On the Expressiveness of Joining

Thomas Given-Wilson
Inria, France

Axel Legay
Inria, France

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 vs intensionality). Here another dimension *coordination* is considered that accounts for the number of processes required for an interaction to occur. Coordination generalises binary languages such as π-calculus to *joining* languages that combine inputs such as the Join Calculus and general rendezvous calculus. By means of possibility/impossibility of encodings, this paper shows coordination is unrelated to the other features. That is, joining languages are more expressive than binary languages, and no combination of the other features can encode a joining language into a binary language. Further, joining is not able to encode any of the other features unless they could be encoded otherwise.

1 Introduction

The expressiveness of process calculi based upon their choice of communication primitives has been explored before [31, 5, 9, 19, 12, 14]. In [19] and [14] 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 versus intensionality. These features are able to represent many popular calculi [19, 14] such as: asynchronous or synchronous, monadic or polyadic π-calculus [28, 29, 27]; Linda [11]; Mobile Ambients [7]; μKlaim [30]; semantic-π [8]; and asymmetric concurrent pattern calculus [13]. Also the intensional features capture significant aspects of Concurrent Pattern Calculus (CPC) [16, 17] and variations [12, 13]; and Psi calculi [1] and sorted Psi calculi [4].

Typically interaction in process calculi is a binary relation, where two processes interact and reduce to a third process. For example in π-calculus the interaction rule is

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

Here the processes $\overline{m}\langle a\rangle.P$ and $m(x).Q$ interact and reduce to a new process $P \mid \{a/x\}Q$. However, there are process calculi that are not binary with their interactions. For example, Concurrent Constraint Programming (CCP) has no direct interaction primitives, instead interactions are between a single process and the constraint environment [32]. In the other direction Join Calculus [10], general rendezvous calculus [2], and m-calculus [33] allow any number of processes to join in a single interaction.

This paper abstracts away from specific calculi in the style of [19, 14] to provide a general account of the expressiveness of the *coordination* of communication primitives. Here coordination can be either *binary* between an explicit input and output (as above), or *joining* where the input may interact with unbounded outputs (but at least one). For example, consider the reduction

$$\overline{m}\langle a\rangle.P_1 \mid \overline{n}\langle b\rangle.P_2 \mid (m(x) \mid n(y)) \triangleright Q \quad \longmapsto \quad P_1 \mid P_2 \mid \{a/x, b/y\}Q$$

S. Knight, I. Lanese, A. Lluch Lafuente and H. T. Vieira (Eds.):
8th Interaction and Concurrency Experience (ICE 2015)
EPTCS 189, 2015, pp. 99–113, doi:10.4204/EPTCS.189.9

where the join $\triangleright$ interacts when the two outputs $\overline{m}\langle a\rangle$ and $\overline{n}\langle b\rangle$ can match the two parts of the input $m(x)$ and $n(y)$, respectively.

By adding the dimension of coordination, the original 24 calculi of [19, 14] are here expanded to 48. This paper details the relations between these calculi, with the following key results.

Joining cannot be encoded into a binary language. This is formalised via the *coordination degree* of a language that is the least upper bound on the number of processes required to yield a reduction. In general a language with a greater coordination degree cannot be encoded into a language with a lesser coordination degree. That is, the joining languages with ∞ coordination degree cannot be encoded into the binary languages with coordination degree 2.

Joining synchronous languages can be encoded into joining asynchronous languages when their binary counterparts allow an encoding from a synchronous language into an asynchronous one. In the other direction synchronous languages cannot be encoded into asynchronous languages that differ only by the addition of joining over binary communication.

Polyadic languages that cannot be encoded into monadic languages in the binary setting cannot be encoded into monadic languages simply with the addition of joining. Indeed, coordination is unrelated to arity despite being similar in having a base case (monadic/binary) and an unbounded case (polyadic/joining).

Channel-based languages cannot be encoded into dataspace-based languages by the addition of joining unless they could be encoded already. In the other direction, the addition of channels does not allow a joining language to be encoded into a binary language.

Intensionality cannot be encoded by joining regardless of other features, this result mirrors the general result that intensionality cannot be represented by any combination of the first four features [14]. Name-matching cannot be encoded by joining into a language without any name-matching, despite the possibility of matching unbounded numbers of names via joining on an unbounded number of channels.

Overall, the results of this paper prove that joining is orthogonal to all the other features, and that joining languages are strictly more expressive than binary languages.

The structure of the paper is as follows. Section 2 introduces the 48 calculi considered here. Section 3 revises the criteria used for encoding and comparing calculi. Section 4 defines the coordination degree of a language and formalises the relation between binary and joining languages. Section 5 considers the relation between synchronism and coordination. Section 6 relates arity and coordination. Section 7 presents results contrasting communication medium with coordination. Section 8 formalises the relation between pattern-matching and coordination. Section 9 concludes, discusses future and related work, and provides 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. With the exception of the joining constructs this is a repetition of prior definitions from [14].

Assume a countable set of names $\mathcal{N}$ ranged over by $a, b, c, \ldots$. Traditionally in π-calculus-style calculi names are used for channels, input bindings, and output data. However, here these are generalised to account for structure. Then, define the *terms* (denoted with $s, t, \ldots$) to be $s, t ::= a \mid s \bullet t$. Terms 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 *input patterns*. The non-pattern-matching languages will simply use binding names (denoted $x,y,z,\ldots$). The *name-matching* patterns, denoted $m,n,o,\ldots$ and defined by $m,n ::= x \mid \ulcorner a \urcorner$ consist of either a *binding name* x, or a *name-match* $\ulcorner a \urcorner$. Lastly the *intensional patterns* (denoted $p,q,\ldots$) will also consider structure and are defined by $p,q ::= m \mid p \bullet q$. The binding names x and name-match $\ulcorner a \urcorner$ are contained in m from the name-matching calculi, the *compound pattern* $p \bullet q$ combines p and q into a single pattern, and is left associative. The free names and binding names of name-matching and intensional patterns are as expected, taking the union of sub-patterns for compound patterns. Note that an intensional pattern is well-formed if and only if all binding names within the pattern are pairwise distinct. The rest of this paper will only consider well-formed intensional patterns.

The (parametric) syntax for the languages is:

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

The different languages are obtained by replacing the output *OutProc* and input *InProc* with the various definitions. The rest of the process forms are as 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 [21, 12].

This paper considers the possible combinations of five features for communication: *synchronism* (asynchronous vs synchronous), *arity* (monadic vs polyadic data), *communication medium* (dataspace-based vs channel-based), *pattern-matching* (simple binding vs name equality vs intensionality), and *co-ordination* (binary vs joining). As a result there exist 48 languages denoted as $\Lambda_{s,a,m,p,b}$ whose generic element is denoted as $\mathcal{L}_{\alpha,\beta,\gamma,\delta,\epsilon}$ where:

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

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 productions in Figure 1. 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 input patterns.

