Expressiveness of Concurrent Intensionality

Ioana Cristescu[a], Thomas Given-Wilson[b,*], Axel Legay[b]

[a]*TAMIS, Inria, Rennes, France*
[b]*Universite Catholique de Louvain, INGI Place Sainte Barbe bte 2 L05.02.01, 1348 Louvain-la-Neuve, Belgium*

Abstract

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*, a feature that extends pattern-matching to allow communication primitives to interact by also matching on the structure of terms. 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. Further, some languages may also be non-linear, where two inputs must be equal to allow interaction. This paper also explores all the relations between linear and non-linear variations of the above languages.

Keywords: Process calculi, pattern-matching, intensionality, linearity, expressiveness, encodings

1. Introduction

The expressiveness of process calculi based upon their choice of communication primitives has been explored before [38, 6, 11, 23, 16, 14, 15, 17, 21, 22]. In [23] 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 such as: asynchronous or synchronous, monadic or polyadic π-calculus [35, 36, 34]; Linda [13]; Mobile Ambients [8]; Klaim [10]; and semantic-π [9]. However, some

*Corresponding author
Email addresses: ioana-domnina.cristescu@inria.fr (Ioana Cristescu), thomas.given-wilson@uclouvain.be (Thomas Given-Wilson), axel.legay@uclouvain.be (Axel Legay)

recent process calculi include communication primitives that support more structured terms such as: Spi calculus [1] and pattern-matching Spi calculus [26]; Concurrent Pattern Calculus (CPC) [19, 20], and variations thereof [14]; and Psi calculi [2] and sorted Psi calculi [3]. Indeed these calculi include communication primitives that account for the structure of the terms being communicated.

In [16, 17] the original work of [23] is extended by considering *intensionality* as a more advanced form of pattern-matching. This paper extends this exploration further by considering in greater detail the results of [16, 17]. Further, this work considers *non-linear* matching primitives that have not been considered in connection to intensionality before.

This paper abstracts away from specific calculi in the style of [23, 16, 14, 15, 17, 21, 22] to provide a general account of the expressiveness of intensional and non-linear 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 matched in communication.

For example consider the process

$$P \stackrel{\text{def}}{=} \langle a \bullet b \rangle$$

where P is an output of the compound $a \bullet b$ that combines the two names a and b into a single structure. Now consider the following processes that can interact with P.

The first is

$$Q \stackrel{\text{def}}{=} (x \bullet y).Q'$$

that has an input $x \bullet y$ that binds two names x and y in a compound structure and then evolves to the process Q'. The interaction of P and Q is

$$P \mid Q \longmapsto \{a/x, b/y\}Q'$$

where the compound $a \bullet b$ is matched against $x \bullet y$ and then each component of the compound is bound separately, yielding the substitution $\{a/x, b/y\}$ that binds a to x and b to y in the process Q'. Observe that here the intensionality is exhibited in the structure of the output and input being required to match for the interaction to occur.

The second is

$$R \stackrel{\text{def}}{=} (z).R'$$

that has an input of a single binding name z that then evolves to the process R'. The interaction of P and R is

$$P \mid R \longmapsto \{a \bullet b/z\}R'$$

where the entire compound $a \bullet b$ is bound to the binding name z in the process R'. Observe that here the intensionality is exhibited by the entire structure $a \bullet b$ being bound to a single name.

The third is

$$S \stackrel{\text{def}}{=} (\ulcorner a \urcorner \bullet \ulcorner b \urcorner).S'$$

that has an input that tests for the structure of the output and also tests the names a and b for equality. The interaction of P and S is

$$P \mid S \longmapsto S'$$

and checks that the structures $a \bullet b$ and $\ulcorner a \urcorner \bullet \ulcorner b \urcorner$ are both compounds, and both have the same names a and b in their components. This ensures equality of the two structures, and yields no substitution over S'. Observe that here the intensionality is in the matching of the structure of the output and input, and then the testing for equality of names. Other possible processes are $\ulcorner a \urcorner \bullet x.S'$ and $y \bullet \ulcorner b \urcorner.S'$ which interact with $a \bullet b$ similarly to above.

This binding of arbitrary structures to a single name, combined with the name equality testing of the pattern-matching in [23] yields a more expressive intensionality in communication [16].

The above processes are all *linear*, that is the binding names in an input appear only once. For instance in Q the binding names x and y must be distinct. In this paper *non-linear* input primitives are also considered where an input may repeat the same binding name any number of times.

For example, consider two processes that perform an output:

$$P \stackrel{\text{def}}{=} \langle a \bullet b \rangle \qquad\qquad Q \stackrel{\text{def}}{=} \langle a \bullet a \rangle \,.$$

Observe that both P and Q output a compound structure with two names. In the case of P this is the same as before, with names a and b (here assumed not to be equal). In the case of Q the structure $a \bullet a$ outputs two instances of the same name combined into a compound. Now consider two possible process that could attempt to interact with P and Q.

The first is

$$R \stackrel{\text{def}}{=} (x \bullet y).R'$$

that has an input of a compound $x \bullet y$ with two distinct binding names x and y. The process R can interact with both P and Q. The interaction with P is as follows

$$P \mid R \longmapsto \{a/x, b/y\}R'$$

where the structure of the compound is matched, and the binding names bind to the components. The interaction with Q is as follows

$$Q \mid R \longmapsto \{a/x, a/y\}R'$$

where the bindings are as above, with both x and y binding to a.

The second is

$$S \stackrel{\text{def}}{=} (z \bullet z).S'$$

that has the same compound structure but the same binding name z repeated twice. This process can only interact with Q as follows

$$Q \mid S \quad \longmapsto \quad \{a/z\}S'$$

where the structure is matched as usual, and the two instances of z are both bound to a. Observe that here the non-linearity of the two instances of the binding name z induces a condition on the matching that they must both bind to the same term, here a. If they would bind to distinct terms then the matching is undefined, and interaction cannot occur. That is

$$P \mid S \quad \not\longmapsto$$

the process P and S cannot interact since, although the structures match, the names a and b are not equal.

Observe that non-linearity can appear in any calculus that has more than one binding name in an input. In practice this includes all polyadic or intensional process calculi, but *not* monadic non-intensional calculi, where at most one binding name appears in an input.

By generalising the pattern-matching feature of a calculus to include intensionality the original sixteen calculi of [23] would here be expanded to twenty-two. The further addition of the non-linear languages increases this number to forty. This paper explores the detail of the relations between these various calculi, including the understanding of the subtle relations brought into play by intensionality and non-linearity.

The main results of this paper are in showing both encodings and separation results when contrasting these forty languages. An overview of these results is shown in Figure 1 detailing the relations between the languages that we consider in this paper. Here languages are denoted by $\mathcal{L}_{\alpha,\beta,\gamma,\delta}$ where the subscripts denote the different features of the language (details in Section 2). The lack of an arrow between two languages indicates that there is no relation between them. The lower part of the figure with arrows in black is adapted from Figure 1 of [23]. This work considers intensional and non-linear extensions: the green arrows are results related to intensionality, the blue arrows related to non-linearity.

Intensionality proves to be more expressive than non-intensionality, with the generalisation to an intensional language always being more expressive than a non-intensional language of the same form. The encoding of a non-intensional language into an intensional language is straightforward, since the structure of intensionality can represent structures to encode polyadicity, channel-names, or other forms of pattern-matching.

The separation results are proven by exploiting the ability for intensional languages to synchronise on arbitrarily complex patterns in a single interaction, while non-intensional languages are always bounded in the complexity of the structure they can consider in a single interaction. This allows a separation result that no intensional language can be encoded into a non-intensional language. This result is not obvious, since there are arguments that: arbitrarily complex patterns can be matched be equality; name-matching generally supports matching of an unbounded number of names; or any complex matching can be computed and the languages considered already support arbitrary computation. A brief comment on the arguments can be summarised as follows.

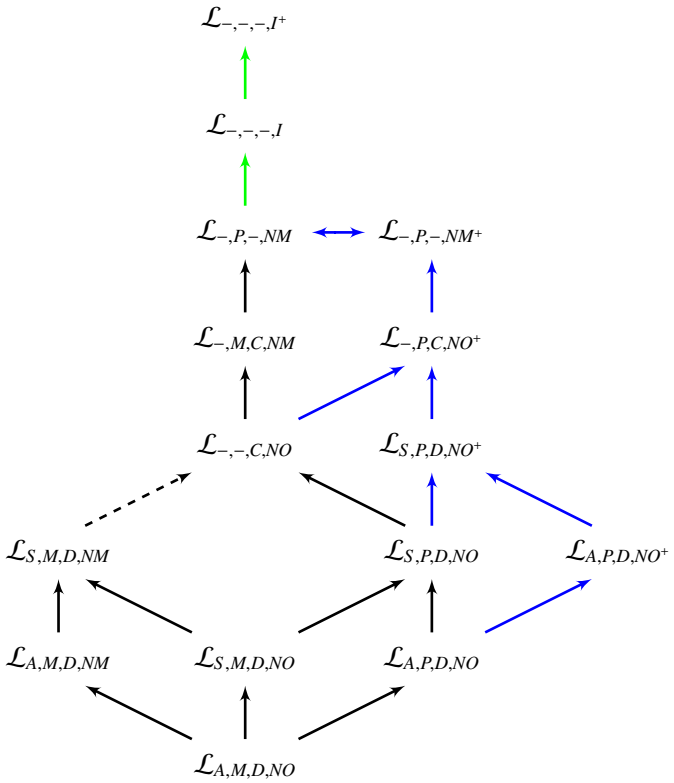

Figure 1: Overview of results and relations between languages.

Intensionality is more expressive than arbitrary equality since intensionality allows sub-patterns to be matched exactly, or inexactly. For example, ensuring a pattern is at least a pair, does not mean equality with a pair, since a sub-component could be more complex. Even allowing names to represent the structures and complexity used does not allow non-intensional languages to encode intensional languages, since there is no bound on the complexity of the structure in an intensional language, and substitutions can always increase the complexity.

The unbounded matching of names in name-matching languages suffers from the above complexity when attempting to encode intensionality. Again the sub-structure problem with encoding intensional languages arises, where an encoding must be sensitive to both the existing structure, and the changes to structure induced by substitutions.

Finally, the argument that arbitrary computation can be used does hold in that the structures could be represented and computed. However, due to the opportunistic and non-deterministic nature of interaction in process calculi, such an approach yields an encoding that can never terminate. This violates one of the goals of an encoding, since the encodings considered here (and in general) avoid introducing non-terminating reductions.

When considering linearity the results are inter-related to those of intensionality. Here all non-intensional non-linear languages can be encoded into linear languages with name-matching. The key idea is that for each pair of binding names in an input an additional name is added that represents whether or not these bining names are to be equal for the interaction to occur. The encoding of an output then uses conditionals (or guards) to output the appropriate (in)equality of the names in the output.

Non-linear intensionality cannot be encoded, again due to the complexity of the structures allowed by intensional languages. These make it impossible to encode a priori all potential pairs of names (or structures) that may be equal in an output. Thus non-linear intensionality cannot be encoded into linear intensionality, and this proves that non-linear intensional languages are the most expressive languages considered here.

The rest of this paper details all these relations, and explores all possible encodings and separation results. This exhaustive coverage informs on which languages can and cannot encode one another, and also on which languages have more expressive power. Furthermore, along the way various tools for showing general separation results, or encoding approaches are developed.

Note that some of the results considered here have been partially presented in [23, 17]. This work revisits a few of these and details them in Sections 3, 5, 6 & 7 as preliminaries to the more extensive results required for the whole of this work.

Motivation

The results in this work show that intensionality provides significant expressiveness when considering process calculi. This section briefly overviews some other motivations for intensional process calculi beyond expressiveness.

Intensionality was shown to increase the computational expressive power in the sequential setting [31]. 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 [31]. When representing sequential computation in process calculi, intensional process calculi allows for both: expressing intensionality in sequential computation [15], and for more elegantly representing Turing Machines [16].

Cryptography, protocols, and security have proved motivating for Spi Calculus [1] and pattern-matching Spi Calculus [26], both of which introduce intensionality, the latter in communications as considered here. However, the intensionality presented here is too strong to support encryption (in the style of Spi Calculus) since it allows cracking the 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 [26, 14].

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 calculus [3]. 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 all intensional languages have the same 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 [14, 20]. Indeed, [14] demonstrates how intensionality can be used to capture the pattern-matching of functional programming and data type constraints with more granularity than a type system.

The structure of the paper is as follows. Section 2 introduces the forty calculi considered here. Section 3 revises the criteria used for encoding and comparing calculi. Section 4 provides an overview of this work and revisits previous results. Section 5 explores synchronism into intensionality. Section 6 details arity into intensionality. Section 7 formalises communication-medium into intensionality. Section 8 presents identity encodings and proves their composability. Section 9 proves that intensionality can encode all other features and non-linear intensional languages are the most expressive. Section 10 proves the remaining existing encodings between non-intensional and non-linear languages. Section 11 begins the impossibility results by showing when non-linear languages cannot be encoded by linear languages. Section 12 presents the impossibility of encoding linear intensionality into any other non-intensional language. Section 13 completes the impossibility results for the non-linear intensional languages. Section 14 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 and output data. Here these need to be generalised to account for structure. Thus define *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 use a subset of names, called *binding names* and denoted $x, y, z, \ldots$, as in the π-calculus.

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*, denoted $\ulcorner a \urcorner$. Lastly the *intensional* patterns, denoted $p, q, \ldots$ also consider structure and are defined by

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

The binding names x and name-matches $\ulcorner a \urcorner$ are contained in m, compounds and names in s, and the *compound pattern* $p \bullet q$ combines p and q into a single pattern. Both compound terms and compound patterns are left associative.

The (parametric) syntax for the languages is:

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

The different languages are obtained by replacing the output $\mathit{OutProc}$ and input $\mathit{IN}.P$ with the various definitions. The rest of the process forms as are usual: $\mathbf{0}$ denotes the null process; restriction $(\nu a)P$ restricts the visibility of a 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 [25, 14, 16, 15, 21, 22].

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). Denote by $\mathcal{L}_{\alpha,\beta,\gamma,\delta}$ an instance of one of these languages 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.
- $\delta = \mathit{NO}$ for no-matching capability, $\delta = \mathit{NM}$ for name-matching, and $\delta = I$ for intensionality.

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:

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

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.

Polyadic or intensional languages also support another feature: *linearity* (linear vs non-linear inputs). A language is *linear* if each binding name of any input can appear exactly once. Languages that allow a binding name to occur more than once in an input are *non-linear*. Non-linear variations of languages will be here denoted by changing the pattern-matching notation, with non-linear being NO^+ for no-matching, NM^+ for name-matching, and I^+ for intensionality. Observe that non-intensional monadic languages are linear by definition.

Define the set of *bound names* of a process, denoted $\mathsf{bn}(P)$, as follows:

$$\begin{aligned}
\mathsf{bn}(\mathbf{0}) = \mathsf{bn}(\surd) &= \emptyset \\
\mathsf{bn}(OUT.P) = \mathsf{bn}(*P) &= \mathsf{bn}(P) \\
\mathsf{bn}((\nu a)P) &= \{a\} \cup \mathsf{bn}(P) \\
\mathsf{bn}(P|Q) = \mathsf{bn}(\textbf{if } s = t \textbf{ then } P \textbf{ else } Q) &= \mathsf{bn}(P) \cup \mathsf{bn}(Q) \\
\mathsf{bn}(s(\widetilde{p}).P) = \mathsf{bn}((\widetilde{p}).P) &= \bigcup_{p_i \in \widetilde{p}} \mathsf{bn}(p_i) \cup \mathsf{bn}(P)
\end{aligned}$$

and the set of *free names* of a process, denoted $\mathsf{fn}(P)$, as follows:

$$\begin{aligned}
\mathsf{fn}(\mathbf{0}) = \mathsf{fn}(\surd) &= \emptyset \\
\mathsf{fn}((\nu a)P) &= \mathsf{fn}(P) \setminus \{a\} \\
\mathsf{fn}(P|Q) &= \mathsf{fn}(P) \cup \mathsf{fn}(Q) \\
\mathsf{fn}(*P) &= \mathsf{fn}(P) \\
\mathsf{fn}(\textbf{if } s = t \textbf{ then } P \textbf{ else } Q) &= \mathsf{fn}(P) \cup \mathsf{fn}(Q) \cup \mathsf{fn}(s) \cup \mathsf{fn}(t) \\
\mathsf{fn}(s(\widetilde{p}).P) &= \Big(\bigcup_{p_i \in \widetilde{p}} \mathsf{fn}(p_i) \cup \mathsf{fn}(s) \Big) \cup \Big(\mathsf{fn}(P) \setminus \bigcup_{p_i \in \widetilde{p}} \mathsf{bn}(p_i) \Big) \\
\mathsf{fn}((\widetilde{p}).P) &= \Big(\bigcup_{p_i \in \widetilde{p}} \mathsf{fn}(p_i) \Big) \cup \Big(\mathsf{fn}(P) \setminus \bigcup_{p_i \in \widetilde{p}} \mathsf{bn}(p_i) \Big) \\
\mathsf{fn}(\overline{s}\langle \widetilde{t} \rangle.P) &= \bigcup_{t_i \in \widetilde{t}} \mathsf{fn}(t_i) \cup \mathsf{fn}(s) \cup \mathsf{fn}(P) \\
\mathsf{fn}(\langle \widetilde{t} \rangle.P) &= \bigcup_{t_i \in \widetilde{t}} \mathsf{fn}(t_i) \cup \mathsf{fn}(P)
\end{aligned}$$

where the set of *bound* and *free names* of patterns and terms are as follows:

$$\begin{aligned}
\mathsf{bn}(p \bullet q) &= \mathsf{bn}(p) \cup \mathsf{bn}(q) \\
\mathsf{bn}(x) &= \{x\} \\
\mathsf{bn}(\ulcorner a \urcorner) &= \emptyset \\
\mathsf{fn}(s \bullet t) &= \mathsf{fn}(s) \cup \mathsf{fn}(t) \\
\mathsf{fn}(p \bullet q) &= \mathsf{fn}(p) \cup \mathsf{fn}(q) \\
\mathsf{fn}(\ulcorner a \urcorner) = \mathsf{fn}(a) &= \{a\} \\
\mathsf{fn}(x) &= \emptyset
\end{aligned}$$

We assume that free and bound names of a process are distinct. We write $\mathsf{names}(P) = \mathsf{bn}(P) \cup \mathsf{fn}(P)$ the set of names in P.

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. Given a substitution σ and a process P, denote with σP the capture-avoiding application of σ to P that is defined in the usual manner:

$$\begin{aligned}
\sigma(\mathbf{0}) &= \mathbf{0} \\
\sigma(\surd) &= \surd \\
\sigma(\overline{s}\langle\widetilde{t}\rangle.P) &= \overline{\sigma(s)}\langle\widetilde{\sigma t}\rangle.\sigma(P) \\
\sigma(s(\widetilde{p}).P) &= \sigma(s)(\widetilde{\sigma p}).\sigma(P) \text{ if } \sigma \text{ avoids } p_i, \text{ for } p_i \in \widetilde{p} \\
\sigma((\nu a).P) &= (\nu a).\sigma(P) \text{ if } \sigma \text{ avoids } a \\
\sigma(P|Q) &= \sigma(P) \mid \sigma(Q) \\
\sigma(\textbf{if } s = t \textbf{ then } P \textbf{ else } Q) &= \textbf{if } \sigma(s) = \sigma(t) \textbf{ then } \sigma(P) \textbf{ else } \sigma(Q) \\
\sigma(*P) &= *\,\sigma(P)
\end{aligned}$$

where σ *avoids* a name a if for all names $b \in \text{domain}(\sigma)$, $a \neq b$ and $a \notin \mathsf{fn}(\sigma(b))$. A substitution σ avoids a pattern p if it avoids all bound names in p. The application of a substitution σ to a pattern or a term is defined as follows:

$$\sigma a = \sigma(a) \text{ if } a \in \text{domain}(\sigma) \qquad \sigma a = a \text{ if } a \notin \text{domain}(\sigma)$$
$$\sigma\ulcorner a \urcorner = \ulcorner(\sigma a)\urcorner \qquad \sigma(p \bullet q) = (\sigma p) \bullet (\sigma q) \qquad \sigma(s \bullet t) = (\sigma s) \bullet (\sigma t)\ .$$

where the name-match syntax can be applied to a term by the following definition:

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

To aid the formalism in the next section, we also define the *constrained union* $\uplus$ of substitutions as follows:

$$\sigma \uplus \rho = \sigma \cup \rho \quad \text{if } \forall a \in (\text{domain}(\sigma) \cap \text{domain}(\rho)) \text{ then } \sigma(a) = \rho(a)\ .$$

where $\cup$ is the union of substitutions. That is, the constrained union is only defined when all names a in the domain of both σ and ρ are mapped to the same term $\sigma(a) = \rho(a)$.

