Pattern Matching and Bisimulation

Thomas Given-Wilson[1] Daniele Gorla[2]

[1] NICTA, Sydney, Australia[3] [2] Dip. di Informatica, "Sapienza" Università di Roma

Abstract. Concurrent Pattern Calculus (CPC) is a minimal calculus whose communication mechanism is based on a powerful form of symmetric pattern unification. However, the richness of patterns and their unification entails some flexibility in the challenge-reply game that underpins bisimulation. This leads to an ordering upon patterns that is used to define the valid replies to a given challenge. Such a theory can be smoothly adapted to accomplish other, less symmetric, forms of pattern matching (e.g. those of Linda, polyadic π-calculus, and π-calculus with polyadic synchronization) without compromising the coincidence of the two equivalences.

1 Introduction

Concurrent Pattern Calculus [20] is a minimal process calculus that uses symmetric pattern unification as the basis of communication. CPC's expressive power is obtained by extending the messages sent during *interaction* from traditional *names* to a class of *patterns* that are *unified* in an *intensional* manner (i.e., inspecting their internal structure). This unification supports equality testing and bi-directional communication in an atomic step.

The exploration of intensionality in the concurrent setting is inspired by the increased expressive power that the intensional SF-calculus has over λ-calculus [23]. Since intensionality, as captured by pattern matching, is more expressive in sequential computation, it is natural to explore the expressiveness of intensionality, as captured by pattern unification, in concurrent computation. Indeed, CPC formally generalises both the sequential intensional computation of SF-calculus and the traditional (non-intensional) concurrent computation of π-calculus [18]. The expressive power of CPC is also testified to by the possibility of encoding some well-known process languages [20, 18]: π-calculus [26], Linda [16] and Spi-calculus [2]. CPC's symmetric form of communication has similarities to Fusion [28]; however, the two calculi are unrelated (neither one can be encoded in the other) [20, 18]. Finally, CPC has been implemented in [17].

The main features of CPC are illustrated in the following sample trade interaction:

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

$$\longmapsto (\nu\ sharesID)(\nu\ bankAcc)(\langle charge\ bankAcc\ for\ sale\rangle \mid \langle save\ sharesID\ as\ proof\rangle)$$

[3] NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

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 to charge (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 $\ulcorner ABCShares \urcorner \bullet sharesID \bullet \lambda x$ and $\ulcorner ABCShares \urcorner \bullet \lambda y \bullet bankAcc$ also specify the details of the shares 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: $(\nu\, accName)(\nu\, accNum)\ulcorner ABCShares \urcorner \bullet \lambda y \bullet (name \bullet accName \bullet number \bullet accNum)$. This more detailed buyer would still match against the seller, now yielding $\langle charge\ name \bullet accName \bullet number \bullet accNum\ for\ sale \rangle$. Indeed, the seller could also specify a desire to only accept bank account information that includes a name and number with the following pattern: $\ulcorner ABCShares \urcorner \bullet sharesID \bullet (\ulcorner name \urcorner \bullet \lambda a \bullet \ulcorner number \urcorner \bullet \lambda b)$ and continuation $\langle charge\ a\ b\ for\ sale \rangle$. This would also match with the detailed buyer information by unifying *name* with $\ulcorner name \urcorner$, *number* with $\ulcorner number \urcorner$, and binding *accName* and *accNum* to a and b respectively. The second seller exploits the intensionality of CPC to only interact with a buyer whose pattern is of the right structure (four subpatterns) and contains the right information (the protected names *name* and *number*, and shared information in the other two positions). CPC is built up around this rich form of pattern unification by using three standard operators taken from the π-calculus: name restriction, parallel composition and replication (not used in this simple example).

The focus of this paper is the investigation of the behavioural theory for CPC. As usual in concurrency theory, this is done by first defining a notion of barbed congruence and then capturing this via a labelled bisimulation-based equivalence. The main difficulty relies in the richness of the pattern unification mechanism adopted, that entails some flexibility in the challenge-reply game underlying the definition of the bisimulation. For example, the challenge $\lambda x \bullet \lambda y$ can be replied to by λz, because of the non-fully decomposing form of pattern matching. (Such as the seller who accepts anything as bank account information.) Indeed, every pattern matching the challenge has the form $p \bullet q$, where p and q are communicable (i.e., they do not contain protected names $\ulcorner n \urcorner$ nor binding names λw), yielding the substitution $\{p/x, q/y\}$. The same pattern also matches λz, now yielding the substitution $\{p \bullet q/z\}$. Of course, for $P \xrightarrow{\lambda x \bullet \lambda y} P'$ to be simulated by $Q \xrightarrow{\lambda z} Q'$, it must be that $\{p/x, q/y\}P'$ is bisimilar to $\{p \bullet q/z\}Q'$. Another subtlety is in the unification of shared information n with protected information $\ulcorner n \urcorner$. Since the latter is a request for the communicating party to also know this information, the two patterns unify. (Such as the more careful seller checking that the buyer provides *name* and *number* for a bank account.) These ideas are formalised via an ordering on patterns that characterises the valid replies to a given challenge: every pattern 'greater than' the challenge is a valid reply, provided that, by applying the resulting substitutions to the respective continuations, bisimilar processes are obtained.

