A Concurrent Pattern Calculus

Thomas Given-Wilson, Daniele Gorla, Barry Jay

A Concurrent Pattern Calculus

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

[1] INRIA, Paris, France *

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

[3] University of Technology, Sydney

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, so that bisimulation becomes 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 [34, 41] holds an honoured position amongst process calculi as a succinct calculus that can capture topological changes in a network, as well as encode computation as represented by λ-calculus [5]. Interaction in π-calculus is done by matching upon a single name known by both the input and output primitives. The polyadic π-calculus extends this by also matching on the length of the tuple of names to be communicated. Linda [15] extends this further by allowing matching on any number of names known by both processes. A more symmetric approach to communication is taken in Fusion calculus [37] that matches a channel name and tuple length, like polyadic π-calculus, but allows symmetric information exchange. Other calculi consider structured information rather than simply names [3], even matching arbitrary structures asymmetrically during communication [7]. Hence it is natural to explore how a concurrent pattern calculus can unify structured patterns with symmetric matching and communication mechanisms.

This paper develops pattern unification in a setting that supports parallel composition, name restriction, and replication. This yields *concurrent pattern calculus* (CPC), where prefixes for input and output are generalised to patterns whose *unification* triggers a symmetric flow of information, as represented by the sole interaction rule

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

*This work has been partially supported by the project ANR-12-IS02-001 PACE.

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

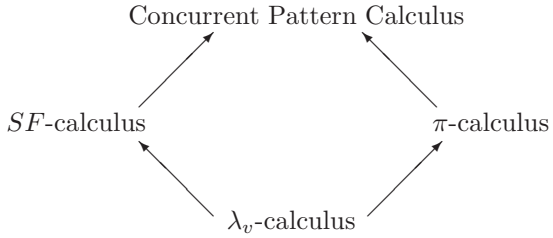
The flexibility of the pattern unification and the symmetry of exchange in CPC align closely with the world of trade. Here the support for discovering a compatible process and exchanging information mirrors the behaviour of trading systems such as a stock market. The main features of CPC are illustrated in the following sample trade interaction:

$$\begin{array}{rl} & (\nu\ \mathit{sharesID})\ulcorner \mathit{ABCShares}\urcorner \bullet \mathit{sharesID} \bullet \lambda x \rightarrow \langle \mathit{charge}\ x\ \mathit{for}\ \mathit{sale}\rangle \\ | & (\nu\ \mathit{bankAcc})\ulcorner \mathit{ABCShares}\urcorner \bullet \lambda y \bullet \mathit{bankAcc} \rightarrow \langle \mathit{save}\ y\ \mathit{as}\ \mathit{proof}\rangle \\ \\ \longmapsto & (\nu\ \mathit{sharesID})(\nu\ \mathit{bankAcc})(\langle \mathit{charge}\ \mathit{bankAcc}\ \mathit{for}\ \mathit{sale}\rangle \\ & \quad | \ \langle \mathit{save}\ \mathit{sharesID}\ \mathit{as}\ \mathit{proof}\rangle) \end{array}$$

The first line models a seller that will synchronise with a buyer, using the protected information *ABCShares*, and exchange its shares (*sharesID*) for bank account information (bound to x). The second line models a buyer. Notice that the information exchange is bidirectional and simultaneous: *sharesID* replaces y in the (continuation of the) buyer and *bankAcc* replaces x in the (continuation of the) seller. Moreover, the two patterns $\ulcorner \mathit{ABCShares}\urcorner \bullet \mathit{sharesID} \bullet \lambda x$ and $\ulcorner \mathit{ABCShares}\urcorner \bullet \lambda y \bullet \mathit{bankAcc}$ also specify the details of the stock being traded, that must be matched for equality in the pattern matching, as indicated by the syntax $\ulcorner \cdot \urcorner$.

Pattern unification in CPC is even richer than indicated in this example, as unification may bind a compound pattern to a single name; that is, patterns do not need to be fully decomposed in unification. For example, the bank account information could be specified, and matched upon, in much more detail. The buyer could provide the account name and number such as in the following pattern: $(\nu\ \mathit{accName})(\nu\ \mathit{accNum})\ulcorner \mathit{ABCShares}\urcorner \bullet \lambda y \bullet (\mathit{name} \bullet \mathit{accName} \bullet \mathit{number} \bullet \mathit{accNum})$. This more detailed buyer would still match against the seller, now yielding $\langle \mathit{charge}\ \mathit{name} \bullet \mathit{accName} \bullet \mathit{number} \bullet \mathit{accNum}\ \mathit{for}\ \mathit{sale}\rangle$. Indeed, the seller could also specify a desire to only accept bank account information whose structure includes some name and number (bound to a and b respectively) with the following pattern: $\ulcorner \mathit{ABCShares}\urcorner \bullet \mathit{sharesID} \bullet (\ulcorner \mathit{name}\urcorner \bullet \lambda a \bullet \ulcorner \mathit{number}\urcorner \bullet \lambda b)$ and continuation $\langle \mathit{charge}\ a\ b\ \mathit{for}\ \mathit{sale}\rangle$. This would also match with the detailed buyer information by unifying *name* with $\ulcorner \mathit{name}\urcorner$, *number* with $\ulcorner \mathit{number}\urcorner$, and binding *accName* and *accNum* to a and b respectively. The second seller exploits the structural matching of CPC to only interact with a buyer whose pattern is of the right structure (four sub-patterns) and contains the right information (the protected names *name* and *number*, and shared information in the other two positions).

The structural patterns of CPC are inspired by those of *pattern calculus* [26, 24] that supports even more computations than λ-calculus, since pattern-matching functions may be *intensional* with respect to their arguments [25]. For example, the pattern $x\ y$ can decompose any compound data structure $u\ v$ into its components u and v. This is different from *extensional* computation, like those of λ- and π-calculus, where arguments are 'atomic'. This rich form of structural interaction, combined with concurrency, makes CPC very expressive, as illustrated by the following diamond [16]:



The λ-calculus sits at the bottom and can be generalised either by SF-calculus [25], by considering intensionality in the sequential setting, or by π-calculus, by considering concurrency in the extensional setting. CPC completes the diamond by adding either concurrency to SF-calculus, or intensionality to π-calculus. Thus, CPC is the most expressive of all by supporting intensional concurrent computation.

The definition of CPC also includes a behavioural theory that defines when two processes are behaviourally equivalent. This is done using a standard approach in concurrency. First, define an intuitive notion of equivalence that equates processes with the same behaviour (i.e., with the same interaction capabilities), 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] or the symbolic characterization of open bisimilarity in the π-calculus [40].

CPC's support for interaction that is both structured and symmetrical allows CPC to simulate many approaches to interaction and reduction in the literature [19]. For example, checking equality of channel names, as in π-calculus [34], can be viewed as a trivial form of pattern unification. It also supports unification of tuples of names, as in Linda [15], or fusing names, as in Fusion [37]. Spi calculus [3] adds patterns for numbers (zero and successors) and encryptions. Also the Psi calculus [7] introduces support for structures, albeit with a limited symmetry.

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 is not encodable into CPC; conversely, the richness of CPC's pattern unification is not encodable in fusion calculus. Similarly, the implicit computation of name equivalence in Psi calculus cannot be encoded within CPC; the converse separation result is ensured by CPC's symmetry.

A natural objection to CPC is that its 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 polyadic π-calculus' arbitrarily large tuple communication nor Linda's pattern matching are atomic, but both underpin many existing programming environments [39, 2, 9, 38]. Indeed the arbitrary complexity of both Psi calculus and CPC patterns can exceed the capability of any computer, yet both have implementations [28, 11]. Simlar comments apply to other process calculi [27, 42]. A further complexity is the secure synchronisation and information exchange between agents in distinct locations [13, 14, 6], however since the focus here is on exploring structured, symmetric, pattern unification and not implementations, we do not attempt to address these details.

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 the paper. Standard proofs have been moved to the Appendix.

Note that this paper significantly builds upon "Concurrent Pattern Calculus" that first introduced the titled work [18]. Moreover, the behavioural theory has been presented in [16, 17].

