On the Expressiveness of Intensional Communication

Thomas Given-Wilson
INRIA, Paris, France *
thomas.given-wilson@inria.fr

The expressiveness of communication primitives has been explored in a common framework based on the π-calculus by considering four features: *synchronism* (asynchronous vs synchronous), *arity* (monadic vs polyadic data), *communication medium* (shared dataspaces vs channel-based), and *pattern-matching* (binding to a name vs testing name equality). Here pattern-matching is generalised to account for terms with internal structure such as in recent calculi like Spi calculi, Concurrent Pattern Calculus and Psi calculi. This paper explores *intensionality* upon terms, in particular communication primitives that can match upon both names and structures. By means of possibility/impossibility of encodings, this paper shows that intensionality alone can encode synchronism, arity, communication-medium, and pattern-matching, yet no combination of these without intensionality can encode any intensional language.

1 Introduction

The expressiveness of process calculi based upon their choice of communication primitives has been explored before [34, 6, 10, 20, 12]. In [20] this is detailed by examining combinations of four features, namely: *synchronism*, asynchronous versus synchronous; *arity*, monadic versus polyadic; *communication medium*, shared dataspaces versus channels; and *pattern-matching*, purely binding names versus name equality. These features are able to represent many popular calculi [20] such as: asynchronous or synchronous, monadic or polyadic π-calculus [30, 31, 29]; Linda [11]; Mobile Ambients [8]; μKlaim [32]; and semantic-π [9]. However, some recent process calculi include communicable primitives that have structure such as: Spi calculus [1] and pattern-matching Spi calculus [23]; Concurrent Pattern Calculus (CPC) [16, 17], and variations thereof [12]; and Psi calculi [2] and sorted Psi calculi [4]. Indeed (with the exception of Spi calculus) these calculi include communication primitives that account for the structure of the terms being communicated.

This paper abstracts away from specific calculi to provide a general account of the expressiveness of *intensional* communication primitives. Here intensionality is an advanced form of pattern-matching that allows *compound* structures of the form $s \bullet t$ to be bound to a single name, or to have their structure and components be matched in communication. For example, consider the following processes:

$$P \stackrel{\text{def}}{=} \langle a \bullet b \rangle \qquad Q \stackrel{\text{def}}{=} (x \bullet y).Q' \qquad R \stackrel{\text{def}}{=} (z).R' \qquad S \stackrel{\text{def}}{=} (\ulcorner a \urcorner \bullet \ulcorner b \urcorner).S'$$

where P is an output of the compound $a \bullet b$. The inputs of Q and R have binding names of the form x in their patterns $x \bullet y$ and z, respectively. The input of S tests the names a and b for equality and performs no binding. These process can be combined to form three possible reductions:

$$P \mid Q \longmapsto \{a/x, b/y\}Q' \qquad P \mid R \longmapsto \{a \bullet b/z\}R' \qquad P \mid S \longmapsto S' .$$

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

J. Borgström, S. Crafa (Eds.): Combined Workshop on Expressiveness in Concurrency and Structural Operational Semantics (EXPRESS/SOS 2014)
EPTCS 160, 2014, pp. 30–46, doi:10.4204/EPTCS.160.4

The first matches the structure of the output of P with the input of Q and binds a and b to x and y, respectively, in Q'. The second binds the entire output of P to the single name z in R'. The third matches the structure and names of the output of P with the structure and names of the input of S, as they match they interact although no binding or substitution occurs. This binding of arbitrary structures to a single name, combined with the name equality testing of the pattern-matching in [20] yields a more expressive intensionality in communication.

By generalising the pattern-matching feature to include intensionality the original sixteen calculi of [20] are here expanded to twenty-four. This paper details how all of the eight new calculi are more expressive than all of the original sixteen.

The key results are that intensionality is sufficient to encode: synchronous communication into asynchronous communication, polyadic communication into monadic communication, channel-based communication into dataspace-based communication, and generalises the original form of pattern-matching. The more interesting results are the encoding of polyadicity and channel-based communication, into monadic and dataspace-based communication, as synchronicity into asynchronicity is straightforward when either channel-based communication, or polyadicity and name-matching are available. (Encoding lesser pattern-matching into intensionality is trivial.)

In the other direction intensionality is impossible to encode with any other combination of the other features. This arises from the complexity of information that can be used to control communication in intensional interactions. The key to the result can be intuited by considering the encoding of a minimal input process $S_0 = (x).P$ and a minimal output process $S_1 = \langle a \rangle$. Their encodings must interact with some arity i, i.e. the reduction $[\![S_0 \mid S_1]\!] \longmapsto$ is between an input and output of arity i. Now a process S_2 can be constructed that outputs greater than i distinct names, along with a process S_3 that exactly matches all these names in a single interaction and evolves to Q. It follows that $[\![S_0 \mid S_2]\!]$ reduces with some arity j and also $[\![S_2 \mid S_3]\!]$ reduces with some arity k. Now, if $i = j = k$ then at least one name is not being tested for equality in the reduction of $[\![S_2 \mid S_3]\!]$ so there exists an S_4 that differs from S_3 by only that name. Thus $[\![S_4]\!]$ reduces with $[\![S_2]\!]$, but this contradicts a reasonable encoding as S_4 does not interact with S_3. If $i \neq j$ or $j \neq k$ then it is possible to show that the encoded process that is involved in the two different arities must be able to take either reduction. It follows that the process can take both reductions and reduce with two other encoded process and this leads to contradiction of the encoding. Either, it would be that $[\![S_0 \mid S_2 \mid S_3]\!] \Longmapsto [\![P \mid Q]\!]$ which is a contradiction as $S_0 \mid S_2 \mid S_3 \not\Longmapsto P \mid Q$. Or, it would be that the encoding introduces divergent computation, which contradicts a reasonable encoding.

The structure of the paper is as follows. Section 2 introduces the twenty-four calculi considered here. Section 3 revises the criteria used for encoding and comparing calculi. Section 4 explores synchronism into intensionality. Section 5 details arity into intensionality. Section 6 formalises communication-medium into intensionality and concludes that intensionality can encode all other features. Section 7 presents the impossibility of encoding intensionality into any other non-intensional calculus. Section 8 concludes, discusses future and related work, and some motivations for intensional calculi.

2 Calculi

This section defines the syntax, operational, and behavioural semantics of the calculi considered here. This relies heavily on the well-known notions developed for the π-calculus, the reference framework, and adapts them when necessary to cope with different features.

2.1 Syntax

Assume a countable set of names $\mathcal{N}$ ranged over by $a, b, c, \ldots$. Traditionally in π-calculus-style calculi names are used for channels, input bindings, and output data. However, here these need to be generalised to account for structure. Thus define the *terms* denoted with $s, t, \ldots$ to be

$$s, t \quad ::= \quad a \mid s \bullet t \,.$$

Terms can consist of names such as a, or of *compounds* $s \bullet t$ that combines two terms into one. The choice of the $\bullet$ as compound operator is similar to Concurrent Pattern Calculus, and also to be clearly distinct from the traditional comma-separated tuples of polyadic calculi.

The input primitives of different languages will exploit different kinds of patterns. The non-pattern-matching languages will simply use binding names, denoted $x, y, z, \ldots$. The *name-matching* patterns, denoted $m, n, o, \ldots$ and defined by

$$m, n \quad ::= \quad x \mid \ulcorner a \urcorner$$

consist of either a *binding name* x, or a *name-match* $\ulcorner a \urcorner$. Lastly the intensional patterns, denoted $p, q, \ldots$ will also consider structure and are defined by

$$p, q \quad ::= \quad m \mid p \bullet q \,.$$

The binding names x and name-match $\ulcorner a \urcorner$ are contained in m from the name-matching calculi, the *compound pattern* $p \bullet q$ combines p and q into a single pattern, and are left associative. The free names and binding names of name-matching and intensional patterns are as expected, taking the union of sub-patterns for compound patterns. Note that an intensional pattern is well-formed if and only if all binding names within the pattern are pairwise distinct. The rest of this paper will only consider well-formed intensional patterns.

