On the Expressiveness of Joining and Splitting

Thomas Given-Wilson, Axel Legay

On the Expressiveness of Joining and Splitting

Thomas Given-Wilson and Axel Legay

Inria

Abstract. An ongoing theme of the work of Bernhard Steffen has been the bringing together of different components in a coordinated manner and with a unified language. This paper explores this approach applied to process calculi that account for *coordination* of different kinds of workflows. Coordination here extends binary interaction to also account for *joining* of multiple outputs into a single input, and *splitting* from a single output to multiple inputs. The results here formalise which process calculi can and cannot be encoded into one another, and thus which language has the required expressiveness for given workflow properties. The combination of with other features of interaction allows for the representation of many systems and workflows in an appropriate calculus.

1 Introduction

An ongoing theme of the work of Bernhard Steffen has been the bringing together of different components in a functional and coordinated manner [55, 42, 14, 12]. This ranges from early work on unifying models [8, 55] to bringing together many components [37], to programming environments that combine components and workflows [42, 14]. This theme has as its core finding common languages to express desirable behaviours, and approaches to unify these behaviours and workflows in a single language [8, 55, 37, 42, 14, 44, 12].

This paper explores languages based on process calculi in the style of π-calculus that focus on coordinating workflows and higher-order process modelling [45, 44]. The expressiveness of process calculi based upon their choice of communication primitives has been explored before [49, 9, 16, 28, 21, 23, 58]. In [28] and [23] this is detailed by examining combinations of four features: synchronism, arity, communication medium, and pattern-matching, and formalising their relations via valid encodings [30]. These four features allow the representation of many languages and many kinds of constraints and typing on interaction [21, 22]. However, recent work [27] has extended *binary* interaction to *joining* as a form of coordination that allows a single process to receive input from many workflow producers in a single interaction. This work generalises to also account for *splitting* where a single process may produce outputs for many workflows in a single interaction (the dual of joining).

Along with the theme or Bernhard Steffen's works, this paper explores the common expressiveness of these languages, and shows which forms of interaction and workflow management can be represented in a single common process calculus. Similarly, the inability to express certain features of interaction is formalised,

demonstrating which languages are required for which kinds of interaction and workflow behaviour.

The structure of the paper is as follows. Section 2 introduces the calculi considered here. Section 3 reviews the criteria used for comparing calculi. Section 4 considers encoding synchronism with coordination. Section 5 explores encoding arity via coordination. Section 6 presents results for encoding communication medium into coordination. Section 7 formalises that coordination cannot encode pattern-matching. Section 8 presents that coordination cannot be encoded by other features (i.e. synchronism, channel-names, pattern-matching). Section 9 considers relations between different forms of coordination. Section 10 concludes and discusses future work.

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 and adapts them when necessary to cope with different features. With the exception of the splitting this repeats many prior definitions from [27], although there are minor syntactic changes for clarity in this work.

Assume a countable set of names $\mathcal{N}$ ranged over by $a, b, c, \ldots$. *Name-match patterns* (denoted m, n), *input patterns (denoted p, q), and* terms *(denoted s, t) are defined according to the following grammar:*

$$\begin{array}{rcll} m, n & ::= & x & \textit{binding name} \\ & | & \ulcorner a \urcorner & \textit{name-match} \\ p, q & ::= & m & \textit{name-match pattern} \\ & | & p \bullet q & \textit{(input pattern) compound} \\ s, t & ::= & a & \textit{name} \\ & | & s \bullet t & \textit{(term) compound.} \end{array}$$

The name-match patterns are used for input, with binding names doing binding, and name-matches testing equality. The input patterns generalise the name-match patterns to also include compounds that support structure. The terms are used for output, with names being the base and compounds adding structure. The free names and binding names for input patterns and terms are as expected, taking the union of sub-patterns for compounds. Note that an input pattern is linear if and only if all binding names within the pattern are pairwise distinct. The rest of this paper will only consider linear input patterns.

This paper considers the possible combinations of five features for communication in a language denoted $\mathcal{L}_{\alpha,\beta,\gamma,\delta,\epsilon}$ where:

- $\alpha =$ A for asynchronous communication (output only prefoxes the empty process), and S for synchronous communication (output may prefix any process).
- $\beta =$ M for monadic data (input or output only a single term), and P for polyadic data (input or output unbounded sequences of terms).

$\mathcal{L}_{A,-,-,-,-}$: $InProc ::= [\mathcal{I}] \triangleright P$ $OutProc ::= [\mathcal{O}]\triangleleft$
$\mathcal{L}_{S,-,-,-,-}$: $InProc ::= [\mathcal{I}] \triangleright P$ $OutProc ::= [\mathcal{O}] \triangleleft P$
$\mathcal{L}_{-,-,-,-,B}$: $\mathcal{I} ::= IN$ $\mathcal{O} ::= OUT$
$\mathcal{L}_{-,-,-,-,J}$: $\mathcal{I} ::= IN \mid \mathcal{I} \mid \mathcal{I}$ $\mathcal{O} ::= OUT$
$\mathcal{L}_{-,-,-,-,L}$: $\mathcal{I} ::= IN$ $\mathcal{O} ::= OUT \mid \mathcal{O} \mid \mathcal{O}$

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

Fig. 1. Syntax of Languages.

- $\gamma =$ D for dataspace-based (interaction without named channels), and C for channel-based communications (interaction uses channel-names).
- $\delta =$ NO for no-matching (inputs can only bind), NM for name-matching (inputs can test equality of names), and I for intensionality (inputs can test name equality and also term structure).
- $\epsilon =$ B for binary (one input and one output interact), J for joining (one input may interact with many outputs), and L for splitting communication (one output may interact with many inputs).

For simplicity a dash $-$ will be used when the instantiation of a feature is unimportant. The (parametric) syntax for the languages is:

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

Most of the process forms as as usual: $\mathbf{0}$ denotes the null process; restriction $(\nu a)P$ restricts the visibility of a to P; parallel composition $P|Q$ allows independent evolution of P and Q; and $*P$ represents replication of the process P. The **if** $s = t$ **then** P **else** Q represents conditional equivalence with **if** $s = t$ **then** P used when Q is $\mathbf{0}$ (like the name match of π-calculus, **if** $s = t$ **then** P **else** Q blocks either P when $s \neq t$ or Q when $s = t$). The $\surd$ is used to represent a success process or state, in other works a specific barb or name has been used, however here by isolating $\surd$ as a specific process it is easier to reason about encodings (as also in) [30, 21]. Finally, different languages are obtained by replacing the output $OutProc$ and input $InProc$ with the various definitions 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, input patterns, etc. (also denote with $|\cdot|$ the size of a set, multiset, or sequence).

As usual $(\nu x)P$ and $[a(\ldots, x, \ldots)]\triangleright P$ and $[(x\bullet\ldots)]\triangleright P$ and $[\ldots \mid a(x) \mid \ldots]\triangleright P$ bind x in P. Observe that in $[a(\ldots, \ulcorner b\urcorner, \ldots)]\triangleright P$ and $[(\ldots\bullet\ulcorner b\urcorner)]\triangleright P$ 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. Further, an input is linear if all binding names in that input occur exactly once (note that this is already assumed

within an input pattern, here this is generalised to whole inputs). This paper shall only consider linear inputs. Finally, the structural congruence relation $\equiv$ is the smallest congruence such that the following hold:

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

Observe that $\mathcal{L}_{A,M,C,NO,B}$, $\mathcal{L}_{A,P,C,NO,B}$, $\mathcal{L}_{S,M,C,NO,B}$, and $\mathcal{L}_{S,P,C,NO,B}$ use the communication paradigm of the asynchronous/synchronous monadic/polyadic π-calculus [40, 41, 38]. The language $\mathcal{L}_{A,P,D,NM,B}$ uses the communication paradigm of LINDA[20]; the languages $\mathcal{L}_{A,M,D,NO,B}$ and $\mathcal{L}_{A,P,D,NO,B}$ the communication paradigm of the monadic/polyadic Mobile Ambients [11]; and $\mathcal{L}_{A,P,C,NM,B}$ that of μKLAIM [15] or semantic-π [13].

Due to the large number of intensional languages of the form $\mathcal{L}_{\alpha,\beta,\gamma,I,\epsilon}$ defined here, many do not match the communication paradigm of well-known calculi. However, the language $\mathcal{L}_{S,M,D,I,B}$ is the asymmetric concurrent pattern calculus of [22] and calculi with other communication paradigms that match some of those here have been mentioned in [21], as variations of Concurrent Pattern Calculus [25, 21] (with their behavioural theory as a specialisation of [24]). Similarly, the language $\mathcal{L}_{S,M,C,I,B}$ uses the communication paradigm of Spi calculus [1, 31] and Psi calculi (albeit with channel equivalence represented by equality and without the possibility of repeated binding names in patterns) [2]. There are also similarities between the communication paradigm of $\mathcal{L}_{S,M,C,I,B}$ and the polyadic synchronization π-calculus [10], although the intensionality in polyadic synchronization π-calculus is limited to the channel, i.e. inputs and outputs of the form $s(x).P$ and $\overline{s}\langle a\rangle.P$, respectively.

The joining languages have several similarities to existing calculi. The language $\mathcal{L}_{A,P,C,NO,J}$ uses a communication paradigm very close to an asynchronous π-calculus with joint input [43]. $\mathcal{L}_{S,P,C,NO,J}$ uses the communication paradigm of the general rendezvous calculus [4], and m-calculus [54], although the latter has higher order constructs and other aspects that are not captured within the features here. The language $\mathcal{L}_{S,M,C,NO,J}$ has a similar communication paradigm to the Quality Calculus [47, 48], however the Quality Calculus has further conditions upon the inputs that cannot be represented by $\mathcal{L}_{S,M,C,NO,J}$. Despite these similarities to many languages related to Join Calculus, the Join Calculus itself is difficult to capture in the π-calculus based framework here. This is due to Join Calculus combining restriction, replication, and input into a single primitive [19]. There are no exact connections for the splitting languages. Although one might consider some similarity with broadcast calculus [53] and $b\pi$-calculus [18] that both allow a single output to communicate with multiple inputs, even the closest splitting language $\mathcal{L}_{S,M,C,NO,L}$ has a fundamentally different communication paradigm. The difference is that in the broadcast calculi the number of inputs required to interact with a broadcast is not fixed, while for $\mathcal{L}_{S,M,C,NO,L}$ the number of inputs is fixed.

Remark 1. The languages $\Lambda_{s,a,m,p}$ can be easily partially ordered; in particular $\mathcal{L}_{\alpha_1,\beta_1,\gamma_1,\delta_1,\epsilon_1}$ is a lesser language than $\mathcal{L}_{\alpha_2,\beta_2,\gamma_2,\delta_2,\epsilon_2}$ 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:

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

This can be understood as the lesser language variation being a special case of the more general language. Asynchronous communication is synchronous communication with all output followed by $\mathbf{0}$. 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, binary communication is: joining communication with all joining inputs having only a single input pattern, and splitting communication with all splitting outputs having only a single output term.

The operational semantics of the languages is given here via reductions as in [38, 33, 23]. An alternative style is via a *labelled transition system* (LTS) such as [28]. Here the reduction based style is to simplify having to define here the (potentially complex) labels that occur when both intensionality, and joining/splitting is in play. The LTS style can be used for intensional languages [2, 21, 24]. Also, for the non-binary languages the techniques used in [5] can be used directly for the no-matching joining languages, and with the techniques of [5, 24] to extend intensionality and other coordination forms.

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

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

Where substitution is as usual on names, and on the understanding that $\ulcorner(s \bullet t)\urcorner \stackrel{\text{def}}{=} \ulcorner s\urcorner \bullet \ulcorner t\urcorner$.

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 [56, 3].

The matching of terms $\widetilde{t}$ with patterns $\widetilde{p}$ is handled in two parts. First, the *match* rule $\{t /\!\!/ p\}$ of a term t with a pattern p to create a substitution σ:

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

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 union of σ_1 and σ_2. (Observe that, since patterns are linear, 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.

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

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 by $\{s /\!\!/ p\}$ 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 to happen between substitutions with disjoint domain by linearity of inputs.) Otherwise the poly-match rule is undefined, for example when a single match fails, or the sequences are of different arity.