2 Concurrent Pattern Calculus

This section presents a *concurrent pattern calculus* (CPC) that uses symmetric pattern unification 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 knowledge of the channel is symmetric but information flows in one direction only. Fusion calculus achieves symmetry by fusing names together but has no intensional patterns. Linda's matching is more intensional as it 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:

Patterns	p	::=	λx	binding name
			x	variable name
			$\ulcorner x \urcorner$	protected name
			$p \bullet p$	compound

A binding name λx denotes information sought, e.g. by a trader; a variable name x represents such information. Note that a binding name binds the underlying process, defined in Section 2.2. Protected names $\ulcorner x \urcorner$ represent information that can be checked but not 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, and pure pattern calculus. The *atoms* are patterns that are not compounds. The atoms x and $\ulcorner x \urcorner$ *know* x.

Binding, variable and protected names are all 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, a form of protected names appear in Linda. There is some subtlety in the relationship of protected names 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 can be used to support channel-based communication, it is also natural that protected names unify with themselves.

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. This leads to the following definition.

Definition 2.1 (Communicable Pattern)**.** *A pattern is* communicable *if it contains no protected or binding names.*

Protection can be extended from names to communicable patterns by defining

$$\ulcorner p \bullet q \urcorner = \ulcorner p \urcorner \bullet \ulcorner q \urcorner$$

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 $\widetilde{n} \cap \mathsf{names}(\sigma) = \{\}$). Note that all substitutions considered in this paper have finite domain. For later convenience, we denote by id_X the identity substitution on a finite 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

$$\begin{aligned}
\sigma x &= \begin{cases} \sigma(x) & \text{if } x \in \mathsf{dom}(\sigma) \\ x & \text{otherwise} \end{cases} \\
\sigma \ulcorner x \urcorner &= \begin{cases} \ulcorner \sigma(x) \urcorner & \text{if } x \in \mathsf{dom}(\sigma) \\ \ulcorner x \urcorner & \text{otherwise} \end{cases} \\
\sigma(\lambda x) &= \lambda x \\
\sigma(p \bullet q) &= (\sigma p) \bullet (\sigma q)
\end{aligned}$$

The action of a substitution σ on patterns can be adapted to produce a function $\hat{\sigma}$ that acts on binding names rather than on free names. In CPC, it is defined by

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

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 unification is undefined.

Proposition 2.2. *If the unification of patterns p and q is defined then any protected name of p is a free name of q.*

Proof: By induction on the structure of p. □

2.2 Processes

The processes of CPC are given by:

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

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; $(\nu x)P$ binds x in P so that it is not visible from the outside. 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.

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

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

The *structural congruence relation* $\equiv$ is defined just as in π-calculus [33]: 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 has no effect on the null 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.

For later convenience, $\widetilde{x}$ denotes a sequence of x's, for example $\widetilde{n}$ can denote the names $n_1, \ldots, n_i$. Similarly, $(\nu n_1)(\ldots((\nu n_i)P))$ will be written $(\nu\widetilde{n})P$; however, due to structural congruence, these shall be considered as a set of names. For clarity set notation is used, for example $(\nu m_1)(\nu m_2)\ldots(\nu m_i)(\nu n_1)(\nu n_2)\ldots(\nu n_j)P$ shall be denoted $(\nu\widetilde{m} \cup \widetilde{n})P$.

The operational semantics of CPC is formulated via a *reduction relation* $\longmapsto$ 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), then the parallel composition of two cases $p \to P$ and $q \to Q$ reduces to the parallel composition of σP and ρQ. Alternatively, if the unification of p and q is undefined, then no interaction occurs. Unlike the sequential setting, there is no need for a rule to capture failure of unification since failure to interact with one process does not prevent interactions with other processes.

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

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

Proposition 2.3. *For every process P and substitution σ, it holds that $\mathsf{fn}(\sigma P) \subseteq \mathsf{fn}(P) \cup \mathsf{fn}(\sigma)$.*

Proof: By definition of the application of σ. □

Proposition 2.4. *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 on k and follows by Proposition 2.3. □

Proposition 2.5. *If $P \longmapsto P'$, then $\sigma P \longmapsto \sigma P'$, for every σ.*

Proof: By induction on the derivation of $P \longmapsto P'$. □

Proposition 2.6. *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 2.2, the free names of q include x and, consequently, x must be free in $q \to Q_1 \mid Q_2$. By Proposition 2.4, x is free in Q. Further, x cannot belong to $\widetilde{n}$, since $x \in \mathsf{fn}(p \to P)$ and $\widetilde{n} \cap \mathsf{fn}(p \to P) = \emptyset$. □

2.4 Trade in CPC

This section uses the example of share trading to explore the potential of CPC. The scenario is that two 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 trade from occurring.

The rest of this section develops three solutions of increasing sophistication that: demonstrate discovery; introduce a registrar to validate the traders; and protects 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

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

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

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

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 unification on s

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

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

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

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 trader is trustworthy.

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

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

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

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

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

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

The registrar creates a new identifier n and delivers it 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

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

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

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

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 $S(b)$.

Solution 3

Although Solution 2 satisfies the desire to validate that traders are legitimate, the freedom of unification 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,

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

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.

This vulnerability (emerging both in Solution 1 and 2) can be repaired by using protected names. For example, the buyer, seller and registrar of Solution 2 can become

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

Now all communications between the buyer, seller and registrar use protected identifiers: $\ulcorner i_B \urcorner, \ulcorner i_S \urcorner, \ulcorner n_B \urcorner, \ulcorner n_S \urcorner$ and $\ulcorner m \urcorner$. Thus, we just need to add the appropriate restrictions:

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

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 (Proposition 2.6).

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. We will prove that the two semantics do coincide. Then the bisimulation technique will be used to prove some sample equational laws for CPC.

3.1 Barbed Congruence

The first step is to characterise the interactions a process can participate in via *barbs*. Since a barb is an opportunity for interaction, a simplistic definition could be the following:

$$P \downarrow \text{ iff } P \equiv p \to P' \mid P'', \text{ for some } p, P' \text{ and } P'' \tag{1}$$

However, this definition is too strong: for example, $(\nu n)(n \to P)$ does not exhibit a barb according to (1), but it can interact with an external process, e.g. $\lambda x \to \mathbf{0}$. Thus, an improvement to (1) is as follows:

$$P \downarrow \text{ iff } P \equiv (\nu \widetilde{n})(p \to P' \mid P''), \text{ for some } \widetilde{n}, p, P' \text{ and } P'' \tag{2}$$

However, this definition is too weak. Consider $(\nu n)(\ulcorner n \urcorner \to P)$: it exhibits a barb according to (2), but cannot interact with any external process. A further refinement on (2) could be:

$$P \downarrow \text{ iff } P \equiv (\nu \widetilde{n})(p \to P' \mid P''), \text{ for some } \widetilde{n}, p, P', P'' \text{ s.t. } \mathsf{pn}(p) \cap \widetilde{n} = \emptyset \tag{3}$$

This definition is not yet the final one, as it is not sufficiently discriminating to have only a single kind of barb. Because of the rich form of interactions in CPC, there is no single identifier such as in CCS and π-calculus π-calculus [35], thus CPC barbs include the set of names that *may* be tested for equality in an interaction, not just those that *must* be equal. This leads to the following definition:

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

For later convenience, define $P \Downarrow_{\widetilde{m}}$ to mean that there exists some P' such that $P \Longmapsto P'$ and $P' \downarrow_{\widetilde{m}}$.

Using this definition, a barbed congruence can be defined in the standard way [35, 23] 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 '$\cdot$'.

Definition 3.2 (Barb preservation)**.** *$\Re$ is barb preserving iff, for every $(P, Q) \in \Re$ and set of names $\widetilde{m}$, 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 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 binary relation on CPC processes that is symmetric, barb preserving, reduction closed and context closed.*

Barbed congruence relates 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.

The challenge in proving barbed congruence is to prove context closure. The typical way of solving the problem is by giving a coinductive (bisimulation-based) characterization of barbed congruence, that provides a manageable proof technique. In turn, this requires an alternative operational semantics, by means of a labelled transition system, on top of which the bisimulation equivalence can be defined.

3.2 Labelled Transition System

The following is an adaption of the late LTS for the π-calculus [34]. *Labels* are defined as follows:

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

where τ is used to label silent transitions.