Alpha-conversion, denoted $=_\alpha$, is the smallest equivalence relation which includes the axioms:

$$\begin{aligned}
(\nu a)P &=_\alpha (\nu b)(\{b/a\}P) && \text{if } b \notin \mathsf{fn}(P) \\
s(\widetilde{p}).P &=_\alpha s(\{y/x\}(\widetilde{p}))P && \text{if } y \notin \mathsf{fn}(P) \cup \mathsf{bn}(p) \text{ and } x \in \mathsf{bn}(p).
\end{aligned}$$

Note that, as expected, substitutions cannot be applied on the bound names of a process, and cannot *capture* names, i.e. transform a free name in a process into a bound one. This is the usual reasonable assumption, as capture can always be avoided by exploiting α-equivalence.

Finally, the structural equivalence relation $\equiv$ is defined in Figure 2.

$$P \mid \mathbf{0} \equiv P \qquad P \mid Q \equiv Q \mid P \qquad P \mid (Q \mid R) \equiv (P \mid Q) \mid R$$
$$P \equiv P' \quad \text{if } P =_\alpha P' \qquad (\nu a)\mathbf{0} \equiv \mathbf{0} \qquad (\nu a)(\nu b)P \equiv (\nu b)(\nu a)P$$
$$P \mid (\nu a)Q \equiv (\nu a)(P \mid Q) \quad \text{if } a \notin \mathsf{fn}(P) \qquad *P \equiv P \mid *P\ .$$

Figure 2: Structural equivalence relation.

Remark 2.1. *The languages $\mathcal{L}_{\alpha,\beta,\gamma,\delta}$ 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:*

$$A \leq S \qquad M \leq P \qquad D \leq C \qquad NO \leq NM \leq I \qquad \delta \leq \delta^+\ .$$

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 outputs followed by $\mathbf{0}$*. 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. 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. Lastly, all linear languages are non-linear languages with binding names always being distinct in all input forms.*

2.2. Operational Semantics

The operational semantics of the languages is given here via reductions as in [34, 29]. An alternative style is via a *labelled transition system* (LTS) such as [23]. Here the reduction based style is used for simplicity, as the labels that occur when intensionality is in play are potentially complex. However, the LTS style can be used for intensional languages [2, 14, 18].

Interaction between processes is handled in two parts: first by matching some terms $\widetilde{t}$ with some patterns $\widetilde{p}$, and secondly, by matching the channel names, in the case of channel-based languages. For the first part, we define a *match* rule $\{t /\!\!/ p\}$ of a term t with a pattern p to create a substitution σ:

$$\{t /\!\!/ x\} \stackrel{\text{def}}{=} \{t/x\} \qquad \{s \bullet t /\!\!/ p \bullet q\} \stackrel{\text{def}}{=} \{s /\!\!/ p\} \uplus \{t /\!\!/ q\}$$
$$\{a /\!\!/ \ulcorner a \urcorner\} \stackrel{\text{def}}{=} \{\} \qquad \{t /\!\!/ p\} \quad \text{undefined otherwise.}$$

Any term t can be matched with a binding name x to generate a substitution $\{t/x\}$. 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 $\{t /\!\!/ q\} = \sigma_2$, and the resulting substitution is the constrained union of σ_1 and σ_2. Otherwise the match is undefined.

The *poly-match* rule $\textsc{Match}(\widetilde{t};\widetilde{p})$ determines matching of a sequence of terms $\widetilde{t}$ with a sequence of patterns $\widetilde{p}$, as 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 constrained union $\uplus$ of substitutions is the result. 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 reduction rules given in Figure 3. The base rule is Sync where the P's are omitted in the asynchronous languages, and the s's are omitted for the dataspace-based languages. The Sync rule 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 s) yields a substitution σ, then reduce to (P in the synchronous languages in parallel with) σ applied to Q. The Par rule is the standard rule for one side of a parallel composition (only one is required here due to the Struct rule). The Restr rule allows reductions to occur under a restriction. The Struct rule allows reduction by exploiting the structural equivalence relation (e.g. in combination with the Par rule to reduce the right hand side of a parallel composition). The CondT and CondF are the only non-obvious rules and allow reductions inside conditionals. The formulation of the rules in this manner is to ensure that unintended consequences cannot occur due to treating conditionals structurally. Consider that for any Q we have that $Q \equiv \textbf{if } s = t \textbf{ then } P \textbf{ else } Q$ when $s \neq t$[1]. Consider as well the substitution σ such that $\sigma(s) = \sigma(t)$. This allows that $\sigma Q \equiv \sigma(\textbf{if } s = t \textbf{ then } P \textbf{ else } Q) \equiv \sigma P$ for any P, which is not reasonable. Thus the formulation here of the CondT and CondF reductions to handle conditionals. The only negative consequence of this choice is that to allow reduction of P in $\textbf{if } s = s \textbf{ then } P \textbf{ else } Q$ requires instantiation of Struct to yield a reduction as shown below. (The CondF instantiation for Q in $\textbf{if } s = t \textbf{ then } P \textbf{ else } Q$ when $s \neq t$ is obtained similarly.)

$$(\textbf{if } s = s \textbf{ then } P \textbf{ else } Q) \equiv (\mathbf{0} \mid \textbf{if } s = s \textbf{ then } P \textbf{ else } Q)$$

$$\textsc{Struct}\;\frac{\wedge\quad \textsc{CondT}\;\dfrac{s = s \qquad \mathbf{0} \mid P \longmapsto P'}{\mathbf{0} \mid \textbf{if } s = s \textbf{ then } P \textbf{ else } Q \longmapsto P'} \qquad \wedge\quad P' \equiv P'}{\textbf{if } s = s \textbf{ then } P \textbf{ else } Q \longmapsto P'}$$

[1]Such rules for the structural congruence have appeared in related works [23] and may cause errors of the type illustrated here.

$$\textsc{Sync}\ \frac{}{\overline{s}\langle\widetilde{t}\rangle.P \mid s(\widetilde{p}).Q \longmapsto P \mid \sigma Q}\ \textsc{Match}(\widetilde{t};\widetilde{p}) = \sigma \qquad \textsc{Par}\ \frac{P \longmapsto P'}{P \mid Q \longmapsto P' \mid Q}$$

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

$$\textsc{CondT}\ \frac{s = t \qquad P \mid Q \longmapsto S}{P \mid \textbf{if } s = t \textbf{ then } Q \textbf{ else } R \longmapsto S} \qquad \textsc{CondF}\ \frac{s \neq t \qquad P \mid R \longmapsto S}{P \mid \textbf{if } s = t \textbf{ then } Q \textbf{ else } R \longmapsto S}$$

Figure 3: Reduction Rules.

Note that the rule

$$\frac{s = s \qquad P \longmapsto P'}{\textbf{if } s = s \textbf{ then } P \textbf{ else } Q \longmapsto P'}$$

does not allow interaction between P (or Q in the false version of the rule) and other processes without either: additional rules in the structural congruence relation with potential side conditions to avoid name capture and scoping; or additional reduction rules similar to those used here.

Define $\longmapsto\!\!\!\!\!\!\Rightarrow$ to be the reflexive transitive closure of the reduction relation $\longmapsto$.

Observe that in intensional languages compound terms $s \bullet t$ may appear in the place of the traditional channel-"name" of π-calculi, and in the equality test **if** $s = t$ **then** $\cdot$ **else** $\cdot$. In both scenarios the syntactic equality of the terms is used where the traditional use of syntactic equality of names would have been used. That is, $\overline{s \bullet t}\langle\rangle$ synchronizes with $t'()$ only if t' has the same syntactic form as $s \bullet t$. Similarly, **if** $s \bullet t = t'$ **then** P **else** Q interacts as P only if t' has the same syntactic form as $s \bullet t$.

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). To avoid name-capture, we assume that for any process P plugged into a context C, $\mathsf{fn}(P) \cap \mathsf{bn}(C) = \emptyset$. Whenever it does not hold, suffices to do an alpha-conversion on the context.

Lastly, for each language let $\simeq$ denote a reference behavioural equivalence. Note that some prior results have explicitly chosen *(strong) barbed congruence* as the reference behavioural equivalence and therefore the same choice is made here when recalling these results. For the non-intensional languages barbed congruences are well known, either from their equivalent language in the literature, such as asynchronous/synchronous monadic/polyadic π-calculus, or from [23]. For the intensional languages the results in [18] can be used.

Frequently the results here rely only upon the constraint that the reference behavioural equivalence is *reduction sensitive*, not that the reference behavioural equivalence is explicitly strong barded congruence. This distinction will be made explicit in places where the choice is significant later, and revisited in Remark 3.8.

Definition 2.2 (Reduction Sensitive)**.** *An equivalence* $\simeq$ *is* reduction sensitive *whenever* $P \simeq P'$ *and* $P \longmapsto$ *implies that* $P' \longmapsto$.

The rules CondF and CondT occasionally slightly complicate the proofs here. This typically appears when impossibility results rely on processes that exhibit success. If $\surd$ is part of a conditional, the process has to be in parallel with another process to evaluate the conditional and exhibit success (by applying the rule Struct as we have shown above). Similarly, for barbs, we can use the $\mathbf{0}$ process to exhibit barbs under an **if** $s = t$ **then** P **else** Q.

3. Encodings

This section recalls and adapts the definition of valid encodings as well as some useful theorems for formally relating process calculi. The validity of such criteria in developing expressiveness studies emerges from the various works [23, 24, 25], that have also recently inspired works similar to the one presented here [32, 33, 44].

An *encoding* of a language $\mathcal{L}_1$ into another language $\mathcal{L}_2$ is a pair $(\llbracket \cdot \rrbracket, \varphi_{\llbracket \rrbracket})$ where $\llbracket \cdot \rrbracket$ translates every $\mathcal{L}_1$-process into an $\mathcal{L}_2$-process and $\varphi_{\llbracket \rrbracket}$ maps every name (of the source language) into a tuple of k terms (of the target language), for $k > 0$, such that $\varphi_{\llbracket \rrbracket}(u) \cap \varphi_{\llbracket \rrbracket}(v) = \emptyset$, for all $u \neq v$. 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 $\varphi_{\llbracket \rrbracket}$. Encodings frequently exploit the concept of a *fresh name* n for a process P defined such that $n \notin \mathsf{names}(P)$ [23, 24, 25, 32, 33, 44, 28, 5, 43]. When constructing an encoding it is common to exploit $\varphi_{\llbracket \rrbracket}$ to introduce specific names that are *fresh* in that they do not exist in any process or term of the source language [23, 24, 25]. Also an encoding may take into account a set of names N as part of the encoding (as in Definition 3.1 below) in which case a fresh name can be explicitly chosen not to intersect with N [44, 43].

We denote with $\longmapsto_i$ and $\Longmapsto_i$ the relations $\longmapsto$ and $\Longmapsto$ in language $\mathcal{L}_i$, denote with $\longmapsto^k$ a sequence of k reductions and with $\longmapsto^\omega$ an infinite sequence. Moreover, we let $\simeq_i$ denote the reference behavioural equivalence for language $\mathcal{L}_i$. Define a hole $\cdot$ to be *top-level* in a context $C(\cdot)$ if it holds that: if $P \longmapsto P'$ then $C(P) \longmapsto C(P')$ and if $P \mid Q$ by interaction of P and Q then $C(P) \mid Q \longmapsto$ by interaction between P and Q. Also, let $P \Downarrow_i$ mean that there exists P' such that $P \Longmapsto_i P'$ and $P' \equiv P'' \mid \surd$, for some P''. Finally, to simplify reading, let S range over processes of the source language (viz., $\mathcal{L}_1$) and T range over processes of the target language (viz., $\mathcal{L}_2$).

Definition 3.1 (Valid Encoding)**.** *An encoding* $(\llbracket \cdot \rrbracket, \varphi_{\llbracket \rrbracket})$ *of* $\mathcal{L}_1$ *into* $\mathcal{L}_2$ *is* valid *if it satisfies the following five properties:*

1. Compositionality: *for every* k*-ary operator* op *of* $\mathcal{L}_1$ *and for every subset of names* N*, there exists a* k*-ary context* $C^N_{\mathsf{op}}(\__1; \ldots; \__k)$ *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 name substitution σ there exists σ' such that for every process S it holds that*

$$[\![\sigma S]\!] \begin{cases} = \sigma'[\![S]\!] & \text{if } \sigma \text{ is injective} \\ \simeq_2 \sigma'[\![S]\!] & \text{otherwise} \end{cases}$$

and $\varphi_{[\![ ]\!]}(\sigma(a)) = \sigma'(\varphi_{[\![ ]\!]}(a))$ for every name a.
3. Operational correspondence:
   - *for all $S \Longmapsto_1 S'$, it holds that $[\![S]\!] \Longmapsto_2 \simeq_2 [\![S']\!]$;*
   - *for all $[\![S]\!] \Longmapsto_2 T$, there exists S' such that $S \Longmapsto_1 S'$ and $T \Longmapsto_2 \simeq_2 [\![S']\!]$.*
4. Divergence reflection: *for every S such that $[\![S]\!] \longmapsto_2^\omega$, it holds that $S \longmapsto_1^\omega$.*
5. Success sensitiveness: *for every S, it holds that $S \Downarrow_1$ if and only if $[\![S]\!] \Downarrow_2$.*

As noted in [24], valid encodings are not composable, but a restriction on the definition of valid encodings, presented below, ensures composability (Proposition 4.19 from [24]). All encodings we consider in this paper satisfy this restriction, and therefore we can rely on existing encodings and their composition in our proofs.

Definition 3.2 (Composable encodings)**.** *An encoding $([\![\cdot]\!], \varphi_{[\![ ]\!]})$ of $\mathcal{L}_1$ into $\mathcal{L}_2$ is* composable *if it satisfies the properties 1,2,4 and 5 from Definition 3.1 and the following properties:*

- Operational Correspondence revisited:
  - *for all $S \Longmapsto_1 S'$, it holds that $[\![S]\!] \Longmapsto_2 [\![S']\!] \mid T$, for some $T \simeq_2 \mathbf{0}$;*
  - *for all $[\![S]\!] \Longmapsto_2 T$, there exists S' such that $S \Longmapsto_1 S'$ and $T \Longmapsto_2 [\![S']\!] \mid T'$, for some $T' \simeq_2 \mathbf{0}$.*
- preserves the equivalence class of $\mathbf{0}$: *for every $S \simeq_1 \mathbf{0}$, $[\![S]\!] \simeq_2 \mathbf{0}$.*
- is homomorphic w.r.t. $|$: *for every S_1, S_2, $[\![S_1 \mid S_2]\!] = [\![S_1]\!] \mid [\![S_2]\!]$.*

The rest of this section recalls results concerning valid encodings that will be exploited later for proving relations between languages. The first such result is an adaptation from [25] used to demonstrate that introducing reductions via the encoding can yield divergence and thus violate a valid encoding.

Proposition 3.3 (Proposition 5.4. from [25])**.** *Let $[\![\cdot]\!]$ be a valid encoding and let $\simeq_2$ be reduction sensitive; then, $S \mapsto\!\!\!\!\!\!\rightarrow_1$ implies that $[\![S]\!] \mapsto\!\!\!\!\!\!\rightarrow_2$.*

Proof. By contradiction, assume that $[\![S]\!] \longmapsto_2 T$, for some $S \mapsto\!\!\!\!\!\!\rightarrow_1$. By operational correspondence, there exists an S' such that $S \Longmapsto_1 S'$ and $T \Longmapsto_2 T' \simeq_2 [\![S']\!]$; but the only such S' is S itself. Since $\simeq_2$ is reduction-sensitive and since $[\![S']\!] = [\![S]\!] \longmapsto_2$, then $T' \longmapsto_2 T''$. Again by operational correspondence $T'' \Longmapsto_2 T''' \simeq_2 [\![S]\!]$, and so on; thus, $[\![S]\!] \longmapsto_2 T \Longmapsto_2 T' \Longmapsto_2 T'' \Longmapsto_2 T''' \longmapsto_2 \ldots$, in contradiction with divergence reflection (since $S \mapsto\!\!\!\!\!\!\rightarrow_1$ implies that S does not diverge). □

To aid in the following results define the notation $block(S)$ that denotes the process $(\nu n)(\nu m)$**if** $n = m$ **then** S where $n, m \notin \mathsf{fn}(S)$. The goal of $block(S)$ is to have a process with the same names as S but which cannot perform any of the reductions or interactions of S.

The next result (also an adaptation from [25]) shows that a valid encoding must maintain encoded processes in a context where they can still interact with other encoded processes.

Proposition 3.4 (Proposition 5.5 from [25])**.** *Let $[\![\cdot]\!] : \mathcal{L}_1 \to \mathcal{L}_2$ be a valid encoding and let $\simeq_2$ be reduction sensitive; then for every set of names N, it holds that $C_{|}^{N}(\cdot_1, \cdot_2)$ has both its holes at top-level.*

Proof. Fix a set of names N and a process S with $\mathsf{fn}(S) = N$. Now consider $S' = \surd \mid block(S)$. By Proposition 3.3 it must be that $[\![S']\!] \mapsto\!\!\!\!\!\!\rightarrow_2$, since $S' \mapsto\!\!\!\!\!\!\rightarrow_1$. By compositionality we must have $[\![S']\!] = C_{|}^{N}([\![\surd]\!], [\![block(S)]\!])$. By success sensitiveness it must be that $[\![S']\!] \Downarrow_2$ since $S' \Downarrow_1$. All these facts entail that the top-level occurrence of $\surd$ in $[\![S']\!]$ is exhibited:

1. either by the translating context and so $C_{|}^{N}(\cdot, \cdot) \Downarrow_2$
2. or by $[\![\surd]\!]$, but this implies that $C_{|}^{N}(\cdot, \cdot)$ has the first hole $\cdot$ at top-level.

Indeed, it is not possible that $\surd$ is exhibited by $[\![block(S)]\!]$ since $block(S) \not\Downarrow_1$. However, the first case is not possible otherwise $[\![block(S) \mid block(S)]\!] \Downarrow_2$, whereas $block(S) \mid block(S) \not\Downarrow_1$. To show that the second hole in $C_{|}^{N}(\cdot, \cdot)$ is at top-level it suffices to reason in the same way using $S' = block(S) \mid \surd$. □

The following result (again an adaptation from [25]) maintains that if two processes can only report success by an interaction, then a valid encoding must also allow their interaction.

Proposition 3.5 (Proposition 5.6. from [25])**.** *Let $[\![\cdot]\!] : \mathcal{L}_1 \to \mathcal{L}_2$ be a valid encoding and $\simeq_2$ be reduction sensitive. If there exist two $\mathcal{L}_1$ terms S_1 and S_2 such that $S_1 \mid S_2 \Downarrow_1$, with $S_i \not\Downarrow_1$ and $S_i \mapsto\!\!\!\!\!\!\rightarrow_1$ for $i = 1, 2$, then $[\![S_1 \mid S_2]\!] \longmapsto_2$ by a synchronization between $[\![S_1]\!]$ and $[\![S_2]\!]$.*

Proof. By success sensitiveness $[\![S_1 \mid S_2]\!] \Downarrow_2$ and by Proposition 3.4 $C_{|}^{N}([\![S_1]\!], [\![S_2]\!])$ has both $[\![S_1]\!]$ and $[\![S_2]\!]$ at top-level. However, since none of $[\![S_1]\!]$, $[\![S_2]\!]$, and $[\![block(S_1) \mid block(S_2)]\!]$ can report success, it must be the case that $[\![S_1 \mid S_2]\!] \longmapsto_2$. This can only happen by interaction between $[\![S_1]\!]$ and $[\![S_2]\!]$. If this was not the case, we would have $[\![S_1 \mid block(S_2)]\!] \longmapsto_2$ or $[\![block(S_1) \mid S_2]\!] \longmapsto_2$ or $[\![block(S_1) \mid block(S_2)]\!] \longmapsto_2$, in violation of Proposition 3.3: indeed $S_1 \mid block(S_2) \mapsto\!\!\!\!\!\!\rightarrow_1$ because $S_1 \mapsto\!\!\!\!\!\!\rightarrow_1$, $block(S_2) \mapsto\!\!\!\!\!\!\rightarrow_1$ and $block(S_2)$ cannot interact with S_1. Similar reasoning holds for $block(S_1) \mid S_2$ and $block(S_1) \mid block(S_2)$. □

The two following lemmas formally state reasoning we exploit in the imposibility results of this work. The first lemma is used to reason about the actions involved in synchronisations in valid encodings.

