Separation and Encodability in Mixed Choice Multiparty Sessions

Kirstin Peters
kirstin.peters@uni-a.de
Augsburg University
Augsburg, Germany

Nobuko Yoshida
nobuko.yoshida@cs.ox.ac.uk
University of Oxford
Oxford, United Kingdom

ABSTRACT

Multiparty session types (MP) are a type discipline for enforcing the structured, deadlock-free communication of concurrent and message-passing programs. Traditional MP have a limited form of choice in which alternative communication possibilities are offered by a single participant and selected by another. *Mixed choice multiparty session types* (MCMP) extend the choice construct to include both selections and offers in the same choice. This paper first proposes a general typing system for a mixed choice synchronous multiparty session calculus, and prove type soundness, communication safety, and deadlock-freedom.

Next we compare expressiveness of nine subcalcli of MCMP-calculus by examining their *encodability* (there exists a *good* encoding from one to another) and *separation* (there exists *no* good encoding from one calculus to another). We prove 8 new encodablity results and 20 new separation results. In summary, MCMP is strictly more expressive than classical multiparty sessions (MP) in [19] and mixed choice in mixed sessions in [8]. This contrasts to the results proven in [8, 51] where mixed sessions [8] do not add any expressiveness to non-mixed fundamental sessions in [65], shedding a light on expressiveness of multiparty mixed choice.

CCS CONCEPTS

• **Theory of computation** → **Process calculi**; **Parallel computing models**; **Distributed computing models**; • **Software and its engineering** → Concurrent programming languages.

KEYWORDS

Session Types, Mixed Choice, Concurrency, Pi-Calculus, Typing System, Protocols, Expressiveness

ACM Reference Format:
Kirstin Peters and Nobuko Yoshida. 2024. Separation and Encodability in Mixed Choice Multiparty Sessions. In *39th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '24), July 8–11, 2024, Tallinn, Estonia.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3661814.3662085

1 INTRODUCTION

Mixed choice, which allows non-deterministic choice between enabled inputs or outputs, has been used to represent mutual exclusion such as semaphores and concurrent scheduling algorithms in communicating systems [37]. Mixed choice offers the ability to rule out alternative options, i.e., discard inputs by selecting an output in the same choice, and vice versa. In concurrent and message-passing programming languages, there has been interest in including and efficiently implementing mixed choice, as exemplified by Concurrent ML [53, 54], and more recently by Go [22] (where choice is synchronous by default). In Esterel [3] and Facile [59], mixed choice is used as a key construct to lump all IO-synchronisations among parallel processes as a single choice.

This paper shows that an introduction of mixed choice in the behavioural type theory based on protocols, *multiparty session types* [29] (MP), not only offers more safe and deadlock-free processes, but also gains *expressiveness* [24, 50], which was not the case in binary (two-party) mixed sessions by Casal et al. [8].

Two Party Mixed Choice Sessions. Session types [28, 58, 68] govern communication behaviours of concurrent programs, ensuring *type error freedom* and *communication safety* (no mismatch between sent and expected data types). The shape of session types originated in Linear Logic [21, 26], where choices are *separated* (not mixed) and *binary* (between two participants). Such choices are either a sum of inputs (*external* choice) or of outputs (*internal* choice).

Using session process notation, we can write *external choice* and *internal choice* processes as:

$$P_{\&} \;=\; s?\ell_1(x_1).P_1 + s?\ell_2(x_2).P_2$$
$$Q_{\oplus} = \text{if } v \text{ then } s!\ell_1\langle v_1\rangle.Q_1 \text{ else } s!\ell_2\langle v_2\rangle.Q_2$$

Here $!$ denotes output, $?$ denotes input, and ℓ is a *label* used for matching. The input process $s?\ell_1(x_1).P_1$ indicates that the recipient at channel s expects to receive a value with label ℓ_1, after which it will continue with behaviour $P_1\{v_1/x_1\}$. The output $s!\ell_1\langle v_1\rangle.Q_1$ selects label ℓ_1, sending value v_1 and continues as Q_1.

A natural next step is the extension of separate binary choice to *mixed choice* (a mixture of synchronous input and outputs in a single choice), making it *non-deterministic*, e.g., a process waits for an input on label ℓ_1, or can *non-deterministically* select to output ℓ_2:

$$P_+ \;=\; s?\ell_1(x_1).P_1 + s!\ell_2\langle v_2\rangle.P_2 \quad Q_+ = s!\ell_1\langle v_1\rangle.Q_1 + s?\ell_2(x_2).Q_2$$

where a parallel composition of P_+ and Q_+ synchronises (reduces) either to $P_1\{v_1/x_1\} \mid Q_1$ or $P_2 \mid Q_2\{v_2/x_2\}$. This mixed choice behaviour follows the standard CCS semantics [37]: an output action always chooses a receive action at a choice in another process, and they are synchronised together. Recently, Casal et al. [8] studied this extension (denoted by CMV^+), and proved its type safety.

Expressiveness. A standard method to compare the abstract *expressive power* of process calculi is to analyse the existence of encodings, i.e., translations from one calculus into another. To rule out trivial or meaningless encodings, they are augmented with a set of quality criteria. An *encodability* result, i.e., an encoding from a *source calculus* into a *target calculus* that satisfies relevant criteria, shows that the target can mimic or *emulate* the behaviours of the

LICS '24, July 8–11, 2024, Tallinn, Estonia

ACM ISBN 979-8-4007-0660-8/24/07
https://doi.org/10.1145/3661814.3662085

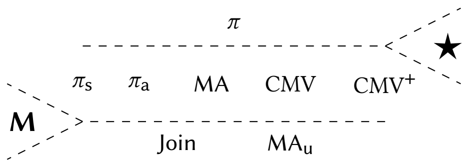


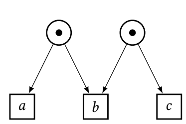
Figure 1: Hierarchy of π-like calculi [51]

source—the target is at least as expressive as the source. Conversely, a *separation* result, i.e., the proof that there is no encoding satisfying the considered criteria, shows that some behaviours of the source cannot be emulated by the target. The combination of encodability and separation results can build hierarchies of the analysed calculi, if these results are based on the same set of criteria. Encodings become stronger by using a stronger criteria, whereas separations become stronger by using weaker criteria. Sets of criteria that are well suited for encodability and separation were discussed e.g. in [16, 17, 23, 24, 42–46, 48, 50, 60, 63]. By following in particular [24, 48], we consider the criteria *compositionality* (the encoding is a function on the operators of the source), *name invariance* (the encoding treats all names of the source in the same way), sound and complete *operational correspondence* (the encoding preserves and reflects the behaviours of the source), *divergence reflection* (the target diverges only if the source diverges), *success-sensitiveness* (reachability of success is preserved and reflected), and *distributability preservation* (the target has the same degree of distribution as the source). Encodings that satisfy these criteria are denoted as *good* encodings.

Unfortunately, Peters and Yoshida [51] have shown that mixed sessions (CMV^+) proposed by Casal et al. [8] do *not* add any expressive power to the binary session calculus (denoted by CMV) [65] which has the standard branching and selection constructs (i.e., separate choices). This is perhaps surprising as it is against a landmark result on expressiveness by Palamidessi [42]—the π-calculus [39] with mixed choice is strictly more expressive than the π-calculus without choice. An open question is under which circumstances, mixed choice in session types is strictly more expressive.

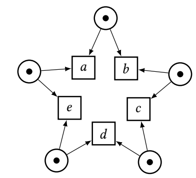
Expressiveness Hierarchy with Binary Mixed Sessions. To explain the above open problem more precisely, we first present a hierarchy of π-like calculi from [51] in Figure 1.

The hierarchy orders calculi along their ability to express the *synchronisation patterns* **M** and $\star$. An **M** (see [61, 62]), as visualised on the right, describes a Petri net that consists of two parallel transitions (a and c) and one transition (b) that is in conflict with both of the former. In other words, it describes a situation where either two parts of the net can proceed independently or they synchronise to perform a single transition together.



As stated by Glabbeek et al. [61, 62], a Petri net specification can be implemented in an asynchronous, fully distributed setting iff it does not contain a fully reachable pure **M**. They also present a description of a fully reachable pure **M** as conditions on a state in a step transition system, which allows us to directly use this pattern to reason about process calculi as shown in [44, 48, 49].

Further, Peters et al. [44, 48] introduce the synchronisation pattern $\star$, which is a chain of conflicting and distributable steps as they occur in an **M** that build a circle of odd length. As visualised on the right, there is e.g. one **M** consisting of the transitions a, b, and c with their corresponding two places. Another **M** is build by the transitions b, c, and d with their corresponding two places and so on.



The **M** captures only a small amount of synchronisation, whereas the $\star$ requires considerably more synchronisation in the calculus. *Asynchronously distributed calculi* that cannot express **M** nor $\star$ such as the Join-calculus from [15] are placed in the bottom layer of Figure 1. *Synchronously distributed calculi* that can express $\star$ and thus also **M** such as the π-calculus with mixed choice (π) are placed in the top layer. The middle layer consists of the calculi that can express **M** but not $\star$. The asynchronous π-calculus without choice [4, 27] (π_{a}), the π-calculus with separate choice [41] (π_{s}), and mobile ambients [7] (MA) are placed in the middle—unless it is restricted to unique ambient names (MA_{u}) [47]. That CMV and CMV^+ are also placed in the middle layer was proven in [51]. Hence, they are *strictly less expressive* than π. This completes Figure 1.

Mixed Choice Multiparty Session Types. In the presence of mixed choice, the extension from two parties to *more than two* parties gives us strictly more expressive power to session types and is placed in the top layer. This is because not only inputs and outputs but also *destinations of messages* represented by *participants* can be mixed in one single choice.

Consider three participants a, b, and c, and assume P_{a}, P_{b} and P_{c} below:

$$P_{\mathrm{a}} = (\mathrm{b}!\ell_1;\mathrm{c}!\ell_2) + \mathrm{b}?\ell_1 + \mathrm{c}?\ell_2 \quad P_{\mathrm{b}} = \mathrm{a}?\ell_1 + (\mathrm{a}!\ell_1;\mathrm{c}!\ell_2) + \mathrm{c}?\ell_2$$
$$P_{\mathrm{c}} = \mathrm{a}?\ell_2 + \mathrm{b}?\ell_2 + (\mathrm{a}!\ell_2;\mathrm{b}!\ell_2)$$

where we omit the payload and nil processes. In multiparty sessions, each process who plays role p is represented as $\mathrm{p} \triangleleft P$. Assuming, e.g., $\mathrm{b}!\ell_1$ in P_{a} matches with $\mathrm{a}?\ell_1$ in P_{b}, their parallel composition $(\mathrm{a}\triangleleft P_{\mathrm{a}} \mid \mathrm{b}\triangleleft P_{\mathrm{b}} \mid \mathrm{c}\triangleleft P_{\mathrm{c}})$ non-deterministically leads to several possible states such as $\mathrm{a} \triangleleft \mathrm{c}!\ell_2 \mid \mathrm{c} \triangleleft P_{\mathrm{c}}$ or $\mathrm{b} \triangleleft P_{\mathrm{b}} \mid \mathrm{c} \triangleleft \mathrm{b}!\ell_2$. We can observe that the choice behaviours of processes are *distributed* to and from two distinct participants.

Expressiveness in Mixed Choice Multiparty Sessions After introducing the typing system of MCMP, we study the expressive power introduced by multiparty mixed choice, i.e., determine in which layer of Figure 1, MCMP is situated. A precise answer to this question was not as simple as expected. To more clearly understand the causes of an increase in expressiveness, we first restrict our attention to a *single multiparty session* which has neither shared names, name passings, session delegation nor interleaved sessions, and consider subcalculi of MCMP. By limiting the form of the choices and number of participants, MCMP includes *9 subcalculi* (including itself). For example, we can define *separated choice from/to multiple participants* (SCMP), e.g.,

$$P_{\mathrm{SCMP}} = \mathrm{a}!\ell_1;\mathrm{b}!\ell_1; + \mathrm{a}!\ell_2;\mathrm{c}?\ell_2; + \mathrm{b}!\ell_3;\mathrm{c}?\ell_2; + \mathrm{b}!\ell_4;\mathrm{a}?\ell_2;$$

and *directed mixed choice* (DMP) where choice is mixed but from/to the same participant, e.g.,

$$P_{\mathrm{DMP}} = \mathrm{a}!\ell_1;\mathrm{b}!\ell_1; + \mathrm{a}?\ell_2\mathrm{c}?\ell_2; + \mathrm{a}!\ell_3;\mathrm{b}!\ell_3;$$

$$\mathsf{p} \triangleleft (\mathsf{q}?\ell(x).P + R_1) \mid \mathsf{q} \triangleleft (\mathsf{p}!\ell\langle v\rangle.Q + R_2) \mid M \longrightarrow \mathsf{p} \triangleleft P\{v/x\} \mid \mathsf{q} \triangleleft Q \mid M \quad [\Sigma]$$

$$\mathsf{p} \triangleleft \mathtt{if\ tt\ then}\ P_1\ \mathtt{else}\ P_2 \mid M \longrightarrow \mathsf{p} \triangleleft P_1 \mid M \quad [\text{If-tt}]$$

$$\mathsf{p} \triangleleft \mathtt{if\ ff\ then}\ P_1\ \mathtt{else}\ P_2 \mid M \longrightarrow \mathsf{p} \triangleleft P_2 \mid M \quad [\text{If-ff}]$$

$$M \equiv M_1 \longrightarrow M_2 \equiv M' \quad \Rightarrow \quad M \longrightarrow M' \quad [\text{Cong}]$$

$$P \equiv_\alpha Q \Rightarrow P \equiv Q \quad \mu \mathsf{X}.P \equiv P\{\mu \mathsf{X}.P/\mathsf{X}\} \quad M \equiv_\alpha M' \Rightarrow M \equiv M'$$

$$P \equiv Q \Rightarrow \mathsf{p} \triangleleft P \mid M \equiv \mathsf{p} \triangleleft Q \mid M \quad \mathsf{p} \triangleleft \mathbf{0} \mid M \equiv M$$

$$M \mid M' \equiv M' \mid M \quad (M \mid M') \mid M'' \equiv M \mid (M' \mid M'')$$

Figure 2: Reduction and structure congruence of MCMP

This makes the expressiveness analysis very subtle and comprehensive: we prove 20 new separation results and 8 new encodability results among 9 subcalculi of MCMP and 4 variants of CMV$^+$ [8]. Our observation is, by allowing a *combination of different communication patterns in a single choice*, we can cross the boundary of expressiveness. For example, MCMP is strictly more expressive than SCMP and can express the pattern ⋆; and SCMP is strictly more expressive than DMP and can express pattern **M**; but DMP and MP have the same expressive power and belong to the bottom.

The highlight is, in spite of the fact that CMV$^+$ enables creating an infinite number of interleaved binary mixed sessions with unrestricted names, MCMP (a single mixed choice multiparty session) cannot be emulated by CMV$^+$. In our results, properties such as *communication safety* and *deadlock-freedom* (if a session terminates, then all participants terminate, completing all actions) ensured by the MCMP typing system play the key role to prove the encodability results.

Contributions. (1) We first introduce the calculus, types and a typing system of *mixed choice multiparty sessions* (MCMP) which subsume the types and typability of the classical synchronous multiparty sessions (MP). We follow a general MP typing system in [56] which does not require global types. We then prove *communication safety* and *deadlock-freedom* of typable sessions. (2) We analyse the expressive power of mixed choice in MCMP showing that it is strictly more expressive than choice in MP. Therefore we use 8 subcalculi of MCMP deliberately chosen to explain the features of choice that raise expressiveness and compare to 4 variants of CMV$^+$. We prove 8 encodability and 20 separation results among the 13 calculi, introducing 8 new *good* encodings. The omitted proofs and more details can be found in [52].

2 MIXED CHOICE MULTIPARTY SESSION π-CALCULUS (MCMP-CALCULUS)

This section introduces the syntax and operational semantics for the *mixed choice multiparty session calculus* (MCMP-calculus), then define its subcalculi.

