Information and Computation 298 (2024) 105164

Contents lists available at ScienceDirect

Information and Computation

journal homepage: www.elsevier.com/locate/yinco

Mixed choice in session types

Kirstin Peters [a,*], Nobuko Yoshida [b,*]

[a] Universität Augsburg, Germany
[b] University of Oxford, United Kingdom

ARTICLE INFO

Article history:
Received 3 March 2023
Received in revised form 18 March 2024
Accepted 21 March 2024
Available online 26 March 2024

Keywords:
Session types
Mixed choice
Expressive power

ABSTRACT

Session types provide a flexible programming style for structuring interaction, and are used to guarantee a safe and consistent composition of distributed processes. Traditional session types include only one-directional input (external) and output (internal) guarded choices. This prevents the session-processes to explore the full expressive power of the π-calculus where mixed choice was proved more expressive. Recently Casal, Mordido, and Vasconcelos proposed binary session types with mixed choices (CMV^+). Surprisingly, in spite of an inclusion of unrestricted channels with mixed choice, CMV^+'s mixed choice is rather separate and not mixed. We prove this negative result using two methodologies (using either the leader election problem or a synchronisation pattern as distinguishing feature), showing that there exists no good encoding from the π-calculus into CMV^+, preserving distribution. We then close their open problem on the encoding from CMV^+ into CMV (without mixed choice), proving its soundness.

1. Introduction

Starting with the landmark result by Palamidessi in [28] and followed up by results such as [27,29,17,33,36,37] it was shown that the key to the expressive power of the full π-calculus in comparison to its sub-calculi such as e.g. the asynchronous π-calculus is *mixed choice*.

Mixed choice in the π-calculus is a choice construct that allows to choose between inputs and outputs. In contrast, e.g. *separate choices* are constructed from either only inputs or only outputs. The additional expressive power of mixed choice relies on its ability to rule out alternative options of the opposite nature, i.e., a term can rule out its possibility to perform an input by doing an output, whereas without mixed choice inputs can rule out alternative inputs only and outputs may rule out only alternative outputs.

To compare calculi with different variants of choice, we try to build an encoding or show that no such encoding exists [3,30]. The existence of an encoding that satisfies relevant criteria shows that the target language is expressive enough to emulate the behaviours in the source language. Gorla [17] and others [30,39] introduced and classified a set of general criteria for encodability which are syntax-agnostic [17,39]: they are now commonly used for claiming expressiveness of a given calculus, defining important features which a "good encoding" should satisfy. These include *compositionality* (homomorphism), *name invariance* (bijectional renaming), sound and complete *operational correspondence* (the source and target can simulate each other), *divergence reflection* (the target diverges only if the source diverges), *observability* (barb-sensitiveness), and *distributability* preservation (the target has the same degree of distribution as the source). Conversely, a *separation result*,

* Corresponding authors.
E-mail addresses: kirstin.peters@uni-a.de (K. Peters), nobuko.yoshida@cs.ox.ac.uk (N. Yoshida).

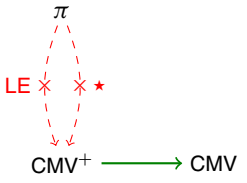
https://doi.org/10.1016/j.ic.2024.105164

i.e., the proof of the absence of an encoding with certain criteria, shows that the source language can represent behaviours that can*not* be expressed in the target. This paper gives a fresh look at expressiveness of typed π-calculi, focusing on choice constructs of *session types*.

Session types [20,44] specify and constrain the communication behaviour as a protocol between components in a system. A session type system excludes any non-conforming behaviour, statically preventing type and communication errors (i.e., mismatch of choice labels). Several languages now have session-type support via libraries and tools [43,1]. As the origin of session types is Linear Logic [18], traditional session types include only one-directional input (external) and output (internal) guarded choices. To explore the full expressiveness of mixed choice from the π-calculus, recently Casal, Mordido, and Vasconcelos proposed the binary session types with mixed choices called *mixed sessions* [7]. We denote their calculus by CMV^+. Mixed sessions include a mixture of branchings (labelled input choices) and selections (labelled output choices) at the same *linear* channel or *unrestricted* channel. This extension gives us many useful and typable *structured* concurrent programming idioms which consist of both unrestricted and linear non-deterministic choice behaviours. We show that in spite of its practical relevance, mixed sessions in CMV^+ are *strictly less expressive* than mixed choice in the π-calculus even with an unrestricted usage of choice channels.

This result surprised us. Accordingly, our motivation for this work is to understand the reasons behind this limitation in the expressive power. We would have expected that using mixed choice with an unrestricted choice channel results into a choice construct comparable to choice in the π-calculus. But, as we show in the following, mixed choice in CMV^+ cannot express essential features of mixed choice in the π-calculus. First we observe that mixed sessions are not expressive enough to solve leader election in symmetric networks. Remember that it was leader election in symmetric networks that was used to show that mixed choice is more expressive than separate choice in the π-calculus (see [28]). Second we observe that mixed sessions cannot express the synchronisation pattern $\star$. Synchronisation patterns were introduced in [39] to capture the amount of synchronisation that can be expressed in distributed systems. The synchronisation pattern $\star$ was identified in [39] as capturing exactly the amount of synchronisation introduced with mixed choice in the π-calculus. Finally, we have a closer look at the encoding from CMV^+ into CMV presented in [7]. CMV is the variant of session types that is extended in [7] with a mixed-choice-construct in order to obtain CMV^+, i.e., CMV has traditional branching and selection but not their mix. As it is the case for many variants of session types, CMV can express separate choice but has no construct for mixed choice. By analysing this encoding, we underpin our claim that mixed choice in CMV^+ is not more expressive than separate choice in the π-calculus.

Our contributions are summarised in the picture on the right. In §3 we prove that there exists no good encoding from the π-calculus (with mixed choice) into CMV^+, where we use the leader election problem by Palamidessi in [28] (LE) as distinguishing feature (the first - - ×- →). In §4 we reprove this result using the *synchronisation pattern* $\star$ from [39] instead as distinguishing feature (the second - - ×- →). Then we prove soundness of the encoding presented in [7] closing their open problem in §5 (⟶). By this encoding source terms in CMV^+ and their literal translations in CMV are related by *coupled similarity* [32], i.e., CMV^+ is encoded into CMV up to coupled similarity. From the separation results in §3 and §4 and the encoding into session types with separate choice in §5 we conclude that *mixed sessions in [7] can express only separate choice.*



The current paper extends the paper [41] presented at the workshop EXPRESS/SOS'22. The main differences are: (1) We present additional technical material in §2 including complete definitions of the three considered languages and the type systems of CMV^+ and CMV. (2) We also extend the definitions of encodability criteria in §2, introduce the notion of distributability from [39] and recall some useful auxiliary results on distributability. (3) In §3 we provide additional information on the presented counterexample and the missing proofs. (4) Similarly, we provide the missing proofs in §4. Moreover, we add the formal definition of the synchronisation pattern **M**, which captures exactly the amount of synchronisation of separate choice in the π-calculus, and show that the presented **M** in CMV^+ is well-typed. To prove the final result of this section, i.e., that there is no good an distributability preserving encoding from the π-calculus into CMV^+, we also prove that any such encoding would need to translate the conflicts given in the counterexample to conflicts in the translation and at least one of these conflicts has to split up. (5) In §5 we additionally present the encoding of [7] and explain it and its non-deterministic choices, as they are an integral part of the encoding. Again we add the missing proofs.

2. Technical preliminaries: mixed sessions and encodability criteria

A *process calculus* is a language $\mathcal{L} = (\mathcal{P}, \longmapsto)$ that consists of a set of process terms $\mathcal{P}$ (its syntax) and a relation $\longmapsto \subseteq \mathcal{P} \times \mathcal{P}$ on process terms (its reduction semantics), typically building upon some structural congruence $\equiv \subseteq \mathcal{P} \times \mathcal{P}$. We often refer to process terms also simply as processes or terms and use upper case letters $P, Q, R, \ldots, P', P_1, \ldots$ to range over them. Typed languages often define their syntax by a grammar defining the untyped processes $\mathcal{P}^{\mathsf{ut}}$ and a set of typing rules that define the subset of well-typed processes $\mathcal{P} \subset \mathcal{P}^{\mathsf{ut}}$ of the language.

Assume a countably-infinite set $\mathcal{N}$, whose elements are called *names*. We use lower case letters such as $a, b, c, \ldots, a', a_1, \ldots$ to range over names. For the π-calculus we additionally assume a set $\{\overline{y} \mid y \in \mathcal{N}\}$ of *co-names*. Let $\tau \notin \mathcal{N} \cup \{\overline{y} \mid y \in \mathcal{N}\}$. The typed languages considered here intuitively distinguish names into *session channels*, ranged over by $x, y, \ldots$, and name

variables, ranged over by $z, \ldots$. There is, however, no need to formally distinguish between different kinds of names. We also assume a set of type variables, ranged over by $t, t', \ldots$, and a set of process variables, ranged over by $X, X', \ldots$.

The *syntax* of a process calculus is usually defined by a context-free grammar defining operators, i.e., functions $\mathrm{op} : \mathcal{N}^n \times \mathcal{P}^m \to \mathcal{P}$. An operator of arity 0, i.e., $m = 0$, is a *constant*. The arguments that are again process terms are called *subterms* of P.

Definition 2.1 *(Subterms).* Let $(\mathcal{P}, \longmapsto)$ be a process calculus and $P \in \mathcal{P}$. The set of *subterms* of $P = \mathrm{op}\,(x_1, \ldots, x_n, P_1, \ldots, P_m)$ is defined recursively as:

$$\{P\} \cup \left\{ P' \mid \exists i \in \{1, \ldots, m\}.\ P' \text{ is a subterm of } P_i \right\}$$

With Definition 2.1, every term is a subterm of itself; constants have no further subterms. Among other operators, languages may introduce *action prefixes* and *conditionals* as indicated with the concrete languages below. Terms that appear as subterm of a term with some action prefix or conditional as outermost operator are called *guarded*, because the guarded subterm can not be executed before the guarding action or conditional has been reduced.

Following [7] *expressions*, ranged over by $e, e', \ldots$, are constructed from variables, unit, and standard boolean operators. We assume a partial evaluation function $\mathsf{eval}(\cdot)$ that evaluates expressions to *values*, ranged over by $v, v', \ldots$.

Definition 2.2 *(Values).* The set of *values* is given as

$$v ::= x \mid \mathsf{true} \mid \mathsf{false} \mid () \qquad \text{Values}$$

where x is a variable and $()$ is the unit term.

A *scope* defines an area in which a particular name is known and can be used. For several reasons, it can be useful to restrict the scope of a name. For instance to forbid interaction between two processes or with an unknown and, hence, potentially untrusted environment. Names whose scope is restricted such that they cannot be used beyond their scope are called *bound names*. The remaining names are called *free names*. Let $\mathsf{fn}(P)$ denote the set of free names of P. In the case of bound names, their syntactical representation as lower case letters serves as a place holder for any fresh name, i.e., any name that does not occur elsewhere in the term. To avoid confusion between free and bound names or different bound names, bound names can be replaced with fresh bound names by *α-conversion* $\equiv_\alpha$.

We assume that the *semantics* is given as an *operational semantics* consisting of inference rules defined on the operators of the language [42]. For many process calculi, the semantics is provided in two forms, as *reduction semantics* and as *labelled transition semantics*. We assume that at least the reduction semantics $\longmapsto$ is given as part of the definition, because its treatment is easier in the context of encodings. A single application of the reduction semantics is called a *(reduction) step* and is written as $P \longmapsto P'$. If $P \longmapsto P'$, then P' is called *derivative* of P. Let $P \longmapsto$ (or $P \not\longmapsto$) denote the existence (absence) of a step from P, and let $\Longmapsto$ denote the reflexive and transitive closure of $\longmapsto$. A sequence of reduction steps is called a *reduction*. We write $P \longmapsto^\omega$ if P has an infinite sequence of steps. We also use *execution* to refer to a reduction starting from a particular term. A process that cannot reduce is called *stuck*.

A substitution σ is a finite mapping from names to names defined by a set of renamings of the form $\{y_1/x_1, \ldots, y_n/x_n\} = \{y_1,\ldots,y_n/x_1,\ldots,x_n\}$, where we assume that the $x_1, \ldots, x_n$ are pairwise distinct. The application $P\{y_1/x_1, \ldots, y_n/x_n\}$ of a substitution on a term is defined as the result of simultaneously replacing all free occurrences of x_i by y_i for $i \in \{1, \ldots, n\}$, possibly applying α-conversion to avoid capture or name clashes. For all names in $\mathcal{N} \setminus \{x_1, \ldots, x_n\}$ the substitution behaves as the identity mapping. We naturally extend substitution to the substitution of name variables by values and type variables by types. In these cases we often denote substitution as the instantiation of the variable by the respective value or type.

To simplify the presentation, we sometimes treat sequences $\tilde{x} = x_1, \ldots, x_n$ as sets and apply set operations (such as union) on sequences.

To reason about environments of terms, we use functions on process terms called contexts. More precisely, a *context* $\mathcal{C}([\cdot]_1, \ldots, [\cdot]_{n+m}) : \mathcal{N}^n \times \mathcal{P}^m \to \mathcal{P}$ with $n + m$ holes is a function from n names and m terms into a term, i.e., the term $\mathcal{C}(x_1, \ldots, x_n, P_1, \ldots, P_m)$ is the result of inserting $x_1, \ldots, x_n, P_1, \ldots, P_m$ in the corresponding order into the holes of $\mathcal{C}$.