The form of pattern unification adopted in CPC generalises other forms of pattern matching already presented in the literature. It is then desirable that CPC's theory and results can be adapted to such simpler forms. Section 4 shows that this job is rather straightforward for the form of pattern matching underlying Linda, for two simple extensions of Linda, for the polyadic π-calculus, and for the π-calculus with polyadic synchronization. This provides a complete behavioural theory for the languages adopting such forms of pattern matching. Moreover, for the π-calculus, the result coincides with the usual notions of barbed congruence and early bisimulation congruence; this can be seen as a confirmation of the validity of the theory presented here.

2 Concurrent Pattern Calculus

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

$$p \ ::= \ \lambda x \mid x \mid \ulcorner x \urcorner \mid p \bullet p$$

Binding names λx denote information sought by a trader; variable names x represent such information. Protected names $\ulcorner x \urcorner$ represent recognised information that cannot be traded. A compound $p \bullet q$ combines the two patterns p and q; compounds are left associative.

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

As protected names are limited to recognition and binding names are being sought, neither should be communicable to another process. Thus, a pattern is *communicable*, able to be traded to another process, if it contains no protected or binding names. Protection of a name can be extended to a communicable pattern p by defining $\ulcorner p \bullet q \urcorner = \ulcorner p \urcorner \bullet \ulcorner q \urcorner$.

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 $\mathsf{dom}(\sigma) \cup \mathsf{fn}(\sigma)$. Notationally, 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 $\theta(\sigma(x))$. For later convenience, define the identity substitution on a set of names X, written id_X: it maps every name in X to itself.

Substitutions are applied to patterns as follows:

$$\sigma x = \begin{cases} \sigma(x) & \text{if } x \in \mathsf{dom}(\sigma) \\ x & \text{otherwise} \end{cases} \qquad \sigma \ulcorner x \urcorner = \begin{cases} \ulcorner \sigma(x) \urcorner & \text{if } x \in \mathsf{dom}(\sigma) \\ \ulcorner x \urcorner & \text{otherwise} \end{cases}$$

$$\sigma(\lambda x) = \lambda x \qquad \sigma(p \bullet q) = (\sigma p) \bullet (\sigma q)$$

The *symmetric matching* (or *unification*) of two patterns p and q, written $\{p \parallel q\}$, attempts to unify p and q by generating substitutions for 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:

$$\begin{aligned}
\{x \parallel x\} = \{x \parallel \ulcorner x \urcorner\} = \{\ulcorner x \urcorner \parallel x\} = \{\ulcorner x \urcorner \parallel \ulcorner x \urcorner\} &\stackrel{\text{def}}{=} (\{\}, \{\}) \\
\{\lambda x \parallel q\} &\stackrel{\text{def}}{=} (\{q/x\}, \{\}) && \text{if } q \text{ is communicable} \\
\{p \parallel \lambda x\} &\stackrel{\text{def}}{=} (\{\}, \{p/x\}) && \text{if } p \text{ is communicable} \\
\{p_1 \bullet p_2 \parallel q_1 \bullet q_2\} &\stackrel{\text{def}}{=} (\sigma_1 \cup \sigma_2\,,\, \rho_1 \cup \rho_2) && \text{if } \{p_i \parallel q_i\} = (\sigma_i, \rho_i) \text{ for } i \in \{1, 2\}
\end{aligned}$$

Variable and protected names unify if they are the same name. A binding name unifies with any communicable pattern to produce a binding for its bound name. Two compounds unify if their corresponding components do; the resulting substitutions are given by taking the union of those produced by unifying the components (necessarily disjoint, as patterns are well-formed). Otherwise the patterns cannot be unified and the matching is undefined. Notice that pattern matching is deterministic because of left-associativity of compounds.

The processes of CPC are given by:

$$P \;::=\; \mathbf{0} \;\mid\; P \mid P \;\mid\; !P \;\mid\; (\nu x)P \;\mid\; p \to P$$

The null process $\mathbf{0}$ is the inactive process; $P \mid Q$ is the parallel composition of processes P and Q, allowing the two processes to evolve independently or by interacting; the replication $!P$ provides as many parallel copies of P as desired; $(\nu x)P$ declares a new name x, visible only within P and distinct from any other name. The traditional input and output primitives of process calculi are replaced by the *case*, viz. $p \to P$, that has a pattern p and a body P. A case with the null process as the body may also be written by only specifying the pattern. For later convenience, $\widetilde{n}$ denotes a collection of names $n_1, \ldots, n_i$; for example, $(\nu n_1)(\ldots((\nu n_i)P))$ will be written $(\nu\widetilde{n})P$.