The (parametric) syntax for the languages is:

$$P, Q, R \quad ::= \quad \mathbf{0} \mid \mathit{OutProc} \mid \mathit{IN}.P \mid (\nu n)P \mid P|Q \mid \textbf{if } s = t \textbf{ then } P \textbf{ else } Q \mid *P \mid \surd \,.$$

The different languages are obtained by replacing the output *OutProc* and input $IN.P$ with the various definitions. The rest of the process forms as are usual: $\mathbf{0}$ denotes the null process; restriction $(\nu n)P$ restricts the visibility of n to P; and parallel composition $P|Q$ allows independent evolution of P and Q. The **if** $s = t$ **then** P **else** Q represents conditional equivalence with **if** $s = t$ **then** P used when Q is $\mathbf{0}$. The $*P$ represents replication of the process P. Finally, the $\surd$ is used to represent a success process or state, exploited for reasoning about encodings as in [22, 12].

This paper considers the possible combinations of four features for communication: *synchronism* (synchronous vs asynchronous), *arity* (monadic vs polyadic data), *communication medium* (message passing vs shared dataspaces), and pattern-matching (simple binding vs name equality vs intensionality). As a result there exist twenty-four languages denoted as $\Lambda_{s,a,m,p}$ whose generic element is denoted as $\mathcal{L}_{\alpha,\beta,\gamma,\delta}$ where:

- $\alpha = A$ for asynchronous communication, and $\alpha = S$ for synchronous communication.
- $\beta = M$ for monadic data, and $\beta = P$ for polyadic data.
- $\gamma = D$ for dataspace-based communication, and $\gamma = C$ for channel-based communications.
- $\delta = NO$ for no matching capability, $\delta = NM$ for name-matching, and $\delta = I$ for intensionality.

$$P \mid \mathbf{0} \equiv P \qquad P \mid Q \equiv Q \mid P \qquad P \mid (Q \mid R) \equiv (P \mid Q) \mid R$$
$$\textbf{if } s = t \textbf{ then } P \textbf{ else } Q \equiv P \quad s = t \qquad \textbf{if } s = t \textbf{ then } P \textbf{ else } Q \equiv Q \quad s \neq t$$
$$P \equiv P' \quad \text{if } P =_\alpha P' \qquad (\nu a)\mathbf{0} \equiv \mathbf{0} \qquad (\nu a)(\nu b)P \equiv (\nu b)(\nu a)P$$
$$P \mid (\nu a)Q \equiv (\nu a)(P \mid Q) \quad \text{if } a \notin \mathsf{fn}(P) \qquad *P \equiv P \mid *P\,.$$

Figure 1: Structural equivalence relation.

For simplicity a dash $-$ will be used when the instantiation of that feature is unimportant.

Thus the syntax of every language is obtained from the following productions:

$\mathcal{L}_{A,-,-,-}$:	$OutProc ::= OUT$		
$\mathcal{L}_{S,-,-,-}$:	$OutProc ::= OUT.P$		
$\mathcal{L}_{-,M,D,NO}$:	$P,Q,R ::= \ldots$	$IN ::= (x)$	$OUT ::= \langle a \rangle$
$\mathcal{L}_{-,M,D,NM}$:	$P,Q,R ::= \ldots$	$IN ::= (m)$	$OUT ::= \langle a \rangle$
$\mathcal{L}_{-,M,D,I}$:	$P,Q,R ::= \ldots$	$IN ::= (p)$	$OUT ::= \langle t \rangle$
$\mathcal{L}_{-,M,C,NO}$:	$P,Q,R ::= \ldots$	$IN ::= a(x)$	$OUT ::= \overline{a}\langle b \rangle$
$\mathcal{L}_{-,M,C,NM}$:	$P,Q,R ::= \ldots$	$IN ::= a(m)$	$OUT ::= \overline{a}\langle b \rangle$
$\mathcal{L}_{-,M,C,I}$:	$P,Q,R ::= \ldots$	$IN ::= s(p)$	$OUT ::= \overline{s}\langle t \rangle$
$\mathcal{L}_{-,P,D,NO}$:	$P,Q,R ::= \ldots$	$IN ::= (\widetilde{x})$	$OUT ::= \langle \widetilde{a} \rangle$
$\mathcal{L}_{-,P,D,NM}$:	$P,Q,R ::= \ldots$	$IN ::= (\widetilde{m})$	$OUT ::= \langle \widetilde{a} \rangle$
$\mathcal{L}_{-,P,D,I}$:	$P,Q,R ::= \ldots$	$IN ::= (\widetilde{p})$	$OUT ::= \langle \widetilde{t} \rangle$
$\mathcal{L}_{-,P,C,NO}$:	$P,Q,R ::= \ldots$	$IN ::= a(\widetilde{x})$	$OUT ::= \overline{a}\langle \widetilde{b} \rangle$
$\mathcal{L}_{-,P,C,NM}$:	$P,Q,R ::= \ldots$	$IN ::= a(\widetilde{m})$	$OUT ::= \overline{a}\langle \widetilde{b} \rangle$
$\mathcal{L}_{-,P,C,I}$:	$P,Q,R ::= \ldots$	$IN ::= s(\widetilde{p})$	$OUT ::= \overline{s}\langle \widetilde{t} \rangle$.

Here the denotation $\widetilde{\cdot}$ represents a sequence of the form $\cdot_1, \cdot_2, \ldots, \cdot_n$ and can be used for names, terms, and both kinds of patterns. As usual $a(\ldots, x, \ldots).P$ and $(\nu x)P$ and $(x \bullet \ldots).P$ bind x in P. Observe that in $a(\ldots, \ulcorner b \urcorner, \ldots).P$ and $(\ldots \bullet \ulcorner b \urcorner).P$ neither a nor b bind in P, both are free. The corresponding notions of free and bound names of a process, denoted $\mathsf{fn}(P)$ and $\mathsf{bn}(P)$, are as usual. Also note that alpha-conversion, denoted $=_\alpha$ is assumed in the usual manner. Lastly, an input is well-formed if all binding names in that input occur exactly once, this paper shall only consider well-formed inputs. Finally, the structural equivalence relation $\equiv$ is defined in Figure 1.

Observe that $\mathcal{L}_{A,M,C,NO}$, $\mathcal{L}_{A,P,C,NO}$, $\mathcal{L}_{S,M,C,NO}$, and $\mathcal{L}_{S,P,C,NO}$ align with the communication primitives of the asynchronous/synchronous monadic/polyadic π-calculus [30, 31, 29]. The language $\mathcal{L}_{A,P,D,NM}$ aligns with LINDA[11]; the languages $\mathcal{L}_{A,M,D,NO}$ and $\mathcal{L}_{A,P,D,NO}$ with the monadic/polyadic Mobile Ambients [8]; and $\mathcal{L}_{A,P,C,NM}$ with that of μKLAIM [32] or semantic-π [9].

The intensional languages do not exactly match any well-known calculi. Indeed, the combinations of asynchrony and intensionality, or polyadicity and intensionality have no obvious candidates in the literature. However, the language $\mathcal{L}_{S,M,D,I}$ has been mentioned in [12], as a variation of Concurrent Pattern Calculus [16, 12], and has a behavioural theory as a specialisation of [15]. Similarly, the language $\mathcal{L}_{S,M,C,I}$ is very similar to pattern-matching Spi calculus [23] and Psi calculi [2], albeit with structural channel terms, and without the assertions or the possibility of repeated binding names in patterns. There are also similarities between $\mathcal{L}_{S,M,C,I}$ and the polyadic synchronous π-calculus of [7], although the intensionality is limited to the channel, i.e. inputs and outputs of the form $s(x).P$ and $\overline{s}\langle a \rangle.P$ respectively.

Remark The languages $\Lambda_{s,a,m,p}$ can be easily ordered; in particular $\mathcal{L}_{\alpha_1,\beta_1,\gamma_1,\delta_1}$ can be encoded into $\mathcal{L}_{\alpha_2,\beta_2,\gamma_2,\delta_2}$ if it holds that $\alpha_1 \leq \alpha_2$ and $\beta_1 \leq \beta_2$ and $\gamma_1 \leq \gamma_2$ and $\delta_1 \leq \delta_2$, where $\leq$ is the least reflexive relation satisfying the following axioms:

$$A \leq S \qquad M \leq P \qquad D \leq C \qquad NO \leq NM \leq I\,.$$

This can be understood as the lesser language variation being a special case of the more general language. Asynchronous communication is synchronous communication with all output followed by **0**. Monadic communication is polyadic communication with all tuples of arity one. Dataspace-based communication is channel-based communication with all k-ary tuples communicating with channel name k. Lastly, all name-matching communication is intensional communication without any compounds, and no-matching capability communication is both without any compounds and with only binding names in patterns.