We use *barbs* or observables to reason about and to compare the behaviour of processes. We write $P{\downarrow_\beta}$ if P emits the barb β, where this definition is language specific, i.e., implemented slightly differently in the considered languages. In all considered languages P reaches a barb β, denoted as $P{\Downarrow_\beta}$, if there is some P' such that $P \Longmapsto P'$ and $P'{\downarrow_\beta}$.

Two terms of a language are usually compared using some kind of a behavioural simulation relation. The most commonly known behavioural simulation relation is bisimulation. A relation $\mathcal{R}$ is a bisimulation if any two related processes mutually simulate their respective sequences of steps, such that the derivatives are again related.

Definition 2.3 *(Bisimulation).* $\mathcal{R}$ is a (weak reduction, barbed) bisimulation if for each $(P, Q) \in \mathcal{R}$:

- $P \Longmapsto P'$ implies $\exists Q'.\ Q \Longmapsto Q' \wedge \left(P', Q'\right) \in \mathcal{R}$
- $Q \Longmapsto Q'$ implies $\exists P'.\ P \Longmapsto P' \wedge \left(P', Q'\right) \in \mathcal{R}$
- $P{\Downarrow_\beta}$ iff $Q{\Downarrow_\beta}$ for all barbs β

$$(\textsc{Com}_\pi)\ \overline{y}z.P + M \mid y(x).Q + N \longmapsto P \mid Q\{z/x\} \qquad (\textsc{Tau}_\pi)\ \tau.P + M \longmapsto P$$

$$(\textsc{Par}_\pi)\ \frac{P \longmapsto P'}{P \mid Q \longmapsto P' \mid Q} \qquad (\textsc{Res}_\pi)\ \frac{P \longmapsto P'}{(\nu x)P \longmapsto (\nu x)P'}$$

$$(\textsc{Struct}_\pi)\ \frac{P \equiv Q \quad Q \longmapsto Q' \quad Q' \equiv P'}{P \longmapsto P'}$$

where structural congruence $\equiv$ is the least congruence that contains α-conversion and satisfies:

$$(\nu x)\mathbf{0} \equiv \mathbf{0} \qquad (\nu x)(\nu y)P \equiv (\nu y)(\nu x)P \qquad P \mid (\nu x)Q \equiv (\nu x)(P \mid Q) \quad \text{if } x \notin \mathsf{fn}(P)$$
$$P \mid \mathbf{0} \equiv P \qquad P \mid Q \equiv Q \mid P \qquad P \mid (Q \mid R) \equiv (P \mid Q) \mid R \qquad !P \equiv P \mid !P$$

Fig. 1. Reduction Semantics ($\longmapsto$) of the π-Calculus.

Two terms are bisimilar if there exists a bisimulation that relates them. For a language $\mathcal{L}$, let $\approx_{\mathcal{L}}$ denote bisimilarity on $\mathcal{L}$.

Another interesting behavioural simulation relation is coupled similarity. It was introduced in [32] and discussed e.g. in [2]. It is strictly weaker than bisimilarity. As pointed out in [32], in contrast to bisimilarity it essentially allows for intermediate states (see §5). Each symmetric coupled simulation is a bisimulation.

Definition 2.4 *(Coupled Simulation).* A relation $\mathcal{R}$ is a (weak reduction, barbed) coupled simulation if for each $(P, Q) \in \mathcal{R}$:

- $P \Longmapsto P'$ implies $\exists Q'.\ Q \Longmapsto Q' \wedge (P', Q') \in \mathcal{R}$
- $P \Longmapsto P'$ also implies $\exists Q'.\ Q \Longmapsto Q' \wedge (Q', P') \in \mathcal{R}$
- $P \Downarrow_\beta$ implies $Q \Downarrow_\beta$ for all barbs β

Two terms are coupled similar if they are related by a coupled simulation in both directions.

2.1. The Pi-calculus with mixed choice

The π-calculus was introduced by Milner, Parrow, and Walker in [25] and is one of the most well-known process calculi. Over the time a large number of variants and extensions of the π-calculus emerged. We are relying on the variant used in [28], since we want to reuse some results and proof techniques from this paper. Accordingly, we consider a variant of the π-calculus with mixed guarded choice and replication but without matching. This variant is often called the synchronous or full π-calculus. In the following we denote this calculus simply as the π-calculus.

Definition 2.5 *(Pi-Calculus).* The set of processes $\mathcal{P}_\pi$ of the π-calculus is given by:

$$\alpha ::= y(x) \mid \overline{y}z \mid \tau \qquad \text{Prefixes}$$
$$P ::= \sum_{i \in \mathrm{I}} \alpha_i.P_i \mid (\nu x)P \mid P \mid P \mid !P \qquad \text{Processes}$$

A choice $\sum_{i\in \mathrm{I}} \alpha_i.P_i$ offers for each i in the index set I a subterm guarded by some action prefix α_i. An action prefix is either an input action $y(x)$, and output action $\overline{y}z$, or an internal action denoted as τ. We abbreviate the empty sum, i.e., $\sum_{i\in \mathrm{I}} \alpha_i.P_i$ for $\mathrm{I} = \emptyset$, by the inactive process $\mathbf{0}$. Moreover, we often write $\alpha_1.P_1 + \ldots + \alpha_n.P_n$ for a choice $\sum_{i\in\{1,\ldots,n\}} \alpha_i.P_i$. The remaining operators introduce restriction $(\nu x)P$, parallel composition $P \mid P$, and replication $!P$.

The name x is bound in P by input $y(x).P$ and restriction $(\nu x)P$. All other names are free. To simplify the presentation we often omit trailing $\mathbf{0}$. Moreover, we sometimes omit the argument of action prefixes if it is irrelevant, i.e., we write $y.P$ instead of $y(x).P$ if $x \notin \mathsf{fn}(P)$ and we write $\overline{y}.P$ instead of $\overline{y}z.P$ if for all matching receivers $y(x).Q$ we have $x \notin \mathsf{fn}(Q)$.

The semantics of the π-calculus is given by the rules in Fig. 1. Since choice is introduced as a set of summands, we naturally have commutativity and associativity of the summands. We rely on the commutativity and associativity of summands then writing choices as $\alpha_1.P_1 + \ldots + \alpha_n.P_n$.

Rule $(\textsc{Com}_\pi)$ defines communication as an interaction of an output and an input action that are composed in parallel choices. As result of the communication step the name x is substituted by the received name z in the subterm of the input. Moreover, the respective alternative summands of the two choices that contained the output and the input as well as the action prefixes of the chosen cases are removed. This step unguards the respective subterms of the output and input.

An internal step with rule $(\textsc{Tau}_\pi)$ reduces only a single choice. As result again the subterm of the internal action is unguarded and the alternative summands are removed.

Rule (PAR_π) allows a process to reduce in the context of parallel processes and (RES_π) allows the subterm of a restriction to reduce. Finally, (STRUCT_π) allows processes to reduce modulo structural congruence.

The observables or barbs of the π-calculus are given as follows.

Definition 2.6 *(Barbs, Pi-Calculus).* A process P emits an output barb $\overline{y}$, denoted as $P{\downarrow_{\overline{y}}}$, if P has an output $\overline{y}z.P'$ as unguarded subterm and if y is free in P, i.e., $y \in \mathsf{fn}(P)$. Similarly, P has an input barb y, denoted as $P{\downarrow_y}$, if P has an input $y(x).P'$ as unguarded subterm with $y \in \mathsf{fn}(P)$.

2.2. Mixed sessions

Mixed sessions are a variant of binary session types introduced by Casal, Mordido, and Vasconcelos in [7] with a choice-construct that combines prefixes for sending and receiving. We denote this language as CMV^+.

A central idea of CMV^+ (and the language CMV it is based on) is that channels are separated in two *channel endpoints* and that interaction is by two processes acting on the respective different ends of such a channel.

Definition 2.7 *(Mixed Sessions).* The set of untyped processes $\mathcal{P}^{\text{ut}}_{\mathsf{CMV}^+}$ of CMV^+ is given as:

$$
\begin{array}{rll}
P & ::= q\, y \sum_{i\in I} M_i \mid P \mid P \mid (\nu yz)P \mid \mathsf{if}\ v\ \mathsf{then}\ P\ \mathsf{else}\ P \mid \mathbf{0} & \text{Processes} \\
M & ::= \mathsf{l}{*}v.P & \text{Branches} \\
* & ::= \,! \mid ? & \text{Polarities} \\
q & ::= \mathsf{lin} \mid \mathsf{un} & \text{Qualifiers}
\end{array}
$$

A choice $q\, y \sum_{i\in I} M_i$ is declared as either linear (lin) or unrestricted (un) by the qualifier q. It proceeds on a single channel endpoint y. For every i in the index set I it offers a branch M_i. A branch $\mathsf{l}{*}v.P$ specifies a label l, a polarity $*$ (! for sending or ? for receiving), a name v (a value in output actions or a variable for input actions), and a continuation P. We abbreviate the empty sum, i.e., $q\, y \sum_{i\in I} M_i$ for $I = \emptyset$, by $\mathbf{0}$. Moreover, we often write $q\, y\, (M_1 + \ldots + M_n)$ for a choice $q\, y \sum_{i\in\{1,\ldots,n\}} M_i$. Restriction $(\nu yz)P$ binds the two channel endpoints y and z of a single channel to P. The remaining operators introduce parallel composition $P \mid P$, conditionals $\mathsf{if}\ v\ \mathsf{then}\ P\ \mathsf{else}\ P$, and inaction $\mathbf{0}$. We sometimes abbreviate $P_1 \mid \ldots \mid P_n$ by $\prod_{i\in\{1,\ldots,n\}} P_i$.

The variable x is bound in P by input branches $\mathsf{l}?x.P$ and the two endpoints of a channel x, y are bound in P by restriction $(\nu xy)P$. All other names are free.

The semantics of CMV^+ is given by the rules in Fig. 2. The commutativity and associativity of summands within choices again follows from choices being defined via a set of summands.

A conditional is reduced to its first subterm with Rule ($\text{R-IFT}_{\mathsf{CMV}^+}$) if its condition evaluates to true and to its second subterm with Rule ($\text{R-IFF}_{\mathsf{CMV}^+}$) if its condition evaluates to false. Communication is by one of the Rules ($\text{R-LINLIN}_{\mathsf{CMV}^+}$), ($\text{R-LINUN}_{\mathsf{CMV}^+}$), ($\text{R-UNLIN}_{\mathsf{CMV}^+}$), or ($\text{R-UNUN}_{\mathsf{CMV}^+}$). In all four cases, the continuation of the sender and the receiver are unguarded and in the receiver x is substituted by the received value v. These four rules differ w.r.t. the qualifiers of the involved choices. Linear choices (qualifier lin) are removed in a reduction step, whereas unrestricted choices are persistent. Rule ($\text{R-PAR}_{\mathsf{CMV}^+}$) allows a process to reduce in the context of parallel processes and ($\text{R-RES}_{\mathsf{CMV}^+}$) allows the subterm of a restriction to reduce. Finally, ($\text{R-STRUCT}_{\mathsf{CMV}^+}$) allows processes to reduce modulo structural congruence.

Definition 2.8 *(Barbs, Mixed Sessions).* The process P emits the barb y, denoted as $P{\downarrow_y}$, if P has an unguarded choice $q\, y \sum_{i\in I} M_i$ on a free channel endpoint $y \in \mathsf{fn}(P)$.

We do not distinguish between output and input barbs here, but instead have barbs on different end points of a channel.