The free names of processes, denoted $\mathsf{fn}(P)$, are defined as usual for all the traditional primitives and $\mathsf{fn}(p \to P) = \mathsf{fn}(p) \cup (\mathsf{fn}(P) \backslash \mathsf{bn}(p))$ for the case, where the binding names of the pattern bind their free occurrences in the body.

The *structural equivalence relation* $\equiv$ is defined just as in π-calculus [25]: it includes α-conversion and its defining axioms are:

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

$$(\nu n)\mathbf{0} \equiv \mathbf{0} \qquad (\nu n)(\nu m)P \equiv (\nu m)(\nu n)P \qquad P \mid (\nu n)Q \equiv (\nu n)(P \mid Q) \;\text{ if } n \notin \mathsf{fn}(P)$$

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

$$(p \to P) \mid (q \to Q) \;\longmapsto\; (\sigma P) \mid (\rho Q) \quad \text{if } \{p \parallel q\} = (\sigma, \rho)$$

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

CPC has one interaction axiom, stating that, if the unification of two patterns p and q is defined and generates (σ, ρ), the substitutions σ and ρ are applied to the bodies P and Q, respectively. If the matching of p and q is undefined then no interaction occurs. The interaction rule is then closed under parallel composition, restriction and structural equivalence in the usual manner.

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 definition of bisimulation for CPC. Some properties of patterns will be explored as a basis for showing coincidence of the semantics.

3.1 Barbed Congruence

The first crucial 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:

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

Now, 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 (the contexts in Definition 9 use two kinds of barbs, to define success and failure). Thus, like in CCS and π-calculus [27], barbs must be indexed, e.g. on some names that give an abstract account of the matching capabilities of the process. Because of the rich form of interactions, CPC barbs also include the set of names that *may* be tested for equality in an interaction, not just those that *must* be equal.

Definition 1 (Barb). *Let $P \downarrow_{\widetilde{m}}$ mean that $P \equiv (\nu \widetilde{n})(p \to P' \mid P'')$ for some $\widetilde{n}, p, P'$ and P'' such that $\mathsf{pn}(p) \cap \widetilde{n} = \emptyset$ and $\widetilde{m} = \mathsf{fn}(p) \backslash \widetilde{n}$.*

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

Definition 2 (Barb preservation). *$\mathfrak{R}$ is barb preserving iff, for every $(P, Q) \in \mathfrak{R}$, it holds that $P \downarrow_{\widetilde{m}}$ implies $Q \downarrow_{\widetilde{m}}$.*

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

$$\textsf{resin}: \quad \frac{P \xrightarrow{(\nu \widetilde{m})p} P'}{(\nu m)P \xrightarrow{(\nu \widetilde{m}, m)p} P'} \quad m \in \textsf{vn}(p) \setminus (\widetilde{n} \cup \textsf{pn}(p) \cup \textsf{bn}(p)) \qquad\qquad \textsf{rep}: \quad \frac{!P \mid P \xrightarrow{\mu} P'}{!P \xrightarrow{\mu} P'}$$

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

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

Fig. 1. LTS (the symmetric version of parint and parext have been omitted)

Definition 3 (Reduction closure). $\mathfrak{R}$ *is reduction closed iff, for every* $(P, Q) \in \mathfrak{R}$*, it holds that* $P \longmapsto P'$ *implies* $Q \longmapsto Q'$*, for some* Q' *such that* $(P', Q') \in \mathfrak{R}$*.*

Definition 4 (Context closure). $\mathfrak{R}$ *is context closed iff, for every* $(P, Q) \in \mathfrak{R}$ *and for every context* $C(\cdot)$*, it holds that* $(C(P), C(Q)) \in \mathfrak{R}$*.*

Definition 5 (Barbed congruence). *Barbed congruence,* $\simeq$*, is the largest symmetric, barb preserving, reduction and context closed binary relation on processes.*

Barbed congruence equates processes with the same behaviour, as captured by barbs: two equivalent processes must exhibit the same behaviours, and this property should hold along every sequence of reductions and in every execution context. This defines the *strong* version of barbed congruence; its *weak* counterpart can be obtained in the usual manner [26, 27], with more complex contexts for proving the completeness theorem.

The problem in proving (strong/weak) barbed congruence is its closure under any context. As is typical we solve this by giving an easier to reason about coinductive (bisimulation-based) characterization using an alternate operation semantics; an LTS.

3.2 Labelled Transition System

The following is an adaption of the standard late LTS for the π-calculus [26]. *Labels* are defined as follows:

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

Labels are used in *transitions* $P \xrightarrow{\mu} P'$ between processes, whose defining rules are given in Figure 1. Rule case states that a case's pattern can be used to interact with external processes. Rule resnon is used when a restricted name does not appear in the names of the label: it simply maintains the restriction on the process after the transition.