2.1 Syntax of MCMP-Calculus and its Family

The syntax of the MCMP-calculus follows the simplest synchronous multiparty session calculus [19, 66] (which consists of only a single multiparty session without session delegations), extended with nondeterministic mixed choices.

Definition 2.1 (syntax)**.** Assume a set of *participants* $\mathbb{P}$ ($\mathsf{p}, \mathsf{q}, \mathsf{r}, \dots$) and a set of *labels* $\mathbb{L}$ ($\ell, \ell', \dots$). *Values* contain either *variables* $(x, y, z, \dots)$ or constants; $\pi, \pi', \dots$ denote *prefixes*; $\mathsf{X}, \mathsf{Y}, \dots$ denote *process variables*; $P, Q, \dots$ denote *processes* and $M, M', \dots$ denote *multiparty sessions* (often called *sessions*).

$$\begin{array}{rll} v ::= & x, y, z, \dots \mid 1, 2, \dots \mid \mathtt{tt}, \mathtt{ff} & \text{(variables, numbers, booleans)} \\ \pi ::= & \mathsf{p}!\ell\langle v\rangle \mid \mathsf{p}?\ell(x) & \text{(output prefix, input prefix)} \\ P ::= & \mathbf{0} \mid \mathsf{X} \mid \mu\mathsf{X}.P & \text{(nil, proc var, recursion)} \\ & \mid \sum_{i\in I} \pi_i.P_i & \text{(mixed choice)} \\ & \mid \mathtt{if}\ v\ \mathtt{then}\ P\ \mathtt{else}\ P & \text{(conditional)} \\ M ::= & \mathsf{p} \triangleleft P \mid M \mid M & \text{(multiparty session, parallel)} \end{array}$$

Output prefix $\mathsf{p}!\ell\langle v\rangle$ which selects label ℓ at participant p by sending value v; and the matching input prefix, $\mathsf{p}?\ell(x)$ which receives a value with label ℓ from participant p and substitutes the value as variable x. We often omit values and variables ($\mathsf{p}!\ell$/$\mathsf{p}?\ell$) and labels for a singleton prefix ($\mathsf{p}!\langle v\rangle$/$\mathsf{p}?(x)$).

Process terms include a *nil*, $\mathbf{0}$, process variables and *recursions* $\mu\mathsf{X}.P$ where X is a binder. We assume P is guarded [19, § 2], i.e.,$\mu\mathsf{X}.\mathsf{X}$ is not allowed. The nondeterministic *mixed choice* $\sum_{i\in I} \pi_i.P_i$ ($I \neq \emptyset$) is a sum of prefixed processes. Conditional $\mathtt{if}\ v\ \mathtt{then}\ P_1\ \mathtt{else}\ P_2$ is standard. We assume standard α-conversion, capture-avoiding substitution, $P\{v/x\}$ and $P\{Q/\mathsf{X}\}$; and use function $\mathrm{fv}(P)$ to denote free variables in P. We often omit $\mathbf{0}$.

A *multiparty session* is a parallel composition of a participant process (denoted by $\mathsf{p} \triangleleft P$) where process P plays the role of participant p, and can interact with other processes playing other roles in M. We assume all participants in M are different.

We sometimes write $\pi_1.P_1 + \cdots + \pi_n.P_n$ for $\sum_{i\in I} \pi_i.P_i$ and $\Pi_{i\in I}\mathsf{p}_i \triangleleft P_i$ for $\mathsf{p}_1 \triangleleft P_1 \mid \cdots \mid \mathsf{p}_n \triangleleft P_n$ with $I = \{1, .., n\}$. We omit Π if I is a singleton, i.e., we write $\mathsf{p}_0 \triangleleft P_0$ for $\Pi_{i\in\{0\}}\mathsf{p}_i \triangleleft P_i$. Similarly for Σ. We often use $P+Q$ to denote $\sum_{i\in\{1,..,n\}} \pi_i.P_i$ with $P = \sum_{i\in\{1,..,k\}} \pi_i.P_i$ and $Q = \sum_{i\in\{k+1,..,n\}} \pi_i.P_i$, and assume commutativity and associativity of $+$. We also use the nested choices $\sum_{j\in J}\sum_{i\in I_j} \pi_{ij}.P_{ij}$ with $J = \{1..n\}$ to denote $\sum_{i\in I_1} \pi_{ij}.P_{ij} + \cdots \sum_{i\in I_n} \pi_{ij}.P_{ij}$.

We next define subcalculi of MCMP which are used in the paper.

Definition 2.2 (Family of MCMP)**.**

- **MSMP:** *Multiparty Mixed Separate Choice per Participant*, which is defined replacing $\sum_{i\in I} \pi_i.P_i$ by $\sum_{i\in I}\sum_{j\in J_i} \mathsf{p}_i!\ell_{ij}\langle v_{ij}\rangle.P_{ij} + \sum_{k\in K}\sum_{h\in H_k} \mathsf{q}_k?\ell_{kh}(v_{kh}).P_{kh}$ where $\{\mathsf{p}_i\}_{i\in I} \cap \{\mathsf{q}_k\}_{k\in K} = \emptyset$ and $\cup_{i\in I} J_i \cup \cup_{k\in K} H_k \neq \emptyset$, i.e., the choices to/from each participant is either outputs or inputs.
- **SCMP:** *Separate Choice Multiparty Session* where we have a sum of inputs $\sum_{i\in I} \mathsf{p}_i?\ell_i(x_i).P_i$ or a sum of outputs $\sum_{i\in I} \mathsf{p}_i!\ell_i\langle x_i\rangle.P_i$ with $I \neq \emptyset$.
- **DMP:** *Directed Mixed Choice Multiparty Session* where we have a mixed choice but from/to the same participant, i.e., a mixed choice $\sum_{i\in I} \mathsf{p}?\ell_i(x_i).P_i + \sum_{j\in J} \mathsf{p}!\ell_j\langle v_j\rangle.P_j$ with $I \cup J \neq \emptyset$.
- **SMP:** *Separate Choice Multiparty Session* where we have a sum of inputs $\sum_{i\in I} \mathsf{p}?\ell_i(x_i).P_i$ or a sum of outputs $\sum_{i\in I} \mathsf{p}!\ell_i\langle x_i\rangle.P_i$ with $I \neq \emptyset$ from the same participant p.
- **MP:** *Multiparty Session* in [19, 66] where we only have a sum of inputs from the same participant $\sum_{i\in I} \mathsf{p}?\ell_i(x_i).P_i$ with $I \neq \emptyset$ and a single selection $\mathsf{p}!\ell\langle v\rangle.P$.
- **MCBS:** *Mixed Choice Binary Session* where MCMP is limited to two parties only.

- **SCBS:** *Separate Choice Binary Session* where SCMP is limited to two parties only. The binary version of MSMP is syntactically same as SCBS.
- **BS:** *Binary Session* where MP is limited to two parties only.

Note that MCMP ⊃ MSMP ⊃ SCMP ⊃ SMP ⊃ MP; and MSMP ⊃ DMP ⊃ SMP; MCBS ⊃ MSBS (= SCBS) ⊃ BS; and DMP ⊃ MCBS; SCMP ⊃ SCBS; and MP ⊃ BS where ⊃ indicates a strict inclusion. Figure 5 shows these set inclusions.

Example 2.1 (A Family of MCMP)**.** We list examples of each calculi from syntactically larger ones. Consider:

MCMP	$P_1 = \mathsf{a}!\ell.\mathsf{b}?\ell + \mathsf{b}!\ell.\mathsf{c}!\ell + \mathsf{a}?\ell.\mathsf{a}?\ell$
MSMP	$P_2 = \mathsf{a}?\ell.\mathsf{b}!\ell + \mathsf{b}!\ell.\mathsf{c}?\ell + \mathsf{c}?\ell.\mathsf{a}!\ell$
SCMP	$P_3 = \mathsf{a}?\ell.\mathsf{b}!\ell + \mathsf{b}?\ell.\mathsf{c}?\ell + \mathsf{c}?\ell.\mathsf{a}?\ell$
DMP	$P_4 = \mathsf{a}!\ell_1.\mathsf{b}!\ell + \mathsf{a}!\ell_2.\mathsf{c}?\ell + \mathsf{a}?\ell.\mathsf{a}?\ell$
SMP	$P_5 = \mathsf{a}!\ell_1.\mathsf{b}!\ell + \mathsf{a}!\ell_2.\mathsf{c}?\ell + \mathsf{a}!\ell_3.\mathsf{a}?\ell$
MP	$P_6 = \mathsf{a}?\ell_1.\mathsf{b}!\ell + \mathsf{a}?\ell_2.\mathsf{c}?\ell + \mathsf{a}?\ell_3.\mathsf{a}!\ell$
MP	$P_7 = \mathsf{a}?\ell_1.\mathsf{b}!\ell_1 + \mathsf{a}?\ell_2.\mathsf{b}?\ell_2$
MCBS	$P_8 = \mathsf{a}?\ell_1.\mathsf{a}!\ell_1 + \mathsf{a}!\ell_2.\mathsf{a}?\ell_3$
SCBS	$P_9 = \mathsf{a}!\ell_1.\mathsf{a}!\ell_1 + \mathsf{a}!\ell_2.\mathsf{a}?\ell_2$
BS	$P_{10} = \mu X.(\mathsf{a}?\ell_1.\mathsf{a}!\ell_1 + \mathsf{a}?\ell_1.\mathsf{a}!\ell_2 + \mathsf{a}?\ell_2.\mathsf{a}?\ell_2.X)$
Untypable	$P_{11} = \mu X.(\mathsf{a}?\ell_1.\mathsf{a}?\ell_1 + \mathsf{a}?\ell_1.\mathsf{a}?\ell_2 + \mathsf{a}?\ell_2.\mathsf{a}?\ell_2.X)$

The table is read as follows: P_1 is MCMP but not MSMP; P_2 is MSMP (hence MCMP) but neither SCMP nor DMP; P_3 is SCMP but neither DMP nor SMP; P_4 is DMP but neither SCMP nor SMP; P_5 is SMP but not MP; P_6 and P_7 are MP but not MCBS; P_8 is MCBS but not MP; P_9 is SCBS but not MP; P_{10} is BS. All processes except P_{11} are typable with appropriate types by rules defined in Definition 4.1. Notice that P_{10} holds the two inputs from a with the same label ℓ_1 in the choice with the different outputs (ℓ_1 and ℓ_2) to a, but it will be typable using subtying and rule [TΣ] in Definition 4.1.

2.2 Reduction Semantics of MCMP-Calculus

The reduction and structural congruence rules are defined in Figure 2. Rule [Σ] represents mixed choice communication: it non-deterministically chooses a pair of an output and an input with the same label ℓ, and at the same time, value v is passed from the sender q to receiver p. Rule [Cong] closes under structural congruence.

As an example of reductions, let us define M as:

$$\mathsf{p} \triangleleft (\mathsf{q}?\ell_1(x).P_1 + \mathsf{q}!\ell_2\langle v_2\rangle.P_2) \mid \mathsf{q} \triangleleft (\mathsf{p}!\ell_1\langle v_1\rangle.Q_1 + \mathsf{p}?\ell_2(y).Q_2)$$

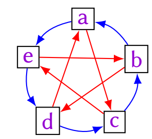
Then by [Σ], we have:

$$M \longrightarrow \mathsf{p} \triangleleft P_1\{v_1/x\} \mid \mathsf{q} \triangleleft Q_1 \text{ or } M \longrightarrow \mathsf{p} \triangleleft P_2 \mid \mathsf{q} \triangleleft Q_2\{v_2/y\}$$

We define a multistep as $\longrightarrow^* = (\equiv \cup \longrightarrow)^*$ (zero or more steps) and $\longrightarrow^+$ (one or more steps). Let $\longrightarrow^\omega$ denote an infinite sequence of steps. We call a term *convergent* if it does not have any infinite sequence of steps; and write $M \not\longrightarrow$ if there is no M' such that $M \longrightarrow M'$.

To showcase our theory, we start with a *leader election protocol* (which is used similar to [42] for a separation result in § 6).

Example 2.2 (Leader election protocol)**.** Consider a protocol that involves five participants (a, b, c, d, and e), interacting in two stages to elect a leader.



In the first stage (depicted by the blue circle) two times a process x asks a process y to become leader by sending y!*leader* that is accepted by y via x?*leader*. The respective two receivers of the first stage continue in the same way asking each other in a second stage (depicted as red star). The receiver in the second stage finally announces its election as leader by sending station!*elect* to external station, where Station = station $\triangleleft P_{\mathsf{station}}$ with $P_{\mathsf{station}} = \sum_{\mathsf{i} \in \{\mathsf{a},\mathsf{b},\mathsf{c},\mathsf{d},\mathsf{e}\}} \left(\mathsf{i}?elect.\, \sum_{\mathsf{i} \in \{\mathsf{a},\mathsf{b},\mathsf{c},\mathsf{d},\mathsf{e}\}} \mathsf{i}!del.\mathbf{0}\right)$.

$$\text{Election} = \mathsf{a} \triangleleft P_{\mathsf{a}} \mid \mathsf{b} \triangleleft P_{\mathsf{b}} \mid \mathsf{c} \triangleleft P_{\mathsf{c}} \mid \mathsf{d} \triangleleft P_{\mathsf{d}} \mid \mathsf{e} \triangleleft P_{\mathsf{e}}$$

$$\begin{aligned} P_{\mathsf{a}} = (&\mathsf{e}!leader.\mathbf{0} \\ &+ \mathsf{b}?leader.\,(\mathsf{c}!leader.\mathbf{0} + \mathsf{d}?leader.\mathsf{station}!elect.\mathbf{0}) \\ &+ \mathsf{station}?del.\mathbf{0}) \end{aligned}$$

$$\sigma = [\mathsf{a} \mapsto \mathsf{b}, \mathsf{b} \mapsto \mathsf{c}, \mathsf{c} \mapsto \mathsf{d}, \mathsf{d} \mapsto \mathsf{e}, \mathsf{e} \mapsto \mathsf{a}]$$

$$P_{\mathsf{b}} = P_{\mathsf{a}}\sigma \quad P_{\mathsf{c}} = P_{\mathsf{b}}\sigma \quad P_{\mathsf{d}} = P_{\mathsf{c}}\sigma \quad P_{\mathsf{e}} = P_{\mathsf{d}}\sigma$$

The complete multiparty session is Election | Station. The participants in Election are symmetric w.r.t. σ, for instance $P_{\mathsf{d}} = P_{\mathsf{c}}\sigma = (\mathsf{c}!leader.\mathbf{0} + \mathsf{e}?leader.\,(\mathsf{a}!leader.\mathbf{0} + \mathsf{b}?leader.\mathsf{station}!elect.\mathbf{0}) + \mathsf{station}?del.\mathbf{0})$. After the winner is announced to station, it garbage collects the process z that did not participate in the first stage by sending z!*del* and then terminates itself.

3 MIXED CHOICE SYNCHRONOUS MULTIPARTY SESSION TYPES

This section introduces the syntax of MCMP types, which describe the interactions between *participants* at the end-point level. The syntax is based on [19], extending to mixed choice.

Definition 3.1 (MCMP types and local contexts)**.** Assume a set of participants $\mathbb{P}$ ($\mathsf{p}, \mathsf{q}, \mathsf{r}, \ldots$) and a set of labels $\mathbb{L}$ ($\ell, \ell', \ldots$). The set of *local types*, $\mathbb{T}$ with $T \in \mathbb{T}$, are defined as:

U	$::= \mathsf{nat} \mid \mathsf{bool}$	(numeric, boolean)
$\dagger$	$::= ! \mid ?$	(message send, message receive)
T	$::= \mathsf{end} \mid \mathsf{t} \mid \mu\mathsf{t}.T$	(inaction, recursion variable and type)
	$\mid \sum_{i \in I} \mathsf{p}_i \dagger_i \ell_i \langle U_i \rangle ; T_i$	(mixed choice)
Δ	$::= \emptyset \mid \Delta, \mathsf{p}{:}T$	(local contexts)

Payload type U ranges over ground types (nat, bool). Termination is represented by end. Recursive types are $\mu\mathsf{t}.T$, with t as the recursive variable. We assume standard capture-avoiding substitution and assume that recursive types are *guarded*, e.g., for type $\mu\mathsf{t}_1.\cdots\mu\mathsf{t}_n.\mathsf{t}$ is not allowed. We take the equirecursive view, i.e. $\mu\mathsf{t}.T$ is identified with $T\{\mu\mathsf{t}.T/\mathsf{t}\}$, see [19, Notation 3.5]. $\mathrm{ftv}(T)$ denotes a set of free type variables in T and T is *closed* if $\mathrm{ftv}(T) = \emptyset$.