Lemma 3.6. *Let $[\![\cdot]\!] : \mathcal{L}_1 \to \mathcal{L}_2$ be a valid encoding and $\simeq_2$ be reduction sensitive. Consider the $\mathcal{L}_1$ terms S_1 and S_2 and S_3 such that the following conditions hold:*

1. $S_1 \mid S_2 \longmapsto_1 R_1$ *and* $S_1 \mid S_3 \longmapsto_1 R_2$ *for two terms* R_1, R_2, *where* S_1 *interacts using the same action for both reductions;*
2. $S_1 \mid S_i \longmapsto$ *has only one possible reduction, for* $i = 2, 3$*;*
3. $S_i \Downarrow_1$ *and* $S_i \not\mapsto_1$, *for* $i = 1, 2, 3$*;*
4. $R_i \Downarrow_1$ *and* $R_i \not\mapsto_1$, *for* $i = 1, 2$*;*
5. $R_1 \mid S_1 \not\mapsto_1$ *and* $R_1 \mid S_3 \not\mapsto_1$ *and* $R_2 \mid S_1 \not\mapsto_1$ *and* $R_2 \mid S_2 \not\mapsto_1$*;*
6. $S_2 \not\approx_1 S_3$.

Then $[\![S_1 \mid S_2]\!] \longmapsto_2$ *by a synchronization between* $[\![S_1]\!]$ *and* $[\![S_2]\!]$ *and* $[\![S_1 \mid S_3]\!] \longmapsto_2$ *by a synchronization between* $[\![S_1]\!]$ *and* $[\![S_3]\!]$. *Moreover* $[\![S_1]\!]$ *uses the same action for both reductions.*

Proof. From Proposition 3.5 and conditions 1, 3 & 4 above we have that $[\![S_1 \mid S_2]\!] \longmapsto_2$ and $[\![S_1 \mid S_3]\!] \longmapsto_2$ using a synchronisation between $[\![S_1]\!]$ and $[\![S_2]\!]$ or $[\![S_3]\!]$, respectively. Observe that by condition 2 above there is only one possible reduction of $S_1 \mid S_2$ and $S_1 \mid S_3$ that can respectively correspond to these reductions and so they must be those of condition 1 and share the same action in S_1. Suppose w.l.o.g. that $[\![S_1]\!]$ does an input and both $[\![S_2]\!]$ and $[\![S_3]\!]$ each do an output in the reductions $[\![S_1 \mid S_2]\!] \longmapsto$ and $[\![S_1 \mid S_3]\!] \longmapsto$.

As both reductions are initial for $[\![S_1]\!]$ there are three possibilities.

- The two inputs in $[\![S_1]\!]$ are concurrent and therefore the two can occur in any order. Let us now consider the process T obtained after $[\![S_1 \mid S_2 \mid S_3]\!]$ does the reduction between $[\![S_1]\!]$ and $[\![S_2]\!]$. Note that due to condition 6 the rest of this case assumes we can distinguish S_2 and S_3 and their derivatives. From operational correspondence it follows that $S_1 \mid S_2 \mid S_3$ reduces to a process S such that $T \Longmapsto [\![S]\!] \mid T'$ for some T' and S is of the form $S = R_1 \mid S_3$ (or $S = R_2 \mid S_2$). Indeed the other possibility, as ensured by condition 3 & 4 above, is for $S = S_1 \mid S_2 \mid S_3$ and this leads to a divergence in the encoded process and breaks the validity of the encoding. Remember that we assumed that T is obtained after reduction between $[\![S_1]\!]$ and $[\![S_2]\!]$. Also, we have that $T \Longmapsto [\![S]\!] \mid T'$. We have the following two subcases depending on whether the reduction between $[\![S_1]\!]$ and $[\![S_3]\!]$ occurs in $T \Longmapsto [\![S]\!] \mid T'$ or in a later reduction.
  - The reduction between $[\![S_1]\!]$ and $[\![S_3]\!]$ is included in $T \Longmapsto [\![S]\!] \mid T'$. We have then $T \Longmapsto [\![R_1 \mid S_3]\!] \mid T'$ but where the output of $[\![S_3]\!]$ was consumed. Therefore T must be able to undo the output in the encoding of $[\![S_3]\!]$ due to validity of the encoding maintaining $\simeq [\![S_3]\!]$.
    However, this leads to a divergence for the encoding of a process $*S_1 \mid S_2 \mid S_3$. This breaks the validity of the encoding as $*S_1 \mid S_2 \mid S_3$ cannot diverge as ensured by condition 5 above.
  - Otherwise, let us consider R the process obtained from $[\![S]\!] \mid T'$ after the reduction between $[\![S_1]\!]$ and $[\![S_3]\!]$. From operational correspondence S must also reduce to a process S' such that $R \Longmapsto [\![S']\!] \mid T''$. However the only possible reduction for S is a reflexive one by conditions 3, 4 & 5 above and therefore $S' = S$. Again we are in the situation above where $R \Longmapsto [\![R_1 \mid S_3]\!]$ but where the ouput of $[\![S_3]\!]$ was consumed. As above we can then show a divergence for the process $*S_1 \mid S_3$.

- The two inputs are in conflict due to an **if** $c_1 = c_2$ **then** T_1 **else** T_2 construct in the process $[\![S_1]\!]$: depending on the equality test $c_1 = c_2$ the process does the input of T_1, to interact with $[\![S_2]\!]$ or the input of T_2, to interact with $[\![S_3]\!]$. As both reductions are initial we have then that c_1 and c_2 are both in the free names of S_1. However as both $[\![S_1 \mid S_2 \mid \mathbf{0}]\!]$ and $[\![S_1 \mid \mathbf{0} \mid S_3]\!]$ reduce using rules CondT or CondF, it implies that the conditional $c_1 = c_2$ evaluates the same for both reductions, therefore this case is not possible.

- The only possible case is then for the two reductions to be in conflict in $[\![S_1 \mid S_2 \mid S_3]\!]$ as both use the same input of $[\![S_1]\!]$.

□

The second lemma below reasons about how a substitution can block a synchronisation by affecting an **if** $\cdot = \cdot$ **then** $\cdot$ **else** $\cdot$ construct.

Lemma 3.7. *Let $[\![\cdot]\!] : \mathcal{L}_1 \to \mathcal{L}_2$ be a valid encoding and $\simeq_2$ be reduction sensitive. Consider S_1 and S_2 to be two $\mathcal{L}_1$ terms such that $[\![S_1 \mid S_2]\!] \longmapsto_2$ by a synchronization between $[\![S_1]\!]$ and $[\![S_2]\!]$ and such that $[\![S_i]\!] \not\mapsto_2$, for $i = 1, 2$. Let $\sigma = \{a/b, b/a\}$ be a substitution for a, b any two names such that $[\![\sigma S_1]\!]$ does not synchronise with $[\![S_2]\!]$. The synchronisation cannot be blocked by an* **if** $c_1 = c_2$ **then** T_1 **else** T_2 *construct in $[\![S_1]\!]$, for any names c_1, c_2 and processes T_1, T_2.*

Proof. Let σ' be the substitution given by name invariance of a valid encoding such that $[\![\sigma S_1]\!] = \sigma' [\![S_1]\!]$. Suppose by contradiction that **if** $c_1 = c_2$ **then** T_1 **else** T_2 blocks the synchronisation in $[\![\sigma S_1]\!]$. Then the equality test is affected by the substitution σ' as otherwise the contradiction is immediate. Note then that c_1 and c_2 are both in the free names of $[\![S_1]\!]$. We proceed by cases on whether $c_1 = c_2$ holds or not before the substitution.

If $c_1 = c_2$ holds then it must be due to a syntactic equality. This follows from the fact that the the reduction $[\![S_1 \mid S_2]\!]$ is the first (non reflexive) reduction of the process. Then $\sigma'(c_1) = \sigma'(c_2)$ must hold as well. Therefore the substitution does not change the result of the equality test and this cannot be the case.

If $c_1 \neq c_2$ then it must be that $\sigma'(c_1) = \sigma'(c_2)$ since if $\sigma'(c_1) \neq \sigma'(c_2)$ the application of σ' does not change the conditional. Further, observe that $\sigma\sigma S_1 = S_1$ and so there exists σ' and σ'' two substitutions such that $\sigma''\sigma' [\![S_1]\!] = [\![S_1]\!]$ by name invariance and validity of the encoding and injectivity of σ. However, this yields contradiction since (as in the previous case) it is impossible for σ'' to undo the syntactic equality of $\sigma'(c_1) = \sigma'(c_2)$, and so $\sigma''\sigma'(c_1) = \sigma''\sigma'(c_2)$ while $c_1 \neq c_2$. □

We conclude the section with a remark on the role of behavioral equivalences in the encodings.

Remark 3.8. *The valid encodings proposed in [23] and the ones proposed here, all satisfy a stronger definition of valid encodings. The operational correspondence is strengthened as follows:*

- *for all $S \Longmapsto_1 S'$, it holds that $[\![S]\!] \Longmapsto_2 [\![S']\!]$;*

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

In particular observe that the behavioural equivalence relation $\simeq_2$ *does not appear in the above stronger operational correspondence.*

Also, all the encodings satisfy a reformulation of name invariance: for every S *and name substitution* σ*, it holds that* $[\![\sigma S]\!] = \sigma[\![S]\!]$*, that is the substitutions are not modified by the encodings. (Under the assumption that any reserved names introduced by a valid encoding do not appear in the source processes or in* σ*.) Therefore showing that an encoding is valid does not involve any constraints on the behavioral equivalences (not even the reduction sensitive assumption).*

However, constraints on the behavioral equivalence are imposed by the composability results (which asks for barbed congruence), the impossibility results in [23] (which we revisit in Section 4 and also ask for barbed congruence) and our impossibility results (which rely on Propositions 3.3 & 3.5 and therefore ask for a reduction sensitive equivalence).

As an example, consider an encoding that is the identity on all process constructors between a language $\mathcal{L}$ *to itself. We take* $\simeq_1$ *to be the largest equivalence (it contains all terms) of* $\mathcal{L}$ *and* $\simeq_2$ *to be the structural congruence. Then the encoding from* $\mathcal{L}$ *with* $\simeq_1$ *to* $\mathcal{L}$ *with* $\simeq_2$ *is valid but not composable.*

4. Overview

The diagram in Figure 4 shows the valid encodings between the languages that we consider in this paper: if there are no arrows between two languages than there is no valid encoding between them. A part of the diagram is from [Figure 1 of [23]]. In this paper we show the results related to the intensional languages: the green arrows in Figure 4 are shown in Sections 5, 6, 7, 9, and the imposibility results in Sections 12, 13; and also results related to non-linear languages: the blue arrows are results shown in Section 10 and the corresponding imposibility results in Section 11.

This section revisits the results of [23] that are used here. Since prior results for slightly different languages are exploited here, some constraints must be imposed. To be more precise, the following constraints are necessary for the lower part of the diagram and for impossibility results here, but not for the valid encodings proposed here.

A first constraint is that all polyadic channel-based languages considered are typed (see Remark 2.2 of [23]). The typing system is arbitrary, as long as the arity of each channel is fixed. Without this constraint the arrow from $\mathcal{L}_{S,P,C,NO}$ to $\mathcal{L}_{S,M,C,NO}$ would not hold.

A second constraint is that the encoding from $\mathcal{L}_{S,M,D,NM}$ to $\mathcal{L}_{S,M,C,NO}$ (the dashed arrow in the diagram) satisfies a weaker version of operational correspondence than the one in Definition 3.1. The author argues in [23] that the encoding is still pertinent and thus it is kept in the diagram. However, the dashed arrow can be replaced with a straight one if instead the encoding from $\mathcal{L}_{S,M,D,NM}$ to $\mathcal{L}_{S,M,C,NM}$ is considered (which follows from Remark 2.1). As the lower part of the diagram does not involve the main contributions of this work, they shall be kept as in [23].

Lastly, barbed congruence is considered the reference behavioral equivalence in [23], and thus is also chosen here (see Section 2).

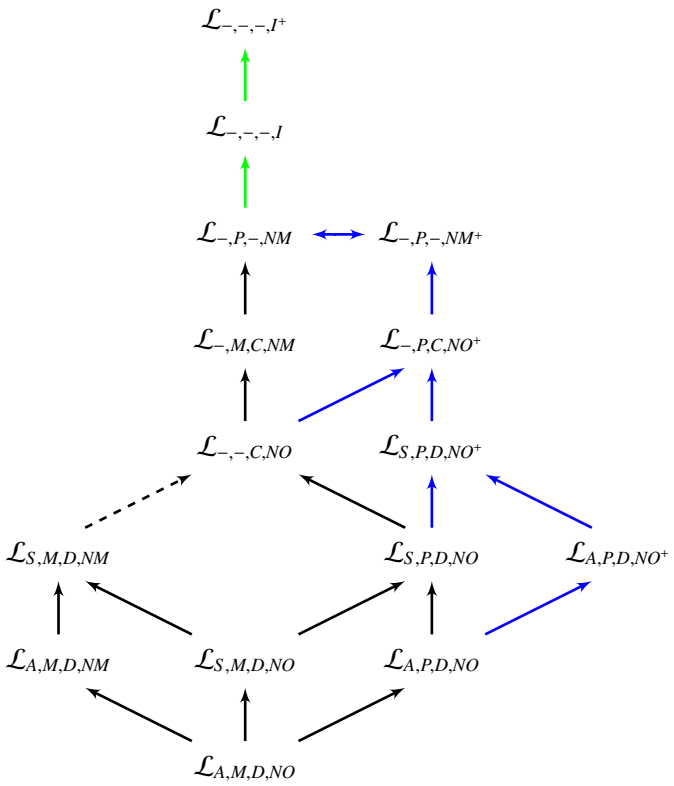

Figure 4: Relations between all languages

The rest of this section revisits the encodings proposed by [23] to show that they are composable (Definition 3.2) and that they satisfy the remark made at the end of Section 3. For the moment we are only considering *non-identity* encodings, identity encodings are considered in Section 9. The identity encodings are the ones covered by Remark 2.1.

The following encodings are homomorphic on all constructors except on the input and output. The encoding from $\mathcal{L}_{S,P,C,NM}$ to $\mathcal{L}_{S,P,D,NM}$ is defined as follows:

$$[\![\, a(\widetilde{m}).P \,]\!] \stackrel{\text{def}}{=} (\ulcorner a \urcorner, \widetilde{m}).[\![\, P \,]\!] \qquad [\![\, \overline{a}\langle\widetilde{b}\rangle.P \,]\!] \stackrel{\text{def}}{=} \langle a, \widetilde{b}\rangle.[\![\, P \,]\!].$$

Theorem 4.1. *The encoding from $\mathcal{L}_{S,P,C,NM}$ to $\mathcal{L}_{S,P,D,NM}$ is composable.*

Proof. Operational correspondence of Definition 3.2 follows from Remark 3.8 and from $[\![\, \mathbf{0} \,]\!] \simeq_2 \mathbf{0}$. To show that the encoding preserves the equivalence class of $\mathbf{0}$, we proceed by contradiction and we suppose that $S \simeq_1 \mathbf{0}$ and that $[\![\, S \,]\!] \not\simeq_2 \mathbf{0}$. Then there exists a context C such that $C[[\![\, S \,]\!]] \not\approx C[\mathbf{0}]$, where we write $\approx$ for barbed bisimulation. There are two cases:

- $C[[\![\, S \,]\!]]$ can do a barb $\langle a, \widetilde{b}\rangle$, whereas the process $C[\mathbf{0}]$ cannot. Either the barb comes from the context $C[\cdot]$ (contradiction, as $C[\mathbf{0}]$ cannot exhibit this barb) or

it comes from the process $[\![S]\!]$. By cases on the encoding, it follows that S can do a barb $\overline{a}\langle\widetilde{b}\rangle$ as well. Contradiction, as $S \simeq_1 \mathbf{0}$.

- $C[[\![S]\!]] \longmapsto_2 T$, then first note that the context $C[\cdot]$ cannot reduce by itself. Indeed if the context can reduce by itself then consider the context $C'[\cdot]$ obtained from $C[\cdot] \longmapsto_2^k C'[\cdot]$ and such that $C'[[\![S]\!]] \not\approx C'[\mathbf{0}]$ either because of a barb (and we are back in the case above) or because $C'[[\![S]\!]] \longmapsto_2 T$ and $C'[\mathbf{0}] \not\longmapsto_2$. We therefore have that $C[\cdot] \not\longmapsto_2$.

  Suppose then that $[\![S]\!]$ does the reduction by itself. By cases on the reduction, we have that $S \longmapsto_1$, contradicting the hypothesis $S \simeq_1 \mathbf{0}$. Lastly, the reduction is the result of a synchronisation between the context and the process $[\![S]\!]$. We have that any action (input or output) available for a synchronisation in $[\![S]\!]$ is also available in S. Then $S \not\simeq_1 \mathbf{0}$: there exists a context which allows S to reduce by using its available action, whereas no context can synchornise with $\mathbf{0}$. This concludes as $S \not\simeq_1 \mathbf{0}$ contradicts our initial assumption.

The encoding of the parallel composition is homomorphic which follows by the definition of the encoding above. □

The encoding from typed $\mathcal{L}_{S,P,C,NO}$ to $\mathcal{L}_{S,M,C,NO}$ is defined on input and output as:

$$[\![a(x_1 \cdots x_n).P]\!] \stackrel{\text{def}}{=} a(x).x(x_1).\cdots .x(x_n).[\![P]\!]$$
$$[\![\overline{a}\langle b_1 \cdots b_n\rangle.P]\!] \stackrel{\text{def}}{=} (\nu c)(\overline{a}\langle c\rangle.\overline{c}\langle b_1\rangle.\cdots .\overline{c}\langle b_n\rangle.[\![P]\!]$$

where the type system ensures that the arity of channel a is n and where $x \cap (\{x_1, \ldots, x_n\} \cup \mathsf{fn}(P)) = \emptyset$ and $c \cap (\{a, b_1, \ldots, b_n\} \cup \mathsf{fn}(P)) = \emptyset$.

Theorem 4.2. *The encoding from typed $\mathcal{L}_{S,P,C,NO}$ to $\mathcal{L}_{S,M,C,NO}$ is composable.*

Proof. Let us show that the operational correspondence of Remark 3.8 holds; the preservation of the equivalence class of zero follows as in Theorem 4.1; homomorphism of parallel is by definition of the encoding.

Whenever a process $S \longmapsto_1 S'$ in $\mathcal{L}_{S,P,C,NO}$ it implies that $S \equiv (\nu\tilde{d})(a(x_1 \cdots x_n).P \mid \overline{a}\langle b_1 \cdots b_n\rangle.Q) \mid P'$ and $S' \equiv (\nu\tilde{d})(P\{b_i/x_i\} \mid Q \mid P')$, with $i \leq n$. In the encoding

$$[\![S]\!] = (\nu\tilde{d})([\![a(x_1 \cdots x_n).P]\!] \mid [\![\overline{a}\langle b_1 \cdots b_n\rangle.Q]\!] \mid [\![P']\!])$$

we have that the process $[\![S]\!]$ can do a sequence of n reductions to produce $(\nu\tilde{d})($ $[\![P]\!]\{b_1/x_1\} \cdots \{b_n/x_n\} \mid [\![Q]\!] \mid [\![P']\!])$.