As usual $a(\ldots,x,\ldots).P$ and $(\nu x)P$ and $(x \bullet \ldots).P$ and $(\ldots \mid a(x) \mid \ldots) \triangleright P$ bind x in P. Observe that in $a(\ldots,\ulcorner b \urcorner,\ldots).P$ and $(\ldots \bullet \ulcorner b \urcorner).P$ neither a nor b bind in P, both are free. The corresponding notions of free and bound names of a process, denoted $\mathsf{fn}(P)$ and $\mathsf{bn}(P)$, are as usual. Also note that α-equivalence, denoted $=_\alpha$ is assumed in the usual manner. Lastly, an input is well-formed if all binding names in that input occur exactly once. This paper shall only consider well-formed inputs. Finally, the structural equivalence relation $\equiv$ is defined by:

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

$$\textbf{if } s = t \textbf{ then } P \textbf{ else } Q \equiv P \quad s = t \qquad \textbf{if } s = t \textbf{ then } P \textbf{ else } Q \equiv Q \quad s \neq t$$

$$P \equiv P' \quad \text{if } P =_\alpha P' \qquad (\nu a)\mathbf{0} \equiv \mathbf{0} \qquad (\nu a)(\nu b)P \equiv (\nu b)(\nu a)P$$

$$P \mid (\nu a)Q \equiv (\nu a)(P \mid Q) \quad \text{if } a \notin \mathsf{fn}(P) \qquad *P \equiv P \mid *P .$$

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

Figure 1: Syntax of Languages.

Observe that $\mathcal{L}_{A,M,C,NO,B}$, $\mathcal{L}_{A,P,C,NO,B}$, $\mathcal{L}_{S,M,C,NO,B}$, and $\mathcal{L}_{S,P,C,NO,B}$ align with the communication primitives of the asynchronous/synchronous monadic/polyadic π-calculus [28, 29, 27]. The language $\mathcal{L}_{A,P,D,NM,B}$ aligns with LINDA[11]; the languages $\mathcal{L}_{A,M,D,NO,B}$ and $\mathcal{L}_{A,P,D,NO,B}$ with the monadic/polyadic Mobile Ambients [7]; and $\mathcal{L}_{A,P,C,NM,B}$ with that of μKLAIM [30] or semantic-π [8]. The intensional languages do not exactly match any well-known calculi. However, the language $\mathcal{L}_{S,M,D,I,B}$ has been mentioned in [12], as a variation of Concurrent Pattern Calculus [16, 12], and has a behavioural theory as a specialisation of [15]. Similarly, the language $\mathcal{L}_{S,M,C,I,B}$ is very similar to pattern-matching Spi calculus [22] and Psi calculi [1], albeit without the assertions or the possibility of repeated binding names in patterns. There are also similarities between $\mathcal{L}_{S,M,C,I,B}$ and the polyadic synchronous π-calculus of [6], although the intensionality is limited to the channel, i.e. inputs and outputs of the form $s(x).P$ and $\overline{s}\langle a \rangle.P$ respectively. For the joining languages: $\mathcal{L}_{A,P,C,NO,J}$ represents Join Calculus [10]; and $\mathcal{L}_{S,P,C,NO,J}$ the general rendezvous calculus [2], and m-calculus [33], although the latter has higher order constructs and other aspects that are not captured within the features here.

Remark 2.1. *The languages $\Lambda_{s,a,m,p,\epsilon}$ can be easily ordered; in particular $\mathcal{L}_{\alpha_1,\beta_1,\gamma_1,\delta_1,\epsilon_1}$ can be encoded into $\mathcal{L}_{\alpha_2,\beta_2,\gamma_2,\delta_2,\epsilon_2}$ if it holds that $\alpha_1 \leq \alpha_2$ and $\beta_1 \leq \beta_2$ and $\gamma_1 \leq \gamma_2$ and $\delta_1 \leq \delta_2$ and $\epsilon_1 \leq \epsilon_2$, where $\leq$ is the least reflexive relation satisfying the following axioms:*

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

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 **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, binary communication is joining communication with all joining inputs having only a single input pattern.*

The operational semantics of the languages is given here via reductions as in [27, 24, 14]. An alternative style is via a *labelled transition system* (LTS) such as [19]. Here the reduction based style is

to simplify having to define here the (potentially complex) labels that occur when both intensionality and joining are in play. However, the LTS style can be used for intensional languages [1, 12, 15], and indeed captures many[1] of the languages here [15]. For the joining languages the techniques used in [3] can be used for the no-matching joining languages, with the techniques of [15] used to extend intensionality[2].

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 (also finite domain) from names to terms. The application of a substitution σ to a pattern p is defined by:

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

Where substitution is as usual on names, and on the understanding that the name-match syntax can be applied to any term as follows $\ulcorner x \urcorner \stackrel{\text{def}}{=} \ulcorner x \urcorner$ and $\ulcorner(s \bullet t)\urcorner \stackrel{\text{def}}{=} \ulcorner s \urcorner \bullet \ulcorner t \urcorner$.

Given a substitution σ and a process P, denote with σP the (capture-avoiding) application of σ to P that behaves in the usual manner. Note that capture can always be avoided by exploiting α-equivalence, which can in turn be assumed [34].

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

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

Any term t can be matched with a binding name x to generate a substitution from the binding name to the term $\{t/x\}$. A single name a can be matched with a name-match for that name $\ulcorner a \urcorner$ to yield the empty substitution. A compound term $s \bullet t$ can be matched by a compound pattern $p \bullet q$ when the components match to yield substitutions $\{s/\!/p\} = \sigma_1$ and $\{t/\!/q\} = \sigma_2$, the resulting substitution is the unification of σ_1 and σ_2. Observe that since patterns are well-formed, the substitutions of components will always have disjoint domain. Otherwise the match is undefined.

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

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

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 union of substitutions is the result. (Like the match rule, the union is ensured disjoint domain by well-formedness of inputs.) Otherwise the poly-match rule is undefined, for example when a single match fails, or the sequences are of unequal arity.

Interaction is now defined by the following axiom for the binary languages:

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

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

[2] This has not been proven as yet, however there appears no reason it should not be straightforward albeit very tedious.

and for the joining languages:

$$\overline{s_1}\langle\widetilde{t_1}\rangle.P_1 \mid \ldots \mid \overline{s_k}\langle\widetilde{t_k}\rangle.P_k \mid (s_1(\widetilde{p_1}) \mid \ldots \mid s_k(\widetilde{p_k})) \triangleright Q \quad \longmapsto \quad P_1 \mid \ldots \mid P_k \mid \sigma Q \qquad \sigma = \bigcup_{\{i \in 1\ldots k\}} \textsc{Match}(\widetilde{t_i}; \widetilde{p_i}) \,.$$

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

The general reduction relation $\longmapsto$ includes the interaction axiom for the language in question as well as the following three rules:

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

The reflexive transitive closure of $\longmapsto$ is denoted by $\Longmapsto$.