Mixed choice type $\sum_{i \in I} \mathsf{p}_i \dagger_i \ell_i \langle U_i \rangle ; T_i$ enables choice between a non-empty collection of input or output local types, $\mathsf{p}_i \dagger_i \ell_i \langle U_i \rangle ; T_i$. Each type denotes sending (!) or receiving (?) a message of label ℓ_i with a payload type U_i from or to some different participant p_i and continues as T_i. We often abbreviate input and output types as $\mathsf{p}\dagger\langle U \rangle ; T$ or $\mathsf{p}\dagger\ell ; T$ if a label or payload is not important; and omit trailing end types. Similarly with processes, we omit $\sum$ when I is a singleton, $\sum_{i \in \{1\}} \mathsf{p}_i \dagger \ell_i \langle U_i \rangle ; T_i = \mathsf{p}_1 \dagger \ell_1 \langle U_1 \rangle ; T_1$. We abbreviate $\mathsf{p}_1 \dagger \ell_1 \langle U_1 \rangle ; T_1 + \cdots + \mathsf{p}_n \dagger \ell_n \langle U_n \rangle ; T_n$ to denote $\sum_{i \in \{1,..,n\}} \mathsf{p}_i \dagger \ell_i \langle U_i \rangle ; T_i$ and write $\sum_{j \in J} \sum_{i \in I_j} \mathsf{p}_{ij} \dagger \ell_{ij} \langle U_{ij} \rangle ; T_{ij}$ with $J = \{1..n\}$ for

$\sum_{i\in I_1}\mathtt{p}_{i1}\dagger\ell_{i1}\langle U_{i1}\rangle;T_{i1}+\cdots+\sum_{i\in I_n}\mathtt{p}_{in}\dagger\ell_{in}\langle U_{in}\rangle;T_{in}$. A choice is commutative and associative with $+$. Function $\mathrm{pt}(T)$ returns a set of *participants* in T defined as: $\mathrm{pt}(\mathtt{end}) = \mathrm{pt}(\mathtt{t}) = \emptyset$; $\mathrm{pt}(\mu\mathtt{t}.T) = \mathrm{pt}(T)$; and $\mathrm{pt}(\sum_{i\in I}\mathtt{p}_i\dagger_i\ell_i\langle U_i\rangle;T_i) = \{\mathtt{p}_i\}_{i\in I}\cup\bigcup_{i\in I}\mathrm{pt}(T_i)$; and function $\mathrm{pre}(T)$ returns *prefixes* of T defined as: $\mathrm{pre}(\mathtt{end}) = \mathrm{pre}(\mathtt{t}) = \emptyset$; $\mathrm{pre}(\mu\mathtt{t}.T) = \mathrm{pre}(T)$; and $\mathrm{pre}(\sum_{i\in I}\mathtt{p}_i\dagger_i\ell_i\langle U_i\rangle;T_i) = \{\mathtt{p}_i\dagger_i\}_{i\in I}$.

We define the *duality* function as $\overline{!} = ?$ and $\overline{?} = !$.

Type $\sum_{i\in I}\mathtt{p}_i\dagger_i\ell_i\langle U_i\rangle;T_i$ is *well-formed* if:

$$[\text{L-}\ell]\qquad (\forall i{\neq}j{\in}I.\ \mathtt{p}_i\dagger_i = \mathtt{p}_j\dagger_j \implies \ell_i\neq\ell_j)$$

i.e., for a mixed choice, any matching choice prefixes must have distinguishable labels. We write $\vdash T$ if all choices in T are well-formed. *Hereafter we assume all types are well-formed.* Note that typable mixed choice processes do not have the same well-formedness requirement as local types, allowing non-deterministic process choice using the same label (see P_{10} in Example 2.1 and Example 4.1).

A *local context* Δ abstracts the behaviour of a set of participants where we assume for all $\mathtt{p}\in\mathrm{dom}(\Delta)$, $\mathrm{ftv}(\Delta)=\emptyset$.

3.1 Subtyping of MCMP

We define the subtyping relation for mixed choice local types, which subsumes the standard subtyping [6, 11, 14].

Definition 3.2 (Subtyping). The subtyping relation $\leqslant$ is *coinductively* defined by:

$$\mathtt{end}\leqslant\mathtt{end}\ [\text{SEnd}]$$

$$\frac{\forall i{\in}I, T_i\leqslant T_i'}{\sum_{i\in I}\mathtt{p}!\ell_i\langle U_i\rangle;T_i\leqslant\sum_{i\in I\cup J}\mathtt{p}!\ell_i\langle U_i\rangle;T_i'}\ [\text{SSl}]$$

$$\frac{\forall i{\in}I, T_i\leqslant T_i'}{\sum_{i\in I\cup J}\mathtt{p}?\ell_i\langle U_i\rangle;T_i\leqslant\sum_{i\in I}\mathtt{p}?\ell_i\langle U_i\rangle;T_i'}\ [\text{SBr}]$$

$$\frac{\forall k{\in}K, T_k\leqslant T_k'\quad \forall i{\neq}j{\in}K, \mathrm{pre}(T_i)\cap\mathrm{pre}(T_j)=\emptyset}{\sum_{k\in K}T_k\leqslant\sum_{k\in K}T_k'}\ [\text{S}\Sigma]$$

$$\frac{T_1\{\mu\mathtt{t}.T_1/\mathtt{t}\}\leqslant T_2}{\mu\mathtt{t}.T_1\leqslant T_2}\ [\text{S}\mu\text{L}]\qquad \frac{T_1\leqslant T_2\{\mu\mathtt{t}.T_2/\mathtt{t}\}}{T_1\leqslant\mu\mathtt{t}.T_2}\ [\text{S}\mu\text{R}]$$

We write $\Delta_1\leqslant\Delta_2$ iff for all $\mathtt{p}\in\mathrm{dom}(\Delta_1)\cap\mathrm{dom}(\Delta_2)$, $\Delta_1(\mathtt{p})\leqslant\Delta_2(\mathtt{p})$ and for all $\mathtt{p}\in\mathrm{dom}(\Delta_1)\setminus\mathrm{dom}(\Delta_2)$ and $\mathtt{q}\in\mathrm{dom}(\Delta_2)\setminus\mathrm{dom}(\Delta_1)$, $\Delta_1(\mathtt{p})=\Delta_2(\mathtt{q})=\mathtt{end}$.

The above subtyping rules without [SΣ] are the standard from [6, 11, 14]: a smaller type has smaller internal choices [SSl]; larger external sums are smaller [SBr]. The subtyping of a mixed choice which combines selection and branching (the premise is inferred by either [SSl] or [SBr]) is invariant [SΣ]. The side condition given by $\mathrm{pre}(T)$ in [SΣ] ensures no overlap with [SSl] and [SBr].

Example 3.1 (Mixed choice subtyping). The mixed choice subtype judgement is given by [SΣ]. This rule partitions each mixed-choice term into a sum of non-mixed choices, i.e., $(\mathtt{p}?\ell_1{+}\mathtt{p}?\ell_2)+(\mathtt{p}!\ell_3)\leqslant(\mathtt{p}?\ell_1)+(\mathtt{p}!\ell_3{+}\mathtt{p}!\ell_4)$. Standard subtyping rules ([SSl] and [SBr]) are then applied pair-wise to each non-mixed choice.

$$\frac{\dfrac{\vdots}{\mathtt{p}?\ell_1;T_1{+}\mathtt{p}?\ell_2;T_2\leqslant\mathtt{p}?\ell_1;T_1}\ [\text{SBr}]\qquad\dfrac{\vdots}{\mathtt{p}!\ell_3;T_3\leqslant\mathtt{p}!\ell_3;T_3{+}\mathtt{p}!\ell_4;T_4}\ [\text{SSl}]}{\mathtt{p}?\ell_1;T_1{+}\mathtt{p}?\ell_2;T_2{+}\mathtt{p}!\ell_3;T_3\leqslant\mathtt{p}?\ell_1;T_1{+}\mathtt{p}!\ell_3;T_3{+}\mathtt{p}!\ell_4;T_4}\ [\text{S}\Sigma]$$

We can also mix different participants such as: $(\mathtt{p}?\ell_1{+}\mathtt{p}?\ell_2)+(\mathtt{q}!\ell_3)\leqslant(\mathtt{p}?\ell_1)+(\mathtt{q}!\ell_3{+}\mathtt{q}!\ell_4)$.

$$\sum_{i\in I}\mathtt{p}_i\dagger_i\ell_i\langle U_i\rangle;T_i\xrightarrow{\mathtt{p}_j\mathtt{q}_j\dagger_j\ell_j\langle U_j\rangle}T_j\ \ (j\in I)\quad[\text{Sum}]$$

$$\frac{\mathtt{p}: T\{\mu\mathtt{t}.T/\mathtt{t}\}\xrightarrow{\lambda}\mathtt{p}: T'}{\mathtt{p}: \mu\mathtt{t}.T\xrightarrow{\lambda}\mathtt{p}: T'}\ [\text{L}\mu]$$

$$\frac{\mathtt{p}: T_1\xrightarrow{\mathtt{pq}!\ell\langle U\rangle}\mathtt{p}: T_1'\quad \mathtt{q}: T_2\xrightarrow{\mathtt{qp}?\ell\langle U\rangle}\mathtt{q}: T_2'}{\Delta,\mathtt{p}: T_1,\mathtt{q}: T_2\xrightarrow{\mathtt{pq}:\ell\langle U\rangle}\Delta,\mathtt{p}: T_1',\mathtt{q}: T_2'}\ [\text{RPass}]$$

Figure 3: Labelled transition systems of types and contexts

Proposition 3.1 (Subtyping). *Suppose $\vdash T_i$ ($i=1,2$). (a) $T_1\leqslant T_2$ is a preorder; and (b) $\Delta_1\leqslant\Delta_2$ is a preorder.*

Proof. Induction on delivation of $T_1\leqslant T_2$. □

Remark 3.1 (Subtyping). Our subtyping relation subsumes the standard branching-selection subtyping relation by omitting [SΣ]. This inclusion is important for proving the expressiveness results. If we replace [SΣ], [SBr] and [SSl] by the following simpler rule [SSet]:

$$\frac{\forall i{\in}I, T_i\leqslant T_i'}{\sum_{i\in I}\mathtt{p}_i\dagger_i\ell_i\langle U_i\rangle;T_i\leqslant\sum_{i\in I\cup J}\mathtt{p}_i\dagger_i\ell_i\langle U_i\rangle;T_i'}\ [\text{SSet}]$$

then Lemma 3.1(2) does *not* hold. The lemma is crucial for proving *deadlock-freedom* of the typed processes (Theorem 5.2), see Remark 3.2(2).

3.2 Labelled Transition System (LTS) of Types and Contexts

This subsection defines the LTS of types and contexts. The behavioural properties (safety and deadlock-freedom) of local contexts defined by the LTS are used to prove the main theorems of typed processes in § 5.

- **Local Types:** The set of actions of local types is defined as $\mathrm{Act}_L=\{\mathtt{pq}!\ell\langle U\rangle,\mathtt{pq}?\ell\langle U\rangle\mid\mathtt{p},\mathtt{q}\in\mathbb{P},\ell\in\mathbb{L}\}$ with $\lambda\in\mathrm{Act}_L$; and the set of local contexts is defined as $\mathbb{R}=\{\Delta\}$. The transition relation $\xrightarrow{\lambda}\subseteq\mathbb{R}\times\mathrm{Act}_L\times\mathbb{R}$ is defined by [Sum] and [Lμ] in Figure 3.
- **Local Contexts:** The set of actions of local contexts is defined as $\mathrm{Act}=\{\mathtt{pq}:\ell\langle U\rangle\mid\mathtt{p},\mathtt{q}\in\mathbb{P},\ell\in\mathbb{L}\}$. The transition relation $\xrightarrow{\mathtt{pq}:\ell\langle U\rangle}\subseteq\mathcal{P}(\mathbb{R})\times\mathrm{Act}\times\mathcal{P}(\mathbb{R})$ is defined by [RPass] in Figure 3.

Notice that the LTS is defined between closed types. The semantics for local types and contexts follow the standard concurrency semantics [37]. Label $\mathtt{pq}!\ell\langle U\rangle$ denotes that participant $\mathtt{p}$ may send a message with label ℓ of type U to participant $\mathtt{q}$. Dually, label $\mathtt{pq}?\ell\langle U\rangle$ denotes that participant $\mathtt{p}$ may receive a message with label ℓ of type U from participant $\mathtt{q}$. Rule [Sum] chooses one of choices and rule [Lμ] is standard. Rule [RPass] states that if two roles exhibit dual local labels, they can synchronise and perform action $\mathtt{pq}:\ell\langle U\rangle$.

3.3 Properties for Mixed Choice Multiparty Session Types

Definition 3.3 (Local context reductions). We write $\Delta\longrightarrow\Delta'$ if $\Delta\xrightarrow{\mathtt{pq}:\ell\langle U\rangle}\Delta'$; $\Delta\longrightarrow^+\Delta'$ for its transitive closure; $\Delta\longrightarrow^*\Delta'$ for either $\Delta=\Delta'$ or $\Delta\longrightarrow^+\Delta'$; and $\Delta\xrightarrow{\mathtt{pq}:\ell\langle U\rangle}$ if $\exists\Delta'.\ \Delta\xrightarrow{\mathtt{pq}:\ell\langle U\rangle}\Delta'$.

To prove the subject reduction theorem, we first introduce the *safety* property from [56, Definition 4.1]. It states that there is no communication mismatch. This property is used to prove the subject reduction theorem and communication safety.

Definition 3.4 (Safety property). Co-inductive property φ is a *safety* property of local context Δ if and only if for all $\{\mathsf{p}: T_1, \mathsf{q}: T_2\} \subseteq \Delta \in \varphi$, if $\mathsf{p}: T_1 \xrightarrow{\mathsf{pq}!\ell\langle U\rangle}$ and $\mathsf{q}: T_2 \xrightarrow{\mathsf{qp}?\ell'\langle U'\rangle}$, then $\Delta \xrightarrow{\mathsf{pq}:\ell\langle U\rangle} \Delta'$ and $\Delta' \in \varphi$. The largest safety property is a union of all safety properties. We say Δ is *safe* and write $\mathsf{safe}(\Delta)$ if $\Delta \in \varphi$ and φ is a safety property.

The safety property says that if the output and input actions are ready each other, they can synchronise with the label provided by the output ℓ and they have the same payload types ($U = U'$) (note that the base types do not have subtyping), and it is preserved after a step. Notice that the safety is *not* symmetric–the label ℓ and type U of the selection are always taken, while *some* label of a branching needs to be reducible.

Next we define *deadlock-freedom* following [56, Figure 5(2)]. It states that if typing context Δ terminates, all participants typed by Δ terminate as nil processes (typed by end).

Definition 3.5 (Deadlock-freedom). Local context Δ is *deadlock-free* if $\Delta \longrightarrow^* \Delta' \not\longrightarrow$, then $\forall \mathsf{p} \in \mathsf{dom}(\Delta')\,.\,\Delta'(\mathsf{p}) = \mathsf{end}$. We denote $\mathsf{dfree}(\Delta)$ if Δ is deadlock-free.

The following is the key lemma to ensure communication safety and deadlock-freedom of typed sessions.