There are now three base reduction rules, one for each of binary, joining, and splitting languages. The binary reduction rule is:

$$[\overline{s}\langle\widetilde{t}\rangle] \lhd P \;\mid\; [s(\widetilde{p})] \rhd Q \quad \longmapsto \quad P \;\mid\; \sigma Q \qquad \textsc{Match}(\widetilde{t}; \widetilde{p}) = \sigma$$

that states that the split $[\overline{s}\langle\widetilde{t}\rangle] \lhd P$ interacts with the join $[s(\widetilde{p})] \rhd Q$ to yield $P \mid \sigma Q$ when the channel name s is the same and the match $\textsc{Match}(\widetilde{t}; \widetilde{p})$ is defined and yields σ. Note that P is omitted in the asynchronous languages and the channel names s are omitted in the dataspace-based languages.

The joining reduction rule is:

$$\begin{aligned}&[\overline{s_1}\langle\widetilde{t_1}\rangle] \lhd P_1 \;\mid\; \ldots \;\mid\; [\overline{s_i}\langle\widetilde{t_i}\rangle] \lhd P_i \;\mid\; [s_1(\widetilde{p_1}) \mid \ldots \mid s_i(\widetilde{p_i})] \rhd Q \\ &\qquad \longmapsto \quad P_1 \;\mid\; \ldots \;\mid\; P_i \;\mid\; \sigma Q \qquad \textsc{Match}(\widetilde{t_1}, \ldots, \widetilde{t_i}; \widetilde{p_1}, \ldots, \widetilde{p_i}) = \sigma\end{aligned}$$

that states that i splits $[\overline{s_i}\langle\widetilde{t_i}\rangle] \lhd P_i$ can interact with a single join when all of the outputs of the splits $\overline{s_i}\langle\widetilde{t_i}\rangle$ can be matched against the inputs of the join $\textsc{Match}(\widetilde{t_1}, \ldots, \widetilde{t_i}; \widetilde{p_1}, \ldots, \widetilde{p_i})$ to yield a substitution σ, and then reduce to the continuations of the splits $P_1 \mid \ldots \mid P_i$ in parallel with σQ.

The splitting reduction rule is the mirror of the joining rule:

$$\begin{aligned}&[\overline{s_1}\langle\widetilde{t_1}\rangle \mid \ldots \mid \overline{s_i}\langle\widetilde{t_i}\rangle] \lhd P \;\mid\; [s_1(\widetilde{p_1})] \rhd Q_1 \;\mid\; \ldots \;\mid\; [s_i(\widetilde{p_i})] \rhd Q_i \\ &\quad \longmapsto \quad P \;\mid\; \sigma_1 Q_1 \;\mid\; \ldots \;\mid\; \sigma_i Q_i \qquad \textsc{Match}(\widetilde{t_j}; \widetilde{p_j}) = \sigma_j \;\; j \in \{1, \ldots, i\}\end{aligned}$$

where all the outputs of a single split $[\overline{s_1}\langle\widetilde{t_1}\rangle \mid \ldots \mid \overline{s_i}\langle\widetilde{t_i}\rangle] \lhd P$ match $\textsc{Match}(\widetilde{t_j}; \widetilde{p_j})$ with separate joins $[s_j(\widetilde{p_j})] \rhd Q_i$ to yield σ_j and reduce to $P \mid \sigma_1 Q_1 \mid \ldots \mid \sigma_i Q_i$ for all $j \in \{1, \ldots, i\}$.

The general reduction relation $\longmapsto$ for all languages also includes:

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

$$\frac{s = t \qquad P \mid Q \longmapsto S}{P \mid \mathbf{if}\ s = t\ \mathbf{then}\ Q\ \mathbf{else}\ R \longmapsto S} \qquad \frac{s \neq t \qquad P \mid R \longmapsto S}{P \mid \mathbf{if}\ s = t\ \mathbf{then}\ Q\ \mathbf{else}\ R \longmapsto S}\ .$$

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

Lastly, for each language let $\simeq$ denote a reduction-sensitive behavioural equivalence for that language. A *reduction-sensitive* behavioural equivalence $\simeq$ is one where it holds that $P \simeq P'$ and $P' \longmapsto$ imply $P \longmapsto$ as in Definition 5.3 of [30] (observe that this rules out weak bisimulations for example). 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 [28]. For the intensional languages the results in [24] can be used. For the joining languages that reflect those of the literature the techniques used in [5] apply. For other combinations of joining, and splitting, as well as the addition of intensionality to non-binary languages, adaptations of [5, 24] should prove adequate.

3 Encodings

This section recalls the definition of valid encodings as well as some useful results (details in [30]) for formally relating process calculi.

The choice of valid encodings here is to align with prior works [28, 30, 23] and where possible reuse prior results. These valid encodings are those used, sometimes with mild adaptations, in [30, 29, 25, 46, 21, 26] and have also inspired similar works [34, 35, 57]. However, there are alternative approaches to encoding criteria or comparing expressive power [6, 17, 10, 50, 57]. Further arguments in favour of, or against, the valid encodings here can be found in [30, 29, 51, 57, 26].

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 $k > 0$. 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 $\varphi_{\llbracket\rrbracket}$.

To aid in the following definition and the results later in the paper, a process P is defined to be at *top-level* when P may be under any combination of restrictions, conditionals, or replications, but that none of these can prevent reduction or interaction by P. For example, P is top-level in $(\nu n)P$ and $*P$ and $\mathbf{if}\ s = s\ \mathbf{then}\ P\ \mathbf{else}\ Q$ and $\mathbf{if}\ s = t\ \mathbf{then}\ Q\ \mathbf{else}\ P$ where $s \neq t$ and $(\nu n)\mathbf{if}\ s = s\ \mathbf{then}\ *(P \mid Q)\ \mathbf{else}\ R$.

Now consider only encodings that satisfy the following properties. Let a *k-ary context* $\mathcal{C}(\cdot_1; \ldots; \cdot_k)$ be a process with k holes. Denote with $\longmapsto^\omega$ an infinite sequence of reductions. Let $P \Downarrow$ mean there exists P' such that $P \longmapsto\!\!\!\!\!\Longrightarrow P'$ and P'

has an instance of $\surd$ at top-level, that is the process P eventually exhibits the success process $\surd$. Moreover, let $\simeq$ denote the reference behavioural equivalence. 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., $\mathcal{L}_2$) and let the notation of a language $\mathcal{L}_i$ be subscripted by i, e.g. $\simeq_i$, $\longmapsto_i$, etc.

Definition 1 (Valid Encoding). *An encoding* $(\llbracket\cdot\rrbracket, \varphi_{\llbracket\rrbracket})$ *of* $\mathcal{L}_1$ *into* $\mathcal{L}_2$ *is* valid *if it satisfies the following five properties:*

1. Compositionality: *for every* k*-ary operator* op *of* $\mathcal{L}_1$ *and for every subset of names* N*, there exists a* k*-ary context* $\mathcal{C}^N_{\mathsf{op}}(\cdot_1;\ldots;\cdot_k)$ *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* $\llbracket \mathsf{op}(S_1,\ldots,S_k)\rrbracket = \mathcal{C}^N_{\mathsf{op}}(\llbracket S_1\rrbracket;\ldots;\llbracket S_k\rrbracket)$.
2. Name invariance: *for every* S *and substitution* σ*, it holds that* $\llbracket \sigma S\rrbracket = \sigma'\llbracket S\rrbracket$ *if* σ *is injective and* $\llbracket \sigma S\rrbracket \simeq_2 \sigma'\llbracket S\rrbracket$ *otherwise where* σ' *is such that* $\varphi_{\llbracket\rrbracket}(\sigma(a)) = \sigma'(\varphi_{\llbracket\rrbracket}(a))$ *for every name* $a \in N$.
3. Operational correspondence: *for all* $S \Longmapsto_1 S'$*, it holds that* $\llbracket S\rrbracket \Longmapsto_2 \simeq_2 \llbracket S'\rrbracket$*; and for all* $\llbracket S\rrbracket \Longmapsto_2 T$*, there exists* S' *such that* $S \Longmapsto_1 S'$ *and* $T \Longmapsto_2 \simeq_2 \llbracket S'\rrbracket$.
4. Divergence reflection: *for every* S *such that* $\llbracket S\rrbracket \longmapsto_2^\omega$*, it holds that* $S \longmapsto_1^\omega$.
5. Success sensitiveness: *for every* S*, it holds that* $S \Downarrow_1$ *if and only if* $\llbracket S\rrbracket \Downarrow_2$.

The existence of encodings $\llbracket\cdot\rrbracket_1$ from $\mathcal{L}_1$ into $\mathcal{L}_2$ and $\llbracket\cdot\rrbracket_2$ from $\mathcal{L}_2$ into $\mathcal{L}_3$ does not ensure that $\llbracket\llbracket\cdot\rrbracket_1\rrbracket_2$ is a valid encoding from $\mathcal{L}_1$ into $\mathcal{L}_3$ [29]. However, compositionality can be ensured by respecting the below definition of compositional valid encodings. All encodings considered in this paper satisfy this restriction, and therefore are compositional and may be used as such in later proofs.

Definition 2 (Compositional Valid Encodings). *An encoding* $(\llbracket\cdot\rrbracket, \varphi_{\llbracket\rrbracket})$ *of* $\mathcal{L}_1$ *into* $\mathcal{L}_2$ *is* compositional *if it satisfies the properties 1,2,4 and 5 from Definition 1 and the following properties:*

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

The following three results are here recalled from prior works as they are useful for later proofs.

Proposition 1 (Proposition 5.5 from [30]). *Let* $\llbracket\cdot\rrbracket$ *be a valid encoding; then,* $S \not\longmapsto$ *implies that* $\llbracket S\rrbracket \not\longmapsto$.

Proof. By contradiction, assume that $\llbracket S\rrbracket \longmapsto T$, for some $S \not\longmapsto$. By operational correspondence, there exists an S' such that $S \Longmapsto S'$ and $T \Longmapsto T' \simeq \llbracket S'\rrbracket$; but the only such S' is S itself. Since $\simeq$ is reduction-sensitive and since $\llbracket S'\rrbracket = \llbracket S\rrbracket \longmapsto$, then $T' \longmapsto T''$. Again by operational correspondence $T'' \Longmapsto T''' \simeq \llbracket S\rrbracket$, and so on; thus, $\llbracket S\rrbracket \longmapsto T \Longmapsto T' \Longmapsto T'' \Longmapsto T''' \longmapsto \ldots$, in contradiction with divergence reflection (since $S \not\longmapsto$ implies $S \not\longmapsto^\omega$).

The following two results (and a few later in the paper) exploit the notation $block(S)$ that denotes the process $(\nu n)(\nu m)\mathbf{if}\ n = m\ \mathbf{then}\ S$ where $n, m \notin \mathsf{fn}(S)$.

Proposition 2 (Proposition 5.6 from [30]). *Let $[\![\cdot]\!]$ be a valid encoding; then for every set of names N, it holds that $\mathcal{C}^N_{|}(\cdot_1, \cdot_2)$ 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 1 it must be that $[\![S']\!] \not\longmapsto$, since $S' \not\longmapsto$. By compositionality we must have $[\![S']\!] = \mathcal{C}^N_{|}([\![\surd]\!], [\![block(S)]\!])$. By success sensitiveness it must be that $[\![S']\!] \Downarrow$ since $S' \Downarrow$. All these facts entail that the top-level occurrence of $\surd$ in $[\![S']\!]$ is exhibited: either by the translating context and so $\mathcal{C}^N_{|}(\cdot, \cdot) \Downarrow$; or by $[\![\surd]\!]$, but this implies that $\mathcal{C}^N_{|}(\cdot, \cdot)$ has the first hole $\cdot$ at top-level. Indeed, it is not possible that $\surd$ is exhibited by $[\![block(S)]\!]$ since $block(S) \not\Downarrow$. However, the first case is not possible otherwise $[\![block(S) \mid block(S)]\!] \Downarrow$, whereas $block(S) \mid block(S) \not\Downarrow$. To show that the second hole in $\mathcal{C}^N_{|}(\cdot, \cdot)$ is at top-level it suffices to reason in the very same way using $S' = block(S) \mid \surd$.