Lastly, for each language let $\simeq$ denote a reduction-sensitive reference behavioural equivalence for that language, e.g. a barbed equivalence. For the non-intensional languages these are mostly already known, either by their equivalent language in the literature, such as asynchronous/synchronous monadic/polyadic π-calculus or Join Calculus, or from [19]. For the intensional languages the results in [15] can be used. For the other joining languages the techniques used in [3] can be used for the no-matching joining languages, with the techniques of [15] used to extend intensionality[3].

3 Encodings

This section recalls the definition of valid encodings as well as some useful theorems (details in [21]) for formally relating process calculi. The validity of such criteria in developing expressiveness studies emerges from the various works [19, 20, 21], that have also recently inspired similar works [25, 26, 18].

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

Now consider only encodings that satisfy the following properties. Let a *k-ary context* $C(\cdot_1;\ldots;\cdot_k)$ be a term where k occurrences of $\mathbf{0}$ are linearly replaced by the holes $\{\cdot_1;\ldots;\cdot_k\}$ (every one of the k holes must occur once and only once). Denote with $\longmapsto_i$ and $\Longmapsto_i$ the relations $\longmapsto$ and $\Longmapsto$ in language $\mathcal{L}_i$; denote with $\longmapsto_i^\omega$ an infinite sequence of reductions in $\mathcal{L}_i$. Moreover, let $\simeq_i$ denote the reference behavioural equivalence for language $\mathcal{L}_i$. Also, let $P \Downarrow_i$ mean that there exists P' such that $P \Longmapsto_i P'$ and $P' \equiv P'' \mid \surd$, for some P''. Finally, to simplify reading, let S range over processes of the source language (viz., $\mathcal{L}_1$) and T range over processes of the target language (viz., $\mathcal{L}_2$).

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

1. *Compositionality:* for every k-ary operator op of $\mathcal{L}_1$ and for every subset of names N, there exists a k-ary context $C^N_{\mathsf{op}}(\cdot_1;\ldots;\cdot_k)$ 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 $\llbracket \mathsf{op}(S_1,\ldots,S_k) \rrbracket = C^N_{\mathsf{op}}(\llbracket S_1 \rrbracket;\ldots;\llbracket S_k \rrbracket)$.

[3] This has not been proven as yet, however there appears no reason it should not be possible, and the results here rely upon the existence of an equivalence relation, not any particular one.

2. *Name invariance:* for every S and substitution σ, it holds that $[\![\sigma S]\!] = \sigma'[\![S]\!]$ if σ is injective and $[\![\sigma S]\!] \simeq_2 \sigma'[\![S]\!]$ otherwise where σ' is such that $\varphi_{[\![\cdot]\!]}(\sigma(a)) = \sigma'(\varphi_{[\![\cdot]\!]}(a))$ 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 $[\![S]\!] \Downarrow_2$.

Now recall two results concerning valid encodings that are useful for later proofs.

Proposition 3.1 (Proposition 5.5 from [21])**.** *Let $[\![\cdot]\!]$ be a valid encoding; then, $S \longmapsto\!\!\!\!\!\!\not\;\;_1$ implies that $[\![S]\!] \longmapsto\!\!\!\!\!\!\not\;\;_2$.*

Proposition 3.2 (Proposition 5.6 from [21])**.** *Let $[\![\cdot]\!]$ be a valid encoding; then for every set of names N, it holds that $C_{|}^{N}(\cdot_1, \cdot_2)$ has both its holes at top-level.*

4 Joining vs Binary

This section considers the expressive power gained by joining. It turns out that joining adds expressive power that cannot be represented by binary languages regardless of other features.

The expressive power gained by joining can be captured by the concept of the *coordination degree* of a language $\mathcal{L}$, denoted $\textsc{Cd}(\mathcal{L})$, as the least upper bound on the number of processes that must coordinate to yield a reduction in $\mathcal{L}$. For example, all the binary languages $\mathcal{L}_{-,-,-,-,B}$ have coordination degree 2 since their reduction axiom is only defined for two processes. By contrast, the coordination degree of the joining languages is ∞ since there is no bound on the number of inputs that can be part of a join.

Theorem 4.1. *If $\textsc{Cd}(\mathcal{L}_1) > \textsc{Cd}(\mathcal{L}_2)$ then there exists no valid encoding $[\![\cdot]\!]$ from $\mathcal{L}_1$ into $\mathcal{L}_2$.*

Proof. By contradiction, assume there is a valid encoding $[\![\cdot]\!]$. Pick i processes S_1 to S_i where $i = \textsc{Cd}(\mathcal{L}_2) + 1$ such that all these processes must coordinate to yield a reduction and yield success. That is, $S_1 \mid \ldots \mid S_i \longmapsto \surd$ but not if any S_j (for $1 \leq j \leq i$) is replaced by the null process $\mathbf{0}$. By validity of the encoding it must be that $[\![S_1 \mid \ldots \mid S_i]\!] \longmapsto$ and $[\![S_1 \mid \ldots \mid S_i]\!] \Downarrow$.

By compositionality of the encoding $[\![S_1 \mid \ldots \mid S_i]\!] = C_S$ where C_S must be of the form $C_{|}^{\mathcal{N}}([\![S_1]\!], C_{|}^{\mathcal{N}}(\ldots, C_{|}^{\mathcal{N}}([\![S_{i-1}]\!], [\![S_i]\!])\ldots))$. Now consider the reduction $[\![S_1 \mid \ldots \mid S_i]\!] \longmapsto$ that can be at most between $i-1$ processes by the coordination degree of $\mathcal{L}_2$. If the reduction does *not* involve some process $[\![S_j]\!]$ then it follows that $[\![S_1 \mid \ldots \mid S_{j-1} \mid \mathbf{0} \mid S_{j+1} \mid \ldots \mid S_i]\!] \longmapsto$ (by replacing the $[\![S_j]\!]$ in the context C_S with $[\![\mathbf{0}]\!]$). By construction of $S_1 \mid \ldots \mid S_i$ and $\textsc{Cd}(\mathcal{L}_2) < i$ there must exist some such S_j. However, this contradicts the validity of the encoding since $S_1 \mid \ldots \mid S_{j-1} \mid \mathbf{0} \mid S_{j+1} \mid \ldots \mid S_i \not\longmapsto$. The only other possibility is if $[\![S_j]\!]$ blocks the reduction by blocking some $[\![S_k]\!]$. This can only occur when $[\![S_k]\!]$ is either underneath an interaction primitive (e.g. $\overline{s}\langle\widetilde{t}\rangle.[\![S_k]\!]$) or inside a conditional (e.g. **if** $s = t$ **then** $[\![S_k]\!]$). Both require that $[\![S_k]\!]$ not be top level in C_S, which can be proven contradictory by $i-1$ applications of Proposition 3.2. □

Corollary 4.2. *There exists no valid encoding from $\mathcal{L}_{-,-,-,-,J}$ into $\mathcal{L}_{-,-,-,-,B}$.*

In the other direction the result is ensured by Remark 2.1. Thus for any two languages which differ only by one being binary and the other joining, the joining language is strictly more expressive than the binary language.