Let us now show the opposite direction. We have that for any T such that $[\![S]\!] \Longmapsto_2$ T, $T \equiv (\nu\tilde{d})(\prod_i (\nu c_i)(C_i \mid \overline{C_i}) \mid [\![P']\!])$ where we write C_i for the process $c_i(x_j).\cdots .c_i(x_{n_i}).[\![P']\!]$, for some j and n_i and some process P'. Informally, C_i stands for a sequence of inputs in the encoded process which uses channel c_i. We write $\overline{C_i}$ for the output counterpart. This follows from

$$S \equiv (\nu\tilde{d})(\prod_i a_i(x_1 \cdots x_{n_i}).P_i \mid \prod_i \overline{a_i}\langle b_1^i \cdots b_{n_i}^i\rangle.Q_i \mid P').$$

We take $S' = (\nu \tilde{d})(\prod_i P_i\{b_j/x_j\} \mid \prod_i Q_i \mid P')$ to be the process in which all available actions in $\prod_i$ are consumed. By reducing T using all available actions in $\prod_i (\nu c_i)(C_i \mid \overline{C_i})$ we obtain

$$\begin{aligned} T \quad &\Longmapsto_2 \quad (\nu \tilde{d})(\prod_i (\nu c_i)([\![P_i]\!]\{b_j/x_j\} \mid [\![Q_i]\!]) \mid [\![P']\!]) \\ &\equiv \quad (\nu \tilde{d})(\prod_i [\![P_i]\!]\{b_j/x_j\} \mid \prod_i [\![Q_i]\!] \mid [\![P']\!]) \end{aligned}$$

as $c_i \notin \mathsf{fn}([\![P_i]\!]) \cup \mathsf{fn}([\![Q_i]\!]) \cup [\![P']\!]$. □

The encoding from $\mathcal{L}_{S,P,D,NO}$ to $\mathcal{L}_{S,M,C,NO}$ is defined as

$$\begin{aligned} [\![(x_1 \cdots x_n).P]\!] &\stackrel{\text{def}}{=} n(x).x(x_1).\cdots .x(x_n).[\![P]\!] \\ [\![\langle b_1 \cdots b_n \rangle.P]\!] &\stackrel{\text{def}}{=} (\nu a)\overline{n}\langle a \rangle.\overline{a}\langle b_1 \rangle.\cdots .\overline{a}\langle b_n \rangle.[\![P]\!]. \end{aligned}$$

where $x \cap (\{x_1, \ldots, x_n\} \cup \mathsf{fn}(P)) = \emptyset$ and $a \cap (\{b_1, \ldots, b_n\} \cup \mathsf{fn}(P)) = \emptyset$.

We can encode $\mathcal{L}_{S,P,C,NM}$ to $\mathcal{L}_{A,P,C,NM}$ as follows:

$$\begin{aligned} [\![a(\widetilde{m}).P]\!] &\stackrel{\text{def}}{=} a(x, \widetilde{m}).(\overline{x}\langle\rangle \mid [\![P]\!]) \\ [\![\overline{a}\langle \widetilde{b} \rangle.P]\!] &\stackrel{\text{def}}{=} (\nu c)(\overline{a}\langle c, \widetilde{b} \rangle \mid c().[\![P]\!]) \end{aligned}$$

where $x \cap (\widetilde{m} \cup \mathsf{fn}(P)) = \emptyset$ and $c \cap (\widetilde{b} \cup \mathsf{fn}(P)) = \emptyset$.

To encode $\mathcal{L}_{S,P,D,NM}$ to $\mathcal{L}_{A,P,D,NM}$ we use the translation:

$$\begin{aligned} [\![(m_1 \cdots m_n).P]\!] &\stackrel{\text{def}}{=} (x, m_1, \cdots, m_n).(\langle x \rangle \mid [\![P]\!]) \\ [\![\langle b_1 \cdots b_n \rangle.P]\!] &\stackrel{\text{def}}{=} (\nu c)(\langle c, b_1, \cdots, b_n \rangle \mid (\ulcorner c \urcorner).[\![P]\!]) \end{aligned}$$

where $x \cap (\{m_1, \ldots, m_n\} \cup \mathsf{fn}(P)) = \emptyset$ and $c \cap (\{b_1, \ldots, b_n\} \cup \mathsf{fn}(P)) = \emptyset$.

Lastly consider the encoding from $\mathcal{L}_{S,M,C,NM}$ to $\mathcal{L}_{A,M,C,NM}$ (explained in details in [23]) below:

$$\begin{aligned} [\![a(x).P]\!] \stackrel{\text{def}}{=} \quad & a_0(x_N).(\overline{a_1}\langle x_N \rangle \mid x_N(c).(\nu z)(\overline{c}\langle z \rangle \\ & \mid z(x_0).(\overline{c}\langle z \rangle \mid z(x_1).[\![P]\!])))) \quad \text{for } c, z \text{ fresh} \\ [\![a(\ulcorner b \urcorner).P]\!] \stackrel{\text{def}}{=} \quad & a_0(\ulcorner b_N \urcorner).(\overline{a_1}\langle b_N \rangle \mid b_N(c).(\nu z)(\overline{c}\langle z \rangle \\ & \mid z(x_0).(\overline{c}\langle z \rangle \mid z(x_1).[\![P]\!])))) \quad \text{for } c, x_0, x_1, z \text{ fresh} \\ [\![\overline{a}\langle b \rangle.P]\!] \stackrel{\text{def}}{=} \quad & \overline{a_0}\langle b_N \rangle \mid a_1(\ulcorner b_N \urcorner).(\nu c)(\overline{b_N}\langle c \rangle \\ & \mid c(z).(\overline{z}\langle b_0 \rangle \mid c(z).(\overline{z}\langle b_1 \rangle \mid [\![P]\!]))) \quad \text{for } c, z \text{ fresh} \\ [\![(\nu a)P]\!] \stackrel{\text{def}}{=} \quad & (\nu a_N)(\nu a_0)(\nu a_1)[\![P]\!] \\ [\![\textbf{if } a = b \textbf{ then } P \textbf{ else } Q]\!] \stackrel{\text{def}}{=} \quad & [\![\textbf{if } a_N = b_N \textbf{ then } P \textbf{ else } Q]\!] \end{aligned}$$

To show composability for these encodings we use a similar proof as in Theorem 4.2.

One might think that the following simple translation would be possible:

$$
\begin{aligned}
[\![\, a(x).P \,]\!] &\stackrel{\text{def}}{=} && a_0(x_N).(\overline{a_1}\langle x_N\rangle \mid x_N(x_0).x_N(x_1).[\![\, P \,]\!]) \\
[\![\, a(\ulcorner b \urcorner).P \,]\!] &\stackrel{\text{def}}{=} && a_0(\ulcorner b_N \urcorner).(\overline{a_1}\langle b_N\rangle \mid b_N(x_0).b_N(x_1).[\![\, P \,]\!]) \text{ for } x_0, x_1 \text{ fresh} \\
[\![\, \overline{a}\langle b\rangle.P \,]\!] &\stackrel{\text{def}}{=} && \overline{a_0}\langle b_N\rangle \mid a_1(\ulcorner b_N \urcorner).(\overline{b_N}\langle b_0\rangle \mid \overline{b_N}\langle b_1\rangle \mid [\![\, P \,]\!]) \\
[\![\, (\nu a)P \,]\!] &\stackrel{\text{def}}{=} && (\nu a_N, a_0, a_1)[\![\, P \,]\!] \\
[\![\, \textbf{if } a = b \textbf{ then } P \textbf{ else } Q \,]\!] &\stackrel{\text{def}}{=} && [\![\, \textbf{if } a_N = b_N \textbf{ then } P \textbf{ else } Q \,]\!]
\end{aligned}
$$

However, consider when

$$[\![\, \overline{a}\langle b\rangle.P_1 \mid \overline{a}\langle b\rangle.P_2 \mid a(x).Q_1 \mid a(y).Q_2 \,]\!]$$

has reduced to the point of

$$
\begin{aligned}
&\overline{b_N}\langle b_0\rangle \mid \overline{b_N}\langle b_1\rangle \mid [\![\, P_1 \,]\!] \mid \overline{b_N}\langle b_0\rangle \mid \overline{b_N}\langle b_1\rangle \mid [\![\, P_2 \,]\!] \\
&\quad \mid b_N(x_0).b_N(x_1).[\![\, Q_1 \,]\!] \mid b_N(y_0).b_N(y_1).[\![\, Q_2 \,]\!]
\end{aligned}
$$

and observe that we can reach the state

$$[\![\, P_1 \,]\!] \mid [\![\, P_2 \,]\!] \mid \{b_0/x_0, b_0/x_1\}[\![\, Q_1 \,]\!] \mid \{b_1/y_0, b_1/y_1\}[\![\, Q_2 \,]\!]$$

which would damage further reductions by messing up b_0 and b_1 in $[\![\, Q_1 \,]\!]$ and $[\![\, Q_2 \,]\!]$. Thus we require new channel names (above c and z) to send and acknowledge parts of the name (here b) and ensure the order is preserved.

We can therefore use the positive results in [23]. The imposibility results hold as well in our framework (even though the formulation of a valid encoding is slightly different), as these results are shown by contradicting Proposition 3.3, operational correspondence or divergence reflection. Thus the rest of the results of [23] are not revisited in detail.

5. Synchronism in Intensionality

This section proves that intensionality is sufficient to encode synchrony. That is, that any language $\mathcal{L}_{S,\beta,\gamma,\delta_1}$ can be encoded into $\mathcal{L}_{A,\beta,\gamma,\delta_2}$ where $I \leq \delta_2$ and $\delta_1 \leq \delta_2$.

The typical approach to encode synchrony into an asynchronous language is to use channels and a fresh name to signal that the output has been received and thus encode synchronicity [28, 5, 23, 43]. The approach here is similar in that a fresh name is used, however the compounds and name-matching of intensionality are used instead of the channel-based communication.

The choice to exploit only intensionality here is to demonstrate that intensionality alone is sufficient for the encoding. Of course when other language features (e.g. channel-based communication) are in play then the traditional approach can be used.

To illustrate, consider the translation $[\![\cdot]\!]$ from $\mathcal{L}_{S,M,D,I}$ into $\mathcal{L}_{A,M,D,I}$ that is the identity on all primitives except for input and output which are as follows:

$$\begin{array}{lcll} [\![(p).P]\!] & \stackrel{\text{def}}{=} & (x \bullet p).(\langle x \rangle \mid [\![P]\!]) & \text{for } x \text{ fresh} \\ [\![\langle t \rangle.Q]\!] & \stackrel{\text{def}}{=} & (\nu a)(\langle a \bullet t \rangle \mid (\ulcorner a \urcorner).[\![Q]\!]) & \text{for } a \text{ fresh.} \end{array}$$

The input is translated into a compound that binds an additional name x as the first component, with the p matching as before. If an interaction occurs, then the term bound to x is output after the interaction, in parallel with $[\![P]\!]$. The output is translated into a compound term that outputs a fresh name a as the first component, compounded with the old term t. This output is in parallel with an input that matches exactly the name a before evolving to $[\![Q]\!]$.

Observe that prior to the translation a reduction between an input and output occurs as follows

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

Here we assumed that $a, x \notin \mathsf{fn}(P) \cup \mathsf{fn}(Q) \cup \mathsf{fn}(p) \cup \mathsf{fn}(t)$ (if this does not hold, it suffices to apply α-conversion). The translation of these two processes can now reduce as follows

$$\begin{array}{lcl} [\![(p).P \mid \langle t \rangle.Q]\!] & = & (x \bullet p).(\langle x \rangle \mid [\![P]\!]) \mid (\nu a)(\langle a \bullet t \rangle \mid (\ulcorner a \urcorner).[\![Q]\!]) \quad (\text{by } [\![\cdot]\!]) \\ & \equiv & (\nu a)((x \bullet p).(\langle x \rangle \mid [\![P]\!]) \mid \langle a \bullet t \rangle \mid (\ulcorner a \urcorner).[\![Q]\!]) \\ & & \qquad (\text{by} \equiv, a \notin \{x\} \cup \mathsf{fn}(p) \cup \mathsf{fn}([\![P]\!])) \\ & \longmapsto & (\nu a)(\{a/x\} \uplus \sigma(\langle x \rangle \mid [\![P]\!]) \mid (\ulcorner a \urcorner).[\![Q]\!]) \\ & & \qquad (\sigma = \textsc{Match}(t; p)) \\ & \equiv & (\nu a)(\langle a \rangle \mid \{a/x\} \uplus \sigma[\![P]\!] \mid (\ulcorner a \urcorner).[\![Q]\!]) \\ & & \qquad (\text{by application of } \{a/x\} \uplus \sigma) \\ & \equiv & (\nu a)(\langle a \rangle \mid \sigma[\![P]\!] \mid (\ulcorner a \urcorner).[\![Q]\!]) \\ & & \qquad (\text{by } x \notin \mathsf{fn}(P)) \\ & \equiv & (\nu a)(\sigma[\![P]\!] \mid \langle a \rangle \mid (\ulcorner a \urcorner).[\![Q]\!]) \qquad (\text{by} \equiv) \\ & \longmapsto & (\nu a)(\sigma[\![P]\!] \mid [\![Q]\!]) \\ & & \qquad (\text{by } \textsc{Match}(\ulcorner a \urcorner; a) = \{\}) \\ & \equiv & \sigma[\![P]\!] \mid [\![Q]\!] \\ & & \qquad (\text{by } a \notin \mathsf{fn}([\![P]\!]) \cup \mathsf{fn}([\![Q]\!]) \ .) \end{array}$$

Observe that the above can be simplified to

$$[\![(p).P \mid \langle t \rangle.Q]\!] \quad \longmapsto\!\!\!\!\longmapsto \quad \sigma[\![P]\!] \mid [\![Q]\!] \qquad\qquad \textsc{Match}(t; p) = \sigma$$

by applications of the proof tree for $\longmapsto$.

This translation approach can be extended to translate any synchronous language $\mathcal{L}_{S,\beta,\gamma,\delta_1}$ into the asynchronous language $\mathcal{L}_{A,\beta,\gamma,\delta_2}$ where $I \leq \delta_2$ and $\delta_1 \leq \delta_2$. Consider the translation $[\![\cdot]\!]$ that is the identity on all primitives except for input and output

which are as follows:

$$\begin{array}{rcll} [\![\, s(p,\widetilde{p}).P \,]\!] & \stackrel{\text{def}}{=} & s(x \bullet p,\widetilde{p}).(\overline{x}\langle x\rangle \mid [\![\, P \,]\!]) & \text{for } x \text{ fresh} \\ [\![\, \overline{s}\langle t,\widetilde{t}\rangle.Q \,]\!] & \stackrel{\text{def}}{=} & (\nu a)(\overline{s}\langle a \bullet t,\widetilde{t}\rangle \mid x(\ulcorner x \urcorner).[\![\, Q \,]\!]) & \text{for } a, x \text{ fresh.} \end{array}$$

When the languages are monadic ($\beta = M$) then the polyadic $\widetilde{p}$ and $\widetilde{t}$ are omitted. When the languages are dataspace-based ($\gamma = D$) then the channel names s are omitted.

As before, the input is translated to bind an additional name x and then output this back to the translated output to signal that the interaction has occurred. Similarly the output restricts a fresh name a 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 5.1. *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 5.2. *For two processes P and Q such that $P \mapsto\!\!\!\!\!\!\rightarrow$ and $Q \mapsto\!\!\!\!\!\!\rightarrow$, then $[\![\, P \mid Q \,]\!] \longmapsto$ if and only if $P \mid Q \longmapsto$.*

Proof. For the forward direction, observe by the definition of the reduction relation that the only possible reduction (rule Sync) must be between an input and an output. Further, observe that the only possible input in the encoding must be from an input in the source language, and similarly an output in the encoding from an output in the source language. Conclude by showing that the interaction relies upon some $\textsc{Match}(y, t; x, p)$ and this implies that $\textsc{Match}(t; p)$ is defined in the source processes and finally $P \mid Q \longmapsto$. The reverse direction is proved similarly. □

Lemma 5.3. *The translation $[\![\, \cdot \,]\!]$ from $\mathcal{L}_{S,\beta,\gamma,\delta_1}$ into $\mathcal{L}_{A,\beta,\gamma,\delta_2}$ (where $I \leq \delta_2$ and $\delta_1 \leq \delta_2$) preserves and reflects reductions. That is:*

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

Proof. Both parts can be proved by a straightforward examination of the derivation tree for the reductions $P \longmapsto P'$ and $[\![\, P \,]\!] \longmapsto Q$. In both cases the base step is rule Sync which follows from Lemma 5.2 and from the definition of $[\![\, \cdot \,]\!]$ and the match rule. For the case where rule Struct is used apply Lemma 5.1. □

Theorem 5.4. *For every language $\mathcal{L}_{S,\beta,\gamma,\delta_1}$ there is a valid encoding into $\mathcal{L}_{A,\beta,\gamma,\delta_2}$ where $I \leq \delta_2$ and $\delta_1 \leq \delta_2$.*

Proof. Compositionality and name invariance hold by definition: substitutions are not modified by the translation. Operational correspondence (with equality = in the place of behavioural equivalence $\simeq$) 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 5.1 we obtain that $[\![P]\!] \longmapsto^{2k} [\![P']\!] \equiv [\![P'']\!] \mid \surd$, i.e. that $[\![P]\!] \Downarrow$. The converse implication can be proved similarly. □

Observe that the results here all use equality = in place of the required behavioural equivalence relation $\simeq$. We would like to differentiate between encodings based on whether = or $\simeq$ is used for proving operational correspondence. Our intuition is that if an encoding exists between two languages that uses = it is preferable to an encoding that uses $\simeq$ as it does not introduce too much spurious content. It is an informal point, nevertheless for each encoding we introduce we will specify whenever it uses = instead of $\simeq$.

Corollary 5.5. *For every language $\mathcal{L}_{S,\beta,\gamma,\delta_1}$ there is a composable valid encoding into $\mathcal{L}_{A,\beta,\gamma,\delta_2}$ where $I \leq \delta_2$ and $\delta_1 \leq \delta_2$.*

Proof. For operational correspondence revisited it suffices to observe that Lemma 5.3 can be trivially restated with

- If $P \longmapsto P'$ then there exists Q such that $[\![P]\!] \longmapsto\!\!\!\!\!\!\longmapsto Q$ and $Q = [\![P']\!] \mid \mathbf{0}$;
- if $[\![P]\!] \longmapsto Q$ then there exists Q' such that $Q \longmapsto Q'$ and $Q' = [\![P']\!] \mid \mathbf{0}$ for some P' such that $P \longmapsto P'$.

and proved easily. Preservation of the equivalence class of $\mathbf{0}$ and homomorphism w.r.t. $\mid$ both hold by construction. □

6. Arity in Intensionality

This section proves that intensionality is sufficient to encode polyadicity. That is, that any language $\mathcal{L}_{\alpha,P,\gamma,\delta_1}$ can be encoded into $\mathcal{L}_{\alpha,M,\gamma,\delta_2}$ where $I \leq \delta_2$ and $\delta_1 \leq \delta_2$.

Encodings of polyadic typed π-calculus in monadic π-calculus [41] or of polyadic dataspace-based communications in monadic channel-based [23] split one reduction with polyadic components into a sequence of reductions, one for each component. Here we use instead the compound of intensionality to encode polyadicity.

The key idea of these encodings is the translation of polyadic input and output into a single pattern or term, respectively. To illustrate, consider the following translation $[\![\cdot]\!]$ from $\mathcal{L}_{A,P,D,I}$ into $\mathcal{L}_{A,M,D,I}$:

$$
\begin{array}{rcl}
[\![(p_1, \ldots, p_i).P]\!] & \stackrel{\text{def}}{=} & (((\ulcorner \mathsf{rn} \urcorner \bullet p_1) \bullet \ldots) \bullet p_i).[\![P]\!] \\
[\![\langle t_1, \ldots, t_i \rangle.P]\!] & \stackrel{\text{def}}{=} & \langle (((\mathsf{rn} \bullet t_1) \bullet \ldots) \bullet t_i \rangle.[\![P]\!]
\end{array}
$$

where rn is a reserved name introduced in the translation. (Note that such a reserved name can be ensured by the renaming policy when constructing a valid encoding [25, 19].) Observe that for both the output and input each of the polyadic components is combined into a single compound, and thus the translation yields a monadic language. Intensionality allows all the polyadic components to be combined, and the match rules ensure they are matched as before in any possible interaction.

The only subtlety in the translation is the introduction of the reserved name rn. This is to ensure that, in the translated terms, every component is checked for a reduction to occur. Observe that if the reserved name was *not* present then the translation of $(x).P$ would be $(x).[\![P]\!]$ and this would be able to match with the translation of $\langle a, b\rangle.Q$ that would yield $\langle a \bullet b\rangle.[\![Q]\!]$. However, these processes do *not* interact before the translation, and this would yield a violation of Proposition 3.3.