To obtain from $\mathcal{P}^{\text{ut}}_{\mathsf{CMV}^+}$ the set $\mathcal{P}_{\mathsf{CMV}^+}$ of well-typed processes of CMV^+, a type system is introduced in [7]. The syntax of types is given as:

$$
\begin{array}{rll}
T & ::= q\#\{B_i\}_{i\in I} \mid \mathsf{end} \mid \mathsf{unit} \mid \mathsf{bool} \mid \mu t.T \mid t & \text{Types} \\
B & ::= \mathsf{l}{*}T.T & \text{Branches} \\
\# & ::= \oplus \mid \& & \text{Views} \\
\Gamma & ::= \cdot \mid \Gamma, x : T & \text{Contexts}
\end{array}
$$

A type of the form $q\#\{B_i\}_{i\in I}$ denotes a channel endpoint, where the view # is either $\oplus$ for internal choice or & for external choice. We often call it a choice type. In a branch $\mathsf{l}{*}T_1.T_2$ the type T_1 specifies the communicated value whereas T_2 is the type of the continuation. Besides channel endpoints there are types for inaction, the base types for unit and boolean, and types for recursion.

$$(\text{R-IfT}_{\text{CMV}^+})\ \text{if true then } P \text{ else } Q \longmapsto P \qquad (\text{R-IfF}_{\text{CMV}^+})\ \text{if false then } P \text{ else } Q \longmapsto Q$$

$$(\text{R-LinLin}_{\text{CMV}^+})\ \frac{(\nu yz)(\text{lin } y\,(\text{l}!v.P + M) \mid \text{lin } z\,(\text{l}?x.Q + N) \mid R) \longmapsto}{(\nu yz)(P \mid Q\{^v/_x\} \mid R)}$$

$$(\text{R-LinUn}_{\text{CMV}^+})\ \frac{(\nu yz)(\text{lin } y\,(\text{l}!v.P + M) \mid \text{un } z\,(\text{l}?x.Q + N) \mid R) \longmapsto}{(\nu yz)(P \mid Q\{^v/_x\} \mid \text{un } z\,(\text{l}?x.Q + N) \mid R)}$$

$$(\text{R-UnLin}_{\text{CMV}^+})\ \frac{(\nu yz)(\text{un } y\,(\text{l}!v.P + M) \mid \text{lin } z\,(\text{l}?x.Q + N) \mid R) \longmapsto}{(\nu yz)(P \mid Q\{^v/_x\} \mid \text{un } y\,(\text{l}!v.P + M) \mid R)}$$

$$(\text{R-UnUn}_{\text{CMV}^+})\ \frac{(\nu yz)(\text{un } y\,(\text{l}!v.P + M) \mid \text{un } z\,(\text{l}?x.Q + N) \mid R) \longmapsto}{(\nu yz)(P \mid Q\{^v/_x\} \mid \text{un } y\,(\text{l}!v.P + M) \mid \text{un } z\,(\text{l}?x.Q + N) \mid R)}$$

$$(\text{R-Par}_{\text{CMV}^+})\ \frac{P \longmapsto P'}{P \mid Q \longmapsto P' \mid Q} \qquad (\text{R-Res}_{\text{CMV}^+})\ \frac{P \longmapsto P'}{(\nu yz)P \longmapsto (\nu yz)P'}$$

$$(\text{R-Struct}_{\text{CMV}^+})\ \frac{P \equiv Q \quad Q \longmapsto Q' \quad Q' \equiv P'}{P \longmapsto P'}$$

where structural congruence $\equiv$ is the least congruence that contains α-conversion and satisfies:

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

$$P \mid (\nu yz)Q \equiv (\nu yz)(P \mid Q) \quad \text{if } y, z \notin \text{fn}(P) \qquad (\nu wx)(\nu yz)P \equiv (\nu yz)(\nu wx)P$$

Fig. 2. Reduction Rules ($\longmapsto$) of CMV$^+$.

Following [7], we assume that the index sets I in types are not empty, that the label-polarity-pairs l$*$ are pairwise distinct in the branches of a choice type, and recursive types are contractive, i.e., contain no subterm of the form $\mu t_1 \ldots \mu t_n.t_1$ with $n \geq 1$. A type variable t is bound in T by $\mu t.T$. All other type variables are free.

Type equivalence $\simeq$ is coinductively defined by the rules:

$$\text{end} \simeq \text{end} \qquad \text{unit} \simeq \text{unit} \qquad \text{bool} \simeq \text{bool}$$

$$\frac{T_i \simeq T_i' \quad U_i \simeq U_i' \quad (\forall i \in \text{I})}{q\#\{\text{l}*_i T_i.U_i\}_{i\in\text{I}} \simeq q\#\left\{\text{l}*_i T_i'.U_i'\right\}_{i\in\text{I}}} \qquad \frac{T\{^{\mu t.T}/_t\} \simeq U}{\mu t.T \simeq U} \qquad \frac{T \simeq U\{^{\mu t.U}/_t\}}{T \simeq \mu t.U}$$

Two types are dual to each other if they describe well-coordinated behaviour of the two endpoints of a channel. In particular, input is dual to output and internal choice is dual to external choice. The operator $\cdot \perp \cdot$ for type duality is defined coinductively by the rules:

$$! \perp ? \qquad ? \perp ! \qquad \oplus \perp \& \qquad \& \perp \oplus \qquad \text{end} \perp \text{end}$$

$$\frac{\# \perp \flat \quad *_i \perp \bullet_i \quad T_i \simeq T_i' \quad U_i \perp U_i' \quad (\forall i \in \text{I})}{q\#\{\text{l}*_i T_i.U_i\}_{i\in\text{I}} \perp q\flat\left\{\text{l}\bullet_i T_i'.U_i'\right\}_{i\in\text{I}}} \qquad \frac{T\{^{\mu t.T}/_t\} \perp U}{\mu t.T \perp U} \qquad \frac{T \perp U\{^{\mu t.U}/_t\}}{T \perp \mu t.U}$$

Subtyping introduces more flexibility to the usage of types. In external choices subtyping allows additional branches in the supertype; for internal choice we have the opposite. The operator $T_1 <: T_2$ (T_1 is a subtype of T_2) is defined coinductively by the rules:

$$\frac{T_2 <: T_1 \quad U_1 <: U_2}{\text{l}!T_1.U_1 <: \text{l}!T_2.U_2} \qquad \frac{T_1 <: T_2 \quad U_1 <: U_2}{\text{l}?T_1.U_1 <: \text{l}?T_2.U_2}$$

$$\text{end} <: \text{end} \qquad \text{unit} <: \text{unit} \qquad \text{bool} <: \text{bool}$$

$$\frac{\text{J} \subseteq \text{I} \quad B_j <: C_j \quad (\forall j \in \text{J})}{q\oplus\{B_i\}_{i\in\text{I}} <: q\oplus\left\{C_j\right\}_{j\in\text{J}}} \qquad \frac{\text{I} \subseteq \text{J} \quad B_i <: C_i \quad (\forall i \in \text{I})}{q\&\{B_i\}_{i\in\text{I}} <: q\&\left\{C_j\right\}_{j\in\text{J}}}$$

$$\frac{T\{^{\mu t.T}/_t\} <: U}{\mu t.T <: U} \qquad \frac{T <: U\{^{\mu t.U}/_t\}}{T <: \mu t.U}$$

The predicate $\cdot\,\text{un}$ that is defined by the rules

$$\text{end un} \qquad \text{unit un} \qquad \text{bool un} \qquad \text{un}\#\{B_i\}_{i\in\text{I}}\,\text{un} \qquad \frac{T\,\text{un}}{\mu t.T\,\text{un}}$$

identifies unrestricted types, i.e., types without an unguarded linear choice type.

$$(\text{T-Unit}_{\text{CMV}^+})\ \frac{\Gamma\ \mathsf{un}}{\Gamma \vdash () : \mathsf{unit}}$$

$$(\text{T-True}_{\text{CMV}^+})\ \frac{\Gamma\ \mathsf{un}}{\Gamma \vdash \mathsf{true} : \mathsf{bool}} \qquad (\text{T-False}_{\text{CMV}^+})\ \frac{\Gamma\ \mathsf{un}}{\Gamma \vdash \mathsf{false} : \mathsf{bool}}$$

$$(\text{T-Var}_{\text{CMV}^+})\ \frac{\Gamma_1, \Gamma_2\ \mathsf{un}}{\Gamma_1, x : T, \Gamma_2 \vdash x : T} \qquad (\text{T-Sub}_{\text{CMV}^+})\ \frac{\Gamma \vdash v : T \quad T <: U}{\Gamma \vdash v : U}$$

$$(\text{T-Out}_{\text{CMV}^+})\ \frac{\Gamma_1 \vdash v : T \quad \Gamma_2 \vdash P}{\Gamma_1 \circ \Gamma_2 \vdash \mathsf{l}!v.P : \mathsf{l}!T.U} \qquad (\text{T-In}_{\text{CMV}^+})\ \frac{\Gamma, x : T \vdash P}{\Gamma \vdash \mathsf{l}?x.P : \mathsf{l}?T.U}$$

$$(\text{T-Inact}_{\text{CMV}^+})\ \frac{\Gamma\ \mathsf{un}}{\Gamma \vdash \mathbf{0}} \qquad (\text{T-Par}_{\text{CMV}^+})\ \frac{\Gamma_1 \vdash P_1 \quad \Gamma_2 \vdash P_2}{\Gamma_1 \circ \Gamma_2 \vdash P_1 \mid P_2}$$

$$(\text{T-If}_{\text{CMV}^+})\ \frac{\Gamma_1 \vdash v : \mathsf{bool} \quad \Gamma_2 \vdash P \quad \Gamma_2 \vdash Q}{\Gamma_1 \circ \Gamma_2 \vdash \mathsf{if}\ v\ \mathsf{then}\ P\ \mathsf{else}\ Q} \qquad (\text{T-Res}_{\text{CMV}^+})\ \frac{\Gamma, x : T, y : U \vdash P \quad T \perp U}{\Gamma \vdash (\nu xy)P}$$

$$(\text{T-Choice}_{\text{CMV}^+})\ \frac{\begin{array}{c}(\Gamma_1 \circ \Gamma_2)\, q_1 \quad \Gamma_1 \vdash x : q_2 \# \{\mathsf{l}_i T_i.U_i\}_{i\in I} \quad \{\mathsf{l}_j\}_{j\in J} = \{\mathsf{l}_i\}_{i\in I} \\ \Gamma_2 + x : U_j \vdash \mathsf{l}_j v_j.P_j : \mathsf{l}_j T_j.U_j \quad (\forall j \in J)\end{array}}{\Gamma_1 \circ \Gamma_2 \vdash q_1\, x \sum_{j\in J} \mathsf{l}_j v_j.P_j}$$

Fig. 3. Typing Rules of CMV^+.

Typing contexts Γ collect assignments $x : T$ of names to their types. We extend the predicate $\cdot\,\mathsf{un}$ to a typing context Γ, by requiring that for $\Gamma\,\mathsf{un}$ all types in Γ are unrestricted. In contrast, all typing contexts are linear, denoted as $\Gamma\,\mathsf{lin}$. The operation $\cdot \circ \cdot$ allows to split a typing context into two typing contexts provided that all assignments with linear types are on distinct names. Assignments with unrestricted types can be shared by the two parts.

$$\cdot = (\cdot \circ \cdot) \qquad \frac{\Gamma_1 \circ \Gamma_2 = \Gamma \quad T\ \mathsf{un}}{\Gamma, x : T = (\Gamma_1, x : T) \circ (\Gamma_2, x : T)}$$

$$\frac{\Gamma_1 \circ \Gamma_2 = \Gamma}{\Gamma, x : \mathsf{lin}\ p = (\Gamma_1, x : \mathsf{lin}\ p) \circ \Gamma_2} \qquad \frac{\Gamma_1 \circ \Gamma_2 = \Gamma}{\Gamma, x : \mathsf{lin}\ p = \Gamma_1 \circ (\Gamma_2, x : \mathsf{lin}\ p)}$$

The operation $\cdot + \cdot$ adds a new assignment to a typing context, while ensuring that in a typing context all assignments are on pairwise distinct names and an assignment can be added to a typing context twice only if its type is unrestricted.

$$\frac{x : U \notin \Gamma}{\Gamma + x : T = \Gamma, x : T} \qquad \frac{T\ \mathsf{un}}{(\Gamma, x : T) + x : T = \Gamma, x : T}$$

A process P is well-typed if there is some typing context Γ such that the *type judgement* $\Gamma \vdash P$ can be derived from the typing rules in Fig. 3.

The typing Rules ($\text{T-Unit}_{\text{CMV}^+}$), ($\text{T-True}_{\text{CMV}^+}$), and ($\text{T-False}_{\text{CMV}^+}$) type constants. Rule ($\text{T-Var}_{\text{CMV}^+}$) checks the type of a variable against its type as stored in the typing context. With Rule ($\text{T-Sub}_{\text{CMV}^+}$) we can use subtyping in type derivations. The Rules ($\text{T-Out}_{\text{CMV}^+}$) and ($\text{T-In}_{\text{CMV}^+}$) type output and input branches. They check that the label and polarity are as described by the type and check the continuation of the branch against the continuation of the type. Note that the type U of the continuation is already captured in the type environment Γ_2 for ($\text{T-Out}_{\text{CMV}^+}$) and Γ for ($\text{T-In}_{\text{CMV}^+}$). Moreover, the type of the submitted value in output branches is checked, whereas for input branches we add a suitable assumption on the type of the variable to the typing context. Rule ($\text{T-Inact}_{\text{CMV}^+}$) checks that the typing context for inactive processes is unrestricted. Rule ($\text{T-Par}_{\text{CMV}^+}$) splits the typing context for checking the two parts of a parallel composition. A conditional is well-typed if its condition is boolean and if its two subterms are well-typed as specified by Rule ($\text{T-If}_{\text{CMV}^+}$). To check the subterm of restriction with Rule ($\text{T-Res}_{\text{CMV}^+}$), we have to add two assignments for the two endpoints of the restricted channel such that the respective types are dual. Choices are checked with Rule ($\text{T-Choice}_{\text{CMV}^+}$). It requires that the typing environment is unrestricted (un) if and only if the analysed choice is qualified as un; else both need to be linear (lin). Then the typing context needs to assign an external or internal choice type to the channel endpoint of this choice, where the qualifier in the type is lin if the choice is qualified as lin. Finally, Rule ($\text{T-Choice}_{\text{CMV}^+}$) checks all branches of the choice term against the branches of the choice type.

The language CMV^+ is composed of well-typed processes and the semantics in Fig. 2, i.e., $\text{CMV}^+ = \langle \mathcal{P}_{\text{CMV}^+}, \longmapsto \rangle$, where $\mathcal{P}_{\text{CMV}^+}$ is the well-typed fragment of $\mathcal{P}^{\mathsf{ut}}_{\text{CMV}^+}$.

