}\ .$$

Given a pattern p the sets of: *variables names*, denoted $\mathsf{vn}(p)$; *protected names*, denoted $\mathsf{pn}(p)$; and *binding names*, denoted $\mathsf{bn}(p)$, are defined as expected with the union being taken for compounds. The *free names* of a pattern p, written $\mathsf{fn}(p)$, is the union of the variable names and protected names of p. A pattern is *well formed* if its binding names are pairwise distinct and different from the free ones. All patterns appearing in the rest of this paper are assumed to be well formed.

As protected names are limited to recognition and binding names are being sought, neither should be communicable to another process. Thus, a pattern is *communicable*, i.e. is able to be traded to another process, if it contains no protected or binding names.

Protection of a name can be extended to a communicable pattern p by defining

$$\ulcorner x \urcorner = \ulcorner x \urcorner \qquad\qquad \ulcorner p \bullet q \urcorner = \ulcorner p \urcorner \bullet \ulcorner q \urcorner\ .$$

A *substitution* σ is defined as a partial function from names to communicable patterns. The *domain* of σ is denoted $\mathsf{dom}(\sigma)$; the free names of σ, written $\mathsf{fn}(\sigma)$, is given by the union of the sets $\mathsf{fn}(\sigma x)$ where $x \in \mathsf{dom}(\sigma)$. The *names* of σ, written $\mathsf{names}(\sigma)$, are $\mathsf{dom}(\sigma) \cup \mathsf{fn}(\sigma)$. A substitution σ *avoids* a name x (or a collection of names $\widetilde{n}$) if $x \notin \mathsf{names}(\sigma)$ (respectively $\widetilde{n} \cap \mathsf{names}(\sigma) = \{\}$). Substitution *composition* is denoted $\sigma_2 \circ \sigma_1$ and is defined by $\sigma_2 \circ \sigma_1(t) = \sigma_2(\sigma_1 t)$. Note that all substitutions considered in this paper have finite domain.

For later convenience, we denote by id_X the identity substitution on a set of names X; it maps every name in X to itself, i.e. $\mathsf{id}_X = \{x/x\}$ for every $x \in X$.

Substitutions are applied to patterns as follows

$$\begin{array}{rcl}
\sigma x & = & \left\{ \begin{array}{ll} \sigma(x) & \text{if } x \in \mathsf{dom}(\sigma) \\ x & \text{otherwise} \end{array} \right. \\
\sigma \ulcorner x \urcorner & = & \left\{ \begin{array}{ll} \ulcorner \sigma(x) \urcorner & \text{if } x \in \mathsf{dom}(\sigma) \\ \ulcorner x \urcorner & \text{otherwise} \end{array} \right. \\
\sigma(\lambda x) & = & \lambda x \\
\sigma(p \bullet q) & = & (\sigma p) \bullet (\sigma q)\ .
\end{array}$$

Similar to pure pattern calculus, the action $\hat{\sigma}$ of a substitution σ behaves as a normal substitution, except operating on binding names rather than on free names. In CPC, it is defined by

$$\begin{array}{rcl}
\hat{\sigma} x & = & x \\
\hat{\sigma} \ulcorner x \urcorner & = & \ulcorner x \urcorner \\
\hat{\sigma}(\lambda x) & = & \left\{ \begin{array}{ll} \sigma(x) & \text{if } x \in \mathsf{dom}(\sigma) \\ \lambda x & \text{otherwise} \end{array} \right. \\
\hat{\sigma}(p \bullet q) & = & (\hat{\sigma} p) \bullet (\hat{\sigma} q)\ .
\end{array}$$

When σ is of the form $\{p_i/x_i\}_{i\in I}$, then $\{p_i/\lambda x_i\}_{i\in I}$ may be used to denote $\hat{\sigma}$.

The *symmetric matching* or *unification* $\{p\|q\}$ of two patterns p and q attempts to unify p and q by generating substitutions upon their binding names. When defined, the result is a pair of substitutions whose domains are the binding names of p and of q, respectively. The rules to generate the substitutions are:

$$
\begin{array}{rcll}
\left.\begin{array}{r} \{x\|x\} \\ \{x\|\ulcorner x\urcorner\} \\ \{\ulcorner x\urcorner\|x\} \\ \{\ulcorner x\urcorner\|\ulcorner x\urcorner\} \end{array}\right\} & = & (\{\},\{\}) & \\
\{\lambda x\|q\} & = & (\{q/x\},\{\}) & \text{if } q \text{ is communicable} \\
\{p\|\lambda x\} & = & (\{\},\{p/x\}) & \text{if } p \text{ is communicable} \\
\{p_1\bullet p_2\|q_1\bullet q_2\} & = & ((\sigma_1\cup\sigma_2),(\rho_1\cup\rho_2)) & \text{if } \{p_i\|q_i\} = (\sigma_i,\rho_i) \text{ for } i\in\{1,2\} \\
\{p\|q\} & = & \text{undefined} & \text{otherwise.}
\end{array}
$$

Two atoms unify if they know the same name. A name that seeks information (i.e., a binding name) unifies with any communicable pattern to produce a binding for its underlying name. Two compounds unify if their corresponding components do; the resulting substitutions are given by taking unions of those produced by unifying the components (necessarily disjoint as patterns are well-formed). Otherwise the patterns cannot be unified and the matching is undefined.

Proposition 2.1. *If the unification of patterns p and q is defined then any protected name of p is a free name of q.*

Proof: Without loss of generality q is a not a binding name λx, as this would imply that p is communicable (and, so, without protected names). Now proceed by induction on the structure of p.

If p is some protected name $\ulcorner x\urcorner$ then q must be either x or $\ulcorner x\urcorner$ for the matching to be defined and x is in the free names of q. If p is any other name then there is nothing to prove.

If p is of the form $p_1\bullet p_2$ then by unification q must be of the form $q_1\bullet q_2$ and $\{p_i\|q_i\}$ is defined for $i\in\{1,2\}$. Now any name x that is protected in either p_1 or p_2 must be free in the corresponding q_1 or q_2 by induction. $\square$

2.2 Processes

The processes of CPC are given by:

$$
\begin{array}{lrlll}
\textit{Processes} & P & ::= & \mathbf{0} & \text{zero} \\
 & & & P|P & \text{parallel composition} \\
 & & & !P & \text{replication} \\
 & & & (\nu x)P & \text{restriction} \\
 & & & p\to P & \text{case.}
\end{array}
$$

The null process, parallel composition, replication and restriction are the classical ones for process calculi: $\mathbf{0}$ is the inactive process; $P \mid Q$ is the parallel composition of processes P and Q, allowing the two processes to evolve independently or to interact; the replication $!P$ provides as many parallel copies of P as desired; $(\nu x)P$ declares a new name x, visible only within P and distinct from any other name. The traditional input and output primitives are replaced by the *case*, viz. $p \to P$, that has a pattern p and a body P. If P is $\mathbf{0}$ then $p \to \mathbf{0}$ may be denoted by p.

For later convenience, $\widetilde{n}$ denotes a sequence of names $n_1, \ldots, n_i$; for example, $(\nu n_1)(\ldots((\nu n_i)P))$ will be written $(\nu\widetilde{n})P$.

The free names of processes, denoted $\mathsf{fn}(P)$, are defined as usual for all the traditional primitives and

$$\mathsf{fn}(p \to P) \quad = \quad \mathsf{fn}(p) \cup (\mathsf{fn}(P) \backslash \mathsf{bn}(p))$$

for the case. As expected the binding names of the pattern bind their free occurrences in the body.

2.3 Operational Semantics

The application σP of a substitution σ to a process P is defined in the usual manner, provided that there is no name capture:

$$\sigma(p \to P) \quad = \quad (\sigma p) \to (\sigma P) \qquad \text{if } \mathsf{names}(\sigma) \cap \mathsf{bn}(p) = \emptyset.$$

Name capture can be avoided by α-conversion, $=_\alpha$, that is the congruence relation generated by the following axioms:

$$\begin{array}{lll} (\nu x)P & =_\alpha & (\nu y)(\{y/x\}P) & \qquad y \notin \mathsf{fn}(P) \\ p \to P & =_\alpha & (\{\lambda y/\lambda x\}p) \to (\{y/x\}P) & \qquad x \in \mathsf{bn}(p), y \notin \mathsf{fn}(P) \cup \mathsf{bn}(p) \;. \end{array}$$

The *structural equivalence relation* $\equiv$ is defined just as in π-calculus [27]: it includes α-conversion and its defining axioms are:

$$P \mid \mathbf{0} \equiv P \qquad P \mid Q \equiv Q \mid P \qquad P \mid (Q \mid R) \equiv (P \mid Q) \mid R$$

$$(\nu n)\mathbf{0} \equiv \mathbf{0} \qquad (\nu n)(\nu m)P \equiv (\nu m)(\nu n)P \qquad !P \equiv P \mid !P$$

$$P \mid (\nu n)Q \equiv (\nu n)(P \mid Q) \quad \text{if } n \notin \mathsf{fn}(P) \;.$$

It states that: $\mid$ is a commutative, associative, monoidal operator, with $\mathbf{0}$ acting as the identity; that restriction is useless when applied to the empty process; that the order of restricted names is immaterial; that replication can be freely unfolded; and that the scope of a restricted name can be freely extended, provided that no name capture arises.

The operational semantics of CPC is formulated via a *reduction relation* between pairs of CPC processes. Its defining rules are:

$$\frac{}{(p \to P) \mid (q \to Q) \longmapsto (\sigma P) \mid (\rho Q)} \ \text{if } \{p \| q\} = (\sigma, \rho)$$

$$\frac{P \longmapsto P'}{P|Q \longmapsto P'|Q} \qquad \frac{P \longmapsto P'}{(\nu n)P \longmapsto (\nu n)P'} \qquad \frac{P \equiv Q \quad Q \longmapsto Q' \quad Q' \equiv P'}{P \longmapsto P'} \;.$$

CPC has one interaction axiom, stating that, if the unification of two patterns p and q is defined and generates (σ, ρ), the substitutions σ and ρ are applied to the bodies P and Q, respectively. If the matching of p and q is undefined then no interaction occurs. Unlike the sequential setting, there is no need for capturing failure of unification. This is due to interaction being opportunistic between processes rather than being forced by application between terms. Indeed, failure to interact with one process should not prevent interactions with other processes.

The interaction rule is then closed under parallel composition, restriction and structural equivalence in the usual manner. Unlike pure pattern calculus, computation does not occur within the body of a case. As usual, $\Longmapsto$ denotes the reflexive and transitive closure of $\longmapsto$.

The section concludes with three simple properties of substitutions and the reduction relation.

Proposition 2.2. *For every process P and substitution σ, it holds that $\mathsf{fn}(\sigma P) \subseteq \mathsf{fn}(P) \cup \mathsf{fn}(\sigma)$.*

Proof: Trivial by definition of the application of σ. $\square$

Proposition 2.3. *If $P \Longmapsto P'$, then $\mathsf{fn}(P') \subseteq \mathsf{fn}(P)$.*

Proof: $P \Longmapsto P'$ means that $P \longmapsto^k P'$, for some $k \geq 0$. The proof is by induction in k and trivially follows by Proposition 2.2. $\square$

Proposition 2.4. *If $P \longmapsto P'$, then $\sigma P \longmapsto \sigma P'$, for every σ.*

Proof: By induction on the inference for $P \longmapsto P'$. $\square$

Proposition 2.5. *Suppose a process $p \to P$ interacts with a process Q. If x is a protected name in p then x must be a free name in Q.*

Proof: For Q to interact with $p \to P$ it must be that $Q \Longmapsto (\nu\widetilde{n})(q \to Q_1 \mid Q_2)$ such that $\widetilde{n} \cap \mathsf{fn}(p \to P) = \emptyset$ and $\{p\|q\}$ is defined. Then, by Proposition 2.1, the free names of q include x and, consequently, x must be free in $q \to Q_1 \mid Q_2$. By Proposition 2.3, x is free in Q. Further, x cannot belong to $\widetilde{n}$, since $x \in \mathsf{fn}(p \to P)$ and $\widetilde{n} \cap \mathsf{fn}(p \to P) = \emptyset$. $\square$

2.4 Trade in CPC

This section uses the example of share trading to explore the potential of CPC. The scenario is that two potential traders, a buyer and a seller, wish to engage in trade. To complete a transaction, the traders need to progress through two stages: *discovering* each other and *exchanging* information. Both traders begin with a pattern for their desired transaction. The discovery phase can be characterised as a pattern-unification problem, where traders' patterns are used to find a compatible partner. The exchange phase occurs when a buyer and seller have agreed upon a transaction. Now each trader wishes to exchange information in a single interaction, preventing any incomplete trades from occurring.

The rest of this section develops a solution in three stages. The first stage demonstrates discovery, the second introduces a registrar to validate the traders, the third extends the second with protected names to ensure privacy.

Solution 1

Consider two traders, a buyer and a seller. The buyer Buy_1 with bank account b and desired shares s can be given by

$$\mathsf{Buy}_1 \quad = \quad s \bullet \lambda m \rightarrow m \bullet b \bullet \lambda x \rightarrow B(x)\ .$$

The first pattern $s \bullet \lambda m$ is used to match with a compatible seller using share information s, and to input a name m to be used as a channel to exchange bank account information b for share certificates bound to x. The transaction successfully concludes with $B(x)$.

The seller Sell_1 with share certificates c and desired share sale s is given by

$$\mathsf{Sell}_1 \quad = \quad (\nu n)s \bullet n \rightarrow n \bullet \lambda y \bullet c \rightarrow S(y)\ .$$

The seller creates a channel name n and then tries to find a buyer for the shares described in s, offering n to the buyer to continue the transaction. The channel is then used to exchange billing information, bound to y, for the share certificates c. The seller then concludes with the successfully completed transaction as $S(y)$.

The discovery phase succeeds when the traders are placed in a parallel composition and discover each other by matching on s

$$\begin{array}{rcl}\mathsf{Buy}_1|\mathsf{Sell}_1 & \equiv & (\nu n)(s \bullet \lambda m \rightarrow m \bullet b \bullet \lambda x \rightarrow B(x) \mid s \bullet n \rightarrow n \bullet \lambda y \bullet c \rightarrow S(y)) \\ & \longmapsto & (\nu n)(n \bullet b \bullet \lambda x \rightarrow B(x) \mid n \bullet \lambda y \bullet c \rightarrow S(y))\ .\end{array}$$

The next phase is to exchange billing information for share certificates, as in

$$(\nu n)(n \bullet b \bullet \lambda x \rightarrow B(x) \mid n \bullet \lambda y \bullet c \rightarrow S(y)) \longmapsto (\nu n)(B(c) \mid S(b)).$$

The transaction concludes with the buyer having the share certificates c and the seller having the billing account b.

This solution allows the traders to discover each other and exchange information atomically to complete a transaction. However, there is no way to determine if a process is a trustworthy trader.

Solution 2

Now add a registrar that keeps track of registered traders. Traders offer their identity to potential partners and the registrar confirms if the identity belongs to a valid trader. The buyer is now

$$\mathsf{Buy}_2 \quad = \quad s \bullet i_B \bullet \lambda j \rightarrow n_B \bullet j \bullet \lambda m \rightarrow m \bullet b \bullet \lambda x \rightarrow B(x)\ .$$

The first pattern now swaps the buyer's identity i_B for the seller's, bound to j. The buyer then consults the registrar using the identifier n_B to validate j; if valid, the exchange continues as before.

Now define the seller symmetrically by

$$\mathsf{Sell}_2 \quad = \quad s \bullet \lambda j \bullet i_S \rightarrow n_S \bullet j \bullet \lambda m \rightarrow m \bullet \lambda y \bullet c \rightarrow S(y)\ .$$

Also define the registrar Reg_2 with identifiers n_B and n_S to communicate with the buyer and seller, respectively, by

$$\mathsf{Reg}_2 \quad = \quad (\nu n)(n_B \bullet \ulcorner i_S \urcorner \bullet n \mid n_S \bullet \ulcorner i_B \urcorner \bullet n)\ .$$

The registrar creates a new identifier n to provide to traders who have been validated; then it makes the identifier available to known traders who attempt to validate another known trader. Although rather simple, the registrar can easily be extended to support a multitude of traders.

Running these processes in parallel yields the following interaction

$$\begin{array}{rl} & \mathsf{Buy}_2 \mid \mathsf{Sell}_2 \mid \mathsf{Reg}_2 \\ \equiv & (\nu n)(s \bullet i_B \bullet \lambda j \to n_B \bullet j \bullet \lambda m \to m \bullet b \bullet \lambda x \to B(x) \mid n_B \bullet \ulcorner i_S \urcorner \bullet n \\ & \quad \mid s \bullet \lambda j \bullet i_S \to n_S \bullet j \bullet \lambda m \to m \bullet \lambda y \bullet c \to S(y) \mid n_S \bullet \ulcorner i_B \urcorner \bullet n) \\ \longmapsto & (\nu n)(n_B \bullet i_S \bullet \lambda m \to m \bullet b \bullet \lambda x \to B(x) \mid n_B \bullet \ulcorner i_S \urcorner \bullet n \\ & \quad \mid n_S \bullet i_B \bullet \lambda m \to m \bullet \lambda y \bullet c \to S(y) \mid n_S \bullet \ulcorner i_B \urcorner \bullet n) \ . \end{array}$$

The share information s allows the buyer and seller to discover each other and swap identities i_B and i_S. The next two interactions involve the buyer and seller validating each other's identity and inputting the identifier to complete the transaction

$$\begin{array}{rl} & (\nu n)(n_B \bullet i_S \bullet \lambda m \to m \bullet b \bullet \lambda x \to B(x) \mid n_B \bullet \ulcorner i_S \urcorner \bullet n \\ & \quad \mid n_S \bullet i_B \bullet \lambda m \to m \bullet \lambda y \bullet c \to S(y) \mid n_S \bullet \ulcorner i_B \urcorner \bullet n) \\ \longmapsto & (\nu n)(n \bullet b \bullet \lambda x \to B(x) \\ & \quad \mid n_S \bullet i_B \bullet \lambda m \to m \bullet \lambda y \bullet c \to S(y) \mid n_S \bullet \ulcorner i_B \urcorner \bullet n) \\ \longmapsto & (\nu n)(n \bullet b \bullet \lambda x \to B(x) \mid n \bullet \lambda y \bullet c \to S(y)) \ . \end{array}$$

Now that the traders have validated each other, they can continue with the exchange step from before

$$(\nu n)(n \bullet b \bullet \lambda x \to B(x) \mid n \bullet \lambda y \bullet c \to S(y)) \quad \longmapsto \quad (\nu n)(B(c) \mid S(b)) \ .$$

The traders exchange information and successfully complete with $B(c)$ and $S(b)$.

Although this solution satisfies the desire to validate that traders are legitimate, the freedom of matching allows for malicious processes to interfere. Consider the promiscuous process Prom given by

$$\mathsf{Prom} \quad = \quad \lambda z_1 \bullet \lambda z_2 \bullet a \to P(z_1, z_2) \ .$$

This process is willing to match any other process that will swap two pieces of information for some arbitrary name a. Such a process could interfere with the traders trying to complete the exchange phase of a transaction. For example,

$$\begin{array}{rl} & (\nu n)(n \bullet b \bullet \lambda x \to B(x) \mid n \bullet \lambda y \bullet c \to S(y)) \mid \mathsf{Prom} \\ \longmapsto & (\nu n)(B(a) \mid n \bullet \lambda y \bullet c \to S(y) \mid P(n, b)) \end{array}$$

where the promiscuous process has stolen the identifier n and the bank account information b. The unfortunate buyer is left with some useless information a and the seller is waiting to complete the transaction.

Solution 3

The vulnerability of Solution 2 can be repaired by using protected names. The buyer, seller and registrar can be repaired to

$$\begin{array}{rcl} \mathsf{Buy}_3 & = & s \bullet i_B \bullet \lambda j \to \ulcorner n_B \urcorner \bullet j \bullet \lambda m \to \ulcorner m \urcorner \bullet b \bullet \lambda x \to B(x) \\ \mathsf{Sell}_3 & = & s \bullet \lambda j \bullet i_S \to \ulcorner n_S \urcorner \bullet j \bullet \lambda m \to \ulcorner m \urcorner \bullet \lambda y \bullet c \to S(y) \\ \mathsf{Reg}_3 & = & (\nu n)(\ulcorner n_B \urcorner \bullet \ulcorner i_S \urcorner \bullet n \mid \ulcorner n_S \urcorner \bullet \ulcorner i_B \urcorner \bullet n) \ . \end{array}$$

Now all communication between the buyer, seller and registrar use protected identifiers: $\ulcorner n_B \urcorner, \ulcorner n_S \urcorner$ and $\ulcorner m \urcorner$. Thus, all that remains is to ensure appropriate restrictions:

$$(\nu n_B)(\nu n_S)(\mathsf{Buy}_3 \mid \mathsf{Sell}_3 \mid \mathsf{Reg}_3) .$$

Therefore, other processes can only interact with the traders during the discovery phase, which will not lead to a successful transaction. The registrar will only interact with the traders as all the registrar's patterns have protected names known only to the registrar and a trader (Lemma 2.5).

3 Behavioural Theory

This section follows a standard approach in concurrency to defining behavioural equivalences, beginning with a barbed congruence and following with a labelled transition system (LTS) and a bisimulation for CPC. Some properties of patterns will be explored as a basis for showing coincidence of the semantics and finally some equational reasoning is given.

3.1 Barbed Congruence

The first step is to characterise the interactions a CPC process can participate in via *barbs*, that provide information about its interaction capabilities. In π-calculus, barbs are defined by the channel name used for communication; because of the richer form of interactions, CPC barbs include a set of names that *may* be tested for equality in an interaction, not just those that *must* be equal. Intuitively, in CPC the barb $P \downarrow_{\widetilde{m}}$ implies the existence of a process $q \to Q$ that can interact with P and such that the protected names of q are $\widetilde{m}$. More formally, barbs are defined as follows.

Definition 3.1 (Barb)**.** *Let $P \downarrow_{\widetilde{m}}$ mean that $P \equiv (\nu\widetilde{n})(p \to P' \mid P'')$ for some $\widetilde{n}, p, P'$ and P'' such that $\mathsf{pn}(p) \cap \widetilde{n} = \emptyset$ and $\widetilde{m} = \mathsf{fn}(p)\backslash\widetilde{n}$.*

Using this definition, a barbed congruence can be defined in the standard way [29, 21] by requiring three properties. Let $\Re$ denote a binary relation on CPC processes, and let a *context* $\mathcal{C}(\cdot)$ be a CPC process with the hole '$\cdot$' replacing one instance of the null process.

Definition 3.2 (Barb preservation)**.** *$\Re$ is barb preserving iff, for every $(P, Q) \in \Re$, it holds that $P \downarrow_{\widetilde{m}}$ implies $Q \downarrow_{\widetilde{m}}$.*

Definition 3.3 (Reduction closure)**.** *$\Re$ is reduction closed iff, for every $(P, Q) \in \Re$, it holds that $P \longmapsto P'$ implies $Q \longmapsto Q'$, for some Q' such that $(P', Q') \in \Re$.*

Definition 3.4 (Context closure)**.** *$\Re$ is context closed iff, for every $(P, Q) \in \Re$ and for every CPC context $\mathcal{C}(\cdot)$, it holds that $(\mathcal{C}(P), \mathcal{C}(Q)) \in \Re$.*

Definition 3.5 (Barbed congruence)**.** *Barbed congruence, $\simeq$, is the least, symmetric, barb preserving, reduction and context closed binary relation on CPC processes.*

Barbed congruence equates processes with the same behaviour, as captured by barbs: two equivalent processes must exhibit the same behaviours, and this property should hold along every sequence of reductions and in every execution context. This defines the *strong* version of barbed congruence; its *weak* counterpart consists of replacing the predicate $\longmapsto$ with $\Longmapsto$ in Definition 3.3, and $\downarrow_{\widetilde{m}}$ with $\Downarrow_{\widetilde{m}}$ in Definition 3.2 in the usual manner [28, 29]. The following proofs are (mostly) simplified by working in the strong setting; however, everything can be rephrased in the weak setting.