More generally, the translation from a language $\mathcal{L}_{\alpha,P,\gamma,\delta_1}$ into a language $\mathcal{L}_{\alpha,M,\gamma,\delta_2}$ where $I \leq \delta_2$ and $\delta_1 \leq \delta_2$ can be defined as follows:

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

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

Theorem 6.1. *For every language $\mathcal{L}_{\alpha,P,\gamma,\delta_1}$ there is a valid encoding into $\mathcal{L}_{\alpha,M,\gamma,\delta_2}$ where $I \leq \delta_2$ and $\delta_1 \leq \delta_2$.*

Proof. The proof proceeds in the same manner as that of Theorem 5.4, although the adaptation of Lemma 5.2 requires accounting for the new structure and the adaptation of Lemma 5.3 has only a single reduction after translation (not two). □

Again observe that the results here all use strict equality = as the behavioural equivalence $\simeq$ of the target language in the encodings.

Corollary 6.2. *For every language $\mathcal{L}_{\alpha,P,\gamma,\delta_1}$ there is a composable valid encoding into $\mathcal{L}_{\alpha,M,\gamma,\delta_2}$ where $I \leq \delta_2$ and $\delta_1 \leq \delta_2$.*

Proof. The proof proceeds in the manner as Corollary 5.5. □

7. Communication-Medium in Intensionality

This section proves that intensionality is sufficient to encode channel-based communication. That is, that any language $\mathcal{L}_{\alpha,\beta,C,\delta_1}$ can be encoded into $\mathcal{L}_{\alpha,\beta,D,\delta_2}$ where $I \leq \delta_2$ and $\delta_1 \leq \delta_2$.

The approach here is conceptually similar to the standard way to translate channel-based communication into dataspace-based communication when name-matching is available [23, 15, 17]: prepend the sequence of polyadic inputs with a name-match on the channel name, and prepend the list of polyadic outputs with the channel name.

To illustrate, consider the translation $[\![\cdot]\!]$ from $\mathcal{L}_{A,M,C,I}$ into $\mathcal{L}_{A,M,D,I}$:

$$\begin{array}{rcl} [\![s(p)]\!].P & \stackrel{\text{def}}{=} & (\ulcorner s \urcorner \bullet p).[\![P]\!] \\ [\![\overline{s}\langle t\rangle]\!] & \stackrel{\text{def}}{=} & \langle s \bullet t\rangle\ . \end{array}$$

For the input the channel-name s is turned into a name-match and compounded with the input pattern p to yield a new input pattern $\ulcorner s \urcorner \bullet p$. For the output the channel-name s is compounded with the output term t to yield a new output term $s \bullet t$.

Observe that the interaction

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

is translated as follows

$$\begin{array}{rcl} [\![\, s(p).P \mid \overline{s}\langle t\rangle \,]\!] & = & (\ulcorner s \urcorner \bullet p).[\![\, P \,]\!] \mid \langle s \bullet t\rangle \qquad\qquad (\text{by } [\![\,\cdot\,]\!]) \\ & \longmapsto & \sigma[\![\, P \,]\!] \\ & & (\sigma = \{s \bullet t /\!\!/ \ulcorner s \urcorner \bullet p\} = \{s /\!\!/ \ulcorner s \urcorner\} \uplus \{t /\!\!/ p\} = \{\} \uplus \{t /\!\!/ p\} = \{t /\!\!/ p\} \ .) \end{array}$$

Thus the channel name is matched as part of the match rule, and the substitution from the original match of the output and input is applied to the translation as before.

In general , the translation $[\![\,\cdot\,]\!]$ from $\mathcal{L}_{\alpha,\beta,C,\delta_1}$ into $\mathcal{L}_{\alpha,\beta,D,\delta_2}$ (where $I \leq \delta_2$ and $\delta_1 \leq \delta_2$) is the identity on all constructors except for the input and output which are translated as follows:

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

where $\widetilde{p}$ and $\widetilde{t}$ are omitted in the monadic case and the Qs are omitted in the asycnhronous 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.

Theorem 7.1. *For every language $\mathcal{L}_{\alpha,\beta,C,\delta_1}$ there is a valid encoding into $\mathcal{L}_{\alpha,\beta,\delta_2}$ where $I \leq \delta_2$ and $\delta_1 \leq \delta_2$.*

Proof. The proof is a straightforward adaptation of Theorem 5.4, although Lemma 5.2 requires considering the channel name s in the match rule and Lemma 5.3 has only a single reduction after translation. □

Again observe that the results here all use equality = as the behavioural equivalence $\simeq$ of the target language in the encodings.

Corollary 7.2. *For every language $\mathcal{L}_{\alpha,\beta,C,\delta_1}$ there is a composable valid encoding into $\mathcal{L}_{\alpha,\beta,D,\delta_2}$ where $I \leq \delta_2$ and $\delta_1 \leq \delta_2$.*

Proof. The proof proceeds in the manner as Corollary 5.5. □

8. Identity encodings

This sections details the intuition behind Remark 2.1 that the languages can be ordered thanks to the fact that there are composable valid encodings.

First we define an *identity* encoding, which is an encoding that is the identity on all process constructors. Examples include the encoding from a monadic language to its polyadic counterpart; the encoding from an asynchronous language to its synchronous counterpart; and the encoding from a linear language to its nonlinear counterpart. The encoding from dataspace-based communication to channel-based communication is

also considered an identity encoding: it translates all outputs of the form $\langle s \rangle.P$ to $\overline{\mathsf{rn}}\langle s \rangle.P$ and inputs $(p).Q$ to $\mathsf{rn}(p).Q$ where rn is a reserved name in the encoding, that we also used in Section 6.

Let us show that all identity encodings are also valid.

Theorem 8.1. *Every identity encoding $[\![\cdot]\!]$ is a valid encoding.*

Proof. The proof is a straightforward adaptation of Theorem 5.4, again Lemma 5.3 has only a single reduction after translation. □

Next is to prove that any valid encoding can be composed with an identity encoding and the resulting encoding is still valid.

Lemma 8.2. *Let $[\![\cdot]\!]_1 : \mathcal{L}_1 \to \mathcal{L}_2$ be an identity encoding and let $[\![\cdot]\!]_2 : \mathcal{L}_2 \to \mathcal{L}_3$ be any valid encoding, then $[\![[\![\cdot]\!]_1]\!]_2 : \mathcal{L}_1 \to \mathcal{L}_3$ is a valid encoding. Similarly for $[\![\cdot]\!]_2 : \mathcal{L}_3 \to \mathcal{L}_1$ any valid encoding, then $[\![[\![\cdot]\!]_2]\!]_1 : \mathcal{L}_3 \to \mathcal{L}_2$ is also valid.*

Proof. The properties of a valid encoding of Definition 3.1 trivially hold. □

Note also that composing identity encodings results in an identity encoding.

Theorem 8.3. *There exists a valid encoding from $\mathcal{L}_{\alpha_1,\beta_1,\gamma_1,\delta_1}$ to $\mathcal{L}_{\alpha_2,\beta_2,\gamma_2,\delta_2}$ 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:*

$$A \leq S \qquad M \leq P \qquad D \leq C \qquad NO \leq NM \leq I \qquad \delta \leq \delta^+ \ .$$

Proof. The result is a consequence of Lemma 8.2. The following encodings are the identity encodings: $\mathcal{L}_{A,\beta,\gamma,\delta}$ to $\mathcal{L}_{S,\beta,\gamma,\delta}$, and $\mathcal{L}_{\alpha,M,\gamma,\delta}$ to $\mathcal{L}_{\alpha,P,\gamma,\delta}$, and $\mathcal{L}_{\alpha,\beta,D,\delta}$ to $\mathcal{L}_{\alpha,\beta,C,\delta}$, and $\mathcal{L}_{\alpha,\beta,\gamma,NO}$ to $\mathcal{L}_{\alpha,\beta,\gamma,NM}$, and $\mathcal{L}_{\alpha,\beta,\gamma,NM}$ to $\mathcal{L}_{\alpha,\beta,\gamma,I}$, and $\mathcal{L}_{\alpha,\beta,\gamma,\delta}$ to $\mathcal{L}_{\alpha,\beta,\gamma,\delta^+}$. The general result follows by composing the identity encodings. □

In Figure 5 the identity encodings between languages are shown. This concludes proving that all encodings considered in the previous sections are not just valid but also composable.

9. Intensionality Encodes Other Communication Primitives

Using the results in the sections above we conclude that intensionality can encode each of: synchrony, polyadicity, channel-based communication, and name-matching. Further, the valid encodings into intensionality are all composable. Thus any language $\mathcal{L}_{\alpha,\beta,\gamma,\delta_1}$ can be encoded into $\mathcal{L}_{A,M,D,\delta_2}$ where $I \leq \delta_2$ and $\delta_1 \leq \delta_2$.

Theorem 9.1. *Any language $\mathcal{L}_{\alpha 1,\beta 1,\gamma 1,\delta 1}$ can be encoded into any language $\mathcal{L}_{\alpha 2,\beta 2,\gamma 2,\delta 2}$ where $I \leq \delta_2$ and $\delta_1 \leq \delta_2$. That is, intensionality alone is sufficient to encode: synchrony, polyadicity, channel-based communication, and pattern-matching.*

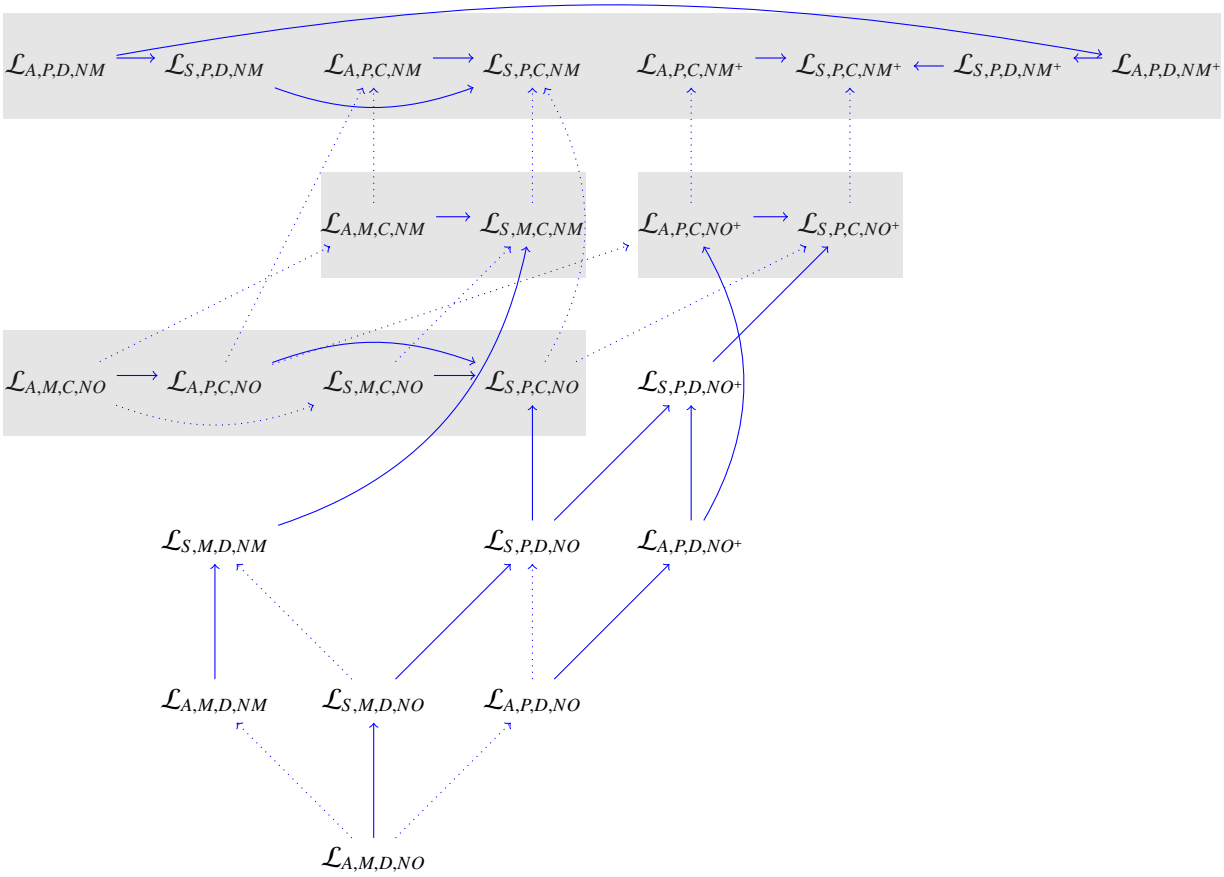

Figure 5: The identity encodings in non-linear non-intensional languages that are used for the impossibility results. Composing identity encodings result in valid encodings, whereas composing any non-identity encoding (not shown here) result in composable encodings. For the sake of clarity we do not show all identity encodings, but only the necessary ones for our proofs. The dotted arrows are used for clarifying the picture but are not necessary in the proofs. Gray rectangles separate classes of equivalent languages. i.e. for which there exists a valid encoding between them.

Proof. When $\alpha 1 \leq \alpha 2$ then trivial by Theorem 8.3, otherwise use Corollary 5.5. When $\beta 1 \leq \beta 2$ then trivial by Theorem 8.3, otherwise use Corollary 6.2. When $\gamma 1 \leq \gamma 2$ then trivial by Theorem 8.3, otherwise use Corollary 7.2. Observe that $\delta 1 \leq \delta 2$ and $I \leq \delta 2$ always hold. Finally, due to all the results being for composable valid encodings, the encodings can be composed as required. □

Observe that by exploiting an identity encoding the following result is easily obtained.

Theorem 9.2. *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, non-linear intensionality alone is sufficient to encode all other communication primitive forms.*

Proof. By the same technique as Theorem 9.1. □

Thus the non-linear intensional languages are clearly the most expressive languages, and further can all encode each other.

The next section considers further encodings between non-linear languages that have not been considered before. After that, the rest of the paper explores negative results showing where valid encodings cannot exist.

10. Non-Linear Non-Intensional Encodings

Non-linearity is not strictly a property of intensional languages since any polyadic language can also be non-linear. This section considers the encodings that exist between non-linear languages that are not intensional. The separations results for non-linear non-intensional languages are shown in the next section. Observe that monadic languages are omitted from the discussion here since, with the exception of intensional languages, the linear and non-linear instances coincide. Hence this section only considers polyadic non-intensional languages.

It is possible to encode non-linearity (non-intensional) languages into linear languages that are at least polyadic and name-matching. To aid in the encoding define the *binding equality* relation $=_b$ that is reflexive, transitive, and satisfies the following rules:

$$x =_b y \stackrel{\text{def}}{=} x = y \qquad x =_b \ulcorner a \urcorner \stackrel{\text{def}}{=} \text{false} \qquad \ulcorner a \urcorner =_b \ulcorner b \urcorner \stackrel{\text{def}}{=} \text{false}\ .$$

That is two binding names x and y are binding equal if $x = y$. A binding name is never binding equal to a name-match. Name-matches are never binding equal to each other.

The following encodings in this section exploit the binding equality relation and a reserved name eq. The key idea is to compare each possible pair of names in the output or input and create an appropriate encoding. Since this rapidly becomes extremely large (encoded arity is the ith triangular number for encoding arity i), the following two examples demonstrate for arity 2 and 3. Consider the encoding $[\![\cdot]\!]$ from $\mathcal{L}_{A,P,D,NO^+}$ of arity 2 into $\mathcal{L}_{A,P,D,NM}$, and with the output and input processes defined as follows:

$$[\![\langle a, b \rangle]\!] \stackrel{\text{def}}{=} (\nu w)\mathbf{if}\ a = b\ \mathbf{then}\ \langle \mathsf{eq}, a, b \rangle\ \mathbf{else}\ \langle w, a, b \rangle \qquad \text{for } w \text{ fresh}$$

$$[\![(x, y).P]\!] \stackrel{\text{def}}{=} \begin{cases} (\ulcorner \mathsf{eq} \urcorner, x, w).[\![P]\!] & x =_b y \wedge w \text{ fresh} \\ (w, x, y).[\![P]\!] & x \neq_b y \wedge w \text{ fresh} \end{cases}$$

The output is encoded into a conditional that compares the pair of output names a and b. If they are the same then the encoding is an output with an additional name indicating equality eq of a and b at the front of the sequence of output names. Otherwise if $a \neq b$ then the first name output is some fresh name w. The encoding of the input depends on the relation of the two binding names. If they are binding equal $x =_b y$ then the encoded input uses a name-match to ensure the output is outputting two equal names, and then bind one to x and bind the other to a fresh name w. Otherwise if they are not binding equal then the first position (indicating equality) is ignored by being bound to some fresh name w.

The same approach can be extended to any length or arity where the first j output names or input patterns are determined by all the possible pairwise comparisons of the arity of the output or input. For example, the following shows the encodings for arity 3 outputs:

$$
\begin{array}{l}
[\![\, \langle a, b, c\rangle \,]\!] \stackrel{\text{def}}{=} \\
\left\{\begin{array}{l}
(\nu w_1)(\nu w_2)(\nu w_3) \\
\textbf{if } a = b \textbf{ then } (\textbf{if } a = c \textbf{ then } (\textbf{if } b = c \textbf{ then } \langle \mathsf{eq}, \mathsf{eq}, \mathsf{eq}, a, b, c\rangle \\
\qquad\qquad\qquad\qquad\qquad\qquad \textbf{else } \langle \mathsf{eq}, \mathsf{eq}, w_1, a, b, c\rangle) \\
\qquad\qquad\qquad\qquad \textbf{else } \ (\textbf{if } b = c \textbf{ then } \langle \mathsf{eq}, w_1, \mathsf{eq}, a, b, c\rangle \\
\qquad\qquad\qquad\qquad\qquad\qquad \textbf{else } \langle \mathsf{eq}, w_1, w_2, a, b, c\rangle)) \\
\qquad\qquad \textbf{else } (\textbf{if } a = c \textbf{ then } (\textbf{if } b = c \textbf{ then } \langle w_1, \mathsf{eq}, \mathsf{eq}, a, b, c\rangle \\
\qquad\qquad\qquad\qquad\qquad\qquad \textbf{else } \langle w_1, \mathsf{eq}, w_2, a, b, c\rangle) \\
\qquad\qquad\qquad\qquad \textbf{else } \ (\textbf{if } b = c \textbf{ then } \langle w_1, w_2, \mathsf{eq}, a, b, c\rangle \\
\qquad\qquad\qquad\qquad\qquad\qquad \textbf{else } \langle w_1, w_2, w_3, a, b, c\rangle))
\end{array}\right\} \ w_1, w_2, w_3 \text{ fresh}
\end{array}
$$

and inputs:

$$
\begin{array}{l}
[\![\, (x, y, z).P \,]\!] \stackrel{\text{def}}{=} \\
\left\{\begin{array}{ll}
(\ulcorner\mathsf{eq}\urcorner, \ulcorner\mathsf{eq}\urcorner, \ulcorner\mathsf{eq}\urcorner, x, w_1, w_2).[\![\, P \,]\!] & x =_b y =_b z \wedge w_1, w_2 \text{ fresh} \\
(\ulcorner\mathsf{eq}\urcorner, w_1, w_2, x, w_3, z).[\![\, P \,]\!] & x =_b y \wedge y \neq_b z \wedge w_1, w_2, w_3 \text{ fresh} \\
(w_1, \ulcorner\mathsf{eq}\urcorner, w_2, x, y, w_3).[\![\, P \,]\!] & x \neq_b y \wedge x =_b z \wedge w_1, w_2, w_3 \text{ fresh} \\
(w_1, w_2, \ulcorner\mathsf{eq}\urcorner, x, y, w_3).[\![\, P \,]\!] & x \neq_b y \wedge y =_b z \wedge w_1, w_2, w_3 \text{ fresh} \\
(w_1, w_2, w_3, x, y, z).[\![\, P \,]\!] & \left\{\begin{array}{l} x \neq_b y \wedge x \neq_b z \wedge y \neq_b z \\ \wedge w_1, w_2, w_3 \text{ fresh} \,. \end{array}\right.
\end{array}\right.
\end{array}
$$

Here the first three names in the output correspond to reporting equality of each possible pair of names of the output: $a = b$ then $a = c$ then $b = c$. Conditionals are used to instantiate them, with eq used when the equality is true, and an otherwise fresh name w_i when they are not equal. The inputs are encoded according to the relations between their binding names. When all names are binding equal then use name-matching to only interact when all output names are equal, i.e. an output of the form $\langle \mathsf{eq}, \mathsf{eq}, \mathsf{eq}, \ldots\rangle$. When only one pair of names are binding equal then name-match for that pair only. In all cases, when other binding names are required (when replacing equal binding names, or binding in the place of a possible eq) then fresh names w_i are used.