Lemma 3.1 (Subtyping and properties).

(1) If $\Delta \leqslant \Delta'$ and $\mathsf{safe}(\Delta')$, then (a) $\mathsf{safe}(\Delta)$. (b) If $\Delta \longrightarrow \Delta''$, then there exists Δ''' such that $\Delta' \longrightarrow \Delta'''$ and $\Delta'' \leqslant \Delta'''$ and $\mathsf{safe}(\Delta''')$.

(2) If $\Delta \leqslant \Delta'$, and $\mathsf{safe}(\Delta')$ and $\mathsf{dfree}(\Delta')$, then (a) $\mathsf{dfree}(\Delta)$. (b) If $\Delta \longrightarrow \Delta''$, then there exists Δ''' such that $\Delta' \longrightarrow \Delta'''$ and $\Delta'' \leqslant \Delta'''$ and $\mathsf{dfree}(\Delta''')$.

(3) Checking $\mathsf{safe}(\Delta)$ and $\mathsf{dfree}(\Delta)$ is decidable.

Proof. (1,2) Induction on Δ and $\Delta \longrightarrow \Delta'$; and (3) Similar with [55, Appendix K]. □

Remark 3.2 (Deadlock-freedom). (1) Let $\Delta = \mathsf{p} : \mathsf{q}!\ell_1{+}\mathsf{q}!\ell_2, \mathsf{q} : \mathsf{p}?\ell_1$. Then we can check $\mathsf{dfree}(\Delta)$ since the only possible transition is $\Delta \xrightarrow{\mathsf{pq}:\ell_1} \Delta'$. But, for $\Delta' = \mathsf{p} : \mathsf{q}!\ell_2, \mathsf{q} : \mathsf{p}?\ell_1$ such that $\Delta' \leqslant \Delta$, we have $\neg\mathsf{dfree}(\Delta')$. Hence without the $\mathsf{safe}(\Delta)$ assumption, $\mathsf{dfree}(\Delta)$ is not preserved by subtyping.
(2) Assume $\Delta_2 = \mathsf{p} : \mathsf{q}!\ell_1{+}\mathsf{q}!\ell_2, \mathsf{q} : \mathsf{p}?\ell_1{+}\mathsf{p}?\ell_2$. Then $\mathsf{safe}(\Delta_2)$ and $\mathsf{dfree}(\Delta_2)$. Suppose we apply a wrong subtyping rule [SSet] in Remark 3.1 to obtain $\Delta_3 = \mathsf{p} : \mathsf{q}!\ell_1, \mathsf{q} : \mathsf{p}?\ell_2$. Then $\neg\mathsf{safe}(\Delta_3)$ and $\neg\mathsf{dfree}(\Delta_3)$, i.e. neither safety nor deadlock-freedom is preserved.

Proposition 3.2 (Local context properties). *(1)* $\mathsf{dfree}(\Delta) \not\Longrightarrow \mathsf{safe}(\Delta)$*; and (2)* $\mathsf{safe}(\Delta) \not\Longrightarrow \mathsf{dfree}(\Delta)$.

Proof. (1) Let $\Delta_3 = \mathsf{p} : \mu\mathsf{t}.\mathsf{q}!\ell_1;\mathsf{t}, \mathsf{q} : \mu\mathsf{t}.\mathsf{p}?\ell_1;\mathsf{t}, \mathsf{r} : \mathsf{r}'!\ell_2\langle\mathsf{bool}\rangle, \mathsf{r}' : \mathsf{r}?\ell_2\langle\mathsf{nat}\rangle$. Then Δ_3 is deadlock-free because $\Delta_3 \xrightarrow{\mathsf{pq}:\ell_1} \Delta_3$. But Δ_3 is not safe because $\mathsf{nat} \neq \mathsf{bool}$. (2) Let $\Delta_2 = \mathsf{p} : \mathsf{q}!\ell$. Then Δ_2 is vacuously safe following Definition 3.4, but not deadlock-free because $\Delta_2 \not\longrightarrow$ and $\Delta_2(\mathsf{p}) \neq \mathsf{end}$. □

$$\Gamma, x{:}\mathsf{nat} \vdash x \triangleright \mathsf{nat}\ \text{[Gr]}$$

$$\Gamma \vdash 1, 2, .. \triangleright \mathsf{nat}\ \text{[Nat]} \qquad \Gamma \vdash \mathtt{tt}, \mathtt{ff} \triangleright \mathsf{bool}\ \text{[Bool]}$$

$$\Gamma, \mathsf{X} : T \vdash \mathsf{X} \triangleright T\ \text{[TVar]} \qquad \Gamma \vdash \mathbf{0} \triangleright \mathsf{end}\ \text{[TInact]}$$

$$\frac{\Gamma \vdash P \triangleright T \quad \Gamma \vdash v \triangleright U}{\Gamma \vdash \mathsf{p}!\ell\langle v\rangle.P \triangleright \mathsf{p}!\ell\langle U\rangle;T}\ \text{[TSnd]}$$

$$\frac{\Gamma, x : U \vdash P \triangleright T}{\Gamma \vdash \mathsf{p}?\ell(x).P \triangleright \mathsf{p}?\ell\langle U\rangle;T}\ \text{[TRcv]}$$

$$\frac{\Gamma \vdash v \triangleright \mathsf{bool} \quad \Gamma \vdash P_i \triangleright T \quad i \in \{1,2\}}{\Gamma \vdash \mathtt{if}\ v\ \mathtt{then}\ P_1\ \mathtt{else}\ P_2 \triangleright T}\ \text{[TIf]}$$

$$\frac{\forall i \in I, \forall j \in J_i.\ (\Gamma \vdash \pi_i.P_j \triangleright \mathsf{p}_i\dagger_i\ell_i\langle U_i\rangle;T_i)}{\Gamma \vdash \sum_{i\in I}\sum_{j\in J_i}\pi_i.P_j \triangleright \sum_{i\in I}\mathsf{p}_i\dagger_i\ell_i\langle U_i\rangle;T_i}\ \text{[T}\Sigma\text{]}$$

$$\frac{\Gamma, \mathsf{X} : T \vdash P \triangleright T}{\Gamma \vdash \mu\mathsf{X}.P \triangleright T}\ \text{[TRec]} \qquad \frac{\Gamma \vdash P \triangleright T \quad T \leqslant T'}{\Gamma \vdash P \triangleright T'}\ \text{[T}\leqslant\text{]}$$

$$\frac{\forall i \in I\ \ \vdash P_i \triangleright T_i \quad \mathsf{safe}(\{\mathsf{p}_i{:}T_i\}_{i\in I})}{\vdash \Pi_{i\in I}\ \mathsf{p}_i \triangleleft P_i \triangleright \{\mathsf{p}_i{:}T_i\}_{i\in I}}\ \text{[TSess+]}$$

Figure 4: Typing rules for MCMP-calculus.

4 TYPING SYSTEM OF MCMP-CALCULUS

We introduce the typing system, extending the system in [19] and highlight the three key rules which differ from [19].

Definition 4.1 (Typing system). We define a *shared context* as:

$$\Gamma ::= \emptyset \ \mid\ \Gamma, \mathsf{X} : T \ \mid\ \Gamma, x{:}\mathsf{nat} \ \mid\ \Gamma, x{:}\mathsf{bool}$$

We write Γ, Γ' if $\mathsf{dom}(\Gamma) \cap \mathsf{dom}(\Gamma') = \emptyset$ and use the notation $\Gamma(x)$ or $\Gamma(\mathsf{X})$ to denote a constant type of x and local type of X. Figure 4 defines the three judgements:

$$\Gamma \vdash v \triangleright U \qquad \Gamma \vdash P \triangleright T \qquad \vdash M \triangleright \Delta$$

where judgement $\Gamma \vdash v \triangleright U$ is read as 'context Γ types value v with type U' and judgement $\Gamma \vdash P \triangleright T$ as 'context Γ types process P with local type T'. Judgement $\vdash M \triangleright \Delta$ is read as 'multiparty session M is typed with local context Δ'.

We highlight three rules in Figure 4. These rules are different from the rules for MP in [19]. Rule [TInact] assigns a nil by end. Rules [TSnd] and [TRcv] type the output and input with the send and input types, respectively. Notice that we usually type the input branching instead of a single receiver as given by [TRcv], but our choice rule allows to type both input and output are mixed, subsuming the standard input branching rule ([T-INPUT-CHOICE] in [19]). Typing recursion and conditional is standard by [TVar], [TRec] and [TIf]. Rule [T⩽] is a subsumption rule.

Typing of mixed choice is given by [TΣ]. We motivate the double sum in the conclusion by considering Example 4.1(1) below, in which two branches ($\mathsf{q}!\ell_1.Q$ and $\mathsf{q}!\ell_1.R$) have a common prefix and both Q and R are typed by the same local type, while the other two branches are typed independently. The outer sum, then, groups similarly prefixed branches ($R_i = \sum_{j\in J_i}\pi_i.P_j$); each branch in R_i is then typed by a common $\mathsf{p}_i\dagger_i\ell_i\langle U_i\rangle;T_i$; finally, the full term $\sum_{i\in I} R_i$ is typed as $\sum_{i\in I}\mathsf{p}_i\dagger_i\ell_i\langle U_i\rangle;T_i$.

Multiparty session M is typed by rule [TSess+] which assumes a local context which consists of types of all participants is safe [56] to guarantee the subject reduction.

Example 4.1 (Typed and untyped mixed choice processes)**.**

(1) We explain how we can type a process with duplicated labels with a single label type. Consider:

$$P \;=\; \mathsf{q}!\ell_1.Q + \mathsf{q}!\ell_1.R + \mathsf{q}?\ell_2.\mathbf{0} + \mathsf{r}?\ell_2.\mathbf{0}$$

Assuming Q and R have the same type T, we first type:

$$\vdash \mathsf{q}!\ell_1.Q + \mathsf{q}!\ell_1.R \triangleright \mathsf{q}!\ell_1;T \qquad \vdash \mathsf{q}?\ell_2.\mathbf{0} \triangleright \mathsf{q}?\ell_2 \qquad \vdash \mathsf{r}?\ell_2.\mathbf{0} \triangleright \mathsf{r}?\ell_2$$

and then use rule [TΣ] to obtain: $\emptyset \vdash P \triangleright \mathsf{q}!\ell_1;T{+}\mathsf{q}?\ell_2{+}\mathsf{r}?\ell_2$. Note that a single local type can type two choices where Q and R might have different behaviours but have the same type:

$$Q = \mathsf{q}!\ell'\langle 5\rangle.\mathbf{0} \;\text{ and }\; R = \mathsf{q}!\ell'\langle 105\rangle.\mathbf{0}$$

with $\vdash Q \triangleright \mathsf{q}!\ell'\langle\mathsf{nat}\rangle$ and $\vdash R \triangleright \mathsf{q}!\ell'\langle\mathsf{nat}\rangle$.

(2) A combination of [TΣ] and subsumption [T⩽] makes a process with duplicated labels with different output labels typable. Consider:

$$Q_1 = \mathsf{a}!\ell_1\langle 5\rangle.\mathbf{0} \text{ and } Q_2 = \mathsf{a}!\ell_2\langle \mathtt{tt}\rangle.\mathbf{0}$$

and let $T = \mathsf{a}!\ell_1\langle\mathsf{nat}\rangle{+}\mathsf{a}!\ell_2\langle\mathsf{bool}\rangle$. Then $\emptyset \vdash Q_1 \triangleright T$ and $\emptyset \vdash Q_2 \triangleright T$ by [TSnd] and [T⩽]. Let $R = \mathsf{a}?\ell(x).Q_1 + \mathsf{a}?\ell(x).Q_2$. Then by [TΣ], R is typable. Similarly, P_{10} in Example 2.1 is typable, but P_{11} is not typable as the input choice types are co-variant.

(3) Session M_1 is typable but its reduction causes a *deadlock*:

$$M_1 = \mathsf{p} \triangleleft (\mathsf{q}!\ell.\mathbf{0}{+}\mathsf{r}!\ell.\mathbf{0}) \mid \mathsf{q} \triangleleft \mathsf{p}?\ell.\mathbf{0} \mid \mathsf{r} \triangleleft \mathsf{p}?\ell.\mathbf{0}$$

(4) Session M_2 causes a type-error when reducing and is not generated from any typable session:

$$M_2 = \mathsf{q}{\triangleleft}(\mathsf{p}!\ell\langle 7\rangle.\mathbf{0}{+}\mathsf{p}!\ell\langle\mathtt{tt}\rangle.\mathbf{0}) \mid \mathsf{p}{\triangleleft}\mathsf{q}?\ell(x).\mathtt{if}\ x\ \mathtt{then}\ P_1\ \mathtt{else}\ P_2$$

M_2 is untypable since the corresponding type $\mathsf{q}!\ell\langle\mathsf{nat}\rangle{+}\mathsf{q}!\ell\langle\mathsf{bool}\rangle$ is not well-formed by [L-ℓ] in § 3.

(5) For Example 2.2, we have

$$\vdash P_{\mathsf{station}} \triangleright \textstyle\sum_{\mathsf{i}\in\{\mathsf{a,b,c,d,e}\}} \mathsf{i}?\mathit{elect};\sum_{\mathsf{i}\in\{\mathsf{a,b,c,d,e}\}} \mathsf{i}!\mathit{del};\mathsf{end}$$

and $\vdash P_{\mathsf{a}} \triangleright T_{\mathsf{a}}$, where T_{a} is

$$\begin{array}{l}\mathsf{e}!\mathit{leader};\mathsf{end}\ +\\ \mathsf{b}?\mathit{leader};(\mathsf{c}!\mathit{leader};\mathsf{end} + \mathsf{d}?\mathit{leader};\mathsf{station}!\mathit{elect};\mathsf{end})\ +\\ \mathsf{station}?\mathit{del};\mathsf{end}\end{array}$$

and $\vdash P_{\mathsf{b}} \triangleright T_{\mathsf{b}}$ with $T_{\mathsf{b}} = T_{\mathsf{a}}\sigma, \ldots, \vdash P_{\mathsf{e}} \triangleright T_{\mathsf{e}}$ with $T_{\mathsf{e}} = T_{\mathsf{d}}\sigma$.

5 PROPERTIES OF TYPED MCMP-CALCULUS

This section proves the main properties of the typed calculus, starting from the subject reduction theorem.

Lemma 5.1 (Subject Congruence)**.** *(1) Assume $\Gamma \vdash P \triangleright T$. If $P \equiv Q$, then $\Gamma \vdash Q \triangleright T$. (2) Assume $\vdash M \triangleright \Delta$. If $M \equiv M'$, then $\vdash M' \triangleright \Delta$.*

Theorem 5.1 (Subject Reduction)**.** *Assume $\vdash M \triangleright \Delta$. If $M \longrightarrow M'$, then there exists such that $\vdash M' \triangleright \Delta'$ and $\Delta \longrightarrow^* \Delta'$.*

Proof. We use the lemma that safety property of local contexts is closed under subtyping (Lemma 3.1). We then use the closure under the structure congruence (Lemma 5.1). ◻

A consequence of Theorem 5.1 is that a well-typed process never reduces to an error state. In MCMP, the error definition needs to consider all possible synchronisations among multiple parallel processes (see Example 5.1).

Definition 5.1 (Session error)**.** A session M is a *label error session* if:

$$M \equiv \mathsf{p} \triangleleft \textstyle\sum_{i\in I} \pi_i.P_i \mid \mathsf{q} \triangleleft \sum_{j\in J} \pi_j.Q_j \mid M'$$

where if there exists $\pi_i = \mathsf{q}!\ell\langle v\rangle$ with $i \in I$, then $\forall k \in j$ such that $\pi_k = \mathsf{p}?\ell_k(x_k)$, we have $\ell_k \neq \ell$ (i.e., all input labelled processes are unmatched). A session M is a *value error process* if:

$$M \equiv \mathsf{p} \triangleleft \mathtt{if}\ v\ \mathtt{then}\ P_1\ \mathtt{else}\ P_2 \mid M' \quad \text{ with } v \notin \{\mathtt{tt},\mathtt{ff}\}$$

We call M is a *session error* if it is either label or value error session.