The challenge in proving (strong/weak) barbed congruence is its closure under any context. So, the typical way of solving the problem is by giving a coinductive (bisimulation-based) characterization, that provides a manageable proof technique. In turn, this requires an alternative operational semantics, by means of an LTS, on top of which the bisimulation equivalence can be defined.

3.2 Labelled Transition System

The following is an adaption of the standard late LTS for the π-calculus [28]. *Labels* are defined as follows:

$$\mu \ ::= \ \tau \quad | \quad (\nu\widetilde{n})p$$

Labels are used in *transitions* $P \xrightarrow{\mu} P'$ between CPC processes, whose defining rules are given in Figure 1. If $P \xrightarrow{\mu} P'$ then P' is a *μ-reduct* of P. Rule case states that a case's pattern can be used to interact with external processes. Rule resnon is used when a restricted name does not appear in the names of the label: it simply maintains the restriction on the process after the transition. By contrast, rule resin is used when a restricted name occurs in the label: as the restricted name is going to be shared with other processes, the restriction is moved from the process to the label (this is called *extrusion*, by using a π-calculus terminology). Rule match defines when two processes can interact to perform an internal action: this can occur whenever the processes exhibit labels with unifiable patterns and with no possibility of clash or capture due to restricted names. Rule parint states that, if either process in a parallel composition can transition by an internal action, then the whole process can transition by an internal action. Rule parext is similar, but is used when the label is visible: when one of the processes in parallel exhibits an external action, then the whole composition exhibits the same external action, as long as the restricted or binding names of the label do not appear free in the parallel component that does not generate the label. Finally, rule rep unfolds the replicated process to infer the action.

Note that α-conversion is always assumed, to satisfy the side conditions whenever needed.

The presentation of the LTS is concluded with the following two results. First, for every P and μ, there are finitely many $\equiv$-equivalence classes of μ-reducts of P (Proposition 3.7). Second, the LTS induces the same operational semantics as the reductions of CPC (Proposition 3.9). As CPC reductions only involve interaction between processes and not external actions, it is sufficient to show that any internal action of the LTS is mimicked by a reduction in CPC, and visa versa.

$$\mathsf{case}: \quad (p \to P) \xrightarrow{p} P$$

$$\mathsf{resnon}: \quad \frac{P \xrightarrow{\mu} P'}{(\nu n)P \xrightarrow{\mu} (\nu n)P'} \quad n \notin \mathsf{names}(\mu)$$

$$\mathsf{resin}: \quad \frac{P \xrightarrow{(\nu\widetilde{n})p} P'}{(\nu m)P \xrightarrow{(\nu\widetilde{n},m)p} P'} \quad m \in \mathsf{vn}(p) \setminus (\widetilde{n} \cup \mathsf{pn}(p) \cup \mathsf{bn}(p))$$

$$\mathsf{match}: \quad \frac{P \xrightarrow{(\nu\widetilde{m})p} P' \quad Q \xrightarrow{(\nu\widetilde{n})q} Q'}{P \mid Q \xrightarrow{\tau} (\nu\widetilde{m},\widetilde{n})(\sigma P' \mid \rho Q')} \quad \begin{array}{l}\{p\|q\} = (\sigma,\rho)\\ \widetilde{m} \cap \mathsf{fn}(Q) = \widetilde{n} \cap \mathsf{fn}(P) = \emptyset\\ \widetilde{m} \cap \widetilde{n} = \emptyset\end{array}$$

$$\mathsf{parint}: \quad \frac{P \xrightarrow{\tau} P'}{P \mid Q \xrightarrow{\tau} P' \mid Q}$$

$$\mathsf{parext}: \quad \frac{P \xrightarrow{(\nu\widetilde{n})p} P'}{P \mid Q \xrightarrow{(\nu\widetilde{n})p} P' \mid Q} \quad (\widetilde{n} \cup \mathsf{bn}(p)) \cap \mathsf{fn}(Q) = \emptyset$$

$$\mathsf{rep}: \quad \frac{!P|P \xrightarrow{\mu} P'}{!P \xrightarrow{\mu} P'}$$

Figure 1: Labelled Transition System for CPC (the symmetric versions of parint and parext have been omitted)

Definition 3.6. *An LTS is* structurally image finite *if, for every P and μ, it holds that $\{P' : P \xrightarrow{\mu} P'\}/_{\equiv}$ contains finitely many elements.*

Proposition 3.7. *The LTS defined in Figure 1 is structurally image finite.*

Proof: First of all, let us define an alternative (but equivalent, up-to $\equiv$) LTS for CPC, written $\overset{\mu}{\twoheadrightarrow}$: it is obtained by replacing rep with the following two rules (all the other rules are the same, with $\twoheadrightarrow$ in place of $\longrightarrow$ everywhere):

$$\frac{P \overset{\mu}{\twoheadrightarrow} P'}{!P \overset{\mu}{\twoheadrightarrow} P' \mid !P} \qquad \frac{P \overset{(\nu\widetilde{m})p}{\twoheadrightarrow} P' \quad P \overset{(\nu\widetilde{n})q}{\twoheadrightarrow} P''}{!P \overset{\tau}{\twoheadrightarrow} (\nu\widetilde{m},\widetilde{n})(\sigma P' \mid \rho P'') \mid !P} \quad \begin{array}{l}\{p\|q\} = (\sigma,\rho)\\ \widetilde{m} \cap \widetilde{n} = \emptyset\end{array}$$

We can prove that: (1) if $P \overset{\mu}{\twoheadrightarrow} P'$ then $P \xrightarrow{\mu} P'$; and (2) if $P \xrightarrow{\mu} P'$ then $P \overset{\mu}{\twoheadrightarrow} P''$, for some $P'' \equiv P'$ (both proofs are done by a straightforward induction on the inference of the premise, whose only interesting case is when $P = \, !Q$, for some Q).

Now define the following measure associated to a process:

$$\|\mathbf{0}\| = 0 \qquad \|p \to P\| = 1 \qquad \|(\nu n)P\| = \|P\|$$

$$\|P_1 \mid P_2\| = \|P_1\| + \|P_2\| + \|P_1\| \cdot \|P_2\| \qquad \|!P\| = \|P\| + \|P\| \cdot \|P\|$$

By induction on the structure of P, we can prove that $|\{P' : P \xrightarrow{\mu}\!\!\!\!\rightarrow P'\}| \leq \| P \|$. By exploiting this fact and (2) above, it follows that there are finitely many (up-to $\equiv$) P' such that $P \xrightarrow{\mu} P'$. $\square$

Lemma 3.8. *If $P \xrightarrow{(\nu\widetilde{m})p} P'$ then there exist $\widetilde{n}$, Q_1 and Q_2 such that $P \equiv (\nu\widetilde{m})(\nu\widetilde{n})(p \to Q_1 \mid Q_2)$, $P' \equiv (\nu\widetilde{n})(Q_1 \mid Q_2)$, $\widetilde{n} \cap \mathsf{names}((\nu\widetilde{m})p) = \emptyset$ and $\mathsf{bn}(p) \cap \mathsf{fn}(Q_2) = \emptyset$.*

Proof: The proof is by induction on the inference for $P \xrightarrow{(\nu\widetilde{m})p} P'$. The base case is when the last rule is case, with $P = (p \to P_1) \xrightarrow{p} P_1 = P'$; conclude by taking $\widetilde{n} = \emptyset$, $Q_1 = P_1$ and $Q_2 = \mathbf{0}$. For the inductive step, consider the last rule in the inference.

- If the last rule is resnon then $P = (\nu o)P_1 \xrightarrow{(\nu\widetilde{m})p} (\nu o)P_1' = P'$, where $P_1 \xrightarrow{(\nu\widetilde{m})p} P_1'$ and $o \notin \mathsf{names}((\nu\widetilde{m})p)$. By induction, there exist $\widetilde{n}'$, Q_1' and Q_2' such that $P_1 \equiv (\nu\widetilde{m})(\nu\widetilde{n}')(p \to Q_1' \mid Q_2')$, $P_1' \equiv (\nu\widetilde{n}')(Q_1' \mid Q_2')$, $\widetilde{n}' \cap \mathsf{names}((\nu\widetilde{m})p) = \emptyset$ and $\mathsf{bn}(p) \cap \mathsf{fn}(Q_2') = \emptyset$. As $o \notin \mathsf{names}((\nu\widetilde{m})p)$ and by α-conversion $o \notin \widetilde{n}'$, conclude with $Q_1 = Q_1'$, $Q_2 = Q_2'$ and $\widetilde{n} = \widetilde{n}', o$.

- If the last rule is resin then $P = (\nu o)P_1 \xrightarrow{(\nu\widetilde{m}',o)p} P_1' = P'$, where $P_1 \xrightarrow{(\nu\widetilde{m}')p} P_1'$, $o \in \mathsf{vn}(p) \backslash (\widetilde{m}' \cup \mathsf{pn}(p) \cup \mathsf{bn}(p))$ and $\widetilde{m} = \widetilde{m}', o$. By induction, there exist $\widetilde{n}'$, Q_1' and Q_2' such that $P_1 \equiv (\nu\widetilde{m}')(\nu\widetilde{n}')(p \to Q_1' \mid Q_2')$, $P_1' \equiv (\nu\widetilde{n}')(Q_1' \mid Q_2')$, $\widetilde{n}' \cap \mathsf{names}((\nu\widetilde{m}')p) = \emptyset$ and $\mathsf{bn}(p) \cap \mathsf{fn}(Q_2') = \emptyset$. Conclude with $\widetilde{n} = \widetilde{n}'$, $Q_1 = Q_1'$ and $Q_2 = Q_2'$.

- If the last rule is parext then $P = P_1 \mid P_2 \xrightarrow{(\nu\widetilde{m})p} P_1' \mid P_2$, where $P_1 \xrightarrow{(\nu\widetilde{m})p} P_1'$ and $\mathsf{fn}(P_2) \cap (\widetilde{m} \cup \mathsf{bn}(p)) = \emptyset$. By induction, there exist $\widetilde{n}'$, Q_1' and Q_2' such that $P_1 \equiv (\nu\widetilde{m})(\nu\widetilde{n}')(p \to Q_1' \mid Q_2')$, $P_1' \equiv (\nu\widetilde{n}')(Q_1' \mid Q_2')$, $\widetilde{n}' \cap \mathsf{names}((\nu\widetilde{m})p) = \emptyset$ and $\mathsf{bn}(p) \cap \mathsf{fn}(Q_2') = \emptyset$. As $\mathsf{bn}(p) \cap \mathsf{fn}(P_2) = \emptyset$, we can conclude with $\widetilde{n} = \widetilde{n}'$, $Q_1 = Q_1'$ and $Q_2 = Q_2' \mid P_2$.

- If the last rule is rep then $P = !Q \xrightarrow{(\nu\widetilde{m})p} P'$, where $Q \mid !Q \xrightarrow{(\nu\widetilde{m})p} P'$. We easily conclude by induction and by the fact that $P \equiv Q \mid !Q$. $\square$

Proposition 3.9. *If $P \xrightarrow{\tau} P'$ then $P \longmapsto P'$. Conversely, if $P \longmapsto P'$ then there exists P'' such that $P \xrightarrow{\tau} P'' \equiv P'$.*

Proof: The first claim is proved by induction on the inference for $P \xrightarrow{\tau} P'$. The base case is with rule match: $P = P_1 \mid Q_1$, where $P_1 \xrightarrow{(\nu\widetilde{m})p} P_1'$, $Q_1 \xrightarrow{(\nu\widetilde{n})q} Q_1'$, $P' = (\nu\widetilde{m},\widetilde{n})(\sigma P_1' \mid \rho Q_1')$, $\{p\|q\} = (\sigma,\rho)$, $\widetilde{m} \cap \mathsf{fn}(Q_1) = \widetilde{n} \cap \mathsf{fn}(P_1) = \emptyset$ and $\widetilde{m} \cap \widetilde{n} = \emptyset$. By Lemma 3.8, it follows that $P_1 \equiv (\nu\widetilde{m})(\nu\widetilde{o})(p \to P_1'' \mid P_2'')$ and $P_1' \equiv (\nu\widetilde{o})(P_1'' \mid P_2'')$, with $\widetilde{o} \cap \mathsf{names}((\nu\widetilde{m})p) = \emptyset$ and $\mathsf{bn}(p) \cap \mathsf{fn}(P_2'') = \emptyset$; similarly, $Q_1 \equiv (\nu\widetilde{n})(\nu\widetilde{r})(q \to Q_1'' \mid Q_2'')$ and $Q_1' \equiv (\nu\widetilde{r})(Q_1'' \mid Q_2'')$, with $\widetilde{r} \cap \mathsf{names}((\nu\widetilde{n})q) = \emptyset$ and $\mathsf{bn}(q) \cap \mathsf{fn}(Q_2'') = \emptyset$. By exploiting α-conversion on the names in $\widetilde{o}, \widetilde{r}$, we have $\widetilde{o}, \widetilde{r} \cap (\mathsf{names}((\nu\widetilde{m})p) \cup \mathsf{names}((\nu\widetilde{n})q)) = \emptyset$; thus, $P_1 \mid Q_1 \equiv (\nu\widetilde{m},\widetilde{n})(\nu\widetilde{o},\widetilde{r})(p \to P_1'' \mid P_2'' \mid q \to Q_1'' \mid Q_2'') \longmapsto (\nu\widetilde{m},\widetilde{n})(\nu\widetilde{o},\widetilde{r})(\sigma P_1'' \mid P_2'' \mid \rho Q_1'' \mid Q_2'')$. Since σ avoids $\widetilde{o}$, $\mathsf{dom}(\sigma) \cap \mathsf{fn}(P_2'') = \emptyset$, ρ avoids $\widetilde{r}$, $\mathsf{dom}(\rho) \cap \mathsf{fn}(Q_2'') = \emptyset$ and $\widetilde{o} \cap \mathsf{fn}(Q_1'' \mid Q_2'') = \widetilde{r} \cap \mathsf{fn}(P_1'' \mid P_2'') = \emptyset$, conclude $P \longmapsto (\nu\widetilde{m},\widetilde{n})(\nu\widetilde{o},\widetilde{r})(\sigma P_1'' \mid P_2'' \mid \rho Q_1'' \mid Q_2'') \equiv (\nu\widetilde{m},\widetilde{n})(\sigma((\nu\widetilde{o})(P_1'' \mid P_2'')) \mid \rho((\nu\widetilde{r})(Q_1'' \mid Q_2''))) \equiv (\nu\widetilde{m},\widetilde{n})(\sigma P_1' \mid \rho Q_1') = P'$.

For the inductive step, reason on the last rule used in the inference.

- If the last rule is parint then $P = P_1 \mid P_2$, for $P_1 \xrightarrow{\tau} P_1'$ and $P' = P_1' \mid P_2$. Apply induction to the transition $P_1 \xrightarrow{\tau} P_1'$ to obtain that $P_1 \longmapsto P_1'$; thus, $P \longmapsto P'$.

- If the last rule is resnon then $P = (\nu n)P_1$, for $P_1 \xrightarrow{\tau} P_1'$ and $P' = (\nu n)P_1'$. Again, trivial by induction.

- If the last rule is rep then $P = !P_1$, for $P_1 \mid !P_1 \xrightarrow{\tau} P'$. By induction, $P_1 \mid !P_1 \longmapsto P'$ and easily conclude, since $P \equiv P_1 \mid !P_1$.

The second claim is by induction on the inference for $P \longmapsto P'$. The base case is when $P = p \to P_1' \mid q \to Q_1'$ and $P' = \sigma P_1' \mid \rho Q_1'$, for $\{p \| q\} = (\sigma, \rho)$. By the match rule in the LTS

$$\frac{(p \to P_1') \xrightarrow{p} P_1' \qquad (q \to Q_1') \xrightarrow{q} Q_1'}{p \to P_1' \mid q \to Q_1' \ \xrightarrow{\tau}\ \sigma P_1' \mid \rho Q_1'} \ \{p\|q\} = (\sigma, \rho)$$

and the result is immediate. For the inductive step, reason on the last rule used in the inference.

- If $P = P_1 \mid P_2$, where $P_1 \longmapsto P_1'$ and $P' = P_1' \mid P_2$, then use the induction and exploit the parint rule.

- If $P = (\nu n)P_1$, where $P_1 \longmapsto P_1'$ and $P' = (\nu n)P_1'$, then use the induction and exploit the resnon rule.

- Otherwise, it must be that $P \equiv Q \longmapsto Q' \equiv P'$. By induction, $Q \xrightarrow{\tau} Q'$ for some $Q'' \equiv Q'$. We now have to prove that structurally equivalent processes have the same τ-transitions, up-to $\equiv$; this is done via a second induction, on the inference of the judgement $P \equiv Q$. The following are two representative base cases; the other ones are easier, as well as the inductive case.

  - $P = !R \equiv R \mid !R = Q$: since $Q = R \mid !R \xrightarrow{\tau} Q''$, for $Q'' \equiv Q'$, we can use rule rep of the LTS and obtain $P \xrightarrow{\tau} Q''$; we can conclude, since $Q'' \equiv Q' \equiv P'$.
  - $P = (\nu n)P_1 \ \mid\ P_2 \equiv (\nu n)(P_1 \mid\ P_2) = Q$, that holds since $n \notin \mathsf{fn}(P_2)$: by the first inductive hypothesis, $(\nu n)(P_1 \mid\ P_2) \xrightarrow{\tau} Q''$, for $Q'' \equiv Q'$. Moreover, by definition of the LTS, the last rule used in this inference must be resnon; thus, $P_1 \mid\ P_2 \xrightarrow{\tau} Q'''$ and $Q'' = (\nu n)Q'''$. There are three possible ways to generate the latter τ-transition:
    * $P_1 \xrightarrow{\tau} P_1'$ and $Q''' = P_1' \mid P_2$: in this case

    $$\frac{\dfrac{P_1 \xrightarrow{\tau} P_1'}{(\nu n)P_1 \xrightarrow{\tau} (\nu n)P_1'}}{P = (\nu n)P_1 \mid P_2 \xrightarrow{\tau} (\nu n)P_1' \mid P_2}$$

    and conclude by noticing that $(\nu n)P_1' \mid P_2 \equiv (\nu n)(P_1' \mid P_2) = Q'' \equiv Q' \equiv P'$.
    * $P_2 \xrightarrow{\tau} P_2'$ and $Q''' = P_1 \mid P_2'$: this case is similar to the previous one, but simpler.

* $P_1 \xrightarrow{(\nu\widetilde{m})p} P_1'$, $P_2 \xrightarrow{(\nu\widetilde{n})q} P_2'$, and $Q''' = (\nu\widetilde{m},\widetilde{n})(\sigma P_1' \mid \rho P_2')$, where $\{p\|q\} = (\sigma,\rho)$, $\widetilde{m}\cap\mathsf{fn}(P_2) = \widetilde{n}\cap\mathsf{fn}(P_1) = \emptyset$ and $\widetilde{m}\cap\widetilde{n} = \emptyset$: this case is similar to the base case of the first claim of this Proposition and, essentially, relies on Lemma 3.8. The details are left to the interested reader.

$\square$

3.3 Bisimulation

The next step is to develop a *bisimulation* relation for CPC that equates processes with the same interactional behaviour as captured by the labels of the LTS. The complexity for CPC is that the labels for external actions contain patterns, and some patterns are more general than others. For example, a transition $P \xrightarrow{\ulcorner n \urcorner} P'$ performs the action $\ulcorner n \urcorner$; however a similar external action of another process could be the variable name n and the transition $Q \xrightarrow{n} Q'$. Both transitions have the same barb, that is $P \downarrow_n$ and $Q \downarrow_n$; however their labels are not identical and, indeed, the latter can interact with a process performing a transition labeled with λx whereas the former cannot. Thus, a *compatibility* relation is defined on patterns that can be used to develop the bisimulation. The rest of this section discusses the development of compatibility and concludes with the definition of bisimulation for CPC.

Bisimilarity of two processes P and Q can be captured by a challenge-reply game based upon the actions the processes can take. One process, say P, issues a *challenge* and evolves to a new state P'. Now Q must perform an action that is a *proper reply* and evolve to a state Q'. If Q cannot perform a proper reply then the challenge issued by P can distinguish P and Q, and shows they are not equivalent. If Q can properly reply then the game continues with the processes P' and Q'. Two processes are bisimilar (or equivalent) if the game can always continue, or neither process can perform any action.

The main complexity in defining a bisimulation to capture this challenge-reply game is the choice of actions, i.e. challenges and replies. In most process calculi, a challenge is replied to with an identical action [26, 28]. However, there are situations in which an exact reply would make the bisimulation equivalence too fine for characterising barbed congruence [4, 11]. This is due to the impossibility for the language contexts to force barbed congruent processes to execute the same action; in such calculi more liberal replies must be allowed. That CPC lies in this second group of calculi is demonstrated by the following two examples.

Example 1 Consider the processes

$$P = \lambda x \bullet \lambda y \rightarrow x \bullet y \qquad \text{and} \qquad Q = \lambda z \rightarrow z$$

together with the challenge $P \xrightarrow{\lambda x \bullet \lambda y} x \bullet y$. One may think that a possible context $\mathcal{C}_{\lambda x \bullet \lambda y}(\cdot)$ to enforce a proper reply could be $\cdot \mid w \bullet w \rightarrow \ulcorner w \urcorner$, for w fresh. Indeed, $\mathcal{C}_{\lambda x \bullet \lambda y}(P) \longmapsto w \bullet w \mid \ulcorner w \urcorner$ and the latter process exhibits a barb over w. However, the exhibition of action $\lambda x \bullet \lambda y$ is *not* necessary for the production of such a barb: indeed, $\mathcal{C}_{\lambda x \bullet \lambda y}(Q) \longmapsto w \bullet w \mid \ulcorner w \urcorner$, but in doing so Q performs λz instead of $\lambda x \bullet \lambda y$.

Example 2 Consider the processes

$$P = \ulcorner n \urcorner \to \mathbf{0} \qquad \text{and} \qquad Q = n \to \mathbf{0}$$

together with the context $\mathcal{C}_{\ulcorner n \urcorner}(\cdot) = n \to \ulcorner w \urcorner$, for w fresh. Although $\mathcal{C}_{\ulcorner n \urcorner}(P) \longmapsto \ulcorner w \urcorner$ and the latter process exhibits a barb over w, the exhibition of action $\ulcorner n \urcorner$ is *not* necessary for the production of such a barb: $\mathcal{C}_{\ulcorner n \urcorner}(Q) \longmapsto \ulcorner w \urcorner$ also exhibits a barb on w, but in doing so Q performs n instead of $\ulcorner n \urcorner$.

Example 1 shows that CPC pattern-unification allows binding names to be contractive: it is not necessary to fully decompose a pattern to bind it. Thus a compound pattern may be bound to a single name or to more than one name in unification. Example 2 illustrates that CPC pattern-unification on protected names only requires the other pattern know the name, but such a name is not necessarily protected in the reply.