The language CMV is the fragment of CMV^+ with a standard branching construct instead of mixed choice (compare to [7]).

Definition 2.9 (CMV). The set of untyped processes $\mathcal{P}^{\mathsf{ut}}_{\text{CMV}}$ replaces the choice construct of $\mathcal{P}^{\mathsf{ut}}_{\text{CMV}^+}$ by the following four constructs

$$(\text{R-LinCom}_{\text{CMV}})\ (\nu xy)(x!v.P \mid \mathsf{lin}\, y?z.Q \mid R) \longmapsto (\nu xy)(P \mid Q\{^v/_z\} \mid R)$$

$$(\text{R-UnCom}_{\text{CMV}})\ (\nu xy)(x!v.P \mid \mathsf{un}\, y?z.Q \mid R) \longmapsto (\nu xy)(P \mid Q\{^v/_z\} \mid \mathsf{un}\, y?z.Q \mid R)$$

$$(\text{R-Case}_{\text{CMV}})\ \frac{j \in \mathrm{I}}{(\nu xy)\big(x \triangleleft \mathrm{l}_j.P \mid y \triangleright \{\mathrm{l}_i : Q_i\}_{i\in\mathrm{I}} \mid R\big) \longmapsto (\nu xy)\big(P \mid Q_j \mid R\big)}$$

and structural congruence $\equiv$ as well as the Rules $(\text{R-If}_{\text{CMV}^+})$, $(\text{R-IfF}_{\text{CMV}^+})$, $(\text{R-Par}_{\text{CMV}^+})$, $(\text{R-Res}_{\text{CMV}^+})$, and $(\text{R-Struct}_{\text{CMV}^+})$ from Fig. 2.

Fig. 4. Reduction Rules ($\longmapsto$) of CMV.

$$y!v.P \mid q\, y?x.P \mid x \triangleleft \mathrm{l}.P \mid x \triangleright \{\mathrm{l}_i : P_i\}_{i\in\mathrm{I}}$$

and keeps the constructs for parallel composition, restriction, conditionals, and inaction.

The output action is implemented by $y!v.P$ and $q\, y?x.P$ implements an input action. Selection $x \triangleleft \mathrm{l}.P$ allows to select the branch with label l from a branching $x \triangleright \{\mathrm{l}_i : P_i\}_{i\in\mathrm{I}}$ provided that $\mathrm{l} \in \{\mathrm{l}_i\}_{i\in\mathrm{I}}$. Selection and branching are action prefixes.

The reduction semantics of CMV is given in Fig. 4. Instead of the four communication rules in CMV$^+$, we have two communication rules—one for a linear input and one for an unrestricted input—and a rule for branching. The remaining Rules $(\text{R-If}_{\text{CMV}})$, $(\text{R-IfF}_{\text{CMV}})$, $(\text{R-Par}_{\text{CMV}})$, $(\text{R-Res}_{\text{CMV}})$, and $(\text{R-Struct}_{\text{CMV}})$ as well as the rules of structural congruence are inherited from CMV$^+$. Also the notions of free names are inherited from CMV$^+$. The definition of barbs has to be adapted.

Definition 2.10 *(Barbs,* CMV*).* The process P emits the barb y, denoted as $P{\downarrow_y}$, if P has an unguarded output $y!v.P$ or an unguarded input $q\, y?x.P$ or an unguarded selection $y \triangleleft \mathrm{l}.P$ or an unguarded branching $y \triangleright \{\mathrm{l}_i : P_i\}_{i\in\mathrm{I}}$ on a free channel endpoint $y \in \mathsf{fn}(P)$.

The set of types of CMV replaces the choice construct in the definition of types of CMV$^+$ by the following two constructs

$$q \ast T.T \mid q\#\{\mathrm{l}_i : T_i\}_{i\in\mathrm{I}}$$

and keeps the constructs for inaction, base types, and recursion.

We adapt the typing rule that allows to compare types for choices to the simpler rule

$$\frac{T_i \simeq T_i'}{q\#\{\mathrm{l}_i : T_i\}_{i\in\mathrm{I}} \simeq q\#\big\{\mathrm{l}_i : T_i'\big\}_{i\in\mathrm{I}}}$$

and keep the remaining rules for inaction, base types, and recursion as well as the rules for type equivalence.

The following two rules replace the rule for choice in the definition of duality.

$$\frac{\bullet \perp \ast \quad T_1 \simeq T_2 \quad U_1 \perp U_2}{q \bullet T_1.U_1 \perp q \ast T_2.U_2} \qquad \frac{\# \perp \flat \quad T_i \perp U_i \quad (\forall i \in \mathrm{I})}{q\#\{\mathrm{l}_i : T_i\}_{i\in\mathrm{I}} \perp q\flat\{\mathrm{l}_i : U_i\}_{i\in\mathrm{I}}}$$

We keep the rules for polarities, views, inaction, and recursion.

The following four rules replace the subtyping rules for inputs, outputs, and choice.

$$\frac{U <: T \quad T' <: U'}{q\,!T.T' <: q\,!U.U'} \qquad \frac{T <: U \quad T' <: U'}{q\,?T.T' <: q\,?U.U'}$$

$$\frac{\mathrm{J} \subseteq \mathrm{I} \quad T_j <: U_j \quad (\forall j \in \mathrm{J})}{q\oplus\{\mathrm{l}_i : T_i\}_{i\in\mathrm{I}} <: q\oplus\big\{\mathrm{l}_j : U_j\big\}_{j\in\mathrm{J}}} \qquad \frac{\mathrm{I} \subseteq \mathrm{J} \quad T_i <: U_i \quad (\forall i \in \mathrm{I})}{q\&\{\mathrm{l}_i : T_i\}_{i\in\mathrm{I}} <: q\&\big\{\mathrm{l}_j : U_j\big\}_{j\in\mathrm{J}}}$$

We keep the subtyping rules for inaction, base types, and recursion.

The following two rules replace the rule for choice in the definition of the predicate $\cdot\,\mathsf{un}$.

$$\mathsf{un} \ast T.U\ \mathsf{un} \qquad \mathsf{un}\#\{\mathrm{l}_i : T_i\}_{i\in\mathrm{I}}\ \mathsf{un}$$

We keep the rules for inaction, base types, and recursion.

The typing rules of CMV are depicted in Fig. 5. The Rules $(\text{T-Out}_{\text{CMV}^+})$, $(\text{T-In}_{\text{CMV}^+})$, and $(\text{T-Choice}_{\text{CMV}^+})$ are replaced by the depicted rules. We inherit the remaining Rules, i.e., the Rules $(\text{T-Unit}_{\text{CMV}})$, $(\text{T-True}_{\text{CMV}})$, $(\text{T-False}_{\text{CMV}})$, $(\text{T-Var}_{\text{CMV}})$, $(\text{T-Sub}_{\text{CMV}})$, $(\text{T-Inact}_{\text{CMV}})$, $(\text{T-Par}_{\text{CMV}})$, $(\text{T-If}_{\text{CMV}})$, and $(\text{T-Res}_{\text{CMV}})$, from CMV$^+$.

Again, $\text{CMV} = \langle \mathcal{P}_{\text{CMV}}, \longmapsto \rangle$, where $\mathcal{P}_{\text{CMV}}$ is the well-typed fragment of $\mathcal{P}^{\mathsf{ut}}_{\text{CMV}}$.

$$(\text{T-OutC}_{\text{CMV}})\ \frac{\Gamma_1 \vdash x : q\,!T.U \quad \Gamma_2 + x : U = \Gamma_3 \circ \Gamma_4 \quad \Gamma_3 \vdash v : T \quad \Gamma_4 \vdash P}{\Gamma_1 \circ \Gamma_2 \vdash x!v.P}$$

$$(\text{T-InC}_{\text{CMV}})\ \frac{(\Gamma_1 \circ \Gamma_2)\, q_1 \quad \Gamma_1 \vdash x : q_2\,?T.U \quad (\Gamma_2 + x : U),\, y : T \vdash P}{\Gamma_1 \circ \Gamma_2 \vdash q_1\, x?y.P}$$

$$(\text{T-Branch}_{\text{CMV}})\ \frac{\Gamma_1 \vdash x : q\&\{\mathrm{l}_i : T_i\}_{i\in \mathrm{I}} \quad \Gamma_2 + x : T_i \vdash P_i \quad (\forall i \in \mathrm{I})}{\Gamma_1 \circ \Gamma_2 \vdash x \triangleright \{\mathrm{l}_i : P_i\}_{i\in \mathrm{I}}}$$

$$(\text{T-Sel}_{\text{CMV}})\ \frac{\Gamma_1 \vdash x : q\oplus\{\mathrm{l} : T\} \quad \Gamma_2 + x : T \vdash P}{\Gamma_1 \circ \Gamma_2 \vdash x \triangleleft \mathrm{l}.P}$$

and the Rules (T-Unit$_{\text{CMV}^+}$), (T-True$_{\text{CMV}^+}$), (T-False$_{\text{CMV}^+}$), (T-Var$_{\text{CMV}^+}$), (T-If$_{\text{CMV}^+}$), (T-Sub$_{\text{CMV}^+}$), (T-Inact$_{\text{CMV}^+}$), (T-Par$_{\text{CMV}^+}$), and (T-Res$_{\text{CMV}^+}$).

Fig. 5. Typing Rules of CMV.

2.3. Encodings, quality criteria, and distributability

Let $\mathcal{L}_{\mathrm{S}} = \langle \mathcal{P}_{\mathrm{S}}, \longmapsto_{\mathrm{S}} \rangle$ and $\mathcal{L}_{\mathrm{T}} = \langle \mathcal{P}_{\mathrm{T}}, \longmapsto_{\mathrm{T}} \rangle$ be two (untyped or typed) process calculi, denoted as *source* and *target language*. In the simplest case, an *encoding* from $\mathcal{L}_{\mathrm{S}}$ into $\mathcal{L}_{\mathrm{T}}$ is a function $[\![\cdot]\!] : \mathcal{P}_{\mathrm{S}} \to \mathcal{P}_{\mathrm{T}}$ that translates source terms into target terms. If the source and target language are typed then we assume an additional encoding function on types and allow the encoding function on terms to use information about the type of source terms. We often use $S, S', S_1, \ldots$ to range over $\mathcal{P}_{\mathrm{S}}$ and $T, T', T_1, \ldots$ to range over $\mathcal{P}_{\mathrm{T}}$. Encodings often translate single source term steps into a sequence or pomset of target term steps. We call such a sequence or pomset an *emulation* of the corresponding source term step.

Within a single calculus, systems are usually compared up to some form of simulation relation that uses the observables of the language to compare the behaviour of the systems. Comparing systems of different languages is more difficult, because they might not share the same set of observables. In order to provide a general framework, Gorla in [17] suggests five criteria well suited for language comparison, because they are language independent and as shown in [35] induce some kind of simulation relation between a source term and its literal translation. They are divided into two structural and three semantic criteria. The structural criteria include (1) *compositionality* and (2) *name invariance*. The semantic criteria include (3) *operational correspondence*, (4) *divergence reflection*, and (5) *success sensitiveness*. These criteria are well suited for *encodability* and *separation* results. An encodability result proves the existence of an encoding, where the criteria rule out trivial or meaningless encodings. A separation result separates two languages by showing that no encoding that satisfies the criteria exists, where the criteria are minimal assumptions on reasonable encodings.

The combination of the semantic criteria ensures that source terms and their literal translation are coupled similar (see [35]), where success sensitiveness (i.e., a form of testing) is used instead of observables. In this paper, we consider languages that do not have the same barbs but barbs that are similar enough to allow for comparisons, because all considered languages are based on the π-calculus. Because of that, we replace the criterion of success sensitiveness by the slightly stronger criterion of barb sensitiveness. We claim that all separation results of this paper remain valid if we replace barb sensitiveness with success sensitiveness. In this case the counterexamples need to be adapted to the reachability of success.

Note that a behavioural equivalence $\asymp$ on the target language is assumed for the definition of name invariance and operational correspondence. Its purpose is to describe the abstract behaviour of a target process, where abstract refers to the behaviour of the source term. Moreover, let $\varphi : \mathcal{N} \to \mathcal{N}^k$ be a *renaming policy*, i.e., a mapping from a (source term) name to a vector of (target term) names that can be used by encodings to split names and to reserve special names, such that no two different names are translated into overlapping vectors of names and reserved names are not confused with translated source term names.

Intuitively, an encoding is compositional if the translation of an operator is the same for all occurrences of that operator in a term. Hence, the translation of that operator can be captured by a context that is allowed in [17] to be parametrised on the free names of the respective source term.