5 Joining and Synchronicity

This section considers the relation between joining and synchronicity. It turns out that the two are orthogonal and do not influence the other's expressiveness.

It is sufficient to consider the languages $\mathcal{L}_{A,M,D,NO,J}$ and $\mathcal{L}_{A,P,D,NO,J}$ and $\mathcal{L}_{A,M,D,NM,J}$. The other asynchronous joining languages can encode their synchronous joining counterparts in the usual manner [23]. For example, the encoding from $\mathcal{L}_{S,M,C,NO,B}$ into $\mathcal{L}_{A,M,C,NO,B}$ given by

$$\begin{aligned}[[\,\overline{n}\langle a\rangle.P\,]] &\stackrel{\text{def}}{=} (\nu z)(\overline{n}\langle z\rangle \mid z(x).(\overline{x}\langle a\rangle \mid [[\,P\,]]))\\ [[\,n(a).Q\,]] &\stackrel{\text{def}}{=} (\nu x)n(z).(\overline{z}\langle x\rangle \mid x(a).[[\,Q\,]])\end{aligned}$$

can be adapted in the obvious manner for $\mathcal{L}_{S,M,C,NO,J}$ into $\mathcal{L}_{A,M,C,NO,J}$ as follows

$$\begin{aligned}[[\,\overline{n}\langle a\rangle.P\,]] &\stackrel{\text{def}}{=} (\nu z)(\overline{n}\langle z\rangle \mid (z(x)) \triangleright (\overline{x}\langle a\rangle \mid [[\,P\,]]))\\ [[\,(n_1(a_1) \mid \ldots \mid n_i(a_i)) \triangleright Q\,]] &\stackrel{\text{def}}{=} (\nu x_1,\ldots,x_i)(n_1(z_1) \mid \ldots \mid n_i(z_i)) \triangleright\\ &\qquad (\overline{z_1}\langle x_1\rangle \mid \ldots \mid \overline{z_i}\langle x_i\rangle \mid (x_1(a_1) \mid \ldots \mid x_i(a_i)) \triangleright [[\,Q\,]])\,.\end{aligned}$$

The idea for binary languages is that the encoded output creates a fresh name z and sends it to the encoded input. The encoded input creates a fresh name x and sends it to the encoded output along channel name z. The encoded output now knows it has communicated and evolves to $[[\,P\,]]$ in parallel with the original a sent to the encoded input along channel name x. When the encoded input receives this it can evolve to $[[\,Q\,]]$. The joining version is similar except the join synchronises with all the encoded outputs at once, sends the fresh names x_j in parallel, and then synchronises on all the a_j in the last step.

The encoding above is shown for $\mathcal{L}_{S,M,C,NO,J}$ into $\mathcal{L}_{A,M,C,NO,J}$ and is the identity on all other process forms. This can be proven to be a valid encoding.

Lemma 5.1. *Given a $\mathcal{L}_{S,M,C,NO,J}$ input P and output Q then $[[P]] \mid [[Q]] \longmapsto$ if and only if $P \mid Q \longmapsto$.*

Lemma 5.2. *If $P \equiv Q$ then $[[P]] \equiv [[Q]]$. Conversely, if $[[P]] \equiv Q$ then $Q = [[P']]$ for some $P' \equiv P$.*

Lemma 5.3. *The translation $[[\cdot]]$ from $\mathcal{L}_{S,M,C,NO,J}$ into $\mathcal{L}_{A,M,C,NO,J}$ preserves and reflects reductions.*

Theorem 5.4. *There is a valid encoding from $\mathcal{L}_{S,M,C,NO,J}$ into $\mathcal{L}_{A,M,C,NO,J}$.*

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

Corollary 5.5. *If there exists a valid encoding from $\mathcal{L}_{S,\beta,\gamma,\delta,B}$ into $\mathcal{L}_{A,\beta,\gamma,\delta,B}$ then there exists a valid encoding from $\mathcal{L}_{S,\beta,\gamma,\delta,J}$ into $\mathcal{L}_{A,\beta,\gamma,\delta,J}$.*

Proof. Theorem 5.4 applies directly for all channel-based languages. The only other cases can encode channels and so use encodings of the channel-based solution above. For the polyadic and name-matching languages this holds by Proposition 4.1 of [19], otherwise for the intensional languages this holds by Theorem 6.4 of [14]. □

The following results complete the formalisation that coordination is orthogonal to synchronicity.

Theorem 5.6. *There exists no valid encoding from $\mathcal{L}_{S,M,D,NM,J}$ into $\mathcal{L}_{A,M,D,NM,J}$.*

Proof. The proof is by contradiction. Consider two processes $P = ((x)) \triangleright$ **if** $x = b$ **then** Ω (where Ω is a divergent process) and $Q = \langle a \rangle.Q'$. Since $P \mid Q \longmapsto$ by validity of the encoding $[\![P \mid Q]\!] \longmapsto$ and this must be between some $R_1 = \langle m \rangle$ for some m and R_2. Observe that $R_1 \mid R_2$ cannot be a reduct of either $[\![P]\!]$ or $[\![Q]\!]$ since then either P or Q would reduce and this contradicts Proposition 3.1.

If R_1 arises from $[\![P]\!]$ then it can be shown that $[\![P]\!]$ must also include a top level join since otherwise there would be no join in $[\![P]\!]$ that can bind some name to x and name invariance or divergence reflection would be shown to fail (i.e. $P \mid Q \longmapsto$ **if** $a = b$ **then** $\Omega \mid Q'$ and $\{b/a\}$**if** $a = b$ **then** $\Omega \mid Q' \longmapsto^{\omega}$ while $C^{\mathcal{N}}_{|}([\![P]\!], [\![Q]\!]) \Longmapsto$ does no inputs on any part of $[\![P]\!]$ and so must always or never diverge regardless of interaction with $[\![Q]\!]$). Thus $[\![P]\!]$ must include a top level join and further it must include an input pattern $(\ulcorner n \urcorner)$ for some $n \neq m$ since otherwise if the join was only $((z_1) \mid \ldots \mid (z_i)) \triangleright R'$ for some $\widetilde{z}$ and R' then $[\![P \mid \ldots \mid P]\!]$ for i instances of P would reduce while $P \mid \ldots \mid P$ does not contradicting Proposition 3.1. It follows that $[\![Q]\!]$ must include $(\ulcorner m \urcorner)$ as part of some join under which there must be an output that is able to send at least one name to $[\![P]\!]$ via an output $\langle d \rangle$ for some d (this could be any number of names, but assume 1 here for simplicity). Now consider the name d. If $d = m$ then $[\![P]\!] \longmapsto$ and this contradicts validity of the encoding since $P \not\longmapsto$. If $d = n$ then n is not bound in $[\![P]\!]$ and so it can be shown that either: this fails name invariance or divergence reflection (again by $P \mid Q \longmapsto$ **if** $a = b$ **then** $\Omega \mid Q'$ and $\{b/a\}$**if** $a = b$ **then** $\Omega \mid Q' \longmapsto^{\omega}$); or there must be a further input in $[\![P]\!]$ that is binding as in the next case. If $d \neq m$ and $d \neq n$ then it can be shown that $[\![P \mid Q \mid P]\!]$ can reduce such that the input under consideration interacts with the $\langle m \rangle$ from the other $[\![P]\!]$ and this ends up contradicting operational correspondence.