These two observations make it clear that some patterns are more discerning than others, i.e. match fewer patterns than others. This leads to the following definition.

Definition 3.10. *Let p and q be patterns each with a* linked *substitution, σ and ρ respectively, such that $\mathsf{bn}(p) = \mathsf{dom}(\sigma)$ and $\mathsf{bn}(q) = \mathsf{dom}(\rho)$. Define that p is* compatible *with q by σ and ρ, denoted $p, \sigma \ll q, \rho$ by induction as follows:*

$$\begin{array}{rcll}
p, \sigma & \ll & \lambda y, \{\hat{\sigma} p / y\} & \text{if } \mathsf{fn}(p) = \emptyset \\
n, \{\} & \ll & n, \{\} & \\
\ulcorner n \urcorner, \{\} & \ll & \ulcorner n \urcorner, \{\} & \\
\ulcorner n \urcorner, \{\} & \ll & n, \{\} & \\
p_1 \bullet p_2, \sigma_1 \cup \sigma_2 & \ll & q_1 \bullet q_2, \rho_1 \cup \rho_2 & \text{if } p_i, \sigma_i \ll q_i, \rho_i, \text{ for } i \in \{1, 2\}
\end{array}$$

The idea behind this definition is that a pattern p is compatible with another pattern q if and only if every other pattern r that matches p by some substitutions (θ, σ) also matches q with substitutions (θ, ρ) such that $p, \sigma \ll q, \rho$. That is, the patterns that match against p are a subset of the patterns that match against q. This will be proved later in Proposition 3.16.

The compatibility relation on patterns provides the concept of proper reply in the challenge-reply game.

Definition 3.11 (Bisimulation)**.** *A symmetric binary relation on CPC processes $\Re$ is a bisimulation if, for every $(P, Q) \in \Re$ and $P \xrightarrow{\mu} P'$, it holds that:*

- *if $\mu = \tau$, then $Q \xrightarrow{\tau} Q'$, for some Q' such that $(P', Q') \in \Re$;*
- *if $\mu = (\nu \widetilde{n})p$ and $(\mathsf{bn}(p) \cup \widetilde{n}) \cap \mathsf{fn}(Q) = \emptyset$, then, for all σ with $\mathsf{dom}(\sigma) = \mathsf{bn}(p)$ and $\mathsf{fn}(\sigma) \cap \widetilde{n} = \emptyset$, there exist q and Q' and ρ such that $Q \xrightarrow{(\nu \widetilde{n})q} Q'$ and $p, \sigma \ll q, \rho$ and $(\sigma P', \rho Q') \in \Re$.*

Denote with $\sim$ the largest bisimulation closed under any substitution.

The definition is inspired by the early bisimulation congruence for the π-calculus [28]: for every possible instantiation σ of the binding names, there exists a proper reply from Q. Of course, σ cannot be chosen arbitrarily: it cannot use in its range names that were restricted in P. Also the action μ cannot be arbitrary, as in the π-calculus: its restricted and binding names cannot occur free in Q. Differently from the π-calculus, however, the reply from Q can be different from the challenge from P: this is due to the fact that contexts in CPC are not powerful enough to enforce an identical reply (as highligthed in Examples 1 and 2). Indeed, this notion of bisimulation allows a challenge p to be replied to by any compatible q, provided that σ is properly adapted (yielding ρ, as described by the compatibility relation) before being applied to Q'.

3.4 Properties of the Ordering on Patterns

This section considers some properties of the compatibility relation on patterns introduced in Section 3.3; they are formalised for later exploitation, even though some of them also illustrate some general features of patterns. In particular, we show that compatibility preserves information used for barbs, is stable under substitution, is reflexive and transitive.

Proposition 3.12. *If* $p, \sigma \ll q, \rho$ *then* $\mathsf{fn}(p) = \mathsf{fn}(q)$*,* $\mathsf{vn}(p) \subseteq \mathsf{vn}(q)$ *and* $\mathsf{pn}(q) \subseteq \mathsf{pn}(p)$*.*

Proof: By definition of compatibility and induction on the structure of q. $\square$

Given two substitutions σ and θ, denote with $\theta[\sigma]$ the composition of σ and θ, with domain limited to the domain of σ, i.e. the substitution mapping every $x \in \mathsf{dom}(\sigma)$ to $\theta(\sigma(x))$.

Lemma 3.13. *If* $p, \sigma \ll q, \rho$ *then* $p, \theta[\sigma] \ll q, \theta[\rho]$*, for every* θ*.*

Proof: By induction on the structure of q. The only non-trivial base case is when $q = \lambda y$; in this case, $p = \lambda x_1 \bullet \ldots \lambda x_n$, for some $n \geq 1$, and for $\rho = \{\hat{\sigma}p/y\} = \{\sigma(x_1) \bullet \ldots \bullet \sigma(x_n)/y\}$. Since $\mathsf{dom}(\theta[\sigma]) = \mathsf{dom}(\sigma)$, we have that $p, \theta[\sigma] \ll q, \vartheta$, for $\vartheta = \{\widehat{\theta[\sigma]}(p)/y\} = \{\theta[\sigma](x_1) \bullet \ldots \bullet \theta[\sigma](x_n)/y\} = \{\theta(\sigma(x_1) \bullet \ldots \bullet \sigma(x_n))/y\} = \{\theta(\rho(y))/y\} = \theta[\rho]$. $\square$

Proposition 3.14 (Compatibility is reflexive)**.** *For all patterns* p *and substitutions* σ *whose domain is* $\mathsf{bn}(p)$*, it holds that* $p, \sigma \ll p, \sigma$*.*

Proof: Trivial by definition of compatibility. $\square$

Proposition 3.15 (Compatibility is transitive)**.** $p, \sigma \ll q, \rho$ *and* $q, \rho \ll r, \theta$ *imply* $p, \sigma \ll r, \theta$*.*

Proof: By induction on r. We have three possible base cases:

- $r = \lambda z$: in this case, $q = \lambda y_1 \bullet \ldots \bullet \lambda y_n$, for some $n \geq 1$, and

  $$\theta = \{\hat{\rho}q/z\} = \{\rho(y_1) \bullet \ldots \bullet \rho(y_n)/z\}.$$

  Again by definition of compatibility, $p = \lambda x_1^1 \bullet \ldots \bullet \lambda x_1^{k_1} \bullet \ldots \bullet \lambda x_n^1 \bullet \ldots \bullet \lambda x_n^{k_n}$, for some $k_1, .., k_n \geq 1$, and

  $$\rho = \{\hat{\sigma}(x_i^1 \bullet \ldots \bullet x_i^{k_i})/y_i\}_{i=1,..,n} = \{\sigma(x_i^1) \bullet \ldots \bullet \sigma(x_i^{k_i})/y_i\}_{i=1,..,n}.$$

  Thus, $\theta = \{\sigma(x_1^1) \bullet \ldots \bullet \sigma(x_1^{k_1}) \bullet \ldots \bullet \sigma(x_n^1) \bullet \ldots \bullet \sigma(x_n^{k_n})/z\} = \{\hat{\sigma}p/z\}$ and $p, \sigma \ll r, \theta$, as desired.

- $r = \ulcorner n \urcorner$: in this case $q = \ulcorner n \urcorner$ and $\theta = \rho = \{\}$. Again by compatibility, $p = \ulcorner n \urcorner$ and $\sigma = \{\}$; thus $p, \sigma \ll r, \theta$.

- $r = n$: in this case q can either be $\ulcorner n \urcorner$ or n, and $\theta = \rho = \{\}$. Again by compatibility, $p = \ulcorner n \urcorner$ or $p = n$ (this is possible only when $q = n$), and $\sigma = \{\}$; in all cases, $p, \sigma \ll r, \theta$.

For the inductive step, let $r = r_1 \bullet r_2$. By compatibility, $q = q_1 \bullet q_2$, $\theta = \theta_1 \cup \theta_2$ and $\rho = \rho_1 \cup \rho_2$, with $q_i, \rho_i \ll r_i, \theta_i$, for $i = 1, 2$. Similarly, $p = p_1 \bullet p_2$ and $\sigma = \sigma_1 \cup \sigma_2$, with $p_i, \sigma_i \ll q_i, \rho_i$, for $i = 1, 2$. By two applications of the inductive hypothesis, we obtain $p_i, \sigma_i \ll r_i, \theta_i$, for $i = 1, 2$, and by definition of compatibility we can conclude. $\square$

The next result captures the idea behind the definition of compatibility: the patterns matched by p are a subset of the patterns matched by q.

Lemma 3.16. *If $p, \sigma \ll q, \rho$ then, for every r and θ such that $\{r \| p\} = (\theta, \sigma)$, we have that $\{r \| q\} = (\theta, \rho)$.*

Proof: The proof is by induction on q. There are three possible base cases:

- If $q = \lambda y$ then $\mathsf{fn}(p) = \emptyset$ and $\rho = \{\hat{\sigma}p/y\}$; for the unification of r and p to be defined, it must be $\theta = \{\}$, $\sigma = \{r_i/x_i\}_{x_i \in \mathsf{bn}(p)}$, each r_i is communicable and $\hat{\sigma}p = r$. It follows that $\{r \| q\} = (\{\}, \{r/y\}) = (\{\}, \{\hat{\sigma}p/y\}) = (\theta, \rho)$.

- If $q = \ulcorner n \urcorner$ then $p = \ulcorner n \urcorner$ and $\sigma = \rho = \{\}$. For r to unify with p, it must be that r is n or $\ulcorner n \urcorner$; in both cases $\theta = \{\}$. Hence, $\{r \| q\} = (\{\}, \{\}) = (\theta, \rho)$.

- If $q = n$ then p is either n or $\ulcorner n \urcorner$, and $\sigma = \rho = \{\}$. In both cases, r can as well be either n or $\ulcorner n \urcorner$. The proof is similar to the previous case.

For the inductive step, $q = q_1 \bullet q_2$; by comparability, $p = p_1 \bullet p_2$. There are two possible cases for r to match with p:

- If $r = \lambda z$, then p must be communicable and $\theta = \{p/z\}$; thus, by definition of comparability, $q = p$ and $\sigma = \rho = \{\}$. Hence, $\{r \| q\} = (\{q/z\}, \{\}) = (\{p/z\}, \{\}) = (\theta, \rho)$.

- Otherwise, for r to unify with p, it must be $r = r_1 \bullet r_2$ with $\{r_i \| p_i\} = (\theta_1, \sigma_i)$, for $i \in \{1, 2\}$, and $\sigma = \sigma_1 \uplus \sigma_2$ and $\theta = \theta_1 \uplus \theta_2$. Conclude by two applications of induction and by definition of compatibility. $\square$

Notice that the converse does not hold. Take $p = n$ and $q = \ulcorner n \urcorner$; we have that, for every r such that $\{r \| p\} = (\theta, \sigma)$, we have that $\{r \| q\} = (\theta, \rho)$ (the only such r's are n and $\ulcorner n \urcorner$, for which $\sigma = \theta = \rho = \{\}$); however, $n, \{\} \not\ll \ulcorner n \urcorner, \{\}$.

The following result is a variation of the previous lemma, that fixes σ to $\mathsf{id}_{\mathsf{bn}(p)}$ but allows an arbitrary substitution in the matching with r.

Lemma 3.17. *If $p, \mathsf{id}_{\mathsf{bn}(p)} \ll q, \rho$ and $\{p \| r\} = (\vartheta, \theta)$, then $\{q \| r\} = (\vartheta[\rho], \theta)$,*

Proof: By induction on q. There are three possible base cases:

- $q = \lambda x$: by Definition 3.10, it must be that $\mathsf{fn}(p) = \emptyset$, i.e. $p = \lambda x_1 \bullet \ldots \bullet \lambda x_k$, for some k. Thus, $\rho = \{x_1 \bullet \ldots \bullet x_k/x\}$, $r = r_1 \bullet \ldots \bullet r_k$ communicable, $\vartheta = \{r_1/x_1 \ldots r_k/x_k\}$ and $\theta = \{\}$. By definition of matching, $\{q \| r\} = (\{r/x\}, \{\})$ and conclude, since $\{r/x\} = \{r_1 \bullet \ldots \bullet r_k/x\} = \vartheta[\rho]$.

- $q = n$: in this case, it must be either $p = n$ or $p = \ulcorner n \urcorner$; in both cases, $\rho = \{\}$. If $p = n$, then trivially conclude, since $q = p$ and $\vartheta[\rho] = \vartheta$. If $p = \ulcorner n \urcorner$, obtain that r can be either n or $\ulcorner n \urcorner$; in both cases $\vartheta = \theta = \{\}$ and trivially conclude.

- $q = \ulcorner n \urcorner$: in this case $p = \ulcorner n \urcorner$, $\rho = \{\}$ and work like in the previous case.

For the inductive case, $q = q_1 \bullet q_2$; thus, by Definition 3.10, $p = p_1 \bullet p_2$ and $p_i, \mathsf{id}_{\mathsf{bn}(p_i)} \ll q_i, \rho_i$, where $\rho_i = \rho|_{\mathsf{bn}(q_i)}$, for $i \in \{1, 2\}$. We have two possibilities for r:

- $r = r_1 \bullet r_2$, where $\{p_i \| r_i\} = (\vartheta_i, \theta_i)$, for $i \in \{1, 2\}$; moreover, $\vartheta = \vartheta_1 \uplus \vartheta_2$ and $\theta = \theta_1 \uplus \theta_2$. Apply induction two times and easily conclude.

- $r = \lambda x$ and p is communicable; thus, $\vartheta = \{\}$ and $\theta = \{p/x\}$. By definition of compatibility, $q = p$ and $\rho = \{\}$. Trivially conclude. □

As compatibility is an ordering upon patterns it is interesting to observe that for every pattern p there is a unique (up-to α-conversion of binding names) maximal pattern with respect to $\ll$.

Proposition 3.18. *For every pattern p there exists a maximal pattern q with respect to $\ll$; this pattern is unique up-to α-conversion of binding names.*

Proof: The proof is by induction on the structure of p:

- If $\mathsf{fn}(p) = \emptyset$, then $q = \lambda y$ for some fresh y.

- If p is n or $\ulcorner n \urcorner$, then q is n.

- If $p = p_1 \bullet p_2$, for $\mathsf{fn}(p) \neq \emptyset$, then proceed by induction on p_1 and p_2.

The only arbitrary choice is the y used in the first item, that can be α-converted to any other fresh name. □

To conclude the properties of the compatibility, it is worth remarking it does not yield a lattice: there is no supremum for the two patterns λx and n.

3.5 Soundness of the Bisimulation

This section proves soundness by showing that the bisimulation relation is included in barbed congruence. This is done by showing that the bisimilarity relation is an equivalence, it is barb preserving, reduction closed and context closed. The first three facts are ensured by the following three lemmas.

Lemma 3.19. *If $P \sim Q$ and $Q \sim R$ then $P \sim R$.*

Proof: Standard, by exploiting Proposition 3.15. □

Lemma 3.20. *$\sim$ is barb preserving.*

Proof: Straightforward by Proposition 3.12 and Lemma 3.8. □

Lemma 3.21. *$\sim$ is reduction closed.*

Proof: Trivial by Proposition 3.9. $\square$

Closure under any context is less easy to prove. The approach here is as follows: prove bisimilarity is closed under case prefixing; then prove closure under name restriction and parallel composition (as in π-calculus, it is necessary to simultaneously handle these two operators, because of name extrusion); finally, prove closure under replication. These three results will easily entail closure under arbitrary contexts (Lemma 3.25).

Lemma 3.22. *If $P \sim Q$ then $p \to P \sim p \to Q$.*

Proof: It is necessary to prove that the relation

$$\Re = \{(p \to P, p \to Q) \ : \ P \sim Q\} \cup \ \sim$$

is a bisimulation. The only possible challenge of $p \to P$ is $p \to P \xrightarrow{p} P$ such that $\mathsf{bn}(p) \cap \mathsf{fn}(Q) = \emptyset$; moreover, fix any σ such that $\mathsf{dom}(\sigma) = \mathsf{bn}(p)$. The only possible reply from $p \to Q$ is $p \to Q \xrightarrow{p} Q$, that is a valid reply (in the sense of Definition 3.11). Indeed, $p, \sigma \ll p, \sigma$, by Proposition 3.14, and $(\sigma P, \sigma Q) \in \Re$, because $P \sim Q$ and $\sim$ is closed under substitutions by definition. $\square$

Lemma 3.23. *If $P \sim Q$ then $(\nu \widetilde{n})(P \mid R) \sim (\nu \widetilde{n})(Q \mid R)$.*

Proof: It is necessary to prove that the relation

$$\Re = \{((\nu \widetilde{n})(P \mid R), (\nu \widetilde{n})(Q \mid R)) \ : \ P \sim Q\}$$

is a bisimulation. Fix any transition $(\nu \widetilde{n})(P \mid R) \xrightarrow{\mu} \hat{P}$ that, by definition of the LTS, has been inferred as follows:

$$\dfrac{\dfrac{P \mid R \xrightarrow{\bar{\mu}} \bar{P}}{\vdots}}{(\nu \widetilde{n})(P \mid R) \xrightarrow{\mu} \hat{P}} \qquad (\star)$$

where $\mu = (\nu \widetilde{m})\bar{\mu}$, $\hat{P} = (\nu \ \widetilde{n} \backslash \widetilde{m})\bar{P}$ and the dots denote repeated applications of resnon (one for every name in $\widetilde{n} \setminus \widetilde{m}$) and resin (one for every name in $\widetilde{m}$).

If $\bar{\mu} = \tau$, then $\widetilde{m} = \emptyset$; moreover, $P \mid R \xrightarrow{\bar{\mu}} \bar{P}$ can be generated in three ways:

- If the transition is
$$\dfrac{P \xrightarrow{\tau} P'}{P \mid R \xrightarrow{\tau} P' \mid R}$$
then by $P \sim Q$ there exists $Q \xrightarrow{\tau} Q'$ such that $P' \sim Q'$; conclude with $(\nu \widetilde{n})(Q \mid R) \xrightarrow{\tau} (\nu \widetilde{n})(Q' \mid R)$.

- If the transition is
$$\dfrac{R \xrightarrow{\tau} R'}{P \mid R \xrightarrow{\tau} P \mid R'}$$
consider $(\nu \widetilde{n})(Q \mid R) \xrightarrow{\tau} (\nu \widetilde{n})(Q \mid R')$ and conclude.

- If the transition is
$$\frac{P \xrightarrow{(\nu\widetilde{l})p} P' \quad R \xrightarrow{(\nu\widetilde{o})r} R'}{P \mid R \xrightarrow{\tau} (\nu\widetilde{l},\widetilde{o})(\sigma P' \mid \theta R')}$$
with $\{p\|r\} = (\sigma,\theta)$ and $\widetilde{l} \cap \mathsf{fn}(R) = \widetilde{o} \cap \mathsf{fn}(P) = \widetilde{l} \cap \widetilde{o} = \emptyset$. Now, there exist q, Q' and ρ such that $Q \xrightarrow{(\nu\widetilde{l})q} Q'$, $p,\sigma \ll q,\rho$ and $\sigma P' \sim \rho Q'$. By Lemma 3.16, $\{q\|r\} = (\rho,\theta)$ and so
$$\frac{Q \xrightarrow{(\nu\widetilde{l})q} Q' \quad R \xrightarrow{(\nu\widetilde{o})r} R'}{Q \mid R \xrightarrow{\tau} (\nu\widetilde{l},\widetilde{o})(\rho Q' \mid \theta R')}$$
where, by α-conversion, we can always let $\widetilde{o} \cap \mathsf{fn}(Q) = \emptyset$ (the other side conditions for applying rule match already hold). By repeated applications of rule resnon, infer $(\nu\widetilde{n})(Q \mid R) \xrightarrow{\tau} (\nu\widetilde{n})(\nu\widetilde{l},\widetilde{o})(\rho Q' \mid \theta R')$ and easily conclude.

If $\bar{\mu} = (\nu\widetilde{l})p$, it must be that $(\mathsf{bn}(p) \cup \widetilde{l}) \cap \mathsf{fn}((\nu\widetilde{n})(Q \mid R)) = \emptyset$. Then, fix any σ such that $\mathsf{dom}(\sigma) = \mathsf{bn}(p)$ and $\mathsf{fn}(\sigma) \cap \widetilde{l} = \emptyset$. The transition $P \mid R \xrightarrow{\bar{\mu}} \bar{P}$ can be now generated in two ways:

- The transition is
$$\frac{P \xrightarrow{(\nu\widetilde{l})p} P'}{P \mid R \xrightarrow{(\nu\widetilde{l})p} P' \mid R} \; (\widetilde{l} \cup \mathsf{bn}(p)) \cap \mathsf{fn}(R) = \emptyset$$
By $P \sim Q$ there exist q, Q' and ρ such that $Q \xrightarrow{(\nu\widetilde{l})q} Q'$, $p,\sigma \ll q,\rho$ and $\sigma P' \sim \rho Q'$. By α-equivalence, let $\mathsf{bn}(q) \cap \mathsf{fn}(R) = \emptyset$; thus, $Q \mid R \xrightarrow{(\nu\widetilde{l})q} Q' \mid R$. By applying the same sequence of rules resnon and resin used for $(\star)$ (this is possible since $\mathsf{fn}(p) = \mathsf{fn}(q)$, see Proposition 3.12), conclude with $(\nu\widetilde{n})(Q \mid R) \xrightarrow{(\nu\widetilde{l},\widetilde{m})q} (\nu\;\widetilde{n}\backslash\widetilde{m})(Q' \mid R) = \hat{Q}$. Since $\mathsf{dom}(\sigma) \cap \mathsf{fn}(R) = \mathsf{bn}(p) \cap \mathsf{fn}(R) = \emptyset$ and substitution application is capture-avoiding by definition, obtain that $\sigma\hat{P} = \sigma((\nu\;\widetilde{n}\backslash\widetilde{m})(P' \mid R)) = (\nu\;\widetilde{n}\backslash\widetilde{m})(\sigma P' \mid R)$. Similarly, $\rho\hat{Q} = (\nu\;\widetilde{n}\backslash\widetilde{m})(\rho Q' \mid R)$. This suffices to conclude $(\sigma\hat{P}, \rho\hat{Q}) \in \Re$, as desired.

- The transition is
$$\frac{R \xrightarrow{(\nu\widetilde{l})p} R'}{P \mid R \xrightarrow{(\nu\widetilde{l})p} P \mid R'} \; (\widetilde{l} \cup \mathsf{bn}(p)) \cap \mathsf{fn}(P) = \emptyset$$
By α-equivalence, let $(\widetilde{l} \cup \mathsf{bn}(p)) \cap \mathsf{fn}(Q) = \emptyset$; this allows us to infer $Q \mid R \xrightarrow{(\nu\widetilde{l})p} Q \mid R'$. Now, by the same sequence of rules resnon and resin used for $(\star)$, we obtain $(\nu\widetilde{n})(Q \mid R) \xrightarrow{(\nu\widetilde{l},\widetilde{m})p} (\nu\;\widetilde{n}\backslash\widetilde{m})(Q \mid R') = \hat{Q}$. By Proposition 3.14, $p,\sigma \ll p,\sigma$. Moreover, since $\mathsf{dom}(\sigma) \cap \mathsf{fn}(P,Q) = \emptyset$ and substitution application is capture-avoiding, obtain that $\sigma\hat{P} = (\nu\;\widetilde{n}\backslash\widetilde{m})(P \mid \sigma R')$ and $\sigma\hat{Q} = (\nu\;\widetilde{n}\backslash\widetilde{m})(Q \mid \sigma R')$. This suffices to conclude $(\sigma\hat{P}, \sigma\hat{Q}) \in \Re$, as desired.