Proposition 3 (Adapted from Proposition 5.7 from [30]). *Let $[\![\cdot]\!]$ be a valid encoding; if there exist two processes S_1 and S_2 such that $S_1 \mid S_2 \Downarrow$, with $S_i \not\Downarrow$ and $S_i \not\longmapsto$ for $i = 1,\ 2$, then $[\![S_1 \mid S_2]\!] \longmapsto$.*

Proof. By success sensitiveness $[\![S_1 \mid S_2]\!] \Downarrow$ and by Proposition 2 $\mathcal{C}^N_{|}([\![S_1]\!], [\![S_2]\!])$ 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 $[\![S_1 \mid S_2]\!] \longmapsto$. This can only happen by interaction between $[\![S_1]\!]$ and $[\![S_2]\!]$. If this was not the case, we would have $[\![S_1 \mid block(S_2)]\!] \longmapsto$ or $[\![block(S_1) \mid S_2]\!] \longmapsto$ or $[\![block(S_1) \mid block(S_2)]\!] \longmapsto$, in violation of Proposition 1: indeed $S_1 \mid block(S_2) \not\longmapsto$ because $S_1 \not\longmapsto$, $block(S_2) \not\longmapsto$ 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 general way to prove the lack of a valid encoding is as follows. By contradiction assuming there is a valid encoding $[\![\cdot]\!]$. Find a pair of processes P and Q that satisfy Proposition 3 such that $P \mid Q \longmapsto$ and $[\![P \mid Q]\!] \longmapsto$. From Q obtain some Q' such that $P \mid Q' \not\longmapsto$ and $[\![P \mid Q']\!] \longmapsto$. Conclude by showing this in contradiction with some properties of the encoding or Proposition 1.

The following result is a consequence of the choices of languages and encoding criteria, which corresponds to formalising Remark 1.

Theorem 1. *If a language $\mathcal{L}_1$ is a lesser language than $\mathcal{L}_2$ (by the $\leq$ relation of Remark 1) then there exists a (compositional) valid encoding $[\![\cdot]\!]$ from $\mathcal{L}_1$ into $\mathcal{L}_2$.*

Proof. The encoding $[\![\cdot]\!]$ is as described in Remark 1. The proof is then straightforward and ensured by definition of the rule for the base reduction. For a detailed example of the proof technique see Theorem 2.

4 Coordination and Synchronism

This section considers the relation between coordination and synchronism. It turns out that coordination is unable to encode synchronism unless it could otherwise be encoded by other features.

In general synchronous communication can be encoded into asynchronous communication when the target language includes: channel names; name-matching and polyadicity; or intensionality. Thus it is sufficient to consider the languages $\mathcal{L}_{A,M,D,NO,-}$ and $\mathcal{L}_{A,P,D,NO,-}$ and $\mathcal{L}_{A,M,D,NM,-}$ since the other asynchronous languages can encode their synchronous joining counterparts in the usual manner [32, 7]. This 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

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

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 1. *If $P \equiv Q$ then $[\![P]\!] \equiv [\![Q]\!]$. Conversely, if $[\![P]\!] \equiv Q$ then $Q \equiv [\![P']\!]$ for some $P' \equiv P$.*

Proof. The only non-trivial cases are the join and split as the others are translated homomorphically. The join and split are also straightforward as the only non-trivial parts are the possible renaming of new restricted names introduced in the translation.

Lemma 2. *Given a $\mathcal{L}_{S,M,C,NO,J}$ join P and split Q then $[\![P]\!] \mid [\![Q]\!] \longmapsto$ if and only if $P \mid Q \longmapsto$.*

Proof. Both parts can be proved by induction on the height of the proof tree for the judgements $[\![\, P \mid Q \,]\!] \longmapsto$ and $P \mid Q \longmapsto$. The base case is ensured by k applications of the poly-match rule when P is of the form $[n_1(x_1) \mid \ldots \mid n_k(x_k)] \rhd P'$. Note that Lemma 1 is used for structural congruence.

Lemma 3. *The translation $[\![\cdot]\!]$ from $\mathcal{L}_{S,M,C,NO,J}$ into $\mathcal{L}_{A,M,C,NO,J}$ preserves and reflects reductions. That is: if $P \longmapsto P'$ then there exists Q such that*

$[\![P]\!] \longmapsto^k Q$ *and* $Q = [\![P']\!]$*; and if* $[\![P]\!] \longmapsto Q$ *then there exists* Q' *such that* $Q \longmapsto^{k-1} Q'$ *and* $Q' = [\![P']\!]$ *for some* P' *such that* $P \longmapsto P'$*.*

Proof. Both parts can be proved by straightforward induction on the judgements $P \longmapsto P'$ and $[\![P]\!] \longmapsto Q$, respectively. In both cases, the base step is the most interesting and follows from Lemma 2, for the second case the step $Q \longmapsto Q'$ is ensured by the definition of the translation and match rule. The size of k (the number of target steps required to simulate one source step) in both cases is $2+i$ where i is the number of inputs of the join involved in $P \longmapsto P'$. The inductive cases where the last rule used is a structural one rely on Lemma 1.

Theorem 2. *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 3. Success sensitiveness can be proved as follows: $P \Downarrow$ means that there exists P' and $k \geq 0$ such that $P \longmapsto^k P'$ and P' has $\surd$ at top-level; by exploiting Lemma 3 k times and Lemma 1 obtain that $[\![P]\!] \longmapsto^j [\![P']\!]$ and P' has $\surd$ at top-level and where j can be determined from the instantiations of Lemma 3, i.e. that $[\![P]\!] \Downarrow$. The converse implication can be proved similarly.

Splitting can be adapted in a similar manner, e.g. consider the encoding from $\mathcal{L}_{S,M,C,NO,L}$ into $\mathcal{L}_{A,M,C,NO,L}$

$$\begin{aligned}
[\![\, [\overline{n_1}\langle a_1\rangle \mid \ldots \mid \overline{n_i}\langle a_i\rangle] \lhd P \,]\!] \stackrel{\text{def}}{=} (\nu z_1, \ldots, z_i)([\overline{n_1}\langle z_1\rangle \mid \ldots \mid \overline{n_i}\langle z_i\rangle] \lhd \mid \\
[z_1(x_1)] \rhd \ldots \rhd [z_i(x_i)] \rhd \\
([\overline{x_1}\langle a_1\rangle \mid \ldots \mid \overline{x_i}\langle a_i\rangle] \lhd \mid [\![P]\!])) \\
[\![\, [a(b)] \rhd Q \,]\!] \stackrel{\text{def}}{=} (\nu x)[a(z)] \rhd ([\overline{z}\langle x\rangle] \lhd \mid [x(b)] \rhd [\![Q]\!])
\end{aligned}$$

The use of fresh names z and x is as before. The splitting version is similar except the split synchronises with all the encoded inputs at once, sending fresh names z_j in parallel, then collects all the responses with fresh names x_j, and then splits sending all the original names a_i at once in the last step.

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

Theorem 3. *There is a valid encoding from* $\mathcal{L}_{S,M,C,NO,L}$ *into* $\mathcal{L}_{A,M,C,NO,L}$*.*

Proof. The same proof technique as Theorem 2 applies here.

Corollary 1. *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* $\mathcal{L}_{S,\beta,\gamma,\delta,\epsilon}$ *into* $\mathcal{L}_{A,\beta,\gamma,\delta,\epsilon}$*.*

Proof. Theorems 2 & 3 provide the foundation for all the channel-based results. For the other encodings where channels are not available in the target language, the target language can already encode channel-based communication and so the above results can still be used. For the polyadic and name-matching languages this holds by Proposition 4.1 of [28], otherwise for the intensional languages this holds by Theorem 6.4 of [23].

These results confirm that the ability to encode synchronous communication into asynchronous communication is not impacted by changes to coordination. Any encoding that holds from a binary synchronous language into a binary asynchronous language also holds when both languages are instead joining, or splitting. Thus no expressiveness is lost by changing from binary languages to other coordination forms, and existing results can easily be transferred.

The following results formalise that there exist no new encodings from a synchronous languages into an asynchronous languages as a result of shifting from both languages being binary, to both languages being joining or splitting. That is, if there exists no valid encoding from $\mathcal{L}_{S,\beta_1,\gamma_1,\delta_1,B}$ into $\mathcal{L}_{A,\beta_2,\gamma_2,\delta_2,B}$, then there exists no valid encoding from $\mathcal{L}_{S,\beta_1,\gamma_1,\delta_1,\epsilon}$ into $\mathcal{L}_{A,\beta_2,\gamma_2,\delta_2,\epsilon}$. The impossibility of encoding $\mathcal{L}_{S,M,D,NM,J}$ into $\mathcal{L}_{A,M,D,NM,J}$ is detailed as it illustrates the key proof technique. The other results are either simpler variations (i.e. without name-matching) or straightforward adaptations to consider splitting.

Theorem 4. *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 $P = [(x)] \triangleright \mathbf{if}\ x = b\ \mathbf{then}\ \surd$ and $Q = [\langle a \rangle] \triangleleft Q'$ where $a \neq b$ and $Q' \not\Downarrow$. Because $P \mid Q \longmapsto$ by validity of the encoding and Proposition 3 it follows that $[\![P \mid Q]\!] \longmapsto$ and this must be between some $R_1 = [\langle m \rangle] \triangleleft$ (for some m) and R_2. (This can be obtained by induction over the derivation tree for $[\![P \mid Q]\!] \longmapsto R$.) Observe that $R_1 \mid R_2$ cannot be a parallel component of either $[\![P]\!]$ or $[\![Q]\!]$ because then by Proposition 1 either P or Q would reduce and this is not the case.

If R_1 is a top-level component of $[\![P]\!]$ then $[\![P]\!]$ must also include a join because otherwise there would be no join in $[\![P]\!]$ that can bind some name(s) to $\varphi_{[\![]\!]} x = \widetilde{x}$ and name invariance or success sensitiveness would be shown to fail (i.e. $P \mid Q \longmapsto \mathbf{if}\ a = b\ \mathbf{then}\ \surd \mid Q'$ and $\{b/a\}\mathbf{if}\ a = b\ \mathbf{then}\ \surd \mid Q' \Downarrow$ while $\mathcal{C}^N_{|}([\![P]\!], [\![Q]\!]) \Longmapsto$ does no inputs on any part of $[\![P]\!]$ and so must always or never succeed regardless of interaction with $[\![Q]\!]$). Because the target language is asynchronous, no output can block any join and so $[\![P]\!]$ must contain an unblocked join that must include an input pattern $(\ulcorner n \urcorner)$ for some $n \neq m$. Otherwise if the join was only $[(x_1) \mid \ldots \mid (x_i)] \triangleright R'$ for some R' then $[\![P \mid \ldots \mid P]\!]$ for i instances of P would reduce while $P \mid \ldots \mid P$ does not, contradicting Proposition 1. It follows that $[\![Q]\!]$ must include both some $\langle n \rangle$ as part of some split, and some $(\ulcorner m \urcorner)$ where $m \neq n$ (this can be name-matches for any number of names $\neq n$, but assume one for simplicity) as part of some join. Otherwise the join must be of the form $[(z_1) \mid \ldots \mid (z_j) \mid (\ulcorner n \urcorner) \mid \ldots \mid (\ulcorner n \urcorner)] \triangleright S$ for k instances of $(\ulcorner n \urcorner)$ and it follows that $j + k$ instances of Q_1 in parallel would reduce when encoded $[\![Q_1 \mid \ldots \mid Q_1]\!] \longmapsto$ while $j+k$ instances of Q_1 in parallel do not reduce unencoded $Q_1 \mid \ldots \mid Q_1 \not\longmapsto$ violating Proposition 1. Thus, observe that $[\![Q]\!]$ must be 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 sent via different outputs, but assume 1 here for simplicity). Now consider the name d.

1. If $d \neq m$ and $d \neq n$ then consider $[\![P \mid Q \mid P]\!]$. After at least the reduction $R_1 \mid R_2 \longmapsto$ then $\langle d \rangle$ must be available from the reduct R of $[\![P \mid Q]\!]$.

Now consider $\mathcal{C}_{|}^{N}(\llbracket P \mid Q \rrbracket, \llbracket P \rrbracket)$ after the reduction $\llbracket P \mid Q \rrbracket \longmapsto R$ and the two top-level outputs: $\langle d \rangle$ available from R, and $\langle m \rangle$ from $\llbracket P \rrbracket$. Clearly the join that would bind d to some name in $\widetilde{x}$ (to be tested in the conditional **if** $x = b$) cannot ensure binding to d and could instead bind to m. Conclude because $d \neq m$ and without the name d being communicated to $\llbracket P \rrbracket$ the conditional **if** $x = b$ can be made to be false when it should be true via substitutions such as $\{a/b\}$ and this contradicts either name invariance or success sensitiveness.