Example 5.1 (Label error session)**.** A *label error session* is a session that contains a pair of input and output with dual participants, but does not have a correct labelled redex. For example, session

$$M = \mathsf{p} \triangleleft (\mathsf{q}!\ell_1.\mathbf{0} + \mathsf{r}?\ell_2.\mathbf{0}) \mid \mathsf{q} \triangleleft \mathsf{p}?\ell_2.\mathbf{0} \mid \mathsf{r} \triangleleft \mathsf{p}!\ell_2.\mathbf{0}$$

is a session error because it has an active redex ($\mathsf{q}!\ell_1$ and $\mathsf{p}?\ell_2$) that is mismatched. M has a type context: $\Delta = \mathsf{p} : (\mathsf{q}!\ell_1{+}\mathsf{r}?\ell_2), \mathsf{q} : \mathsf{p}!\ell_2, \mathsf{r} : \mathsf{p}!\ell_2$ and Δ is not safe. Note that session M' defined below is *not* a session error.

$$M' = \mathsf{p} \triangleleft (\mathsf{q}?\ell_1.\mathbf{0} + \mathsf{q}?\ell_2.\mathbf{0}) \mid \mathsf{q} \triangleleft \mathsf{p}!\ell_2.\mathbf{0}$$

as in the standard (multiparty) session types [11, 56].

From Theorem 5.1 and the fact that error sessions are untypable by a safe context, we have:

Corollary 5.1 (Communication Safety)**.** Assume $\vdash M \triangleright \Delta$. For all M', such that $M \longrightarrow^* M'$, M' is not a session error.

Deadlock-freedom (from [56, Definition 5.1]) states that a process either terminates, completing all actions, or makes progress.

Definition 5.2 (Deadlock-freedom)**.** Session M is *deadlock-free* iff for all M' such that $M \longrightarrow^* M'$ either (1) $M' \not\longrightarrow$ and $M' \equiv \mathsf{p} \triangleleft \mathbf{0}$ for some p, or (2) there exists M'' such that $M' \longrightarrow M''$.

Theorem 5.2 (Deadlock-freedom)**.** *Assume $\vdash M \triangleright \Delta$ and $\mathrm{dfree}(\Delta)$. Then M is deadlock-free.*

Proof. Assume $M \longrightarrow^* M' \not\longrightarrow$ and $\vdash M' \triangleright \Delta'$. Then by Theorem 5.1, there exists Δ' such that $\Delta \longrightarrow^* \Delta' \not\longrightarrow$ and $\vdash M' \triangleright \Delta'$. By the definition of $\mathrm{dfree}(\Delta)$, $\Delta' = \{\mathsf{p}_i : \mathsf{end}\}_{i\in I}$. Hence $M' = \Pi_{i\in I}\ \mathsf{p}_i \triangleleft \mathbf{0}$ and $M' \equiv \mathsf{p}_k \triangleleft \mathbf{0}$ for some $k \in I$ by definition of $\equiv$. ◻

Remark 5.1 (Properties)**.** In this paper, we follow a *general typing system* in [56] where a property φ of a session M is guranteed by checking the same propery φ of its typing context Δ [56, Definition 5.1, Theorem 5.15]. Hence if we replace $\mathrm{dfree}(\Delta)$ in Theorem 5.2 to a liveness property in [20], then a typed session M can gurantee a liveness property. We selected deadlock-freedom since it is used to prove all the encodability results (see § 6). This methodology from [56] is often called *bottom-up*. The classical MP [12, 30, 66] takes the *top-down* approach where the user first writes a *global type* as a protocol specification. See § 7 for further discussions.

6 HIERARCHY OF EXPRESSIVENESS OF SESSION CALCULI

We analyse the expressive power of the mixed choice construct in the MCMP-calculus by several separation and encodability results. In all subcalculi of MCMP, we assume that they are typed and deadlock-free. The hierarchy of expressiveness of the 9 subcalculi and 4 variants of CMV^{+} [8] is given in Figure 5. We label arrows with a reference to the respective result. More precisely:

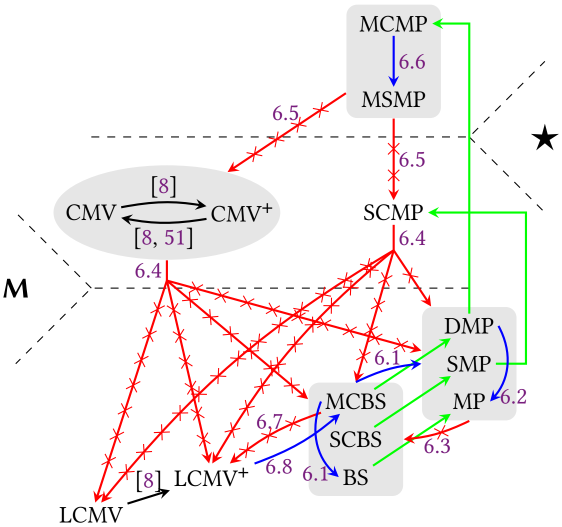


Figure 5: Hierarchy of Session Calculi

- $L_1 \not\rightarrow L_2$: There is no good encoding from L_1 into L_2.
- $L_1 \longrightarrow L_2$: There is a good encoding from L_1 into L_2 and the arrow presents a new encodability result.
- $L_1 \longrightarrow L_2$: There is a good encoding from L_1 into L_2 that has been proven in the literature.

If $L_1 \subset L_2$, then identity is a good encoding from L_1 into L_2. If L_1 is placed straight below L_2 then $L_1 \subset L_2$. Additional inclusions are depicted by $\longrightarrow$ (e.g. SMP $\longrightarrow$ SCMP since SMP $\subset$ SCMP).

A grey shape captures calculi of the same expressive power, i.e., there is a good encoding between any pair of calculi in the same shape (e.g., MCMP and MSMP). Similarly, arrows from (or into) a shape are understood as an arrow from (or into) each calculus of the shape (e.g. from $L_1 \in \{\text{MCBS}, \text{SCBS}, \text{BS}\}$ into $L_2 \in \{\text{DMP}, \text{SMP}, \text{MP}\}$).

Encodability Criteria. We combine the encodability criteria from [24] and [48]. Two steps are in *conflict*, if performing one step disables the other step, i.e., if both reduce the same choice. Otherwise they are *distributable*. M is distributable into $M_1, \ldots, M_n$ if and only if we have $M \equiv M_1 \mid \ldots \mid M_n$, where $\equiv$ does not unfold any recursions. We add $\checkmark$ (*successful termination*) to all calculi in addition to $\mathbf{0}$ and type it in the same way as $\mathbf{0}$. $M \Downarrow_{\diamond\checkmark}$ if $M \longrightarrow^* M'$ and M' has an unguarded occurrence of $\checkmark$. Moreover, let the equivalence $\asymp$ be a success respecting (weak) reduction bisimulation, i.e., if $M_1 \asymp M_2$ then (1) $M_1 \longrightarrow M_1'$ implies $M_2 \longrightarrow^* M_2'$ and $M_1' \asymp M_2'$, (2) $M_2 \longrightarrow M_2'$ implies $M_1 \longrightarrow^* M_1'$ and $M_1' \asymp M_2'$, and (3) $M_1 \Downarrow_{\diamond\checkmark}$ iff $M_2 \Downarrow_{\diamond\checkmark}$.

We consider an encoding $[\![\cdot]\!]$ to be *good* if it is

compositional: The translation of an operator $\mathbf{op}$ is a function $C_{\mathbf{op}}^{\mathbb{P}}$ on the translations of the subterms of the operator, i.e., $[\![\mathbf{op}\,(M_1, \ldots, M_n)]\!] = C_{\mathbf{op}}^{\mathbb{P}}([\![M_1]\!], \ldots, [\![M_n]\!])$ for all $M_1, \ldots, M_n$ with $\mathbb{P} = \text{pt}(M_1) \cup \ldots \cup \text{pt}(M_n)$.

name invariant: For every M and every substitution σ, it holds that $[\![M\sigma]\!] \asymp [\![M]\!]\,\sigma$.

operationally complete: For all $M \longrightarrow M'$, $[\![M]\!] \longrightarrow\asymp [\![M']\!]$.

operationally sound: For all $[\![M]\!] \longrightarrow N$, there is an M' such that $M \longrightarrow M'$ and $N \longrightarrow\asymp [\![M']\!]$.

divergence reflecting: For every M, $[\![M]\!] \longrightarrow^\omega$ implies $M \longrightarrow^\omega$.

success sensitive: For every M, $M \Downarrow_{\diamond\checkmark}$ iff $[\![M]\!] \Downarrow_{\diamond\checkmark}$.

distributability preserving: For every M and for all $M_1, \ldots, M_n$ that are distributable within M there are some $N_1, \ldots, N_n$ that are distributable within $[\![M]\!]$ s.t. $N_i \asymp [\![M_i]\!]$ for all $1 \leq i \leq n$.

Operational correspondence is the combination of operational *completeness* and *soundness*.

Hierarchy. We summarise the keys to the presented separation and encodability results, i.e., the counterexample for separation and the interesting parts of the encoding functions, in Figure 6 and explain them below. The parts of encodings given in Figure 6 always present the encoding of a participant p.

The encodings in Figure 6 use a total order $<$ on participants. To ensure compositionality and name invariance, $<$ is not assumed but constructed by the encoding functions from the tree-structure of parallel composition in source terms.

We start with the subcalculi of MCMP and go from bottom to top. The counter in the following subsection headings refers to the number of encodability or separation results, but we are not counting results obtained from calculus inclusion. Figure 6(n) denotes the nth row in the figure.

6.1 Encodability (1–3): Binary Sessions

In binary sessions, there is no difference in the expressive power between separate or mixed choice.

Theorem 6.1 (Binary Sessions). *Let $\mathcal{L} = \{BS, SCBS, MCBS\}$. There is a good encoding from any $L_1 \in \mathcal{L}$ into any $L_2 \in \mathcal{L}$.*

From SCBS to BS. In contrast to SCBS with separate choice, there are only single outputs in BS and no choices on outputs. Therefore the encoding SCBS $\longrightarrow$ BS translates an output-guarded choice to an input-guarded choice with the same input-prefix $\text{q}?enc_o$ and the respective outputs as continuations (see Figure 6(1)). As explained in Example 4.1(1,2), such choices are typable in BS because of subtyping. The encoding of input-guarded choice starts with the matching output $\text{q}!enc_o$ followed by the original choice. Accordingly, a single interaction in the source term between some output-guarded and some input-guarded choice is translated into two steps on the target side.

Deadlock-freedom ensures the existence of the respective communication partner for the first step (with the fresh label enc_o) and safety ensures that for every output that was picked in the first step there will be a matching input in the second step. The rest of the encoding SCBS $\longrightarrow$ BS is homomorphic.

From MCBS to SCBS. The encoding MCBS $\longrightarrow$ SCBS has to translate mixed into separate choices (see Figure 6(2)). Therefore, it translates a choice from p to q with p $<$ q into an output-guarded choice and with p $>$ q into an input-guarded choice, where $\text{q}!enc_i$ (or $\text{q}?enc_i$) are used to guard the outputs in the input-guarded choice (the inputs in the output-guarded choice).

Accordingly, an interaction of an output in p and an input in q with p $<$ q can be emulated by a single step, whereas the case p $>$ q requires two steps: one interaction with the fresh label enc_i