Definition 2.11 *(Compositionality, [17]).* The encoding $[\![\cdot]\!]$ is *compositional* if, for every operator $\mathbf{op} : \mathcal{N}^n \times \mathcal{P}_{\mathrm{S}}^m \to \mathcal{P}_{\mathrm{S}}$ of $\mathcal{L}_{\mathrm{S}}$ and for every subset of names N, there exists a context $\mathcal{C}^N_{\mathbf{op}}([\cdot]_1, \ldots, [\cdot]_{n'+m}) : \mathcal{N}^{n'} \times \mathcal{P}_{\mathrm{S}}^m \to \mathcal{P}_{\mathrm{T}}$ and $y_1, \ldots, y_{n'} \in \mathcal{N}$ such that, for all $x_1, \ldots, x_n \in \mathcal{N}$ and all $S_1, \ldots, S_m \in \mathcal{P}_{\mathrm{S}}$ with $\mathsf{fn}(S_1) \cup \ldots \cup \mathsf{fn}(S_m) = N$ and $\{y_1, \ldots, y_{n'}\} \subseteq \varphi(x_1) \cup \ldots \cup \varphi(x_n)$, it holds that:

$$[\![\mathbf{op}\,(x_1, \ldots, x_n, S_1, \ldots, S_m)]\!] = \mathcal{C}^N_{\mathbf{op}}(y_1, \ldots, y_{n'}, [\![S_1]\!], \ldots, [\![S_m]\!])$$

Name invariance ensures that encodings are independent of specific names in the source. We use projection to obtain the respective elements of a translated name, i.e., if $\varphi(a) = (a_1, a_2, a_3)$ then $\varphi(a).2 = a_2$. Slightly abusing notation, we sometimes use the tuples that are generated by the renaming policy as sets, i.e., we require e.g. $\varphi(a) \cap \varphi(b) = \emptyset$ whenever $a \neq b$. An encoding is name invariant if it preserves substitutions modulo the relation $\asymp$ on the target language.

Definition 2.12 *(Name Invariance, [17]).* The encoding $[\![\cdot]\!]$ is *name invariant* w.r.t. $\asymp$ if, for every $S \in \mathcal{P}_S$ and every substitution σ, it holds that

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

where σ' is such that $\varphi(\sigma(a)) = \sigma'(\varphi(a))$ for all $a \in \mathcal{N}$.

To simplify the presentation in the workshop paper that we presented at EXPRESS/SOS'22, we omit the renaming policy and instead assumed that the names reserved by the encoding function from CMV^+ into CMV of [7] are different from all source term names. Under this assumption the encoding of [7] satisfies the variant

For every S and every substitution σ, it holds that $[\![S\sigma]\!] \asymp [\![S]\!]\sigma$.

of name invariance, that we present as name invariance criterion in our workshop paper. Indeed the purpose of the renaming policy is to allow to implement such an assumption. When we prove the correctness of the encoding from CMV^+ into CMV of [7], we consider both variants of name invariance.

The first semantic criterion is operational correspondence. It consists of a soundness and a completeness condition. *Completeness* requires that every computation of a source term can be emulated by its translation. *Soundness* requires that every computation of a target term corresponds to some computation of the corresponding source term.

Definition 2.13 *(Operational Correspondence, [17]).* The encoding $[\![\cdot]\!]$ satisfies *operational correspondence* if it satisfies:

Completeness: For all $S \Longmapsto_S S'$, it holds $[\![S]\!] \Longmapsto_T \asymp [\![S']\!]$.
Soundness: For all $[\![S]\!] \Longmapsto_T T$, there exists an S' such that $S \Longmapsto_S S'$ and $T \Longmapsto_T \asymp [\![S']\!]$.

The definition of operational correspondence relies on the equivalence $\asymp$ to get rid of junk possibly left over within computations of target terms. Sometimes, we refer to the completeness criterion of operational correspondence as *operational completeness* and, accordingly, for the soundness criterion as *operational soundness*.

The next criterion concerns the role of infinite computations in encodings.

Definition 2.14 *(Divergence Reflection, [17]).* The encoding $[\![\cdot]\!]$ *reflects divergence* if, for every source term S, $[\![S]\!] \longmapsto_T^\omega$ implies $S \longmapsto_S^\omega$.

The last criterion links the behaviour of source terms to the behaviour of their encodings.

Definition 2.15 *(Barb Sensitiveness, [35]).* The encoding $[\![\cdot]\!]$ is *barb-sensitive* if, for every source term S and every barb y, $S \Downarrow_y$ iff $[\![S]\!] \Downarrow_y$.

This criterion only links the behaviours of source terms and their literal translations, but not of their derivatives. To do so, Gorla relates success sensitiveness and operational correspondence by requiring that the equivalence on the target language $\asymp$ never relates two processes with different success behaviours. Similarly, we require that $\asymp$ respects barbs.

Definition 2.16 *(Barb Respecting).* $\asymp$ is *barb respecting* if, for every P and Q and every barb y with $P \Downarrow_y$ and $Q \not\Downarrow_y$, it holds that $P \not\asymp Q$.

According to [17] a "good" equivalence $\asymp$ is often defined in the form of a barbed equivalence (as described e.g. in [26]) or can be derived directly from the reduction semantics and is often a congruence, at least with respect to parallel composition. For the separation results presented in this paper, we require only that $\asymp$ is a barb respecting reduction bisimulation.

Since [7] considers an encoding between two typed languages, they use an additional criterion, called *type soundness*. It requires that a source term that is well-typed w.r.t. some type environment Γ is translated into a target term that is well-typed w.r.t. the translation of Γ. We do not explicitly consider this criterion here, because it was already shown in [7] that it is satisfied for the encoding $[\![\cdot]\!]_{\mathsf{CMV}}^{\mathsf{CMV}^+}$ from CMV^+ into CMV presented in [7].

Both of the papers [7] and [28] require as additional criterion that the parallel operator is translated homomorphically. As explained in [28] this criterion was mend to ensure that encodings preserve the degree of distribution in terms. Indeed, [36] presents an encoding of the π-calculus with mixed choice into the asynchronous π-calculus without choice that respects all of the above criteria. Requiring that the degree of distribution is preserved is essential for the separation result in [28]. Unfortunately, as explained in [36,33] the homomorphic translation of the parallel operator is rather strict and rules out encodings that intuitively do preserve the degree of distribution. Because of that, [36,33,39] propose an alternative criterion

for the preservation of the degree of distribution that we will use here to strengthen our separation results. The encoding of [7] that we discuss in §5 translates the parallel operator homomorphically.

Intuitively, a distribution of a process means the extraction (or separation) of its (sequential) components and their association to different locations. In order to formalise the identification of sequential components and following [33,39], we assume for each process calculus a so-called *labelling* on the *capabilities* of processes. As capabilities we denote the parts of a term that are removed in reduction steps. The capabilities of the π-calculus are the action prefixes, where the capability of a choice is the conjunction of the prefixes of all its branches—considered as single capability. In CMV^+ the capabilities are the choices and the conditionals. Since CMV replaces the choice construct of CMV^+, the capabilities of CMV are the outputs, inputs, selections, branching, and conditionals.

The labelling has to ensure that (1) each capability has a label, (2) no label occurs more than once in a labelled term, (3) a label disappears only when the corresponding capability is reduced in a reduction step, and (4), once it has disappeared, it will not appear in the execution any more. The last three conditions are called unicity, disappearence, and persistence in [5] which defines a labelling method to establish such a labelling for processes of the π-calculus. Note that such a labelling can be derived from the syntax tree of processes possibly augmented with some information about the history of the process, as it is done in [5]. However, we assume that, once the labelling of a term is fixed, the labels are preserved by the rules of structural congruence as well as by the reduction semantics of the respective calculus. Because of replication, new subterms with fresh labels for their capabilities may arise from applications of structural congruence. Since we need the labels only to distinguish syntactically similar components of a term, and to track them alongside reductions, we do not restrict the domain of the labels nor the method used to obtain them as long as the resulting labelling satisfies the above properties for all terms and all their derivatives in the respective calculus. In order not to clutter the development with the details of labelling, we prefer to argue at the corresponding informal level. More precisely, we assume that **all** processes in the following are implicitly labelled. Remember that we need these labels only to distinguish between syntactical equivalent capabilities, e.g. to distinguish between the left and the right $\overline{y}$ in $\overline{y} \mid \overline{y}$.

Since all languages considered in this paper are based on the π-calculus, we can rely on the intuition that the parallel operator splits locations. Accordingly, a process P is distributable into $P_1, \ldots, P_n$ if and only if we have $P \equiv (\nu \tilde{y})(P_1 \mid \ldots \mid P_n)$ for $P \in \mathcal{P}_\pi$ or $P \equiv (\nu \tilde{y}\tilde{z})(P_1 \mid \ldots \mid P_n)$ for $P \in \mathcal{P}_{\mathsf{CMV}}$ or $P \in \mathcal{P}_{\mathsf{CMV}^+}$. Since we require that structural congruence preserves the labels of capabilities, P and $P_1, \ldots, P_n$ contain the same capabilities and there are no two occurrences of the same capability in $P_1, \ldots, P_n$, i.e., no label occurs twice. If P is distributable into $P_1, \ldots, P_n$ then we also say that $P_1, \ldots, P_n$ are distributable within P.

Preservation of distributability means that the target term is at least as distributable as the source term.

Definition 2.17 *(Preservation of Distributability, [39]).* An encoding $[\![\cdot]\!] : \mathcal{P}_{\mathrm{S}} \to \mathcal{P}_{\mathrm{T}}$ *preserves distributability* if for every $S \in \mathcal{P}_{\mathrm{S}}$ and for all terms $S_1, \ldots, S_n \in \mathcal{P}_{\mathrm{S}}$ that are distributable within S there are some $T_1, \ldots, T_n \in \mathcal{P}_{\mathrm{T}}$ that are distributable within $[\![S]\!]$ such that $T_i \asymp [\![S_i]\!]$ for all $1 \leq i \leq n$.

In essence, this requirement is a distributability-enhanced adaptation of operational completeness. It respects both the intuition on distribution as separation on different locations—an encoded source term is at least as distributable as the source term itself—as well as the intuition on distribution as independence of processes and their executions—implemented by $T_i \asymp [\![S_i]\!]$.

The preservation of distributability completes our set of criteria for encodings.

Definition 2.18 *(Good Encoding).* We consider an encoding $[\![\cdot]\!]$ to be *good* if it (1) is compositional, (2) is name invariant, (3) satisfies operational correspondence, (4) reflects divergence, (5) is barb-sensitive, and (6) preserves distributability. Moreover we require that the equivalence $\asymp$ is a barb respecting (weak) reduction bisimulation.

We inherit some of the machinery introduced in [39] to work with distributability. As explained in [39] there are some calculi such as the join-calculus, for that the parallel operator does not sufficiently reflect distribution. Because of that, [39] distinguishes between parallel and distributable processes or steps. Since all considered languages in this paper are based on the π-calculus, in that the parallel operator sufficiently reflects distribution, we do not need to distinguish between parallel and distributable processes or steps. If a single process can perform two different steps, then we call these steps alternative to each other, where we identify steps modulo structural congruence. Since every step in the three considered language reduces exactly one or exactly two capabilities, two steps of one process are different if they do not reduce the same set of capabilities. Two alternative steps are in *conflict*, if performing one step disables the other step, i.e., if both reduce the same capability. Otherwise they are *distributable*. For instance the reductions on the channels a and b are distributable in the term $\overline{a} \mid \overline{b} \mid a \mid b$, but they are in conflict in $\overline{a} \mid \overline{b} \mid a + b$, because the choice is reduced in both steps. Remember that two capabilities are the same only if they have the same label. Hence also the reductions in $\overline{a} \mid a \mid \overline{a} \mid a$ of the respective left output/input and right output/input on a are distributable. More precisely, two steps in the π-calculus are in conflict if they reduce the same choice, two steps in CMV^+ are in conflict if they reduce the same choice or the same conditional, and two steps in CMV are in conflict if they reduce the same output, input, selection prefix, branching prefix, or conditional. Note that reducing the same choice does not necessarily mean to reduce the same summand in this choice. Distributable steps are independent, i.e., confluent.

Lemma 2.19 *(Confluence of Distributable Steps). Let* $\mathcal{P} \in \{\mathcal{P}_\pi, \mathcal{P}_{\mathsf{CMV}^+}, \mathcal{P}_{\mathsf{CMV}}\}$, $P, P_1, P_2 \in \mathcal{P}$, *and* $P \longmapsto P_1$, $P \longmapsto P_2$ *be distributable. Then there is some* $P_3 \in \mathcal{P}$ *such that* $P_1 \longmapsto P_3$ *and* $P_2 \longmapsto P_3$, *where* $P_1 \longmapsto P_3$ *reduces exactly the same capabilities in the same way as* $P \longmapsto P_2$ *and* $P_2 \longmapsto P_3$ *reduces exactly the same capabilities in the same way as* $P \longmapsto P_1$.

Proof. By induction using structural congruence and the respective reduction rule for parallel composition. □

Accordingly, distributable steps can be applied in any order as described by the confluence lemma above. We lift the definition of conflict and distributable steps to executions, i.e., sequences of steps.

Definition 2.20 *(Distributable Executions).* Let $\langle \mathcal{P}, \longmapsto \rangle$ be a process calculus, $P \in \mathcal{P}$, and let A and B denote two executions of P. A and B are in *conflict*, if a step of A and a step of B are in conflict, else A and B are *distributable*.

As shown in [33], two executions of a term P are distributable iff P is distributable into two subterms such that each performs one of these executions.