If R_1 arises from $[\![Q]\!]$ then it can be shown that $[\![Q]\!]$ must also include a top level join since otherwise when $Q' = \Omega$ then $[\![Q]\!]$ would always diverge or never diverge regardless of interaction with $[\![P]\!]$ and this contradicts divergence reflection. Thus $[\![Q]\!]$ must include a top level join and further it must include an input pattern $(\ulcorner n \urcorner)$ for some $n \neq m$ since otherwise if the join was only $((z_1) \mid \ldots \mid (z_i)) \triangleright R'$ for some $\widetilde{z}$ and R' then $[\![Q \mid \ldots \mid Q]\!]$ for i instances of Q would reduce while $Q \mid \ldots \mid Q$ does not contradicting Proposition 3.1. Now consider when $Q' =$ **if** $a = b$ **then** $\surd$ and the substitution $\sigma = \{b/a\}$. Clearly $P \mid \sigma Q \mid Q \longmapsto S$ where either: $S \longmapsto^{\omega}$ and $S \Downarrow$; or $S \not\longmapsto^{\omega}$ and $S \not\Downarrow$. However it can be shown that the top level join in $[\![Q]\!]$ is not able to discriminate and thus that there exist two possible reductions $[\![P \mid \sigma Q \mid Q]\!] \longmapsto R'$ to an R' where either: $R' \longmapsto^{\omega}$ and $R' \not\Downarrow$; or $R' \not\longmapsto^{\omega}$ and $R \Downarrow$; both of which contradict divergence reflection and success sensitiveness. □

Theorem 5.7. *There exists no valid encoding from $\mathcal{L}_{S,\beta,D,NO,J}$ into $\mathcal{L}_{A,\beta,D,NO,J}$.*

Proof. This is proved in the same manner as Theorem 5.6. □

That joining does not allow for an encoding of synchronous communication alone is not surprising, since there is no control in the input of which outputs are interacted with (without some other control such as channel names or pattern-matching). Thus, being able to consume more outputs in a single interaction does not capture synchronous behaviours.

6 Joining and Arity

This section considers the relation between joining and arity. It turns out that these are orthogonal. Although there appear to be some similarities in that both have a base case (monadic or binary), and an unbounded case (polyadic or joining, respectively), these cannot be used to encode one-another. This is captured by the following result.

Theorem 6.1. *There exists no valid encoding from $\mathcal{L}_{A,P,D,NO,B}$ into $\mathcal{L}_{A,M,D,NO,J}$.*

Proof. The proof is by contradiction, assume there exists a valid encoding $[\![\cdot]\!]$. Consider the $\mathcal{L}_{A,P,D,NO,B}$ processes $P = \langle a, b\rangle$ and $Q = (x,y).\surd$. Clearly it holds that $P \mid Q \longmapsto \surd$ and so $[\![P \mid Q]\!] \longmapsto$ and $[\![P \mid Q]\!] \Downarrow$ by validity of the encoding. Now consider the reduction $[\![P \mid Q]\!] \longmapsto$.

The reduction must be of the form $\langle m_1\rangle \mid \ldots \mid \langle m_i\rangle \mid ((z_1) \mid \ldots \mid (z_i)) \triangleright T'$ for some $\widetilde{m}$ and $\widetilde{z}$ and i and T'. Now consider the process whose encoding produces $((z_1) \mid \ldots \mid (z_i)) \triangleright T'$, assume Q although the results do not rely on this assumption. If any $\langle m_j\rangle$ are also from the encoding of Q then it follows that the encoding of i instances of Q in parallel will reduce, i.e. $[\![Q \mid \ldots \mid Q]\!] \longmapsto$, while $Q \mid \ldots \mid Q \not\longmapsto$. Now consider two fresh processes S and T such that $S \mid T \longmapsto$ with some arity that is not 2 and $S \not\longmapsto$ and $T \not\longmapsto$. It follows that $[\![S \mid T]\!] \longmapsto$ (and $[\![S]\!] \not\longmapsto$ and $[\![T]\!] \not\longmapsto$) and $[\![S \mid T]\!]$ must include at least one $\langle n\rangle$ to do so. This $\langle n\rangle$ must arise from either $[\![S]\!]$ or $[\![T]\!]$, and conclude by showing that the encoding of i instances of either S or T in parallel with Q reduces, while the un-encoded processes do not. □

Corollary 6.2. *If there exists no valid encoding from $\mathcal{L}_{\alpha,P,\gamma,\delta,B}$ into $\mathcal{L}_{\alpha,M,\gamma,\delta,B}$, then there exists no valid encoding from $\mathcal{L}_{\alpha,P,\gamma,\delta,-}$ into $\mathcal{L}_{\alpha,M,\gamma,\delta,J}$.*

Proof. The technique in Theorem 6.1 applies to all dataspace-based no-matching languages. Dataspace-based name-matching languages build upon Theorem 6.1 with $Q = (x,y).$**if** $a = x$ **then** $\surd$ to then ensure that binding occurs and not only name-matching, the proof is concluded via contradiction of name invariance and success sensitiveness as in Theorem 5.6. For the channel-based communication it is easier to refer to Theorem 7.1 to illustrate that this is not possible than to extend the proof above. □

Thus joining does not allow for encoding polyadicity in a monadic language unless it could already be encoded by some other means. In the other direction, the inability to encode joining into a binary language is already ensured by Corollary 4.2.

7 Joining and Communication Medium

This section considers the relation between joining and communication medium. Again joining turns out to be orthogonal to communication medium and neither can encode the other. The key to this is captured in the following result.