By contrast, rule resin is used when a restricted name occurs in the label: as the restricted name is going to be shared with other processes, the restriction is moved from the process to the label (this is called *extrusion*, by using a π-calculus terminology). Of course an extruded name cannot already be restricted, cannot be protected (as this would prevent interaction), and cannot be a binding name. Rule match defines when two processes can interact to perform an internal action: this can occur whenever the processes exhibit labels with unifiable patterns and with no possibility of clash or capture due to restricted names. Rule rep unfolds the replicated process to infer the action. Rule parint states that, if either process in a parallel composition can transition by an internal action, then the whole process can transition by an internal action. Rule parext is similar, but is used when the label is visible: when one of the processes in parallel exhibits an external action, then the whole composition exhibits the same external action, as long as the restricted or binding names of the label do not appear free in the parallel component that does not generate the label.

Note that α-conversion is always assumed to satisfy the side conditions whenever needed and the symmetric rules have been omitted for brevity.

The presentation of the LTS is concluded with the following two results. First, the LTS is structurally image finite, i.e. for every P and μ, there are finitely many $\equiv$-equivalence classes of μ-reducts of P (Proposition 1). Second, the τ's in the LTS induce the same operational semantics as the reductions (Proposition 2).

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

Proposition 2. *If $P \xrightarrow{\tau} P'$ then $P \longmapsto P'$. Conversely, if $P \longmapsto P'$ then there exists P'' such that $P \xrightarrow{\tau} P'' \equiv P'$.*

3.3 Bisimulation

The next step is to develop a *bisimulation* relation that equates processes with the same interactional behaviour as captured by the labels of the LTS. The complexity is that the labels for external actions contain patterns, and some patterns are 'more general' than others, in terms of their matching capabilities. Two examples can clarify the point.

Example 1. Consider the processes $P = \lambda x \bullet \lambda y \to x \bullet y$ and $Q = \lambda z \to z$. Every process that can interact with P (by exhibiting a pattern matching against $\lambda x \bullet \lambda y$) can interact with Q, but not vice versa: e.g., $n \to \mathbf{0}$ can interact with Q but not with P. In this sense, the pattern λz is considered 'more general' than $\lambda x \bullet \lambda y$.

Example 2. Consider the processes $P = \ulcorner n \urcorner \to \mathbf{0}$ and $Q = n \to \mathbf{0}$. Every process that can interact with P can interact with Q, but not vice versa: consider, e.g., $\lambda x \to \mathbf{0}$. Thus, the pattern n is considered 'more general' than $\ulcorner n \urcorner$.

Now define an order relation on patterns that can be used to develop the bisimulation. In most process calculi, a challenge is replied to with an identical action [26]. However, there are situations in which an exact reply would make the bisimulation equivalence too fine for characterising barbed congruence [3, 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, as here for CPC. To this aim, define $\hat{\sigma}$ as a normal substitution, except that it operates on binding names rather than on free ones. Formally:

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

Definition 6. *Let p, q, σ and ρ be such that $\mathsf{bn}(p) = \mathsf{dom}(\sigma)$ and $\mathsf{bn}(q) = \mathsf{dom}(\rho)$. Define inductively that p is* compatible *with q by σ and ρ, denoted $p, \sigma \ll q, \rho$, whenever:*

$$p, \sigma \ll \lambda y, \{\hat{\sigma}p/y\} \;\; \textit{if} \;\; \mathsf{fn}(p) = \emptyset \qquad\qquad n, \{\} \ll n, \{\}$$

$$\ulcorner n \urcorner, \{\} \ll \ulcorner n \urcorner, \{\} \qquad\qquad \ulcorner n \urcorner, \{\} \ll n, \{\}$$

$$p_1 \bullet p_2, \sigma_1 \cup \sigma_2 \;\ll\; q_1 \bullet q_2, \rho_1 \cup \rho_2 \;\; \textit{if } p_i, \sigma_i \ll q_i, \rho_i, \textit{ for } i \in \{1, 2\} \; .$$

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

Lemma 1. *$p, \sigma \ll q, \rho$ and $\{p \parallel r\} = (\sigma, \theta)$ implies $\{q \parallel r\} = (\rho, \theta)$.*

Moreover, compatibility preserves information used for barbs, is stable under substitution composition, is reflexive and transitive.