2. If $d = n$ then this fails name invariance or success sensitiveness (by $P \mid Q \longmapsto$ **if** $a = b$ **then** $\surd \mid Q'$ and $\{b/a\}$**if** $a = b$ **then** $\surd \mid Q' \Downarrow$); or d must be bound to some name in $\widetilde{x}$ as in the previous case.
3. If $d = m$ then consider where $\langle d \rangle$ appears in $\llbracket Q \rrbracket$.
   - If $\langle d \rangle$ is top-level in $\llbracket Q \rrbracket$ then there exist some k such that k instances of Q_1 in parallel would reduce if encoded $\llbracket Q_1 \mid \ldots \mid Q_1 \rrbracket \longmapsto$ while k instances of Q_1 in parallel unencoded do not reduce $Q_1 \mid \ldots \mid Q_1 \mapsto\!\!\!\!\!\!\rightarrow$ and this violates Proposition 1.
   - If $\langle d \rangle$ is not top-level in $\llbracket Q \rrbracket$, instead $\langle d \rangle$ is top-level in some S where $\llbracket P \mid Q \rrbracket \Longmapsto S$. Conclude in the same manner as in the first case.

If R_1 is a top-level component of $\llbracket Q \rrbracket$ then $\llbracket Q \rrbracket$ must also include a top-level join because otherwise if $Q' = \Omega$ (where Ω is a divergent process) then $\llbracket Q \rrbracket$ would always diverge or never diverge regardless of interaction with $\llbracket P \rrbracket$ and this contradicts divergence reflection or operational correspondence. Thus $\llbracket Q \rrbracket$ must include a top-level join and further it must include an input pattern $(\ulcorner n \urcorner)$ for some $n \neq m$ (reasoning as above for R_1 in $\llbracket P \rrbracket$). Otherwise if the join was only $[(z_1) \mid \ldots \mid (z_i)] \rhd R'$ for some $\widetilde{z}$ and R' then $\llbracket Q \mid \ldots \mid Q \rrbracket$ for i instances of Q would reduce while $Q \mid \ldots \mid Q$ does not contradicting Proposition 1. Consider when $Q' =$ **if** $a = b$ **then** Ω and the substitution $\sigma = \{b/a\}$. Clearly $P \mid \sigma Q \mid Q \longmapsto S$ where either: $S \longmapsto^{\omega}$ and $S \Downarrow$; or $S \mapsto\!\!\!\!\!\!\rightarrow^{\omega}$ and $S \not\Downarrow$. Now consider the reduction $\llbracket P \mid \sigma Q \mid Q \rrbracket \longmapsto R'$ that must be between some component of $\llbracket P \rrbracket$ and either a component of $\llbracket \sigma Q \rrbracket$ or $\llbracket Q \rrbracket$ (because if $\llbracket P \rrbracket$ was not involved then $\mathcal{C}_{|}^{N}(\mathcal{C}_{|}^{N}(\llbracket (\nu n)[(\ulcorner n \urcorner)] \rhd [\langle b \rangle] \lhd \mathbf{0} \rrbracket, \llbracket \sigma Q \rrbracket), \llbracket Q \rrbracket)$ would reduce which contradicts Proposition 1). If this reduction is the initial one between $\llbracket P \rrbracket$ and $\llbracket \sigma Q \rrbracket$ then the output $\langle n \rangle$ must now be available in R' because otherwise the reduct of $\llbracket \sigma Q \rrbracket$ would be unable to reduce further and this would contradict operational correspondence (because $P \mid \sigma Q \longmapsto^{\omega}$ while $R' \mapsto\!\!\!\!\!\!\rightarrow^{\omega}$). However, this $\langle n \rangle$ can now reduce with $\llbracket Q \rrbracket$ instead of $\llbracket \sigma Q \rrbracket$, which leads to $R' \Downarrow$ and $R' \mapsto\!\!\!\!\!\!\rightarrow^{\omega}$ which contradicts operational correspondence via lack of divergence of R'.

Theorem 5. *There exists no valid encoding from $\mathcal{L}_{S,M,D,NM,L}$ into $\mathcal{L}_{A,M,D,NM,L}$.*

Proof. This is proved in a very similar manner to Theorem 4.

Corollary 2. *If there exists no valid encoding from $\mathcal{L}_{S,\beta_1,\gamma_1,\delta_1,B}$ into $\mathcal{L}_{A,\beta_2,\gamma_2,\delta_2,B}$, then there exists no valid encoding from $\mathcal{L}_{S,\beta_1,\gamma_1,\delta_1,\epsilon}$ into $\mathcal{L}_{A,\beta_2,\gamma_2,\delta_2,\epsilon}$.*

Proof. The techniques in Theorems 4 & 5 apply to all monadic joining and splitting languages, respectively. Monadic no-matching languages are simpler variants of the same proof technique, while polyadic no-matching (because polyadic name-matching can encode synchronous communication into asynchronous) is a simple generalisation of the above proofs.

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

This formalizes that there is no change to results within languages grouped by their coordination form. Separation results between coordination forms, and that synchronism and coordination are orthogonal are concluded in Section 8.

5 Coordination and Arity

This section considers the relation between non-binary coordination and arity. Although there appears to be some similarities in that both have a base case (monadic or binary), and unbounded cases (polyadic or joining/splitting, respectively), these cannot be used to encode arity into coordination unless they could be encoded otherwise.

The interesting results here are the separation results that ensure no new encodings or expressiveness. The proof technique is clearly illustrated by the following result for the joining setting.

Theorem 6. *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 $P = [\langle a, b\rangle]\triangleleft$ and $Q = [(x, y)] \triangleright \surd$. Clearly it holds that $P \mid Q \longmapsto \surd$ and so $[\![P \mid Q]\!] \Downarrow$ and $[\![P \mid Q]\!] \longmapsto$ by validity of the encoding and Proposition 3. Now consider the reduction $[\![P \mid Q]\!] \longmapsto$.

The reduction must be of some top-level component of $[\![P]\!]$ and $[\![Q]\!]$ (because of Proposition 2) of the form $[\langle a_1\rangle]\triangleleft \mid \ldots \mid [\langle a_i\rangle]\triangleleft \mid [(x_1) \mid \ldots \mid (x_i)] \triangleright R'$ for some $\widetilde{a}$ and $\widetilde{x}$ and i and R'. Now consider the process whose encoding produces $[(x_1) \mid \ldots \mid (x_i)] \triangleright R'$ at top-level, assume Q although the results do not rely on this assumption. Observe that no $[\langle a_j\rangle]\triangleleft$ are also from the encoding of Q because it follows that the encoding of i instances of Q in parallel will reduce, i.e. $[\![Q \mid \ldots \mid Q]\!] \longmapsto$, while $Q \mid \ldots \mid Q \not\longmapsto$ and this yields contradiction. Now consider two fresh processes $S = [\langle c_1, \ldots, c_k\rangle]\triangleleft$ and $T = [(z_1, \ldots, z_k)] \triangleright \mathbf{0}$ where $k \neq 2$. By validity of the encoding, since $S \mid T \longmapsto \mathbf{0}$ and $S \not\longmapsto$ and $T \not\longmapsto$, it follows that $[\![S \mid T]\!] \longmapsto$ (by Proposition 3) and $[\![S]\!] \not\longmapsto$ and $[\![T]\!] \not\longmapsto$. As above, the reduction $[\![S \mid T]\!] \longmapsto$ must be of the form $[\langle d_1\rangle] \triangleleft \mid \ldots \mid [\langle d_k\rangle] \triangleleft \mid [(z_1) \mid \ldots \mid (z_k)] \triangleright T'$ for some $\widetilde{d}$ and $\widetilde{z}$ and k and T'. Again, assume that $[\![T]\!]$ has $[(z_1) \mid \ldots \mid (z_k)] \triangleright T'$ at top-level (although the result does not rely on this assumption). Now $[\![S]\!]$ must contain at least one $[\langle d_j\rangle]\triangleleft$

(since otherwise $[\![T]\!] \longmapsto$ in violation of Proposition 1), and this must be at top-level by Proposition 2. Conclude by showing that $[\![S \mid \ldots \mid S \mid Q]\!] \longmapsto$ while $S \mid \ldots \mid S \mid Q \not\longmapsto$ for i instances of S in contradiction with Proposition 1.

The splitting result is very similar with only minor adaptations to the proof.

Theorem 7. *There exists no valid encoding from $\mathcal{L}_{A,P,D,NO,B}$ into $\mathcal{L}_{A,M,D,NO,L}$.*

Proof. A straightforward adaptation of Theorem 6.

Corollary 3. *If there exists no valid encoding from $\mathcal{L}_{\alpha_1,P,\gamma_1,\delta_1,B}$ into $\mathcal{L}_{\alpha_2,M,\gamma_2,\delta_2,B}$, then there exists no valid encoding from $\mathcal{L}_{\alpha_1,P,\gamma_1,\delta_1,\epsilon}$ into $\mathcal{L}_{\alpha_2,M,\gamma_2,\delta_2,\epsilon}$.*

Proof. The techniques in Theorems 6 & 7 apply to all joining and splitting languages, respectively. Name-matching requires only a small change of $Q = [(x, y)] \rhd \textbf{if } a = x \textbf{ then } \surd$ to then ensure binding occurs and not only name-matching; this is then proved via contradiction of name invariance and success sensitiveness like in Theorem 5. The techniques in Theorem 11 more elegantly show that channel-based communication is insufficient, so they are omitted here.

Thus any form of non-binary coordination does not allow for encoding a polyadic language into a monadic language unless it could already be encoded by some other means.

The other main results are to show that existing encodings between binary languages can be reproduced in other forms of coordination. This turns out to be a straightforward adaptation of the usual techniques.

Consider the usual encoding of $\mathcal{L}_{S,P,D,NO,B}$ into $\mathcal{L}_{S,M,C,NO,B}$ [39]:

$$[\![[\langle \widetilde{a} \rangle] \lhd P]\!] \stackrel{\text{def}}{=} (\nu c)[\overline{n}\langle c \rangle] \lhd [\overline{c}\langle a_1 \rangle] \lhd \ldots \lhd [\overline{c}\langle a_n \rangle] \lhd [\![P]\!]$$
$$[\![[(\widetilde{x})] \rhd Q]\!] \stackrel{\text{def}}{=} [n(z)] \rhd [z(x_1)] \rhd \ldots \rhd [z(x_n)] \rhd [\![Q]\!]$$

where c is not in the free names of $[\langle \widetilde{a} \rangle] \lhd P$, and z is not in the free names of $[(\widetilde{x})] \rhd Q$ or $\widetilde{x}$. Also n is derived from $\widetilde{a}$ since $\widetilde{a} = a_1, \ldots, a_n$ (and similarly for $\widetilde{x}$). Thus when an output and input agree upon their arity n then they interact with the output sending a fresh name c used for sending the n names $\widetilde{a}$.

This can be adapted in the obvious manner, shown below for the encoding of $\mathcal{L}_{S,P,D,NO,J}$ into $\mathcal{L}_{S,M,C,NO,J}$.

$$[\![[\langle \widetilde{a} \rangle] \lhd P]\!] \stackrel{\text{def}}{=} (\nu c)[\overline{n}\langle c \rangle] \lhd [\overline{c}\langle a_1 \rangle] \lhd \ldots \lhd [\overline{c}\langle a_n \rangle] \lhd [\![P]\!]$$
$$[\![[(\widetilde{x1}) \mid \ldots \mid (\widetilde{xk})] \rhd Q]\!] \stackrel{\text{def}}{=} [n1(z_1) \mid \ldots \mid nk(z_k)] \rhd [z_1(x1_1)] \rhd \ldots \rhd [z_1(x1_{n1})] \rhd$$
$$\ldots \rhd [z_k(xk_1)] \rhd \ldots \rhd [z_k(xk_{nk})] \rhd [\![Q]\!]$$

where the restrictions on z are here extended to distinct $z_1, \ldots, z_k$ for each input.

Theorem 8. *There is a valid encoding from $\mathcal{L}_{S,P,D,NO,J}$ into $\mathcal{L}_{S,M,C,NO,J}$.*

Proof. The proof technique is identical to Theorem 2.