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, alternatively the transition $P \xrightarrow{\mu} P'$ indicates that P is able to *perform* μ and reduces to 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 open 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*, in π-calculus terminology). Rule unify 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

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

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

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

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

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

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

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

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

restricted names. Rule parint states that, if either process in a parallel composition can evolve with an internal action, then the whole process can evolve with 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, so that the side conditions can always be satisfied in practice.

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 vice versa. All proofs are in Appendix A, because they are quite standard.

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.*

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

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

To conclude, it is known that α-conversion must be handled with care [43]. A way in which we can leave it out from our presentation is to follow [8], that also has the advantage of being implementable in Isabelle/HOL [36]. However, we prefer to follow a more traditional approach in our presentation.

3.3 Bisimulation

We now develop a *bisimulation* relation for CPC that equates processes with the same interactional behaviour; this is 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, in that they unify with more things. 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 any challenge by one can be answered by a proper reply from the other.

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 [31, 34]. However, there are situations in which an exact reply would make the bisimulation equivalence too fine for characterising barbed congruence [4, 12]. 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

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

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 $\cdot \mid w \bullet w \to \ulcorner w \urcorner$, 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. unify with fewer patterns than others. This leads to the following definitions.

Definition 3.10. *Define a* match (p, σ) *to be a pattern* p *and substitution* σ *such that* $\mathsf{bn}(p) = \mathsf{dom}(\sigma)$.

Definition 3.11. *Let* (p, σ) *and* (q, ρ) *be matches. Define inductively that* p *is* compatible *with* q *by* σ *and* ρ*, denoted* $p, \sigma \ll q, \rho$ *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 with substitutions (σ, ρ) if and only if every other pattern r that unifies p by some substitutions (θ, σ) also unifies with q with substitutions (θ, ρ). That is, the patterns that unify with p are a subset of the patterns that unify with q. This will be proved later in Proposition 3.18.

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

Definition 3.12 (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 matches* (p, σ) *with* $\mathsf{fn}(\sigma) \cap \widetilde{n} = \emptyset$*, there exist a match* (q, ρ) *and* Q' *such that* $p, \sigma \ll q, \rho$ *and* $Q \xrightarrow{(\nu \widetilde{n}) q} Q'$ *and* $(\sigma P', \rho Q') \in \Re$*.*

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

The definition is inspired by the early bisimulation congruence for the π-calculus [34] (actually, it is what in [41] is called *strong full bisimilarity* – see Definition 2.2.2 therein): 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 opened by P. Also the action μ cannot be arbitrary, as in the π-calculus: its restricted and binding names cannot occur free in Q.

Unlike 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 highlighted 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'. This feature somehow resembles the symbolic characterization of open bisimilarity given in [40, 10]. There, labels are pairs made up of an action and a set of equality constraints. A challenge can be replied to by a smaller (i.e. less constraining) set. However, the action in the reply must be the same (in [40]) or becomes the same once we apply the name identifications induced by the equality constraints (in [10]).

An alternative approach may consider a standard bisimulation defined on top of an LTS that directly captures the difficulties of Examples 1 and 2. That is, allow $\lambda z \to z$ to reduce with label $\lambda x \bullet \lambda y$ to $x \bullet y$; similarly, allow $n \to 0$ to reduce with label $\ulcorner n \urcorner$ to 0. For example, this would allow the transition $\lambda w \bullet \lambda x \bullet n \to P \xrightarrow{\lambda w \bullet (\lambda y \bullet \lambda z) \bullet \ulcorner n \urcorner} \{y \bullet z/x\}P$. The difficulty with this approach is that, for every binding name in a pattern, there would be an infinite collection of transitions. For example, $\lambda x \to P$ would have transitions $\lambda x \to P \xrightarrow{\mu} \sigma P$, where μ can be $\lambda x_1 \bullet \ldots \bullet \lambda x_i$ (for every $i \geq 1$), but also $(\lambda y_1 \bullet \ldots \bullet \lambda y_j) \bullet \lambda z$ and $\lambda y \bullet (\lambda z_1 \bullet \ldots \bullet \lambda z_k)$, and so forth (with σ adapted appropriately). This would make working with the LTS very heavy (the LTS would not be finitely branching anymore); so the simplicity of relating patterns by compatibility is used here.

3.4 Properties of Compatibility

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.

Lemma 3.13. *If $p, \sigma \ll q, \rho$ then $\mathsf{fn}(p) = \mathsf{fn}(q)$ and $\mathsf{vn}(p) \subseteq \mathsf{vn}(q)$ and $\mathsf{pn}(q) \subseteq \mathsf{pn}(p)$.*

Proof: By definition of compatibility and induction on the structure of q. □

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 3.14. *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 interesting base case is when $q = \lambda y$. Since $\mathsf{dom}(\theta[\sigma]) = \mathsf{dom}(\sigma)$, we have that $p, \theta[\sigma] \ll q, \vartheta$, for $\vartheta = \{\widehat{\theta[\sigma]}(p)/y\} = \{\theta(\rho(y))/y\} = \theta[\rho]$. □

Proposition 3.15 (Compatibility is reflexive)**.** *For all matches (p, σ), it holds that $p, \sigma \ll p, \sigma$.*

Proof: By definition of compatibility. □

Proposition 3.16 (Compatibility is closed under substitution)**.** *If $p, \sigma \ll q, \rho$ then for all substitutions θ there exists σ' and ρ' such that $\theta p, \sigma' \ll \theta q, \rho'$.*

Proof: By induction on the structure of q. □

Proposition 3.17 (Compatibility is transitive)**.** *If $p, \sigma \ll q, \rho$ and $q, \rho \ll r, \theta$ then $p, \sigma \ll r, \theta$.*

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, $q = q_1 \bullet q_2$ and $\theta = \theta_1 \cup \theta_2$ 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. □

The next result captures the idea behind the definition of compatibility: the patterns that unify with p are a subset of the patterns that unify with q.

Proposition 3.18. *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 $\theta = \{\}$ and $\sigma = \{r_i/x_i\}_{x_i \in \mathsf{bn}(p)}$, 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 unify 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. □

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 unification with r.

Proposition 3.19. *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 3.11, 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 unification, $\{q \| r\} = (\{r/x\}, \{\})$ 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 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 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 3.11, $p = p_1 \bullet p_2$ 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 to conclude.

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

As compatibility is an ordering on matches, it is interesting to observe that, for every pattern p, there is a unique (up to α-conversion) maximal pattern w.r.t. $\ll$. Note that, as in Proposition 3.19 the substitution can be fixed (or indeed entirely elided).

Proposition 3.20. *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 the largest q w.r.t. $\ll$ is λy for some fresh y.
- If p is n or $\ulcorner n \urcorner$, then the largest q w.r.t. $\ll$ is n.
- Otherwise, if $p = p_1 \bullet p_2$, 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 that 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.

Lemma 3.21. *If $P \sim Q$ and $Q \sim R$ then $P \sim R$.*

Proof: Standard, by exploiting Proposition 3.17. □

Lemma 3.22. *$\sim$ is barb preserving.*

Proof: Straightforward by Lemmata 3.13 and 3.8, and by definition of the LTS. □

Closure under any context is less easy to prove. The approach here is as follows: prove bisimilarity is closed under case prefixing, name restriction and parallel composition; finally, prove closure under replication. Proofs of these lemmata are in Appendix B, because they are adaptions of the corresponding results for the π-calculus. These three results will easily entail closure under arbitrary contexts (Lemma 3.27).

Lemma 3.23. *If $P \sim Q$ then $p \to P \sim p \to Q$.*

Lemma 3.24. *If $P \sim Q$ then $(\nu \widetilde{n})P \sim (\nu \widetilde{n})Q$.*

Lemma 3.25. *If $P \sim Q$ then $P \mid R \sim Q \mid R$.*

Lemma 3.26. *If $P \sim Q$ then $!P \sim !Q$.*

Lemma 3.27. *$\sim$ 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. For the inductive step, we reason by case analysis on the outer operator of $\mathcal{C}(\cdot)$: then, we simply use the inductive hypothesis and Lemmata 3.23, 3.24, 3.25 and 3.26, respectively. □

Finally, we have to prove that bisimilarity is reduction closed; to this aim, we first need to prove that structural congruence is contained in bisimilarity.