Proposition 3. *If $p, \sigma \ll q, \rho$ then $\mathsf{fn}(p) = \mathsf{fn}(q)$ and $\mathsf{fn}(\sigma) = \mathsf{fn}(\rho)$. Moreover, $\mathsf{vn}(p) \subseteq \mathsf{vn}(q)$ and $\mathsf{pn}(q) \subseteq \mathsf{pn}(p)$.*

Lemma 2. *If $p, \sigma \ll q, \rho$ then $p, \theta[\sigma] \ll q, \theta[\rho]$, for every θ.*

Proposition 4. *Given p and σ such that $\mathsf{dom}(\sigma) = \mathsf{bn}(p)$, then $p, \sigma \ll p, \sigma$.*

Proposition 5. *$p, \sigma \ll q, \rho$ and $q, \rho \ll r, \theta$ imply $p, \sigma \ll r, \theta$.*

Definition 7 (Bisimulation). *A symmetric binary relation on processes $\mathfrak{R}$ is a bisimulation if, for every $(P, Q) \in \mathfrak{R}$ and $P \xrightarrow{\mu} P'$, it holds that:*

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

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

The definition is inspired by the early bisimulation congruence for the π-calculus [26]: first of all, to be a congruence, we need to consider its closure under all possible substitutions (otherwise, it would not be closed under prefixes). Then, for every possible instantiation σ of the binding names, there exists a proper reply from Q. Of course, σ cannot be chosen arbitrarily: it cannot use in its range names that were restricted in P. Also the action μ cannot be arbitrary, as in the π-calculus: its restricted and binding names cannot occur free in Q.

Differently from the π-calculus, however, the reply from Q can be different from the challenge from P: this is due to the fact that CPC contexts are not powerful enough to enforce an identical reply (as highligthed in Examples 1 and 2). Indeed, this notion of bisimulation allows a challenge p to be replied to by any compatible q, provided that σ is properly adapted (yielding ρ, as described by the compatibility relation) before being applied to Q'. This feature somehow resembles the symbolic characterization of open bisimilarity given in [29, 6]. 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 [29]) or becomes the same once we apply the name identifications induced by the equality constraints (in [6]).

3.4 Soundness and Completeness of Bisimulation

Soundness is proved by showing that the bisimilarity relation is included in barbed congruence; this is done by showing that $\sim$ is an equivalence, it is barb preserving, reduction closed and context closed. All the details can be found in [19].

Theorem 1 (Soundness of bisimilarity). $\sim \subseteq \simeq$.

Completeness is proved by showing that barbed congruence is a bisimulation. First, is to show that barbed congruence is closed under substitutions.

Lemma 3. *If $P \simeq Q$ then $\sigma P \simeq \sigma Q$, for every σ.*

Second, is to show that, for any challenge, a proper reply can be yielded via closure under an appropriate 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.

First of all, define a notion of success and failure that can be reported. A fresh name w is used for reporting success, with a barb $\downarrow_w$ indicating success, and $\Downarrow_w$ indicating a reduction sequence that eventually reports success. Failure is handled similarly using the fresh name f. A process P *succeeds* if $P \Downarrow_w$ and $P \not\Downarrow_f$; P is *successful* if $P \equiv (\nu\widetilde{n})(\ulcorner w \urcorner \bullet p \mid P')$, for some $\widetilde{n}$, p and P' such that $w \notin \widetilde{n}$ and $P' \not\Downarrow_f$. *P becomes successful* if it can reduce to a successful process.

Now develop 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 by relying on Lemma 2. The context for forcing a proper reply is developed in three steps. The first step presents the *specification* of a pattern and a set of names N (to be thought of as the free names of the processes being compared for bisimilarity); this is the information required to build a reply context. The second step develops auxiliary processes to test specific components of a pattern, based on information from the specification. The third step combines these into a reply context that becomes successful if and only if it interacts with a process that exhibits a proper reply to the challenge. In what follows, we use the *first projection* $\mathsf{fst}(-)$ and *second projection* $\mathsf{snd}(-)$ of a set of pairs.

Definition 8. *The* specification $\mathsf{spec}^N(p)$ *of a pattern* p *with respect to a finite set of names* N *is defined as follows:*

$$\mathsf{spec}^N(\lambda x) = x, \{\}, \{\} \qquad \mathsf{spec}^N(\ulcorner n \urcorner) = \ulcorner n \urcorner, \{\}, \{\}$$

$$\mathsf{spec}^N(n) = \begin{cases} \lambda x, \{(x,n)\}, \{\} & \text{if } n \in N \text{ and } x \notin N \cup \{n\} \\ \lambda x, \{\}, \{(x,n)\} & \text{if } n \notin N \text{ and } x \notin N \cup \{n\} \end{cases}$$

$$\mathsf{spec}^N(p \bullet q) = p' \bullet q', F_p \uplus F_q, R_p \uplus R_q \text{ if } \begin{cases} \mathsf{spec}^N(p) = p', F_p, R_p \\ \mathsf{spec}^N(q) = q', F_q, R_q \end{cases}$$

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

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

From now on, 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 9. *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* $\widetilde{y} = \{y_1, \ldots, y_j\} = \mathsf{fst}(F) \cup \mathsf{fst}(R)$*.*

Although the details of the tests are omitted here (see [19] for details), their behaviour is described by the following Lemmas.