This illustrates the key ideas for the following general result, that requires only straightforward adaptations of the proofs in the obvious manner. It is worth noting that all such results rely on the use of a channel-name, or an equivalent pattern match of some form to detect compatible arity and then ensure the right processes communicate. This is clearly available when adding channel-based communication, or when exploiting intensionality.

Theorem 9. *If there exists a valid encoding from* $\mathcal{L}_{\alpha_1,P,\gamma_1,\delta_1,B}$ *into* $\mathcal{L}_{\alpha_2,M,\gamma_2,\delta_2,B}$ *then there exists a valid encoding from* $\mathcal{L}_{\alpha_1,P,\gamma_1,\delta_1,\epsilon}$ *into* $\mathcal{L}_{\alpha_2,M,\gamma_2,\delta_2,\epsilon}$.

This confirms that encodings in the binary setting still exist in different coordination settings. Thus no expressiveness differences between languages are lost by changing coordination form, and existing results can be transferred.

6 Coordination and Communication Medium

This section considers the relation between coordination and communication medium. In general coordination is unable to encode communication medium unless it could otherwise be encoded by other features. This is proved by two main results: that if there is no valid encoding from a channel-based binary language to a dataspace-based binary language then there is no encoding when replacing binary with joining or splitting; and that if there exists a valid encoding from a channel-based binary language into a dataspace-based binary language then there exists an encoding with binary replaced by joining or splitting.

The base result for joining is illustrated in the following theorem, generalised in the corollary that follows.

Theorem 10. *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 and uses a very similar to that of Theorem 6. The differences are to use the $\mathcal{L}_{A,M,C,NO,B}$ processes $P = [\overline{a}\langle b\rangle]\triangleleft$ and $Q = [a(x)] \triangleright \surd$ initially, and then $S = [\overline{c}\langle b\rangle]\triangleleft$ and $T = [c(z)] \triangleright \mathbf{0}$ where $c \neq a$.

Theorem 11. *There exists no valid encoding from* $\mathcal{L}_{A,M,C,NO,B}$ *into* $\mathcal{L}_{A,M,D,NO,L}$.

Proof. The proof is by contradiction and very similar to that of Theorem 7, the main differences are to consider the $\mathcal{L}_{A,M,C,NO,B}$ processes $P = [\overline{a}\langle b\rangle]\triangleleft$ and $Q = [a(x)] \triangleright \surd$, and then $S = [\overline{c}\langle d\rangle]\triangleleft$ and $T = [c(z)] \triangleright \mathbf{0}$.

Corollary 4. *If there exists no valid encoding from* $\mathcal{L}_{\alpha_1,\beta_1,C,\delta_1,B}$ *into* $\mathcal{L}_{\alpha_2,\beta_2,D,\delta_2,B}$, *then there exists no valid encoding from* $\mathcal{L}_{\alpha_1,\beta_1,C,\delta_1,\epsilon}$ *into* $\mathcal{L}_{\alpha_2,\beta_2,D,\delta_2,\epsilon}$.

Proof. The technique in Theorems 10 & 11 apply to all monadic languages (the addition of name-matching can be proved using the techniques as in Theorem 4). For the polyadic no-matching setting the results above holds by observing that the arity must remain fixed for an encoding, i.e. $[\![\, [\overline{a}\langle b_1, \ldots, b_i\rangle]\triangleleft \,]\!]$ is encoded to inputs/outputs all of some arity j. If the arity is not uniform

then the encoding fails either Proposition 1 (by showing that the reduction $[\![\, [a(x)] \rhd \mathbf{0} \mid [\overline{a}\langle b_1, b_2\rangle] \lhd \,]\!] \longmapsto$ must occur) or divergence reflection (by showing that $[\![\, [a(x)] \rhd \mathbf{0} \mid [\overline{a}\langle b_1, b_2\rangle] \lhd \,]\!] \longmapsto\!\Longmapsto [\![\, [a(x)] \rhd \mathbf{0} \mid [\overline{a}\langle b_1, b_2\rangle] \lhd \,]\!]$ and so $[\![\, [a(x)] \rhd \mathbf{0} \mid [\overline{a}\langle b_1, b_2\rangle] \lhd \,]\!] \longmapsto^{\omega}$).

Thus any form of non-binary coordination does not allow for encoding channels in a dataspace-based language unless it could already be encoded.

The positive encoding results are the typical adaptations of the positive encoding results in the binary setting. The adaptation of the usual encoding for $\mathcal{L}_{S,P,C,NM,J}$ into $\mathcal{L}_{S,P,D,NM,J}$ is the obvious one as below.

$$[\![\, [\overline{a}\langle \widetilde{c}\rangle] \lhd P \,]\!] \stackrel{\text{def}}{=} [\langle a, \widetilde{c}\rangle] \lhd [\![\, P \,]\!]$$
$$[\![\, [a_1(\widetilde{x1}) \mid \ldots \mid a_k(\widetilde{xk})] \rhd Q \,]\!] \stackrel{\text{def}}{=} [(\ulcorner a_1 \urcorner, \widetilde{x1}) \mid \ldots \mid (\ulcorner a_k \urcorner, \widetilde{xk})] \rhd [\![\, Q \,]\!]$$

For the each channel-based output $\overline{a}\langle \widetilde{c}\rangle$ the channel name a is moved to the first position of the dataspace-based output $\langle a, \widetilde{c}\rangle$ in the encoding. The same is done for each channel-based input $a(\widetilde{x})$ becoming a dataspace-based input $(a, \widetilde{x})$.

Theorem 12. *There is a valid encoding from $\mathcal{L}_{S,P,C,NM,J}$ into $\mathcal{L}_{S,P,D,NM,J}$.*

Proof. The proof technique is identical to Theorem 2 (albeit simpler since each reduction in the source language corresponds to exactly one reduction in the target language and vice versa).

This illustrates the key ideas for the following general result, that requires only straightforward adaptations of the proofs in the obvious manner. Again all such results rely upon the use of pattern-matching, either via name-matching or intensionality, to represent the channel.

Theorem 13. *If there exists a valid encoding from $\mathcal{L}_{\alpha_1,\beta_1,C,\delta_1,B}$ into $\mathcal{L}_{\alpha_2,\beta_2,D,\delta_2,B}$ then there exists a valid encoding from $\mathcal{L}_{\alpha_1,\beta_1,C,\delta_1,\epsilon}$ into $\mathcal{L}_{\alpha_2,\beta_2,D,\delta_2,\epsilon}$.*

This confirms that encodings of channel-based communication into dataspace-based communication in the binary setting still exist in different coordination settings. Thus no expressiveness differences between languages are lost by changing coordination form, and existing results can be transferred.

7 Coordination and Pattern-Matching

This section considers the relations between coordination and pattern-matching. Intensionality cannot be encoded into a name-matching (or no-matching) language by exploiting joining or splitting. Similarly name-matching cannot be encoded into a no-matching language by exploiting joining or splitting.

To assist with the below theorem, define the *maximal interaction patterns* $\text{MIP}(P)$ of a process P as follows:

$$\text{MIP}(\mathbf{0}) = 0 \quad \text{MIP}((\nu a)P) = \text{MIP}(P) \quad \text{MIP}(*P) = \text{MIP}(P) \quad \text{MIP}(\surd) = 0$$
$$\left.\begin{array}{r}\text{MIP}(P \mid Q)\\ \text{MIP}(\textbf{if } s = t \textbf{ then } P \textbf{ else } Q)\end{array}\right\} = \begin{cases}\text{MIP}(P) & \text{if } \text{MIP}(P) > \text{MIP}(Q)\\ \text{MIP}(Q) & \text{otherwise}\end{cases}$$
$$\text{MIP}([\langle\widetilde{t_1}\rangle \mid \ldots \mid \langle\widetilde{t_i}\rangle] \lhd P) = \textstyle\sum_{j=0,\ldots,i} |\widetilde{t_j}|$$
$$\text{MIP}([\overline{s_1}\langle\widetilde{t_1}\rangle \mid \ldots \mid \overline{s_i}\langle\widetilde{t_i}\rangle] \lhd P) = i + \textstyle\sum_{j=0,\ldots,i} |\widetilde{t_j}|$$
$$\text{MIP}([(\widetilde{p_1}) \mid \ldots \mid (\widetilde{p_i})] \rhd P) = \textstyle\sum_{j=0,\ldots,i} |\widetilde{p_j}|$$
$$\text{MIP}([a_1(\widetilde{p_1}) \mid \ldots \mid a_i(\widetilde{p_i})] \rhd P) = i + \textstyle\sum_{j=0,\ldots,i} |\widetilde{p_j}| \; .$$

The intuition is that $\text{MIP}(P)$ indicates the maximum number of patterns that can be matched by any single split or join of P (i.e. any single *OutProc* or *InProc*). For the null process, restriction, parallel composition, replication, conditional, and success process this is straightforward, the only non-trivial case is the conditional **if** $s = t$ **then** P **else** Q where both P and Q can be considered (this is to allow flexibility when substitutions may allow either P or Q to be possible). For the splits (resp. joins), when the language is dataspace-based then this is the sum of the arities of all outputs in the split (resp. inputs in the join), and when the language is channel-based the maximum interaction patterns also counts the channel terms ($i+$ above).

Lemma 4. *Given a process P (for any language), for all substitutions σ it holds that* $\text{MIP}(P) = \text{MIP}(\sigma(P))$.

Proof. The proof is straightforward by induction on the structure of P.

Observe that in name-matching languages, for any process P then $\text{MIP}(P)$ is the upper bound on the number of names that can be matched in any split or join of P. For no-matching languages the upper bound is at most $\frac{\text{MIP}(P)}{2}$ (when the maximum arity of any output in a split or input in a join is 1), although this is less significant to the result below. (For intensional languages there is no upper bound, related to $\text{MIP}(P)$ or otherwise, however since the goal is to use $\text{MIP}(P)$ to reason about non-intensional languages, this is not relevant.)

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

Theorem 14. *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. 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 $P = [(\ulcorner a \urcorner)] \rhd P'$ and $Q = [\langle a \rangle] \lhd$. Because $P \mid Q \longmapsto$ then by Proposition 3 $[\![P \mid Q]\!] \longmapsto$. Now define $k = \text{MIP}([\![P \mid Q]\!])$ and define $\sigma = \{b_1 \bullet \ldots \bullet b_{k+1}/a\}$. Observe that $\sigma(P \mid Q) \longmapsto$ and so $[\![\sigma(P \mid Q)]\!] \longmapsto$ by Proposition 3, and that the reduction $[\![\sigma(P \mid Q)]\!] \longmapsto$ can match at most k

names because $k \geq$ the maximum possible patterns of any join in $[\![\sigma(P \mid Q)]\!]$ by Lemma 4 and $\delta \neq I$. Therefore, there must exist at least one name b_j (but assume only b_j for simplicity here) that is not being tested for equality either by a name match or channel name in the reduction $[\![\sigma(P \mid Q)]\!] \longmapsto$. Define $P' = [\langle m \rangle]\lhd$ and $\rho = \{m/b_j, b_j/m\}$. Now since b_j is not tested for equality in the reduction $[\![\sigma P \mid \sigma Q]\!] \longmapsto$ it follows that $[\![\rho\sigma P \mid \sigma Q]\!] \longmapsto$. Conclude by showing that $*(\rho\sigma P \mid \sigma Q)$ does not reduce (or diverge) while because $[\![\rho\sigma P \mid \sigma Q]\!] \longmapsto$ it follows that $[\![*(\rho\sigma P \mid \sigma Q)]\!] \longmapsto^{\omega}$ in violation of divergence reflection.

Theorem 15. *There exists no valid encoding from $\mathcal{L}_{A,M,D,I,B}$ into $\mathcal{L}_{-,-,-,\delta,L}$ where $\delta \neq I$.*

Proof. The same technique as in Theorem 14 can be applied for splitting.

Corollary 5. *If there exists no valid encoding from $\mathcal{L}_{\alpha_1,\beta_1,\gamma_1,I,B}$ into $\mathcal{L}_{\alpha_2,\beta_2,\gamma_2,\delta,B}$, then there exists no valid encoding from $\mathcal{L}_{\alpha_1,\beta_1,\gamma_1,I,\epsilon}$ into $\mathcal{L}_{\alpha_2,\beta_2,\gamma_2,\delta,\epsilon}$.*

Proof. The joining case is by Theorem 14 and the splitting by Theorem 15.