1	SCBS $\longrightarrow$ BS SMP $\longrightarrow$ MP	$\llbracket \sum_{i \in I} q!\ell_i\langle v_i\rangle.P_i \rrbracket = \sum_{i \in I} q?enc_o . q!\ell_i\langle v_i\rangle. \llbracket P_i \rrbracket$ $\llbracket \sum_{j \in J} q?\ell_j(x_j).P_j \rrbracket = q!enc_o . \sum_{j \in J} q?\ell_j(x_j). \llbracket P_j \rrbracket$
2	MCBS $\longrightarrow$ SCBS DMP $\longrightarrow$ SMP	$\left\llbracket (\sum_{i \in I} q!\ell_i\langle v_i\rangle.P_i) + \left(\sum_{j \in J} q?\ell_j(x_j).P_j\right) \right\rrbracket$ $= \begin{cases} (\sum_{i \in I} q!\ell_i\langle v_i\rangle. \llbracket P_i \rrbracket) + \\ q!enc_i . \left(\left(\sum_{j \in J} q?\ell_j(x_j). \llbracket P_j \rrbracket\right) + q?reset . \sum_{i \in I} q!\ell_i\langle v_i\rangle. \llbracket P_i \rrbracket\right) & \text{if } I \neq \emptyset \neq J \wedge p < q \\ \left(\sum_{j \in J} q?\ell_j(x_j). \llbracket P_j \rrbracket\right) + q?enc_i . \sum_{i \in I} q!\ell_i\langle v_i\rangle. \llbracket P_i \rrbracket & \text{if } I \neq \emptyset \neq J \wedge p > q \\ \sum_{i \in I} q!\ell_i\langle v_i\rangle. \llbracket P_i \rrbracket & \text{if } J = \emptyset \wedge p < q \\ q?enc_i . \sum_{i \in I} q!\ell_i\langle v_i\rangle. \llbracket P_i \rrbracket & \text{if } J = \emptyset \wedge p > q \\ q!enc_i . \sum_{j \in J} q?\ell_j(x_j). \llbracket P_j \rrbracket & \text{if } I = \emptyset \wedge p < q \\ \left(\sum_{j \in J} q?\ell_j(x_j). \llbracket P_j \rrbracket\right) + q?enc_i . q!reset . \sum_{j \in J} q?\ell_j(x_j). \llbracket P_j \rrbracket & \text{if } I = \emptyset \wedge p > q \end{cases}$
3	MCBS $\longrightarrow$ BS DMP $\longrightarrow$ MP	$\left\llbracket (\sum_{i \in I} q!\ell_i\langle v_i\rangle.P_i) + \left(\sum_{j \in J} q?\ell_j(x_j).P_j\right) \right\rrbracket$ $= \begin{cases} \left(\sum_{i \in I} q?enc_o . q!\ell_i\langle v_i\rangle. \llbracket P_i \rrbracket\right) + \\ q?enc_o . q!enc_i . \left(\left(\sum_{j \in J} q?\ell_j(x_j). \llbracket P_j \rrbracket\right) + q?reset . \sum_{i \in I} q!\ell_i\langle v_i\rangle. \llbracket P_i \rrbracket\right) & \text{if } I \neq \emptyset \neq J \wedge p < q \\ q!enc_o . \left(\left(\sum_{j \in J} q?\ell_j(x_j). \llbracket P_j \rrbracket\right) + q?enc_i . \sum_{i \in I} q!\ell_i\langle v_i\rangle. \llbracket P_i \rrbracket\right) & \text{if } I \neq \emptyset \neq J \wedge p > q \\ \sum_{i \in I} q?enc_o . q!\ell_i\langle v_i\rangle. \llbracket P_i \rrbracket & \text{if } J = \emptyset \wedge p < q \\ q!enc_o . q?enc_i . \sum_{i \in I} q!\ell_i\langle v_i\rangle. \llbracket P_i \rrbracket & \text{if } J = \emptyset \wedge p > q \\ q?enc_o . q!enc_i . \sum_{j \in J} q?\ell_j(x_j). \llbracket P_j \rrbracket & \text{if } I = \emptyset \wedge p < q \\ q!enc_o . \left(\left(\sum_{j \in J} q?\ell_j(x_j). \llbracket P_j \rrbracket\right) + q?enc_i . q!reset . \sum_{j \in J} q?\ell_j(x_j). \llbracket P_j \rrbracket\right) & \text{if } I = \emptyset \wedge p > q \end{cases}$
4	MP $\not\twoheadrightarrow$ MCBS	three distributable steps in $M_{\mathrm{MP}} = a \triangleleft (b!l.\checkmark + b!l.\mathbf{0}) \mid b \triangleleft a?l.\mathbf{0} \mid c \triangleleft (d!l.\checkmark + d!l.\mathbf{0}) \mid d \triangleleft c?l.\mathbf{0} \mid e \triangleleft (f!l.\checkmark + f!l.\mathbf{0}) \mid f \triangleleft e?l.\mathbf{0}$
5	SCMP $\not\twoheadrightarrow$ Bottom Layer	the synchronisation pattern $\mathbf{M}$ in $M^{\mathbf{M}}_{\mathrm{SCMP}} = a \triangleleft (b!l.d!l.\mathbf{0} + d!l'.\mathbf{0}) \mid b \triangleleft (a?l.\mathbf{0} + c?l.\mathbf{0}) \mid c \triangleleft (b!l.\mathbf{0} + d!l.\mathbf{0}) \mid d \triangleleft (c?l.a?l.\checkmark + a?l'.\mathbf{0})$
6	CMV $\not\twoheadrightarrow$ Bottom Layer	the synchronisation pattern $\mathbf{M}$ in $M^{\mathbf{M}}_{\mathrm{CMV}} = (\nu xy)\, (x!\mathtt{tt}.\mathbf{0} \mid \mathsf{lin}\, y?z.\mathsf{if}\, z\, \mathsf{then}\, \mathbf{0}\, \mathsf{else}\, \mathbf{0} \mid x!\mathtt{ff}.\mathbf{0} \mid \mathsf{lin}\, y?z.\mathsf{if}\, z\, \mathsf{then}\, \mathbf{0}\, \mathsf{else}\, \checkmark)$
7	CMV^{+} $\not\twoheadrightarrow$ Bottom Layer	the synchronisation pattern $\mathbf{M}$ in $M^{\mathbf{M}}_{\mathrm{CMV}^{+}} = (\nu xy)\, (\mathsf{lin}\, x\, l!\mathtt{tt}.\mathbf{0} \mid \mathsf{lin}\, y\, l?z.\mathsf{if}\, z\, \mathsf{then}\, \mathbf{0}\, \mathsf{else}\, \mathbf{0} \mid \mathsf{lin}\, x\, l!\mathtt{ff}.\mathbf{0} \mid \mathsf{lin}\, y\, l?z.\mathsf{if}\, z\, \mathsf{then}\, \mathbf{0}\, \mathsf{else}\, \checkmark)$
8	MSMP $\not\twoheadrightarrow$ Lower Layers	the synchronisation pattern $\star$ in $M^{\star}_{\mathrm{MSMP}} = M_a \mid M_b \mid M_c \mid M_d \mid M_e \mid M_{gc} \qquad M_a = a \triangleleft (e!l.\mathbf{0} + b?l.P_a + gc?del.\mathbf{0})$ $M_{gc} = \sum_{i \in \{a,b,c,d,e\}} i!del.\mathbf{0} \qquad \sigma = [a \mapsto b, b \mapsto c, c \mapsto d, d \mapsto e, e \mapsto a]$ $M_b = M_a\sigma \quad M_c = M_b\sigma \quad M_d = M_c\sigma \quad M_e = M_d\sigma \qquad P_a, \ldots, P_e \in \{\checkmark, \mathbf{0}\}$
9	MCMP $\longrightarrow$ MSMP	$\left\llbracket \sum_q \left((\sum_{i \in I} q!\ell_i\langle v_i\rangle.P_i) + \left(\sum_{j \in J} q?\ell_j(x_j).P_j\right)\right) \right\rrbracket = \sum_q \{$case distinction of MCBS $\longrightarrow$ SCBS
10	MCBS $\not\twoheadrightarrow$ LCMV^{+}	$M_{\mathrm{MCBS}} = p \triangleleft \left(q!l_1.\mathbf{0} + q?l_2.\checkmark\right) \mid q \triangleleft \left(p?l_1.\mathbf{0} + p?l_3.\checkmark\right)$
11	LCMV^{+} $\longrightarrow$ MCBS	$\llbracket \Gamma \vdash (\nu xy)P \rrbracket = \llbracket \Gamma' \vdash P \rrbracket_{x,y}$ $\left\llbracket \Gamma \vdash \mathsf{lin}\, k\, \left(P = (\sum_{i \in I} \ell_i!v_i.P_i) + \left(\sum_{j \in J} \ell_j?z_j.P_j\right)\right) \right\rrbracket_{x,y}$ $= \begin{cases} k \triangleleft \mathbf{0} & \text{if } \mathrm{fv}(P) = \{x,y\} \cup \{k\} \\ k \triangleleft \left(\sum_{i \in I} \overline{k}!\ell_i \cdot o\, \langle v_i\rangle. \llbracket \Gamma_i \vdash P_i \rrbracket_{\overline{k}}\right) & \text{if } \mathrm{fv}(P) \neq \{x,y\}, \overline{k} \in \{x,y\} \setminus \{k\}, \\ \quad + \left(\sum_{j \in J} \overline{k}!\ell_i \cdot i\, . \overline{k}?\ell_i(z). \llbracket \Gamma_j \vdash P_j \rrbracket_{\overline{k}}\right) & \quad \text{and } P \text{ is typed as internal in } \Gamma \\ k \triangleleft \left(\sum_{i \in I} \overline{k}?\ell_i \cdot i\, . \overline{k}!\ell_i\langle v_i\rangle. \llbracket \Gamma_i \vdash P_i \rrbracket_{\overline{k}}\right) & \text{if } \mathrm{fv}(P) \neq \{x,y\}, \overline{k} \in \{x,y\} \setminus \{k\}, \\ \quad + \left(\sum_{j \in J} \overline{k}?\ell_i \cdot o\, (z). \llbracket \Gamma_j \vdash P_j \rrbracket_{\overline{k}}\right) & \quad \text{and } P \text{ is typed as external in } \Gamma \end{cases}$

Figure 6: Keys to Separation and Encodability Results

and the interaction of the respective out- and input. In the case of p > q, deadlock-freedom guarantees the existence of the respective communication partner for the first step, and safety the existence of a matching input for every output in the second step.

In the case p < q, additional inputs in p without matching outputs in q require a special attention. The type system of the target calculus SCBS forces us to ensure that the output $\mathsf{q}!enc_i$, that guards the inputs in the translation of p, is matched by an input $\mathsf{p}?enc_i$ in the translation of q (see Corollary 5.1). We add $\mathsf{p}?enc_i\,.\,\mathsf{p}!reset\,.\sum_{j\in J}\mathsf{p}?\ell_j(x_j).\,[\![P_j]\!]$ to the translation of q (the last summand in the last line of Figure 6(2) with p and q swapped). If p has outputs and inputs but no input has a match in q, then the emulation of a single source term step takes either one or three steps.

The rest of MCBS $\longrightarrow$ SCBS is homomorphic (except for the construction of <).

From MCBS to BS. The encoding MCBS $\longrightarrow$ BS combines the ideas of SCBS $\longrightarrow$ BS and MCBS $\longrightarrow$ SCBS (see Figure 6(3)).

Inclusion. In the remaining cases of $L_1, L_2 \in \{\text{BS}, \text{SCBS}, \text{MCBS}\}$ identity is a good encoding, because $L_1 = L_2$ or $L_1 \subset L_2$ since BS $\subset$ SCBS $\subset$ MCBS.

6.2 Encodability (4–6): Multiparty Sessions with Choices on a Single Participant

In multiparty sessions, in that choice is limited to one participant, mixed choice does not increase the expressive power.

Theorem 6.2 (Multiparty Sessions with Choices on a Single Participant)**.** *Let $\mathcal{L} = \{MP, SMP, DMP\}$. There is a good encoding from any $L_1 \in \mathcal{L}$ into any $L_2 \in \mathcal{L}$.*

The encoding SMP $\longrightarrow$ MP translates choice in the same way as SCBS $\longrightarrow$ BS. Encoding DMP $\longrightarrow$ SMP inherits the encoding of mixed to separate choices from MCBS $\longrightarrow$ SCBS. DMP $\longrightarrow$ MP combines SMP $\longrightarrow$ MP and DMP $\longrightarrow$ SMP. In all three encodings, all remaining operators are translated homomorphically (except for the construction of <). In the remaining cases of $L_1, L_2 \in \{\text{MP}, \text{SMP}, \text{DMP}\}$, identity is a good encoding because $L_1 = L_2$ or $L_1 \subset L_2$ since MP $\subset$ SMP $\subset$ DMP.

6.3 Encodability: Binary into Multiparty with Choice on a Single Participant

We now consider the blue arrow between the grey squares in the bottom layer. It tells us, that binary sessions with mixed choice can be simulated by the classical multiparty sessions (MP), since in both cases mixed choice does not add expressive power.

Corollary 6.1 (Binary into Multiparty with Choice on a Single Participant). *There is a good encoding from any $L_1 \in \{BS, SCBS, MCBS\}$ into any $L_2 \in \{MP, SMP, DMP\}$.*

Corollary 6.1 follows from Theorem 6.1 and calculus inclusion. The encoding MCBS $\longrightarrow$ BS is also a good encoding from MCBS into MP, since BS $\subset$ MP. With MP $\subset$ SMP then this encoding is also a good encoding from MCBS into SMP. Identity is a good encoding from MCBS into DMP, because MCBS $\subset$ DMP. Similarly, we obtain good encodings for L_1 = SCBS and L_1 = BS from the encoding SCBS $\longrightarrow$ BS and the calculus inclusions BS $\subset$ MP $\subset$ SMP $\subset$ DMP and SCBS $\subset$ SMP.

6.4 Separate (1) Multiparty Sessions with Choice on a Single Participant from Binary Sessions

We now show the first separation result: there exists no good encoding from MP into MCBS.

Theorem 6.3 (Separate Multiparty with Choice on a Single Participant from Binary Sessions)**.** *There is no good encoding from any $L_1 \in \{MP, SMP, DMP\}$ into any $L_2 \in \{BS, SCBS, MCBS\}$.*

Note that the binary versions BS, SCBS, and MCBS include only a single binary session (hence 2 parties). Such a system cannot perform three distributable steps, nor can it emulate such behaviour while respecting the criterion on the preservation of distributability. We use M_{MP} from Figure 6(4) as counterexample.

By MP $\subset$ SMP $\subset$ DMP, M_{MP} is also contained in SMP and DMP. If there would be a good encoding from MP into SCBS, then this encoding would also be a good encoding from MP into MCBS, because SCBS $\subset$ MCBS; i.e., such an encoding contradicts MP $\not\longrightarrow$ MCBS. Hence, the remaining separation results in Theorem 6.3 follow from MP $\not\longrightarrow$ MCBS and calculus inclusion.

6.5 Separate (2–13) the Middle from the Bottom

Going further upwards in Figure 5, we observe two dashed lines above DMP (see also Figure 1). These two dashed lines divide Figure 5 into three layers along the ability to express the synchronisation patterns $\star$ and M from [48, 62] and thus the different amounts of synchronisation they capture. Calculi in the bottom layer can express neither M nor $\star$ and are considered as asynchronously distributed calculi.

The pattern M. A system is an M if it is distributable into two parts that can act independently by performing the distributable steps a and c; but that may also interact in a step b that is in conflict to both a and c.

Definition 6.2 (Synchronisation Pattern M*).* Let M^{M} be a process such that:

- M^{M} can perform at least three alternative steps a: $M^{\mathsf{M}} \longrightarrow M_a$, b: $M^{\mathsf{M}} \longrightarrow M_b$, and c: $M^{\mathsf{M}} \longrightarrow M_c$ such that M_a, M_b, and M_c are pairwise different.
- The steps a and c are distributable in M^{M}.
- But b is in conflict with both a and c.

In this case, we denote the process M^{M} as M.

In an M, the two parts of the system decide whether they interact or proceed independently, but are able to make this decision consistently without any form of interaction. This is a minimal form of synchronisation. The system $M^{\mathsf{M}}_{\text{SCMP}}$ in Figure 6(5) is an example of an M in SCMP, where step a is an interaction of a and b, step b is an interaction of b and c, and step c is an interaction of c and d.

Mixed Sessions. In the bottom layer we find LCMV and LCMV$^+$ that we introduce as subcalculi of CMV and the *mixed sessions* CMV$^+$ introduced in [8]. We briefly recap their main concepts (see [52] for more). The syntax of CMV$^+$ is given as

$$P ::= q\,x\,\textstyle\sum_{i\in I} M_i \;\mid\; P \mid P \;\mid\; (\nu xy)P \;\mid\; \text{if } v \text{ then } P \text{ else } P \;\mid\; \mathbf{0}$$

where $M ::= \ell{*}v.P$, $* ::= \,!\ \mid\ ?$, and $q ::= \mathsf{lin}\ \mid\ \mathsf{un}$ denote *linear* and *unrestricted* choices. The double restriction $(\nu xy)P$ introduces two matching *session endpoints* x and y. Interaction is limited to two matching endpoints

$$(\nu yz)(\mathsf{lin}\, y\ (\ell!v.P + M) \mid \mathsf{lin}\, z\ (\ell?x.Q + N) \mid R) \longrightarrow (\nu yz)(P \mid Q\{v/x\} \mid R)$$

where linear choices are consumed, whereas unrestricted choices are persistent. In CMV, choice is replaced by output $y!v.P$, input $q\, y?x.P$, selection $x \triangleleft \ell.P$, and branching $x \triangleright \{\ell_i : P_i\}_{i \in I}$.

LCMV$^+$ and LCMV result from restricting CMV$^+$ and CMV to a single sessions, i.e., at most one restriction, forbidding delegation, i.e., only values can be transmitted, allowing only linear choices that are typed as linear.

The Bottom Layer. LCMV, LCMV$^+$, BS, SCBS, MCBS, MP, SMP, and DMP are placed in the bottom, because they do not contain the synchronisation pattern $\mathbf{M}$.

Lemma 6.1 (No $\mathbf{M}$)**.** There are no $\mathbf{M}$ in LCMV, LCMV$^+$, BS, SCBS, MCBS, MP, SMP, and DMP.

We show that these languages have no distributable steps a and c that are both in conflict to a step b.

Separation. Accordingly, the ability to combine different communication partners in choice is the key to the $\mathbf{M}$ in $M^{\mathbf{M}}_{\mathrm{SCMP}}$, because the typing discipline of MCMP and its variants forbids different participants with the same name.

Theorem 6.4 (Separate Middle from Bottom)**.** *There is no good encoding from any $L_1 \in \{CMV, CMV^+, SCMP\}$ into any calculus $L_2 \in \{LCMV, LCMV^+, BS, SCBS, MCBS, MP, SMP, DMP\}$.*

In the proof we show that the $\mathbf{M}$ is preserved by the criteria of a good encoding, i.e., to emulate the behaviour of an $\mathbf{M}$ the target calculus needs an $\mathbf{M}$. Then $M^{\mathbf{M}}_{\mathrm{SCMP}}$ in Figure 6(5) is used as counterexample if $L_1 = \mathrm{SCMP}$, $M^{\mathbf{M}}_{\mathrm{CMV}}$ in Figure 6(6) if $L_1 = \mathrm{CMV}$, and $M^{\mathbf{M}}_{\mathrm{CMV^+}}$ in Figure 6(7) if $L_1 = \mathrm{CMV}^+$.

6.6 Separate (14–18) the Top Layer from the Rest

The $\star$ can only be expressed by synchronously distributed calculi in the top layer. It describes the amount of synchronisation that is e.g. necessary to solve leader election in symmetric networks and captures the power of synchronisation of π with mixed choice.

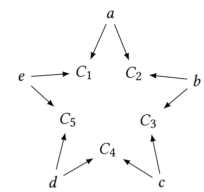
Definition 6.3 (Synchronisation Pattern $\star$). Let $M^\star$ be a process such that:

- $M^\star$ can perform at least five alternative steps $i : M^\star \longrightarrow M_i$ for $i \in \{a, b, c, d, e\}$ such that the M_i are pairwise different;
- the steps a, b, c, d, and e form a circle such that a is in conflict with b, b is in conflict with c, c is in conflict with d, d is in conflict with e, and e is in conflict with a; and
- every pair of steps in $\{a, b, c, d, e\}$ that is not in conflict due to the previous condition is distributable in $M^\star$.