Lemma 4. *Let* θ *be such that* $\{n, w\} \cap \mathsf{dom}(\theta) = \emptyset$*; then,* $\theta(\mathsf{free}(x, n, w))$ *succeeds if and only if* $\theta(x) = n$*.*

Lemma 5. *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* $\theta(x) \in \mathcal{N} \setminus N$*.*

Lemma 6. *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* $\theta(x) = \theta(y)$*.*

Definition 10. *A* reply context $C_p^N(\cdot)$ *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:*

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

It can be proved (see [19]) that the minimum number of reductions required for $C_p^N(Q)$ to become successful (for any Q) is the number of reduction steps for $\theta(\mathsf{tests}_{F,R}^N)$ to become successful plus 1; this number only depends on N and p, i.e. not on θ. Denote this number as $\textsc{Lb}(N, p)$. The main feature of $C_p^N(\cdot)$ is described by the following key Lemma.

Lemma 7. *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* $Q \xrightarrow{(\nu\widetilde{m})q} Q'$ *and there exists* ρ *such that* $p, \mathsf{id}_{\mathsf{bn}(p)} \ll q, \rho$*, then* $C_p^N(Q) \longmapsto^k (\nu\widetilde{n})(\rho Q' \mid \ulcorner w \urcorner \bullet \widetilde{n} \mid Z)$*, where* $k = \textsc{Lb}(N, p)$ *and* $Z \simeq \mathbf{0}$*. Conversely, if* $C_p^N(Q)$ *becomes successful in* $\textsc{Lb}(N, p)$ *reduction steps, then there exist* q*,* Q' *and* ρ *such that* $Q \xrightarrow{(\nu\widetilde{m})q} Q'$ *and* $p, \mathsf{id}_{\mathsf{bn}(p)} \ll q, \rho$*.*

The last result needed for proving Theorem 2 is 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 8. *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* $P \simeq Q$*.*

Theorem 2 (Completeness of the bisimulation). $\simeq \subseteq \sim$.

4 On Variations of Pattern Matching

The form of pattern unification used so far in CPC is very rich. More limited forms of pattern matching have been used in the literature; as shown below, they can all be adopted in our language without compromising the coincidence of barbed congruence and bisimilarity.

The first variant is the form of pattern matching used in Linda [16]. Differently from CPC, Linda distinguishes between input and output patterns (the latter are usually called *tuples* in a *tuplespace*):

$$p ::= \pi \mid \varpi \qquad \pi ::= \lambda x \mid \ulcorner x \urcorner \mid \pi \bullet \pi \qquad \varpi ::= x \mid \varpi \bullet \varpi$$

Thus, communication is asymmetric; consequently, the pattern matching function is defined only between an input and an output pattern and yields a single substitution. It is defined as:

$$\{\ulcorner x \urcorner \parallel x\} \stackrel{\text{def}}{=} \{\} \qquad \{\lambda x \parallel n\} \stackrel{\text{def}}{=} \{n/x\} \qquad \{\pi \bullet \pi' \parallel \varpi \bullet \varpi'\} \stackrel{\text{def}}{=} \{\pi \parallel \varpi\} \cup \{\pi' \parallel \varpi'\} \quad (4)$$

From the second rule, it is apparent that communicable patterns in Linda are single variable names. The operational rules for matching in the reductions and in the LTS are the following:

$$(\pi \to P) \mid (\varpi \to Q) \longmapsto \sigma P \mid Q \quad \text{if } \{\pi \parallel \varpi\} = \sigma \qquad \frac{P \xrightarrow{\pi} P' \quad Q \xrightarrow{(\nu \widetilde{n})\varpi} Q' \quad \{\pi \parallel \varpi\} = \sigma}{P \mid Q \xrightarrow{\tau} (\nu \widetilde{n})(\sigma P' \mid Q')} \; \widetilde{n} \cap \mathsf{fn}(P) = \emptyset$$

The theory of bisimulation is simplified in this setting, as $\ll$ is the identity. Barbed congruence can be defined as in Section 3.1 and the two equivalences do coincide.

Two interesting extensions of Linda's pattern matching (intermediate between Linda's and CPC's ones) are:

1. Accept a "non-fully decomposing" form of pattern matching; e.g., λx can match $n \bullet m$. In this case, it suffices to modify the definition of pattern matching by generalizing the second axiom in (4) to

   $$\{\lambda x \parallel n_1 \bullet \ldots \bullet n_k\} \stackrel{\text{def}}{=} \{n_1 \bullet \ldots \bullet n_k / x\}$$

   (i.e., by rolling back to the original definition of communicable patterns as sequences of variable names) and by defining $\ll$ as in Definition 6, except for the fourth axiom (that must be ignored).
2. Allow the output process to specify which names can be passed and which ones can only be used for testing equality; e.g., $n \bullet \ulcorner m \urcorner$ can be matched by $\lambda x \bullet \ulcorner m \urcorner$, but not by $\lambda x \bullet \lambda y$. In this case, output patterns are defined as

   $$\varpi ::= x \mid \ulcorner x \urcorner \mid \varpi \bullet \varpi$$

   This is resolved by adding to (4) the axiom $\{\ulcorner x \urcorner \parallel \ulcorner x \urcorner\} \stackrel{\text{def}}{=} \{\}$ and by defining $\ll$ as in Definition 6, except for the first axiom (that must be ignored).