Lemma 3.28. *$\equiv$ is a bisimulation closed under substitutions.*

Proof: For every structural axiom $LHS \equiv RHS$ it suffices to show that $\{(LHS, RHS)\} \cup \sim$ is a bisimulation. Closure under contexts follows from Lemma 3.27. Closure under substitutions follows from the fact that the axioms only involve bound names. The only exception is the rule for scope extension; however, the fact that substitution application is capture-avoiding allows us to conclude. □

Lemma 3.29. *$\sim$ is reduction closed.*

Proof: By Proposition 3.9 and Lemma 3.28. □

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

Theorem 3.30 (Soundness of bisimilarity)**.** *$\sim \subseteq \simeq$.*

Proof: Lemma 3.22, Lemma 3.27 and Lemma 3.29 entail that $\sim$ satisfies the conditions of Definition 3.5. □

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 obtained via closure under an appropriate context. To this aim, we define notions 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 can eventually report 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 $\widetilde{n}$ and p and P' such that $w \notin \widetilde{n}$ and $P' \not\Downarrow_f$.

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

Lemma 3.31. *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. □

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 3.48 by relying on Lemma 3.14.

The context for forcing a proper reply for a challenge of the form $((\nu\widetilde{n})p, \mathsf{id}_{\mathsf{bn}(p)})$ is developed in three steps. The outcome will be a process that interacts by some pattern p' and reduces to a collection of tests T such that θT succeeds if and only if $\{p \| p'\} = (\sigma, \theta)$. 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 succeeds 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 3.32. *The* specification $\mathsf{spec}^N(p)$ *of a pattern p with respect to a finite set of names N is defined follows:*

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

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 $R_p \uplus R_q$).

Observe that, since we only consider well formed patterns, the disjoint unions $F_p \uplus F_q$ and $R_p \uplus R_q$ are defined and the choices of the binding names for p' are pairwise distinct.

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 name (free in the process being tested) it will be bound to;

3. finally, R is a collection of pairs (x,n) made up by a binding name in p' and the expected name (restricted in the process being tested) it will be bound to.

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.

Lemma 3.33. *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. □

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:

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

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

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

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

Lemma 3.35. *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: Straightforward. □

Lemma 3.36. *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 3.37. *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 3.38. *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: Straightforward 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 3.39. *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*

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

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 3.40. *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 Lemmata 3.35, 3.36 and 3.37. Indeed, by Definition 3.32, $\mathsf{fst}(F \cup R) \cap (\mathsf{snd}(F \cup R) \cup N) = \emptyset$; moreover, freshness of w and f implies that $\{w, f\} \cap \mathsf{dom}(\theta) = \emptyset$. □

Note that the following results will 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 3.41. *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 $\widetilde{x} = \mathsf{fst}(R)$ and $Z \simeq \mathbf{0}$ and k depends only on F and 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 3.38. □

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 3.12) 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 3.42. *A* reply context $\mathcal{C}_p^N(\cdot)$ *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:*

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

Proposition 3.43. *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 3.42, 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 unifies 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 $\widetilde{m}$ and q and 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 3.41, $\theta(\mathsf{tests}_{F,R}^N)$ is successful after k reductions; thus, $\mathcal{C}_p^N(Q)$ is successful after $k+1$ reductions, and this is the minimum over all possible Q's. □

Denote the number of reductions put forward by Proposition 3.43 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 $\mathcal{C}_p^N(Q)$ is successful after $\textsc{Lb}(N, p)$ reductions if and only if there exist (q, ρ) and Q' such that $Q \xrightarrow{(\nu\widetilde{n})q} Q'$ and $p, \mathsf{id}_{\mathsf{bn}(p)} \ll q, \rho$. This fact is proved by Propositions 3.44 and 3.46.

Proposition 3.44. *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 Lemma 3.33 $\{p\|p'\} = (\sigma, \theta)$ where $\sigma = \mathsf{id}_{\mathsf{bn}(p)}$ and $\theta = \{n/x\}_{(x,n)\in F\cup R}$. By Proposition 3.18 $\{q\|p'\} = (\rho, \theta)$; 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 3.40. The thesis follows by Lemma 3.41. □

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 3.45. *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 unification, $q \in \{x, \ulcorner x \urcorner, \lambda y\}$, 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:
   (a) If $n \in N$, then $F = \{(x, n)\}$ and $R = \emptyset$. By definition of pattern unification, q must be communicable, $\rho = \{\}$ and $\theta = \{q/x\}$. Since $\mathsf{tests}^N_{F,R}$ only contains $\mathsf{free}(x, n)$, by Lemma 3.35 it holds that $q = \theta(x) = n$. This suffices to conclude, since $\mathsf{vn}(p) \setminus N = \mathsf{vn}(q) \setminus N = \emptyset$ and $n, \{\} \ll n, \{\}$.
   (b) 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 3.36 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 $n, \{\} \ll n, \{\}$.
3. If $p = \ulcorner n \urcorner$, then $p' = \ulcorner n \urcorner$ and $F = R = \emptyset$. By definition of pattern unification, $q \in \{n, \ulcorner n \urcorner\}$ and $\rho = \theta = \{\}$. In any case, $\mathsf{vn}(p) \setminus N = \mathsf{vn}(q) \setminus N = \emptyset$ and $p, \{\} \ll q, \{\}$.

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 unification, there are two possibilities for q:

1. If $q = \lambda z$, for some z, then p' must be communicable and $\theta = \{\}$ and $\rho = \{p'/z\}$. If p' is communicable then by definition of specification $\mathsf{vn}(p) = \emptyset = \mathsf{vn}(q)$ and $\mathsf{vn}(p) \setminus N = \mathsf{vn}(q) \setminus N = \emptyset$ and conclude with $p, \mathsf{id}_{\mathsf{bn}(p)} \ll \lambda z, \rho$.
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 show that every name $m \in W_i$ is in the domain of ζ_i and that $\zeta_i(m) \in V_i$. Further, that if m is in only one of W_i then $\zeta_i(m)$ only appears in the corresponding V_i, alternatively if m is in both W_1 and W_2 then $\zeta_1(m) = \zeta_2(m) = n$ for some $n \in V_1 \cap V_2$.

(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 3.37). 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 3.37). 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. □

Proposition 3.46. *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 $\mathcal{C}_p^N(Q)$ is successful after $\textsc{Lb}(N, p)$ reduction steps, then there exist (q, ρ) and Q' such that $Q \xrightarrow{(\nu\widetilde{n})q} Q'$ and $p, \mathsf{id}_{\mathsf{bn}(p)} \ll q, \rho$.*

Proof: By Proposition 3.43, there must be a reduction $\mathcal{C}_p^N(Q) \longmapsto (\nu\widetilde{m})(\theta(\mathsf{tests}^N_{F,R}) \mid \rho Q'')$ obtained because $Q \xrightarrow{(\nu\widetilde{m})q'} Q''$ and $\{p' \| q'\} = (\theta, \rho)$. Since $w \notin \mathsf{fn}(Q, p')$ and $\mathcal{C}_p^N(Q) \Downarrow_w$, it must be that $\theta(\mathsf{tests}^N_{F,R})$ succeeds; by Proposition 3.43, 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 3.45 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 3.47. *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 3.30, it suffices to prove that

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

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 2.6 (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 α-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

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

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

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

  The first reduction is obtained by unifying $\ulcorner w \urcorner \bullet \widetilde{m}$ 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 Proposition 3.44 (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 2.5 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 3.31. The last reduction is obtained by unifying $\ulcorner w \urcorner \bullet \widetilde{n}$ 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, $\mathcal{C}((\nu\widetilde{m})(Q|\ulcorner w\urcorner \bullet \widetilde{m}|Z)) \longmapsto^{\text{LB}(N,p)+2} \hat{Q}$ such that $\hat{P} \simeq \hat{Q}$. As $\hat{P}$ has a barb containing w', so must $\hat{Q}$; by definition of $\mathcal{C}(\cdot)$, this can happen only if $\mathcal{C}_p^N(Q)$ is successful after $\text{LB}(N,p)$ steps. By Proposition 3.46, this entails that there exist (q,ρ) and Q' 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 $\hat{Q} = (\nu\widetilde{n} \cup \widetilde{m})(\sigma[\rho](Q') \mid \ulcorner w'\urcorner \bullet \widetilde{n} \bullet \widetilde{m} \mid Z|\sigma Z')$; indeed, in this case the application of Proposition 3.44 yields $\mathcal{C}_p^N(Q) \longmapsto^{\text{LB}(N,p)} (\nu\widetilde{n})(\rho Q'|\ulcorner w\urcorner \bullet \widetilde{n}|Z')$.