2.2 Operational Semantics

The operational semantics of the languages is given here via reductions as in [29, 25]. An alternative style is via a *labelled transition system* (LTS) such as [20]. Here the reduction based style is to simplify having to define here the (potentially complex) labels that occur when intensionality is in play. However, the LTS style can be used for intensional languages [2, 12, 15], and indeed captures many[1] of the languages here [15].

Substitutions, denoted $\sigma,\rho,\ldots$, in non-pattern-matching and name-matching languages are mappings (with finite domain) from names to names. For intensional languages substitutions are mappings from names to terms. The application of a substitution σ to a pattern p is defined as follows:

$$\sigma x = \sigma(x)\ \ x \in \mathrm{domain}(\sigma) \qquad \sigma x = x\ \ x \notin \mathrm{domain}(\sigma) \qquad \sigma\ulcorner x\urcorner = \ulcorner(\sigma x)\urcorner \qquad \sigma(p \bullet q) = (\sigma p) \bullet (\sigma q)\,.$$

Where substitution is as usual on names, and on the understanding that the name-match syntax can be applied to any term by the following definition:

$$\ulcorner x\urcorner \stackrel{\text{def}}{=} \ulcorner x\urcorner \qquad \ulcorner(s \bullet t)\urcorner \stackrel{\text{def}}{=} \ulcorner s\urcorner \bullet \ulcorner t\urcorner\,.$$

Given a substitution σ and a process P, denote with σP the (capture-avoiding) application of σ to P that behaves in the usual manner. Note that capture can always be avoided by exploiting α-equivalence, which can in turn be assumed due to [39, 3].

Interaction between processes is handled by matching some terms $\widetilde{t}$ with some patterns $\widetilde{p}$, and possibly also equivalence of channel-names. This is handled in two parts, the *match* rule $\{t/\!/p\}$ of a single term t with a single pattern p to create a substitution σ. This is defined as follows:

$$\begin{array}{rcl} \{t/\!/x\} & \stackrel{\text{def}}{=} & \{t/x\} \\ \{a/\!/\ulcorner a\urcorner\} & \stackrel{\text{def}}{=} & \{\} \end{array} \qquad \begin{array}{rcll} \{s \bullet t/\!/ p \bullet q\} & \stackrel{\text{def}}{=} & \{s/\!/p\} \cup \{t/\!/q\} & \\ \{t/\!/p\} & \text{undefined} & \text{otherwise.} & \end{array}$$

Any term t can be matched with a binding name x to generate a substitution from the binding name to the term $\{t/x\}$. A single name a can be matched with a name-match for that name $\ulcorner a\urcorner$ to yield the empty substitution. A compound term $s \bullet t$ can be matched by a compound pattern $p \bullet q$ when the components match to yield substitutions $\{s/\!/p\} = \sigma_1$ and $\{t/\!/q\} = \sigma_2$, the resulting substitution is the unification of σ_1

[1] Perhaps all of the languages here, although this has not been proven.

and σ_2. Observe that since patterns are well-formed, the substitutions of components will always have disjoint domain. Otherwise the match is undefined.

The general case is then the *poly-match* rule $\textsc{Match}(\widetilde{t};\widetilde{p})$ that determines matching of a sequence of terms $\widetilde{t}$ with a sequence of patterns $\widetilde{p}$, that is defined below.

$$\textsc{Match}(;) = \emptyset \qquad \frac{\{s /\!/ p\} = \sigma_1 \qquad \textsc{Match}(\widetilde{t};\widetilde{q}) = \sigma_2}{\textsc{Match}(s,\widetilde{t};p,\widetilde{q}) = \sigma_1 \uplus \sigma_2}.$$

The empty sequence matches with the empty sequence to produce the empty substitution. Otherwise when there is a sequence of terms $s,\widetilde{t}$ and a sequence of patterns $p,\widetilde{q}$, the first elements are matched $\{s /\!/ p\}$ and the remaining sequences use the poly-match rule. If both are defined and yield substitutions, then the disjoint union $\uplus$ of substitutions is the result. (Like the match rule, the disjoint union is ensured by well-formedness of inputs.) Otherwise the poly-match rule is undefined, for example when a single match fails, or the sequences are of unequal arity.

Interaction is now defined by the following axiom:

$$\overline{s}\langle\widetilde{t}\rangle.P \mid s(\widetilde{p}).Q \qquad \longmapsto \qquad P \mid \sigma Q \qquad\qquad \textsc{Match}(\widetilde{t};\widetilde{p}) = \sigma$$

where the Ps are omitted in the asynchronous languages, and the s's are omitted for the dataspace-based languages. The axiom states that when the poly-match of the terms of an output $\widetilde{t}$ match with the patterns of an input $\widetilde{p}$ (and in the channel-based setting the output and input are along the same channel) yields a substitution σ, then reduce to (P in the synchronous languages in parallel with) σ applied to Q.

The general reduction relation $\longmapsto$ is defined as follows:

$$\frac{}{\overline{s}\langle\widetilde{t}\rangle.P \mid s(\widetilde{p}).Q \quad \longmapsto \quad P \mid \sigma Q} \quad \textsc{Match}(\widetilde{t};\widetilde{p}) = \sigma$$

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

with $\Longmapsto$ denoting the reflexive, transitive closure of $\longmapsto$.

Lastly, for each language let $\simeq$ denote a reduction-sensitive reference behavioural equivalence for that language, e.g. a barbed equivalence. For the non-intensional languages these are already known, either by their equivalent language in the literature, such as asynchronous/synchronous monadic/polyadic π-calculus, or from [20]. For the intensional languages the results in [15] can be used.

3 Encodings

This section recalls and adapts the definition of valid encodings as well as some useful theorems (details in [22]) for formally relating process calculi. The validity of such criteria in developing expressiveness studies emerges from the various works [20, 21, 22], that have also recently inspired similar works [27, 28, 19].

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* $C(\__1;\ldots;\__k)$ 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., $\mathcal{L}_2$).

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 $C^N_{\mathsf{op}}(\__1;\ldots;\__k)$ 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 $\llbracket \mathsf{op}(S_1,\ldots,S_k)\rrbracket = C^N_{\mathsf{op}}(\llbracket S_1\rrbracket;\ldots;\llbracket S_k\rrbracket)$.
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 $\llbracket S\rrbracket \Downarrow_2$.

Now recall a result concerning valid encodings that is useful for showing i.e. for proving that no valid encoding can exist between a pair of languages $\mathcal{L}_1$ and $\mathcal{L}_2$.

Proposition 3.1 (Proposition 5.5 from [22])**.** *Let* $\llbracket\cdot\rrbracket$ *be a valid encoding; then,* $S \longmapsto\!\!\!\!\!\!\not\;\;_1$ *implies that* $\llbracket S\rrbracket \longmapsto\!\!\!\!\!\!\not\;\;_2$.

4 Synchronism in Intensionality

This section proves that intensionality is sufficient to encode synchronicity. That is, that any language $\mathcal{L}_{S,\beta,\gamma,-}$ can be encoded into $\mathcal{L}_{A,\beta,\gamma,I}$.

The typical approach is to use channels and a fresh name to signal that the output has been received and thus encode synchronicity [20]. The approach here exploits a fresh name and intensionality, with optional channel-based communication to encode synchronicity in asynchronicity. Consider the translation $\llbracket\cdot\rrbracket$ that is the identity on all primitives except for input and output which are as follows:

$$\begin{aligned} \llbracket s(p,\widetilde{p}).P\rrbracket &\stackrel{\text{def}}{=} s(x\bullet p,\widetilde{p}).(\overline{x}\langle x\rangle \mid \llbracket P\rrbracket) \\ \llbracket \overline{s}\langle t,\widetilde{t}\rangle.Q\rrbracket &\stackrel{\text{def}}{=} (\nu x)(\overline{s}\langle x\bullet t,\widetilde{t}\rangle \mid x(\ulcorner x\urcorner).\llbracket Q\rrbracket) \end{aligned}$$

where x is not in the free names of s, p, $\widetilde{p}$, t, $\widetilde{t}$, P, or Q, and where $\widetilde{p}$ and $\widetilde{t}$ are omitted in the monadic case, and the channels are omitted in the dataspace-based communication case. The input is translated to receive an additional name x and then output this back to the translated output to signal interaction has occurred. Similarly the output restricts a fresh name x and then transmits this along with the original term. The continuation of the output is then placed under an input that only interacts with the fresh name.