$\square$

Lemma 3.24. *If $P \sim Q$ then $!P \sim !Q$.*

Proof: This proof rephrases the similar one in [33]. First, define the n-th approximation of the bisimulation:

$$\begin{array}{rcl}
\sim_0 & = & Proc \times Proc \\
\sim_{n+1} & = & \{(P,Q) \; : \\
& & \quad \forall \; P \xrightarrow{\mu} P' \\
& & \qquad \mu = \tau \;\Rightarrow\; \exists \; Q \xrightarrow{\tau} Q'.\ (P',Q') \in \sim_n \\
& & \qquad \mu = (\nu\widetilde{n})p \;\Rightarrow\; \forall \sigma \; s.t. \; \mathsf{dom}(\sigma) = \mathsf{bn}(p) \land \\
& & \qquad\qquad \mathsf{fn}(\sigma) \cap \widetilde{n} = \emptyset \land \\
& & \qquad\qquad (\mathsf{bn}(p) \cup \widetilde{n}) \cap \mathsf{fn}(Q) = \emptyset \\
& & \qquad\qquad \exists \; q, Q', \rho \; s.t. \; Q \xrightarrow{(\nu\widetilde{n})q} Q' \land \\
& & \qquad\qquad p, \sigma \ll q, \rho \land (\sigma P', \rho Q') \in \sim_n \\
& & \quad \text{Symmetrically for transitions of } Q.\}
\end{array}$$

Trivially, $\sim_0 \supseteq \sim_1 \supseteq \sim_2 \supseteq \cdots$.

We now prove that, since the LTS is structurally image finite (see Proposition 3.7), it follows that

$$\sim \; = \bigcap_{n \geq 0} \sim_n \quad . \tag{1}$$

One inclusion is trivial: by induction on n, it can be proved that $\sim \subseteq \sim_n$ for every n, and so $\sim \subseteq \bigcap_{n\geq 0} \sim_n$. For the converse, fix $P \xrightarrow{\mu} P'$ and consider the case for $\mu = (\nu\widetilde{m})p$, since the case for $\mu = \tau$ can be proved like in π-calculus. For every $n \geq 0$, since $P \sim_{n+1} Q$, there exist q_n, Q_n and ρ_n such that $Q \xrightarrow{(\nu\widetilde{m})q_n} Q_n$, $p, \sigma \ll q_n, \rho_n$ and $\sigma P' \sim_n \rho_n Q_n$. However, by Proposition 3.18, there are finitely many (up-to α-equivalence) such q_n's; thus, there must exist (at least) one q_k that leads to infinitely many Q_n's that, because of Proposition 3.7, cannot be all different (up-to $\equiv$). Fix one of such q_k's; there must exist (at least) one Q_h such that $Q \xrightarrow{(\nu\widetilde{m})q_k} Q_h$ and there are infinitely many Q_n's such that $Q \xrightarrow{(\nu\widetilde{m})q_k} Q_n$ and $Q_n \equiv Q_h$. Fix one of such Q_h's. It suffices to prove that $\sigma P' \sim_n \rho_h Q_h$, for every n. This fact trivially holds whenever $n \leq h$: in this case, we have that $\sim_n \supseteq \sim_h$. So, let $n > h$. If $Q_n \equiv Q_h$, conclude, since $\equiv$ is closed under substitutions (notice that $\rho_n = \rho_h$ since $q_n = q_h = q_k$) and $\equiv \subseteq \sim_n$, for every n. Otherwise, there must exist $m > n$ such that $Q_m \equiv Q_h$ (otherwise there would not be infinitely many Q_n's structurally equivalent to Q_h): thus, $\sigma P' \sim_m \rho_h Q_h$ that implies $\sigma P' \sim_n \rho_h Q_h$, since $m > n$.

Thus, $!P \sim !Q$ if and only if $!P \sim_n !Q$, for all n. Let P^n denote the parallel composition of n copies of the process P (and similarly for Q). Now, it can be proved that

$$!P \sim_n P^{2n} \quad \text{and} \quad !Q \sim_n Q^{2n} \; . \tag{2}$$

The proof is by induction on n and the details are left to the interested reader. By repeatedly exploiting Lemma 3.23, it follows that $P^{2n} \sim Q^{2n}$ and so by (1)

$$P^{2n} \sim_n Q^{2n} \; . \tag{3}$$

Now by (3) it follows that $P \sim Q$ implies that $P^{2n} \sim_n Q^{2n}$, for all n. By (2) and Lemma 3.19 (that also holds with $\sim_n$ in place of $\sim$), it follows that $!P \sim_n !Q$, for all n. By (1), conclude that $!P \sim !Q$. $\square$

Lemma 3.25. *$\sim$ is contextual.*

Proof: Given two bisimilar processes P and Q, it is necessary to show that for any context $\mathcal{C}(\cdot)$ it holds that $\mathcal{C}(P) \sim \mathcal{C}(Q)$. The proof is by induction on the structure of the context.

- If $\mathcal{C}(\cdot) \stackrel{\text{def}}{=} \cdot$ then the result is immediate.
- If $\mathcal{C}(\cdot) \stackrel{\text{def}}{=} \mathcal{C}'(\cdot) \mid R$ or $\mathcal{C}(\cdot) \stackrel{\text{def}}{=} (\nu n)\mathcal{C}'(\cdot)$, then $\mathcal{C}'(P) \sim \mathcal{C}'(Q)$ by induction and conclude by Lemma 3.23.
- If $\mathcal{C}(\cdot) \stackrel{\text{def}}{=} !\mathcal{C}'(\cdot)$, then $\mathcal{C}'(P) \sim \mathcal{C}'(Q)$ by induction and conclude by Lemma 3.24.
- If $\mathcal{C}(\cdot) \stackrel{\text{def}}{=} p \to \mathcal{C}'(\cdot)$, then $\mathcal{C}'(P) \sim \mathcal{C}'(Q)$ by induction and conclude by Lemma 3.22.

$\square$

The soundness of bisimilarity w.r.t. barbed congruence now trivially follows.

Theorem 3.26 (Soundness of bisimilarity)**.** $\sim \subseteq \simeq$.

Proof: Lemma 3.20, Lemma 3.21 and Lemma 3.25 entail that $\sim$ satisfies the conditions of Definition 3.5. $\square$

3.6 Completeness of the Bisimulation

Completeness is proved by showing that barbed congruence is a bisimulation. There are two results required: showing that barbed congruence is closed under substitutions, and showing that, for any challenge, a proper reply can be yielded via closure under an appropriate context. To this aim, we define a notion of success and failure that can be reported. A fresh name w is used for reporting success, with a barb $\downarrow_w$ indicating success, and $\Downarrow_w$ indicating a reduction sequence that eventually reports success. Failure is handled similarly using the fresh name f. A process P *succeeds* if $P \Downarrow_w$ and $P \not\Downarrow_f$; P is *successful* if $P \equiv (\nu\widetilde{n})(\ulcorner w \urcorner \bullet p \mid P')$, for some $\widetilde{n}$, p and P' such that $w \notin \widetilde{n}$ and $P' \not\Downarrow_f$. P *becomes successful* if it can reduce to a successful process and $P \not\Downarrow_f$.

The next lemma shows that barbed congruence is closed under any substitution.

Lemma 3.27. *If $P \simeq Q$ then $\sigma P \simeq \sigma Q$, for every σ.*

Proof: Given a substitution σ, choose patterns p and q such that $\{p\|q\} = (\sigma, \{\})$; to be explicit, $p = \lambda x_1 \bullet \ldots \bullet \lambda x_k$ and $q = \sigma(x_1) \bullet \ldots \bullet \sigma(x_k)$, for $\{x_1, \ldots, x_k\} = \mathsf{dom}(\sigma)$. Define $\mathcal{C}(\cdot) \stackrel{\text{def}}{=} p \to \cdot \mid q$; by context closure, $\mathcal{C}(P) \simeq \mathcal{C}(Q)$. By reduction closure, the reduction $\mathcal{C}(P) \longmapsto \sigma P$ can be replied to only by $\mathcal{C}(Q) \longmapsto \sigma Q$; hence, $\sigma P \simeq \sigma Q$, as desired. $\square$

The other result to be proved is that challenges can be tested for a proper reply by a context. When the challenge is an internal action, the reply is also an internal action; thus, the empty context suffices, as barbed congruence is reduction closed. The complex scenario is when the challenge is a pattern together with a set of restricted names, i.e., a label of the form $(\nu\widetilde{n})p$. Observe that in the bisimulation such challenges also fix a substitution σ whose domain is the binding names of p. Most of this section develops a reply for a challenge of the form $((\nu\widetilde{n})p, \mathsf{id}_{\mathsf{bn}(p)})$; the general setting (with an arbitrary σ) will be recovered in Theorem 3.44 by relying on Lemma 3.13.

The context for forcing a proper reply is developed in three steps. The first step presents the *specification* of a pattern and a set of names N (to be thought of as the free names of the processes being compared for bisimilarity); this is the information required to build a reply context. The second step develops auxiliary processes to test specific components of a pattern, based on information from the specification. The third step combines these into a reply context that becomes successful if and only if it interacts with a process that exhibits a proper reply to the challenge.

For later convenience, we define the *first projection* $\mathsf{fst}(-)$ and *second projection* $\mathsf{snd}(-)$ of a set of pairs: e.g., $\mathsf{fst}(\{(x, m), (y, n)\}) = \{x, y\}$ and $\mathsf{snd}(\{(x, m), (y, n)\}) = \{m, n\}$, respectively.

Definition 3.28. *The* specification $\mathsf{spec}^N(p)$ *of a pattern p with respect to a finite set of names N is defined follows:*

$$\begin{array}{rcl}
\mathsf{spec}^N(\lambda x) & = & x, \{\}, \{\} \\
\mathsf{spec}^N(n) & = & \begin{cases} \lambda x, \{(x, n)\}, \{\} & \textit{if } n \in N \textit{ and } x \textit{ is fresh for } N \textit{ and } p \\ \lambda x, \{\}, \{(x, n)\} & \textit{if } n \notin N \textit{ and } x \textit{ is fresh for } N \textit{ and } p \end{cases} \\
\mathsf{spec}^N(\ulcorner n \urcorner) & = & \ulcorner n \urcorner, \{\}, \{\} \\
\mathsf{spec}^N(p \bullet q) & = & p' \bullet q', F_p \uplus F_q, R_p \uplus R_q \quad \textit{if } \begin{cases} \mathsf{spec}^N(p) = p', F_p, R_p \\ \mathsf{spec}^N(q) = q', F_q, R_q \end{cases}
\end{array}$$

where $F_p \uplus F_q$ denotes $F_p \cup F_q$, provided that $\mathsf{fst}(F_p) \cap \mathsf{fst}(F_q) = \emptyset$ (a similar meaning holds for $R_p \uplus R_q$).

Given a pattern p, the specification $\mathsf{spec}^N(p) = p', F, R$ of p with respect to a set of names N has three components: (1) p', called the *complementary pattern*, is a pattern used to ensure that the context interacts with a process that exhibits a pattern q such that p is compatible with q (via some substitutions); (2) F is a collection of pairs (x, n) made up by a binding name in p' and the expected (free) name it will be bound to; finally, (3) R is a collection of pairs (x, n) made up by a binding name in p' and the expected (restricted) name it will be bound to. Observe that p' is well formed as all binding names are (pairwise) different.

The specification is straightforward for binding names, protected names and compounds. When p is a variable name, p' is a fresh binding name λx and the intended binding of x to n is recorded in F or R, according to whether n is free or restricted, respectively.

Proposition 3.29. *Given a pattern p and a finite set of names N, let* $\mathsf{spec}^N(p) = p', F, R$. *Then,* $\{p \| p'\} = (\mathsf{id}_{\mathsf{bn}(p)}, \{n/x\}_{(x,n) \in F \cup R})$.

Proof: By straightforward induction on the structure of p. $\square$

To simplify the definitions, let $\prod_{x \in S} \mathcal{P}(x)$ be the parallel composition of processes $\mathcal{P}(x)$, for each x in S. The tests also exploit a check $\mathsf{check}(x, m, y, n, w)$

to ensure equality or inequality of name substitutions:

$$\mathsf{check}(x,m,y,n,w) = \begin{cases} (\nu z)(\ulcorner z\urcorner \bullet \ulcorner x\urcorner \mid \ulcorner z\urcorner \bullet \ulcorner y\urcorner \rightarrow \ulcorner w\urcorner) & \text{if } m = n \\ \ulcorner w\urcorner \mid (\nu z)(\ulcorner z\urcorner \bullet \ulcorner x\urcorner \mid \ulcorner z\urcorner \bullet \ulcorner y\urcorner \rightarrow \ulcorner f\urcorner \bullet \lambda z) & \text{otherwise} \end{cases}$$

Observe that failure here is indicated by pattern $\ulcorner f\urcorner \bullet \lambda z$; in this way, two failure barbs cannot unify and so they cannot disappear during computations.

Definition 3.30 (Tests)**.** *Let w and f be fresh names, i.e. different from all the other names around. Then define:*

$$\begin{aligned}
\mathsf{free}(x,n,w) &= (\nu m)(\ulcorner m\urcorner \bullet \ulcorner n\urcorner \rightarrow \ulcorner w\urcorner \mid \ulcorner m\urcorner \bullet \ulcorner x\urcorner) \\
\mathsf{rest}^N(x,w) &= \ulcorner w\urcorner \mid (\nu m)(\nu z)(\ \ulcorner m\urcorner \bullet x \bullet z \\
&\qquad\qquad \mid \ulcorner m\urcorner \bullet (\lambda y_1 \bullet \lambda y_2) \bullet \lambda z \rightarrow \ulcorner f\urcorner \bullet \lambda z \\
&\qquad\qquad \mid \textstyle\prod_{n\in N} \ulcorner m\urcorner \bullet \ulcorner n\urcorner \bullet \lambda z \rightarrow \ulcorner f\urcorner \bullet \lambda z\ ) \\
\mathsf{equality}^R(x,m,w) &= (\nu \widetilde{w_y})(\ \ulcorner w_{y_1}\urcorner \rightarrow \ldots \rightarrow \ulcorner w_{y_i}\urcorner \rightarrow \ulcorner w\urcorner \\
&\qquad\qquad \mid \textstyle\prod_{(y,n)\in R} \mathsf{check}(x,m,y,n,w_y)\ ) \\
&\quad \textit{where } \widetilde{y} = \{y_1,\ldots,y_i\} = \mathsf{fst}(R)
\end{aligned}$$

The behaviour of the tests just defined is formalized by the following three results.

Lemma 3.31. *Let θ be such that $\{n,w\} \cap \mathsf{dom}(\theta) = \emptyset$; then, $\theta(\mathsf{free}(x,n,w))$ succeeds if and only if $\theta(x) = n$.*

Proof: Trivial. □

Lemma 3.32. *Let θ be such that $(N \cup \{w,f\}) \cap \mathsf{dom}(\theta) = \emptyset$; then, $\theta(\mathsf{rest}^N(x,w))$ succeeds if and only if $\theta(x) \in \mathcal{N} \setminus N$.*

Proof: Straightforward. □

Lemma 3.33. *Let θ be such that $(\mathsf{snd}(R) \cup \{w,f,m\}) \cap \mathsf{dom}(\theta) = \emptyset$; then, $\theta(\mathsf{equality}^R(x,m,w))$ succeeds if and only if, for every $(y,n) \in R$, $m = n$ if and only if $\theta(x) = \theta(y)$.*

Proof: In order for $\theta(\mathsf{equality}^R(x,m,w))$ to succeed by exhibiting a barb $\ulcorner w\urcorner$, each check $\theta(\mathsf{check}(x,m,y,n,w_y))$ must succeed by producing $\ulcorner w_y\urcorner$. The rest of the proof is straightforward. □

Lemma 3.34. *Let T be a test and θ be a substitution such that $\theta(T)$ succeeds; there are exactly k reductions of $\theta(T)$ to a successful process, where k depends only on the structure of T.*

Proof: Trivial for free and restricted tests, for which $k = 1$ and $k = 0$, respectively. For an equality test $\mathsf{equality}^R(x,m,w)$ it suffices to observe that each successful check has an exact number of reductions to succeed (1, if $m = n$, 0 otherwise) and then there is a reduction to consume the success barb of each check. Thus, $k = |R| + h$, where h is the number of pairs in R whose second component equals m. □

From now on, we adopt the following notation: if $\widetilde{n} = n_1, \ldots, n_i$, then $\ulcorner w \urcorner \bullet \widetilde{n}$ denotes $\ulcorner w \urcorner \bullet n_1 \bullet \ldots \bullet n_i$. Moreover, $\theta(\widetilde{n})$ denotes $\theta(n_1), \ldots, \theta(n_i)$; hence, $\ulcorner w \urcorner \bullet \theta(\widetilde{n})$ denotes $\ulcorner w \urcorner \bullet \theta(n_1) \bullet \ldots \bullet \theta(n_i)$.

Definition 3.35. *The* characteristic process $\mathsf{char}^N(p)$ *of a pattern p with respect to a finite set of names N is* $\mathsf{char}^N(p) = p' \to \mathsf{tests}^N_{F,R}$ *where* $\mathsf{spec}^N(p) = p', F, R$ *and*

$$\begin{array}{rcl}\mathsf{tests}^N_{F,R} & \stackrel{\text{def}}{=} & (\nu\widetilde{w_x})(\nu\widetilde{w_y})(\\ & & \quad \ulcorner w_{x_1} \urcorner \to \ldots \to \ulcorner w_{x_i} \urcorner \to \ulcorner w_{y_1} \urcorner \to \ldots \to \ulcorner w_{y_j} \urcorner \to \ulcorner w \urcorner \bullet \widetilde{x} \\ & & \quad \mid \prod_{(x,n)\in R} \mathsf{equality}^R(x, n, w_x) \\ & & \quad \mid \prod_{(y,n)\in F} \mathsf{free}(y, n, w_y) \\ & & \quad \mid \prod_{(y,n)\in R} \mathsf{rest}^N(y, w_y)\ )\end{array}$$

where $\widetilde{x} = \{x_1, \ldots, x_i\} = \mathsf{fst}(R)$ *and* $\widetilde{y} = \{y_1, \ldots, y_j\} = \mathsf{fst}(F) \cup \mathsf{fst}(R)$.

Lemma 3.36. *Let θ be such that* $\mathsf{dom}(\theta) = \mathsf{fst}(F) \cup \mathsf{fst}(R)$*; then,* $\theta(\mathsf{tests}^N_{F,R})$ *succeeds if and only if*

1. *for every $(x, n) \in F$ it holds that $\theta(x) = n$;*
2. *for every $(x, n) \in R$ it holds that $\theta(x) \in \mathcal{N} \setminus N$;*
3. *for every (x, n) and $(y, m) \in R$ it holds that $n = m$ if and only if $\theta(x) = \theta(y)$.*

Proof: By induction on $|F \cup R|$ and Lemmas 3.31, 3.32 and 3.33. Indeed, by Definition 3.28, $\mathsf{fst}(F \cup R) \cap (\mathsf{snd}(F \cup R) \cup N) = \emptyset$; moreover, freshness of w and f implies that $\{w, f\} \cap \mathsf{dom}(\theta) = \emptyset$. □

Note that the following results consider the number of reductions required to succeed. These are significant to proving the results in the strong setting, but unimportant in the weak setting, i.e. with $\longmapsto$ replaced by $\Longmapsto$.

Lemma 3.37. *Given* $\mathsf{char}^N(p)$ *and any substitution θ such that* $\mathsf{dom}(\theta) = \mathsf{fst}(F) \cup \mathsf{fst}(R)$ *and* $\theta(\mathsf{tests}^N_{F,R})$ *succeeds, then there are exactly k reduction steps* $\theta(\mathsf{tests}^N_{F,R}) \longmapsto^k \ulcorner w \urcorner \bullet \theta(\widetilde{x}) \mid Z$*, where* $\widetilde{x} = \mathsf{fst}(R)$*,* $Z \simeq \mathbf{0}$ *and k depends only on F, R and N; moreover, no sequence of reductions shorter than k can yield a successful process.*

Proof: By induction on $|F \cup R|$ and Lemma 3.34. □

Notice that k does not depend on θ; thus, we shall refer to k as the number of reductions for $\mathsf{tests}^N_{F,R}$ to become successful. The crucial result we have to show is that the characterisation of a pattern p with respect to a set of names N can yield a reduction via a proper reply (according to Definition 3.11) to the challenge $(\nu\widetilde{n})p$ when $\widetilde{n}$ does not intersect N. A reply context for a challenge $((\nu\widetilde{n})p, \mathsf{id}_{\mathsf{bn}(p)})$ with a finite set of names N can be defined by exploiting the characteristic process.

Definition 3.38. *A* reply context $\mathcal{C}^N_p(\cdot)$ *for the challenge* $((\nu\widetilde{n})p, \mathsf{id}_{\mathsf{bn}(p)})$ *with a finite set of names N such that $\widetilde{n}$ is disjoint from N is defined as follows:*

$$\mathcal{C}^N_p(\cdot) \quad \stackrel{\text{def}}{=} \quad \mathsf{char}^N(p) \mid \cdot$$

Proposition 3.39. *Given a reply context $\mathcal{C}_p^N(\cdot)$, the minimum number of reductions required for $\mathcal{C}_p^N(Q)$ to become successful (for any Q) is the number of reduction steps for $\mathsf{tests}_{F,R}^N$ to become successful plus 1.*

Proof: By Definition 3.38, success can be generated only after removing the case p' from $\mathsf{char}^N(p)$; this can only be done via a reduction together with Q, i.e. Q must eventually yield a pattern q that matches with p'. The minimum number of reductions is obtained when Q already yields such a q, i.e. when Q is a process of the form $(\nu\widetilde{m})(q \to Q_1 \mid Q_2)$, for some $\widetilde{m}$, q, Q_1 and Q_2 such that $\{p'\|q\} = (\theta, \rho)$ and $\theta(\mathsf{tests}_{F,R}^N)$ succeeds. In this case, $\mathsf{dom}(\theta) = \mathsf{bn}(p') = \mathsf{fst}(F \cup R)$; by Lemma 3.37, $\theta(\mathsf{tests}_{F,R}^N)$ becomes successful in k reductions; thus, $\mathcal{C}_p^N(Q)$ becomes successful in $k+1$ reductions, and this is the minimum over all possible Q's. $\square$

Denote the number of reductions put forward by Proposition 3.39 as $\textsc{Lb}(N, p)$. The main feature of $\mathcal{C}_p^N(\cdot)$ is that, when the hole is filled with a process Q, it holds that $\mathcal{C}_p^N(Q)$ becomes successful in $\textsc{Lb}(N, p)$ reductions if and only if there exist q, Q' and ρ such that $Q \xrightarrow{(\nu\widetilde{n})q} Q'$ and $p, \mathsf{id}_{\mathsf{bn}(p)} \ll q, \rho$. This fact is proved by Theorems 3.40 and 3.42.