Theorem 7.1. *There exists no valid encoding from $\mathcal{L}_{A,M,C,NO,B}$ into $\mathcal{L}_{A,M,D,NO,J}$.*

Proof. The proof is by contradiction, assume there exists a valid encoding $[\![\cdot]\!]$. Consider the $\mathcal{L}_{A,M,C,NO,B}$ processes $P = \overline{a}\langle b\rangle$ and $Q = a(x).\surd$. Clearly it holds that $P \mid Q \longmapsto \surd$ and so $[\![P \mid Q]\!] \longmapsto$ and $[\![P \mid Q]\!] \Downarrow$ by validity of the encoding. Now consider the reduction $[\![P \mid Q]\!] \longmapsto$.

The reduction must be of the form $\langle m_1\rangle \mid \ldots \mid \langle m_i\rangle \mid ((z_1) \mid \ldots \mid (z_i)) \triangleright T'$ for some $\widetilde{m}$ and $\widetilde{z}$ and i and T'. Now consider the process whose encoding produces $((z_1) \mid \ldots \mid (z_i)) \triangleright T'$, assume Q although the results do not rely on this assumption. If any $\langle m_j\rangle$ are also from the encoding of Q then it follows that the encoding of i instances of Q in parallel will reduce, i.e. $[\![Q \mid \ldots \mid Q]\!] \longmapsto$, while $Q \mid \ldots \mid Q \not\longmapsto$. Now consider two fresh processes $S = \overline{c}\langle d\rangle$ and $T = c(z).\mathbf{0}$. Since $S \mid T \longmapsto$ it follows that $[\![S \mid T]\!] \longmapsto$ and must include at least one $\langle n\rangle$ to do so. This $\langle n\rangle$ must arise from either $[\![S]\!]$ or $[\![T]\!]$, and conclude by showing that the encoding of i instances of either S or T in parallel with Q reduces, while the un-encoded processes do not. □

Corollary 7.2. *If there exists no valid encoding from $\mathcal{L}_{\alpha,\beta,C,\delta,B}$ into $\mathcal{L}_{\alpha,\beta,D,\delta,B}$, then there exists no valid encoding from $\mathcal{L}_{\alpha,\beta,C,\delta,-}$ into $\mathcal{L}_{\alpha,\beta,D,\delta,J}$.*

Proof. The technique in Theorem 7.1 applies to all monadic languages (the addition of name-matching can be proved using the techniques as in Theorem 5.6). For the polyadic no-matching setting the result above holds by observing that the arity must remain fixed for an encoding, i.e. $[\![\,\overline{a}\langle b_1,\ldots,b_i\rangle\,]\!]$ is encoded to inputs/outputs all of some arity j. If the arity is not uniform then the encoding fails either operational correspondence (i.e. $[\![\,a(x).\mathbf{0} \mid \overline{a}\langle b_1,b_2\rangle\,]\!] \longmapsto$) or divergence reflection as in sub-case (2) of Theorem 8.1 except here with arity instead of number of names. □

Thus joining does not allow for encoding channels in a dataspace-based language unless it could already be encoded by some other means. In the other direction, the inability to encode joining into a binary language is already ensured by Corollary 4.2.

8 Joining and Pattern-Matching

This section considers the relations between joining and pattern-matching. The great expressive power of name matching [19] and intensionality [14] prove impossible to encode with joining. In the other direction, joining cannot be encoded by any form of pattern-matching.

The first result is to prove that intensionality cannot be encoded by joining. Recall that since intensionality alone can encode all other features aside from joining, it is sufficient to consider $\mathcal{L}_{A,M,D,I,B}$.

Theorem 8.1. *There exists no valid encoding from $\mathcal{L}_{A,M,D,I,B}$ into $\mathcal{L}_{-,-,-,\delta,J}$ where $\delta \neq I$.*

Proof. The proof is by contradiction and similar to Theorem 7.1 of [14]. Assume there exists a valid encoding $[\![\cdot]\!]$ from $\mathcal{L}_{A,M,D,I,B}$ into $\mathcal{L}_{\alpha,\beta,\gamma,\delta,J}$ for some α and β and γ and δ where $\delta \neq I$. Consider the encoding of the processes $S_0 = ((x)) \triangleright \langle m\rangle$ and $S_1 = \langle a\rangle$. Clearly $[\![S_0 \mid S_1]\!] \longmapsto$ since $S_0 \mid S_1 \longmapsto$. There exists a reduction $[\![S_0 \mid S_1]\!] \longmapsto$ that must be between a join and some outputs that have combined maximal arity k. (The combined arity is the sum of the arities of all the input-patterns of the join involved, e.g. $((a,b) \mid (c)) \triangleright \mathbf{0}$ has combined arity 3.)

Now define the following processes $S_2 \stackrel{\text{def}}{=} \langle a_1 \bullet \ldots \bullet a_{2k+1}\rangle$ and $S_3 \stackrel{\text{def}}{=} (\ulcorner a_1 \urcorner \bullet \ldots \bullet \ulcorner a_{2k+1} \urcorner).\langle m\rangle$ where S_2 outputs $2k+1$ distinct names in a single term, and S_3 matches all of these names in a single intensional pattern. Since $S_2 \mid S_0 \longmapsto$ it must be that $[\![S_2 \mid S_0]\!] \longmapsto$ for the encoding to be valid. Now consider the maximal combined arity of the reduction $[\![S_2 \mid S_0]\!] \longmapsto$.

- If the arity is k consider the reduction $[\![S_2 \mid S_3]\!] \longmapsto$ with the combined maximal arity j which must exist since $S_2 \mid S_3 \longmapsto$. Now consider the relationship of j and k.
  1. If $j = k$ then the upper bound on the number of names that are matched in the reduction is $2k$ (when each name is matched via a distinct channel). Since not all $2k+1$ 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 $i \in \{1,\ldots,2k+1\}$ that is not being matched in the interaction $[\![S_2 \mid S_3]\!] \longmapsto$. Now construct S_4 that differs from S_3 only by swapping one such name a_i with m: $S_4 \stackrel{\text{def}}{=} (\ulcorner a_1 \urcorner \bullet \ldots \ulcorner a_{i-1} \urcorner \bullet \ulcorner m \urcorner \bullet \ulcorner a_{i+1} \urcorner \ldots \ulcorner a_{k+2} \urcorner).\langle a_i\rangle$. Now consider the context $C_{|}^{N}([\![S_2]\!],[\![\cdot]\!]) = [\![S_2 \mid \cdot]\!]$ where $N = \{\widetilde{a} \cup m\}$. Clearly neither $C_{|}^{N}([\![S_2]\!],[\![\mathbf{0}]\!]) \longmapsto$ nor $C_{|}^{N}([\![S_2]\!],[\![S_4]\!]) \longmapsto$ as this would contradict Proposition 3.1. However, since S_3 and S_4 differ only by the position of one name whose tuple $\varphi_{[\![\,]\!]}(\cdot)$ does not appear in the reduction $[\![S_2 \mid S_3]\!] \longmapsto$, it follows

that the reason $C_{|}^{N}([\![S_2]\!],[\![S_4]\!]) \not\longmapsto$ must be due to a structural congruence difference between $C_{|}^{N}([\![S_2]\!],[\![S_3]\!])$ and $C_{|}^{N}([\![S_2]\!],[\![S_4]\!])$. Further, by compositionality of the encoding the difference can only be between $[\![S_3]\!]$ and $[\![S_4]\!]$. Since Proposition 3.1 ensures that $[\![S_3]\!] \not\longmapsto$ and $[\![S_4]\!] \not\longmapsto$, the only possibility is a structural difference between $[\![S_3]\!]$ and $[\![S_4]\!]$. Now exploiting $\sigma = \{m/a_i, a_i/m\}$ such that $\sigma S_4 = S_3$ yields contradiction.