Lemma 4.1. *Given a synchronous input P and a synchronous output Q then $[\![P]\!] \mid [\![Q]\!] \longmapsto$ if and only if $P \mid Q \longmapsto$.*

Proof. The proof is by definition of the poly-match rule. □

Lemma 4.2. *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. □

Lemma 4.3. *The translation $[\![\cdot]\!]$ from $\mathcal{L}_{S,\beta,\gamma,-}$ into $\mathcal{L}_{A,\beta,\gamma,I}$ preserves and reflects reductions. That is:*

1. *If $P \longmapsto P'$ then there exists Q such that $[\![P]\!] \longmapsto\!\!\!\!\longmapsto Q$ and $Q \equiv [\![P']\!]$;*
2. *if $[\![P]\!] \longmapsto Q$ then there exists Q' such that $Q \longmapsto Q'$ and $Q' \equiv [\![P']\!]$ for some P' such that $P \longmapsto P'$.*

Proof. Both parts can be proved by straightforward induction on the judgements $P \longmapsto P'$ and $[\![P]\!] \longmapsto Q$, respectively. In both cases, the base step is the most interesting and follows from Lemma 4.1, for the second case the step $Q \longmapsto Q'$ is ensured by the definition of the translation and match rule. The inductive cases where the last rule used is a structural one then rely on Lemma 4.2. □

Theorem 4.4. *For every language $\mathcal{L}_{S\beta,\gamma,-}$ there is a valid encoding into $\mathcal{L}_{A,\beta,\gamma,I}$.*

Proof. Compositionality and name invariance hold by construction. Operational correspondence (with structural equivalence in the place of $\simeq$) and divergence reflection follow from Lemma 4.3. Success sensitiveness can be proved as follows: $P \Downarrow$ means that there exists P' and $k \geq 0$ such that $P \longmapsto^k P' \equiv P'' \mid \surd$; by exploiting Lemma 4.3 k times and Lemma 4.2 obtain that $[\![P]\!] \longmapsto^{2k} [\![P']\!] \equiv [\![P'']\!] \mid \surd$, i.e. that $[\![P]\!] \Downarrow$. The converse implication can be proved similarly. □

5 Arity in Intensionality

This section proves that intensionality is sufficient to encode polyadicity. That is, that any language $\mathcal{L}_{\alpha,P,\gamma,-}$ can be encoded into $\mathcal{L}_{\alpha,M,\gamma,I}$.

The key to these encodings is the translation of the polyadic input and output forms into a single pattern or term, respectively. The translation $[\![\cdot]\!]$ is the identity on all forms except the input and output, which exploit a single reserved name rn (note that such a reserved name can be ensured by the renaming policy [22, 16]) and are translated as follows:

$$[\![s(p_1,\ldots,p_i).P]\!] \stackrel{\text{def}}{=} s((\ldots(\ulcorner \mathsf{rn} \urcorner \bullet p_1) \bullet \ldots) \bullet p_i).[\![P]\!]$$
$$[\![\overline{s}\langle t_1,\ldots,t_i\rangle.Q]\!] \stackrel{\text{def}}{=} \overline{s}\langle(\ldots(\mathsf{rn} \bullet t_1) \bullet \ldots) \bullet t_i\rangle.[\![Q]\!] \,.$$

Where the Qs are omitted in the asynchronous case, and the s's are omitted in the dataspace-based communication case.

Lemma 5.1. *Given a polyadic input P and a polyadic output Q then $[\![P]\!] \mid [\![Q]\!] \longmapsto$ if and only if $P \mid Q \longmapsto$.*

Proof. The proof is by induction on the arity of the polyadic input. □

Lemma 5.2. *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. □

Lemma 5.3. *The translation $[\![\cdot]\!]$ from $\mathcal{L}_{\alpha,P,\gamma,-}$ into $\mathcal{L}_{\alpha,M,\gamma,I}$ preserves and reflects reductions. That is:*

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

Proof. Both parts can be proved by straightforward induction on the judgements $P \longmapsto P'$ and $[\![P]\!] \longmapsto Q$, respectively. In both cases, the base step is the most interesting and follows from Lemma 5.1; the inductive cases where the last rule used is a structural one then rely on Lemma 5.2. □

Theorem 5.4. *For every language $\mathcal{L}_{\alpha,P,\gamma,-}$ there is a valid encoding into $\mathcal{L}_{\alpha,M,\gamma,I}$.*

Proof. Compositionality and name invariance hold by construction. Operational correspondence (with structural equivalence in the place of $\simeq$) and divergence reflection follow from Lemma 5.3. Success sensitiveness can be proved as follows: $P \Downarrow$ means that there exists P' and $k \geq 0$ such that $P \longmapsto^k P' \equiv P'' \mid \surd$; by exploiting Lemma 5.3 k times and Lemma 5.2 obtain that $[\![P]\!] \longmapsto^k [\![P']\!] \equiv [\![P'']\!] \mid \surd$, i.e. that $[\![P]\!] \Downarrow$. The converse implication can be proved similarly. □

6 Communication-Medium in Intensionality

This section proves that intensionality is sufficient to encode channel-based communication. That is, that any language $\mathcal{L}_{\alpha,\beta,C,-}$ can be encoded into $\mathcal{L}_{\alpha,\beta,D,I}$.

Similar to the polyadic into intensional case, the key is in the translation of the input and output forms. In general the translation $[\![\cdot]\!]$ is the identity on all forms except the input and output which are translated as follows:

$$[\![s(p,\widetilde{p}).P]\!] \stackrel{\text{def}}{=} (\ulcorner s \urcorner \bullet p, \widetilde{p}).[\![P]\!]$$
$$[\![\overline{s}\langle t,\widetilde{t}\rangle.Q]\!] \stackrel{\text{def}}{=} \langle s \bullet t, \widetilde{t}\rangle.[\![Q]\!]$$

where $\widetilde{p}$ and $\widetilde{t}$ are omitted in the monadic case. The input is translated into a pattern that compounds a name-match of the channel name with the pattern. The output is a simple compounding of the channel name with term (and the Qs are omitted in the asynchronous case).

Lemma 6.1. *Given a channel-based communication input P and a channel-based communication output Q then $[\![P]\!] \mid [\![Q]\!] \longmapsto$ if and only if $P \mid Q \longmapsto$.*

Proof. The proof is trivial by the definition of the poly-match and match rules. □

Lemma 6.2. *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. □

Lemma 6.3. *The translation $[\![\cdot]\!]$ from $\mathcal{L}_{\alpha,\beta,C,-}$ into $\mathcal{L}_{\alpha,\beta,D,I}$ preserves and reflects reductions. That is:*

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

Proof. Both parts can be proved by straightforward induction on the judgements $P \longmapsto P'$ and $[\![P]\!] \longmapsto Q$, respectively. In both cases, the base step is the most interesting and follows from Lemma 6.1; the inductive cases where the last rule used is a structural one then rely on Lemma 6.2. □

Theorem 6.4. *For every language $\mathcal{L}_{\alpha,\beta,C,-}$ there is a valid encoding into $\mathcal{L}_{\alpha,\beta,D,I}$.*

Proof. Compositionality and name invariance hold by construction. Operational correspondence (with structural equivalence in the place of $\simeq$) and divergence reflection follow from Lemma 6.3. Success sensitiveness can be proved as follows: $P \Downarrow$ means that there exists P' and $k \geq 0$ such that $P \longmapsto^k P' \equiv P'' \mid \surd$; by exploiting Lemma 6.3 k times and Lemma 6.2 obtain that $[\![P]\!] \longmapsto^k [\![P']\!] \equiv [\![P'']\!] \mid \surd$, i.e. that $[\![P]\!] \Downarrow$. The converse implication can be proved similarly. □

This concludes proving that intensionality can encode: synchronicity, polyadicity, channel-based communication, and name-matching. Thus any language $\mathcal{L}_{\alpha,\beta,\gamma,\delta}$ can be encoded into $\mathcal{L}_{A,M,D,I}$, and so all the intensional languages can encode each other, and thus are equally expressive.