Theorem 3.40. *Suppose given a challenge $((\nu\widetilde{n})p, \mathsf{id}_{\mathsf{bn}(p)})$, a finite set of names N, a process Q and fresh names w and f such that $(\widetilde{n} \cup \{w, f\}) \cap N = \emptyset$ and $(\mathsf{fn}((\nu\widetilde{n})p) \cup \mathsf{fn}(Q)) \subseteq N$. If Q has a transition of the form $Q \xrightarrow{(\nu\widetilde{n})q} Q'$ and there is a substitution ρ such that $p, \mathsf{id}_{\mathsf{bn}(p)} \ll q, \rho$ then $\mathcal{C}_p^N(Q)$ succeeds and has a reduction sequence $\mathcal{C}_p^N(Q) \longmapsto^k (\nu\widetilde{n})(\rho Q' \mid \ulcorner w \urcorner \bullet \widetilde{n} \mid Z)$, where $k = \textsc{Lb}(N, p)$ and $Z \simeq \mathbf{0}$.*

Proof: We assume, by α-conversion, that binding names of p are fresh, in particular do not appear in Q. By Proposition 3.29 $\{p\|p'\} = (\sigma, \theta)$ where $\sigma = \mathsf{id}_{\mathsf{bn}(p)}$ and $\theta = \{n/x\}_{(x,n) \in F \cup R}$. By Lemma 3.16 $\{q\|p'\} = (\rho, \theta)$; thus $\mathcal{C}_p^N(Q) \longmapsto (\nu\widetilde{n})(\rho Q' \mid \theta(\mathsf{tests}_{F,R}^N))$. Since w and f do not appear in Q, the only possibility of producing a successful process is when $\theta(\mathsf{tests}_{F,R}^N)$ succeeds; this is ensured by Lemma 3.36. The thesis follows by Lemma 3.37. $\square$

The main difficulty in proving the converse result is the possibility of renaming restricted names. Thus, we first need a technical Lemma that ensures us the possibility of having the same set of restricted names both in the challenge and in the reply, as required by the definition of bisimulation.

Lemma 3.41. *Let p and N be such that $\mathsf{pn}(p) \subseteq N$, $\mathsf{bn}(p) \cap N = \emptyset$ and $\mathsf{spec}^N(p) = p', F, R$. If q is such that $\mathsf{bn}(p) \cap \mathsf{fn}(q) = \emptyset$ and $\{p'\|q\} = (\theta, \rho)$ such that $\theta(\mathsf{tests}_{F,R}^N)$ succeeds, then:*

- $|\mathsf{vn}(p) \setminus N| = |\mathsf{vn}(q) \setminus N|$;
- *there exists a bijective renaming ζ of $\mathsf{vn}(q) \setminus N$ into $\mathsf{vn}(p) \setminus N$ such that $p, \mathsf{id}_{\mathsf{bn}(p)} \ll \zeta q, \rho$;*
- $\theta = \{n/x\}_{(x,n) \in F} \ \cup \ \{\zeta^{-1}(n)/x\}_{(x,n) \in R}$.

Proof: The proof is by induction on the structure of p. We have three possible base cases:

1. If $p = \lambda x$, then $p' = x$ and $F = R = \emptyset$. By definition of pattern matching, $q \in \{x, \ulcorner x \urcorner, \lambda y\}$, for any y. Since $x \in \mathsf{bn}(p)$ and $\mathsf{bn}(p) \cap \mathsf{fn}(q) = \emptyset$, it can only be $q = \lambda y$; then, $\theta = \{\}$ and $\rho = \{x/y\}$. This suffices to conclude, since $\mathsf{vn}(p) \setminus N = \mathsf{vn}(q) \setminus N = \emptyset$ and $\lambda x, \mathsf{id}_{\{x\}} \ll \lambda y, \rho$.

2. If $p = n$, then $p' = \lambda x$, for x fresh. Let us distinguish two subcases:

   - If $n \in N$, then $F = \{(x, n)\}$ and $R = \emptyset$. By definition of pattern matching, q must be communicable, $\rho = \{\}$ and $\theta = \{q/x\}$. Since $\mathsf{tests}^N_{F,R}$ only contains $\mathsf{free}(x, n)$, by Lemma 3.31 it holds that $q = \theta(x) = n$. This suffices to conclude, since $\mathsf{vn}(p) \setminus N = \mathsf{vn}(q) \setminus N = \emptyset$ and $n, \{\} \ll n, \rho$.
   - If $n \notin N$, then $F = \emptyset$ and $R = \{(x, n)\}$. Like before, q must be communicable, $\rho = \{\}$ and $\theta = \{q/x\}$. Since $\mathsf{tests}^N_{F,R}$ contains $\mathsf{rest}^N(x)$, by Lemma 3.32 it holds that $q = \theta(x) = m \in \mathcal{N} \setminus N$; thus, $|\mathsf{vn}(p) \setminus N| = |\mathsf{vn}(q) \setminus N| = 1$. This suffices to conclude, by taking $\zeta = \{n/m\}$, since $n, \{\} \ll n, \rho$.

3. If $p = \ulcorner n \urcorner$, then $p' = \ulcorner n \urcorner$ and $F = R = \emptyset$. By definition of pattern matching, $q \in \{n, \ulcorner n \urcorner\}$ and $\rho = \theta = \{\}$. In any case, $\mathsf{vn}(p) \setminus N = \mathsf{vn}(q) \setminus N = \emptyset$ and $p, \{\} \ll q, \rho$.

For the inductive case, let $p = p_1 \bullet p_2$. By definition of specification, $p' = p'_1 \bullet p'_2$, $F = F_1 \uplus F_2$ and $R = R_1 \uplus R_2$, where $\mathsf{spec}^N(p_i) = p'_i, F_i, R_i$, for $i \in \{1, 2\}$. By definition of pattern matching, there are two possibilities for q:

1. If $q = \lambda z$, for some z, then p' must be communicable (i.e., $p' = x_1 \bullet \ldots \bullet x_n$, for some n), $\theta = \{\}$ and $\rho = \{x_1 \bullet \ldots \bullet x_n / z\}$. If $p' = x_1 \bullet \ldots \bullet x_n$, then, by definition of specification, $p = \lambda x_1 \bullet \ldots \bullet \lambda x_n$. Hence, $\mathsf{vn}(p) \setminus N = \mathsf{vn}(q) \setminus N = \emptyset$ and $\lambda x_1 \bullet \ldots \bullet \lambda x_n, \mathsf{id}_{\{x_1,\ldots,x_n\}} \ll \lambda z, \rho$.

2. Otherwise, it must be that $q = q_1 \bullet q_2$, with $\{p'_i \| q_i\} = (\theta_i, \rho_i)$, for $i \in \{1, 2\}$; moreover, $\theta = \theta_1 \cup \theta_2$ and $\rho = \rho_1 \cup \rho_2$. Since the first components of F_1 and F_2 are disjoint (and similarly for R_1 and R_2), $\theta(\mathsf{tests}^N_{F,R})$ succeeds implies that both $\theta(\mathsf{tests}^N_{F_1,R_1})$ and $\theta(\mathsf{tests}^N_{F_2,R_2})$ succeed, since every test of $\theta(\mathsf{tests}^N_{F_i,R_i})$ is a test of $\theta(\mathsf{tests}^N_{F,R})$. Now, by two applications of the induction hypothesis, we obtain that, for $i \in \{1, 2\}$:

   - $|V_i| = |W_i|$, where $V_i = \mathsf{vn}(p_i) \setminus N$ and $W_i = \mathsf{vn}(q_i) \setminus N$;
   - there exists a bijective renaming ζ_i of W_i into V_i such that $p_i, \mathsf{id}_{\mathsf{bn}(p_i)} \ll \zeta_i q_i, \rho_i$;
   - $\theta_i = \{n/x\}_{(x,n) \in F_i} \ \cup \ \{\zeta_i^{-1}(n)/x\}_{(x,n) \in R_i}$.

   We now prove the following facts:

   (a) *if $m \in W_1 \setminus W_2$, then $\zeta_1(m) \in V_1 \setminus V_2$:*
   by contradiction, assume that $\zeta_1(m) = n \in V_1 \cap V_2$ (indeed, $\zeta_1(m) \in V_1$, by construction of ζ_1). By construction of the specification, there exists $(x, n) \in R_1$. Moreover, since $n \in V_2$, there exists $m' \in W_2$ such that $\zeta_2(m') = n$ but $m' \neq m$. Again by construction of the specification, there exists $(y, n) \in R_2$. By inductive

hypothesis, $\theta_1(x) = \zeta_1^{-1}(n) = m$ and $\theta_2(x) = \zeta_2^{-1}(n) = m'$. But then $\theta(\mathsf{check}(x, n, y, n))$, that is part of $\theta(\mathsf{tests}^N_{F,R})$, cannot succeed, since $\theta_1(x) \neq \theta_2(y)$ (see Lemma 3.33). Contradiction.

(b) *if* $m \in W_2 \setminus W_1$*, then* $\zeta_2(m) \in V_2 \setminus V_1$*:*
similar to the previous case.

(c) *if* $m \in W_1 \cap W_2$*, then* $\zeta_1(m) = \zeta_2(m) \in V_1 \cap V_2$*:*
let $n_i = \zeta_i(m) \in V_i$; by construction of the specification, there exists $(x_i, n_i) \in R_i$. By contradiction, assume that $n_1 \neq n_2$. Then, $\theta(\mathsf{check}(x_1, n_1, x_2, n_2))$, that is part of $\theta(\mathsf{tests}^N_{F,R})$, reports failure, since by induction $\theta_1(x_1) = \zeta_1^{-1}(x_1) = m = \zeta_2^{-1}(x_2) = \theta_2(x_2)$ (see Lemma 3.33). Contradiction.

Thus, $V_1 \cup V_2$ and $W_1 \cup W_2$ have the same cardinality; moreover, $\zeta = \zeta_1 \cup \zeta_2$ is a bijection between them and it is well-defined (in the sense that ζ_1 and ζ_2 coincide on all elements of $\mathsf{dom}(\zeta_1) \cap \mathsf{dom}(\zeta_2)$ – see point (c) above). Thus, $p_1, \mathsf{id}_{\mathsf{bn}(p_1)} \ll \zeta q_1, \rho_1$ and $p_2, \mathsf{id}_{\mathsf{bn}(p_2)} \ll \zeta q_2, \rho_2$; so, $p, \mathsf{id}_{\mathsf{bn}(p)} \ll \zeta q, \rho$. Moreover, $\theta = \theta_1 \cup \theta_2 = \{n/x\}_{(x,n)\in F_1} \cup \{\zeta^{-1}(n)/x\}_{(x,n)\in R_1} \cup \{n/x\}_{(x,n)\in F_2} \cup \{\zeta^{-1}(n)/x\}_{(x,n)\in R_2} = \{n/x\}_{(x,n)\in F} \cup \{\zeta^{-1}(n)/x\}_{(x,n)\in R}$, as desired. □

Theorem 3.42. *Suppose given a challenge* $((\nu\widetilde{n})p, \mathsf{id}_{\mathsf{bn}(p)})$*, a finite set of names* N*, a process* Q *and fresh names* w *and* f *such that* $\mathsf{bn}(p) \cap N = (\widetilde{n} \cup \{w, f\}) \cap N = \emptyset$ *and* $(\mathsf{fn}((\nu\widetilde{n})p) \cup \mathsf{fn}(Q)) \subseteq N$*. If* $\mathcal{C}^N_p(Q)$ *becomes successful in* $\textsc{Lb}(N, p)$ *reduction steps, then there exist* q*,* Q' *and* ρ *such that* $Q \xrightarrow{(\nu\widetilde{n})q} Q'$ *and* $p, \mathsf{id}_{\mathsf{bn}(p)} \ll q, \rho$*.*

Proof: By Proposition 3.39, there must be a reduction $\mathcal{C}^N_p(Q) \longmapsto (\nu\widetilde{m})(\theta(\mathsf{tests}^N_{F,R}) \mid \rho Q'')$ obtained because $Q \xrightarrow{(\nu\widetilde{m})q'} Q''$ and $\{p' \| q'\} = (\theta, \rho)$. Since $w \notin \mathsf{fn}(Q, p')$ and $\mathcal{C}^N_p(Q) \Downarrow_w$, it must be that $\theta(\mathsf{tests}^N_{F,R})$ becomes successful; by Proposition 3.39, this happens in $\textsc{Lb}(N, p) - 1$ reduction steps.

By hypothesis, $\mathsf{fn}((\nu\widetilde{n})p) \subseteq N$; thus, $\mathsf{vn}(p) \setminus N = \widetilde{n}$. Moreover, by α-conversion, $\widetilde{m} \cap \mathsf{fn}(Q) = \emptyset$; thus, by $\mathsf{fn}((\nu\widetilde{m})q') \subseteq \mathsf{fn}(Q) \subseteq N$, we have that $\mathsf{vn}(q') \setminus N = \widetilde{m}$. Since $\mathsf{bn}(p) \cap N = \emptyset$, we also have that $\mathsf{bn}(p) \cap \mathsf{fn}(q') = \emptyset$; thus, we can use Lemma 3.41 and obtain a bijection $\zeta = \{\widetilde{n}/\widetilde{m}\}$ such that $p, \mathsf{id}_{\mathsf{bn}(p)} \ll \zeta q', \rho$; moreover, by α-conversion, $Q \xrightarrow{(\nu\widetilde{n})\zeta q'} \zeta Q''$. We can conclude by taking $q = \zeta q'$ and $Q' = \zeta Q''$. □

We are almost ready to give the completeness result, we just need an auxiliary Lemma that allows us to remove success and dead processes from both sides of a barbed congruence, while also opening the scope of the names exported by the success barb.

Lemma 3.43. *Let* $(\nu\widetilde{m})(P \mid \ulcorner w \urcorner \bullet \widetilde{m} \mid Z) \simeq (\nu\widetilde{m})(Q \mid \ulcorner w \urcorner \bullet \widetilde{m} \mid Z)$*, for* $w \notin \mathsf{fn}(P, Q, \widetilde{m})$ *and* $Z \simeq \mathbf{0}$*; then* $P \simeq Q$*.*

Proof: By Theorem 3.26, it suffices to prove that

$$\Re = \{(P, Q) \ : \ (\nu\widetilde{m})(P \mid \ulcorner w \urcorner \bullet \widetilde{m} \mid Z) \simeq (\nu\widetilde{m})(Q \mid \ulcorner w \urcorner \bullet \widetilde{m} \mid Z) \wedge w \notin \mathsf{fn}(P, Q, \widetilde{m}) \ \wedge \ Z \simeq \mathbf{0}\}$$

is a bisimulation. Consider the challenge $P \xrightarrow{\mu} P'$ and reason by case analysis on μ.

- If $\mu = \tau$, then $(\nu\widetilde{m})(P \mid \ulcorner w \urcorner \bullet \widetilde{m} \mid Z) \xrightarrow{\tau} (\nu\widetilde{m})(P' \mid \ulcorner w \urcorner \bullet \widetilde{m} \mid Z) = \hat{P}$. By Proposition 3.9 and reduction closure, $(\nu\widetilde{m})(Q \mid \ulcorner w \urcorner \bullet \widetilde{m} \mid Z) \xrightarrow{\tau} \hat{Q}$ such that $\hat{P} \simeq \hat{Q}$. By Proposition 2.5 (since $w \notin \mathsf{fn}(Q)$) and $Z \simeq \mathbf{0}$, it can only be that $\hat{Q} = (\nu\widetilde{m})(Q' \mid \ulcorner w \urcorner \bullet \widetilde{m} \mid Z)$, where $Q \xrightarrow{\tau} Q'$. By definition of $\Re$, we conclude that $(P', Q') \in \Re$.

- If $\mu = (\nu\widetilde{n})p$, for $(\mathsf{bn}(p) \cup \widetilde{n}) \cap \mathsf{fn}(Q) = \emptyset$. By alpha-conversion, we can also assume that $\mathsf{bn}(p) \cap (\widetilde{n} \cup \widetilde{m} \cup \mathsf{fn}(P)) = \emptyset$. Let us now fix a substitution σ such that $\mathsf{dom}(\sigma) = \mathsf{bn}(p)$ and $\mathsf{fn}(\sigma) \cap \widetilde{n} = \emptyset$. Consider the context

  $$\mathcal{C}(\cdot) = \ \cdot \ \mid \ \ulcorner w \urcorner \bullet \widetilde{\lambda m} \to (\sigma(\mathsf{char}^N(p)) \mid \ulcorner w \urcorner \bullet \widetilde{\lambda n} \to \ulcorner w' \urcorner \bullet \widetilde{n} \bullet \widetilde{m})$$

  for w' fresh (in particular, different from w). Consider now the following sequence of reductions:

  $$\begin{array}{ll} \mathcal{C}((\nu\widetilde{m})(P|\ulcorner w \urcorner \bullet \widetilde{m}|Z)) & \\ \longmapsto & (\nu\widetilde{m})(\sigma(\mathcal{C}_p^N(P)) \mid Z \mid \ulcorner w \urcorner \bullet \widetilde{\lambda n} \to \ulcorner w' \urcorner \bullet \widetilde{n} \bullet \widetilde{m}) \\ \longmapsto^{\mathrm{LB}(N,p)} & (\nu\widetilde{m})((\nu\widetilde{n})(\sigma P'|\ulcorner w \urcorner \bullet \widetilde{n}|\sigma Z') \mid Z \mid \ulcorner w \urcorner \bullet \widetilde{\lambda n} \to \ulcorner w' \urcorner \bullet \widetilde{n} \bullet \widetilde{m}) \\ \longmapsto & (\nu\widetilde{n}, \widetilde{m})(\sigma P' \mid \ulcorner w' \urcorner \bullet \widetilde{n} \bullet \widetilde{m} \mid Z|\sigma Z') \quad = \quad \hat{P} \end{array}$$

  The first reduction is obtained by matching $\ulcorner w \urcorner \bullet \widetilde{m}$ with the first case of $\mathcal{C}(\cdot)$; this replaces the binding names $\widetilde{m}$ in the context with the variable names $\widetilde{m}$ and the scope of the restriction is extended consequently. Moreover, $\sigma(\mathsf{char}^N(p)) \mid P = \sigma(\mathsf{char}^N(p) \mid P) = \sigma(\mathcal{C}_p^N(P))$: the first equality holds because $\mathsf{dom}(\sigma) = \mathsf{bn}(p)$ and $\mathsf{bn}(p) \cap \mathsf{fn}(P) = \emptyset$; the second equality holds by definition of reply context. The second sequence of reductions follows by Theorem 3.40 (ensuring that $\mathcal{C}_p^N(P) \longmapsto^{\mathrm{LB}(N,p)} (\nu\widetilde{n})(P'|\ulcorner w \urcorner \bullet \widetilde{n}|Z')$, for $Z' \simeq \mathbf{0}$), Proposition 2.4 and by the fact that $\sigma((\nu\widetilde{n})(P'|\ulcorner w \urcorner \bullet \widetilde{n}|Z')) = (\nu\widetilde{n})(\sigma P'|\ulcorner w \urcorner \bullet \widetilde{n}|\sigma Z')$ (indeed, w is fresh and $\mathsf{names}(\sigma) \cap \widetilde{n} = \emptyset$). Moreover, notice that $\sigma Z' \simeq \sigma\mathbf{0} \simeq \mathbf{0}$, because of Lemma 3.27. The last reduction is obtained by matching $\ulcorner w \urcorner \bullet \widetilde{n}$ with the case $\ulcorner w \urcorner \bullet \widetilde{\lambda n}$ of the context; this replaces the binding names $\widetilde{n}$ in the context with the variable names $\widetilde{n}$ and the scope of the restriction is extended consequently.

  Consider now $\mathcal{C}((\nu\widetilde{m})(Q|\ulcorner w \urcorner \bullet \widetilde{m}|Z))$; by reduction closure, $\mathcal{C}((\nu\widetilde{m})(Q|\ulcorner w \urcorner \bullet \widetilde{m}|Z)) \longmapsto^{\mathrm{LB}(N,p)+2} \hat{Q}$ such that $\hat{P} \simeq \hat{Q}$. Since $\hat{P}$ has a barb containing w', also $\hat{Q}$ must; by definition of $\mathcal{C}(\cdot)$, this can happen only if $\mathcal{C}_p^N(Q)$ becomes successful in $\mathrm{LB}(N,p)$ steps. By Theorem 3.42, this entails that there exist q, Q' and ρ such that $Q \xrightarrow{(\nu\widetilde{n})q} Q'$ and $p, \mathsf{id}_{\mathsf{bn}(p)} \ll q, \rho$. Moreover, with a reasoning similar to that for the reductions of $\mathcal{C}((\nu\widetilde{m})(P|\ulcorner w \urcorner \bullet \widetilde{m}|Z))$, we can conclude that $\hat{Q} = (\nu\widetilde{n}, \widetilde{m})(\sigma[\rho](Q') \mid \ulcorner w' \urcorner \bullet \widetilde{n} \bullet \widetilde{m} \mid Z|\sigma Z')$; indeed, in this case the application of Theorem 3.40 yields $\mathcal{C}_p^N(Q) \longmapsto^{\mathrm{LB}(N,p)} (\nu\widetilde{n})(\rho Q'|\ulcorner w \urcorner \bullet \widetilde{n}|Z')$.

  To state that $(\sigma P', \sigma[\rho](Q')) \in \Re$, it suffices to notice that $Z|\sigma Z' \simeq \mathbf{0}$; this holds because of contextuality of barbed congruence. Finally, Lemma 3.13

entails that $p, \sigma \ll q, \sigma[\rho]$: indeed, $\sigma[\mathsf{id}_{\mathsf{bn}(p)}] = \sigma$ because $\mathsf{dom}(\sigma) = \mathsf{bn}(p)$. This shows that q, Q' and $\sigma[\rho]$ is a proper reply to the challenge $P \xrightarrow{(\nu\widetilde{n})p} P'$ together with σ. $\square$

Theorem 3.44 (Completeness of the bisimulation)**.** $\simeq \subseteq \sim$.

Proof: It is sufficient to prove that, for every pair of processes P and Q such that $P \simeq Q$ and for every transition $P \xrightarrow{\mu} P'$, there exists a proper reply (according to the definition of the bisimulation) of Q and the reducts are still barbed congruent. This is trivial when $\mu = \tau$, due to reduction closure and Proposition 3.9. The difficult case if when $\mu = (\nu\widetilde{n})p$, for $(\mathsf{bn}(p) \cup \widetilde{n}) \cap \mathsf{fn}(Q) = \emptyset$. In this case fix a substitution σ such that $\mathsf{dom}(\sigma) = \mathsf{bn}(p)$ and $\mathsf{fn}(\sigma) \cap \widetilde{n} = \emptyset$.

By Theorem 3.40 and Proposition 3.14, $\mathcal{C}_p^N(P)$ becomes successful in k reduction steps, where $k = \mathrm{L}\text{B}(N, p)$. It follows by barbed congruence that $\mathcal{C}_p^N(Q)$ becomes successful in k reduction steps too; Theorem 3.42 then implies that $Q \xrightarrow{(\nu\widetilde{n})q} Q'$ for some q, Q' and ρ' such that $p, \mathsf{id}_{\mathsf{bn}(p)} \ll q, \rho'$.