Lemma 2.21 *(Distributable Executions, [33]). Let* $\mathcal{L} = \langle \mathcal{P}, \longmapsto \rangle$ *be a process calculus,* $P \in \mathcal{P}$, *and* $A_1, \ldots, A_n$ *a set of executions of* P. *The executions* $A_1, \ldots, A_n$ *are pairwise distributable within* P *iff* P *is distributable into* $P_1, \ldots, P_n \in \mathcal{P}$ *such that, for all* $1 \leq i \leq n$, A_i *is an execution of* P_i, *i.e., during* A_i *only capabilities of* P_i *are reduced or removed.*

Because of that, an operationally complete encoding is distributability-preserving only if it preserves the distributability of sequences of source term steps.

Lemma 2.22 *(Distributability-Preservation, [39]). An operationally complete encoding* $[\![\cdot]\!] : \mathcal{P}_S \to \mathcal{P}_T$ *that preserves distributability also preserves distributability of executions, i.e., for all source terms* $S \in \mathcal{P}_S$ *and all sets of pairwise distributable executions of* S, *there exists an emulation of each execution in this set such that all these emulations are pairwise distributable in* $[\![S]\!]$.

3. Separating mixed sessions and the Pi-calculus via leader election

The first expressiveness result on the π-calculus that focuses on mixed choice is the separation result by Palamidessi in [28,29]. This result uses the problem of leader election in symmetric networks as distinguishing feature. In contrast to [28] we have to replace the asynchronous π-calculus by CMV^+. The proofs in [28] strongly rely on the absence of choices in the asynchronous π-calculus, whereas CMV^+ does contain choices with arbitrary mixtures of inputs and outputs. Moreover, in contrast to the π-calculus communication in CMV^+ is on restricted names only. Because of that, we can reuse the main proof strategy of [28] but have to adapt the technical details in the definitions and the proofs.

Following [28] we assume that the set of names $\mathcal{N}$ contains names that identify the processes of the network and that are never used as bound names within electoral systems. For simplicity, we use natural numbers for this kind of names. A leader is announced by unguarding an output on its id. Then a network $P = (\nu \tilde{x})(P_1 \mid \ldots \mid P_k)$ in $\mathcal{P}_\pi$ or $P = (\nu \tilde{x}\tilde{y})(P_1 \mid \ldots \mid P_k)$ in $\mathcal{P}_{\mathsf{CMV}^+}$ is an *electoral system* if in every maximal execution exactly one leader is announced. We adapt the definition of electoral systems of [28] to obtain electoral systems in the π-calculus and in CMV^+.

Definition 3.1 *(Electoral System).* A network $P = (\nu \tilde{x})(P_1 \mid \ldots \mid P_k)$ in $\mathcal{P}_\pi$ or a network $P = (\nu \tilde{x}\tilde{y})(P_1 \mid \ldots \mid P_k)$ in $\mathcal{P}_{\mathsf{CMV}^+}$ is an *electoral system* if for every execution $E : P \Longmapsto P'$ there exists an extension $E' : P \Longmapsto P' \Longmapsto P''$ and some $n \in \{1, \ldots, k\}$ (the leader) such that $P'' \downarrow_n$ for all P'' with $P' \Longmapsto P''$, but $P' \not\Downarrow_m$ for any $m \in \{1, \ldots, k\}$ with $m \neq n$.

Accordingly, an electoral system in the π-calculus announces a leader by unguarding some output on n that cannot be reduced or removed, where n is the id of the leader. In CMV^+ a leader is announced by unguarding a choice on the channel n. Since n is free this choice cannot be removed. A network is an electoral system if in every maximal execution exactly one leader n is announced.

We adapt the definition of hypergraphs that are associated to a network of processes in the π-calculus defined in [28] to networks in CMV^+. The hypergraph connects the nodes $1, \ldots, k$ of the network by edges representing the free channels that they share, where we ignore the outer restrictions of the network.

Definition 3.2 *(Hypergraph).* Given a network $P = (\nu \tilde{x})(P_1 \mid \ldots \mid P_k)$ in $\mathcal{P}_\pi$ or a network $P = (\nu \tilde{x}\tilde{y})(P_1 \mid \ldots \mid P_k)$ in $\mathcal{P}_{\mathsf{CMV}^+}$, the *hypergraph* associated to P is $\mathsf{H}(P) = \langle N, X, t \rangle$ with $N = \{1, \ldots, k\}$, $X = \mathsf{fn}(P_1 \mid \ldots \mid P_n) \setminus N$, and $t(x) = \{n \mid x \in \mathsf{fn}(P_n)\}$ for each $x \in X$.

Because we ignore the outer restrictions of the network in the above definition, the hypergraphs of two structural congruent networks may be different. However, this is not crucial for our results.

Given a hypergraph $H = \langle N, X, t\rangle$, an automorphism on H is a pair $\sigma = \langle \sigma_N, \sigma_X\rangle$ such that $\sigma_N : N \to N$ and $\sigma_X : X \to X$ are permutations which preserve the type of arcs. For simplicity, we usually do not distinguish between σ_N and σ_X and simply write σ. Moreover, since σ is a substitution, we allow to apply σ on terms P, denoted as $P\sigma$. The orbit $O_\sigma(n)$ of $n \in N$ generated by σ is defined as the set of nodes in which the various iterations of σ map n, i.e., $O_\sigma(n) = \{n, \sigma(n), \ldots, \sigma^{h-1}(n)\}$, where σ^i represents the composition of σ with itself i times and $\sigma^h = \mathsf{id}$. We also adapt the notion of a symmetric system of [28] to obtain symmetric systems in the π-calculus as well as in CMV^+.

Definition 3.3 *(Symmetric System).* Consider a network $P = (\nu\tilde{x})(P_1 \mid \ldots \mid P_k)$ in $\mathcal{P}_\pi$ or a network $P = (\nu\tilde{x}\tilde{y})(P_1 \mid \ldots \mid P_k)$ in $\mathcal{P}_{\mathsf{CMV}^+}$, and let σ be an isomorphism on its associated hypergraph $\mathsf{H}(P) = \langle N, X, t\rangle$. P is *symmetric w.r.t.* σ iff $P_{\sigma(i)} \approx_\pi P_i\sigma$ or $P_{\sigma(i)} \approx_{\mathsf{CMV}^+} P_i\sigma$ for each node $i \in N$. P is *symmetric* if it is symmetric w.r.t. all the automorphisms of $\mathsf{H}(P)$.

In contrast to [28] we use bisimilarity—$\approx_\pi$ and $\approx_{\mathsf{CMV}^+}$—instead of alpha conversion in the definition of symmetry. With this weaker notion of symmetry, we compensate for the weaker criterion on distributability that we use instead of the homomorphic translation of the parallel operator. Accordingly, we also consider networks as symmetric if they behave in a symmetric way; they do not necessarily need to be structurally symmetric.

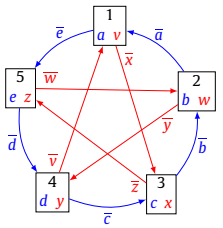
In the π-calculus we find symmetric electoral systems for many kinds of hypergraphs. We use such a solution of leader election in a network with five nodes as counterexample to separate CMV^+ from the π-calculus.

Example 3.4 *(Leader Election in the π-Calculus).* Consider the network

$$\mathsf{S}_\pi^{\mathsf{LE}} = (\nu a, b, c, d, e, v, w, x, y, z)\,(S_1 \mid S_2 \mid S_3 \mid S_4 \mid S_5)$$

where $S_1 = \overline{e} + a.(\overline{x} + v.\overline{1})$, $S_2 = \overline{a} + b.(\overline{y} + w.\overline{2})$, $S_3 = \overline{b} + c.(\overline{z} + x.\overline{3})$, $S_4 = \overline{c} + d.(\overline{v} + y.\overline{4})$, and $S_5 = \overline{d} + e.(\overline{w} + z.\overline{5})$. □

$\mathsf{S}_\pi^{\mathsf{LE}}$ is symmetric. Consider e.g. the permutation σ that permutes the channels as follows: $a \to b \to c \to d \to e \to a$, $v \to w \to x \to y \to z \to v$, and $1 \to 2 \to 3 \to 4 \to 5 \to 1$. Then $S_{\sigma(i)} = S_i\sigma$ for all $i \in \{1, \ldots, 5\}$. The network elects a leader in two stages. The first stage (depicted as blue circle) uses mixed choices on the channels a, b, c, d, e; in the second stage (depicted as a red star) we have mixed choices on the channels v, w, x, y, z. (For interpretation of the references to colour please refer to the web version of this article.) The picture on the right gives $\mathsf{H}(\mathsf{S}_\pi^{\mathsf{LE}})$ extended by arrow heads to visualise the direction of interactions and the respective action prefixes. The senders in the two stages are losing the leader election game, i.e., are not becoming the leader. In the first stage two processes can be receivers and continue with the second stage. The process that is neither sender nor receiver in the first stage is stuck and also loses. The receiver of the second stage then becomes the leader by unguarding an output on its id. The channels used by $\mathsf{S}_\pi^{\mathsf{LE}}$ in its two stages are summarised in the tabular:



Process ID	1	2	3	4	5
Input in First Stage	a	b	c	d	e
Input in Second Stage	v	w	x	y	z

Let $\tilde{n} = a, b, c, d, e, v, w, x, y, z$. The network $\mathsf{S}_\pi^{\mathsf{LE}}$ has 10 maximal executions (modulo structural congruence):

$$\mathsf{S}_\pi^{\mathsf{LE}} \longmapsto (\nu\tilde{n})(\overline{x} + v.\overline{1} \mid S_3 \mid S_4 \mid S_5) \longmapsto (\nu\tilde{n})(\overline{x} + v.\overline{1} \mid \overline{z} + x.\overline{3} \mid S_5) \longmapsto \overline{3} \mid (\nu\tilde{n})S_5 \not\longmapsto$$
$$\mathsf{S}_\pi^{\mathsf{LE}} \longmapsto (\nu\tilde{n})(\overline{x} + v.\overline{1} \mid S_3 \mid S_4 \mid S_5) \longmapsto (\nu\tilde{n})(\overline{x} + v.\overline{1} \mid S_3 \mid \overline{v} + y.\overline{4}) \longmapsto \overline{1} \mid (\nu\tilde{n})S_3 \not\longmapsto$$
$$\mathsf{S}_\pi^{\mathsf{LE}} \longmapsto (\nu\tilde{n})(S_1 \mid \overline{y} + w.\overline{2} \mid S_4 \mid S_5) \longmapsto (\nu\tilde{n})(S_1 \mid \overline{y} + w.\overline{2} \mid \overline{v} + y.\overline{4}) \longmapsto \overline{4} \mid (\nu\tilde{n})S_1 \not\longmapsto$$
$$\mathsf{S}_\pi^{\mathsf{LE}} \longmapsto (\nu\tilde{n})(S_1 \mid \overline{y} + w.\overline{2} \mid S_4 \mid S_5) \longmapsto (\nu\tilde{n})(\overline{y} + w.\overline{2} \mid S_4 \mid \overline{w} + z.\overline{5}) \longmapsto \overline{2} \mid (\nu\tilde{n})S_4 \not\longmapsto$$
$$\mathsf{S}_\pi^{\mathsf{LE}} \longmapsto (\nu\tilde{n})(S_1 \mid S_2 \mid \overline{z} + x.\overline{3} \mid S_5) \longmapsto (\nu\tilde{n})(\overline{x} + v.\overline{1} \mid \overline{z} + x.\overline{3} \mid S_5) \longmapsto \overline{3} \mid (\nu\tilde{n})S_5 \not\longmapsto$$
$$\mathsf{S}_\pi^{\mathsf{LE}} \longmapsto (\nu\tilde{n})(S_1 \mid S_2 \mid \overline{z} + x.\overline{3} \mid S_5) \longmapsto (\nu\tilde{n})(S_2 \mid \overline{z} + x.\overline{3} \mid \overline{w} + z.\overline{5}) \longmapsto \overline{5} \mid (\nu\tilde{n})S_2 \not\longmapsto$$
$$\mathsf{S}_\pi^{\mathsf{LE}} \longmapsto (\nu\tilde{n})(S_1 \mid S_2 \mid S_3 \mid \overline{v} + y.\overline{4}) \longmapsto (\nu\tilde{n})(\overline{x} + v.\overline{1} \mid S_3 \mid \overline{v} + y.\overline{4}) \longmapsto \overline{1} \mid (\nu\tilde{n})S_3 \not\longmapsto$$
$$\mathsf{S}_\pi^{\mathsf{LE}} \longmapsto (\nu\tilde{n})(S_1 \mid S_2 \mid S_3 \mid \overline{v} + y.\overline{4}) \longmapsto (\nu\tilde{n})(S_1 \mid \overline{y} + w.\overline{2} \mid \overline{v} + y.\overline{4}) \longmapsto \overline{4} \mid (\nu\tilde{n})S_1 \not\longmapsto$$
$$\mathsf{S}_\pi^{\mathsf{LE}} \longmapsto (\nu\tilde{n})(S_2 \mid S_3 \mid S_4 \mid \overline{w} + z.\overline{5}) \longmapsto (\nu\tilde{n})(\overline{y} + w.\overline{2} \mid S_4 \mid \overline{w} + z.\overline{5}) \longmapsto \overline{2} \mid (\nu\tilde{n})S_4 \not\longmapsto$$
$$\mathsf{S}_\pi^{\mathsf{LE}} \longmapsto (\nu\tilde{n})(S_2 \mid S_3 \mid S_4 \mid \overline{w} + z.\overline{5}) \longmapsto (\nu\tilde{n})(S_2 \mid \overline{z} + x.\overline{3} \mid \overline{w} + z.\overline{5}) \longmapsto \overline{5} \mid (\nu\tilde{n})S_2 \not\longmapsto$$

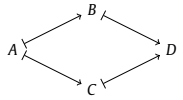
These executions can be obtained from the first execution in the above list by symmetry on the first two steps. In each maximal execution exactly one leader is elected.