Theorem 6.5. *Any language $\mathcal{L}_{\alpha 1,\beta 1,\gamma 1,\delta 1}$ can be encoded into any language $\mathcal{L}_{\alpha 2,\beta 2,\gamma 2,I}$. That is, intensionality alone is sufficient to encode: synchronicity, polyadicity, channel-based communication, and pattern-matching.*

Proof. When $\alpha 1 \leq \alpha 2$ then trivial by Remark 2.1, otherwise use Theorem 4.4. When $\beta 1 \leq \beta 2$ then trivial by Remark 2.1, otherwise use Theorem 5.4. When $\gamma 1 \leq \gamma 2$ then trivial by Remark 2.1, otherwise use Theorem 6.4. Finally, observe that $\delta 1 \leq \delta 2$ always holds and is trivial by Remark 2.1. □

7 Impossible Encodings

This section considers the impossibility of encoding intensionality with any combination of other properties. That is, that any language $\mathcal{L}_{\alpha 1,\beta 1,\gamma 1,I}$ cannot be encoded into $\mathcal{L}_{\alpha 2,\beta 2,\gamma 2,\delta}$ where $\delta \leq NM$. The key to the proof is to exploit the contractive nature of intensionality; that an arbitrarily large term can be bound to a single name, and also the possibility to match an infinite number of names in a single interaction. These two properties can be exploited to show that any attempt at encoding yields contradiction.

Theorem 7.1. *There is no valid encoding of any language $\mathcal{L}_{-,-,-,I}$ into $\mathcal{L}_{-,-,-,\delta}$ where $\delta \leq NM$.*

Proof. For simplicity the proof will show that any encoding from $\mathcal{L}_{A,M,D,I}$ into a language $\mathcal{L}_{\alpha,\beta,\gamma,\delta}$ is impossible, the generalisation follows from the fact that composition of encodings is an encoding.

The proof is by contradiction. Assume there exists a valid encoding $[\![\cdot]\!]$ from $\mathcal{L}_{A,M,D,I}$ into $\mathcal{L}_{\alpha,\beta,\gamma,\delta}$ where $\delta \leq NM$. Consider the encoding of the processes $S_0 = (x).\langle m\rangle$ and $S_1 = \langle a\rangle$. Clearly $[\![S_0 \mid S_1]\!] \longmapsto$ since $S_0 \mid S_1 \longmapsto$. There exists a reduction $[\![S_0 \mid S_1]\!] \longmapsto$ that must be between an input and output that both have (the same) maximal arity k.

Now define the following processes $S_2 \stackrel{\text{def}}{=} \langle a_1 \bullet \ldots \bullet a_{k+2}\rangle$ and $S_3 \stackrel{\text{def}}{=} (\ulcorner a_1 \urcorner \bullet \ldots \bullet \ulcorner a_{k+2} \urcorner).\langle m\rangle$ where S_2 outputs $k+2$ distinct names in a single term, and S_3 matches all of these names in a single intensional pattern.

Since $S_2 \mid S_0 \longmapsto$ it must be that $[\![S_2 \mid S_0]\!] \longmapsto$ for the encoding to be valid. Now consider the maximal arity of the reduction $[\![S_2 \mid S_0]\!] \longmapsto$:

- If the arity is k consider the reduction $[\![S_2 \mid S_3]\!] \longmapsto$ with the maximal arity j and that must exist since $S_2 \mid S_3 \longmapsto$. Now consider the relationship of j and k.

1. If $j = k$ then the upper bound on the number of names that are matched in the reduction is $k+1$. Since not all $k+2$ tuples of names from $\varphi_{[\![\,]\!]}(a_i)$ can be matched in the reduction then there must be at least one tuple $\varphi_{[\![\,]\!]}(a_i)$ for $i \in \{1,\ldots,k+2\}$ that is not being matched in the interaction $[\![S_2 \mid S_3]\!] \longmapsto$. Now construct S_4 that differs from S_3 only by swapping one such name a_i with m: $S_4 \stackrel{\text{def}}{=} (\ulcorner a_1 \urcorner \bullet \ldots \ulcorner a_{i-1} \urcorner \bullet \ulcorner m \urcorner \bullet \ulcorner a_{i+1} \urcorner \ldots \ulcorner a_{k+2} \urcorner).\langle a_i \rangle$. Now consider the context $C_{|}^{\mathcal{N}}([\![S_2]\!],[\![\cdot]\!]) = [\![S_2 \mid \cdot]\!]$ where $N = \{\widetilde{a} \cup m\}$. Clearly neither $C_{|}^{\mathcal{N}}([\![S_2]\!],[\![\mathbf{0}]\!]) \longmapsto$ nor $C_{|}^{\mathcal{N}}([\![S_2]\!],[\![S_4]\!]) \longmapsto$ as this would contradict Proposition 3.1. However, since S_3 and S_4 differ only by the position of one name whose tuple $\varphi_{[\![\,]\!]}(\cdot)$ does not appear in the reduction $[\![S_2 \mid S_3]\!] \longmapsto$, it follows that the reason $C_{|}^{\mathcal{N}}([\![S_2]\!],[\![S_4]\!]) \not\longmapsto$ must be due to a structural congruence difference between $C_{|}^{\mathcal{N}}([\![S_2]\!],[\![S_3]\!])$ and $C_{|}^{\mathcal{N}}([\![S_2]\!],[\![S_4]\!])$. Further, by compositionality of the encoding the difference can only be between $[\![S_3]\!]$ and $[\![S_4]\!]$. Since Proposition 3.1 ensures that $[\![S_3]\!] \not\longmapsto$ and $[\![S_4]\!] \not\longmapsto$, the only possibility is a structural congruence difference between $[\![S_3]\!]$ and $[\![S_4]\!]$. Now exploiting the substitution $\sigma = \{m/a_i, a_i/m\}$ that when applied to S_4 makes it S_3 yields contradiction.

2. If $j \neq k$ then we have that $[\![S_2]\!]$ must be able to interact with both arity k and arity j. That is, $[\![S_2 \mid \cdot]\!] = C_{|}^{\mathcal{N}}([\![S_2]\!],[\![\cdot]\!])$ where $N = \{\widetilde{a} \cup m\}$ and that $C_{|}^{\mathcal{N}}([\![S_2]\!],[\![S_0]\!])$ reduces with arity k and $C_{|}^{\mathcal{N}}([\![S_2]\!],[\![S_3]\!])$ reduces with arity j. Now it is straightforward, if tedious, to show that since $S_0 \mid S_3 \not\longmapsto$ that $C_{|}^{\mathcal{N}}([\![S_2]\!],[\![S_0 \mid S_3]\!])$ can perform the same initial reductions as either $C_{|}^{\mathcal{N}}([\![S_2]\!],[\![S_0 \mid \mathbf{0}]\!])$ or $C_{|}^{\mathcal{N}}([\![S_2]\!],[\![\mathbf{0} \mid S_3]\!])$ by exploiting operational correspondence and Proposition 3.1.

   Thus, it can be shown that $C_{|}^{\mathcal{N}}([\![S_2]\!],[\![S_0 \mid S_3]\!])$ can perform both the k arity reduction of $[\![S_2 \mid S_0]\!] \longmapsto$ and the j arity reduction of $[\![S_2 \mid S_3]\!] \longmapsto$. Now by exploiting the structural congruence rules it follows that neither of these initial reductions can prevent the other occurring. Thus, $C_{|}^{\mathcal{N}}([\![S_2]\!],[\![S_0 \mid S_3]\!])$ must be able to do both of these initial reductions in any order. Now consider the process R that has performed both of these initial reductions. By operational correspondence it must be that $R \not\Longrightarrow \simeq [\![\langle m \rangle \mid \langle m \rangle]\!]$ since $S_2 \mid S_0 \mid S_3 \not\Longrightarrow \langle m \rangle \mid \langle m \rangle$. Therefore, R must be able to roll-back the initial step with arity j; i.e reduce to a state that is equivalent to the reduction not occurring. Now consider how many names are being matched in the initial reduction with arity j. If $j < k+1$ the technique of differing on one name used in the case of $j = k$ can be used to show that this would introduce divergence on the potential roll-back and thus contradict a valid encoding. Therefore it must be that $j \geq k+1$. Finally, by exploiting name invariance and substitutions like $\{(b_1 \bullet \ldots \bullet b_{j+1})/a_1\}$ applied to S_2 and S_3 it follows that either $j \geq k + j + 1$ or both S_2 and S_3 must have infinitely many initial reductions which yields divergence.