By two applications of Theorem 3.40 it follows that $\mathcal{C}_p^N(P) \longmapsto^k (\nu\widetilde{n})(P' \mid \ulcorner w \urcorner \bullet \widetilde{n} \mid Z)$, for $Z \simeq \mathbf{0}$, and $\mathcal{C}_p^N(Q) \longmapsto^k (\nu\widetilde{n})(\rho' Q' \mid \ulcorner w \urcorner \bullet \widetilde{n} \mid Z)$. Notice that, by Lemma 3.39 and definition of the reply context, these are the only possibilities that yield a success barb in k reductions. Furthermore, reduction closure of $\simeq$ and Lemma 3.43 imply that $P' \simeq \rho' Q'$. By Lemma 3.27, we obtain $\sigma P' \simeq \sigma(\rho' Q') = \sigma[\rho'](Q')$. By Lemma 3.13, $p, \mathsf{id}_{\mathsf{bn}(p)} \ll q, \rho'$ implies $p, \sigma \ll q, \sigma[\rho']$. This suffices to conclude. $\square$

3.7 Equational Reasoning

This section considers some examples where bisimulation can be used to show equivalence of processes. The first example exploits the unification of protected names with both variable and protected names:

$$\ulcorner n \urcorner \to P \mid !n \to P \quad \sim \quad !n \to P\ .$$

It states that the processes $\ulcorner n \urcorner \to P \mid !n \to P$ can be subsumed by the more compact process $!n \to P$; indeed, any interaction of the left hand processes can be properly responded to by the right hand process and vice versa.

The second example considers the contractive nature of binding names in CPC: a case with the pattern $\lambda x \bullet \lambda y$ can be subsumed by a case with the pattern λz as long as some conditions are met. For example:

$$\lambda x \bullet \lambda y \to P \mid !\lambda z \to Q \quad \sim \quad !\lambda z \to Q \qquad \text{if } P \sim \{x \bullet y / z\} Q\ .$$

The side condition requires that the bodies of the cases must be bisimilar under a substitution that preserves the structure of any pattern bound by $\lambda x \bullet \lambda y$ in the process Q.

These examples both arise from pattern-unification and also appear in the compatibility relation. Indeed, the examples above are instances of a general result:

Theorem 3.45. *Let $P = p \to P' \mid !q \to Q'$ and $Q = !q \to Q'$. If there exists ρ such that $p, \mathsf{id}_{\mathsf{bn}(p)} \ll q, \rho$ and $P' \sim \rho Q'$, then $P \sim Q$.*

Proof: It suffices to prove that

$$\Re = \{(p \to P'|Q|R, Q|R) \,:\, Q = !q \to Q' \wedge \exists \rho\,.\, p, \mathsf{id}_{\mathsf{bn}(p)} \ll q, \rho \wedge P' \sim \rho Q'\} \cup \sim$$

is a bisimulation. To this aim, consider every challenge from $p \to P'|Q|R$ and show that there exists a transition from $Q|R$ that is a proper reply (according to the bisimulation). The converse (when the challenge comes from $Q \mid R$) is easier.

So, let $p \to P'|Q|R \xrightarrow{\mu} \hat{P}$; there are two possibilities for μ:

1. $\mu = (\nu\widetilde{n})p'$: in this case, we also have to fix a substitution σ such that $\mathsf{dom}(\sigma) = \mathsf{bn}(p')$ and $\mathsf{fn}(\sigma) \cap \widetilde{n} = \emptyset$. There are three possible ways for producing μ:

   (a) $\mu = p$ and $\hat{P} = P'|Q|R$: in this case, since the action comes from $p \to P'$, by the side condition of rule parext, it must be that $\mathsf{bn}(p) \cap \mathsf{fn}(Q|R) = \emptyset$. Now, consider $Q \xrightarrow{q} Q'|Q$ with $\mathsf{bn}(q) \cap \mathsf{fn}(Q|R) = \emptyset$ (this can always be done, by using α-conversion); thus, $Q|R \xrightarrow{q} Q'|Q|R = \hat{Q}$. Let ρ be such that $p, \mathsf{id}_{\mathsf{bn}(p)} \ll q, \rho$; by Lemma 3.13, $p, \sigma \ll q, \sigma[\rho]$, where $\sigma[\mathsf{id}_{\mathsf{bn}(p)}] = \sigma$ because $\mathsf{dom}(\sigma) = \mathsf{bn}(p)$. Now it suffices to prove that $(\sigma\hat{P}, \sigma[\rho]\hat{Q}) \in \Re$. This follows from the hypothesis that $P' \sim \rho Q'$: indeed, by closure of $\sim$ under substitutions, $\sigma P' \sim \sigma(\rho Q') = \sigma[\rho]Q'$; by Lemma 3.23, $\sigma P'|Q|R \sim \sigma[\rho]Q'|Q|R$. Now conclude: since $\mathsf{dom}(\sigma) = \mathsf{bn}(p)$ and $\mathsf{bn}(p) \cap \mathsf{fn}(Q|R) = \emptyset$, it holds that $\sigma\hat{P} = \sigma P'|Q|R$; since $\mathsf{dom}(\sigma[\rho]) = \mathsf{dom}(\rho) = \mathsf{bn}(q)$ and $\mathsf{bn}(q) \cap \mathsf{fn}(Q|R) = \emptyset$, it holds that $\sigma[\rho]\hat{Q} = \sigma[\rho]Q'|Q|R$; finally, by definition, $\sim \subseteq \Re$.

   (b) $\mu = q$ and $\hat{P} = p \to P'|Q'|Q|R$: in this case, since the action comes from Q, by the side condition of rule parext, it must be that $\mathsf{bn}(q) \cap \mathsf{fn}(p \to P'|R) = \emptyset$. Now, consider $Q|R \xrightarrow{q} Q'|Q|R = \hat{Q}$. By Lemma 3.14, $q, \sigma \ll q, \sigma$. It suffices to prove that $(\sigma\hat{P}, \sigma\hat{Q}) \in \Re$. This follows from the definition of $\Re$: since $\mathsf{dom}(\sigma) = \mathsf{bn}(q)$ and $\mathsf{bn}(q) \cap \mathsf{fn}(p \to P'|R) = \emptyset$, it holds that $\sigma\hat{P} = p \to P'|\sigma Q'|Q|R$ and $\sigma\hat{Q} = \sigma Q'|Q|R$.

   (c) $\mu = (\nu\widetilde{n})r$, $R \xrightarrow{\mu} R'$ and $\hat{P} = p \to P'|Q|R'$: in this case, by the side condition of rule parext, it must be that $\mathsf{bn}(r) \cap \mathsf{fn}(p \to P'|Q) = \emptyset$. Now, consider $Q|R \xrightarrow{\mu} Q|R' = \hat{Q}$ and reason like in the previous case, obtaining that $\sigma\hat{P} = p \to P'|Q|\sigma R' \;\Re\; Q|\sigma R' = \sigma\hat{Q}$.

2. $\mu = \tau$: in this case, there are five possible ways for producing μ:

   (a) $R \xrightarrow{\tau} R'$ and $\hat{P} = p \to P'|Q|R'$: this case is trivial.

   (b) $\hat{P} = \vartheta P'|\theta(Q'|Q)|R$, where $\{p\|q\} = (\vartheta, \theta)$: Let ρ be such that $p, \mathsf{id}_{\mathsf{bn}(p)} \ll q, \rho$; by Lemma 3.17, $\{q\|q\} = (\vartheta[\rho], \theta)$. But a pattern can match itself only if it is communicable; this entails that $\theta = \{\}$ and that p is communicable, thus $\vartheta = \{\}$. Hence, $\hat{P} = P'|Q'|Q|R$ and conclude by taking $Q|R \xrightarrow{\tau} Q'|Q'|Q|R = \hat{Q}$, since by hypothesis $P' \sim Q'$.

   (c) $\hat{P} = (\nu\widetilde{n})(\vartheta P'|Q|\theta R')$, where $R \xrightarrow{(\nu\widetilde{n})r} R'$ and $\{p\|r\} = (\vartheta, \theta)$: by α-conversion, now let $\mathsf{bn}(p) \cap \mathsf{fn}(Q) = \emptyset$ and $\widetilde{n} \cap \mathsf{fn}(p \to P'|Q) = \emptyset$.

Now consider $Q \xrightarrow{q} Q'|Q$ with $\mathsf{bn}(q) \cap \mathsf{fn}(Q) = \emptyset$; by Lemma 3.17, the hypothesis $p, \mathsf{id}_{\mathsf{bn}(p)} \ll q, \rho$ entails $\{q \| r\} = (\vartheta[\rho], \theta)$. Thus, $Q|R \xrightarrow{\tau} (\nu\widetilde{n})(\vartheta[\rho](Q'|Q)|\theta R') = (\nu\widetilde{n})(\vartheta[\rho]Q'|Q|\theta R') = \hat{Q}$, where the first equality holds because $\mathsf{dom}(\vartheta[\rho]) = \mathsf{dom}(\rho) = \mathsf{bn}(q)$ and $\mathsf{bn}(q) \cap \mathsf{fn}(Q) = \emptyset$. Conclude by using the hypothesis $P' \sim \rho Q'$, thanks to closure of $\sim$ under substitutions, parallel and restriction.

(d) $\hat{P} = p \rightarrow P' \mid (\nu\widetilde{n})(\vartheta(Q'|Q)|\theta R')$, where $R \xrightarrow{(\nu\widetilde{n})r} R'$ and $\{q\|r\} = (\vartheta, \theta)$: this case is simple, by considering $Q|R \xrightarrow{\tau} (\nu\widetilde{n})(\vartheta(Q'|Q)|\theta R') = \hat{Q}$ and by observing that $\mathsf{dom}(\vartheta) = \mathsf{bn}(q)$, with $\mathsf{bn}(q) \cap \mathsf{fn}(Q) = \emptyset$.

(e) $\hat{P} = p \rightarrow P' \mid \vartheta Q'|\theta Q'|Q|R$, where $\{q\|q\} = (\vartheta, \theta)$: this case is trivial, by observing that $\vartheta = \theta = \{\}$. $\square$

To conclude, notice that the more general claim

> Let $P = p \rightarrow P' \mid !q \rightarrow Q'$ and $Q = !q \rightarrow Q'$; if there are σ and ρ such that $p, \sigma \ll q, \rho$ and $\sigma P' \sim \rho Q'$, then $P \sim Q$

does *not* hold. To see this, consider the following two processes:

$$\begin{array}{lll} P = \lambda x \rightarrow P' \mid Q & \text{with} & P' = (\nu n)(\ulcorner n \urcorner \bullet x \mid \ulcorner n \urcorner \bullet m \rightarrow \ulcorner w \urcorner) \quad \text{for } x \neq m \\ Q = !\lambda x \rightarrow Q' & \text{with} & Q' = (\nu n)(\ulcorner n \urcorner \bullet m \mid \ulcorner n \urcorner \bullet m \rightarrow \ulcorner w \urcorner) \end{array}$$

Trivially $\lambda x, \{m/x\} \ll \lambda x, \{m/x\}$ and $\{m/x\}P' \sim \{m/x\}Q' = Q'$; however, P is *not* bisimilar to Q. Indeed, in the context $\mathcal{C}(\cdot) = \cdot \mid k \rightarrow \mathbf{0}$, for $k \neq m$, they behave differently: $\mathcal{C}(P)$ can reduce in one step to a process that is stuck and cannot exhibit any barb on w; by contrast, every reduct of $\mathcal{C}(Q)$ reduces in another step to a process that exhibits a barb on w.[1] Theorem 3.45 is more demanding: it does not leave us free to choose whatever σ we want, but it forces us working with $\mathsf{id}_{\mathsf{bn}(p)}$. Now, the only ρ such that $\lambda x, \{x/x\} \ll \lambda x, \rho$ is $\{x/x\}$; with such a substitution, the second hypothesis of the theorem, in this case $P' \sim Q'$, does not hold and so we cannot conclude that $P \sim Q$.

4 Comparison with Other Process Calculi

This section exploits the techniques developed in [18, 19] to formally asses the expressive power of CPC with respect to π-calculus, Linda, Spi calculus and Fusion calculus. After briefly recalling these models and some basic material from [19], the relation to CPC is formalised. First, let each model, including CPC, be augmented with a reserved process '$\surd$', used to signal successful termination. This feature is needed to formulate what is a *valid* encoding in Definition 4.1. Moreover, we consider the same operators for all the calculi, i.e. parallel, restriction and replication. In particular, we avoid non-deterministic choice from π-calculus and Fusion, and the **case** construct from Psi. In this way, we focus our attention to the essential features of the calculi.

[1] As usual, for proving equivalences it is easier to rely on bisimulation, while for proving inequivalences it is easier to rely on barbed congruence. Thanks to Theorems 3.26 and 3.44 this approach works perfectly.

4.1 Some Process Calculi

π-calculus [28, 33]. The π-calculus processes are given by the following grammar:

$$P \ ::= \ \mathbf{0} \ | \ \surd \ | \ \overline{a}\langle b\rangle.P \ | \ a(x).P \ | \ (\nu n)P \ | \ P|Q \ | \ !P$$

and the only reduction axiom is

$$\overline{a}\langle b\rangle.P \mid a(x).Q \quad \longmapsto \quad P \mid \{b/x\}Q\ .$$

The reduction relation is obtained by closing this interaction rule by parallel, restriction and the same structural equivalence relation defined for CPC.

Linda [12]. Consider an instance of Linda formulated to follow CPC's syntax. Processes are defined as:

$$P \ ::= \ \mathbf{0} \ | \ \surd \ | \ \langle b_1,\ldots,b_k\rangle \ | \ (t_1,\ldots,t_k).P \ | \ (\nu n)P \ | \ P|Q \ | \ !P$$

where b ranges over names and t denotes a template field, defined by:

$$t \ ::= \ \lambda x \quad | \quad \ulcorner b \urcorner\ .$$

Assume that input variables occurring in templates are all distinct. This assumption rules out template $(\lambda x, \lambda x)$, but accepts $(\lambda x, \ulcorner b\urcorner, \ulcorner b\urcorner)$. Templates are used to implement Linda's pattern matching, defined as follows:

$$\textsc{Match}(\ ;\ ) = \{\} \qquad \textsc{Match}(\ulcorner b\urcorner; b) = \{\} \qquad \textsc{Match}(\lambda x; b) = \{b/x\}$$

$$\frac{\textsc{Match}(t;b) = \sigma_1 \qquad \textsc{Match}(\widetilde{t};\widetilde{b}) = \sigma_2}{\textsc{Match}(t,\widetilde{t}\,;\ b,\widetilde{b}) = \sigma_1 \uplus \sigma_2}$$

where '$\uplus$' denotes the union of partial functions with disjoint domains. The interaction axiom is:

$$\langle\widetilde{b}\rangle \mid (\widetilde{t}).P \longmapsto \sigma P \qquad \text{if } \ \textsc{Match}(\widetilde{t};\widetilde{b}) = \sigma\ .$$

The reduction relation is obtained by closing this interaction rule by parallel, restriction and the same structural equivalence relation defined for CPC.

Spi calculus [3]. This language is unusual as names are now generalised to *terms* of the form

$$M, N \quad ::= \quad n \quad | \quad x \quad | \quad (M,N) \quad | \quad 0 \quad | \quad suc(M) \quad | \quad \{M\}_N$$

They are rather similar to the patterns of CPC in that they may have internal structure. Of particular interest are the pair, successor and encryption that may be bound to a name and then decomposed later by an intensional reduction.

The processes of the Spi calculus are:

$$\begin{array}{rcl} P,Q & ::= & 0 \ | \ \surd \ | \ P|Q \ | \ !P \ | \ (\nu m)P \ | \ M(x).P \ | \ \overline{M}\langle N\rangle.P \\ & & | \ [M \ is \ N]P \ | \ let\ (x,y) = M \ in \ P \\ & & | \ case \ M \ of \ \{x\}_N : P \ | \ case \ M \ of \ 0 : P \ suc(x) : Q\ . \end{array}$$

The nil process, parallel composition, replication and restriction are all familiar. The input $M(x).P$ and output $M\langle N\rangle.P$ are generalised from π-calculus to allow arbitrary terms in the place of channel names and output arguments. The match $[M\ is\ N]P$ determines equality of M and N. The splitting $let\ (x, y) = M\ in\ P$ decomposes pairs. The decryption $case\ M\ of\ \{x\}_N : P$ decrypts M and binds the encrypted message to x. The integer test $case\ M\ of\ 0 : P\ suc(x) : Q$ branches according to the number. Note that the last four processes can all get stuck if M is an incompatible term. Furthermore, the last three are intensional, i.e. they depend on the internal structure of M.

Concerning the operational semantics, we consider a slightly modified version of Spi calculus where interaction is generalised to

$$\overline{M}\langle N\rangle.P \mid M(x).Q \quad \longmapsto \quad P \mid \{N/x\}Q$$

where M is any term of the Spi calculus. The remaining axioms are:

$$\begin{array}{rcl} [M\ is\ M]P & \longmapsto & P \\ let\ (x, y) = (M, N)\ in\ P & \longmapsto & \{M/x, N/y\}P \\ case\ \{M\}_N\ of\ \{x\}_N : P & \longmapsto & \{M/x\}P \\ case\ 0\ of\ 0 : P\ suc(x) : Q & \longmapsto & P \\ case\ suc(N)\ of\ 0 : P\ suc(x) : Q & \longmapsto & \{N/x\}Q \end{array}$$

Again, the reduction relation is obtained by closing the interaction axiom under parallel, restriction and the structural equivalence of CPC.

Fusion [30]. Following the presentation in [35], processes are defined as:

$$P \ ::= \ \mathbf{0} \ \mid \ \surd \ \mid \ P|P \ \mid \ (\nu x)P \ \mid \ !P \ \mid \ \overline{u}\langle\widetilde{x}\rangle.P \ \mid \ u(\widetilde{x}).P\ .$$

The interaction rule for Fusion is

$$(\nu\widetilde{u})(\overline{u}\langle\widetilde{x}\rangle.P \mid u(\widetilde{y}).Q \mid R) \longmapsto \sigma P \mid \sigma Q \mid \sigma R \quad \begin{array}{l} \text{with } \mathsf{dom}(\sigma) \cup \mathsf{ran}(\sigma) \subseteq \{\widetilde{x}, \widetilde{y}\} \\ \text{and } \widetilde{u} = \mathsf{dom}(\sigma) \setminus \mathsf{ran}(\sigma) \\ \text{and } \sigma(v) = \sigma(w) \\ \text{iff } (v, w) \in E(\widetilde{x} = \widetilde{y}) \end{array}$$

where $E(\widetilde{x} = \widetilde{y})$ is the least equivalence relation on names generated by the equalities $\widetilde{x} = \widetilde{y}$ (that is defined whenever $|\widetilde{x}| = |\widetilde{y}|$). Fusion's reduction relation is obtained by closing the interaction axiom under parallel, restriction and the structural equivalence of CPC.

Psi [6]. For our purposes, Psi-calculi are parametrized w.r.t. two sets: terms $\mathbf{T}$, ranged over by $M, N, \ldots$, and assertions $\mathbf{A}$, ranged over by Ψ. The empty assertion is written $\mathbf{1}$. We also assume two operators: channel equivalence, $\dot{\leftrightarrow} \subseteq \mathbf{T} \times \mathbf{T}$, and assertion composition, $\otimes : \mathbf{A} \times \mathbf{A} \to \mathbf{A}$. It is also required that $\dot{\leftrightarrow}$ is transitive and symmetric, and that $(\otimes, \mathbf{1})$ is a commutative monoid.

Processes in Psi are defined as:

$$P \ ::= \ \mathbf{0} \ \mid \ \surd \ \mid \ P|P \ \mid \ (\nu x)P \ \mid \ !P \ \mid \ \overline{M}\langle N\rangle.P \ \mid \ M(\lambda\widetilde{x})N.P \ \mid \ (\!|\, \Psi \,|\!)$$

We now give a reduction semantics, by isolating the τ actions of the LTS given in [6]. To this aim, we recall the definiton of frame of a process P, written $\mathcal{F}(P)$, as the set of unguarded assertions occurring in P. Formally:

$$\mathcal{F}((\!|\, \Psi \,|\!)) = \Psi \qquad \mathcal{F}((\nu x)P) = (\nu x)\mathcal{F}(P) \qquad \mathcal{F}(P|Q) = \mathcal{F}(P) \otimes \mathcal{F}(Q)$$

and is $\mathbf{1}$ in all other cases. We denote as $(\nu\widetilde{b}_P)\Psi_P$ the frame of P. The structural laws are the same as in π-calculus. The reduction relation is inferred by the following laws:

$$\frac{\Psi \vdash M \dot{\leftrightarrow} N}{\Psi \rhd \overline{M}\langle K\rangle.P \mid N(\lambda\widetilde{x})H.Q \longmapsto P \mid \{\widetilde{L}/\widetilde{x}\}Q}\; K = H[\widetilde{x} := \widetilde{L}]$$

$$\frac{\Psi \otimes \Psi_Q \rhd P \longmapsto P'}{\Psi \rhd P \mid Q \longmapsto P' \mid Q}\; \mathcal{F}(Q) = (\nu\widetilde{b}_Q)\Psi_Q, \widetilde{b}_Q \text{ fresh for } \Psi \text{ and } P$$

$$\frac{\Psi \rhd P \longmapsto P'}{\Psi \rhd (\nu x)P \longmapsto (\nu x)P'}\; x \notin \text{names}(\Psi) \qquad \frac{P \equiv Q \quad \Psi \rhd Q \longmapsto Q' \quad Q' \equiv P'}{\Psi \rhd P \longmapsto P'}$$

We write $P \longmapsto P'$ whenever $\mathbf{1} \rhd P \longmapsto P'$.

4.2 Valid Encodings and their Properties

An *encoding* of a language $\mathcal{L}_1$ into another language $\mathcal{L}_2$ is a pair $(\llbracket \cdot \rrbracket, \varphi_{\llbracket\rrbracket})$ where $\llbracket \cdot \rrbracket$ translates every $\mathcal{L}_1$-process into an $\mathcal{L}_2$-process and $\varphi_{\llbracket\rrbracket}$ maps every source name into a k-tuple of (target) names, for $k > 0$. The translation $\llbracket \cdot \rrbracket$ turns every source term into a target term; in doing this, the translation may fix some names to play a precise rôle or may translate a single name into a tuple of names. This can be obtained by exploiting $\varphi_{\llbracket\rrbracket}$ (details in [19]).

Now consider only encodings that satisfy the following properties, that are justified and discussed at length in [19]. Let a *k-ary context* $\mathcal{C}(\_1; \ldots; \_k)$ be a term where k occurrences of $\mathbf{0}$ are linearly replaced by the holes $\{\_1; \ldots; \_k\}$ (every hole must occur once and only once). Moreover, denote with $\longmapsto_i$ and $\Longmapsto_i$ the relations $\longmapsto$ and $\Longmapsto$ in language $\mathcal{L}_i$; denote with $\longmapsto_i^\omega$ an infinite sequence of reductions in $\mathcal{L}_i$. Moreover, we let $\simeq_i$ denote the reference behavioural equivalence for language $\mathcal{L}_i$. Also, let $P \Downarrow_i$ mean that there exists P' such that $P \Longmapsto_i P'$ and $P' \equiv P'' \mid \surd$, for some P''. Finally, to simplify reading, let S range over processes of the source language (viz., $\mathcal{L}_1$) and T range over processes of the target language (viz., $\mathcal{L}_2$).