In both cases, reductions and LTS are like Linda's ones; barbed congruence and bisimulation are defined as in Section 3 and, again, they do coincide.

Another well-known form of pattern matching is the one underlying the *polyadic π-calculus* [25]. In this case, (input and output) patterns have the form

$$\pi ::= \ulcorner a \urcorner \bullet \lambda x_1 \bullet \ldots \bullet \lambda x_k \qquad\qquad \varpi ::= \ulcorner a \urcorner \bullet n_1 \bullet \ldots \bullet n_k$$

for any $k > 0$ (these are usually written as $a(x_1, \ldots, x_k)$ and $\bar{a}(n_1, \ldots, n_k)$). Now pattern matching is defined as in (4), but with $\{\ulcorner x \urcorner \parallel \ulcorner x \urcorner\} \stackrel{\text{def}}{=} \{\}$ in place of the first axiom. Reductions, LTS and compatibility are like in Linda. Notice that the first two relations are the usual ones for the polyadic π-calculus; similarly, the bisimulation arising in this framework is the same as the standard early bisimulation congruence defined for the calculus. It is worth noticing that the barbs we exploit are different from the traditional ones for the π-calculus [27], where only the channel and the kind of action (either input or output) are observed. In our formulation of the polyadic π-calculus, input and output barbs can be usually distinguished: $\ulcorner a \urcorner \bullet \lambda x$ generates $\downarrow_{\{a\}}$ whereas $\ulcorner a \urcorner \bullet n$ generates $\downarrow_{\{a,n\}}$ (the two are indistinguishable only if $n = a$). In general, our barbs are more informative

than π-calculus' ones, since they also observe the argument of the output. However, since this barbed congruence coincides with the early bisimulation (that, in turn, coincides with the barbed congruence relying on the "standard" π-calculus' barbs), by transitivity we obtain that the two kinds of barbs yield the same congruence.

Similarly, also the form of pattern matching underlying the π-calculus with polyadic synchronization [8] can be easily rendered. It suffices to take

$$\pi ::= \ulcorner a_1 \urcorner \bullet \ldots \bullet \ulcorner a_k \urcorner \bullet \lambda x \qquad\qquad \varpi ::= \ulcorner a_1 \urcorner \bullet \ldots \bullet \ulcorner a_k \urcorner \bullet n$$

(usually written $a_1 \cdot \ldots \cdot a_k(x)$ and $\overline{a_1 \cdot \ldots \cdot a_k}(n)$). Pattern matching, reductions, LTS and compatibility are then the same as in polyadic π-calculus.

5 Conclusions and Future Work

CPC demonstrates the expressive power possible with a minimal process calculus whose interaction is defined by symmetric pattern unification. The behavioural theory required to capture CPC turns out to have some interesting properties based on patterns and pattern matching. Perhaps, the most curious one is that a symmetric relation (viz., bisimilarity) is defined by an (asymmetric) ordering upon patterns. Indeed, the resulting bisimulation can be smoothly and modularly adapted to cope with other forms of pattern matching and other process calculi.

Related work. To the best of our knowledge, there are very few notions of behavioural equivalences for process calculi that rely on pattern matching. We start with a few calculi based on a Linda-like pattern matching. A first example is [13], where the authors develop a testing framework; however, no coinductive and label-based equivalence is provided. Another paper where a Linda-like pattern matching is explored for bisimulation is [12]; however, there the focus is on the distribution and connectivity of processes and, consequently, the pattern matching is simplified by relying on patterns of length 1. A similar choice is taken in other works, e.g. [7, 10, 11]. Of course, this choice radically simplifies the theory.

Recently, Psi [4] has emerged as a rich framework that can encode different process calculi, including calculi with sophisticated forms of pattern matching. However, CPC and Psi are uncomparable: CPC cannot encode formulae (e.g. the indirect computation of channel equality), while Psi cannot encode self-matching processes (same as π, see [20]). The same holds for the applied π-calculus [1], because of the presence of active substitutions.

A more complex notion of bisimulation is the one for the Join calculus [14] given in [15]. The difficult part lays in the definition of the LTS, since some names can be marked as visible from outside their definition and, consequently, interact with the execution context. The definition of bisimilarity is then standard and, hence, the interplay with pattern matching is totally hidden within the LTS. We prefer to make it explicit in the bisimulation, both to keep the LTS as standard as possible and for showing the exact impact that pattern matching has on the semantics of processes. By the way, the form of pattern matching used in Join cannot be rendered in CPC. Indeed, in a process like **def** $a(x) \mid b(y) \triangleright P$ **in** R, process R can independently produce the outputs on a and b

needed to activate P. This would correspond to some form of "unordered and multiparty pattern matching" that is far from the design choices of CPC.