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

The next results show that coordination is insufficient to encode name matching. Unlike Theorem 14, these need to be separated into two results due to the encoding from $\mathcal{L}_{A,M,D,NM,B}$ into $\mathcal{L}_{A,M,C,NO,B}$ [28].

Theorem 16. *There exists no valid encoding from $\mathcal{L}_{A,M,D,NM,B}$ into $\mathcal{L}_{\alpha,\beta,D,NO,J}$.*

Proof. The proof is by contradiction, assume there exists a valid encoding $[\![\cdot]\!]$. Consider the $\mathcal{L}_{A,M,D,NM,B}$ processes $P = [\langle a \rangle]\lhd$ and $Q = [(\ulcorner a \urcorner)] \rhd ([\langle b \rangle] \lhd \mid \surd)$. Observe that $P \mid Q \longmapsto$ and $P \mid Q \Downarrow$ and so $[\![P \mid Q]\!] \longmapsto$ and $[\![P \mid Q]\!] \Downarrow$ by Proposition 3 and validity of the encoding. Now consider the substitution $\sigma = \{c/a\}$, it follows that $P \mid \sigma Q \mapsto\!\!\!\!\!\!\rightarrow$ and so $[\![P \mid \sigma Q]\!] \mapsto\!\!\!\!\!\!\rightarrow$ by Proposition 1. Now if there is no blocking via an **if** $a_1 = a_2$ **then** S_1 **else** S_2 then this yields a contradiction in the usual manner (see Theorem 4) via either: $[\![P \mid \sigma Q]\!] \Downarrow$ while $P \mid \sigma Q \not\Downarrow$, or $[\![\sigma(P \mid Q)]\!] \not\Downarrow$ while $\sigma(P \mid Q) \Downarrow$. Therefore there must be a conditional **if** $a_1 = a_2$ **then** S_1 **else** S_2 that prevents reduction (there may be many, but assume one for simplicity). Further, this must be in $[\![Q]\!]$ because otherwise this would violate compositionality and success sensitiveness with $\mathcal{C}^N_{\mid}([\![P]\!], \cdot)$ replacing $\cdot$ with $[\![Q]\!]$ or $[\![\sigma Q]\!]$. It must be that $a_1 \neq a_2$ in $[\![Q]\!]$ because otherwise if $a_1 = a_2$ then no substitution σ' (defined by name invariance $\sigma'[\![\cdot]\!] = [\![\sigma(\cdot)]\!]$) could make $\sigma' a_1 \neq \sigma' a_2$ when $a_1 = a_2$. Therefore, it must be that $\sigma' a_1 = \sigma' a_2$, however by considering the substitution $\rho = \{a/c\}$ (and associated ρ' from name invariance) it must be that $\rho'\sigma'[\![Q]\!] \simeq [\![Q]\!]$, yet ρ' cannot induce inequality in $\sigma' a_1 = \sigma' a_2$ and because no other mechanism can prevent interaction then because $[\![P \mid \rho\sigma Q]\!] \longmapsto$ (by Proposition 3) it follows that $[\![P \mid \sigma Q]\!] \longmapsto$ in violation of Proposition 1.

Theorem 17. *There exists no valid encoding from* $\mathcal{L}_{A,P,D,NM,B}$ *into* $\mathcal{L}_{\alpha,\beta,\gamma,NO,J}$.

Proof. The proof is by contradiction, assume there exists a valid encoding $[\![\cdot]\!]$. If $\gamma = D$ then the proof of Theorem 16 applies, so the rest of this proof shall assume $\gamma = C$. Consider the $\mathcal{L}_{A,P,D,NM,B}$ processes $P = [\langle a, b\rangle]\triangleleft$ and $Q = [(\ulcorner a \urcorner, \ulcorner b \urcorner)] \rhd Q'$. Observe that $P \mid Q \longmapsto$ and so $[\![P \mid Q]\!] \longmapsto$ by Proposition 3 and validity of the encoding. The reduction $[\![P \mid Q]\!] \longmapsto$ must be of the form $[\overline{c_1}\langle\widetilde{m_1}\rangle] \triangleleft \mid \ldots \mid [\overline{c_i}\langle\widetilde{m_i}\rangle] \triangleleft \mid [c_1(\widetilde{z_1}) \mid \ldots \mid c_i(\widetilde{z_i})] \rhd R'$ for some $\widetilde{c}$ and $\widetilde{m}$ and $\widetilde{z}$ and i and R'. Now consider the substitutions $\sigma = \{c/a\}$ and $\rho = \{d/b\}$ (and their associated substitutions on encoded processes σ' and ρ' determined by name invariance and validity of the encoding). Observe that because $\sigma P \mid Q \not\longmapsto$ it follows that $\mathcal{C}^N_{|}(\sigma'[\![P]\!], [\![Q]\!]) \not\longmapsto$ by Proposition 1. The reduction can only be prevented by either a conditional or the changing of a channel name via the substitution σ'. (Observe that conditionals may introduce or remove splits and joins accounting for missing components or changes in arity, and so the only other possibility for preventing reduction is by changing the channel name.) If the reduction is prevented due to a conditional then contradiction can be achieved as in Theorem 16, so it must be that $\sigma'(c_j) \neq c_j$ for some $j \in \{1, \ldots, i\}$. The same can be shown for $\rho P \mid Q$ and some c_k such that $\rho'(c_k) \neq c_k$. Further, by exploiting the inverse substitutions denoted $\mathsf{inv}(\sigma)$ for the inversve of σ (defined in the obvious manner) it must be that $j \neq k$, because otherwise $\mathsf{inv}(\sigma')\rho'(c_j) = c_j$ and contradiction could be shown because $\mathsf{inv}(\sigma)\rho P \mid Q \not\longmapsto$. Finally, because the join and all the splits involved in the reductions $[\![P \mid Q]\!] \longmapsto$ must be at top-level by Proposition 2, conclude by observing that $[\![\sigma P \mid \rho P \mid Q \mid \rho\sigma Q]\!] \longmapsto$ because all the components required for interaction are at top-level and because $[\![Q \mid \rho\sigma Q]\!]$ provides all the outputs (or inputs) required for the inputs (or outputs) of σP and ρP. However, because $\sigma P \mid \rho P \mid Q \mid \rho\sigma Q \not\longmapsto$ this contradicts Proposition 1.

The next two results are the splitting version of the two theorems above.

Theorem 18. *There exists no valid encoding from* $\mathcal{L}_{A,M,D,NM,B}$ *into* $\mathcal{L}_{\alpha,\beta,D,NO,L}$.

Proof. The same technique as in Theorem 16 can be applied here.

Theorem 19. *There exists no valid encoding from* $\mathcal{L}_{A,P,D,NM,B}$ *into* $\mathcal{L}_{\alpha,\beta,\gamma,NO,L}$.

Proof. The same technique as in Theorem 17 can be applied here.

Corollary 6. *If there exists no valid encoding from* $\mathcal{L}_{\alpha_1,\beta_1,\gamma_1,NM,B}$ *into* $\mathcal{L}_{\alpha_2,\beta_2,\gamma_2,\delta,B}$, *then there exists no valid encoding from* $\mathcal{L}_{\alpha_1,\beta_1,\gamma_1,NM,\epsilon}$ *into* $\mathcal{L}_{\alpha_2,\beta_2,\gamma_2,\delta,\epsilon}$.

Proof. The joining cases are covered by Theorems 16 & 17 and the splitting by Theorems 18 & 19.

Thus coordination does not allow for encoding name-matching into a no-matching language unless it could already be encoded by some other means.

For the positive results that remain it is straightforward to adapt the existing encodings in the same manner as for Corollary 1, and Theorems 9 & 13.

Theorem 20. *If there exists a valid encoding from* $\mathcal{L}_{\alpha,\beta,\gamma,\delta_1,B}$ *into* $\mathcal{L}_{\alpha,\beta,\gamma,\delta_2,B}$ *where* $\delta_1 \leq \delta_2$ *then there exists a valid encoding from* $\mathcal{L}_{\alpha,\beta,\gamma,\delta_1,\epsilon}$ *into* $\mathcal{L}_{\alpha,\beta,\gamma,\delta_2,\epsilon}$.

Proof. The same techniques as Corollary 1, and Theorems 9 & 13 can be applied.

Finally, the positive results that preserve encodings when changing the coordination feature can be combined into a single general result.

Corollary 7. *If there exists a valid encoding from* $\mathcal{L}_{\alpha_1,\beta_1,\gamma_1,\delta_1,B}$ *into* $\mathcal{L}_{\alpha_2,\beta_2,\gamma_2,\delta_2,B}$ *then there exists a valid encoding from* $\mathcal{L}_{\alpha_1,\beta_1,\gamma_1,\delta_1,\epsilon}$ *into* $\mathcal{L}_{\alpha_2,\beta_2,\gamma_2,\delta_2,\epsilon}$.

Proof. By combining Corollary 1, and Theorems 9, 13, & 20.

8 Coordination and Other Features

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

The expressive power gained by joining or splitting 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 particular 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 non-binary languages is ∞ since there is no bound on the number of inputs that can be part of a join, or outputs that can be part of a split.

Theorem 21. *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 $[\![\cdot]\!]$. Fix N and pick i processes S_1 to S_i where $i = \textsc{Cd}(\mathcal{L}_2) + 1$ and $N = \bigcup \mathsf{fn}(S_j)$ for $j \in \{1, \ldots, i\}$ 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 $\mathit{block}(S_j)$. By validity of the encoding and Proposition 3 it must be that $[\![S_1 \mid \ldots \mid S_i]\!] \Downarrow$ and $[\![S_1 \mid \ldots \mid S_i]\!] \longmapsto$.

By compositionality of the encoding $[\![S_1 \mid \ldots \mid S_i]\!] = \mathcal{C}_S = \mathcal{C}^N_|([\![S_1]\!], \mathcal{C}^N_|(\ldots, \mathcal{C}^N_|([\![S_{i-1}]\!], [\![S_i]\!])))$. 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 $[\![S_1 \mid \ldots \mid S_{j-1} \mid \mathit{block}(S_j) \mid S_{j+1} \mid \ldots \mid S_i]\!] \longmapsto$ (by replacing the $[\![S_j]\!]$ in the context $\mathcal{C}_S$ with $[\![\mathit{block}(S_j)]\!]$). 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 $S_1 \mid \ldots \mid S_{j-1} \mid \mathit{block}(S_j) \mid S_{j+1} \mid \ldots \mid S_i \not\longmapsto$. The only other possibility to prevent reduction of $[\![S_1 \mid \ldots \mid S_{j-1} \mid \mathit{block}(S_j) \mid S_{j+1} \mid \ldots \mid S_i]\!]$ is if $[\![\mathit{block}(S_j)]\!]$ blocks the reduction by blocking some $[\![S_k]\!]$. This can only occur when $[\![S_k]\!]$ is either underneath an interaction primitive (e.g. $[\overline{s}\langle\widetilde{t}\rangle] \lhd [\![S_k]\!]$) or inside a conditional (e.g. **if** $s = t$ **then** $[\![S_k]\!]$ where $s \neq t$). Both require that $[\![S_k]\!]$ not be top-level in $\mathcal{C}_S$, which can be proven contradictory by Proposition 2.

The above may not appear intuitive when some implementations of n-ary coordination are achieved by 2-ary coordination. However, the result shows that such implementations must have conditions under which they begin coordination when the coordination cannot be completed and so either: become stuck waiting for further coordination; or must roll-back to a prior state. The first case would here invalidate the encoding by blocking an alternative valid coordination, while the second case would here indicate an infinite reduction sequence again invalidating the encoding.

Corollary 8. *There exists no valid encoding from* $\mathcal{L}_{-,-,-,-,\epsilon}$ *into* $\mathcal{L}_{-,-,-,-,B}$ *where* $\epsilon \neq B$.

In the other direction the result is ensured by Remark 1. Thus for any languages $\mathcal{L}_{\alpha,\beta,\gamma,\delta,\epsilon_1}$ and $\mathcal{L}_{\alpha,\beta,\gamma,\delta,\epsilon_2}$ where $\epsilon_1 < \epsilon_2$ then it holds that $\mathcal{L}_{\alpha,\beta,\gamma,\delta,\epsilon_2}$ is strictly more expressive than $\mathcal{L}_{\alpha,\beta,\gamma,\delta,\epsilon_1}$. That is, joining or splitting languages are strictly more expressive than binary languages.

Thus coordination turns out to be orthogonal to all other features, since from the prior sections coordination cannot encode any other feature, and here it is proven that other features cannot encode coordination.