Definition 4.1 (Valid Encoding)**.** *An encoding* $(\llbracket \cdot \rrbracket, \varphi_{\llbracket\rrbracket})$ *of* $\mathcal{L}_1$ *into* $\mathcal{L}_2$ *is* valid *if it satisfies the following five properties:*

1. Compositionality: *for every k-ary operator* op *of $\mathcal{L}_1$ and for every subset of names N, there exists a k-ary context $\mathcal{C}^N_{\mathsf{op}}(\_1; \ldots; \_k)$ of $\mathcal{L}_2$ such that, for all $S_1, \ldots, S_k$ with $\mathsf{fn}(S_1, \ldots, S_k) = N$, it holds that $\llbracket \mathsf{op}(S_1, \ldots, S_k) \rrbracket = \mathcal{C}^N_{\mathsf{op}}(\llbracket S_1 \rrbracket; \ldots; \llbracket S_k \rrbracket)$.*

2. Name invariance: *for every S and name substitution σ, it holds that*

$$\llbracket \sigma S \rrbracket \begin{cases} = \sigma'\llbracket S \rrbracket & \text{if } \sigma \text{ is injective} \\ \simeq_2 \sigma'\llbracket S \rrbracket & \text{otherwise} \end{cases}$$

where σ' is such that $\varphi_{\llbracket\rrbracket}(\sigma(a)) = \sigma'(\varphi_{\llbracket\rrbracket}(a))$ for every name a.

3. Operational correspondence:

   - *for all $S \Longmapsto_1 S'$, it holds that $\llbracket S \rrbracket \Longmapsto_2 \simeq_2 \llbracket S' \rrbracket$;*

- *for all* $[\![S]\!] \Longmapsto_2 T$*, there exists* S' *such that* $S \Longmapsto_1 S'$ *and* $T \Longmapsto_2 \simeq_2 [\![S']\!]$.

4. Divergence reflection: *for every* S *such that* $[\![S]\!] \longmapsto_2^\omega$*, it holds that* $S \longmapsto_1^\omega$.

5. Success sensitiveness: *for every* S*, it holds that* $S \Downarrow_1$ *if and only if* $[\![S]\!] \Downarrow_2$.

[19] contains some results concerning valid encodings. In particular, it shows some proof-techniques for showing separation results, i.e. for proving that no valid encoding can exist between a pair of languages $\mathcal{L}_1$ and $\mathcal{L}_2$ satisfying certain conditions. Here, these languages will be limited to CPC and those introduced in Section 4.1. In *loc.cit.*, the valid encodings considered are assumed to be *semi-homomorphic*, i.e. where the interpretation of parallel composition is via a context of the form $(\nu\widetilde{n})(\_{1} \mid \_{2} \mid R)$, for some $\widetilde{n}$ and R that only depend on the free names of the translated processes. This assumption simplified the proofs of the following results in general, i.e. without relying on any specific process calculus; in our setting, since the languages are fixed, we can prove the same results without assuming semi-homomorphism.

Proposition 4.2 (from [19])**.** *Let* $[\![\cdot]\!]$ *be a valid encoding; then,* $S \not\longmapsto_1$ *implies that* $[\![S]\!] \not\longmapsto_2$.

Theorem 4.3 (from [19])**.** *Assume that there exists* S *such that* $S \not\longmapsto_1$*,* $S \not\Downarrow_1$ *and* $S \mid S \Downarrow_1$*; moreover, assume that every* T *that does not reduce is such that* $T \mid T \not\longmapsto_2$*. Then, there cannot exist any valid encoding of* $\mathcal{L}_1$ *into* $\mathcal{L}_2$.

To state the following proof-technique, define the *matching degree* of a language $\mathcal{L}$, written $\textsc{Md}(\mathcal{L})$, as the least upper bound on the number of names that must be matched to yield a reduction in $\mathcal{L}$. For example, $\textsc{Md}(\pi\text{-calculus}) = 1$, since the only name matched for performing a reduction is the name of the channel where the communication happens, whereas $\textsc{Md}(\text{Linda}) = \textsc{Md}(\text{CPC}) = \infty$, since there is no upper bound on the number of names that can be matched in a reduction.

Theorem 4.4 (from [19])**.** *If* $\textsc{Md}(\mathcal{L}_1) > \textsc{Md}(\mathcal{L}_2)$*, then there exists no valid encoding of* $\mathcal{L}_1$ *into* $\mathcal{L}_2$.

4.3 CPC vs π-calculus and Linda

A hierarchy of sets of process calculi with different communication primitives is obtained by Gorla [18] via combining four features: synchronism (synchronous vs asynchronous), arity (monadic vs polyadic data exchange), communication medium (channels vs shared dataspaces), and the presence of a form of pattern matching (that checks the arity of the tuple of names and equality of some specific names). This hierarchy is built upon a very similar notion of encoding to that presented in Definition 4.1 and, in particular, it is proved that Linda [12] (called $L_{\,\text{A,P,D,PM}}$ in [18]) is more expressive than monadic/polyadic π-calculus [28, 27] (called $L_{\,\text{S,M,C,NO}}$ and $L_{\,\text{S,P,C,NO}}$, respectively, in [18]).

As Linda is more expressive than π-calculus, it is sufficient to show that CPC is more expressive than Linda. However, apart from being a corollary of such a result, the lack of a valid encoding of CPC into π-calculus can also be shown by exploiting the matching degree, i.e. Theorem 4.4: the matching degree of π-calculus is one, while the matching degree of CPC is infinite.

Theorem 4.5. *There is no valid encoding of CPC into Linda.*

Proof: The self-matching CPC process $S = x \to \surd$ is such that $S \not\longmapsto$ and $S \not\Downarrow$, however $S \mid S \longmapsto$ and $S \mid S \Downarrow$. Every Linda process T such that $T \mid T \longmapsto$ can reduce in isolation, i.e. $T \longmapsto$: this fact can be easily proved by induction on the structure of T. Conclude by Theorem 4.3. $\square$

The next step is to show a valid encoding of Linda into CPC. The encoding $[\![\cdot]\!]$ is homomorphic with respect to all operators except for input and output which are encoded as follows:

$$\begin{aligned} [\![(\widetilde{t}).P]\!] &\stackrel{\text{def}}{=} \mathsf{pat\text{–}t}(\widetilde{t}) \to [\![P]\!] \\ [\![\langle\widetilde{b}\rangle]\!] &\stackrel{\text{def}}{=} \mathsf{pat\text{–}d}(\widetilde{b}) \to \mathbf{0}\ . \end{aligned}$$

The functions $\mathsf{pat\text{–}t}(\cdot)$ and $\mathsf{pat\text{–}d}(\cdot)$ are used to translate templates and data, respectively, into CPC patterns. The functions are defined as follows:

$$\begin{aligned} \mathsf{pat\text{–}t}(\ ) &\stackrel{\text{def}}{=} \lambda x \bullet \mathsf{in} && \text{for } x \text{ a fresh name} \\ \mathsf{pat\text{–}t}(t,\widetilde{t}) &\stackrel{\text{def}}{=} t \bullet \mathsf{in} \bullet \mathsf{pat\text{–}t}(\widetilde{t}) \\ \mathsf{pat\text{–}d}(\ ) &\stackrel{\text{def}}{=} \mathsf{in} \bullet \lambda x \\ \mathsf{pat\text{–}d}(b,\widetilde{b}) &\stackrel{\text{def}}{=} b \bullet \lambda x \bullet \mathsf{pat\text{–}d}(\widetilde{b}) && \text{for } x \text{ a fresh name} \end{aligned}$$

where in is any name (a symbolic name is used for clarity but no result relies upon this choice). Moreover, the function $\mathsf{pat\text{–}d}(\cdot)$ associates a bound variable to every name in the sequence; this fact ensures that a pattern that translates a datum and a pattern that translates a template match only if they have the same length (this is a feature of Linda's pattern matching but not of CPC's). It is worth noting that the simpler translation $[\![\langle b_1,\ldots,b_n\rangle]\!] \stackrel{\text{def}}{=} b_1 \bullet \ldots \bullet b_n \to \mathbf{0}$ would not work: the Linda process $\langle b\rangle \mid \langle b\rangle$ does not reduce, whereas its encoding would, in contradiction with Proposition 4.2.

Next is to prove that this encoding is valid. This is an easy corollary of the following lemma, stating a strict correspondence between Linda's pattern matching and CPC's one (on patterns arising from the translation).

Lemma 4.6. $\textsc{Match}(\widetilde{t};\widetilde{b}) = \sigma$ *if and only if* $\{\mathsf{pat\text{–}t}(\widetilde{t})\|\mathsf{pat\text{–}d}(\widetilde{b})\} = (\sigma \cup \{\mathsf{in}/x\}, \{\mathsf{in}/x_0,\ldots,\mathsf{in}/x_n\})$, *where* $\{x_0,\ldots,x_n\} = \mathsf{bn}(\mathsf{pat\text{–}d}(\widetilde{b}))$ *and* $\mathsf{dom}(\sigma) \uplus \{x\} = \mathsf{bn}(\mathsf{pat\text{–}t}(\widetilde{t}))$ *and* σ *maps names to names.*

Proof: In both directions the proof is by induction on the length of $\widetilde{t}$. The forward direction is as follows.

- The base case is when $\widetilde{t}$ is the empty sequence of template fields; thus, $\mathsf{pat\text{–}t}(\widetilde{t}) = \lambda x \bullet \mathsf{in}$. By definition of $\textsc{Match}$, it must be that $\widetilde{b}$ is the empty sequence and that σ is the empty substitution. Thus, $\mathsf{pat\text{–}d}(\widetilde{b}) = \mathsf{in} \bullet \lambda x$ and the thesis easily follows.

- For the inductive step $\widetilde{t} = t,\widetilde{t'}$ and $\mathsf{pat\text{–}t}(\widetilde{t}) = t \bullet \mathsf{in} \bullet \mathsf{pat\text{–}t}(\widetilde{t'})$. By definition of $\textsc{Match}$, it must be that $\widetilde{b} = b,\widetilde{b'}$ and $\textsc{Match}(t,b) = \sigma_1$ and $\textsc{Match}(\widetilde{t'},\widetilde{b'}) = \sigma_2$ and $\sigma = \sigma_1 \uplus \sigma_2$. By the induction hypothesis, $\{\mathsf{pat\text{–}t}(\widetilde{t'})\|\mathsf{pat\text{–}d}(\widetilde{b'})\} = (\sigma_2 \cup \{\mathsf{in}/x\};\{\mathsf{in}/x_1,\ldots,\mathsf{in}/x_n\})$, where

$\{x_1, \ldots, x_n\} = \mathsf{bn}(\mathsf{pat\text{–}d}(\widetilde{b'}))$ and $\mathsf{dom}(\sigma_2) \uplus \{x\} = \mathsf{bn}(\mathsf{pat\text{–}t}(\widetilde{t'}))$. There are now two sub-cases to consider according to the kind of template field t.

  - If $t = \ulcorner b \urcorner$ then $\sigma_1 = \{\}$; thus, $\sigma = \sigma_2$ and $\{\mathsf{pat\text{–}t}(\widetilde{t}) \| \mathsf{pat\text{–}d}(\widetilde{b})\} = (\sigma \cup \{\mathsf{in}/x\}, \{\mathsf{in}/x_0, \ldots, \mathsf{in}/x_n\})$.
  - If $t = \lambda y$ then $\sigma_1 = \{b/y\}$ and $y \notin \mathsf{dom}(\sigma_2)$. Thus, $\mathsf{pat\text{–}t}(\widetilde{t})$ is a pattern in CPC and it follows that $\{\mathsf{pat\text{–}t}(\widetilde{t}) \| \mathsf{pat\text{–}d}(\widetilde{b})\} = (\sigma_1 \cup \sigma_2 \cup \{\mathsf{in}/x\}, \{\mathsf{in}/x_0, \ldots, \mathsf{in}/x_n\}) = (\sigma \cup \{\mathsf{in}/x\}, \{\mathsf{in}/x_0, \ldots, \mathsf{in}/x_n\})$.

The reverse direction is as follows.

- The base case is when $\widetilde{t}$ is the empty sequence of template fields; thus, $\mathsf{pat\text{–}t}(\widetilde{t}) = \lambda x \bullet \mathsf{in}$. Now proceed by contradiction. Assume that $\widetilde{b}$ is not the empty sequence. In this case, $\mathsf{pat\text{–}d}(\widetilde{b}) = b_0 \bullet \lambda x_0 \bullet (b_1 \bullet \lambda x_1 \bullet (\ldots (b_n \bullet \lambda x_n \bullet (\mathsf{in} \bullet \lambda x_{n+1})) \ldots)$, for some $n > 0$. By definition of pattern matching in CPC, $\mathsf{pat\text{–}d}(\widetilde{b})$ and $\mathsf{pat\text{–}t}(\widetilde{t})$ cannot match, and this would contradict the hypothesis. Thus, it must be that $\widetilde{b}$ is the empty sequence and we easily conclude.

- The inductive case is when $\widetilde{t} = t, \widetilde{t'}$ and thus, $\mathsf{pat\text{–}t}(\widetilde{t}) = t \bullet \mathsf{in} \bullet \mathsf{pat\text{–}t}(\widetilde{t'})$. If $\widetilde{b}$ was the empty sequence, then $\mathsf{pat\text{–}d}(\widetilde{b}) = \mathsf{in} \bullet \lambda x$ and it would not match against $\mathsf{pat\text{–}t}(\widetilde{t})$. Hence, $\widetilde{b} = b, \widetilde{b'}$ and so $\mathsf{pat\text{–}d}(\widetilde{b}) = b \bullet \lambda x \bullet \mathsf{pat\text{–}d}(\widetilde{b'})$. By definition of pattern-unification in CPC it follows that $\{t \| b\} = (\sigma_1, \{\})$ and $\{\mathsf{pat\text{–}t}(\widetilde{t'}) \| \mathsf{pat\text{–}d}(\widetilde{b'})\} = (\sigma_2 \cup \{\mathsf{in}/x\}, \{\mathsf{in}/x_1, \ldots, \mathsf{in}/x_n\})$ and $\sigma = \sigma_1 \cup \sigma_2$. Now consider the two sub-cases according to the kind of the template field t.

  - If $t = \ulcorner b \urcorner$ then $\sigma_1 = \{\}$ and so $\sigma_2 = \sigma$. By induction hypothesis, $\textsc{Match}(\widetilde{t'}; \widetilde{b'}) = \sigma$, and so $\textsc{Match}(\widetilde{t}; \widetilde{b}) = \sigma$.
  - If $t = \lambda y$ then $\sigma_1 = \{b/y\}$ and $\sigma_2 = \{n_i/y_i\}$ for $y_i \in \mathsf{dom}(\sigma) \backslash \{y\}$ and $n_i = \sigma y_i$. Thus, $y \notin \mathsf{dom}(\sigma_2)$ and so $\sigma = \sigma_1 \uplus \sigma_2$. By the induction hypothesis, $\textsc{Match}(\widetilde{t'}; \widetilde{b'}) = \sigma_2$; moreover, $\textsc{Match}(t; b) = \sigma_1$. Thus, $\textsc{Match}(\widetilde{t}; \widetilde{b}) = \sigma$. □

Lemma 4.7. *If $P \equiv Q$ then $[\![P]\!] \equiv [\![Q]\!]$. Conversely, if $[\![P]\!] \equiv Q$ then $Q = [\![P']\!]$, for some $P' \equiv P$.*

Proof: Trivial, from the fact that $\equiv$ acts only on operators that $[\![\cdot]\!]$ translates homomorphically. □

Theorem 4.8. *The translation $[\![\cdot]\!]$ from Linda into CPC preserves and reflects reductions. That is:*

- *If $P \longmapsto P'$ then $[\![P]\!] \longmapsto [\![P']\!]$;*
- *if $[\![P]\!] \longmapsto Q$ then $Q = [\![P']\!]$ for some P' such that $P \longmapsto P'$.*

Proof: Both parts can be easily proved by a straightforward induction on judgements $P \longmapsto P'$ and $[\![P]\!] \longmapsto Q$, respectively. In both cases, the base step is the most interesting one and it trivially follows from Lemma 4.6; the inductive cases where the last rule used is the structural one rely on Lemma 4.7. □

Corollary 4.9. *The encoding of Linda into CPC is valid.*

Proof: Compositionality and name invariance hold by construction. Operational correspondence and divergence reflection easily follow from Theorem 4.8. Success sensitiveness can be proved as follows: $P \Downarrow$ means that there exist P' and $k \geq 0$ such that $P \longmapsto^k P' \equiv P'' \mid \surd$; by exploiting Theorem 4.8 k times and Lemma 4.7, we obtain that $[\![P]\!] \longmapsto^k [\![P']\!] \equiv [\![P'']\!] \mid \surd$, i.e. that $[\![P]\!] \Downarrow$. The converse implication can be proved similarly. □

4.4 CPC vs Spi

That CPC cannot be encoded into Spi calculus is a corollary of Theorem 4.3 and identical to the technique used in Section 4.3: the self-matching CPC process $x \to \surd$ cannot be properly rendered in Spi.

The remainder of this section develops an encoding of Spi calculus into CPC. The terms can be encoded as patterns using the reserved names pair, encr, 0 and suc by

$$\begin{array}{rclcrcl}
[\![n]\!] & \stackrel{\mathrm{def}}{=} & n & \qquad & [\![suc(M)]\!] & \stackrel{\mathrm{def}}{=} & \mathsf{suc} \bullet [\![M]\!] \\
[\![x]\!] & \stackrel{\mathrm{def}}{=} & x & & [\![(M,N)]\!] & \stackrel{\mathrm{def}}{=} & \mathsf{pair} \bullet [\![M]\!] \bullet [\![N]\!] \\
[\![0]\!] & \stackrel{\mathrm{def}}{=} & 0 & & [\![\{M\}_N]\!] & \stackrel{\mathrm{def}}{=} & \mathsf{encr} \bullet [\![M]\!] \bullet [\![N]\!] \, .
\end{array}$$

The tagging is used for safety, as otherwise there are potential pathologies in the translation: for example, without tags, the representation of an encrypted term could be confused with a pair.

The encoding of the familiar process forms are homomorphic as expected. The input and output both encode as cases:

$$\begin{array}{rcll}
[\![M(x).P]\!] & \stackrel{\mathrm{def}}{=} & [\![M]\!] \bullet \lambda x \bullet \mathsf{in} \to [\![P]\!] & \\
[\![\overline{M}\langle N \rangle.P]\!] & \stackrel{\mathrm{def}}{=} & [\![M]\!] \bullet ([\![N]\!]) \bullet \lambda x \to [\![P]\!] & \qquad x \text{ is a fresh name.}
\end{array}$$

The symbolic name in (input) and fresh name x (output) are used to ensure that encoded inputs will only match with encoded outputs as for Linda.

The four remaining process forms all require pattern matching and so translate to cases in parallel. In each encoding a fresh name n is used to prevent interaction with other processes, see Lemma 2.5. As in the Spi calculus, the encodings will reduce only after a successful matching and will be stuck otherwise. The encodings are

$$\begin{array}{rcl}
[\![[M \ is \ N]P]\!] & \stackrel{\mathrm{def}}{=} & (\nu n)(\ulcorner n \urcorner \bullet [\![M]\!] \to [\![P]\!] \mid \ulcorner n \urcorner \bullet [\![N]\!]) \\
[\![let \ (x,y) = M \ in \ P]\!] & \stackrel{\mathrm{def}}{=} & (\nu n)(\ulcorner n \urcorner \bullet (\ulcorner \mathsf{pair} \urcorner \bullet \lambda x \bullet \lambda y) \to [\![P]\!] \\
& & \quad \mid \ulcorner n \urcorner \bullet [\![M]\!]) \\
[\![case \ M \ of \ \{x\}_N : P]\!] & \stackrel{\mathrm{def}}{=} & (\nu n)(\ulcorner n \urcorner \bullet (\ulcorner \mathsf{encr} \urcorner \bullet \lambda x \bullet [\![N]\!]) \to [\![P]\!] \\
& & \quad \mid \ulcorner n \urcorner \bullet [\![M]\!]) \\
[\![case \ M \ of \ 0 : P \ suc(x) : Q]\!] & \stackrel{\mathrm{def}}{=} & (\nu n)(\ulcorner n \urcorner \bullet \ulcorner 0 \urcorner \to [\![P]\!] \\
& & \quad \mid \ulcorner n \urcorner \bullet (\ulcorner \mathsf{suc} \urcorner \bullet \lambda x) \to [\![Q]\!] \\
& & \quad \mid \ulcorner n \urcorner \bullet [\![M]\!]) \, .
\end{array}$$

The match $[M\ is\ N]P$ only reduces to P if $M = N$, thus the encoding creates two patterns using $[\![M]\!]$ and $[\![N]\!]$ with one reducing to $[\![P]\!]$. The encoding of pair splitting $let\ (x, y) = M\ in\ P$ creates a case with a pattern that matches a tagged pair and binds the components to x and y in $[\![P]\!]$. This is put in parallel with another case that has $[\![M]\!]$ in the pattern. The encoding of a decryption $case\ M\ of\ \{x\}_N : P$ checks whether $[\![M]\!]$ is encoded with key $[\![N]\!]$ and retrieves the value encrypted by binding it to x in the continuation. Lastly the encoding of an integer test $case\ M\ of\ 0 : P\ suc(x) : Q$ creates a case for each of the zero and the successor possibilities. These cases match the tag and the reserved names 0, reducing to $[\![P]\!]$, or suc and binding x in $[\![Q]\!]$. The term to be compared $[\![M]\!]$ is as in the other cases.

Let us now prove validity of this encoding.

Lemma 4.10. *If $P \equiv Q$ then $[\![P]\!] \equiv [\![Q]\!]$. Conversely, if $[\![P]\!] \equiv Q$ then $Q = [\![P']\!]$, for some $P' \equiv P$.*

Proof: Trivial, from the fact that $\equiv$ acts only on operators that $[\![\cdot]\!]$ translates homomorphically. $\square$

Theorem 4.11. *The translation $[\![\cdot]\!]$ from Spi calculus into CPC preserves and reflects reductions, up-to CPC's barbed congruence. That is:*

- *If $P \longmapsto P'$ then $[\![P]\!] \longmapsto\simeq_2 [\![P']\!]$;*
- *if $[\![P]\!] \longmapsto Q$ then $Q \simeq_2 [\![P']\!]$ for some P' such that $P \longmapsto P'$*

Proof: The first claim can be easily proved by a straightforward induction on judgement $P \longmapsto P'$. The base case is proved by reasoning on the Spi axiom used to infer the reduction. Although all the cases are straightforward, a reduction rule for integers is shown for illustration. Consider the reduction for a successor as the reduction for zero is simpler. In this case, $P = case\ suc(M)\ of\ 0 : P_1\ suc(x) : P_2$ and $P' = \{M/x\}P_2$. Then,

$$\begin{aligned} [\![P]\!] \stackrel{\text{def}}{=} & (\nu n)(\ulcorner n \urcorner \bullet (\mathsf{num} \bullet \ulcorner 0 \urcorner) \rightarrow [\![P_1]\!] \\ & \quad | \ulcorner n \urcorner \bullet (\mathsf{num} \bullet (\ulcorner \mathsf{suc} \urcorner \bullet \lambda x)) \rightarrow [\![P_2]\!] \\ & \quad | \ulcorner n \urcorner \bullet (\mathsf{num} \bullet (\mathsf{suc} \bullet [\![M]\!])) \rightarrow \mathbf{0}) . \end{aligned}$$

and it can only reduce to

$$\{[\![M]\!]/x\}[\![P_2]\!] \mid (\nu n)\ulcorner n \urcorner \bullet (\mathsf{num} \bullet \ulcorner 0 \urcorner) \rightarrow [\![P_1]\!]$$

By a straightforward induction on the structure of P_2 it is easy to prove that $\{[\![M]\!]/x\}[\![P_2]\!] = [\![\{M/x\}P_2]\!]$. Thus, $[\![P]\!] \longmapsto [\![\{M/x\}P_2]\!] \mid (\nu n)\ulcorner n \urcorner \bullet (\mathsf{num} \bullet \ulcorner 0 \urcorner) \rightarrow [\![P_1]\!] \simeq_2 [\![P']\!]$, where the last equivalence follows from Lemma 2.5. The inductive case is straightforward, with the structural case relying on Lemma 4.10.