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 3.14 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 $(q, \sigma[\rho])$ and Q' is a proper reply to the challenge $P \xrightarrow{(\nu\widetilde{n})p} P'$ together with σ.

Closure under substitution holds by definition of $\Re$. □

Theorem 3.48 (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 straightforward 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 Propositions 3.44 and 3.15, $\mathcal{C}_p^N(P)$ is successful after k reduction steps, where $k = \text{LB}(N,p)$. It follows by barbed congruence that $\mathcal{C}_p^N(Q)$ is successful after k reduction steps too; Proposition 3.46 then implies that $Q \xrightarrow{(\nu\widetilde{n})q} Q'$ for some (q,ρ') and Q' such that $p, \mathsf{id}_{\mathsf{bn}(p)} \ll q, \rho'$.

By two applications of Proposition 3.44 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 Proposition 3.43 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 3.47 imply that $P' \simeq \rho' Q'$. By Lemma 3.31, we obtain $\sigma P' \simeq \sigma(\rho' Q') = \sigma[\rho'](Q')$. By Lemma 3.14, $p, \mathsf{id}_{\mathsf{bn}(p)} \ll q, \rho'$ implies $p, \sigma \ll q, \sigma[\rho']$. This suffices to conclude. □

To conclude, we want to stress that we have developed our semantic theories in the *strong* setting just for the sake of simplicity. Their *weak* counterparts, consisting of allowing multiple τs/reductions in every predicate, can be obtained in the usual manner [34, 35]. As usual, the main difficulty is in the completeness proof just shown. Indeed, to force a proper reply via contexts, we need to use "fresh" barbs (viz, the ws and f in our tests). Such barbs have to be removed after the forced action; however, this can be done only if the left and right hand side of the equated processes have the same shape (see Lemma 3.47). This is straightforward in the strong case, where the number of reductions (viz, $\text{LB}(N,p)$ – see Propositions 3.44 and 3.46) ensures this property. By contrast, in the weak case we can stop along this sequence of $\text{LB}(N,p)$ reductions, since the weak barbs will cover the missing steps. This requires a different proof of

Lemma 3.47. To achieve this, we can follow the traditional path: instead of having a single success barb w, our tests have two (say, w and w') in mutual exclusion (i.e., internal choice); we then consider an additional reduction that excludes w'. This ensures that the other process must also reach the reduction that excludes w' and thus complete the proof.

3.7 Example Equivalences

This section considers some examples where bisimulation can be used to show the 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:

Proposition 3.49. *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.

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 $\mathsf{bn}(q) \cap \mathsf{fn}(Q|R) = \emptyset$ (ensured by 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 3.14, $p, \sigma \ll q, \sigma[\rho]$,

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 3.25, $\sigma P'|Q|R \sim \sigma[\rho]Q'|Q|R$. 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 3.15, $q, \sigma \ll q, \sigma$. 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 straightforward.

(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 Proposition 3.19, $\{q\|q\} = (\vartheta[\rho], \theta)$. But a pattern can unify with itself only if it contains no binding names; this entails that $\theta = \{\}$ and $\vartheta = \{\}$. 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 Proposition 3.19, 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) $\hat{P} = p \to P' \mid (\nu\widetilde{n})(\vartheta(Q'|Q)|\theta R')$, 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) $\hat{P} = p \to P' \mid \vartheta Q'|\theta Q'|Q|R$, where $\{q\|q\} = (\vartheta, \theta)$: this case is straightforward, by observing that $\vartheta = \theta = \{\}$.

Closure under substitution is straightforward by Proposition 3.16. □

To conclude, notice that the more general claim

> Let $P = p \to P' \mid !q \to Q'$ and $Q = !q \to Q'$; 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:

$$\begin{array}{lll} P = \lambda x \to P' \mid Q & \text{with} & P' = x \mid m \to \ulcorner w \urcorner \quad \text{for } x \neq m \\ Q = !\lambda x \to Q' & \text{with} & Q' = m \mid m \to \ulcorner w \urcorner \end{array}$$

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 $\mathcal{C}(\cdot) = \cdot \mid k \to \mathbf{0}$, 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. (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 3.30 and 3.48.) Proposition 3.49 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 [19, 21] to formally assess the expressive power of CPC with respect to π-calculus, Linda, Spi calculus, Fusion and Psi calculus. After briefly recalling these models and some basic material from [21], 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 a *valid* encoding is in Definition 4.1.

4.1 Some Process Calculi

π-calculus [34, 41]. The π-calculus processes are given by the following grammar:

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

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 congruence relation defined for CPC.

Linda [15]. Consider the following variant of Linda formulated to follow CPC's syntax. Processes are defined as:

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

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

$$t \ ::= \ \lambda x \quad \mid \ \ulcorner b \urcorner$$

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 congruence relation defined for CPC.

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

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

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. Note that i denotes a natural number greater than zero, and is considered equal to the ith successor of zero.

The processes of the Spi calculus are:

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

The null process, parallel composition, replication and restriction are all familiar. The input $M(x).P$ and output $\overline{M}\langle N \rangle.P$ are generalised from π-calculus to allow arbitrary terms in the place of channel names and output arguments. The match $[M \ \mathit{is}\ N]P$ determines equality of M and N. The splitting $\mathit{let}\ (x, y) = M\ \mathit{in}\ P$ decomposes pairs. The decryption $\mathit{case}\ M\ \mathit{of}\ \{x\}_N : P$ decrypts M and binds the encrypted message to x. The integer test $\mathit{case}\ M\ \mathit{of}\ 0 : P\ \mathit{suc}(x) : Q$ 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:

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

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

Fusion [37]. 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 taken from [45]:

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

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 congruence of CPC.

Psi [7]. 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 [7]. 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

This section recalls and adapts the definition of valid encodings as well as some useful theorems (details in [21]) for formally relating process calculi. The validity of such criteria in developing expressiveness studies emerges from the various works [19, 20, 21], that have also recently inspired similar works [29, 30, 44].

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 name (of the source language) into a tuple of k names (of the target language), for $k > 0$. The translation $\llbracket\cdot\rrbracket$ turns every term of the source language into a term of the target; 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 $\varphi_{\llbracket\rrbracket}$.

Now consider only encodings that satisfy the following properties. 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 one of the k holes 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 $\llbracket S\rrbracket \Longmapsto_2 T$, there exists S' such that $S \Longmapsto_1 S'$ and $T \Longmapsto_2 \simeq_2 \llbracket S'\rrbracket$.*

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

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

The criteria we have just presented may seem quite demanding; for example, they do not allow schemes including parameters that are changed along the way of the encoding (the most notable of such examples is Milner's encoding of

λ-calculus into π-calculus [32]). Of course, this makes our separation results slightly weaker and leaves room for further improvement. Moreover, as fully explained in [22], we do not consider full abstraction as a validity criterion for expressiveness.

Now recall some results concerning valid encodings, in particular 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. Originally valid encodings considered were 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.

Theorem 4.2. *Assume that there exists S such that $S \mapsto\!\!\!\!\!\!\rightarrow_1$ and $S \Downarrow\!\!\!\!\!\!\downarrow_1$ and $S \mid S \Downarrow_1$; moreover, assume that every T that does not reduce is such that $T \mid T \mapsto\!\!\!\!\!\!\rightarrow_2$. 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 4.3. *If $\textsc{Md}(\mathcal{L}_1) > \textsc{Md}(\mathcal{L}_2)$, then there exists no valid encoding of $\mathcal{L}_1$ into $\mathcal{L}_2$.*

The previous proof techniques can be directly used in some cases: for example, we shall prove that CPC cannot be encoded in any other sample calculus by exploiting Theorem 4.2. However, not all separation results can be obtained as corollaries of such results. The following technique provides a very useful tool when the theorems above cannot be applied.