Observe that the encoding above for the outputs is naive by considering all possible combinations of equality, even those that may be redundant or impossible. (For example, $a = b$ and $a = c$ ensures $b = c$.) A more simplified encoding could use the same

approach as that used for encoding the inputs, but the more verbose is chosen here for clarity.

The following proves that the above encoding is valid for $\mathcal{L}_{A,P,D,NO^+}$ into $\mathcal{L}_{A,P,D,NM}$.

Lemma 10.1. *Let P and Q be two $\mathcal{L}_{A,P,D,NO^+}$ processes such that $P \mapsto\!\!\!\!\!\!/\;$ and $Q \mapsto\!\!\!\!\!\!/\;$. Then $[\![P]\!] \mid [\![Q]\!] \longmapsto$ if and only if $P \mid Q \longmapsto$.*

Proof. The proof is by observing that an interaction must be between an input of the form $(\widetilde{m}).T_1$ and an output of the form $\langle\widetilde{a}\rangle.T_2$ where $|\widetilde{m}| = i = |\widetilde{a}|$ and i is the jth triangular number for some j. The proof then proceeds by observing that the source language must have processes $(\widetilde{x})S_1$ and $\langle\widetilde{b}\rangle S_2$ where $|\widetilde{x}| = j = |\widetilde{b}|$. Conclude since $\{\widetilde{a}/\!\!/\widetilde{m}\}$ if and only if $\{\widetilde{b}/\!\!/\widetilde{x}\}$. □

Theorem 10.2. *The encoding from $\mathcal{L}_{A,P,D,NO^+}$ into $\mathcal{L}_{A,P,D,NM}$ is valid.*

Proof. The proof is a straightforward adaptation of Theorem 5.4, with Lemma 10.1 adapting Lemma 5.2 and the adaptation of Lemma 5.3 has only a single reduction after translation. □

Observe that the same approach can be used to encode $\mathcal{L}_{A,P,D,NM^+}$ into $\mathcal{L}_{A,P,D,NM}$. The encoding technique can easily be extended to channel-based communication by simply maintaining the channel names from source to target language. Further, since the target language has name-matching, channel-based communication can be easily encoded into dataspace-based communication by prefixing the inputs and outputs with the channel names in the usual manner.

Theorem 10.3. *For every language $\mathcal{L}_{\alpha,P,\gamma,\delta}$ where $\delta \leq NM^+$ there is a valid encoding into $\mathcal{L}_{\alpha,P,\gamma,NM}$.*

Proof. The same encoding and proof technique as Theorem 10.2 applies for all instantiations of α and γ and δ. □

Corollary 10.4. *For every language $\mathcal{L}_{\alpha,P,\gamma,\delta}$ where $\delta \leq NM^+$ there is a composable valid encoding into $\mathcal{L}_{\alpha,P,\gamma,NM}$.*

Proof. The proof proceeds in the manner as Corollary 5.5. □

Corollary 10.5. *There exist valid encodings from $\mathcal{L}_{-,P,-,\delta}$ into $\mathcal{L}_{A,P,D,NM}$ where $\delta = NO^+$ or $\delta = NM^+$.*

Proof. Corollary 10.4 is used to encode $\mathcal{L}_{-,P,-,\delta}$, with $\delta = NO^+$ or $\delta = NM^+$ into $\mathcal{L}_{-,P,-,NM}$. Then the encoding of synchronism or channel-names in $\mathcal{L}_{A,P,D,NM}$ follows from Figure 4 and composability due to Corollary 10.4 and Theorem 8.3. As all are composable encodings, we obtain a composable, and therefore valid, encoding from $\mathcal{L}_{-,P,-,\delta}$ into $\mathcal{L}_{A,P,D,NM}$, with $\delta = NO^+$ or $\delta = NM^+$. □

The above show that non-linear non-intensional languages can be encoded into linear languages as long as they have unbounded matching degree. Indeed, unbounded matching degree is needing to test the potential equality of each pair of names, since arity is unbounded. Although it may appear possible to represent all this information in

a single name, encoded inputs may then require multiple encoded inputs. For example, suppose that the encoding of an output uses a single name instead of a sequence such as

$$\begin{array}{rcl} [\![\langle a, a, a \rangle]\!] & \stackrel{\text{def}}{=} & \langle \mathsf{eqeqeq}, a, a, a \rangle \\ [\![\langle a, a, b \rangle]\!] & \stackrel{\text{def}}{=} & \langle \mathsf{eq}w_1w_2, a, a, b \rangle \end{array}$$

where eqeqeq and $\mathsf{eq}w_1w_2$ are single names determined by the encoding. Now consider the encoding:

$$[\![(x, x, y).P]\!] \stackrel{\text{def}}{=} (m, x, w, y).[\![P]\!] \qquad w \notin \mathsf{fn}(P) \cup \mathsf{fn}(m) \cup \{x, y\} \ .$$

Clearly m should match with eqeqeq and also $\mathsf{eq}w_1w_2$, which can only occur if either: m is a name-match of the form $\ulcorner n \urcorner$ and $\mathsf{eqeqeq} = n = \mathsf{eq}w_1w_2$; or m is a binding name. It is then straightforward in both cases to show that this does not yield a valid encoding. Note however that this approach can be used in languages with a *synchronisation algebra* (where a synchronisation is decided by a function on names that is not necessarily the equality on names) [45], [2]. However such languages are outside the scope of this paper.

Now consider the following two non-identity encodings that exists between non-linear non-intensional languages.

The first is from $\mathcal{L}_{S,P,D,NO^+}$ into $\mathcal{L}_{A,P,C,NO^+}$ by translating all process constructors as the identity translation except for the input and output which are translated as follows:

$$\begin{array}{rcl} [\![(\widetilde{x}).P]\!] & \stackrel{\text{def}}{=} & \mathsf{rn}(y, \widetilde{x}).(\overline{y}\langle\rangle \mid [\![P]\!]), \text{ for } y \text{ fresh} \\ [\![\langle\widetilde{b}\rangle.P]\!] & \stackrel{\text{def}}{=} & (\nu c)\overline{\mathsf{rn}}\langle c, \widetilde{b}\rangle \mid c().[\![P]\!], \text{ for } c \text{ fresh} \end{array}$$

and where rn is a reserved name introduced in the translation.

The second is from $\mathcal{L}_{S,P,C,NO^+}$ into $\mathcal{L}_{A,P,C,NO^+}$ which consists of translating all process constructors as the identity except for the input and the output which are defined as follows:

$$\begin{array}{rcl} [\![a(\widetilde{x}).P]\!] & \stackrel{\text{def}}{=} & a(y, \widetilde{x}).(\overline{y}\langle\rangle \mid [\![P]\!]), \text{ for } y \text{ fresh} \\ [\![\overline{a}\langle\widetilde{b}\rangle.P]\!] & \stackrel{\text{def}}{=} & (\nu c)\overline{a}\langle c, \widetilde{b}\rangle \mid c().[\![P]\!], \text{ for } c \text{ fresh} \ . \end{array}$$

The following results show that both of these encodings from $\mathcal{L}_{S,P,\gamma,NO^+}$ into $\mathcal{L}_{A,P,C,NO^+}$ are valid.

Lemma 10.6. *Given two processes P and Q such that $P \not\longmapsto$ and $Q \not\longmapsto$, then $[\![P \mid Q]\!] \longmapsto$ if and only if $P \mid Q \longmapsto$.*

Proof. An interaction in the source language must be between an input $(\widetilde{x})$ (or $a(\widetilde{x})$) of P and an output $\langle\widetilde{b}\rangle$ (or $\overline{a}\langle\widetilde{b}\rangle$) of Q. From the translations above we have that $[\![P]\!]$ does an input on $\mathsf{rn}(y, \widetilde{x})$ (or $a(y, \widetilde{x})$) and $[\![Q]\!]$ does an output on $\overline{\mathsf{rn}}\langle b, \widetilde{a}\rangle$ (or $\overline{a}\langle c, \widetilde{b}\rangle$) and thus the two interact. For the reverse direction, also note that an input of $[\![P]\!]$ and an output of $[\![Q]\!]$ are needed which are necessarily the result of the translations above. □

Theorem 10.7. *The encodings from $\mathcal{L}_{S,P,\gamma,NO^+}$ into $\mathcal{L}_{A,P,C,NO^+}$ is valid.*

Proof. The proof is a straightforward adaptation of Theorem 5.4, with Lemma 10.6 adapting Lemma 5.2 and the adaptation of Lemma 5.3 has only a single reduction after translation. □

Corollary 10.8. *For every language $\mathcal{L}_{S,P,\gamma,NO^+}$ there is a compositional valid encoding into $\mathcal{L}_{A,P,C,NO^+}$.*

Proof. The proof proceeds in the manner as Corollary 5.5. □

The following sections explore separation results where encodings are not possible.

11. Non-Linearity Without Bounded Matching

From Sections 8–10 the encodings into linear and non-linear languages that have unbounded name-matching capability have been shown. This section begins the various impossibility results by considering target languages with bounded name-matching and where either the source or target language are non-linear. Note that impossibility results related specifically to intensionality are handled in Section 12 and non-linear intensionality in Section 13.

The first impossible encoding is from $\mathcal{L}_{-,P,-,NO^+}$ into any language $\mathcal{L}_{-,-,-,NO}$.

Theorem 11.1. *There exists no valid encoding from $\mathcal{L}_{A,P,D,NO^+}$ into $\mathcal{L}_{\alpha,\beta,\gamma,NO}$.*

Proof. The proof is by contradiction, assume there exists a valid encoding $[\![\cdot]\!]$ from $\mathcal{L}_{A,P,D,NO^+}$ into $\mathcal{L}_{S,P,C,NO}$ (the least expressive $\mathcal{L}_{-,P,-,NO^+}$ language into the most expressive of the $\mathcal{L}_{-,-,-,NO}$ languages w.r.t. the identity encodings of Remark 2.1). Consider the $\mathcal{L}_{A,P,D,NO^+}$ processes $S_1 = \langle a, b\rangle$ and $S_2 = (x, y).\surd$ and $S_3 = (x, x).\surd$ and the substitution $\sigma = \{a/b\}$. As $S_1 \mid S_2 \Downarrow_1$ and $(\sigma S_1) \mid S_2 \Downarrow_1$ and $(\sigma S_1) \mid S_3 \Downarrow_1$ by Proposition 3.5, it follows that $[\![S_1 \mid S_2]\!] \longmapsto$ and $[\![(\sigma S_1) \mid S_2]\!] \longmapsto$ and $[\![(\sigma S_1) \mid S_3]\!] \longmapsto$, respectively. From Proposition 3.3 and from $S_1 \mid S_3 \not\longmapsto$, we have that $[\![S_1 \mid S_3]\!] \not\longmapsto$.

Consider the reductions $[\![(\sigma S_1) \mid S_2]\!] \longmapsto$ and $[\![(\sigma S_1) \mid S_3]\!] \longmapsto$. In the process $[\![(\sigma S_1) \mid S_2 \mid S_3]\!] \longmapsto$ we have that $[\![(\sigma S_1)]\!]$ can synchronize with both $[\![S_2]\!]$ and $[\![S_3]\!]$. From Lemma 3.6 $[\![(\sigma S_1)]\!]$ must interact with $[\![S_2]\!]$ and $[\![S_3]\!]$ with the same action.

Suppose w.l.o.g. that $[\![S_1]\!]$ interacts with an output $\overline{c}\langle\widetilde{e}\rangle$ with $\sigma'(c) = d$, for some name d (where σ' is defined via validity of the encoding for σ). Then $[\![S_3]\!]$ and $[\![S_2]\!]$ interact with inputs $d(\widetilde{u})$ and $d(\widetilde{v})$ respectively, where both inputs have the same arity as the output $\overline{c}\langle\widetilde{e}\rangle$.

Now by applying Lemma 3.6 we obtain that $[\![S_2]\!]$ interacts with the input $d(\widetilde{u})$ with both $[\![(\sigma S_1)]\!]$ and $[\![S_1]\!]$, and again with the same arity. It follows that $[\![S_1]\!]$ can do an ouput on channel d. Since $[\![S_3]\!]$ has an input on channel d (from $[\![(\sigma S_1) \mid S_3]\!] \longmapsto$) it follows that $[\![S_1]\!]$ and $[\![S_3]\!]$ can synchronise on channel d. The only possibility for $[\![S_1 \mid S_3]\!] \mapsto\!\!\!\!\!\!\not\longrightarrow$ is if the context for parallel composition blocks the synchronisation. However $[\![\sigma S_1 \mid S_3]\!] \longmapsto$ and $[\![\sigma S_1]\!] = \sigma'[\![S_1]\!]$, which means that the context blocks the reduction and by applying the substitution σ' on the context, the reduction becomes possible. There are now two cases to consider:

- The context for encoding S_1 blocks the synchronisation. By reasoning on the reduction $[\![S_1 \mid S_2]\!] \longmapsto$ this case is not possible.

- The context for parallel composition blocks the synchronisation using an **if** $c_1 = c_2$ **then** $\cdot$ **else** $\cdot$ construct. As before, as the same input is used in the reduction $[\![S_1 \mid S_2]\!] \longmapsto$ this case is not possible.

□

Theorem 11.2. *There exists no valid encoding from $\mathcal{L}_{A,P,D,NO^+}$ into $\mathcal{L}_{\alpha,M,\gamma,NM}$.*

Proof. Let us suppose, by contradiction, that there exists a valid encoding $[\![\cdot]\!]$ from $\mathcal{L}_{A,P,D,NO^+}$ into $\mathcal{L}_{S,M,C,NM}$ (the least expressive $\mathcal{L}_{-,P,-,NO^+}$ language into the most expressive of the $\mathcal{L}_{-,M,-,NM}$ languages w.r.t. the identity encodings of Figure 5). Consider the $\mathcal{L}_{A,P,D,NO^+}$ processes $S_1 = \langle a, a, b\rangle$ and $S_2 = (x, x, y).\surd$ and $S_3 = (x, x, x).\surd$ and a substitution $\sigma = \{a/b\}$ with $S_4 = \sigma S_1 = \langle a, a, a\rangle$.

Using Lemma 3.6 we conclude that in the process $[\![S_4 \mid S_1 \mid S_2]\!]$, $[\![S_2]\!]$ synchronizes on the same input $c(m)$ with both $[\![S_4]\!]$ and $[\![S_1]\!]$. Similarly for $[\![S_4 \mid S_2 \mid S_3]\!]$, where the channel c is used for both synchronisations.

As $S_1 \mid S_3 \mapsto\!\!\!\!\!\!\rightarrow$ it follows from Proposition 3.3 that $[\![S_1 \mid S_3]\!] \mapsto\!\!\!\!\!\!\rightarrow$. Then $[\![S_1]\!]$ and $[\![S_3]\!]$ can synchronise on channel c. The only possibility for $[\![S_1 \mid S_3]\!] \mapsto\!\!\!\!\!\!\rightarrow$ is if the context for parallel composition blocks the synchronisation. However $[\![\sigma S_1 \mid S_3]\!] \longmapsto$ and $[\![\sigma S_1]\!] = \sigma'[\![S_1]\!]$, which means that the context blocks the reduction and by applying the substitution σ' on the context, the reduction becomes possible. As in Theorem 11.1 we have that the translation context for parallel composition cannot block the synchronisation. Therefore this case is possible only if the input on channel c in $[\![S_3]\!]$ uses a name-match, that we write as $c(\ulcorner n \urcorner)$. It follows that $[\![S_4]\!]$ outputs $\overline{c}\langle n\rangle$ and $[\![S_1]\!]$ outputs $\overline{c}\langle m\rangle$ for $n \neq m$.

We apply now the same reasoning on the processes $S_1 = \langle a, a, b\rangle$ and $S_2 = (x, x, y).\surd$ and $S_5 = (x, y, z).\surd$ and $S_6 = \langle a, b, c\rangle$. First observe that all synchronising pairs of processes still use the same channel c in the encoding. This follows from Lemma 3.6. As $S_6 \mid S_2 \mapsto\!\!\!\!\!\!\rightarrow$ we have two cases.

- The context of the parallel composition blocks the synchronisation. We treat this case as above.

- Alternatively, the process $[\![S_2]\!]$ uses a name-match to prevent the synchronisation on channel c with $[\![S_6]\!]$. We write the input of $[\![S_2]\!]$ as $c(\ulcorner p \urcorner)$, for some name p such that $p = m$, as it synchornizes with the output of $[\![S_1]\!]$. We reach a contradiction, as $[\![S_2]\!]$ also synchronizes with the output of $[\![S_4]\!]$ and therefore $p = n = m$.

□

Note that Theorem 11.1 does not follow from Theorem 11.2 as there is no identity encoding from $\mathcal{L}_{S,P,C,NO}$ to any language $\mathcal{L}_{\_,M,\_,NM}$.

These results show that to encode non-linearity requires unbounded name-matching in the target language. Merely having one channel-name or a monadic name-match is insufficient to encode non-linearity.

Let us now show the imposibility results on non-linear languages that complete Figure 4. The following theorem shows that there is no valid encoding from $\mathcal{L}_{S,\beta,D,\delta}$ into $\mathcal{L}_{A,P,D,NO^+}$, for $\delta \leq NO^+$. The proof sketch below illustrates the key points needed to adapt the proof to exploit the result in [23], full details are omitted here due to the original using LTS rather than reduction semantics.

Theorem 11.3. *There exists no valid encoding from $\mathcal{L}_{S,M,D,NO}$ into $\mathcal{L}_{A,P,D,NO^+}$.*

Proof Sketch. The proof is by contradiction, assume there exists a valid encoding $[\![\cdot]\!]$ from $\mathcal{L}_{S,M,D,NO}$ into $\mathcal{L}_{A,P,D,NO^+}$. (Observe that we choose the least expressive $\mathcal{L}_{S,\beta,D,\delta}$ language for $\delta \leq NO^+$ as the source language to illustrate the proof with minimal complexity due to other language features.)

We follow the same proof structure as in Theorem 5.6. from [23] which shows that $\mathcal{L}_{S,M,D,NO}$ cannot be encoded into $\mathcal{L}_{A,P,D,NO}$; we extend the proof to cover non-linearity as well. The proof of Theorem 5.6. from [23] uses Proposition 2.1(3) from [23][2] which does not hold in general for non-linearity, but holds in the context of this proof. We revisit therefore the begining of the proof of Theorem 5.6. (which uses Proposition 2.1(3)).

We have that for any valid encoding into $\mathcal{L}_{A,P,D,NO^+}$ and for $\simeq$ barbed congruence[3], $[\![P \mid Q]\!] \simeq (\nu\widetilde{n})([\![P]\!] \mid [\![Q]\!] \mid R)$ for some $(\nu\widetilde{n})$ and R introduced only by the encoding. This is straightforward from Lemma 5.4. and Proposition 3.5.

Now consider the processes $S_1 = \langle a \rangle.S'_1$ and $S_2 = (x).S'_2$, for which we have that $[\![S_1 \mid S_2]\!] \simeq (\nu\widetilde{n})([\![S_1]\!] \mid [\![S_2]\!] \mid R) \Longmapsto\simeq [\![S'_1 \mid \{a/x\}S'_2]\!] \simeq (\nu\widetilde{n})([\![S'_1]\!] \mid [\![\{a/x\}S'_2]\!] \mid R')$, by the argument above and by operational correspondence. Then necessarily $[\![S_1]\!] \overset{\phi_1 \cdots \phi_n}{\Longmapsto} [\![S'_1]\!] \mid T_1$ and $[\![S_2]\!] \overset{\overline{\phi_1} \cdots \overline{\phi_n}}{\Longmapsto} [\![S'_2]\!] \mid T_2$, where $T_1 \simeq \mathbf{0} \simeq T_2$. We denoted with ϕ_i an input or an output, and with $\overline{\phi_i}$ its dual, in the LTS of $\mathcal{L}_{A,P,D,NO^+}$. As mentioned in Section 2.2, we are using a reduction semantics, but a corresponding LTS can be found in (or derived from) the literature.

Now let us suppose that in the sequence $[\![S_1]\!] \overset{\phi_1 \cdots \phi_n}{\Longmapsto} [\![S'_1]\!]$ there are only outputs for which we show a contradiction. (This is to satisfy the assumption of the proof of Theorem 5.6 in [23] that $\phi_1 \cdots \phi_n$ must contain an input.) Take $S'_2 = \mathbf{if}\ a = x\ \mathbf{then}\ \omega\ \mathbf{else}\ \mathbf{0}$ and $S'_1 = S_2$ where ω is a divergent process. Then $[\![S_1]\!] \overset{\phi_1 \cdots \phi_n}{\Longmapsto} \simeq [\![S'_1]\!] = [\![S_2]\!] \overset{\overline{\phi_1} \cdots \overline{\phi_n}}{\Longmapsto} \simeq [\![S'_2]\!]$. As $[\![S_1]\!]$ is an asynchronous term, we have that any output in $\phi_1 \cdots \phi_n$ is concurrent with any input in $\overline{\phi_1} \cdots \overline{\phi_n}$. Moreover the two sequences can synchronize and we have that $[\![S_1]\!] \longmapsto^\omega$, whereas $S_1 \not\longmapsto^\omega$, and we reach a contradiction.