The second part can be proved by induction on judgement $[\![P]\!] \longmapsto Q$. There is just one base case, i.e. when $[\![P]\!] = p \rightarrow Q_1 \mid q \rightarrow Q_2$ and $Q = \sigma Q_1 \mid \rho Q_2$ and $\{p \| q\} = (\sigma, \rho)$. By definition of the encoding, it can only be that $p = [\![M]\!] \bullet \lambda x \bullet \mathsf{in}$ and $Q_1 = [\![P_1]\!]$ and $q = [\![M]\!] \bullet ([\![N]\!]) \bullet \lambda x$ and $Q_2 = [\![P_2]\!]$ for some P_1, P_2, M and N. This means that $P = M(x).P_1 \mid \overline{M}\langle N \rangle.P_2$ and that $Q = \{[\![N]\!]/x\}[\![P_1]\!] \mid [\![P_2]\!] = [\![\{N/x\}P_1 \mid P_2]\!]$. To conclude, it suffices to take $P' = \{N/x\}P_1 \mid P_2$. For the inductive case there are two possibilities.

- The inference of $[\![P]\!] \longmapsto Q$ ends with an application of the rule for parallel composition or for structural equivalence: this case can be proved by a straightforward induction.

- The inference of $[\![P]\!] \longmapsto Q$ ends with an application of the rule for restriction; thus, $[\![P]\!] = (\nu n)Q'$, with $Q' \longmapsto Q''$ and $Q = (\nu n)Q''$. If $Q' = [\![P'']\!]$, for some P'', apply a straightforward induction. Otherwise, there are the following four possibilities.

  - $Q' = \ulcorner n \urcorner \bullet \ulcorner [\![M]\!] \urcorner \to [\![P_1]\!] \mid \ulcorner n \urcorner \bullet \ulcorner [\![N]\!] \urcorner$ and, hence, $Q'' = [\![P_1]\!]$. By definition of the encoding, $P = [M \ is \ N]P_1$. Notice that the reduction $Q' \longmapsto Q''$ can happen only if $[\![M]\!]$ and $[\![N]\!]$ match; by construction of the encoding of Spi-terms, this can happen only if $M = N$ and, hence, $P \longmapsto P_1$. The thesis follows by letting $P' = P_1$, since n is a fresh name and so $Q = (\nu n)[\![P_1]\!] \equiv [\![P_1]\!]$.
  - $Q' = \ulcorner n \urcorner \bullet (\ulcorner \mathsf{pair} \urcorner \bullet (\lambda x \bullet \lambda y)) \to [\![P_1]\!] \mid \ulcorner n \urcorner \bullet (\mathsf{pair} \bullet ([\![M]\!] \bullet [\![N]\!]))$ and, hence, $Q'' = \{[\![M]\!]/x, [\![N]\!]/y\}[\![P_1]\!]$. This case is similar to the previous one, by letting P be $let\ (x, y) = (M, N)\ in\ P_1$.
  - $Q' = \ulcorner n \urcorner \bullet (\ulcorner \mathsf{encr} \urcorner \bullet (\lambda x \bullet [\![N]\!])) \to [\![P_1]\!] \mid \ulcorner n \urcorner \bullet (\mathsf{encr} \bullet ([\![M]\!] \bullet [\![N]\!]))$ and, hence, $Q'' = \{[\![M]\!]/x\}[\![P_1]\!]$. This case is similar to the previous one, by letting P be $case\ \{M\}_N\ of\ \{x\}_N : P_1$.
  - $Q' = \ulcorner n \urcorner \bullet (\mathsf{num} \bullet \ulcorner 0 \urcorner) \to [\![P_1]\!] \mid \ulcorner n \urcorner \bullet (\mathsf{num} \bullet (\ulcorner \mathsf{suc} \urcorner \bullet \lambda x)) \to [\![P_2]\!] \mid \ulcorner n \urcorner \bullet [\![M]\!]$. Hence, $P = case\ M\ of\ 0 : P_1\ suc(x) : P_2$. According to the kind of $[\![M]\!]$, there are two sub-cases (notice that, since $Q' \longmapsto Q''$, no other possibility is allowed for $[\![M]\!]$).
    * $[\![M]\!] = \mathsf{num} \bullet 0$: in this case, $Q'' = [\![P_1]\!] \mid \ulcorner n \urcorner \bullet (\mathsf{num} \bullet (\ulcorner \mathsf{suc} \urcorner \bullet \lambda x)) \to [\![P_2]\!]$ and so $Q = (\nu n)Q'' \equiv [\![P_1]\!] \mid (\nu n)\ulcorner n \urcorner \bullet (\mathsf{num} \bullet (\ulcorner \mathsf{suc} \urcorner \bullet \lambda x)) \to [\![P_2]\!] \simeq_2 [\![P_1]\!]$. In this case, $M = 0$ and so $P \longmapsto P_1$; to conclude, it suffices to let P' be P_1.
    * $[\![M]\!] = \mathsf{num} \bullet (\mathsf{suc} \bullet [\![M']\!])$, for some M': in this case, $Q'' = \{[\![M']\!]/x\}[\![P_2]\!] \mid \ulcorner n \urcorner \bullet (\mathsf{num} \bullet \ulcorner 0 \urcorner) \to [\![P_1]\!]$ and so $Q = (\nu n)Q'' \equiv [\![\{M'/x\}P_2]\!] \mid (\nu n)\ulcorner n \urcorner \bullet (\mathsf{num} \bullet 0) \to [\![P_1]\!] \simeq_2 [\![\{M'/x\}P_2]\!]$. In this case, $M = suc(M')$ and so $P \longmapsto \{M'/x\}P_2$; to conclude, it suffices to let P' be $\{M'/x\}P_2$. □

Corollary 4.12. *The encoding of Spi calculus into CPC is valid.*

Proof: See the proof for Corollary 4.9. □

Notice that the criteria for a valid encoding do not imply full abstraction of the encoding (actually, they were defined as an alternative to full abstraction [18, 19]). This means that the encoding of equivalent Spi calculus processes can be distinguished by contexts in CPC that do not result from the encoding of any Spi calculus context. Indeed, while this encoding allows Spi calculus to be modelled in CPC, it does *not* entail that cryptography can be properly rendered. Consider the pattern $\mathsf{encr} \bullet \lambda x \bullet \lambda y$ that could match the encoding of an encrypted term to bind the message and key, so that CPC can break any encryption! Indeed this is an artefact of the straightforward approach to encoding taken here. Some discussion of alternative approaches to encryption in CPC are detailed in the first authors PhD dissertation [13].

4.5 CPC vs Fusion

As the separation results for CPC and the other process calculi presented so far can all be proved via symmetry, the relationship between Fusion and CPC is of particular interest. Such calculi are *unrelated*, in the sense that there exists no valid encoding from one into the other. The impossibility for a valid encoding of CPC into Fusion can be proved in two ways, by exploiting the matching degree or symmetry of CPC.

Theorem 4.13. *There is no valid encoding of CPC into Fusion.*

Proof: The matching degree of Fusion is 1 while the matching degree of CPC is infinite; conclude by Theorem 4.4. Alternatively, reuse the proof for Theorem 4.5 as every Fusion process T is such that $T \mid T \longmapsto$ implies $T \longmapsto$. $\square$

The converse separation result is ensured by the following theorem.

Theorem 4.14. *There exists no valid encoding of Fusion into CPC.*

Proof: By contradiction, assume that there exists a valid encoding $[\![\cdot]\!]$ of Fusion into CPC. Consider the Fusion process $P \stackrel{\text{def}}{=} (\nu x)(\overline{u}\langle x\rangle \mid u(y).\surd)$, for x, y and u pairwise distinct. By success sensitiveness, $P \Downarrow$ entails that $[\![P]\!] \Downarrow$.

We first prove that $[\![P]\!]$ must reduce before reporting success, i.e. that every occurrence of $\surd$ in $[\![P]\!]$ falls underneath some prefix. By compositionality, $[\![P]\!] \stackrel{\text{def}}{=} \mathcal{C}^{\{u,x,y\}}_{(\nu x)}(\mathcal{C}^{\{u,x,y\}}_{|}([\![\overline{u}\langle x\rangle]\!];[\![u(y).\surd]\!]))$. If $[\![P]\!]$ had a top-level unguarded occurrence of $\surd$, then such an occurrence could be in $\mathcal{C}^{\{u,x,y\}}_{(\nu x)}(\_)$, in $\mathcal{C}^{\{u,x,y\}}_{|}(\_{1};\_{2})$, in $[\![\overline{u}\langle x\rangle]\!]$ or in $[\![u(y).\surd]\!]$; in any case, it would also follow that at least one of $[\![(\nu x)(\overline{u}\langle x\rangle \mid y(u).\surd)]\!]$ or $[\![(\nu x)(\overline{x}\langle u\rangle \mid u(y).\surd)]\!]$ would report success, whereas both $(\nu x)(\overline{u}\langle x\rangle \mid y(u).\surd) \not\Downarrow$ and $(\nu x)(\overline{x}\langle u\rangle \mid u(y).\surd) \not\Downarrow$, against success sensitiveness of $[\![\cdot]\!]$. Thus, the only possibility for $[\![P]\!]$ to report success is to perform some reduction steps (at least one) and then exhibit a top-level unguarded occurrence of $\surd$.

We now prove that every possible reduction leads to contradict validity of $[\![\cdot]\!]$; this suffices to conclude. There are five possibilities for $[\![P]\!] \longmapsto$.

1. Either $\mathcal{C}^{\{u,x,y\}}_{(\nu x)} \longmapsto$, or $\mathcal{C}^{\{u,x,y\}}_{|} \longmapsto$, or $[\![\overline{u}\langle x\rangle]\!] \longmapsto$ or $[\![u(y).\surd]\!] \longmapsto$. In any of these cases, at least one out of $[\![(\nu x)(\overline{u}\langle x\rangle \mid y(u).\surd)]\!]$ or $[\![(\nu x)(\overline{x}\langle u\rangle \mid u(y).\surd)]\!]$ would reduce; however, $(\nu x)(\overline{u}\langle x\rangle \mid y(u).\surd) \not\longmapsto$ and $(\nu x)(\overline{x}\langle u\rangle \mid u(y).\surd) \not\longmapsto$, against Proposition 4.2 (that must hold whenever $[\![\cdot]\!]$ is valid).

2. Reduction is generated by interaction between $\mathcal{C}^{\{u,x,y\}}_{(\nu x)}$ and $\mathcal{C}^{\{u,x,y\}}_{|}$. As before, $[\![(\nu x)(\overline{u}\langle x\rangle \mid y(u).\surd)]\!] \longmapsto$ whereas $(\nu x)(\overline{u}\langle x\rangle \mid y(u).\surd) \not\longmapsto$, against Proposition 4.2.

3. Reduction is generated by interaction between $\mathcal{C}^{\{u,x,y\}}_{\mathsf{op}}$ and $[\![\overline{u}\langle x\rangle]\!]$, for $\mathsf{op} \in \{(\nu x), |\}$. Like case 2.

4. Reduction is generated by interaction between $\mathcal{C}^{\{u,x,y\}}_{\mathsf{op}}$ and $[\![u(y).\surd]\!]$, for $\mathsf{op} \in \{(\nu x), |\}$. As before it follows that $[\![(\nu x)(\overline{x}\langle u\rangle \mid u(y).\surd)]\!] \longmapsto$ whereas $(\nu x)(\overline{x}\langle u\rangle \mid u(y).\surd) \not\longmapsto$, against Proposition 4.2.

5. The reduction is generated by an interaction between the processes $[\![\, \overline{u}\langle x\rangle \,]\!]$ and $[\![\, u(y).\surd \,]\!]$. In this case, it follows that $[\![\, \overline{u}\langle x\rangle \mid u(y).\surd \,]\!] \longmapsto$ whereas $\overline{u}\langle x\rangle \mid u(y).\surd \not\longmapsto$: indeed, the interaction rule of Fusion imposes that at least one between x and y must be restricted to yield the interaction. $\square$

4.6 CPC vs Psi

CPC and Psi are *unrelated*, in the sense that there exists no valid encoding from one into the other. The impossibility for a valid encoding of CPC into Psi can be proved in two ways, by exploiting the matching degree or symmetry of CPC.

Theorem 4.15. *There is no valid encoding of CPC into Psi.*

Proof: The matching degree of Psi is 1 while the matching degree of CPC is infinite; conclude by Theorem 4.4. Alternatively, reuse the proof for Theorem 4.5 as every Psi process T is such that $T \mid T \longmapsto$ implies $T \longmapsto$. $\square$

The converse separation result is ensured by the following theorem.

Theorem 4.16. *There exists no valid encoding of Psi into CPC.*

Proof: Assume that there exists a valid encoding $[\![\cdot]\!]$ of Psi into CPC. Consider the Psi process $P \stackrel{\text{def}}{=} (\bar{a}.c \mid b.(\surd \mid c)) \mid (\!| a \dot{\leftrightarrow} b |\!)$, where we have omitted the argument of the actions because useless, and choose a, b and c pairwise distinct; also consider the reduction

$$\dfrac{\dfrac{\{a \dot{\leftrightarrow} b\} \vdash a \dot{\leftrightarrow} b}{\{a \dot{\leftrightarrow} b\} \rhd \bar{a}.c \mid b.(\surd \mid c) \longmapsto c \mid \surd \mid c}}{\mathbf{1} \rhd P \longmapsto (c \mid \surd \mid c) \mid (\!| a \dot{\leftrightarrow} b |\!)}$$

Thus, $P \Downarrow$ and, by success sensitiveness, $[\![\, P \,]\!] \Downarrow$. By compositionality, $[\![\, P \,]\!] \stackrel{\text{def}}{=} \mathcal{C}_{\mid}^{\{a,b,c\}}(\mathcal{C}_{\mid}^{\{a,b,c\}}([\![\, \bar{a}.c \,]\!];[\![\, b.(\surd \mid c) \,]\!]);[\![\, (\!| a \dot{\leftrightarrow} b |\!) \,]\!])$. Like in the proof of Theorem 4.14, it is easy to prove that the only possibility for $[\![\, P \,]\!]$ to report success is to perform some reduction steps (at least one) and then exhibit a top-level unguarded occurrence of $\surd$.

We now prove that every possible reduction leads to contradict validity of $[\![\cdot]\!]$; this suffices to conclude. Of course, none of $[\![\, \bar{a}.c \,]\!]$, $[\![\, b.(\surd \mid c) \,]\!]$ and $[\![\, (\!| a \dot{\leftrightarrow} b |\!) \,]\!]$ can reduce, because $\bar{a}.c$, $b.(\surd \mid c)$ and $(\!| a \dot{\leftrightarrow} b |\!)$ do not reduce. Thus, there are seven possibilities for $[\![\, P \,]\!] \longmapsto$.

1. Either $\mathcal{C}_{\mid}^{\{a,b,c\}} \longmapsto$ or the reduction is obtained by synchronizing the two copies of $\mathcal{C}_{\mid}^{\{a,b,c\}}$. In both cases, also $[\![\, (\bar{c}.a \mid b.(\surd \mid c)) \mid (\!| a \dot{\leftrightarrow} b |\!) \,]\!]$ would reduce, whereas $(\bar{c}.a \mid b.(\surd \mid c)) \mid (\!| a \dot{\leftrightarrow} b |\!) \not\longmapsto$, against Proposition 4.2 (that must hold whenever $[\![\cdot]\!]$ is valid).

2. The reduction is obtained by synchronizing $[\![\, \bar{a}.c \,]\!]$ with (one of the two copies of) $\mathcal{C}_{\mid}^{\{a,b,c\}}$. In this case, also $[\![\, (\bar{a}.c \mid c.(\surd \mid b)) \mid (\!| a \dot{\leftrightarrow} b |\!) \,]\!]$ would reduce, whereas $(\bar{a}.c \mid c.(\surd \mid b)) \mid (\!| a \dot{\leftrightarrow} b |\!) \not\longmapsto$.

3. The reduction is obtained by synchronizing $[\![\, b.(\surd \mid c)\,]\!]$ with (one of the two copies of) $\mathcal{C}_{|}^{\{a,b,c\}}$. This case is proved impossible like case 1 above.

4. The reduction is obtained by synchronizing $[\![\, (\!| a \dot{\leftrightarrow} b |\!) \,]\!]$ with (one of the two copies of) $\mathcal{C}_{|}^{\{a,b,c\}}$. This case is proved impossible like cases 1 and 2 above.

5. The reduction is obtained by synchronizing $[\![\, \bar{a}.c]\!]$ with $[\![\, b.(\surd \mid c)\,]\!]$. In this case, also $[\![\, (\bar{a}.c \mid b.(\surd \mid c)) \mid (\!| a \dot{\leftrightarrow} c |\!)]\!]$ would reduce, whereas $(\bar{a}.c \mid b.(\surd \mid c)) \mid (\!| a \dot{\leftrightarrow} c |\!) \not\longmapsto$.

6. The reduction is obtained by synchronizing $[\![\, \bar{a}.c]\!]$ with $[\![\, (\!| a \dot{\leftrightarrow} b |\!)]\!]$. This case is proved impossible like case 2 above.

7. The reduction is obtained by synchronizing $[\![\, b.(\surd \mid c)\,]\!]$ with $[\![\, (\!| a \dot{\leftrightarrow} b |\!)]\!]$. This case is proved impossible like case 1 above.

□

5 Conclusions and Future Work

Concurrent pattern calculus uses patterns to represent input, output and tests for equality, whose interaction is driven by unification that allows a two-way flow of information. This symmetric information exchange provides a concise model of trade in the information age. This is illustrated by the example of traders who can discover each other in the open and then close the deal in private.

As patterns drive interaction in CPC their properties heavily influence CPC's behaviour theory. As pattern unification may match any number of names these must all be accounted for in the definition of barbs. More delicately, some patterns are compatible with others, in that their unifications yield similar results. The resulting bisimulation requires that the transitions be compatible patterns rather than exact.

CPC supports valid encodings of many popular concurrent calculi such as π-calculus, Spi calculus and Linda as its patterns describe more structures. However, these three calculi do not support valid encodings of CPC because, among other things, they are insufficiently symmetric. On the other hand, while fusion calculus is completely symmetric, it has an incompatible approach to interaction.

Future work may progress along several different directions. Expressiveness wise there are closer links to: sequential computation, in particular to SF-calculus [23]; and to other process calculi. Another path for exploration is the development of programming languages and implementations of CPC.

Acknowledgements. Thanks to ...

References

[1] bondi programming language. `www-staff.it.uts.edu.au/~cbj/bondi`.

[2] CppLINDA: C++ LINDA implementation, 2010. Retrieved 18 November 2010, from http://sourceforge.net/projects/cpplinda/.

[3] M. Abadi and A. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1 – 70, 1999.

[4] R. M. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous π-calculus. *Theoretical Computer Science*, 195(2):291–324, 1998.

[5] H. P. Barendregt. *The Lambda Calculus. Its Syntax and Semantics.* Studies in Logic and the Foundations of Mathematics. Elsevier Science Publishers B.V., 1985.

[6] J. Bengtson, M. Johansson, J. Parrow, and B. Victor. Psi-calculi: Mobile processes, nominal data, and logic. In *Proc. of LICS*, pages 39–48. IEEE, 2009.

[7] L. Bettini, R. De Nicola, and R. Pugliese. KLAVA: a Java package for distributed and mobile applications. *Software – Practice and Experience*, 32(14):1365–1394, 2002.

[8] A. L. Brown, C. Laneve, and L. G. Meredith. Piduce: A process calculus with native XML datatypes. In *In Proc. of EPEW05/WS-FM05, volume 3670 of Lect*, pages 18–34. Springer, 2005.

[9] T. Caml language. The Caml language: Home, 2011. Retrieved 8 January 2011, from http://caml.inria.fr/.

[10] Concurrent bondi. Concurrent bondi, 2011. Retrieved 4 August 2011, from `http://www-staff.it.uts.edu.au/~tgwilson/concurrent_bondi/`.

[11] R. De Nicola, D. Gorla, and R. Pugliese. Basic observables for a calculus for global computing. *Information and Computation*, 205(10):1491–1525, 2007.

[12] D. Gelernter. Generative communication in LINDA. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.

[13] T. Given-Wilson. PhD thesis, Sydney.

[14] T. Given-Wilson. *Interpreting the Untyped Pattern Calculus in **bondi***. Honours Thesis, University of Technology, Sydney, Sydney, Australia, August 2007.

[15] T. Given-Wilson. Concurrent pattern calculus in **bondi**. *Young Researchers Workshop on Concurrency Theory (YR-CONCUR)*, 2010.

[16] T. Given-Wilson, F. Huang, and B. Jay. Multi-polymorphic programming in **bondi**, 2007.

[17] T. Given-Wilson and B. Jay. Getting the goods with concurrent **bondi**. *Proceedings of the Fourth Workshop on Programming Language Approaches to Concurrency and Communication-cEntric Software (PLACES)*, 2011.

[18] D. Gorla. Comparing communication primitives via their relative expressive power. *Information and Computation*, 206(8):931–952, 2008.

[19] D. Gorla. Towards a unified approach to encodability and separation results for process calculi. *Information and Computation*, 208(9):1031–1053, 2010.

[20] J. Gosling, B. Joy, and G. Steele. *The Java Language Specification*. Addison-Wesley Professional, third edition, 2005.

[21] K. Honda and N. Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 151(2):437–486, 1995.

[22] B. Jay. *Pattern Calculus: Computing with Functions and Data Structures*. Springer, 2009.

[23] B. Jay and T. Given-Wilson. A combinatory account of internal structure. *Journal of Symbolic Logic*, 2011.

[24] B. Jay and D. Kesner. First-class patterns. *Journal of Functional Programming*, 19(2):34 pages, 2009.

[25] JoCaml. The JoCaml system, 2010. Retrieved 1 February 2011, from http://jocaml.inria.fr/.

[26] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[27] R. Milner. The polyadic π-calculus: A tutorial. In F. L. Bauer, W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specification*, volume 94 of *Series F*. NATO ASI, Springer, 1993.

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

[29] R. Milner and D. Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *Proceedings of ICALP '92*, volume 623 of *LNCS*, pages 685–695. Springer, 1992.

[30] J. Parrow and B. Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *Proc. of LICS*, pages 176–185. IEEE Computer Society, 1998.

[31] G. Picco, A. Murphy, and G.-C. Roman. LIME: Linda Meets Mobility. In D. Garlan, editor, *Proc. of the 21st Int. Conference on Software Engineering (ICSE'99)*, pages 368–377. ACM Press, 1999.

[32] B. C. Pierce and D. N. Turner. Pict: A programming language based on the pi-calculus. In *PROOF, LANGUAGE AND INTERACTION: ESSAYS IN HONOUR OF ROBIN MILNER*, pages 455–494. MIT Press, 1997.

[33] D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[34] A. Unyapoth and P. Sewell. Nomadic Pict: Correct communication infrastructures for mobile computation. In *Proc. of POPL'01*, pages 116–127. ACM, 2001.

[35] L. Wischik and P. Gardner. Explicit fusions. *Theor. Comput. Sci.*, 340(3):606–630, 2005.