We show that there exists no symmetric electoral system for networks of size five in CMV^+; or more generally no symmetric electoral system for networks of odd size in CMV^+. We discuss the relevance of odd sizes of networks after the proof of Lemma 3.6. A key ingredient to separate the π-calculus with mixed choice from the asynchronous π-calculus in [28] is a confluence lemma. It states that in the asynchronous π-calculus a step reducing an output and an alternative step reducing an input cannot conflict with each other and thus can be executed in any order. In the full π-calculus this confluence lemma is not valid, because inputs and outputs can be combined within a single choice construct and can thus be in conflict. For CMV^+ we observe that steps that reduce different endpoints can also not be in conflict to each other, because different channel endpoints cannot be combined in a single choice.

Lemma 3.5 *(Confluence). Let $P, Q \in \mathcal{P}_{\mathsf{CMV}^+}$. Assume that $A = (\nu \tilde{x}\tilde{y})(P \mid Q)$ can make two steps $A \longmapsto (\nu \widetilde{x_1}\widetilde{y_1})(P_1 \mid Q_1) = B$ and $A \longmapsto (\nu \widetilde{x_2}\widetilde{y_2})(P_2 \mid Q_2) = C$ such that P_1 is obtained modulo $\equiv$ from P by reducing a choice on channel endpoint a and P_2 is obtained modulo $\equiv$ from P by reducing a choice on channel endpoint b with $a \neq b$. Then there exist $P_3, Q_3 \in \mathcal{P}_{\mathsf{CMV}^+}$ and $D = (\nu \widetilde{x_3}\widetilde{y_3})(P_3 \mid Q_3)$ such that $B \longmapsto D$ and $C \longmapsto D$, where $\widetilde{x_3} = \widetilde{x_1} \cup \widetilde{x_2}$ and $\widetilde{y_3} = \widetilde{y_1} \cup \widetilde{y_2}$.*

Proof. Assume the two steps $A \longmapsto B$ and $A \longmapsto C$ as described above. By Fig. 2, the steps $A \longmapsto B$ and $A \longmapsto C$ imply that P contains at least two choices, one on channel a and one on channel b, that are modulo structural congruence combined in parallel (possibly surrounded by restrictions). Since the choice on a (or b) is the only choice reduced in P, another choice on the matching endpoint is reduced in Q. Regardless of whether a and b are matching endpoints or not, we obtain with the same kind of reasoning that also Q contains at least two choices, one on the channel endpoint that matches a and one on the channel endpoint that matches b, that are modulo structural congruence combined in parallel (possibly surrounded by restrictions). We conclude that the two steps of $A = (\nu \tilde{x}\tilde{y})(P \mid Q)$ are distributable. By Lemma 2.19, then these two steps can be executed in any order as required. □

The proof of this confluence lemma relies on the observation that the two steps of A to B and C have to reduce distributable parts of A. Then these two steps are distributable, which in turn allows us to perform them in any order. Thus the expressive power of choice in CMV^+ is limited by the fact that syntactically the choice construct is fixed on a single channel endpoint. With this alternative confluence lemma, we can show that there is no electoral system of odd degree in CMV^+.



Lemma 3.6 *(No Electoral System). Consider a network $P = (\nu \tilde{x}\tilde{y})(P_1 \mid \ldots \mid P_k)$ in CMV^+ with $k > 1$ being an odd number. Assume that the associated hypergraph $\mathsf{H}(P)$ admits an automorphism $\sigma \neq \mathsf{id}$ with only one orbit, and that P is symmetric w.r.t. σ. Then P cannot be an electoral system.*

Proof. Assume by contradiction that P is an electoral system. We will show that we can then construct a potentially infinite execution $E : P \Longrightarrow P^0 \Longrightarrow P^1 \Longrightarrow \ldots$ such that, for each j, $E_j : P \Longrightarrow P^j$ does not announce a unique leader and P^j is still symmetric w.r.t. σ_j, where σ_j is the original automorphism enriched with associations on the new names possibly introduced by the communication actions. This is a contradiction, because the limit of this sequence is an infinite computation for P which does not announce exactly one leader.

The proof is by induction on the current length, denoted by h, of the potentially infinite symmetric execution we have to construct. Notice that the assumption of σ generating only one orbit implies that $O_\sigma(i) = \{i, \sigma(i), \ldots, \sigma^{k-1}(i)\} = \{1, \ldots, k\}$, for each $i \in \{1, \ldots, k\}$. Since σ_j is obtained from σ by adding substitutions on restricted names and since $1, \ldots, k$ are not used as bound names in electoral systems, the same holds for all the σ_j.

Base Case ($h = 0$): Define E_0 to be the empty execution, i.e., $E_0 : P \Longrightarrow P^0$ with $P^0 = P$.

Induction Step ($h + 1$): Given $E_h : P \Longrightarrow P^h = (\nu \widetilde{x_h}\widetilde{y_h})(P_1^h \mid \ldots \mid P_k^h)$, we construct $E_{h+1} : P \Longrightarrow P^{h+1}$ as follows.

If P^h announces a leader i, then $P^h \downarrow_i$ for some $1 \leq i \leq k$. By symmetry, then $P^h \Downarrow_{\sigma(i)}$, i.e., more than one leader is announced. This is a contradiction.

Since P is an electoral system but P^h does not yet announce a leader, P^h has to be able to reduce, i.e., there is some P' such that $P^h \longmapsto P'$. This step was performed by one or two of the processes in the network, i.e., either $P_i^h \longmapsto P_i'$ and $P' = (\nu \widetilde{x_h}\widetilde{y_h})(P_1^h \mid \ldots \mid P_i' \mid \ldots \mid P_k^h)$ or $(\nu \widetilde{x_h}\widetilde{y_h})\left(P_i^h \mid P_j^h\right) \longmapsto \left(\nu \widetilde{x_{h,1}}\widetilde{y_{h,1}}\right)(P_{i,1} \mid P_{j,1})$ and $P' = \left(\nu \widetilde{x_{h,1}}\widetilde{y_{h,1}}\right)(P_1^h \mid \ldots \mid P_{i,1} \mid \ldots \mid P_{j,1} \mid \ldots \mid P_k^h)$ with $i \neq j$.

$P_i^h \longmapsto P_i'$: Regardless of whether the step $P_i^h \longmapsto P_i'$ reduces a conditional or performs a communication within part i of the network, symmetry ensures that the other parts of the network can perform a sequence of steps that leads to a state symmetric to P_i'. We choose $P_i^{h+1} = P_i'$. By symmetry, $P^h_{\sigma_h(i)} \Longrightarrow P^{h+1}_{\sigma_h(i)}, \ldots, P^h_{\sigma_h^{k-1}(i)} \Longrightarrow P^{h+1}_{\sigma_h^{k-1}(i)}$ with $P_i^{h+1}\sigma_h \approx_{\mathsf{CMV}^+} P^{h+1}_{\sigma_h(i)}, \ldots, P_i^{h+1}\sigma_h^{k-1} \approx_{\mathsf{CMV}^+} P^{h+1}_{\sigma_h^{k-1}(i)}$. Since the steps

of the different parts of the network are distributable, we obtain $E_{h+1} : P \Longmapsto P^h \Longmapsto P^{h+1}$, where $P^{h+1} = (\nu \widetilde{x_h}\widetilde{y_h})\left(P_1^{h+1} \mid \ldots \mid P_k^{h+1}\right)$ and P^{h+1} is still symmetric w.r.t. $\sigma_{h+1} = \sigma_h$.

$(\nu \widetilde{x_h}\widetilde{y_h})\left(P_i^h \mid P_j^h\right) \longmapsto (\nu \widetilde{x_{h,1}}\widetilde{y_{h,1}})\left(P_{i,1} \mid P_{j,1}\right)$**:** Let us denote this sequence of one step by S_1. A step performed by two processes of the network (in CMV^+) is a communication. By Fig. 2, S_1 reduces a choice on some endpoint a in P_i^h and a choice on some endpoint b in P_j^h such that a and b are matching endpoints of the same channel and thus $a \neq b$. By symmetry,

$$S_2 : (\nu \widetilde{x_h}\widetilde{y_h})\left(P^h_{\sigma_h(i)} \mid P^h_{\sigma_h(j)}\right) \Longmapsto (\nu \widetilde{x_{h,2}}\widetilde{y_{h,2}})\left(P_{\sigma_h(i),2} \mid P_{\sigma_h(j),2}\right)$$

$$\vdots$$

$$S_k : (\nu \widetilde{x_h}\widetilde{y_h})\left(P^h_{\sigma_h^{k-1}(i)} \mid P^h_{\sigma_h^{k-1}(j)}\right) \Longmapsto (\nu \widetilde{x_{h,k}}\widetilde{y_{h,k}})\left(P_{\sigma_h^{k-1}(i),k} \mid P_{\sigma_h^{k-1}(j),k}\right),$$

where we apply α-conversion to ensure that the pairwise intersection of elements in $\widetilde{x_{h,1}}, \ldots, \widetilde{x_{h,k}}$ is always $\widetilde{x_h}$ and similarly the pairwise intersection of elements in $\widetilde{y_{h,1}}, \ldots, \widetilde{y_{h,k}}$ is always $\widetilde{y_h}$. In the sequences of steps $S_1, \ldots, S_k$ each component of the network is used exactly twice to reduce a choice on endpoints $\sigma_h^m(a)$ and $\sigma_h^n(b)$ for some $m, n \in \{0, \ldots, k-1\}$ with $m \neq n$. Since σ_h is an automorphism with only one orbit and since k is odd, $\sigma_h^m(a) \neq \sigma_h^n(b)$ for all such cases. By repeatedly applying Lemma 3.5, then we can perform $S_1, \ldots, S_k$ in sequence, i.e., there are some $P_1^{h+1}, \ldots, P_k^{h+1}$ such that $E_{h+1} : P \Longmapsto P^h \Longmapsto P^{h+1} = (\nu \widetilde{x_{h+1}}\widetilde{y_{h+1}})\left(P_1^{h+1} \mid \ldots \mid P_k^{h+1}\right)$, where $\widetilde{x_{h+1}}$ is the union of $\widetilde{x_{h,1}}, \ldots, \widetilde{x_{h,k}}$, similarly $\widetilde{y_{h+1}}$ is the union of $\widetilde{y_{h,1}}, \ldots, \widetilde{y_{h,k}}$, the sequence $P^h \Longmapsto P^{h+1}$ is obtained from $S_1, \ldots, S_k$, and we apply scope extrusion and the Rule (R-Struct) to push restrictions to the outside. Let σ_{h+1} be the automorphism obtained from σ_h by adding permutations for the names in $\widetilde{x_{h+1}} \setminus \widetilde{x_h}$ and $\widetilde{y_{h+1}} \setminus \widetilde{y_h}$. Then $P^{h+1}_{\sigma_{h+1}(i)} \approx_{\mathsf{CMV}^+} P_i^{h+1}\sigma_{h+1}$, because P^h is symmetric and in $P^h \Longmapsto P^{h+1}$ each component of the network is used exactly twice to reduce a choice on endpoints $\sigma_h^m(a)$ and $\sigma_h^n(b)$ for some $m, n \in \{0, \ldots, k-1\}$ with $m \neq n$ such that $P^h_{\sigma_h(i)} \Longmapsto P^{h+1}_{\sigma_{h+1}(i)}, \ldots, P^h_{\sigma_h^{k-1}(i)} \Longmapsto P^{h+1}_{\sigma_{h+1}^{k-1}(i)}$. Thus P^{h+1} is still symmetric w.r.t. σ_{h+1}. □

In the proof we construct a potentially infinite sequence of steps such that the system constantly restores symmetry, i.e., whenever a step destroys symmetry we can perform a sequence of steps that restores the symmetry. Therefore we rely on the assumption of σ generating only one orbit. This implies that $O_\sigma(i) = \{i, \sigma(i), \ldots, \sigma^{k-1}(i)\} = \{1, \ldots, k\}$, for each $i \in \{1, \ldots, k\}$. Because of that, whenever part i performs a step that destroys symmetry or parts i and j together perform a step that destroys symmetry, the respective other parts of the originally symmetric network can perform symmetric steps to restore the symmetry of the network. Because of the symmetry, the constructed sequence of steps does not elect a unique leader. Accordingly, the existence of this sequence ensures that P is not an electoral system.

In contrast to [28], the above lemma is for networks of odd degree. This is necessary to ensure that in the last case of the proof the mentioned $\sigma_h^m(a)$ and $\sigma_h^n(b)$ reduced by a component of the network are distinct such that we can apply our confluence property of Lemma 3.5, which in turn ensures that we can always perform a sequence of steps to restore symmetry after the step that destroys the symmetry.