The only other technicality is to show that R and R' do not interfere with the behaviour, however it is straightforward (if tedious) to show that these are placeholders

[2][23] uses an LTS, however we can reformulate the proposition as follows: if $P \in \mathcal{L}_{A,\_,\_,\_}$ and P can communicate using an output $\overline{s}\langle \widetilde{t} \rangle$ and then using an action α such that $\widetilde{t} \cap \mathsf{names}(\alpha) \cap \mathsf{bn}(P) = \emptyset$ then P can communicate on both $\overline{s}\langle \widetilde{t} \rangle$ and α in parallel.

[3]A little delicacy needs to be taken with the definition of the behavioural equivalence here and any barbed congruence may not be sufficient. However, we refer to [18] for barbs that can handle the complexities of the languages considered here.

for artifacts of the encoding, and cannot cause problems without yielding the encoding invalid due to introducing divergence.

The rest of the proof is straightforward from Theorem 5.6 from [23]. □

Theorem 11.4. *There exists no valid encoding from $\mathcal{L}_{\alpha,\beta,C,NO}$ into $\mathcal{L}_{S,P,D,NO^+}$.*

Proof. We show that there is no valid encoding from $\mathcal{L}_{A,M,C,NO}$ into $\mathcal{L}_{S,P,D,NO^+}$ from which the more general result follows, see Figure 5. The proof is similar to the one of Theorem 4.6. from [23]. Let $S_1 = \overline{a}\langle b\rangle$, $S_2 = a(x).\surd$ and let $\sigma = \{a/b, b/a\}$ with $S_3 = \sigma S_1 = \overline{b}\langle a\rangle$ be three processes such that, $[\![S_1 \mid S_2]\!] \longmapsto$ and $[\![S_3 \mid S_2]\!] \not\longmapsto$. We have that the substitution prevents $[\![S_3]\!]$ from synchronising. There are only two possible cases:

- If $[\![S_2]\!]$ has a non linear input, for instance (x, x), such that it synchronises with the output of $[\![S_1]\!]$ but not with the output of $[\![S_3]\!]$. Let us write $\langle c_1, c_2\rangle$ for the output of $[\![S_1]\!]$, for which necessarily $c_1 = c_2$. But then the substitution either acts on c_1 and c_2 or on neither of them, hence $\langle c, c\rangle$ for some c is the output of $[\![S_3]\!]$.
- An **if** $c_1 = c_2$ **then** T_1 **else** T_2 blocks the synchronisation in $[\![S_3]\!]$. Using Lemma 3.7 we have that this case is not possible.

The substitution in $[\![S_3]\!]$ therefore cannot prevent a transition in $\mathcal{L}_{S,P,D,NO^+}$ and therefore $[\![S_1 \mid S_3]\!] \longmapsto$, contradicting Proposition 3.3. □

Theorem 11.5. *There exists no valid encoding from $\mathcal{L}_{\alpha_1,P,C,NO^+}$ into $\mathcal{L}_{\alpha_2,P,D,NO^+}$.*

Proof. This is a corollary of Theorem 11.4. Suppose that there exists a valid encoding from $\mathcal{L}_{A,P,C,NO^+}$ into $\mathcal{L}_{S,P,D,NO^+}$. Then compose it with the identity encoding from $\mathcal{L}_{A,P,C,NO}$ into $\mathcal{L}_{A,P,C,NO^+}$ and obtain a valid encoding from $\mathcal{L}_{A,P,C,NO}$ to $\mathcal{L}_{S,P,D,NO^+}$, which contradicts Theorem 11.4. Similarly for the encodings from $\mathcal{L}_{S,P,C,NO^+}$, or to $\mathcal{L}_{A,P,D,NO^+}$. □

Next, we show that there is no valid encoding from $\mathcal{L}_{\alpha_1,M,C,NM}$ into $\mathcal{L}_{\alpha_2,P,\gamma,NO^+}$.

Theorem 11.6. *There exists no valid encoding from $\mathcal{L}_{A,M,C,NM}$ into $\mathcal{L}_{\alpha,P,\gamma,NO^+}$.*

Proof. Suppose by contradiction that there exists a valid encoding from $\mathcal{L}_{A,M,C,NM}$ into $\mathcal{L}_{S,P,C,NO^+}$ (the least expressive $\mathcal{L}_{-,M,C,NM}$ language into the most expressive of the $\mathcal{L}_{-,P,-,NO^+}$ languages w.r.t. the identity encodings of Figure 5).

We proceed similarly to the proof of Theorem 4.3. from [23]. The only cases that are different here consists of showing that applying a substitution on a non linear input or on the **if** $c_1 = c_2$ **then** T_1 **else** T_2 construct cannot block a synchronisation.

Consider the processes $S_1 = \overline{a}\langle b\rangle$, $S_2 = a(\ulcorner b \urcorner).\surd$ and we can consider w.l.o.g. that $[\![S_2]\!]$ does an input $d(\widetilde{x})$ and $[\![S_1]\!]$ does an output $\overline{d}\langle\widetilde{e}\rangle$. Take c such that $d \notin \varphi_{[\![\,]\!]}(c)$. We have two cases.

- If $d \notin \varphi_{[\![\,]\!]}(b)$ then take the substitution $\sigma = \{c/b, b/c\}$, with $S_3 = \sigma S_1 = \overline{a}\langle c\rangle$. We have that $[\![S_1]\!]$ synchronises with $[\![S_2]\!]$ whereas $[\![S_2]\!]$ does not synchronise with $[\![S_3]\!]$ (i.e. $[\![\sigma S_1]\!]$).

– If $\sigma'(d) \neq d$ (where σ' defined by validity of the encoding such that $[\![\sigma S_1]\!] = \sigma'[\![S_1]\!]$) then $d \notin \varphi_{[\![\,]\!]}(a)$ as a is not modified by σ. Then necessarily d is in the free names of the translation context but then d is also in the context of $[\![\overline{a'}\langle b\rangle]\!]$, for any name $a \neq a'$. It follows that $[\![\overline{a'}\langle b\rangle]\!]$ synchronizes on d with $[\![S_2]\!]$, whereas $\overline{a'}\langle b\rangle \mid S_2 \not\mapsto$ yielding contradiction.

– If the input in $[\![S_2]\!]$ is non linear, then it is of the form $d(x_1, x_2, \ldots, x_n)$ where some $x_i = x_j$ for $i \neq j$ and $i, j \in \{1, \ldots, n\}$, then $[\![S_2]\!]$ interacts with the output $\overline{d}\langle c_1, c_2, \ldots, c_n\rangle$ where $c_i = c_j$. A substitution cannot block the synchronisation for such an output.

– An **if** $c_1 = c_2$ **then** T_1 **else** T_2 blocks the synchronisation in $[\![S_3]\!]$. From Lemma 3.7 this case is not possible.

- If $d \in \varphi_{[\![\,]\!]}(b)$ then by the definition of a valid renaming policy $d \notin \varphi_{[\![\,]\!]}(a)$. Take then the substitution $\sigma_1 = \{c/a, a/c\}$, with $S_4 = \sigma_1 S_1 = \overline{c}\langle b\rangle$. The proof then follows as above.

□

Theorem 11.7. *There exists no valid encoding from $\mathcal{L}_{\alpha,P,\gamma,NM}$ into $\mathcal{L}_{\alpha,P,\gamma,NO^+}$.*

Proof. Suppose by contradiction that there exists a valid encoding from $\mathcal{L}_{A,P,D,NM}$ into $\mathcal{L}_{S,P,C,NO^+}$ (the least expressive $\mathcal{L}_{-,P,-,NM}$ language into the most expressive of the $\mathcal{L}_{-,P,-,NO^+}$ languages w.r.t. the identity encodings of Figure 5).

It suffices to consider the $\mathcal{L}_{A,P,D,NM}$ processes $S_1 = \langle a, b\rangle$ and $S_2 = (\ulcorner a \urcorner, \ulcorner b \urcorner).\surd$. Observe that these follow the same structure as in Theorem 11.6 by requiring matching on the names a and b to synchronise. The rest of the proof can be derived in the same manner as Theorem 11.6. □

Theorem 11.8. *There exists no valid encoding from $\mathcal{L}_{\alpha_1,M,D,NM}$ into $\mathcal{L}_{\alpha_2,P,D,NO^+}$.*

Proof. Suppose by contradiction that there exists a valid encoding from $\mathcal{L}_{A,M,D,NM}$ into $\mathcal{L}_{S,P,D,NO^+}$. The rest follows by the composition of the identity encodings of Figure 5.

Consider the processes $S_1 = \langle a\rangle$, $S_2 = (\ulcorner a \urcorner).\surd$ and the substitution $\sigma = \{b/a, a/b\}$, with $S_3 = \sigma S_1 = \langle b\rangle$. We have that $[\![S_1]\!]$ synchronises with $[\![S_2]\!]$ whereas the substitution blocks the synchronisation in $[\![S_3]\!]$. As in the proof of Theorem 11.6, process constructors (in particular, non linear inputs) cannot block the synchronisation with $[\![S_3]\!]$, and thus we reach a contradiction as $[\![S_3]\!]$ and $[\![S_1]\!]$ can reduce. □

12. Intensionality Cannot be Encoded

This section considers the impossibility of encoding intensionality with any combination of other properties. That is, that any language $\mathcal{L}_{\alpha 1,\beta 1,\gamma 1,\delta 1}$ where $I \leq \delta 1$ cannot be encoded into $\mathcal{L}_{\alpha 2,\beta 2,\gamma 2,\delta 2}$ where $\delta 2 \leq NM^+$. 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 unbounded number of names in a single interaction. These two properties can be exploited to show that any attempt at encoding yields contradiction.

The following result details how encoding intensionality into a non-intensional language is impossible by considering an encoding of the simplest intensional language $\mathcal{L}_{A,M,D,I}$ into a non-intensional language.

Theorem 12.1. *These exists no valid encoding from $\mathcal{L}_{A,M,D,I}$ into any language $\mathcal{L}_{\alpha,\beta,\gamma,\delta}$, where $\delta \leq NM^+$.*

Proof. 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 $S_0 = (x).(\langle m\rangle \mid \surd)$ and $S_1 = \langle a\rangle$ where $a \neq m$. Using Proposition 3.5 we have that $S_0 \mid S_1 \longmapsto$ implies $[\![S_0 \mid S_1]\!] \longmapsto$. By derivation of the reduction $[\![S_0 \mid S_1]\!] \longmapsto$ there must exist an interaction 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 $S_3 \stackrel{\text{def}}{=} (\ulcorner a_1 \urcorner \bullet \ldots \bullet \ulcorner a_{k+2} \urcorner).(\langle m\rangle \mid \surd)$ where S_2 outputs $k+2$ distinct names (that are also distinct from a and m) in a single term, and S_3 matches all of these names in a single intensional pattern.

Since $S_2 \mid S_0 \longmapsto$ from Proposition 3.5 it must be that $[\![S_2 \mid S_0]\!] \longmapsto$ for the encoding to be valid. Now consider the interaction with the greatest arity k' in the reduction derivation for $[\![S_2 \mid S_0]\!] \longmapsto$:

- If $k' \leq k$[4] (this must hold for $\beta = M$, i.e. encoding into a monadic language since $k = 1$) consider the interaction in $[\![S_2 \mid S_3]\!] \longmapsto$ with the maximal arity j and that must exist by Proposition 3.5 since $S_2 \mid S_3 \longmapsto$. Now consider the relationship of j and k.

  1. If $j \leq k$ (this must hold for $\beta = M$, i.e. encoding into a monadic language since $k = 1 = j$) 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 name-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:

$$S_4 \stackrel{\text{def}}{=} (\ulcorner a_1 \urcorner \bullet \ldots \ulcorner a_{i-1} \urcorner \bullet \ulcorner m \urcorner \bullet \ulcorner a_{i+1} \urcorner \ldots \ulcorner a_{k+2} \urcorner).(\langle a_i\rangle \mid \surd)\ .$$

     Now consider the context $C(\cdot) = C^{\mathcal{N}}_{|}([\![\,S_2\,]\!], [\![\,\cdot\,]\!]) = [\![\,S_2 \mid \cdot\,]\!]$ where $\mathcal{N} = \{\widetilde{a} \cup m\}$.
     From Proposition 3.3 it follows that $C([\![\,\mathbf{0}\,]\!]) \mapsto\!\!\!\!\!\!\rightarrow$ and $C([\![\,S_4\,]\!]) \mapsto\!\!\!\!\!\!\rightarrow$. From Proposition 3.5, we have that $C([\![\,S_3\,]\!]) \longmapsto$. We proceed by structural induction on $C(\cdot)$ to show that a contradiction is eventually reached.

     - $C(\cdot) = (\nu n)C'(\cdot)$. It implies that $C'([\![\,S_4\,]\!]) \mapsto\!\!\!\!\!\!\rightarrow$ and $C'([\![\,S_3\,]\!]) \longmapsto$ and the result i.e. a contradiction, follows by induction. Informally, the argument is that restriction cannot prevent or create reductions.

[4] We only need $k' = k$ for this case, but the proof applies to any $k' \leq k$.

– $C(\cdot) = C'(\cdot) \mid T$, for some term T. There are three ways the reduction $C([\![S_3]\!]) \longmapsto$ can occur:
  * If $C'([\![S_3]\!]) \longmapsto$ and $T \not\longmapsto$ then $C'([\![S_4]\!]) \not\longmapsto$ and the contradiction follows by induction.
  * If $T \longmapsto$ then $C([\![S_4]\!]) \longmapsto$ and we reach a contradiction.
  * Lastly, the reduction is a synchronisation between $C'([\![S_3]\!])$ and T. Suppose w.l.o.g. that $C'([\![S_3]\!])$ contributes with the input action. It implies that $C'([\![S_4]\!])$ cannot perform an input action to match the output of T. We can consider the translation context for the InProc $[\![S_3]\!] = C^{\mathcal{N}'}_{InProc}([\![(\ulcorner a_1 \urcorner \bullet \ldots \bullet \ulcorner a_{k+2} \urcorner)]\!], [\![\langle m \rangle \mid \surd]\!])$ with $\mathcal{N}' = \mathcal{N}$. Given the substitution $\sigma = \{m/a_i, a_i/m\}$ we have by name invariance of the valid encoding that there exists σ' such that $[\![\sigma S_4]\!] = \sigma'[\![S_4]\!] = [\![S_3]\!]$. It follows that $[\![S_4]\!] = C^{\mathcal{N}'}_{InProc}([\![(\ulcorner a_1 \urcorner \bullet \ldots \ulcorner m \urcorner \ldots \bullet \ulcorner a_{k+2} \urcorner)]\!], [\![\langle a_i \rangle \mid \surd]\!])$. We proceed by structural induction on $C'(C^{\mathcal{N}'}_{InProc}(\_1; \_2))$. The base case, when the context is empty, implies that if T synchronizes with $C'([\![S_3]\!])$ but not with $C'([\![S_4]\!])$ it is due to a name matching capability in the inputs $[\![(\ulcorner a_1 \urcorner \bullet \ldots \bullet \ulcorner a_{k+2} \urcorner)]\!]$ and $[\![(\ulcorner a_1 \urcorner \bullet \ldots \ulcorner m \urcorner \ldots \bullet \ulcorner a_{k+2} \urcorner)]\!]$.
    · If either a_i or m is matched in the synchronisation, this contradicts our hypothesis that T synchronizes with $C'([\![S_3]\!])$ but not with $C'([\![S_4]\!])$ due to a name matching capability.
    · One of the two inputs uses non-linearity capability and the other does not. Suppose that the first input is of the form $b(x, x, \cdots)$ (we can use non linearity for the first two arguments w.l.o.g.); and the second input is of the form $b(x, y, \cdots)$, for $x \neq y$. However $[\![(\ulcorner a_1 \urcorner \bullet \ldots \ulcorner a_i \urcorner \ldots \bullet \ulcorner a_{k+2} \urcorner)]\!] = \sigma'[\![(\ulcorner a_1 \urcorner \bullet \ldots \ulcorner m \urcorner \ldots \bullet \ulcorner a_{k+2} \urcorner)]\!]$ and $\widetilde{a}, m$ are distinct by definition and a substitution cannot change binding names. This case is therefore not possible either.

  Let us now consider the process constructors that can block an input:
    · $(\nu n)C''(\_1; \_2)$. If $C''([\![(\ulcorner a_1 \urcorner \bullet \ldots \ulcorner m \urcorner \ldots \bullet \ulcorner a_{k+2} \urcorner)]\!]; \_2)$ blocks the input then the name n is affected by the substitution σ'. This is not possible as n is bound.
    · **if** $c_1 = c_2$ **then** T_1 **else** T_2, where T_1 and T_2 are possibly contexts. The only interesting case is if the equality test is affected by the substitution σ' as otherwise the result follows by induction. The case where an **if** $c_1 = c_2$ **then** T_1 **else** T_2 construct blocks the synchronisation in $\sigma[\![S_3]\!]$ is not possible as follows from Lemma 3.7.

  The remaining process constructors cannot block an input (except for InProc or OutProc when $\alpha = S$ but these cannot block the initial interaction).
– $C(\cdot) = \text{OutProc}.C'(\cdot)$ or $C(\cdot) = \text{IN}.C'(\cdot)$ then $C([\![S_3]\!]) \not\longmapsto$ and we reach a contradiction.

- $C(\cdot) = \cdot$ similarly, as $C([\![S_3]\!]) \mapsto\!\!\!\!\!\!\rightarrow$.
- $C(\cdot) =$ **if** $c_1 = c_2$ **then** $C'(\cdot)$ **else** $C''(\cdot)$. 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 $[\![\sigma S_4]\!] = \sigma'[\![S_4]\!] = [\![S_3]\!]$. There are two cases: either the substitution changes the values of the equality test $c_1 = c_2$ or it does not. In the first case the result follows by induction from either $C'(\cdot)$ or $C''(\cdot)$. In the latter, we reach a contradiction as we have shown above using Lemma 3.7.

2. If $j > k$ (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, from Proposition 3.5 $[\![S_2 \mid \cdot]\!] = C_{|}^{N}([\![S_2]\!], [\![\cdot]\!])$ 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. Moreover these reductions are due to interaction between $[\![S_2]\!]$ and $[\![S_0]\!]$, $[\![S_3]\!]$, respectively.
   We use Lemma 3.6 on $[\![S_2]\!]$ and $[\![S_0]\!]$ and $[\![S_3]\!]$ to show that this case is not possible: the two reductions are due to the same action in $[\![S_2]\!]$ and thus have the same arity.

- If $k' > k$ then proceed as the second case above for the term $[\![S_0 \mid S_1 \mid S_2]\!]$ which can do two reductions one of arity k and one of arity k'.

□

The proof above shows the techniques for proving the impossibility of encoding intensionality into a non-intensional language. The general case below easily follows by using the above result since the chosen examples are a subset of the processes for all other languages (the only non-trivial case is channel-based, by taking all channel-names to be a single fixed name b unrelated to any of the other names in the above proof).

Corollary 12.2. *There is no valid encoding of any language $\mathcal{L}_{-,-,-,I}$ into $\mathcal{L}_{-,-,-,\delta}$ where $\delta \leq NM^{+}$.*

Proof. Suppose, by contradiction, that there exists a valid encoding from $\mathcal{L}_{-,-,-,I}$ into $\mathcal{L}_{-,-,-,\delta}$ where $\delta \leq NM^{+}$. Using the identity encodings one can compose a path from $\mathcal{L}_{A,M,D,I}$ to any $\mathcal{L}_{-,-,-,I}$ language. Thus one obtains a valid encoding from $\mathcal{L}_{A,M,D,I}$ to a $\mathcal{L}_{-,-,-,\delta}$ language using Theorem 8.3. This yields contradiction with Theorem 12.1. □

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 [23, 24], show that languages that allow for an arbitrary number of names to be matched in interaction cannot be encoded into languages that can only match a limited number of names in interaction.