Other complex notions of bisimulation equivalences for process calculi are [5, 30]. However, these exploit environmental knowledge, whereas in our work we do not have such knowledge and need only satisfy compatibility.

Future work. One interesting path of further development is to introduce types into CPC and extend the pattern unification mechanism by taking types into account, as done e.g. in [9]. The study of *typed* equivalences would then be the most natural path to follow, by combing the theory in this paper with the assumed types. Another intriguing direction is the introduction of richer forms of pattern matching, based, e.g., on regular expressions [22]; in this case, it would be very challenging to devise the ordering on patterns that defines the 'right' bisimulation. A natural way to follow is Kozen's axiomatization for inclusion of regular language [24]. Indeed, in this proof system, a regular expression e_1 is smaller than e_2 if and only if every string belonging to the language generated by e_1 also belongs to the language generated by e_2. This corresponds to the same intuition as our ordering on patterns (Lemma 1), once we consider the language generated by a pattern as the set of patterns that it matches, together with the associated substitutions.

References

1. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. of POPL*, pages 104–115. ACM, 2001.
2. M. Abadi and A. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1 – 70, 1999.
3. R. M. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous π-calculus. *Theoretical Computer Science*, 195(2):291–324, 1998.
4. 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.
5. M. Boreale, R. De Nicola, and R. Pugliese. Proof techniques for cryptographic processes. *SIAM J. Comput.*, 31(3):947–986, 2001.
6. 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.
7. N. Busi, R. Gorrieri, and G. Zavattaro. A process algebraic view of linda coordination primitives. *Theoretical Computer Science*, 192(2):167–199, 1998.
8. M. Carbone and S. Maffeis. On the expressive power of polyadic synchronisation in pi-calculus. *Nordic Journal on Computing*, 10(2):70–98, 2003.
9. G. Castagna. Patterns and types for querying xml documents. In *10th Inter.Symp.on Database Programming Languages*, volume 3774 of *LNCS*, pages 1–26. Springer, 2005.
10. P. Ciancarini, R. Gorrieri, and G. Zavattaro. Towards a calculus for generative communication. In *Proc. of FMOODS*, pages 283–297. Chapman & Hall, 1996.
11. F. S. de Boer, J. W. Klop, and C. Palamidessi. Asynchronous communication in process algebra. In *Proc. of LICS*, pages 137–147. IEEE, 1992.
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. R. De Nicola and R. Pugliese. A process algebra based on linda. In *Proc. of COORDINATION*, volume 1061 of *LNCS*, pages 160–178. Springer, 1996.

14. C. Fournet and G. Gonthier. The reflexive cham and the join-calculus. In *Proc. of POPL*, pages 372–385. ACM Press, 1996.
15. C. Fournet and C. Laneve. Bisimulations in the join-calculus. *Theoretical Computer Science*, 266(1-2):569–603, 2001.
16. D. Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.
17. T. Given-Wilson. Concurrent pattern calculus in **bondi**. *Young Researchers Workshop on Concurrency Theory (YR-CONCUR)*, 2010.
18. T. Given-Wilson. Concurrent pattern unification, 2012. `http://www.progsoc.org/~sanguinev/files/GivenWilson-PhD-simple.pdf`.
19. T. Given-Wilson, D. Gorla, and B. Jay. Concurrent pattern calculus (extended version). `wwwusers.di.uniroma1.it/~gorla/papers/cpc-full.pdf`.
20. T. Given-Wilson, D. Gorla, and B. Jay. Concurrent pattern calculus. In *Proc. of IFIP-TCS*, volume 323 of *IFIP AICT*, pages 244 – 258. Springer, 2010.
21. K. Honda and N. Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 151(2):437–486, 1995.
22. H. Hosoya and B. Pierce. Regular expression pattern matching for XML. *Journal of Functional Programming*, 13(6):961–1004, 2003.
23. B. Jay and T. Given-Wilson. A combinatory account of internal structure. *Journal of Symbolic Logic*, 76(3):807–826, 2011.
24. D. Kozen. A completeness theorem for kleene algebras and the algebra of regular events. *Information and Computation*, 110(2):366–390, 1994.
25. R. Milner. The polyadic π-calculus: A tutorial. In *Logic and Algebra of Specification*, volume 94 of *Series F*. NATO ASI, Springer, 1993.
26. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part I/II. *Information and Computation*, 100:1–77, 1992.
27. R. Milner and D. Sangiorgi. Barbed bisimulation. In *Proc. of ICALP*, volume 623 of *LNCS*, pages 685–695. Springer, 1992.
28. 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.
29. D. Sangiorgi. A theory of bisimulation for the pi-calculus. *Acta Informatica*, 33(1):69–97, 1996.
30. D. Sangiorgi, N. Kobayashi, and E. Sumii. Environmental bisimulations for higher-order languages. *ACM Trans. Program. Lang. Syst.*, 33(1):5, 2011.