2. If $j \neq k$ then obtain that $[\![S_2]\!]$ must be able to interact with both combined arity k and combined arity j. That is, $[\![S_2 \mid \cdot]\!] = C_{|}^{N}([\![S_2]\!],[\![\cdot]\!])$ where $N = \{\widetilde{a} \cup m\}$ and that $C_{|}^{N}([\![S_2]\!],[\![S_0]\!])$ reduces with combined arity k and $C_{|}^{N}([\![S_2]\!],[\![S_3]\!])$ reduces with combined arity j. Now it is straightforward, if tedious, to show that since $S_0 \mid S_3 \not\longmapsto$ that $C_{|}^{N}([\![S_2]\!],[\![S_0 \mid S_3]\!])$ can perform the same initial reductions as either $C_{|}^{N}([\![S_2]\!],[\![S_0 \mid \mathbf{0}]\!])$ or $C_{|}^{N}([\![S_2]\!],[\![\mathbf{0} \mid S_3]\!])$ by exploiting operational correspondence and Proposition 3.1. Thus, it can be shown that $C_{|}^{N}([\![S_2]\!],[\![S_0 \mid S_3]\!])$ can perform both the k combined arity reduction of $[\![S_2 \mid S_0]\!] \longmapsto$ and the j combined arity reduction of $[\![S_2 \mid S_3]\!] \longmapsto$. Now by exploiting the structural congruence rules it follows that neither of these initial reductions can prevent the other occurring. Thus, $C_{|}^{N}([\![S_2]\!],[\![S_0 \mid S_3]\!])$ must be able to do both of these initial reductions in any order. Now consider the process R that has performed both of these initial reductions. By operational correspondence it must be that $R \not\Longmapsto \simeq [\![\langle m\rangle \mid \langle m\rangle]\!]$ since $S_2 \mid S_0 \mid S_3 \not\Longmapsto \langle m\rangle \mid \langle m\rangle$. Therefore, R must be able to roll-back the initial step with combined arity j; i.e reduce to a state that is equivalent to the reduction not occurring. (Or the initial step with arity k, but either one is sufficient as by operational correspondence $R \Longmapsto \simeq [\![\langle m\rangle \mid S_3]\!]$.) Now consider how many names are being matched in the initial reduction with combined arity j. If $j < k$ the technique of differing on one name used in the case of $j = k$ can be used to show that this would introduce divergence on the potential roll-back and thus contradict a valid encoding. Therefore it must be that $j > k$. Finally, by exploiting name invariance and substitutions like $\{(b_1 \bullet \ldots \bullet b_j)/a_1\}$ applied to S_2 and S_3 it follows that either $j > k + j$ or both S_2 and S_3 must have infinitely many initial reductions which yields divergence.

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

Corollary 8.2. *If there exists no valid encoding from $\mathcal{L}_{\alpha,\beta,\gamma,I,B}$ into $\mathcal{L}_{\alpha,\beta,\gamma,\delta,B}$, then there exists no valid encoding from $\mathcal{L}_{\alpha,\beta,\gamma,I,-}$ into $\mathcal{L}_{\alpha,\beta,\gamma,\delta,J}$.*

It follows that joining cannot represent intensionality in a language that does not have intensionality already (including name-matching or no-matching languages).

The next result shows that name matching is insufficient to encode joining.

Theorem 8.3. *There exists no valid encoding from $\mathcal{L}_{A,M,D,NM,B}$ into $\mathcal{L}_{\alpha,\beta,\gamma,NO,J}$.*

Proof. The proof is by contradiction, assume there exists a valid encoding $[\![\cdot]\!]$. Consider the $\mathcal{L}_{A,M,D,NM,B}$ processes $P = \langle a\rangle$ and $Q = (\ulcorner a \urcorner).(\langle b\rangle \mid \surd)$. Clearly it holds that $P \mid Q \longmapsto$ and $P \mid Q \Downarrow$ and so $[\![P \mid Q]\!] \longmapsto$ and $[\![P \mid Q]\!] \Downarrow$ by validity of the encoding. Now consider γ.

- If $\gamma = D$ then consider the substitution $\sigma = \{a/b, b/a\}$, it is clear that $P \mid \sigma Q \not\longmapsto$ and so $[\![P \mid \sigma Q]\!] \not\longmapsto$, however the only possibility that this holds is when $[\![\sigma Q]\!]$ is blocked from interacting. It is then straightforward if tedious to show that any such blocking of reduction would either imply $[\![\sigma(P \mid Q)]\!] \not\longmapsto$ or $\sigma(P \mid Q) \not\longmapsto$ and thus contradict the validity of the encoding.
- Otherwise it must be that $\gamma = C$. Now consider the reduction $[\![P \mid Q]\!] \longmapsto$ that must be of the form $\overline{c_1}\langle\widetilde{m_1}\rangle \mid \ldots \mid \overline{c_i}\langle\widetilde{m_i}\rangle \mid (c_1(\widetilde{z_1}) \mid \ldots \mid c_i(\widetilde{z_i})) \triangleright T_1$ for some $\widetilde{c}$ and $\widetilde{m}$ and $\widetilde{z}$ and i and T_1. Again consider