In this case, we denote the process $M^\star$ as $\star$.

An example of a $\star$ in MSMP is $M^\star_{\mathrm{MSMP}}$ in Figure 6(8). Since MSMP $\subset$ MCMP, $M^\star_{\mathrm{MSMP}}$ is also a $\star$ in MCMP. All other calculi in Figure 5 do not contain $\star$s.

Lemma 6.2 (No $\star$)**.** There are no $\star$ in CMV, CMV$^+$, SCMP, LCMV, LCMV$^+$, BS, SCBS, MCBS, MP, SMP, and DMP.



Assume by contradiction, that SCMP contains a $\star$. Since a step reducing a conditional cannot be in conflict with any other step, the five steps of the $\star$ in SCMP are interactions that reduce an output-guarded and an input-guarded choice, respectively. Then in the steps a, b, c, d, e five choices $C_1, \ldots, C_5$ are reduced as depicted on the right, where e.g. the step a reduces the choices C_1 and C_2.

Since SCMP does not allow for mixed choices, each of the five choices $C_1, \ldots, C_5$ contains either only outputs or only inputs. W.o.g. assume that C_1 contains only outputs and C_2 only inputs. Then the choice C_3 needs to be on outputs again, because step b reduces C_2 (with only inputs) and C_3. Then C_4 is on inputs and C_5 is on outputs. But then step e reduces two choices C_1 and C_5 that are both on outputs. Since the reduction semantics of SCMP does not allow such a step, this is a contradiction.

The proofs for the absence of $\star$s in CMV and CMV$^+$ are similar and were already discussed in [51]. For the remaining cases, Lemma 6.2 follows from calculus inclusion.

Separation. We observe, that is indeed mixed choice (in contrast to only separate choice) that is the key to the $\star$ $M^\star_{\mathrm{MSMP}}$ in MSMP.

Theorem 6.5 (Separate the Top Layer)**.** *There is no good encoding from any $L_1 \in \{MSMP, MCMP\}$ into any $L_2 \in \{CMV, CMV^+\} \cup \{SCMP, LCMV, LCMV^+, BS, SCBS, MCBS, MP, SMP, DMP\}$.*

In the proof we show again that the $\star$ is preserved by the criteria of a good encoding, i.e., to emulate the behaviour of a $\star$ the target calculus needs a $\star$. Then $M^\star_{\mathrm{MSMP}}$ in Figure 6(8) is used as counterexample if $L_1 = \mathrm{MSMP}$. The other case, i.e., $L_1 = \mathrm{MCMP}$, follows then from MSMP $\subset$ MCMP.

6.7 Encodability (7): Mixed Choice into Separate Choice per Participant

The smallest calculus in Figure 5 that contains a $\star$ is MSMP, i.e., multiparty sessions with mixed choice that allow only separate choice per participant. In § 6.5 we learnt that the key to the synchronisation pattern $\mathbf{M}$ is the ability to combine different communication partners in choice. Here, we learn that also further up in the hierarchy, the important feature is the combination of different communication partners in a choice. Whether or not the choice limits the summands of the same participant in a mixed choice to either outputs or inputs does not change the expressive power.

Theorem 6.6 (Mixed Choice into Separate Choice per Participant)**.** *There is a good encoding from MCMP into MSMP and vice versa.*

The encoding MCMP $\longrightarrow$ MSMP translates mixed choices into choices that are separate per participant. Therefore, we apply for each participant the idea of MCBS $\longrightarrow$ SCBS and then combine the resulting choices for each participant in a choice (see Figure 6(9)).

6.8 Separate (19) MCBS from LCMV$^+$

Since all variants of MCMP describe only a single session without any form of shared names, comparing directly with CMV$^+$ would cause negative results that are completely independent of the respective choice constructs. Because of that, we introduced LCMV and LCMV$^+$ with LCMV $\subset$ CMV and LCMV$^+$ $\subset$ CMV$^+$ in Figure 5. We observe, that LCMV$^+$ cannot emulate all behaviours of MCBS.

Theorem 6.7 (Separate MCBS from LCMV$^+$). *There is no good encoding from MCBS into LCMV$^+$.*

The counterexample used for this separation result, namely M_{MCBS} in Figure 6(10), utilises the different typing mechanisms. The type system of CMV$^+$ requires duality of the types of the interacting choices, where subtyping increases flexibility but allows only choices typed as external to have additional summands, i.e., summands that are not matched by the communication partner such as $\mathrm{p}?l_2.\checkmark$ and $\mathrm{p}?l_3.\checkmark$ in M_{MCBS}. In MCBS, on the other hand side, additional inputs are allowed in *both* choices.

A translation of M_{MCBS} would need to turn one of the two choices into choice that is typed as external without additional summands. This prevents a good encoding from MCBS into LCMV$^+$.

6.9 Encodability (8): From LCMV$^+$ into MCBS

In the opposite direction there is an encoding from LCMV$^+$ into MCBS.

Theorem 6.8 (From LCMV$^+$ into MCBS). *There is a good encoding from LCMV$^+$ into MCBS.*

The subtyping in LCMV$^+$—in contrast to MCBS—does not only allow for additional inputs but also additional outputs. Hence, we use the type information in Γ, whether the choice we are translating is typed as internal or external. We translate all summands of a choice typed as external to outputs and all summands of a choice typed as internal to inputs. As done in [8], we extend the labels by o or i to ensure that the original matches are respected.

LCMV$^+$ does not guarantee deadlock-freedom. Fortunately, the limitation to a single session ensures that the only typed and deadlocked cases are systems in LCMV$^+$ which compose both communication partners sequentially. Accordingly, we translate choices that have both session endpoints as free variables to $\mathbf{0}$ in the first case of the translation of choice in Figure 6(11). The translations of the continuations $[\![P_i]\!]_{\overline{\mathrm{k}}}$ and $[\![P_j]\!]_{\overline{\mathrm{k}}}$ are similar to the encoding of choice $[\![\ldots]\!]_{x,y}$ but without the first case and without $\mathrm{k}\triangleleft$.

6.10 Separation (20) via Leader Election

The synchronisation patterns distribute the hierarchy in Figure 1 and 5 into asynchronously and synchronously distributed calculi. To further support the practical implications of our analysis, we are studying the problem of leader election in symmetric networks. Also the landmark result in [42] to separate π from π_{a} uses this problem as distinguishing feature.

A network $M = M_1 \mid \ldots \mid M_n$ is *symmetric* iff $M_{\sigma(i)} = M_i\sigma$ for each $i \in \{1, \ldots, n\}$ and for all permutations σ on participants. M is an *electoral system* if in every maximal execution exactly one leader is announced. The system Election in Example 2.2 presents a leader election protocol in a symmetric network with five participants in MSMP. Since calculi in the middle or bottom layer do not contain symmetric electoral systems, we can use this problem to separate MSMP from SCMP.

Theorem 6.9 (Separation via Leader Election). *There is no good and barb sensitive encoding from MSMP into SCMP.*

We prove first that SCMP cannot solve leader election in symmetric networks. To elect a leader, the initially symmetric network has to break its symmetry such that only one leader is elected. Without mixed choice, any attempt to do so can be counteracted by steps of the symmetric parts of the network that restore the original symmetry. This leads to an infinite sequence of steps in that no unique leader is elected. Then, we use Election as counterexample to show that there is no good and barb sensitiveness encoding. Therefore, we show that the combination of operational correspondence, divergence reflection, and barb sensitiveness ensures that the translation of a symmetric electoral system is again a symmetric electoral system.

Note that we use here barb sensitiveness, i.e., a source network and its translation may reach the same barbs, as additional encodability criterion, where *barbs* [40] are the standard observables used in π. Barb sensitiveness ensures that the announcement of the leader is respected by the encoding function, i.e., that the translation of an electoral system again announces exactly one leader.

6.11 Summary

The considered binary variants of MCMP have the same expressive power. The encodings introduce additional steps on the fresh labels enc_o, enc_i, and $reset$. Therefore, safety and deadlock-freedom of the source help us to ensure operational correspondence, i.e., that the target does not introduce new behaviours by introducing reductions that are stuck. Similarly, the variants of MP with choice on a single participant can be encoded into each other.

There is an increase in the expressive power for separate and mixed choice but *only if we combine different communication partners in a choice*, i.e., with SCMP for separate choice and MCMP for mixed choice; however, no difference between MSMP and MCMP.

Notice that each calculus plays a distinct role, i.e. it is novel from the viewpoint of types and/or expressiveness.

(1) MCMP, MSMP and SCMP are the first calculi ensuring safety and deadlock-freedom for which an increasing expressive power of choice was shown;
(2) MSMP is a realistic subform of general mixed choice in MCMP. A server may wait to receive requests from clients but also may send status information e.g. for load balancing;
(3) SCMP is the only calculus in the middle layer, that guarantees deadlock-freedom; and
(4) DMP has never appeared in the literature, but it can be viewed one step extension from MP with mixed choice construct, and MCBS is a binary version of DMP;
(5) SCBS and BS are a binary version of SMP and MP respectively, and BS can express MCBS.

Finally, we observe that MCBS is *strictly more expressive* than LCMV$^+$. The way to type choices in LCMV$^+$ is strictly less expressive than typing for choice in MCBS (see § 6.8 and § 6.9).

7 RELATED AND FUTURE WORK

Mixed Choice in the π-Calculus. Mixed choice has been proposed as a fundamental process calculi construct and extensively studied in the context of the π-calculi. Choice $P + Q$ first appeared in the original π-calculus syntax [39] where P and Q can contain any form of processes (such as parallel compositions). Later Milner proposed *guarded mixed choice* [38] where each process in the choice branch is either an input or an output. Palamidessi [42] has shown that mixed guarded choice cannot be encoded into separate guarded choice through a symmetric and divergence preserving encoding. Example 2.2 is motivated by leader election used to show this separation result in [42]. See Figure 1 in § 1 for a relationship with other calculi. Our guarded mixed choice differs from those in the standard π-calculus, where it can freely use distinct channels (like $P = a!v.Q_1 + b?(x).Q_2 + c!w.Q_3$); in MCMP, we use *distinct participants* to express mixed choice. For instance, we can mimic above P as $\mathsf{q}!v.Q_1 + \mathsf{r}?(x).Q_2 + \mathsf{p}!w.Q_3$ (with q, r, s pairwise distinct).

In [64], Glabbeek encodes a variant of the π-calculus with implicit matching into a variant of CCS, where the result of a synchronisation of two actions is itself an action subject to relabelling or restriction. A comparison between this variant of CCS and MCMP based on the results in [64] is an interesting future study; maybe using a syntactic similarity between CCS and local types.

Binary Mixed Sessions. Casal et al. [8] have proposed binary mixed sessions and provided an encoding from CMV$^+$ into a traditional variant of binary sessions with selection and branching (CMV) [65], and proved the encoding criteria except for operational soundness, which was left open. This problem was solved by Peters and Yoshida [51]. Further Casal et al. [8] provided a good encoding from CMV into CMV$^+$. Hence, CMV$^+$ and CMV have the same expressive power, i.e., mixed choice does not increase the expressive power in binary sessions in [65]. Additionally, Peters and Yoshida [51] have proved that CMV$^+$ cannot emulate the calculi in the top layer of Figure 1 (hence staying in the middle), proving leader election in symmetric networks cannot be solved in CMV$^+$. Here we prove that MCMP and its weaker form (MSMP) belong to the top.

Scalas and Yoshida [56] present a multiparty session type system which does not require global type correspondence; rather it identifies and computes the *desired* properties against a set of local types. We apply this general approach to build a typing system of mixed choice, but limit our attention to a single multiparty session, extending from MP in [19]. This allowed us to focus on which mixed choice construct can strictly raise the expressive power. Our future work is to analyse how much expressive power is added to multiparty session types by inclusions of session delegation from [56] and shared names from [12, 29, 30]. A related research question is a more detailed comparisons of SCMP with π_s and of MCMP with π in Figure 1. Remember that SCMP and MCMP are typed calculi that ensure e.g. deadlock-freedom, but the π-calculus does not. Since there are deadlock-free processes in π which are untypable by MCMP, considering of the minimal set of operators that, if added to MCMP, close a gap to π is an interesting question.

Global Types with Flexible Choices. In the context of multiparty session types, the research so far focuses to explore the *top-down* approach extend *global types* to more flexible choice. In the top-down approach [12, 30, 66, 67], a global type is projected into a set of local types; and if each participant typed by each local type, participants in a session automatically satisfies safety and deadlock-freedom (*correctness by construction*).

Castagna et al. [9] present a *semantic* procedure to check well-formedness of global types with parallel composition and mixed choice, which is undecidable due to infinite FIFO buffered semantics. They also propose a decidable algorithm for projecting a limited class of global types with their extension. Jongmans and Yoshida [32] extend global types with a mixed choice operator, an existential quantification over roles, and unbounded interleaving of subprotocols. It presents a bisimulation technique for developing a correspondence between global types and local interactions. Hamers and Jongmans [25] propose a *runtime verification framework* based on the domain-specific language (Disclojure) to verify programs against multiparty session types with mixed choice. The work concentrates on tool implementation, hence no theorem for correctness is provided. Majumdar et al. [36] present a generalised decidable projection procedure for multiparty session types with infinite FIFO buffered semantics, which extends the original syntax of global types to one sender with multiple receivers. They use a message causality analysis based on message sequence chart techniques to check the projectability of global types. This approach is further extended in [35] to enable a sound and complete projection from a global type to deadlock-free communicating automata [5]. Jongmans and Ferreira [31] propose a *synthetic* typing system which directly uses an operational semantics of *implicit* local types in a typing judgement of synchronous multiparty processes. An implicit local type has no explicit syntax but represents abstract behaviour of a global type wrt each participant. Their approach allows flexible type syntax including mixed choice in global types, but requires stronger conditions for realisable global types which are similar with those in [9]. Flexible choice for choreographies is studied in [13]. In [13] selection/branching (separate choice) is combined with multicoms/multisels. Multicoms/multisels group multiple actions, but as concurrent and not as choice actions (all actions can happen, not just one).

None of the above work with flexible choice [9, 31, 35, 36] has studied expressiveness of processes typed by their systems. It is an interesting future work to compare their expressive powers extending the work by Beauxis et al. [2] which studies encodability and separation results for the (untyped) π-calculi with mixed choice, stack, bags and FIFO queues.

Applications. The integration of model checking in a type system of the π-calculus is piloted by Chaki et al. [10], where the tool checks LTL formulae against behavioural types. Those ideas are applied to MPST in [56], which is extended to a crash-stop failure model in [1]. As programming language applications, Lange et al. [33, 34] and Gabet and Yoshida [18] extract behavioural types from Go source code, and Scalas et al. [57] design Scala library for communication programs with behavioural dependent types. These works use the mCLR2 to validate safety and deadlock-free properties through type-level behaviours. Notably, [18, 33, 34] use internal and external choices which consist of input, output and τ-prefixes to model `select` construct in Go. Extension of the model-checking tool in [56] to mixed choice and studying expressiveness of choice with τ-action are interesting topics for future research.

Acknowledgements. Work partially supported by: EPSRC EP/T006544/2, EP/K011715/1, EP/K034413/1, EP/L00058X/1, EP/N027833/2, EP/N028201/1, EP/T014709/2, EP/V000462/1, EP/X015955/1, NCSS/EPSRC VeTSS, and Horizon EU TaRDIS 101093006. We thank Dimitrios Kouzapas for his collaboration on an early version of a related topic.