- If the arity is not k then proceed like the second case above.

□

The proof above is for the general case, there are existing proofs in the literature that can be exploited for partial results. In particular, the techniques in [7], generalised in [20, 21], show that languages that allow for an arbitrary number of names to be matched in interaction cannot be encoded into to languages that can only match a limited number of names in interaction.

8 Conclusions and Future Work

Intensional communication primitives alone are highly expressive and can encode the behaviours of synchronous, polyadic, channel-based, and name-matching communication primitives. Thus, even the least intensional language can encode both the greatest non-intensional language and all the other intensional languages.

There are some languages that include intensionality in their operators, both outside of communication as in the Spi calculus [1], or as part of communication as in Concurrent Pattern Calculus (CPC) [16, 17] (and variations thereof [12]) and Psi calculi [2, 4]. However, only one variation of CPC matches any of the family of intensional languages defined here, that being $\mathcal{L}_{S,M,D,I}$ [12]. The equivalent expressiveness of all intensional languages here makes exploring each variant less interesting from a theoretical perspective, but also provides assurance that none is "better" than another from an expressiveness perspective. This also allows the expressiveness results here to be applied to both CPC and Psi calculi.

Future work in this area could include exploring the rôle of either symmetry or logics in communication. Symmetry of primitives would allow for better understanding of languages in the style of fusion calculus [35] or CPC. This may be of particular interest since symmetry has been used to show separation results, i.e. impossibility of encodings, from CPC into many languages represented here, as well as both fusion calculus and Psi calculi [16, 15, 17]. Alternatively, considering logics that play a rôle in communication would allow for capturing the behaviours of languages like Concurrent Constraint Programming [37] or Psi calculi. Again the rôle of logics has been used to show separation of Psi calculi from CPC [15].

Related Work

This section does not attempt to provide a detailed account of all related works as this would require an entire paper alone. Instead, some of the more closely related works are referenced here along with those that provide the best argument for and against the choices made here. Further, related works involving calculi with intensional communication primitives are highlighted.

Expressiveness in process calculi and similar languages has been widely explored, even when focusing mostly upon the choice of communication primitives [34, 6, 7, 7, 10, 24, 20, 16, 12, 17]. The choice of valid encodings here is that used, sometimes with mild adaptations, in [22, 21, 16, 33, 12, 17] and has also inspired similar works [27, 28, 19]. However, there are alternative approaches to encoding criteria or comparing expressive power [5, 38, 7, 36, 19]. Further arguments for, and against, the valid encodings here can be found in [22, 21, 19, 17].

There are also some results that fit in between the original 16 languages of [20] and those presented here with full intensionality. The polyadic synchronisation π-calculi [7] allows a vector of names in place of the channel name/term considered in [20] and here. This is likely to have similar expressive power to a name-matching polyadic languages, and can be easily represented in $\mathcal{L}_{S,M,C,I}$, with inputs and outputs of the form $s(x).P$ and $\overline{s}\langle a\rangle.P$ respectively.

There are already existing specific results for the intensional process calculi mentioned here. Concurrent Pattern Calculus (CPC) can homomorphically encode: π-calculus, Linda, and Spi Calculus while none of them can encode CPC [16, 17]. Meanwhile fusion calculus and Psi calculi are unrelated to CPC in that neither can encode CPC, and CPC cannot encode either of them [16, 12, 17]. Similarly Psi calculi can homomorphically encode π-calculus [2] and indirectly many other calculi, or directly using the techniques here. Impossibility of encoding results for both CPC and Psi calculi into many calculi can be derived from the results here.

Motivation

Clearly intensionality provides significant expressiveness when considering process calculi. However, there are further motivations for intensional process calculi that this paper has not attempted to address.

When considering computational expressiveness intensionality proves to increase expressive power in the sequential setting [26]. By allowing for functions that can match on the structure of their arguments (in the style that patterns can match against terms here), combinatory logics exist that prove more expressive than λ-calculus [26]. Indeed, when relating sequential computation to process calculi, intensionality in the latter allows for both expressing intensionality in the former [14], and for more elegantly capturing Turing Machines [13].

Cryptography, protocols, and security have proved motivating for Spi Calculus [1] and pattern-matching Spi Calculus [23], both of which introduce intensionality, the latter in communication as considered here. However, the intensionality presented here is too strong to support encryption (in the style of Spi Calculus) since it allows cracking of encryption via patterns of the form $\mathsf{enc} \bullet (\lambda p \bullet \lambda k)$ where p binds to the plaintext and k to the key [23, 12].

Psi calculi and sorted Psi calculi attempt to present a general framework for process calculi that can represent many existing process calculi as an instance of a (sorted) Psi calculi [4]. This paper has similar goals in that the results here improve understanding of the relations of many process calculi to one another. Further, the results allow for considering the most general language $\mathcal{L}_{S,P,C,I}$ while also recognising that any intensional language has equal expressiveness.

Similarly, the motivation for Concurrent Pattern Calculus is to both generalise the interaction approaches of many process calculi, and to represent desirable modeling properties such as exchange [12, 17]. Indeed, [12, 18] demonstrate how intensionality can be used to capture the pattern-matching of functional programming and data type constraints with more granularity than a type system.

References

[1] Martín Abadi & Andrew D. Gordon (1997): *A Calculus for Cryptographic Protocols: The Spi Calculus*. In: *Proceedings of the 4th ACM Conference on Computer and Communications Security*, CCS '97, ACM, New York, NY, USA, pp. 36–47, doi:10.1145/266420.266432.

[2] Jesper Bengtson, Magnus Johansson, Joachim Parrow & Björn Victor (2011): *Psi-calculi: a framework for mobile processes with nominal data and logic*. *Logical Methods in Computer Science* 7(1), doi:10.2168/LMCS-7(1:11)2011.

[3] Jesper Bengtson & Joachim Parrow (2009): *Formalising the pi-calculus using nominal logic*. *Logical Methods in Computer Science* 5(2), doi:10.2168/LMCS-5(2:16)2009.

[4] Johannes Borgström, Ramunas Gutkovas, Joachim Parrow, Björn Victor & Johannes Åman Pohjola (2013): *A Sorted Semantic Framework for Applied Process Calculi (Extended Abstract)*. In: *Trustworthy Global Computing*, pp. 103–118, doi:10.1007/978-3-319-05119-2_7.

[5] G. Boudol (1985): *Notes on Algebraic Calculi of Processes*. In Krzysztof R. Apt, editor: *Logics and Models of Concurrent Systems*, Springer-Verlag New York, Inc., New York, NY, USA, pp. 261–303, doi:10.1007/978-3-642-82453-1_9.

[6] Nadia Busi, Roberto Gorrieri & Gianluigi Zavattaro (2000): *On the Expressiveness of Linda Coordination Primitives*. *Information and Computation* 156(1-2), pp. 90–121, doi:10.1006/inco.1999.2823.

[7] Marco Carbone & Sergio Maffeis (2003): *On the Expressive Power of Polyadic Synchronisation in π-calculus*. *Nordic Journal of Computing* 10(2), pp. 70–98. Available at `http://dl.acm.org/citation.cfm?id=941344.941346`.

[8] Luca Cardelli & Andrew D. Gordon (1998): *Mobile Ambients*. In: *Foundations of Software Science and Computation Structures: First International Conference, FoSSaCS '98*, pp. 140–155, doi:10.1007/BFb0053547.

[9] Giuseppe Castagna, Rocco De Nicola & Daniele Varacca (2008): *Semantic Subtyping for the Pi-calculus*. *Theoretical Computer Science* 398(1-3), pp. 217–242, doi:10.1016/j.tcs.2008.01.049.

[10] Rocco De Nicola, Daniele Gorla & Rosario Pugliese (2006): *On the Expressive Power of KLAIM-based Calculi*. *Theoretical Computer Science* 356(3), pp. 387–421, doi:10.1016/j.tcs.2006.02.007.

[11] David Gelernter (1985): *Generative communication in* Linda. *ACM Transactions on Programming Languages and Systems* 7(1), pp. 80–112, doi:10.1145/2363.2433.