the substitution $\sigma = \{a/b, b/a\}$, it is clear that $\sigma P \mid Q \not\longmapsto$ and so $[\![\sigma P \mid Q]\!] \not\longmapsto$. The only way this can occur without contradicting the validity of the encoding (as in the previous case) is when there is at least one c_k in the domain of some σ' where $\sigma'(c_k) \neq c_k$ and $[\![\sigma P]\!] \simeq \sigma'[\![P]\!]$ by definition of the encoding. Now consider the process $S = (x).S'$, clearly $P \mid S \longmapsto$ and so $[\![P \mid S]\!] \longmapsto$ as well. The reduction $[\![P \mid S]\!] \longmapsto$ must be from the form $\overline{d_1}\langle\widetilde{n_1}\rangle \mid \ldots \mid \overline{d_j}\langle\widetilde{n_j}\rangle \mid (d_1(\widetilde{w_1}) \mid \ldots \mid d_j(\widetilde{w_j})) \triangleright T_2$ for some $\widetilde{d}$ and $\widetilde{n}$ and $\widetilde{w}$ and j and T_2. Now if $i = j$ it follows that for each $k \in \{1 \ldots i\}$ then $c_k = d_k$. However, this contradicts the validity of the encoding since there is some c_k in the domain of σ' such that $\sigma'(c_k) \neq c_k$ and $\sigma P \mid S \longmapsto$ while $[\![\sigma P \mid S]\!] \not\longmapsto$. Otherwise it must be that $i > j$ (otherwise if $i < j$ then $[\![P \mid S]\!] \not\longmapsto$) and that $c_k \in \{c_{j+1}, \ldots, c_i\}$. Now consider when $S' = \textbf{if } x = a \textbf{ then } \Omega$, clearly $P \mid S \longmapsto \equiv \Omega$ and $\sigma P \mid S \longmapsto \equiv \mathbf{0}$ and so $[\![P \mid S]\!]$ diverges and $[\![\sigma P \mid S]\!] \Longmapsto \simeq \mathbf{0}$. Now it can be shown that $P \mid \sigma P \mid S \mid Q \longmapsto\longmapsto \equiv \surd$ while $[\![P \mid \sigma P \mid S \mid Q]\!] \Downarrow$ and diverges since $[\![\sigma P]\!]$ can satisfy the first j input patterns of $[\![Q]\!]$ and $[\![\sigma P]\!]$ the remaining $i - j$, leaving the first j input patterns of $[\![P]\!]$ to interact with $[\![S]\!]$ and yield divergence. The only other possibility is that $[\![P \mid \sigma P \mid S \mid Q]\!] \not\Downarrow$. However, this requires that T_1 check some binding name in $\widetilde{z}$ for equality with a before yielding success (i.e. **if** $z_1 = a$ **then** $\surd$). This can in turn be shown to contradict the validity of the encoding by adding another instance of P. □

Corollary 8.4. *If there exists no valid encoding from $\mathcal{L}_{\alpha,\beta,\gamma,NM,B}$ into $\mathcal{L}_{\alpha,\beta,\gamma,\delta,B}$, then there exists no valid encoding from $\mathcal{L}_{\alpha,\beta,\gamma,NM,-}$ into $\mathcal{L}_{\alpha,\beta,\gamma,\delta,J}$.*

Thus joining does not allow for encoding name-matching into a no-matching language unless it could already be encoded by some other means. In the other direction, the inability to encode joining into a binary language is already ensured by Corollary 4.2.

9 Conclusions and Future Work

Languages with non-binary coordination have been considered before, although less often than binary languages. It turns out that increases in coordination degree correspond to increases in expressive power. For example, an intensional binary language cannot be encoded by a non-intensional joining language. However, encodings from lower coordination degree languages into higher coordination degree languages are still dependent upon other features.

This formalises that languages like the Join Calculus, general rendezvous calculus, and m-calculus cannot be validly encoded into binary languages, regardless of other features. Although there exist encodings from (for example) Join Calculus into π-calculus [10] these do not meet the criteria for a *valid encoding* used here. The general approach used in such encodings is to encode joins by $[\![(m(x) \mid n(y)) \triangleright P]\!] = m(x).n(y).[\![P]\!]$, however this can easily fail operational correspondence, divergence reflection, or success sensitivity. For example $(c_1(w) \mid c_2(x)) \triangleright \surd \mid (c_2(y) \mid c_1(z)) \triangleright \Omega \mid \overline{c_1}\langle a\rangle \mid \overline{c_2}\langle b\rangle$ will either report success or diverge, but its encoding can deadlock. Even ordering the channel names to prevent this can be shown to fail under substitutions.

Future work along this line can consider coordination not merely to be binary or joining. Indeed, a *splitting* language could be one where several output terms can be combined into a split $(\overline{m}\langle a\rangle \mid \overline{n}\langle b\rangle) \triangleright P$ while inputs remain of the form $m(x).Q$. Further, languages could support both joining and splitting primitives for full coordination.

Acknowledgments. We would like to thank the reviewers for their constructive and helpful criticism.

References

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

[2] Laura Bocchi & Lucian Wischik (2004): *A Process Calculus of Atomic Commit*. *Electronic Notes in Theoretical Computer Science* 105(0), pp. 119 – 132, doi:10.1016/j.entcs.2004.05.003. Proceedings of the First International Workshop on Web Services and Formal Methods (WSFM 2004).

[3] M. Boreale, C. Fournet & C. Laneve (1998): *Bisimulations in the Join-Calculus*. In: *Programming Concepts and Methods PROCOMET 98*, IFIP The International Federation for Information Processing, Springer US, pp. 68–86, doi:10.1007/978-0-387-35358-6_9.

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

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

[6] Marco Carbone & Sergio Maffeis (2003): *On the Expressive Power of Polyadic Synchronisation in π-calculus*. *Nordic Journal of Computing* 10(2), pp. 70–98, doi:10.1.1.15.577.

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

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

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

[10] Cedric Fournet & Georges Gonthier: *The reflexive CHAM and the join-calculus*. In: *Proceedings of the 23rd ACM Symposium on Principles of Programming Languages*, ACM Press, pp. 372–385, doi:10.1.1.495.7510.

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

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

[13] Thomas Given-Wilson (2014): *An Intensional Concurrent Faithful Encoding of Turing Machines*. In Ivan Lanese, Alberto Lluch-Lafuente, Ana Sokolova & Hugo Torres Vieira, editors: *Proceedings 7th Interaction and Concurrency Experience, ICE 2014, Berlin, Germany, 6th June 2014.*, *EPTCS* 166, pp. 21–37, doi:10.4204/EPTCS.166.4.

[14] Thomas Given-Wilson (2014): *On the Expressiveness of Intensional Communication*. In: *Combined 21th International Workshop on Expressiveness in Concurrency and 11th Workshop on Structural Operational Semantics*, Rome, Italie, doi:10.4204/EPTCS.160.4.

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

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

[17] Thomas Given-Wilson, Daniele Gorla & Barry Jay (2014): *A Concurrent Pattern Calculus*. *Logical Methods in Computer Science* 10(3), doi:10.2168/LMCS-10(3:10)2014.

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

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

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

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

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

[23] Kohei Honda & Mario Tokoro (1991): *An object calculus for asynchronous communication*. In: *ECOOP'91 European Conference on Object-Oriented Programming*, Springer, pp. 133–147, doi:10.1.1.53.4500.

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

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

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

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

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

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

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

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

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

[33] Alan Schmitt & Jean-Bernard Stefani (2003): *The m-calculus: a higher-order distributed process calculus*. In: *Conference Record of POPL 2003: The 30th SIGPLAN-SIGACT Symposium on Principles of Programming Languages, New Orleans, Louisisana, USA, January 15-17, 2003*, pp. 50–61, doi:10.1145/640128.604136.

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