REFERENCES

- [1] Adam D. Barwell, Alceste Scalas, Nobuko Yoshida, and Fangyi Zhou. 2022. Generalised Multiparty Session Types with Crash-Stop Failures. In *33rd International Conference on Concurrency Theory (LIPIcs, Vol. 243)*. Dagstuhl, 35:1–35:25. https://doi.org/10.4230/LIPIcs.CONCUR.2022.35
- [2] Romain Beauxis, Catuscia Palamidessi, and Frank D. Valencia. 2008. On the Asynchronous Nature of the Asynchronous pi-Calculus. In *Concurrency, Graphs and Models (LNCS, Vol. 5065)*. Springer, 473–492.
- [3] Gérard Berry and Georges Gonthier. 1992. The Esterel synchronous programming language: design, semantics, implementation. *Science of Computer Programming* 19, 2 (1992), 87–152. https://doi.org/10.1016/0167-6423(92)90005-V
- [4] Gérard Boudol. 1992. *Asynchrony and the π-calculus (note)*. Technical Report. Rapport de Recherche 1702, INRIA, Sophia-Antipolis.
- [5] Daniel Brand and Pitro Zafiropulo. 1983. On Communicating Finite-State Machines. *J. ACM* 30, 2 (apr 1983), 323–342. https://doi.org/10.1145/322374.322380
- [6] Marco Carbone, Kohei Honda, and Nobuko Yoshida. 2007. Structured Communication-Centred Programming for Web Services. In *Proceedings of ESOP 2007 (LNCS, Vol. 4421)*. Springer, 2–17.
- [7] Luca Cardelli and Andrew D. Gordon. 2000. Mobile Ambients. *Theoretical Computer Science* 240, 1 (2000), 177–213. Special Issue on Coordination, D. Le Métayer Editor.
- [8] Filipe Casal, Andreia Mordido, and Vasco T. Vasconcelos. 2022. Mixed sessions. *Theoretical Computer Science* 897 (2022), 23–48. https://doi.org/10.1016/j.tcs.2021.08.005
- [9] Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, and Luca Padovani. 2012. On Global Types and Multi-Party Session. *Logical Methods in Computer Science* 8, 1 (2012). https://doi.org/10.2168/LMCS-8(1:24)2012
- [10] Sagar Chaki, Sriram K. Rajamani, and Jakob Rehof. 2002. Types as Models: Model Checking Message-Passing Programs. In *Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Portland, Oregon) *(POPL '02)*. Association for Computing Machinery, New York, NY, USA, 45–57. https://doi.org/10.1145/503272.503278
- [11] Tzu-Chun Chen, Mariangiola Dezani-Ciancaglini, Alceste Scalas, and Nobuko Yoshida. 2017. On the Preciseness of Subtyping in Session Types. *Logical Methods in Computer Science* 13, 2 (2017). https://doi.org/10.23638/LMCS-13(2:12)2017
- [12] Mario Coppo, Mariangiola Dezani-Ciancaglini, Luca Padovani, and Nobuko Yoshida. 2015. A Gentle Introduction to Multiparty Asynchronous Session Types. In *15th International School on Formal Methods for the Design of Computer, Communication and Software Systems: Multicore Programming (LNCS, Vol. 9104)*. Springer, 146–178.
- [13] Luís Cruz-Filipe, Fabrizio Montesi, and Marco Peressotti. 2018. Communications in Choreographies, Revisited. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. 1248–1255. https://doi.org/10.1145/3167132.3167267
- [14] Romain Demangeon and Kohei Honda. 2011. Full Abstraction in a Subtyped pi-Calculus with Linear Types. In *Proceedings of CONCUR 2011 (LNCS, Vol. 6901)*. Springer, 280–296.
- [15] Cédric Fournet and Georges Gonthier. 1996. The Reflexive Chemical Abstract Machine and the Join-Calculus. In *Proc. of POPL*, Jr.G̃uy Steele (Ed.). ACM, 372–385. https://doi.org/10.1145/237721.237805
- [16] Yuxi Fu. 2016. Theory of Interaction. *Theoretical Computer Science* 611 (2016), 1–49. https://doi.org/10.1016/j.tcs.2015.07.043
- [17] Yuxi Fu and Hao Lu. 2010. On the expressiveness of interaction. *Theor. Comput. Sci.* 411, 11-13 (2010), 1387–1451. https://doi.org/10.1016/j.tcs.2009.11.011
- [18] Julia Gabet and Nobuko Yoshida. 2020. Static Race Detection and Mutex Safety and Liveness for Go Programs. In *34th European Conference on Object-Oriented Programming (LIPIcs)*. Schloss Dagstuhl.
- [19] Silvia Ghilezan, Svetlana Jaksic, Jovanka Pantovic, Alceste Scalas, and Nobuko Yoshida. 2019. Precise subtyping for synchronous multiparty sessions. *Journal of Logical and Algebraic Methods in Programming* 104 (2019), 127–173. https://doi.org/10.1016/j.jlamp.2018.12.002
- [20] Silvia Ghilezan, Jovanka Pantovic, Ivan Prokic, Alceste Scalas, and Nobuko Yoshida. 2021. Precise Subtyping for Asynchronous Multiparty Sessions. In *Proceedings of the ACM on Programming Languages (POPL, Vol. 5)*. ACM, 16:1–16:28.
- [21] Jean-Yves Girard. 1987. Linear Logic. *TCS* 50 (1987), 1–102.
- [22] Go Language. 2023. The Go Programming Language homepage. www.go.dev.
- [23] Daniele Gorla. 2009. On the Relative Expressive Power of Calculi for Mobility. In *Proc. of MFPS (ENTCS, Vol. 249)*. 269–286. https://doi.org/10.1016/j.entcs.2009.07.094
- [24] Daniele Gorla. 2010. Towards a unified approach to encodability and separation results for process calculi. *Inf. Comput.* 208, 9 (2010), 1031–1053. https://doi.org/10.1016/j.ic.2010.05.002
- [25] Ruben Hamers and Sung-Shik Jongmans. 2020. Discourje: Runtime Verification of Communication Protocols in Clojure. In *Tools and Algorithms for the Construction and Analysis of Systems*, Armin Biere and David Parker (Eds.). Springer International Publishing, Cham, 266–284.
- [26] Kohei Honda. 1993. Types for Dyadic Interaction. In *CONCUR'93 (LNCS, Vol. 715)*, Eike Best (Ed.). Springer-Verlag, 509–523.
- [27] Kohei Honda and Mario Tokoro. 1991. An Object Calculus for Asynchronous Communication. In *Proceedings of ECOOP 1991 (LNCS, Vol. 512)*. Springer, 133–147.
- [28] Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. 1998. Language Primitives and Type Disciplines for Structured Communication-based Programming. In *Proceedings of ESOP 1998 (LNCS, Vol. 1381)*. Springer, 22–138.
- [29] Kohei Honda, Nobuko Yoshida, and Marco Carbone. 2008. Multiparty Asynchronous Session Types. In *Proceedings of POPL 2008*. ACM, 273–284.
- [30] Kohei Honda, Nobuko Yoshida, and Marco Carbone. 2016. Multiparty Asynchronous Session Types. *Journal of the ACM* 63, 1 (2016), 9:1–9:67. https://doi.org/10.1145/2827695
- [31] Sung-Shik Jongmans and Francisco Ferreira. 2023. Synthetic Behavioural Typing: Sound, Regular Multiparty Sessions via Implicit Local Types. In *37th European Conference on Object-Oriented Programming (ECOOP 2023) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 263)*, Karim Ali and Guido Salvaneschi (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 42:1–42:30. https://doi.org/10.4230/LIPIcs.ECOOP.2023.42
- [32] Sung-Shik Jongmans and Nobuko Yoshida. 2020. Exploring Type-Level Bisimilarity towards More Expressive Multiparty Session Types. In *29th European Symposium on Programming (LNCS, Vol. 12075)*. Springer, 251–279.
- [33] Julien Lange, Nicholas Ng, Bernardo Toninho, and Nobuko Yoshida. 2017. Fencing off Go: liveness and safety for channel-based programming. In *Proceedings of POPL 2017*. ACM, 748–761. http://dl.acm.org/citation.cfm?id=3009847
- [34] Julien Lange, Nicholas Ng, Bernardo Toninho, and Nobuko Yoshida. 2018. A static verification framework for message passing in Go using behavioural types. In *Proceedings of ICSE 2018*. ACM, 1137–1148. https://doi.org/10.1145/3180155.3180157
- [35] Elaine Li, Felix Stutz, Thomas Wies, and Damien Zufferey. 2023. Complete Multiparty Session Type Projection with Automata. In *Computer Aided Verification*, Constantin Enea and Akash Lal (Eds.). Springer Nature Switzerland, 350–373.
- [36] Rupak Majumdar, Madhavan Mukund, Felix Stutz, and Damien Zufferey. 2021. Generalising Projection in Asynchronous Multiparty Session Types. In *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference (LIPIcs, Vol. 203)*, Serge Haddad and Daniele Varacca (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 35:1–35:24. https://doi.org/10.4230/LIPIcs.CONCUR.2021.35
- [37] Robin Milner. 1989. *Communication and Concurrency*. Prentice-Hall, Inc., USA.
- [38] Robin Milner. 1991. *The Polyadic pi-calculus: A Tutorial*. LFCS, Department of Computer Science, University of Edinburgh. https://books.google.co.uk/books?id=ht-9PQAACAAJ
- [39] Robin Milner, Joachim Parrow, and David Walker. 1992. A Calculus of Mobile Processes, Parts I and II.
- [40] Robin Milner and Davide Sangiorgi. 1992. Barbed Bisimulation. In *Proc. of ICALP (LNCS, Vol. 623)*. 685–695. https://doi.org/10.1007/3-540-55719-9_114
- [41] Uwe Nestmann. 2000. What is a "Good" Encoding of Guarded Choice? *Information and Computation* 156, 1-2 (2000), 287–319.
- [42] Catuscia Palamidessi. 2003. Comparing The Expressive Power Of The Synchronous And Asynchronous Pi-Calculi. *Mathematical Structures in Computer Science* 13, 5 (2003), 685–719.
- [43] Joachim Parrow. 2008. Expressiveness of Process Algebras. *Electr. Notes Theor. Comput. Sci.* 209 (2008), 173–186. https://doi.org/10.1016/j.entcs.2008.04.011
- [44] Kirstin Peters. 2012. *Translational Expressiveness*. Ph. D. Dissertation. TU Berlin. http://opus.kobv.de/tuberlin/volltexte/2012/3749/
- [45] Kirstin Peters. 2019. Comparing Process Calculi Using Encodings. In *Proc. of EXPRESS/SOS (EPTCS)*. 19–38. https://doi.org/10.48550/arXiv.1908.08633
- [46] Kirstin Peters and Uwe Nestmann. 2012. Is It a "Good" Encoding of Mixed Choice?. In *Foundations of Software Science and Computational Structures - 15th International Conference, FOSSACS 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 (Lecture Notes in Computer Science, Vol. 7213)*, Lars Birkedal (Ed.). Springer, 210–224. https://doi.org/10.1007/978-3-642-28729-9_14
- [47] Kirstin Peters and Uwe Nestmann. 2020. Distributability of Mobile Ambients. *Information and Computation* 275 (2020), 104608. https://doi.org/10.1016/j.ic.2020.104608
- [48] Kirstin Peters, Uwe Nestmann, and Ursula Goltz. 2013. On Distributability in Process Calculi. In *Proc. of ESOP 2013 (LNCS, Vol. 7792)*. Springer, 310–329. https://doi.org/10.1007/978-3-642-37036-6_18
- [49] Kirstin Peters, Jens-Wolfhard Schicke-Uffmann, and Uwe Nestmann. 2011. Synchrony vs Causality in the Asynchronous Pi-Calculus. In *Proc. of EXPRESS (EPTCS, Vol. 64)*. 89–103. https://doi.org/10.4204/EPTCS.64.7
- [50] Kirstin Peters and Rob J. van Glabbeek. 2015. Analysing and Comparing Encodability Criteria. In *Proc. of EXPRESS/SOS 2015 (EPTCS, Vol. 190)*. 46–60. https://doi.org/10.4204/EPTCS.190.4
- [51] Kirstin Peters and Nobuko Yoshida. 2022. On Expressiveness of Mixed Choice Sessions. *EPTCS* 368 (2022), 113–130.

[52] Kirstin Peters and Nobuko Yoshida. 2024. Separation and Encodability in Mixed Choice Multiparty Sessions (Technical Report). arXiv:2405.08104 [cs.LO]

[53] John Reppy, Claudio V. Russo, and Yingqi Xiao. 2009. Parallel Concurrent ML. *SIGPLAN Not.* 44, 9 (aug 2009), 257–268. https://doi.org/10.1145/1631687.1596588

[54] John H. Reppy. 1991. CML: A Higher-Order Concurrent Language. In *PLDI*. 293–305.

[55] Alceste Scalas and Nobuko Yoshida. 2018. *Less is More: Multiparty Session Types Revisited*. DoC Technical Report DTRS-18-6. Imperial College London.

[56] Alceste Scalas and Nobuko Yoshida. 2019. Less is More: Multiparty Session Types Revisited. *Proc. ACM Program. Lang.* 3, POPL, Article 30 (Jan. 2019), 29 pages. https://doi.org/10.1145/3290343

[57] Alceste Scalas, Nobuko Yoshida, and Elias Benussi. 2019. Verifying Message-Passing Programs with Dependent Behavioural Types. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Phoenix, AZ, USA) *(PLDI 2019)*. Association for Computing Machinery, New York, NY, USA, 502–516. https://doi.org/10.1145/3314221.3322484

[58] Kaku Takeuchi, Kohei Honda, and Makoto Kubo. 1994. An Interaction-based Language and its Typing System. In *PARLE'94 (LNCS, Vol. 817)*. 398–413.

[59] Bent Thomsen, Lone Leth, and Tsung-Min Kuo. 1996. A Facile tutorial. In *CONCUR '96: Concurrency Theory*, Ugo Montanari and Vladimiro Sassone (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 278–298.

[60] Rob van Glabbeek. 2018. A Theory of Encodings and Expressiveness (Extended Abstract). In *Proc. of FoSSaCS (LNCS, Vol. 10803)*. 183–202. https://doi.org/10.1007/978-3-319-89366-2_10

[61] Rob van Glabbeek, Ursula Goltz, and Jens-Wolfhard Schicke. 2008. On Synchronous and Asynchronous Interaction in Distributed Systems. In *Proc. of MFCS (LNCS, Vol. 5162)*. 16–35. https://doi.org/10.1007/978-3-540-85238-4

[62] Rob van Glabbeek, Ursula Goltz, and Jens-Wolfhard Schicke-Uffmann. 2012. On Distributability of Petri Nets. In *Proc. of FoSSaCS (LNCS, Vol. 7213)*. 331–345. https://doi.org/10.1007/978-3-642-28729-9_22

[63] Rob J. van Glabbeek. 2012. Musings on Encodings and Expressiveness. In *Proc. of EXPRESS/SOS 2012 (EPTCS, Vol. 89)*. 81–98. https://doi.org/10.4204/EPTCS.89.7

[64] Rob J. van Glabbeek. 2022. Comparing the expressiveness of the π-calculus and CCS. In *Proc. of ETAPS (LNCS, Vol. 13240)*, Ilya Sergey (Ed.). Springer, 548–574. https://doi.org/10.1007/978-3-030-99336-8_20

[65] Vasco T. Vasconcelos. 2012. Fundamentals of session types. *Information and Computation* 217 (2012), 52–70. https://doi.org/10.1016/j.ic.2012.05.002

[66] Nobuko Yoshida and Lorenzo Gheri. 2020. A Very Gentle Introduction to Multiparty Session Types. In *16th International Conference on Distributed Computing and Internet Technology (LNCS, Vol. 11969)*. Springer, 73–93.

[67] Nobuko Yoshida and Ping Hou. 2024. Less is More Revisit. arXiv:2402.16741 [cs.PL] Accepted by Cliff B. Jones Festschrift Proceeding.

[68] Nobuko Yoshida and Vasco Thudichum Vasconcelos. 2007. Language Primitives and Type Discipline for Structured Communication-Based Programming Revisited: Two Systems for Higher-Order Session Communication. *Electr. Notes Theor. Comput. Sci.* 171, 4 (2007), 73–93.