Proposition 4.4. *Let $[\![\cdot]\!]$ be a valid encoding; then, $S \mapsto\!\!\!\!\!\!\rightarrow_1$ implies that $[\![S]\!] \mapsto\!\!\!\!\!\!\rightarrow_2$.*

The way in which we shall use this technique is the following. To prove that $\mathcal{L}_1$ cannot be encoded into $\mathcal{L}_2$ we reason by contradiction and assume a valid encoding $[\![\cdot]\!]$. Then, we pick an $\mathcal{L}_1$-process P that reduces; we first show that this implies that $[\![P]\!]$ also reduces. We then analyze how the latter reduction may have happened and, for every possible case, show a process P' obtained from P (usually, by just swapping two names) such that $[\![P']\!]$ reduces whereas P' does not. This contradicts Proposition 4.4 and allows us to conclude that no valid encoding exists.

4.3 CPC vs π-calculus and Linda

A hierarchy of sets of process calculi with different communication primitives is obtained in [19] 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 [15] (called $L_{\text{A,P,D,PM}}$ in [19]) is more expressive than monadic/polyadic π-calculus [34, 33] (called $L_{\text{S,M,C,NO}}$ and $L_{\text{S,P,C,NO}}$, respectively, in [19]).

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 4.3: 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 proved by induction on the structure of T. Conclude by Theorem 4.2. □

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 unification). 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 4.4.

Next is to prove that this encoding is valid. This is a corollary of the following lemma, stating a strict correspondence between Linda's pattern matching and CPC's unification (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 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 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 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 unification in CPC, $\mathsf{pat\text{–}d}(\widetilde{b})$ and $\mathsf{pat\text{–}t}(\widetilde{t})$ cannot unify, and this would contradict the hypothesis. Thus, it must be that $\widetilde{b}$ is the empty sequence and we 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 unify with $\mathsf{pat\text{–}t}(\widetilde{t})$. 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: Straightforward, 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 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 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 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

CPC cannot be encoded into Spi calculus, as a corollary of Theorem 4.2. This can be proved as in Theorem 4.5: the self-unifyinging 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, suc, and the natural numbers > 0 ranged over by i with

$$\begin{array}{llll}
[\![\,n\,]\!] \stackrel{\text{def}}{=} n & & [\![\,suc(M)\,]\!] \stackrel{\text{def}}{=} \mathsf{suc} \bullet [\![\,M\,]\!] \\
[\![\,x\,]\!] \stackrel{\text{def}}{=} x & & [\![\,(M,N)\,]\!] \stackrel{\text{def}}{=} \mathsf{pair} \bullet [\![\,M\,]\!] \bullet [\![\,N\,]\!] \\
[\![\,0\,]\!] \stackrel{\text{def}}{=} 0 & & [\![\,\{M\}_N\,]\!] \stackrel{\text{def}}{=} \mathsf{encr} \bullet [\![\,M\,]\!] \bullet [\![\,N\,]\!] \\
[\![\,i\,]\!] \stackrel{\text{def}}{=} \mathsf{suc}^i 0 & &
\end{array}$$

where $\mathsf{suc}^i 0$ denotes suc compounded i times with 0. 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:

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

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

The four remaining process forms all require pattern unification and so translate to cases in parallel. In each encoding a fresh name n is used to prevent interaction with other processes, see Proposition 2.6. As in the Spi calculus, the encodings will reduce only after a successful unification and will be stuck otherwise. The encodings are

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

The unification $[M \ is \ N]P$ 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 unifies with 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 unify 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: Straightforward, from the fact that $\equiv$ acts only on operators that $[\![\, \cdot \,]\!]$ translates homomorphically. □

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

Proof: The first claim can be 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,

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

and it can only reduce to

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

By a straightforward induction on the structure of P_2 proves that $\{[\![M]\!]/x\}[\![P_2]\!] = [\![\{M/x\}P_2]\!]$. Thus, $[\![P]\!] \longmapsto [\![\{M/x\}P_2]\!] \mid (\nu n)\ulcorner n \urcorner \bullet \ulcorner 0 \urcorner \to [\![P_1]\!] \simeq [\![P']\!]$, where the last equivalence follows from Proposition 2.6. 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 $[\![P]\!] = p \to Q_1 \mid q \to Q_2$ 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 congruence: 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 $[\![N]\!]$ unify; 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 $case \ \{M\}_N \ of \ \{x\}_N : P_1$.
  - $Q' = \ulcorner n \urcorner \bullet \ulcorner 0 \urcorner \to [\![P_1]\!] \mid \ulcorner n \urcorner \bullet (\ulcorner \mathsf{suc} \urcorner \bullet \lambda x) \to [\![P_2]\!] \mid \ulcorner n \urcorner \bullet [\![M]\!]$. 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 $[\![M]\!]$).
    - $[\![M]\!] = 0$: in this case, $Q'' = [\![P_1]\!] \mid \ulcorner n \urcorner \bullet (\ulcorner \mathsf{suc} \urcorner \bullet \lambda x) \to [\![P_2]\!]$ and so $Q = (\nu n)Q'' \equiv [\![P_1]\!] \mid (\nu n)\ulcorner n \urcorner \bullet (\ulcorner \mathsf{suc} \urcorner \bullet \lambda x) \to [\![P_2]\!] \simeq [\![P_1]\!]$. In this case, $M = 0$ and so $P \longmapsto P_1$; to conclude, it suffices to let P' be P_1.
    - $[\![M]\!] = \mathsf{suc} \bullet [\![M']\!]$, for some M': in this case, $Q'' = \{[\![M']\!]/x\}[\![P_2]\!] \mid \ulcorner n \urcorner \bullet \ulcorner 0 \urcorner \to [\![P_1]\!]$ and so $Q = (\nu n)Q'' \equiv [\![\{M'/x\}P_2]\!] \mid (\nu n)\ulcorner n \urcorner \bullet 0 \to [\![P_1]\!] \simeq [\![\{M'/x\}P_2]\!]$. 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 [19, 21]). 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 unify with 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 [16].

4.5 CPC vs Fusion

The separation results for CPC and the other process calculi presented so far have all been proved via symmetry; thus, 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 either the matching degree or the 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 4.3. Alternatively, reuse the proof for Theorem 4.5 as every Fusion process T is such that $T \mid T \longmapsto$ implies $T \longmapsto$. □

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:

$$[\![(\nu x)(\overline{u}\langle x\rangle \mid y(u).\surd)]\!] \tag{4}$$

that has $u(y)$ replaced with $y(u)$; or

$$[\![(\nu x)(\overline{x}\langle u\rangle \mid u(y).\surd)]\!] \tag{5}$$

that has $\overline{u}\langle x\rangle$ replaced with $\overline{x}\langle u\rangle$, would report success, whereas both Equation 4$\not\Downarrow$ and Equation 5$\not\Downarrow$, 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 contradiction of the 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 $[\![\text{Equation 4}]\!]$ or $[\![\text{Equation 5}]\!]$ would reduce; however, Equation 4 $\not\longmapsto$ and Equation 5 $\not\longmapsto$, against Proposition 4.4 (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, $[\![\text{Equation 4}]\!] \longmapsto$ whereas Equation 4 $\not\longmapsto$, against Proposition 4.4.

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 $[\![\text{Equation 5}]\!] \longmapsto$ whereas Equation 5 $\not\longmapsto$, against Proposition 4.4.

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. □

4.6 CPC vs Psi

CPC and Psi are *unrelated*, in the sense that there exists no valid encoding from one into the other. As in Theorem 4.5, the impossibility for a valid encoding of CPC into Psi can be proved by exploiting the symmetry of CPC. The converse separation result is ensured by the following theorem.