9 Within Coordination

This section considers relations between different forms of coordination. It turns out that there are some encodings from joining languages into splitting languages and vice versa, however most joining and splitting languages are unrelated.

A joining (resp. splitting) language without matching capabilities can be encoded into a splitting (resp. joining) language. For example, consider the encoding from $\mathcal{L}_{S,M,C,NO,J}$ to $\mathcal{L}_{S,M,C,NO,L}$ that is the identity on all forms except the output and join as follows:

$$\begin{aligned} [\![\, [\overline{a}\langle b\rangle] \lhd P \,]\!] &\stackrel{\text{def}}{=} [a(c)] \rhd [\overline{c}\langle b\rangle] \lhd [\![\, P \,]\!] \\ [\![\, [a_1(x_1) \mid \ldots \mid a_i(x_i)] \rhd Q \,]\!] &\stackrel{\text{def}}{=} (\nu \widetilde{c})([\overline{a_1}\langle c_1\rangle \mid \ldots \mid \overline{a_i}\langle c_i\rangle] \lhd \\ &\qquad [c_1(x_1)] \rhd \ldots \rhd [c_i(x_i)] \rhd [\![\, Q \,]\!]) \end{aligned}$$

where c is not b or in the free names of P; and $\widetilde{c}$ does not intersect with $\widetilde{a}$ or $\widetilde{x}$ or the free names of Q. The key idea is that the direction of communication is reversed; splits become joins (with outputs becoming inputs), and joins become splits (with inputs becoming outputs), a fresh name c is transmitted to be used for then sending the original name b from the output to the encoded join. Thus the requirement that all inputs of a join interact at once is maintained by all the outputs of the split. Observe that this is similar in concept to the encoding of synchrony into asynchrony by Honda & Tokoro [32].

Theorem 22. *The encoding from* $\mathcal{L}_{S,M,C,NO,J}$ *into* $\mathcal{L}_{S,M,C,NO,L}$ *is valid.*

Proof. The proof technique is identical to Theorem 2.

The same approach can be used to encode $\mathcal{L}_{S,M,C,NO,L}$ into $\mathcal{L}_{S,M,C,NO,J}$ with adjustments to the split and join as follows:

$$[\![\, [\overline{a_1}\langle b_1\rangle \mid \ldots \mid \overline{a_i}\langle b_i\rangle] \lhd P \,]\!] \stackrel{\text{def}}{=} [a_1(c_1) \mid \ldots \mid a_i(c_i)] \rhd [\overline{c_1}\langle b_1\rangle] \lhd \ldots \lhd [\overline{c_i}\langle b_i\rangle] \lhd [\![\, P \,]\!]$$

$$[\![\, [a(x)] \rhd Q \,]\!] \stackrel{\text{def}}{=} (\nu c)[\overline{a}\langle c\rangle] \lhd [c(x)] \rhd [\![\, Q \,]\!]$$

$\widetilde{c}$ does not intersect $\widetilde{b}$ or free names of P; and c is not a or in free names of Q.

Theorem 23. *The encoding from $\mathcal{L}_{S,M,C,NO,L}$ into $\mathcal{L}_{S,M,C,NO,J}$ is valid.*

Proof. The proof technique is identical to Theorem 2.

Interestingly there are encodings that do not require channel names for the language that are dataspace-based and no-matching. Consider the following encoding from $\mathcal{L}_{S,M,D,NO,J}$ to $\mathcal{L}_{S,M,D,NO,L}$ that is the identity of all forms except the split and join as follows:

$$[\![\, [\langle a\rangle] \lhd P \,]\!] \stackrel{\text{def}}{=} [(x)] \rhd [\langle a\rangle] \lhd [\![\, P \,]\!]$$

$$[\![\, [(x_1) \mid \ldots \mid (x_i)] \rhd Q \,]\!] \stackrel{\text{def}}{=} (\nu \widetilde{c})([\langle c_1\rangle \mid \ldots \mid \langle c_i\rangle] \lhd [(x_1)] \rhd \ldots \rhd [(x_i)] \rhd [\![\, Q \,]\!])$$

where x is not a or in the free names of $[\![\, P \,]\!]$; and $\widetilde{c}$ does not intersect the free names of $[\![\, Q \,]\!]$. Again the key idea is to reverse the direction of communication, only now no attempt is made to maintain the relation of which encoded process initiated communication with which. This turns out not to be a concern since the split that represents the encoded join ensures sufficient encoded outputs are available before reduction, although the actual binding of names may not match that initial reduction. For example, $[\![\, [\langle a\rangle] \lhd \mid [\langle b\rangle] \lhd \,]\!]$ may begin a reduction with $[\![\, [(x) \mid (y)] \rhd P \,]\!]$, i.e. $[\![\, [\langle a\rangle] \lhd \mid [\langle b\rangle] \lhd \mid [(x) \mid (y)] \rhd P \,]\!] \longmapsto S$ and similarly, $[\![\, [\langle c\rangle] \lhd \,]\!]$ and $[\![\, [(z)] \rhd Q \,]\!]$ may begin with a reduction $[\![\, [\langle c\rangle] \lhd [(z)] \rhd Q \,]\!] \longmapsto T$. Despite these initial reductions, it is still possible for $S \mid T \Longmapsto \{b/x, c/y\}P \mid \{a/z\}Q$. This may seem unusual, but despite this lack of control over where the actual names are communicated after the initial reductions of an encoded join, this still meets the criteria for a valid encoding.

Theorem 24. *The encoding from $\mathcal{L}_{S,M,D,NO,J}$ into $\mathcal{L}_{S,M,D,NO,L}$ is valid.*

Proof. The proof technique is identical to Theorem 2.

Theorem 25. *There exists a valid encoding from $\mathcal{L}_{S,M,D,NO,L}$ into $\mathcal{L}_{S,M,D,NO,J}$.*

Proof. The same approach is used as in Theorem 23.

The same techniques can be applied to the asynchronous and polyadic variations of the above languages.

Theorem 26. *The languages $\mathcal{L}_{-,\beta,\gamma,NO,J}$ and $\mathcal{L}_{-,\beta,\gamma,NO,L}$ can validly encode each other.*

Proof. The proof technique is identical to Theorem 2.

However there are usually not encodings between joining and splitting languages. This can be illustrated by considering attempts to encode any sort of name-matching from either joining or splitting into the other.

Theorem 27. *There exists no valid encoding from* $\mathcal{L}_{A,M,D,NM,J}$ *into* $\mathcal{L}_{-,-,-,-,L}$.

Proof. The proof is by contradiction. (Note that the proof assumes channels in the target language as this is more general, they are simply omitted for the data-space based languages.) Consider the processes $P = [(\ulcorner a \urcorner) \mid (\ulcorner b \urcorner)] \rhd P'$ and $Q_1 = [\langle a \rangle] \lhd$ and $Q_2 = [\langle b \rangle] \lhd$. Because $P \mid Q_1 \mid Q_2 \longmapsto \surd$ by instantiating $P' = \surd$, then by validity of the encoding and Proposition 3 $[\![P \mid Q_1 \mid Q_2]\!] \longmapsto$, now consider this reduction. It must be between $R_1 = [\overline{s_1}\langle \widetilde{t}_1 \rangle \mid \ldots \mid \overline{s_i}\langle \widetilde{t}_i \rangle] \lhd R_1'$ and R_2 for some $\widetilde{s}$ and $\widetilde{t}$ and R_1' and R_2 such that $R_1 \mid R_2 \longmapsto$. Observe that $R_1 \mid R_2$ cannot be a parallel component of $[\![P \mid Q_1 \mid \mathbf{0}]\!]$ or $[\![P \mid \mathbf{0} \mid Q_2]\!]$ or $[\![\mathbf{0} \mid Q_1 \mid Q_2]\!]$ because this would contradict Proposition 1.

If R_1 is a top level component of $[\![Q_1]\!]$ or $[\![Q_2]\!]$ then $[\![P]\!]$ must exhibit some join $[s(\widetilde{p})] \rhd R_2'$ that interacts with R_1 because otherwise $[\![\mathbf{0} \mid Q_1 \mid Q_2]\!] \longmapsto$ which contradicts Proposition 1. Now by Proposition 3 and considering the substitution $\sigma = \{c/a, c/b\}$ it must be that $[s(\widetilde{p})] \rhd R_2'$ tests equality of some translated names of both $\varphi_{[\![]\!]}(a)$ and $\varphi_{[\![]\!]}(b)$ because otherwise one of $[\![P \mid \sigma Q_1 \mid Q_2]\!]$ or $[\![P \mid Q_1 \mid \sigma Q_2]\!]$ or $[\![P \mid \sigma Q_1 \mid \sigma Q_2]\!]$ would reduce in contradiction with Proposition 1. Further, because $[s(\widetilde{p})] \rhd R_2'$ must test names from both $[\![Q_1]\!]$ and $[\![Q_2]\!]$ then R_1 must come from only the encoding $[\![Q_1 \mid Q_2]\!]$ and not from either of $[\![Q_1]\!]$ and $[\![Q_2]\!]$ alone. However, by considering $S = [(x)] \rhd S_1 \mid [(y)] \rhd S_2$ and the fact that $S \mid Q_1 \mid Q_2 \longmapsto$ and $S \mid Q_1 \mid \mathbf{0} \longmapsto$ and $S \mid \mathbf{0} \mid Q_2 \longmapsto$ it follows that either: both $[\![Q_1]\!]$ and $[\![Q_2]\!]$ must exhibit a top-level split, or $[\![Q_1 \mid Q_2]\!]$ must exhibit more than one top-level split. In both cases this yields a contradiction via $[\![S \mid P \mid Q_1 \mid Q_2]\!]$ with $S_1 = \surd$ and $P' = \Omega$ by violating either success sensitiveness or operational correspondence (as in concluding Theorem 4).

Therefore, it must be that R_1 is a top-level component of $[\![P]\!]$, so consider the process $S = [(z)] \rhd S'$ such that $Q_1 \mid S \longmapsto$ and $[\![Q_1 \mid S]\!] \longmapsto$ (by instantiating $S' = \surd$ and Proposition 3). Observe that $[\![Q_1]\!]$ interacts with $[\![P]\!]$ via some $[s_j(\widetilde{p})] \rhd Q_1'$ (there may be many such, but assume one for simplicity because the following can be proved for all of them). Now consider the reduction $[\![Q_1 \mid S]\!] \longmapsto$:

- If it is via the same $[s_j(\widetilde{p})] \rhd Q_1'$ that interacts with $[\![P]\!]$ then there must be some $[\ldots \mid \overline{s_j}\langle \widetilde{t} \rangle \mid \ldots] \lhd T'$ in $[\![S]\!]$ such that $\textsc{Match}(\widetilde{t}, \widetilde{p})$ is defined. Observe that this must not rely on equality/matching of any names that depend upon a because otherwise the substitution $\sigma = \{c/a\}$ would prevent the reduction of $[\![Q_1 \mid \sigma S]\!]$ yet $Q_1 \mid \sigma S \longmapsto$ and so this yields contradiction via Proposition 3. However, because no name in $[s_j(\widetilde{p})] \rhd Q_1'$ depends upon a it follows that $[\![P \mid \sigma Q_1 \mid Q_2]\!] \longmapsto$ which contradicts Proposition 1.
- Otherwise it must be that the reduction is via some different input or output in $[\![Q_1]\!]$. However, then contradiction can be achieved via success sensitive-

ness or divergence reflection in a similar manner to the conclusion of Theorem 4 by instantiating $P' = \surd$ and $S' = \textbf{if } z = a \textbf{ then } \Omega$ and considering $[\![P \mid Q_1 \mid Q_2 \mid S \mid \sigma Q_1]\!]$.

Theorem 28. *There exists no valid encoding from $\mathcal{L}_{A,M,D,NM,L}$ into $\mathcal{L}_{-,-,-,-,J}$.*

Proof. The proof is by contradiction in a similar manner to Theorem 27 by starting with the processes $P = [\langle a \rangle \mid \langle b \rangle] \triangleleft$ and $Q_1 = [(\ulcorner a \urcorner)] \rhd Q_1'$ and $Q_2 = [(\ulcorner b \urcorner)] \rhd Q_2'$.

These results show that once name-matching (or intensionality) is in play it is no longer possible for splitting or joining languages to encode one another.