[12] Thomas Given-Wilson (2012): *Concurrent Pattern Unification*. PhD thesis, University of Technology, Sydney, Australia.

[13] Thomas Given-Wilson (2014): *An Intensional Concurrent Faithful Encoding of Turing Machines*. In: *7th Interaction and Concurrency Experience (ICE 2014)*, Berlin, Germany. Available at `http://hal.inria.fr/hal-00987594`.

[14] Thomas Given-Wilson (2014): *Expressiveness via Intensionality and Concurrency*. In: *To appear in: Proceedings of the 11th International Colloquium on Theoretical Aspects of Computing*, ICTAC 2014. Available at `http://arxiv.org/abs/1404.0956`.

[15] Thomas Given-Wilson & Daniele Gorla (2013): *Pattern Matching and Bisimulation*. In Rocco De Nicola & Christine Julien, editors: *Coordination Models and Languages*, *Lecture Notes in Computer Science* 7890, Springer Berlin Heidelberg, pp. 60–74, doi:10.1007/978-3-642-38493-6_5.

[16] Thomas Given-Wilson, Daniele Gorla & Barry Jay (2010): *Concurrent Pattern Calculus*. In Cristian S. Calude & Vladimiro Sassone, editors: *Theoretical Computer Science*, *IFIP Advances in Information and Communication Technology* 323, Springer Berlin Heidelberg, pp. 244–258, doi:10.1007/978-3-642-15240-5_18.

[17] Thomas Given-Wilson, Daniele Gorla & Barry Jay (2014): *A Concurrent Pattern Calculus*. *To appear in: Logical Methods in Computer Science*. Available at `http://hal.inria.fr/hal-00987578`.

[18] Thomas Given-Wilson & Barry Jay (2011): *Getting the Goods with Concurrent* **bondi**. *Proceedings of the Fourth Workshop on Programming Language Approaches to Concurrency and Communication-cEntric Software (PLACES)*.

[19] Rob J. van Glabbeek (2012): *Musings on Encodings and Expressiveness*. In: *Proceedings of EXPRESS/SOS*, *EPTCS* 89, pp. 81–98, doi:10.4204/EPTCS.89.7.

[20] D. Gorla (2008): *Comparing Communication Primitives via their Relative Expressive Power*. *Information and Computation* 206(8), pp. 931–952, doi:10.1016/j.ic.2008.05.001.

[21] D. Gorla (2010): *A Taxonomy of Process Calculi for Distribution and Mobility*. *Distributed Computing* 23(4), pp. 273–299, doi:10.1007/s00446-010-0120-6.

[22] D. Gorla (2010): *Towards a Unified Approach to Encodability and Separation Results for Process Calculi*. *Information and Computation* 208(9), pp. 1031–1053, doi:10.1016/j.ic.2010.05.002.

[23] Christian Haack & Alan Jeffrey (2006): *Pattern-matching Spi-calculus*. *Information and Computation* 204(8), pp. 1195–1263, doi:10.1016/j.ic.2006.04.004.

[24] Bjørn Haagensen, Sergio Maffeis & Iain Phillips (2008): *Matching Systems for Concurrent Calculi*. *Electronic Notes in Theoretical Computer Science* 194(2), pp. 85 – 99, doi:10.1016/j.entcs.2007.11.004. Proceedings of the 14th International Workshop on Expressiveness in Concurrency (EXPRESS 2007).

[25] Kohei Honda & Nobuko Yoshida (1995): *On reduction-based process semantics*. *Theoretical Computer Science* 152, pp. 437–486, doi:10.1016/0304-3975(95)00074-7.

[26] Barry Jay & Thomas Given-Wilson (2011): *A combinatory account of internal structure*. *Journal of Symbolic Logic* 76(3), pp. 807–826, doi:10.2178/jsl/1309952521.

[27] Ivan Lanese, Jorge A. Pérez, Davide Sangiorgi & Alan Schmitt (2010): *On the Expressiveness of Polyadic and Synchronous Communication in Higher-Order Process Calculi*. In: *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP), LNCS* 6199, Springer, pp. 442–453, doi:10.1007/978-3-642-14162-1_37.

[28] Ivan Lanese, Cátia Vaz & Carla Ferreira (2010): *On the Expressive Power of Primitives for Compensation Handling*. In: *Proceedings of the 19th European Conference on Programming Languages and Systems*, ESOP'10, Springer-Verlag, Berlin, Heidelberg, pp. 366–386, doi:10.1007/978-3-642-11957-6_20.

[29] Robin Milner (1993): *The Polyadic π-Calculus: A Tutorial*. In: *Logic and Algebra of Specification, Series F* 94, NATO ASI, Springer, doi:10.1007/978-3-642-58041-3_6.

[30] Robin Milner, Joachim Parrow & David Walker (1992): *A Calculus of Mobile Processes, I*. *Information and Computation* 100(1), pp. 1–40, doi:10.1016/0890-5401(92)90008-4.

[31] Robin Milner, Joachim Parrow & David Walker (1992): *A Calculus of Mobile Processes, II*. *Information and Computation* 100(1), pp. 41–77, doi:10.1016/0890-5401(92)90009-5.

[32] Rocco De Nicola, Gian Luigi Ferrari & Rosario Pugliese (1998): *KLAIM: A Kernel Language for Agents Interaction and Mobility*. *IEEE Transactions on Software Engineering* 24(5), pp. 315–330, doi:10.1109/32.685256.

[33] Lasse Nielsen, Nobuko Yoshida & Kohei Honda (2010): *Multiparty Symmetric Sum Types*. In: *Proceedings of the 17th International Workshop on Expressiveness in Concurrency (EXPRESS 2010)*, pp. 121–135, doi:10.4204/EPTCS.41.9.

[34] Catuscia Palamidessi (2003): *Comparing the Expressive Power of the Synchronous and Asynchronous pi-calculi*. *Mathematical. Structures in Comp. Sci.* 13(5), pp. 685–719, doi:10.1017/S0960129503004043.

[35] J. Parrow & B. Victor (1998): *The fusion calculus: expressiveness and symmetry in mobile processes*. In: *Proceedings of Thirteenth Annual IEEE Symposium on Logic in Computer Science*, pp. 176–185, doi:10.1109/LICS.1998.705654.

[36] Joachim Parrow (2008): *Expressiveness of Process Algebras*. *Electronic Notes in Theoretical Computer Science* 209, pp. 173–186, doi:10.1016/j.entcs.2008.04.011.

[37] Vijay A. Saraswat, Martin Rinard & Prakash Panangaden (1991): *The Semantic Foundations of Concurrent Constraint Programming*. In: *Proceedings of the 18th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '91, ACM, New York, NY, USA, pp. 333–352, doi:10.1145/99583.99627.

[38] Robert de Simone (1985): *Higher-Level Synchronising Devices in Meije-SCCS*. *Theoretical Computer Science* 37, pp. 245–267, doi:10.1016/0304-3975(85)90093-3.

[39] Christian Urban, Stefan Berghofer & Michael Norrish (2007): *Barendregts Variable Convention in Rule Inductions*. In Frank Pfenning, editor: *Automated Deduction CADE-21, Lecture Notes in Computer Science* 4603, Springer Berlin Heidelberg, pp. 35–50, doi:10.1007/978-3-540-73595-3_4.

Appendix A

This appendix contains a more detailed proof for Theorem 7.1.

Proof. For simplicity the proof will show that any encoding from $\mathcal{L}_{A,M,D,I}$ into a language $\mathcal{L}_{\alpha,\beta,\gamma,\delta}$ is impossible, the generalisation follows from the fact that composition of encodings is an encoding.

The proof is by contradiction. Assume there exists a valid encoding $[\![\cdot]\!]$ from $\mathcal{L}_{A,M,D,I}$ into $\mathcal{L}_{\alpha,\beta,\gamma,\delta}$ where $\delta \leq NM$. Consider the encoding of the processes $S_0 = (x).\langle m\rangle$ and $S_1 = \langle a\rangle$. Clearly $[\![S_0 \mid S_1]\!] \longmapsto$ since $S_0 \mid S_1 \longmapsto$. There exists a reduction $[\![S_0 \mid S_1]\!] \longmapsto$ that must be between an input and output that both have (the same) maximal arity k. (Observe that when $\beta = M$, i.e. encoding into a monadic language, then k must be 1.)