Theorem 4.15. *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 $P \stackrel{\text{def}}{=} (\bar{a}.c \mid b.(\surd \mid c)) \mid (\!|a \leftrightarrows b|\!)$, where we have omitted the argument of the actions to simplify the proofs, and chosen a, b and c pairwise distinct; also consider the reduction

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

Thus, $P \Downarrow$ and, by success sensitiveness, $[\![P]\!] \Downarrow$. By compositionality, $[\![P]\!] \stackrel{\text{def}}{=} \mathcal{C}^{\{a,b,c\}}_{|}(\mathcal{C}^{\{a,b,c\}}_{|}([\![\bar{a}.c]\!]; [\![b.(\surd \mid c)]\!]); [\![(\!|a \leftrightarrows b|\!)]\!])$. Like in the proof of Theorem 4.14, it can be proven 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 contradiction of the 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 $\mathcal{C}_{|}^{\{a,b,c\}} \longmapsto$ or the reduction is obtained by synchronizing the two copies of $\mathcal{C}_{|}^{\{a,b,c\}}$. In both cases, $[\![\,(\bar{c}.a \mid b.(\surd \mid c)) \mid (\!|\,a \dot{\leftrightarrow} b\,|\!)\,]\!]$ (with $\bar{a}.c$ replaced by $\bar{c}.a$) would also reduce, whereas $(\bar{c}.a \mid b.(\surd \mid c)) \mid (\!|\,a \dot{\leftrightarrow} b\,|\!) \not\longmapsto$, against Proposition 4.4 (that must hold whenever $[\![\cdot]\!]$ is valid).

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

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

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. Further, the pattern-matching bisimulation developed for CPC can easily account for other kinds of pattern-matching, such as in polyadic π-calculus and Linda [17].

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. Similarly, Psi calculus is unrelated to CPC due to supporting implicit computations, while also being less symmetric.

Another path of development for a process calculus is implementation in a programming language [39, 9, 2, 27]. The **bondi** programming language is based upon pattern matching as the core of reduction and the theory of pattern calculus [24, 1]. A Concurrent **bondi** has also been developed that extends **bondi** with concurrency and interaction based on the pattern unification and theory of CPC [16, 11].

Acknowledgments We would like to thank the anonymous reviewers for their fruitful comments and for their constructive attitude towards our paper.

Appendix A: Proofs of Section 3.2

Proof of Proposition 3.7 First of all, let us define an alternative (but equivalent, up-to $\equiv$) LTS for CPC, written $\xrightarrow{\mu}\!\!\!\!\rightarrow$: it is obtained by replacing rep with the following two rules (all the other rules are the same, with $\rightarrow\!\!\!\!\rightarrow$ in place of $\rightarrow$ everywhere):

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

We can prove that: (1) if $P \xrightarrow{\mu}\!\!\!\!\rightarrow P'$ then $P \xrightarrow{\mu} P'$; and (2) if $P \xrightarrow{\mu} P'$ then $P \xrightarrow{\mu}\!\!\!\!\rightarrow P''$, for some $P'' \equiv P'$ (both proofs are done by a straightforward induction on the derivation 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 $P \xrightarrow{\mu} P'$. □

Proof of Lemma 3.8 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 $\widetilde{n} = \emptyset$ and $Q_1 = P_1$ 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 $\widetilde{n}'$ and Q_1'

and Q_2' such that $P_1 \equiv (\nu\widetilde{m})(\nu\widetilde{n}')(p \to Q_1' \mid Q_2')$ and $P_1' \equiv (\nu\widetilde{n}')(Q_1' \mid Q_2')$ and $\widetilde{n}' \cap \mathsf{names}((\nu\widetilde{m})p) = \emptyset$ 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 $Q_1 = Q_1'$ and $Q_2 = Q_2'$ and $\widetilde{n} = \widetilde{n}', o$.

- If the last rule is open then $P = (\nu o)P_1 \xrightarrow{(\nu\widetilde{m}', o)p} P_1' = P'$, where $P_1 \xrightarrow{(\nu\widetilde{m}')p} P_1'$ and $o \in \mathsf{vn}(p) \backslash (\widetilde{m}' \cup \mathsf{pn}(p) \cup \mathsf{bn}(p))$ and $\widetilde{m} = \widetilde{m}', o$. By induction, there exist $\widetilde{n}'$ and Q_1' and Q_2' such that $P_1 \equiv (\nu\widetilde{m}')(\nu\widetilde{n}')(p \to Q_1' \mid Q_2')$ and $P_1' \equiv (\nu\widetilde{n}')(Q_1' \mid Q_2')$ and $\widetilde{n}' \cap \mathsf{names}((\nu\widetilde{m}')p) = \emptyset$ and $\mathsf{bn}(p) \cap \mathsf{fn}(Q_2') = \emptyset$. Conclude with $\widetilde{n} = \widetilde{n}'$ and $Q_1 = Q_1'$ 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 $\widetilde{n}'$ and Q_1' and Q_2' such that $P_1 \equiv (\nu\widetilde{m})(\nu\widetilde{n}')(p \to Q_1' \mid Q_2')$ and $P_1' \equiv (\nu\widetilde{n}')(Q_1' \mid Q_2')$ and $\widetilde{n}' \cap \mathsf{names}((\nu\widetilde{m})p) = \emptyset$ and $\mathsf{bn}(p) \cap \mathsf{fn}(Q_2') = \emptyset$. As $\mathsf{bn}(p) \cap \mathsf{fn}(P_2) = \emptyset$, we can conclude with $\widetilde{n} = \widetilde{n}'$ and $Q_1 = Q_1'$ 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 conclude by induction and by the fact that $P \equiv Q \mid !Q$. □

Proof of Proposition 3.9 The first claim is proved by induction on the inference for $P \xrightarrow{\tau} P'$. The base case is with rule unify: $P = P_1 \mid Q_1$, where $P_1 \xrightarrow{(\nu\widetilde{m})p} P_1'$ and $Q_1 \xrightarrow{(\nu\widetilde{n})q} Q_1'$ and $P' = (\nu\widetilde{m} \cup \widetilde{n})(\sigma P_1' \mid \rho Q_1')$ and $\{p\|q\} = (\sigma, \rho)$ and $\widetilde{m} \cap \mathsf{fn}(Q_1) = \widetilde{n} \cap \mathsf{fn}(P_1) = \emptyset$ 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 $\widetilde{o}$ and $\widetilde{r}$, we have $(\widetilde{o} \cup \widetilde{r}) \cap (\mathsf{names}((\nu\widetilde{m})p) \cup \mathsf{names}((\nu\widetilde{n})q)) = \emptyset$; thus, $P_1 \mid Q_1 \equiv (\nu\widetilde{m} \cup \widetilde{n})(\nu\widetilde{o} \cup \widetilde{r})(p \to P_1'' \mid P_2'' \mid q \to Q_1'' \mid Q_2'') \longmapsto (\nu\widetilde{m} \cup \widetilde{n})(\nu\widetilde{o} \cup \widetilde{r})(\sigma P_1'' \mid P_2'' \mid \rho Q_1'' \mid Q_2'')$. Since σ avoids $\widetilde{o}$, $\mathsf{dom}(\sigma) \cap \mathsf{fn}(P_2'') = \emptyset$ and 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 $P \longmapsto (\nu\widetilde{m} \cup \widetilde{n})(\nu\widetilde{o} \cup \widetilde{r})(\sigma P_1'' \mid P_2'' \mid \rho Q_1'' \mid Q_2'') \equiv (\nu\widetilde{m} \cup \widetilde{n})(\sigma((\nu\widetilde{o})(P_1'' \mid P_2'')) \mid \rho((\nu\widetilde{r})(Q_1'' \mid Q_2''))) \equiv (\nu\widetilde{m} \cup \widetilde{n})(\sigma P_1' \mid \rho Q_1') = P'$.