Corollary 9. *There exists no valid encoding from $\mathcal{L}_{A,M,D,I,J}$ into $\mathcal{L}_{-,-,-,-,L}$.*

Corollary 10. *There exists no valid encoding from $\mathcal{L}_{A,M,D,I,L}$ into $\mathcal{L}_{-,-,-,-,J}$.*

Thus although there are some languages where a difference only of joining or splitting prove equally expressive, in general different forms of coordination usually indicate differences in expressive power.

10 Conclusions

In the theme of Barnhard Steffen's work this paper demonstrates expressiveness of different approaches to workflow coordination and their relation to other language features. This paper formalises that increases in coordination always correspond to increases in expressive power: both joining and splitting languages are strictly more expressive than binary languages. However, this expressive power does not allow coordination to encode other aspects of communication; increasing coordination does not allow encoding of other features unless they could already be encoded.

This formalizes that languages using Join Calculus style joins such as general rendezvous calculus, and m-calculus cannot be validly encoded into binary languages, regardless of other features. Although there exist approaches to encoding from these kinds of languages into π-calculus, these often do not meet the criteria for a *valid encoding* used here. A common approach [19] used in such encodings is to encode joins by $[\![[m(x) \mid n(y)] \rhd P]\!] = m(x).n(y).[\![P]\!]$, however this can easily fail operational correspondence, or success sensitivity. For example consider $P_1 = [c_1(w) \mid c_2(x)] \rhd \surd$ and $P_2 = [c_2(y) \mid c_1(z)] \rhd \Omega$ and $Q = \overline{c_1}\langle a \rangle \mid \overline{c_2}\langle b \rangle$. Together $P_1 \mid P_2 \mid Q$ can either report success or diverge, but their encoding $[\![P_1 \mid P_2 \mid Q]\!]$ can deadlock. Even ordering the channel names to prevent this can be shown to fail under substitutions. However, there are different forms of encodings between such calculi and π-calculi that do not meet the criteria used here [19, 51]. The interesting cases are where joining calculi are encoded into π-calculi. Those in [19] still suffer the problem above for the criteria used here although they are not an issue for the full abstraction result

obtained. In [51] the author asserts the existence of an encoding from Join Calculus into a π-calculus with the same communication paradigm as $\mathcal{L}_{A,M,C,N,B}$ here. However, they choose a different instantiation of Gorla's encoding criteria to here, opting for a non-reduction-sensitive equivalence relation. These different choices in the formal relations mean the results do not quite conflict with those here, instead illustrating the impact of different encoding criteria. This aligns with other results [52] where it is shown that the communication primitives of joins cannot be encoded into the communication primitives of π-calculi under different definitions of encoding.

That a language with coordination degree n cannot be encoded into a language with coordination degree less than n aligns with some recent results. Laneve and Vitale considered "synchronization" from a perspective that appears similar but is in fact rather different [36]. They consider languages to have *n-join* forms where n is the number of inputs a process can have. Thus, $[a_1(x_1) \mid \ldots \mid a_i(x_i)] \rhd P$ has an i-join. They then show that an n-join language cannot be encoded into an $(n-1)$-join language, this agrees with the results here. (Indeed, the results here generalise this by considering both joining and splitting.) They further show that if mixed "joins" are allowed that can contain both inputs and outputs (e.g. $[a(x) \mid \overline{b}\langle c\rangle] \rhd P$) then any n-join language can be encoded into a 3-join language. However, in doing this the number of processes that must coordinate to perform the encoded reduction increases, i.e. the coordination degree must increase. This further reinforces 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. Bengtson and J. Parrow. Formalising the pi-calculus using nominal logic. *Logical Methods in Computer Science*, 5(2), 2009.
4. L. Bocchi and L. Wischik. A process calculus of atomic commit. *Electronic Notes in Theoretical Computer Science*, 105(0):119 – 132, 2004. Proceedings of the First International Workshop on Web Services and Formal Methods (WSFM 2004).
5. M. Boreale, C. Fournet, and C. Laneve. Bisimulations in the join-calculus. In *Programming Concepts and Methods PROCOMET '98*, IFIP – The International Federation for Information Processing, pages 68–86. Springer US, 1998.
6. G. Boudol. Notes on algebraic calculi of processes. In *Logics and Models of Concurrent Systems*, pages 261–303. Springer-Verlag New York, Inc., New York, NY, USA, 1985.
7. G. Boudol. Asynchrony and the pi-calculus. *Rapport de Recherche 1702*, 1992.
8. O. Burkart, D. Caucal, and B. Steffen. Bisimulation collapse and the process taxonomy. In U. Montanari and V. Sassone, editors, *CONCUR '96, Concurrency Theory, 7th International Conference, Pisa, Italy, August 26-29, 1996, Proceedings*, volume 1119 of *Lecture Notes in Computer Science*, pages 247–262. Springer, 1996.

9. N. Busi, R. Gorrieri, and G. Zavattaro. On the expressiveness of linda coordination primitives. *Information and Computation*, 156(1-2):90–121, 2000.
10. 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.
11. 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.
12. S. Cassel, F. Howar, B. Jonsson, M. Merten, and B. Steffen. A succinct canonical register automaton model. *J. Log. Algebr. Meth. Program.*, 84(1):54–66, 2015.
13. G. Castagna, R. De Nicola, and D. Varacca. Semantic subtyping for the pi-calculus. *Theoretical Computer Science*, 398(1-3):217–242, May 2008.
14. J. de Lara and A. Zisman, editors. *Fundamental Approaches to Software Engineering - 15th International Conference, FASE 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, volume 7212 of *Lecture Notes in Computer Science*. Springer, 2012.
15. 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.
16. 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.
17. R. de Simone. Higher-level synchronising devices in Meije-SCCS. *Theoretical Computer Science*, 37:245–267, 1985.
18. C. Ene and T. Muntean. A broadcast-based calculus for communicating systems. In *Parallel and Distributed Processing Symposium, International*, volume 3, pages 30149b–30149b. IEEE Computer Society, 2001.
19. C. Fournet and G. Gonthier. The reflexive cham and the join-calculus. In *Proceedings of the 23rd ACM Symposium on Principles of Programming Languages*, pages 372–385. ACM Press, 1996.
20. D. Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.
21. T. Given-Wilson. *Concurrent Pattern Unification*. PhD thesis, University of Technology, Sydney, Australia, 2012.
22. 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.
23. T. Given-Wilson. On the Expressiveness of Intensional Communication. In *Combined 21th International Workshop on Expressiveness in Concurrency and 11th Workshop on Structural Operational Semantics*, Rome, Italie, Sept. 2014.
24. T. Given-Wilson and D. Gorla. Pattern matching and bisimulation. In *Coordination Models and Languages*, volume 7890 of *Lecture Notes in Computer Science*, pages 60–74. Springer Berlin Heidelberg, 2013.
25. T. Given-Wilson, D. Gorla, and B. Jay. Concurrent pattern calculus. In *Theoretical Computer Science*, volume 323 of *IFIP Advances in Information and Communication Technology*, pages 244–258. Springer Berlin Heidelberg, 2010.
26. T. Given-Wilson, D. Gorla, and B. Jay. A Concurrent Pattern Calculus. *Logical Methods in Computer Science*, 10(3), 2014.
27. T. Given-Wilson and A. Legay. On the Expressiveness of Joining. In *8th Interaction and Concurrency Experience (ICE 2015)*, Grenoble, France, June 2015.

28. D. Gorla. Comparing communication primitives via their relative expressive power. *Information and Computation*, 206(8):931–952, 2008.
29. D. Gorla. A taxonomy of process calculi for distribution and mobility. *Distributed Computing*, 23(4):273–299, 2010.
30. D. Gorla. Towards a unified approach to encodability and separation results for process calculi. *Information and Computation*, 208(9):1031–1053, 2010.
31. C. Haack and A. Jeffrey. Pattern-matching spi-calculus. *Information and Computation*, 204(8):1195–1263, Aug. 2006.
32. 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.
33. K. Honda and N. Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 152:437–486, 1995.
34. I. Lanese, J. A. Pérez, D. Sangiorgi, and A. Schmitt. On the expressiveness of polyadic and synchronous communication in higher-order process calculi. In *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP)*, volume 6199 of *LNCS*, pages 442–453. Springer, 2010.
35. 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.
36. C. Laneve and A. Vitale. The expressive power of synchronizations. In *Logic in Computer Science (LICS), 2010 25th Annual IEEE Symposium on*, pages 382–391. IEEE, 2010.
37. T. Margaria and B. Steffen. Middleware: just another level for orchestration. In *Proceedings of the Workshop on Middleware for Next-Generation Converged Networks and Applications, MNCNA 2007, Newport Beach, California, USA, November 26, 2007*, page 4. ACM, 2007.
38. R. Milner. The polyadic π-calculus: A tutorial. In *Logic and Algebra of Specification*, volume 94 of *Series F*. NATO ASI, Springer, 1993.
39. R. Milner. *Communicating and mobile systems - the Pi-calculus*. Cambridge University Press, 1999.
40. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I. *Information and Computation*, 100(1):1–40, Sept. 1992.
41. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, II. *Information and Computation*, 100(1):41–77, Sept. 1992.
42. S. Naujokat, A. Lamprecht, and B. Steffen. Tailoring process synthesis to domain characteristics. In I. Perseil, K. K. Breitman, and R. Sterritt, editors, *16th IEEE International Conference on Engineering of Complex Computer Systems, ICECCS 2011, Las Vegas, Nevada, USA, 27-29 April 2011*, pages 167–175. IEEE Computer Society, 2011.
43. U. Nestmann. On the expressive power of joint input. *Electronic Notes in Theoretical Computer Science*, 16(2):145–152, 1998.
44. J. Neubauer and B. Steffen. Plug-and-play higher-order process integration. *IEEE Computer*, 46(11):56–62, 2013.
45. J. Neubauer, B. Steffen, and T. Margaria. Higher-order process modeling: Product-lining, variability modeling and beyond. In A. Banerjee, O. Danvy, K. Doh, and J. Hatcliff, editors, *Semantics, Abstract Interpretation, and Reasoning about Programs: Essays Dedicated to David A. Schmidt on the Occasion of his Sixtieth Birthday, Manhattan, Kansas, USA, 19-20th September 2013.*, volume 129 of *EPTCS*, pages 259–283, 2013.

46. 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.
47. H. R. Nielson, F. Nielson, and R. Vigo. A calculus for quality. In *Formal Aspects of Component Software*, pages 188–204. Springer, 2012.
48. H. R. Nielson, F. Nielson, and R. Vigo. A calculus of quality for robustness against unreliable communication. *Journal of Logical and Algebraic Methods in Programming*, 84(5):611–639, 2015.
49. C. Palamidessi. Comparing the expressive power of the synchronous and asynchronous pi-calculi. *Mathematical. Structures in Comp. Sci.*, 13(5):685–719, Oct. 2003.
50. J. Parrow. Expressiveness of process algebras. *Electronic Notes in Theoretical Computer Science*, 209:173–186, Apr. 2008.
51. K. Peters. *Translational expressiveness: comparing process calculi using encodings*. PhD thesis, Technische Universität Berlin, Fakultät IV - Elektrotechnik und Informatik, Germany, 2012.
52. K. Peters, U. Nestmann, and U. Goltz. On distributability in process calculi. In *ESOP*, pages 310–329. Springer, 2013.
53. K. V. Prasad. A calculus of broadcasting systems. *Science of Computer Programming*, 25(2):285–327, 1995.
54. A. Schmitt and J. Stefani. 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*, pages 50–61, 2003.
55. B. Steffen. Unifying models. In R. Reischuk and M. Morvan, editors, *STACS 97, 14th Annual Symposium on Theoretical Aspects of Computer Science, Lübeck, Germany, February 27 - March 1, 1997, Proceedings*, volume 1200 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 1997.
56. C. Urban, S. Berghofer, and M. Norrish. Barendregt's variable convention in rule inductions. In *Automated Deduction – CADE-21*, volume 4603 of *Lecture Notes in Computer Science*, pages 35–50. Springer Berlin Heidelberg, 2007.
57. R. J. van Glabbeek. Musings on encodings and expressiveness. In *Proceedings of EXPRESS/SOS*, volume 89 of *EPTCS*, pages 81–98, 2012.
58. R. J. van Glabbeek. On the validity of encodings of the synchronous in the asynchronous π-calculus. *Inf. Process. Lett.*, 137:17–25, 2018.