Now define the following processes $S_2 \stackrel{\text{def}}{=} \langle a_1 \bullet \ldots \bullet a_{k+2}\rangle$ and $S_3 \stackrel{\text{def}}{=} (\ulcorner a_1 \urcorner \bullet \ldots \bullet \ulcorner a_{k+2} \urcorner).\langle m\rangle$ where S_2 outputs $k+2$ distinct names in a single term, and S_3 matches all of these names in a single intensional pattern.

Since $S_2 \mid S_0 \longmapsto$ it must be that $[\![S_2 \mid S_0]\!] \longmapsto$ for the encoding to be valid. Now consider the maximal arity of the reduction $[\![S_2 \mid S_0]\!] \longmapsto$:

- If the arity is k (this must hold for $\beta = M$, i.e. encoding into a monadic language) consider the reduction $[\![S_2 \mid S_3]\!] \longmapsto$ with the maximal arity j and that must exist since $S_2 \mid S_3 \longmapsto$. Now consider the relationship of j and k.
  1. If $j = k$ (this must hold for $\beta = M$, i.e. encoding into a monadic language) then the upper bound on the number of names that are matched in the reduction is $k+1$ when $\gamma = C$ and $\delta = NM$, i.e. encoding into a channel-based pattern-matching language. (The upper bound is k for $\gamma = D$ and $\delta = NM$; 1 for $\gamma = C$ and $\delta = NO$; and 0 for $\gamma = D$ and $\delta = NO$.) Since not all $k+2$ tuples of names from $\varphi_{[\![\,]\!]}(a_i)$ can be matched in the reduction then there must be at least one tuple $\varphi_{[\![\,]\!]}(a_i)$ for $i \in \{1,\ldots,k+2\}$ that is not being matched in the interaction $[\![S_2 \mid S_3]\!] \longmapsto$. Now construct S_4 that differs from S_3 only by swapping one such name a_i with m:
$$S_4 \stackrel{\text{def}}{=} (\ulcorner a_1 \urcorner \bullet \ldots \ulcorner a_{i-1} \urcorner \bullet \ulcorner m \urcorner \bullet \ulcorner a_{i+1} \urcorner \ldots \ulcorner a_{k+2} \urcorner).\langle a_i\rangle\ .$$
  Now consider the context $C_{|}^{\mathcal{N}}([\![S_2]\!], [\![\cdot]\!]) = [\![S_2 \mid \cdot]\!]$ where $N = \{\widetilde{a} \cup m\}$. Clearly neither $C_{|}^{\mathcal{N}}([\![S_2]\!], [\![\mathbf{0}]\!]) \longmapsto$ nor $C_{|}^{\mathcal{N}}([\![S_2]\!], [\![S_4]\!]) \longmapsto$ as this would contradict Proposition 3.1. However, since S_3 and S_4 differ only by the position of one name whose tuple $\varphi_{[\![\,]\!]}(\cdot)$ does not appear in the reduction $[\![S_2 \mid S_3]\!] \longmapsto$, it follows that the reason $C_{|}^{\mathcal{N}}([\![S_2]\!], [\![S_4]\!]) \not\longmapsto$ must be due to a structural congruence difference between $C_{|}^{\mathcal{N}}([\![S_2]\!], [\![S_3]\!])$ and $C_{|}^{\mathcal{N}}([\![S_2]\!], [\![S_4]\!])$. Further, by compositionality of the encoding the difference can only be between $[\![S_3]\!]$ and $[\![S_4]\!]$. Since Proposition 3.1 ensures that $[\![S_3]\!] \not\longmapsto$ and $[\![S_4]\!] \not\longmapsto$, the only possibility is a structural congruence difference between $[\![S_3]\!]$ and $[\![S_4]\!]$.
  Now proceed by induction on the structure of the encoded processes to determine their structural difference.
  (a) If the difference is a restriction (νc) then observe that restrictions can only prevent reductions, not introduce reductions. Since the reduction is prevented for S_4 it follows that $[\![S_4]\!]$ must be of the form $(\nu c)R_4$ where R_4 includes an input or output that contains c. Now consider the substitution $\sigma = \{m/a_i, a_i/m\}$ that when applied to S_4 makes it S_3. By name invariance it must hold that there exists σ' such that $[\![\sigma S_4]\!] = \sigma'[\![S_4]\!] = [\![S_3]\!]$. However, by definition of the application of a substitution σ' cannot rename c and so

$C_|^N([\![S_2]\!], \sigma'[\![S_4]\!]) \not\longmapsto$ which means that $C_|^N([\![S_2]\!], [\![S_3]\!]) \not\longmapsto$ which yields contradiction.

(b) If the difference is an **if** $c_1 = c_2$ **then** S **else** T construct then consider the relation of c_1 to c_2 that can only depend upon the translation and must relate a_i or m. As in the previous case, name invariance and the substitution $\sigma = \{m/a_i, a_i/m\}$ that when applied to S_4 makes it S_3, can be used to show contradiction.

(c) Otherwise proceed by induction.

2. If $j \neq k$ (then it must be that $\beta = P$, i.e. encoding into a polyadic language) then we have that $[\![S_2]\!]$ must be able to interact with both arity k and arity j. That is, $[\![S_2 \mid \cdot]\!] = C_|^N([\![S_2]\!], [\![\cdot]\!])$ where $N = \{\widetilde{a} \cup m\}$ and that $C_|^N([\![S_2]\!], [\![S_0]\!])$ reduces with arity k and $C_|^N([\![S_2]\!], [\![S_3]\!])$ reduces with arity j. Now it is straightforward, if tedious, to show that since $S_0 \mid S_3 \not\longmapsto$ that $C_|^N([\![S_2]\!], [\![S_0 \mid S_3]\!])$ can perform the same initial reductions as either $C_|^N([\![S_2]\!], [\![S_0 \mid \mathbf{0}]\!])$ or $C_|^N([\![S_2]\!], [\![\mathbf{0} \mid S_3]\!])$ by exploiting operational correspondence and Proposition 3.1.

   Thus, it can be shown that $C_|^N([\![S_2]\!], [\![S_0 \mid S_3]\!])$ can perform both the k arity reduction of $[\![S_2 \mid S_0]\!] \longmapsto$ and the j arity reduction of $[\![S_2 \mid S_3]\!] \longmapsto$. Now by exploiting the structural congruence rules it follows that neither of these initial reductions can prevent the other occurring. (The only structural congruence rule that could prevent an interaction occurring is the **if** $c_1 = c_2$ **then** P **else** Q where c_1 or c_2 is modified by the first interaction. However, this would require that this congruence be under the input involved in the interaction, which would then mean the, now prevented, interaction could not occur before the other which contradicts the encoding.) Thus, $C_|^N([\![S_2]\!], [\![S_0 \mid S_3]\!])$ must be able to do both of these initial reductions in any order.

   Now consider the process R that has performed both of these initial reductions. By operational correspondence it must be that $R \not\Longmapsto \simeq [\![\langle m \rangle \mid \langle m \rangle]\!]$ since $S_2 \mid S_0 \mid S_3 \not\Longmapsto \langle m \rangle \mid \langle m \rangle$. Therefore, R must be able to roll-back the initial step with arity j; i.e reduce to a state that is equivalent to the reduction not occurring. (Or the initial step with arity k, but either one is sufficient as by operational correspondence $R \Longmapsto \simeq [\![\langle m \rangle \mid S_3]\!]$.)

   Now consider how many names are being matched in the initial reduction with arity j. If $j < k+1$ the technique of differing on one name used in the case of $j = k$ can be used to show that this would introduce divergence on the potential roll-back and thus contradict a valid encoding. (The initial reduction could have occurred when one or more names are different, and thus would happen anyway and need to be rolled back. Since the roll-back must not change the ability of the processes to interact with other processes, this can be shown to lead to an infinite reduction sequence, and thus contradict divergence reflection.)

   Therefore it must be that $j \geq k+1$. Finally, by exploiting name invariance and substitutions like $\{(b_1 \bullet \ldots \bullet b_{j+1})/a_1\}$ applied to S_2 and S_3 it follows that either $j \geq k + j + 1$ or both S_2 and S_3 must have infinitely many initial reductions which yields divergence.

- If the arity is not k then proceed like the second case above.

□