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, conclude 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 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

unify 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 base cases are easier, as is 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.
    - $P_1 \xrightarrow{(\nu \widetilde{m})p} P_1'$ and $P_2 \xrightarrow{(\nu \widetilde{n})q} P_2'$, and $Q''' = (\nu \widetilde{m} \cup \widetilde{n})(\sigma P_1' \mid \rho P_2')$, 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. □

Appendix B: Proofs of Section 3.5

Proof of Lemma 3.23 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 3.12). Indeed, $p, \sigma \ll p, \sigma$, by Proposition 3.15, and $(\sigma P, \sigma Q) \in \Re$, because $P \sim Q$ and $\sim$ is closed under substitutions by definition. Closure under substitution holds by definition of $\Re$. □

Proof of Lemmata 3.24 and 3.25 The two lemmata have to be proved together; as in π-calculus, this is necessary because of name extrusion. We can conclude if we show 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 $\mu = (\nu\widetilde{m})\bar{\mu}$ and $\hat{P} = (\nu\ \widetilde{n}\backslash\widetilde{m})\bar{P}$ and the dots denote repeated applications of resnon (one for every name in $\widetilde{n} \setminus \widetilde{m}$) and open (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
$$\dfrac{P \xrightarrow{(\nu\widetilde{l})p} P' \quad R \xrightarrow{(\nu\widetilde{o})r} R'}{P \mid R \xrightarrow{\tau} (\nu\widetilde{l} \cup \widetilde{o})(\sigma P' \mid \theta R')}$$
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, ρ) and Q' such that $Q \xrightarrow{(\nu\widetilde{l})q} Q'$ and $p, \sigma \ll q, \rho$ and $\sigma P' \sim \rho Q'$. By Proposition 3.18, $\{q \| r\} = (\rho, \theta)$ and so
$$\dfrac{Q \xrightarrow{(\nu\widetilde{l})q} Q' \quad R \xrightarrow{(\nu\widetilde{o})r} R'}{Q \mid R \xrightarrow{\tau} (\nu\widetilde{l} \cup \widetilde{o})(\rho Q' \mid \theta R')}$$

where, by α-conversion, we can always let $\widetilde{o} \cap \mathsf{fn}(Q) = \emptyset$ (the other side conditions for applying rule unify already hold). By repeated applications of rule resnon, infer $(\nu\widetilde{n})(Q \mid R) \xrightarrow{\tau} (\nu\widetilde{n})(\nu\widetilde{l} \cup \widetilde{o})(\rho Q' \mid \theta R')$ and 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, ρ) and Q' such that $Q \xrightarrow{(\nu\widetilde{l})q} Q'$ and $p, \sigma \ll q, \rho$ 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 open used for $(\star)$ (this is possible since $\mathsf{fn}(p) = \mathsf{fn}(q)$, see Lemma 3.13), 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 open used for $(\star)$, we obtain $(\nu\widetilde{n})(Q \mid R) \xrightarrow{(\nu\widetilde{l}\cup\widetilde{m})p} (\nu\ \widetilde{n}\backslash\widetilde{m})(Q \mid R') = \hat{Q}$. By Proposition 3.15, $p, \sigma \ll p, \sigma$. 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.

Closure under substitution holds by definition of $\Re$. □

Proof of Lemma 3.26 This proof rephrases the similar one in [41]. First,

define the n-th approximation of the bisimulation:

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

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

$$\sim\ = \bigcap_{n \geq 0} \sim_n \tag{6}$$

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, ρ_n) and Q_n such that $Q \xrightarrow{(\nu \widetilde{m})q_n} Q_n$ and $p, \sigma \ll q_n, \rho_n$ and $\sigma P' \sim_n \rho_n Q_n$. However, by Proposition 3.20, 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

$$!P \sim_n P^{2n} \quad \text{and} \quad !Q \sim_n Q^{2n} \tag{7}$$

The proof is by induction on n and exploits a Lemma similar to Lemma 3.25 (with $\sim_n$ in place of $\sim$); the details are left to the interested reader. By repeatedly exploiting Lemma 3.25, it follows that $P^{2n} \sim Q^{2n}$ and so by (6)

$$P^{2n} \sim_n Q^{2n} \tag{8}$$

Now by (8) it follows that $P \sim Q$ implies that $P^{2n} \sim_n Q^{2n}$, for all n. By (7) and Lemma 3.21 (that also holds with $\sim_n$ in place of $\sim$), it follows that $!P \sim_n\ !Q$, for all n. By (6), conclude that $!P \sim\ !Q$. □

References

[1] bondi programming language. `http://bondi.it.uts.edu.au/`.

[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, K. Bhargavan, C. Fournet, A. D. Gordon, and S. Maffeis. Refinement types for secure implementations. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 33(2):8, 2011.

[7] J. Bengtson, M. Johansson, J. Parrow, and B. Victor. Psi-calculi: a framework for mobile processes with nominal data and logic. *Logical Methods in Computer Science*, 7(1), 2011.

[8] J. Bengtson and J. Parrow. Formalising the pi-calculus using nominal logic. *Logical Methods in Computer Science*, 5(2), 2009.

[9] 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.

[10] M. G. Buscemi and U. Montanari. Open bisimulation for the concurrent constraint pi-calculus. In *Proc. of ESOP*, volume 4960 of *LNCS*, pages 254–268. Springer, 2008.

[11] Concurrent bondi. Concurrent bondi, 2011. Retrieved 1 June 2013, from `http://www-staff.it.uts.edu.au/~tgwilson/concurrent_bondi/`.

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

[13] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[14] C. Fournet, A. D. Gordon, and S. Maffeis. A type discipline for authorization in distributed systems. In *In CSF '07: Proceedings of the 20th IEEE Computer Security Foundations Symposium*, pages 31–48. IEEE Computer Society, 2007.

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

[16] T. Given-Wilson. *Concurrent Pattern Unification*. PhD thesis, University of Technology, Sydney, Australia, 2012.

[17] T. Given-Wilson and D. Gorla. Pattern matching and bisimulation. In *Proc. of COORDINATION*, volume 7890 of *LNCS*, pages 60–74. Springer, 2013.

[18] T. Given-Wilson, D. Gorla, and B. Jay. Concurrent pattern calculus. In *Proc. of IFIP-TCS*, volume 323 of *IFIP Advances in Information and Communication Technology*, pages 244–258. Springer, 2010.

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

[20] D. Gorla. A taxonomy of process calculi for distribution and mobility. *Distributed Computing*, 23(4):273–299, 2010.

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

[22] D. Gorla and U. Nestmann. Full abstraction: history, myths and facts. To appear in *Mathematical Structures in Computer Science*.

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

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

[25] B. Jay and T. Given-Wilson. A combinatory account of internal structure. *Journal of Symbolic Logic*, 76(3):807–826, 2011.

[26] B. Jay and D. Kesner. First-class patterns. *Journal of Functional Programming*, 19(2):191–225, 2009.

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

[28] A. Khorsandiaghai. Implementing typed psi-calculi. Master's thesis, Uppsala University, Department of Information Technology, 2012.

[29] I. Lanese, J. A. Pérez, D. Sangiorgi, and A. Schmitt. On the expressiveness of polyadic and synchronous communication in higher-order process calculi. In *Proc. of ICALP*, volume 6199 of *LNCS*, pages 442–453. Springer, 2010.

[30] I. Lanese, C. Vaz, and C. Ferreira. On the expressive power of primitives for compensation handling. In *Proc. of ESOP*, volume 6012 of *LNCS*, pages 366–386. Springer, 2010.

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

[32] R. Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2(2):119–141, 1992.

[33] R. Milner. The polyadic π-calculus: A tutorial. In *Logic and Algebra of Specification*, volume 94 of *Series F*. NATO ASI, Springer, 1993.

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

[35] R. Milner and D. Sangiorgi. Barbed bisimulation. In *Proc. of ICALP*, volume 623 of *LNCS*, pages 685–695. Springer, 1992.

[36] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[37] 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.

[38] G. Picco, A. Murphy, and G.-C. Roman. LIME: Linda Meets Mobility. In *Proc. ICSE*, pages 368–377. ACM Press, 1999.

[39] B. C. Pierce and D. N. Turner. Pict: A programming language based on the pi-calculus. In *Poof, Language and Interaction: Essays in honour of Robin Milner*, pages 455–494. MIT Press, 1997.

[40] D. Sangiorgi. A theory of bisimulation for the pi-calculus. *Acta Informatica*, 33(1):69–97, 1996.

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

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

[43] C. Urban, S. Berghofer, and M. Norrish. Barendregt's variable convention in rule inductions. In *Proc. of CADE*, volume 4603 of *LNCS*, pages 35–50. Springer, 2007.

[44] R. J. van Glabbeek. Musings on encodings and expressiveness. In *Proc. of EXPRESS/SOS*, volume 89 of *EPTCS*, pages 81–98, 2012.

[45] L. Wischik and P. Gardner. Explicit fusions. *Theoretical Computer Science*, 340(3):606–630, 2005.