13. Non-Linear Intensionality

This section considers non-linearity in the presence of intensionality. Observe that Theorem 8.3 ensures that linear intensionality can always be encoded into non-linear

intensionality. We now show that non-linear intensionality cannot be encoded into linear intensionality. To do this, we rely on the following result.

Theorem 13.1. *There exists no valid encoding from $\mathcal{L}_{A,M,D,I^+}$ into $\mathcal{L}_{A,M,D,I}$.*

Proof. The proof is by contradiction, let us assume that there exists a valid encoding $[\![\cdot]\!]$. Consider the $\mathcal{L}_{A,M,D,I^+}$ processes $S_1 = \langle a \rangle$ and $S_2 = (x \bullet y).\surd$ and $S_3 = (x \bullet x).\surd$ and the substitutions $\sigma = \{b \bullet c/a\}$ and $\sigma_1 = \{b/c\}$ and their counterparts σ' and σ'_1 by validity of the encoding. Observe that $\sigma(S_1) \mid S_2 \Downarrow_1$ and $\sigma_1\sigma(S_1) \mid S_2 \Downarrow_1$ and $\sigma_1\sigma(S_1) \mid S_3 \Downarrow_1$ and so it follows for $N = \{a\}$ that $C^N_{|}(\sigma'[\![S_1]\!], [\![S_2]\!]) \longmapsto$ and $C^N_{|}(\sigma'_1\sigma'[\![S_1]\!], [\![S_2]\!]) \longmapsto$ and $C^N_{|}(\sigma'_1\sigma'[\![S_1]\!], [\![S_3]\!]) \longmapsto$ by Proposition 3.5 and also from $\sigma(S_1) \mid S_3 \not\longmapsto$ and Proposition 3.3 that $C^N_{|}(\sigma'[\![S_1]\!], [\![S_3]\!]) \not\longmapsto$.

Now consider the difference between $\sigma'[\![S_1]\!]$ and $\sigma'_1\sigma'[\![S_1]\!]$. There are four ways that reduction $C^N_{|}(\sigma'[\![S_1]\!], [\![S_3]\!]) \not\longmapsto$ can be prevented.

- If $\sigma'[\![S_1]\!]$ interacts with an output $\langle \sigma' s \rangle$ and $\sigma'_1\sigma'[\![S_1]\!]$ with $\langle \sigma'_1\sigma' s \rangle$ then the difference must be some name or structure in $\langle \sigma'_1\sigma' s \rangle$. Consider the input (p_3) from $[\![S_3]\!]$ such that $\{\sigma' s /\!\!/ p_3\}$ is undefined while $\{\sigma'_1\sigma' s /\!\!/ p_3\}$ is defined. This can only occur when some name or structure in $\sigma' s$ does not match with some name or structure in p_3. Let us write $t = \sigma' s$. Proceed by induction on the structure of p_3:

  - If $p_3 = x$, for x a binding name then $\{t /\!\!/ x\}$ is always defined and this case is not possible.
  - If $p_3 = \ulcorner n \urcorner$ for a name n, and $t \neq n$. Observe that t must be a single name since otherwise $\sigma'_1 t$ could not be n as required by $\{\sigma'_1 t /\!\!/ \ulcorner n \urcorner\}$ being defined (otherwise if t is not a single name the only other possibility is one of the other cases below). We have three cases.
    - if $t \notin \varphi_{[\![ ]\!]}(b) \cup \varphi_{[\![ ]\!]}(c)$ then t is in the names of the context of $[\![\sigma_1 S_1]\!]$. To make the context explicit we rewrite $\sigma_1 S_1 \equiv \sigma_1 S_1 \mid \mathbf{0}$. We have then $[\![\sigma_1 S_1 \mid \mathbf{0}]\!] = C^{\{b,c\}}_{|}([\![\sigma_1 S_1]\!], [\![\mathbf{0}]\!])$ with $t \in \mathsf{names}(C)$. As the substitution changes t, it follows that $t \notin \mathsf{bn}(C)$, hence $t \in \mathsf{fn}(C)$. Therefore t depends on the structure of the output: whether the output is intensional or how many different names are used. We take the process $\langle a \bullet d \rangle$ and have that t is an output in $[\![\langle a \bullet d \rangle]\!]$. As the substitution acts on t we have that $[\![\sigma_1\langle a \bullet d \rangle \mid \mathbf{0} \mid S_3]\!] \longmapsto_2$ whereas $\sigma_1\langle a \bullet d \rangle \mid \mathbf{0} \mid S_3 \mapsto\!\!\!\!\!\rightarrow_1$.
    - if $t \in \varphi_{[\![ ]\!]}(b)$ then from the property of valid renaming policies we have that

      $$\sigma'_1 t = n \qquad \sigma'_1(\varphi_{[\![ ]\!]}(b)) = \varphi_{[\![ ]\!]}(\sigma_1 b) = \varphi_{[\![ ]\!]}(b)$$

      and therefore $n \in \varphi_{[\![ ]\!]}(b)$. From the condition on renamings that for any two names u, v, $\varphi_{[\![ ]\!]}(u) \cap \varphi_{[\![ ]\!]}(v) = \emptyset$ it follows that for any name $u \notin \{b, c\}$, $t \notin \varphi_{[\![ ]\!]}(u)$. As $\{b, c\} \not\subset \mathsf{fn}(S_3)$, it follows that $n \notin p_3$. Contradiction with our assumption that $p_3 = \ulcorner n \urcorner$.

* if $t \in \varphi_{[\![\,]\!]}(c)$ then again, from $\sigma'_1 t = n$ and from $\sigma'_1(\varphi_{[\![\,]\!]}(c)) = \varphi_{[\![\,]\!]}(\sigma_1 c) = \varphi_{[\![\,]\!]}(b)$, we have that $n \in \varphi_{[\![\,]\!]}(b)$. As above, this contradicts the assumption that $p_3 = \ulcorner n \urcorner$.

– If $p_3 = q_1 \bullet q_2$ and $\sigma'_1 t$ is a compound then we have several cases:

* If t is not a compound, i.e. $t = n$, for n a name, then let us suppose that $[\![\, S_2 \,]\!]$ interacts with an input (p_2) for the reduction $\sigma'[\![\, S_1 \,]\!] \mid [\![\, S_2 \,]\!] \longmapsto$. Then note that p_2 is not a compound pattern for the reduction to work. However $\sigma'_1\sigma'[\![\, S_1 \,]\!] \mid [\![\, S_2 \,]\!] \longmapsto$ where $\sigma'_1 t$ is a compound and therefore the match between $\sigma'_1 t$ and p_2 is undefined. Using Lemma 3.6 on σS_1, $\sigma_1\sigma S_1$ and S_2, we have that the same input action p_2 is used for both reductions above and we therefore reach a contradiction.
* If $t = t_1 \bullet t_2$ is a compound and $\{t_1 \bullet t_2 /\!\!/ q_1 \bullet q_2\}$ is undefined it must be that either $\{t_1 /\!\!/ q_1\}$ or $\{t_2 /\!\!/ q_2\}$ is undefined. This case follows by induction.

- If $\sigma'[\![\, S_1 \,]\!]$ interacts with an input (p), then so does $\sigma'_1\sigma'[\![\, S_1 \,]\!]$ and we have that both $[\![\, S_2 \,]\!]$ and $[\![\, S_3 \,]\!]$ interact using an output. As the substitution σ'_1 enables the interaction with $[\![\, S_3 \,]\!]$, it implies that there is a name match in (p). Let us assume $(p) = (\ulcorner n \urcorner)$, for some name n, as other cases reduce to this one. Then $\sigma'_1(\ulcorner n \urcorner) = (\ulcorner n_1 \urcorner)$, for some name n_1 with $n \neq n_1$. Therefore the output of $[\![\, S_3 \,]\!]$ is $\langle n_1 \rangle$. But as $\sigma'[\![\, S_1 \,]\!]$ interacts with $[\![\, S_2 \,]\!]$ it follows that the output of $[\![\, S_2 \,]\!]$ is $\langle n_1 \rangle$ and therefore this output cannot be used for the interaction with $\sigma'_1\sigma'[\![\, S_1 \,]\!]$. Again, we use Lemma 3.6 on σS_1, $\sigma_1\sigma S_1$ and S_2, to conclude that the same output is used for both interactions of $[\![\, S_2 \,]\!]$ and we reach a contradiction.

- If the reduction is blocked by a restriction (νd) then it is sufficient to observe that substitutions cannot change the bound names.

- Otherwise the reduction must be prevented by a conditional **if** $d_1 = d_2$ **then** T_1 **else** T_2 where the relation $d_1 = d_2$ relates b and c and then T_1 interacts with $[\![\, S_3 \,]\!]$ and T_2 does not (or vice versa).

  First note that if $d_1 = d_2$ in $\sigma'[\![\, S_1 \,]\!]$, then $\sigma'_1(d_1) = \sigma'_1(d_2)$ in $\sigma'_1\sigma'[\![\, S_1 \,]\!]$. Therefore it must be that $d_1 \neq d_2$ in $\sigma'[\![\, S_1 \,]\!]$ and $\sigma'_1(d_1) = \sigma'_1(d_2)$ in $\sigma'_1\sigma'[\![\, S_1 \,]\!]$. We conclude that there exists an action in T_2 which is used to synchronise with $[\![\, S_2 \,]\!]$ and we recall one of the cases above to show why the synchronisation between that action in T_2 and $[\![\, S_3 \,]\!]$ does not work.

□

Corollary 13.2. *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, non-linear intensionality alone is sufficient to encode: synchrony, polyadicity, channel-based communication, and intensionality.*

Proof. A straightforward adaptation of Theorem 9.1 and Theorem 8.3. □

Theorem 13.3. *The non-linear intensional languages are more expressive than the linear intensional languages.*

Proof. From Theorem 13.1 there is no valid encoding from $\mathcal{L}_{A,M,D,I^+}$ into $\mathcal{L}_{A,M,D,I}$. $\mathcal{L}_{A,M,D,I^+}$ is the least expressive language among the intensional non-linear languages and therefore the result holds for all other intensional non-linear languages. To reason for all intensional linear languages we use Proposition 11 of [24] which states that for two valid encodings $[\![\cdot]\!]$ and $\{\cdot\}$ and such that $[\![\cdot]\!]$ is composable then their composition $[\![\{\cdot\}]\!]$ is also a valid encoding. Suppose then, by contradiction, that for any language $\mathcal{L}_{\alpha,\beta,\delta,I}$, there exists a valid encoding $\{\cdot\}$ from $\mathcal{L}_{A,M,D,I^+}$ into $\mathcal{L}_{\alpha,\beta,\delta,I}$. We can use the encodings from $\mathcal{L}_{\alpha,\beta,\delta,I}$ into $\mathcal{L}_{A,M,D,I}$ which from Corollaries 5.5, 6.2 and 7.2 are composable. We obtain then a valid encoding from $\mathcal{L}_{A,M,D,I^+}$ into $\mathcal{L}_{A,M,D,I}$. Contradiction. □

This concludes showing that non-linear intensionality is strictly more expressive than linear intensionality. The result is not particularly surprising, although it is distinct from similar reasoning in calculi for computation, where non-linear patterns are considered equally expressive as linear patterns [30]. The distinction is that in the calculi for computation generally reduction occurs via application regardless of successful matching, whereas in concurrency reduction may not occur at all.

14. 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) [19, 20] (and variations thereof [14]) and Psi calculi [2, 3]. However, only one variation of CPC matches any of the family of intensional languages defined here, that being $\mathcal{L}_{S,M,D,I}$ [14]. 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 [40] 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 [19, 18, 20]. Alternatively, considering logics that play a rôle in communication would allow for capturing the behaviours of languages like Concurrent Constraint Programming [42] or Psi calculi. Again the rôle of logics has been used to show separation of Psi calculi from CPC [18].

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 most 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 [38, 6, 7, 11, 27, 23, 19, 14, 20]. The choice of valid encodings here is that used, sometimes with mild adaptations, in [25, 24, 19, 37, 14, 20] and has also inspired similar works [32, 33, 44]. However, there are alternative approaches to encoding criteria or comparing expressive power [4, 12, 7, 39, 44]. Further arguments for, and against, the valid encodings here can be found in [25, 24, 44, 20].

Some of the languages introduced here correspond to existing calculi. 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 [35, 36, 34]. The language $\mathcal{L}_{A,P,D,NM}$ aligns with LINDA[13]; 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 [10] or semantic-π [9]. The intensional languages do not exactly match any well-known calculi. However, the language $\mathcal{L}_{S,M,D,I}$ has been mentioned in [14, 18], as a variation of Concurrent Pattern Calculus [19, 14], and has a behavioural theory as a specialisation of [18]. Similarly, the language $\mathcal{L}_{S,M,C,I^+}$ is very similar to pattern-matching Spi calculus [26] and Psi calculi [2], albeit with structural channel terms. There are some results that fit in between the original 16 languages of [23] 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 [23] 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 [19, 20]. Meanwhile fusion calculus and Psi calculi are unrelated to CPC in that neither can encode CPC, and CPC cannot encode either of them [19, 14, 20]. 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.

References

[1] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, CCS '97, pages 36–47, New York, NY, USA, 1997. ACM.

[2] 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.

[3] J. Borgström, R. Gutkovas, J. Parrow, B. Victor, and J. Å. Pohjola. A sorted semantic framework for applied process calculi (extended abstract). In *Trustworthy Global Computing*, pages 103–118, 2013.

[4] G. Boudol. Notes on algebraic calculi of processes. In K. R. Apt, editor, *Logics and Models of Concurrent Systems*, pages 261–303. Springer-Verlag New York, Inc., New York, NY, USA, 1985.

[5] G. Boudol. Asynchrony and the Pi-calculus. Research Report RR-1702, INRIA, 1992.

[6] N. Busi, R. Gorrieri, and G. Zavattaro. On the expressiveness of linda coordination primitives. *Information and Computation*, 156(1-2):90–121, 2000.

[7] M. Carbone and S. Maffeis. On the expressive power of polyadic synchronisation in π-calculus. *Nordic Journal of Computing*, 10(2):70–98, May 2003.

[8] L. Cardelli and A. D. Gordon. Mobile ambients. In *Foundations of Software Science and Computation Structures: First International Conference, FoSSaCS '98*, pages 140–155, 1998.

[9] G. Castagna, R. De Nicola, and D. Varacca. Semantic subtyping for the pi-calculus. *Theoretical Computer Science*, 398(1-3):217–242, May 2008.

[10] R. De Nicola, G. L. Ferrari, and R. Pugliese. KLAIM: A kernel language for agents interaction and mobility. *IEEE Transactions on Software Engineering*, 24(5):315–330, 1998.

[11] R. De Nicola, D. Gorla, and R. Pugliese. On the expressive power of klaim-based calculi. *Theoretical Computer Science*, 356(3):387–421, May 2006.

[12] R. de Simone. Higher-level synchronising devices in Meije-SCCS. *Theoretical Computer Science*, 37:245–267, 1985.

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

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

[15] T. Given-Wilson. Expressiveness via intensionality and concurrency. In G. Ciobanu and D. Méry, editors, *Theoretical Aspects of Computing - ICTAC 2014 - 11th International Colloquium, Bucharest, Romania, September 17-19, 2014. Proceedings*, volume 8687 of *Lecture Notes in Computer Science*, pages 206–223. Springer, 2014.

[16] T. Given-Wilson. An intensional concurrent faithful encoding of turing machines. In I. Lanese, A. Lluch-Lafuente, A. Sokolova, and H. T. Vieira, editors, *Proceedings 7th Interaction and Concurrency Experience, ICE 2014, Berlin, Germany, 6th June 2014.*, volume 166 of *EPTCS*, pages 21–37, 2014.

[17] T. Given-Wilson. On the expressiveness of intensional communication. In J. Borgström and S. Crafa, editors, *Proceedings Combined 21st International Workshop on Expressiveness in Concurrency and 11th Workshop on Structural Operational Semantics, EXPRESS 2014, and 11th Workshop on Structural Operational Semantics, SOS 2014, Rome, Italy, 1st September 2014.*, volume 160 of *EPTCS*, pages 30–46, 2014.

[18] T. Given-Wilson and D. Gorla. Pattern matching and bisimulation. In R. De Nicola and C. Julien, editors, *Coordination Models and Languages*, volume 7890 of *Lecture Notes in Computer Science*, pages 60–74. Springer Berlin Heidelberg, 2013.

[19] T. Given-Wilson, D. Gorla, and B. Jay. Concurrent pattern calculus. In C. Calude and V. Sassone, editors, *Theoretical Computer Science*, volume 323 of *IFIP Advances in Information and Communication Technology*, pages 244–258. Springer Berlin Heidelberg, 2010.

[20] T. Given-Wilson, D. Gorla, and B. Jay. A Concurrent Pattern Calculus. *Logical Methods in Computer Science*, 10(3), 2014.

[21] T. Given-Wilson and A. Legay. On the expressiveness of joining. In S. Knight, I. Lanese, A. Lluch-Lafuente, and H. T. Vieira, editors, *Proceedings 8th Interaction and Concurrency Experience, ICE 2015, Grenoble, France, 4-5th June 2015.*, volume 189 of *EPTCS*, pages 99–113, 2015.

[22] T. Given-Wilson and A. Legay. On the expressiveness of symmetric communication. In A. Sampaio and F. Wang, editors, *Theoretical Aspects of Computing - ICTAC 2016 - 13th International Colloquium, Taipei, Taiwan, ROC, October 24-31, 2016, Proceedings*, volume 9965 of *Lecture Notes in Computer Science*, pages 139–157, 2016.

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

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

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

[26] C. Haack and A. Jeffrey. Pattern-matching spi-calculus. *Information and Computation*, 204(8):1195–1263, Aug. 2006.

[27] B. Haagensen, S. Maffeis, and I. Phillips. Matching systems for concurrent calculi. *Electronic Notes in Theoretical Computer Science*, 194(2):85 – 99, 2008. Proceedings of the 14th International Workshop on Expressiveness in Concurrency (EXPRESS 2007).

[28] K. Honda and M. Tokoro. An object calculus for asynchronous communication. In *ECOOP'91 European Conference on Object-Oriented Programming*, pages 133–147. Springer, 1991.

[29] K. Honda and N. Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 152:437–486, 1995.

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

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

[32] I. Lanese, J. Pérez, D. Sangiorgi, and A. Schmitt. On the expressiveness of polyadic and synchronous communication in higher-order process calculi. In S. Abramsky, C. Gavoille, C. Kirchner, F. Meyer auf der Heide, and P. Spirakis, editors, *Automata, Languages and Programming*, volume 6199 of *Lecture Notes in Computer Science*, pages 442–453. Springer Berlin Heidelberg, 2010.

[33] I. Lanese, C. Vaz, and C. Ferreira. On the expressive power of primitives for compensation handling. In *Proceedings of the 19th European Conference on Programming Languages and Systems*, ESOP'10, pages 366–386, Berlin, Heidelberg, 2010. Springer-Verlag.

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

[35] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I. *Information and Computation*, 100(1):1–40, Sept. 1992.

[36] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, II. *Information and Computation*, 100(1):41–77, Sept. 1992.

[37] L. Nielsen, N. Yoshida, and K. Honda. Multiparty symmetric sum types. In *Proceedings of the 17th International Workshop on Expressiveness in Concurrency (EXPRESS 2010)*, pages 121–135, 2010.

[38] C. Palamidessi. Comparing the expressive power of the synchronous and asynchronous pi-calculi. *Mathematical. Structures in Comp. Sci.*, 13(5):685–719, Oct. 2003.

[39] J. Parrow. Expressiveness of process algebras. *Electronic Notes in Theoretical Computer Science*, 209:173–186, Apr. 2008.

[40] J. Parrow and B. Victor. The fusion calculus: expressiveness and symmetry in mobile processes. In *Proceedings of Thirteenth Annual IEEE Symposium on Logic in Computer Science*, pages 176–185, Jun 1998.

[41] P. Quaglia and D. Walker. Types and full abstraction for polyadic π-calculus. *Information and Computation*, 200(2):215 – 246, 2005.

[42] V. A. Saraswat, M. Rinard, and P. Panangaden. The semantic foundations of concurrent constraint programming. In *Proceedings of the 18th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '91, pages 333–352, New York, NY, USA, 1991. ACM.

[43] R. van Glabbeek. On the validity of encodings of the synchronous in the asynchronous π-calculus. *CoRR*, abs/1802.09182, 2018.

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

[45] G. Winskel. Event structures. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Applications and Relationships to Other Models of Concurrency*, pages 325–392, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.