By the preservation of distributability, encodings preserve the structure of networks; and by name invariance, they also preserve the symmetry of networks. With operational correspondence and barb-sensitiveness, any good encoding of $\mathsf{S}_\pi^{\mathsf{LE}}$ is again a symmetric electoral system of size five, since the combination of these two criteria allows to distinguish between an electoral system and a system that does not elect exactly one leader in every maximal execution. Since by Lemma 3.6 this is not possible, we can separate CMV^+ from the π-calculus by using $\mathsf{S}_\pi^{\mathsf{LE}}$ from Example 3.4 as counterexample.

Theorem 3.7 *(Separate CMV^+ from the π-Calculus via Leader Election). There is no good encoding from the π-calculus into CMV^+.*

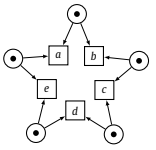
Proof. Assume the contrary, i.e., there is a good encoding $[\![\cdot]\!]$ from the π-calculus into CMV^+ with the renaming policy φ. Then this encoding translates $\mathsf{S}_\pi^{\mathsf{LE}}$ in Example 3.4. By Definition 2.17,

$$\left[\!\left[\mathsf{S}_\pi^{\mathsf{LE}}\right]\!\right] \equiv (\nu \tilde{y}\tilde{z})\left(T_{\varphi(1)} \mid T_{\varphi(2)} \mid T_{\varphi(3)} \mid T_{\varphi(4)} \mid T_{\varphi(5)}\right)$$

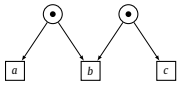
such that $T_{\varphi(i)} \asymp [\![S_i]\!]$ for all $i \in \{1, \ldots, 5\}$. Remember that $\mathsf{S}_\pi^{\mathsf{LE}}$ is symmetric. Below Example 3.4 we present an example for a permutation σ, but here we consider all automorphisms of $\mathsf{H}(\mathsf{S}_\pi^{\mathsf{LE}})$. For all such automorphisms σ we have $S_{\sigma(i)} = S_i\sigma$ for all $i \in \{1, \ldots, 5\}$. Fix σ, i.e., let σ be an arbitrary such automorphism, and let σ' be such that $\varphi(\sigma(a)) = \sigma'(\varphi(a))$ for all $a \in \mathcal{N}$. Then σ' is a permutation (on translated source term names). By Definition 2.12, then $T_{\sigma'(\varphi(i))} = T_{\varphi(\sigma(i))} \asymp [\![S_{\sigma(i)}]\!] = [\![S_i\sigma]\!] \asymp [\![S_i]\!]\sigma' \asymp T_{\varphi(i)}\sigma'$ for all $i \in \{1, \ldots, 5\}$. Since $\asymp$ is a barb respecting weak reduction

bisimulation (Definition 2.18), then $T_{\sigma'(\varphi(i))} \approx_{\mathsf{CMV}^+} T_{\varphi(i)}\sigma'$ for all $i \in \{1,\dots,5\}$ i.e., $[\![\mathsf{S}^{\mathsf{LE}}_{\pi}]\!]$ is symmetric. By the combination of Definition 2.13 and Definition 2.15, $[\![\mathsf{S}^{\mathsf{LE}}_{\pi}]\!]$ is an electoral system, because every maximal execution has to be emulated with the same reachable barbs. Then $[\![\mathsf{S}^{\mathsf{LE}}_{\pi}]\!]$ is a symmetric electoral system of size five. This contradicts Lemma 3.6. We conclude that there is no good encoding from the π-calculus into CMV^+. □

4. Separating mixed sessions and the Pi-calculus via synchronisation pattern



In [39] the technique used in [28] and its relation to synchronisation are analysed. Two synchronisation patterns, the pattern **M** and the pattern $\star$, are identified that describe two different levels of synchronisation and allow to more clearly separate languages along their ability to express synchronisation. These patterns are called **M** and $\star$, because their respective representations as a Petri net (see left and right picture) have these shapes. The pattern $\star$ captures the power of synchronisation of the π-calculus. In particular it captures what is necessary to solve the leader election problem.



The pattern **M** captures a very weak form of synchronisation, not enough to solve leader election but enough to make a fully distributed implementation of languages with this pattern difficult (see also [38]). This pattern was originally identified in [11] when studying the relevance of synchrony and distribution on Petri nets. As shown in [33,39], the ability to express these different amounts of synchronisation in the π-calculus lies in its different forms of choices: to express the pattern $\star$ the π-calculus needs mixed choice, whereas separate choice allows to express the pattern **M**. Again we have to adapt the proof technique presented in [39] to CMV^+. Indeed we find the pattern **M** in CMV^+, but there are no $\star$ in CMV^+.

We inherit the definition of the synchronisation pattern **M** from [39], where we do not distinguish between local and non-local **M** since in the π-calculus there is no difference between parallel and distributable steps.

Definition 4.1 *(Synchronisation Pattern* **M***).* Let $\langle \mathcal{P}, \longmapsto \rangle$ be a process calculus and $\mathsf{P}^{\mathbf{M}} \in \mathcal{P}$ such that:

1. $\mathsf{P}^{\mathbf{M}}$ can perform at least three alternative steps a: $\mathsf{P}^{\mathbf{M}} \longmapsto P_a$, b: $\mathsf{P}^{\mathbf{M}} \longmapsto P_b$, and c: $\mathsf{P}^{\mathbf{M}} \longmapsto P_c$ such that P_a, P_b, and P_c are pairwise different.
2. The steps a and c are parallel/distributable in $\mathsf{P}^{\mathbf{M}}$.
3. But b is in conflict with both a and c.

In this case, we denote the process $\mathsf{P}^{\mathbf{M}}$ as **M**.

There are occurrences of the pattern **M** in CMV^+ as for instance the next example.

Example 4.2 *(The* **M** *in* CMV^+*).* Consider the term $\mathsf{P}^{\mathsf{CMV}^+}_{\mathbf{M}}$ and the types $T_1 \perp T_2$ given as:

$$\begin{aligned}\mathsf{P}^{\mathsf{CMV}^+}_{\mathbf{M}} &= (\nu xy)(\mathsf{lin}\, x\,(\mathsf{l!true}.P_1 + \mathsf{l?}z.P_2) \mid \mathsf{lin}\, x\,(\mathsf{l!false}.P_3 + \mathsf{l?}z.P_4) \mid \\ &\qquad \mathsf{lin}\, y\,(\mathsf{l?}z.P_5 + \mathsf{l!true}.P_6) \mid \mathsf{lin}\, y\,(\mathsf{l?}z.P_7 + \mathsf{l!false}.P_8))\\ T_1 &= \mathsf{un}\oplus\{\mathsf{l!bool}.T_{1,1}, \mathsf{l?bool}.T_{1,2}\}\\ T_2 &= \mathsf{un}\&\{\mathsf{l?bool}.T_{2,1}, \mathsf{l!bool}.T_{2,2}\}\end{aligned}$$ □

The process $\mathsf{P}^{\mathsf{CMV}^+}_{\mathbf{M}}$ is a **M** in CMV^+:

$$\mathsf{P}^{\mathsf{CMV}^+}_{\mathbf{M}} = (\nu xy)\Big(\underbrace{\begin{array}{l}\mathsf{lin}\, x\,(\mathsf{l!true}.P_1 + \mathsf{l?}z.P_2)\\ \mathsf{lin}\, y\,(\mathsf{l?}z.P_5 + \mathsf{l!true}.P_6)\end{array}}_{\text{location 1}} \mid \underbrace{\begin{array}{l}\mathsf{lin}\, x\,(\mathsf{l!false}.P_3 + \mathsf{l?}z.P_4)\\ \mathsf{lin}\, y\,(\mathsf{l?}z.P_7 + \mathsf{l!false}.P_8)\end{array}}_{\text{location 2}} \mid \Big)$$

For instance we can pick the steps a, b, and c as:

$$\begin{aligned}&\textbf{Step } a\textbf{:} && \mathsf{P}^{\mathsf{CMV}^+}_{\mathbf{M}} \longmapsto (\nu xy)\,(P_1 \mid \mathsf{lin}\, x\,(\mathsf{l!false}.P_3 + \mathsf{l?}z.P_4) \mid P_5\{^{\mathsf{true}}/_z\} \mid \mathsf{lin}\, y\,(\mathsf{l?}z.P_7 + \mathsf{l!false}.P_8))\\ &\textbf{Step } b\textbf{:} && \mathsf{P}^{\mathsf{CMV}^+}_{\mathbf{M}} \longmapsto (\nu xy)\,(P_1 \mid \mathsf{lin}\, x\,(\mathsf{l!false}.P_3 + \mathsf{l?}z.P_4) \mid \mathsf{lin}\, y\,(\mathsf{l?}z.P_5 + \mathsf{l!true}.P_6) \mid P_7\{^{\mathsf{true}}/_z\})\\ &\textbf{Step } c\textbf{:} && \mathsf{P}^{\mathsf{CMV}^+}_{\mathbf{M}} \longmapsto (\nu xy)\,(\mathsf{lin}\, x\,(\mathsf{l!true}.P_1 + \mathsf{l?}z.P_2) \mid P_3 \mid \mathsf{lin}\, y\,(\mathsf{l?}z.P_5 + \mathsf{l!true}.P_6) \mid P_7\{^{\mathsf{false}}/_z\})\end{aligned}$$

Step a reduces in location 1 and step c in location 2. Since these two locations are composed in parallel, the steps a and c are parallel/distributable. Step b reduces on channel x in location 1 and on channel y in location 2. Thereby, it is in conflict with step a and step c. Step b disables a and c by consuming a capability the respective other step needs.

The process $\mathsf{P}_{\mathbf{M}}^{\mathsf{CMV}^+}$ with $P_1 = \ldots = P_8 = \mathbf{0}$ and $T_{1,1} = T_{1,2} = T_{2,1} = T_{2,2} = \mathsf{end}$ is well-typed:

$$D = \dfrac{\dfrac{D_1 \quad D_2 \quad D_3 \quad D_4}{x : T_1, y : T_2 \vdash \ldots}(\text{T-Par}_{\mathsf{CMV}^+})}{\vdash \mathsf{P}_{\mathbf{M}}^{\mathsf{CMV}^+}}(\text{T-Res}_{\mathsf{CMV}^+})$$

$$D_1 = \dfrac{\dfrac{\dfrac{}{x : \mathsf{end}, y : T_2 \vdash \mathsf{true} : \mathsf{bool}}(\text{T-True}_{\mathsf{CMV}^+}) \quad D_{1,1}}{x : \mathsf{end}, y : T_2 \vdash \mathsf{l!true}.\mathbf{0} : \mathsf{l!bool.end}}(\text{T-Out}_{\mathsf{CMV}^+}) \quad D_{1,2}}{x : T_1, y : T_2 \vdash \mathsf{lin}\, x\, (\mathsf{l!true}.\mathbf{0} + \mathsf{l?}z.\mathbf{0})}(\text{T-Choice}_{\mathsf{CMV}^+})$$

$$D_{1,1} = \dfrac{}{x : \mathsf{end}, y : T_2 \vdash \mathbf{0}}(\text{T-Inact}_{\mathsf{CMV}^+})$$

$$D_{1,2} = \dfrac{\dfrac{}{x : \mathsf{end}, y : T_2, z : \mathsf{bool} \vdash \mathbf{0}}(\text{T-Inact}_{\mathsf{CMV}^+})}{x : \mathsf{end}, y : T_2 \vdash \mathsf{l?}z.\mathbf{0} : \mathsf{l?bool.end}}(\text{T-In}_{\mathsf{CMV}^+})$$

where the derivations of D_2, D_3, and D_4 are similar to the derivation of D_1:

$$D_2 = \dfrac{\dfrac{\dfrac{}{x : \mathsf{end}, y : T_2 \vdash \mathsf{false} : \mathsf{bool}}(\text{T-False}_{\mathsf{CMV}^+}) \quad D_{1,1}}{x : \mathsf{end}, y : T_2 \vdash \mathsf{l!false}.\mathbf{0} : \mathsf{l!bool.end}}(\text{T-Out}_{\mathsf{CMV}^+}) \quad D_{1,2}}{x : T_1, y : T_2 \vdash \mathsf{lin}\, x\, (\mathsf{l!false}.\mathbf{0} + \mathsf{l?}z.\mathbf{0})}(\text{T-Choice}_{\mathsf{CMV}^+})$$

$$D_3 = \dfrac{D_{3,1} \quad \dfrac{\dfrac{}{x : T_1, y : \mathsf{end} \vdash \mathsf{true} : \mathsf{bool}}(\text{T-True}_{\mathsf{CMV}^+}) \quad D_{3,2}}{x : T_1, y : \mathsf{end} \vdash \mathsf{l!true}.\mathbf{0} : \mathsf{l!bool.end}}(\text{T-Out}_{\mathsf{CMV}^+})}{x : T_1, y : T_2 \vdash \mathsf{lin}\, y\, (\mathsf{l?}z.\mathbf{0} + \mathsf{l!true}.\mathbf{0})}(\text{T-Choice}_{\mathsf{CMV}^+})$$

$$D_{3,1} = \dfrac{\dfrac{}{x : T_1, y : \mathsf{end}, z : \mathsf{bool} \vdash \mathbf{0}}(\text{T-Inact}_{\mathsf{CMV}^+})}{x : T_1, y : \mathsf{end} \vdash \mathsf{l?}z.\mathbf{0} : \mathsf{l?bool.end}}(\text{T-In}_{\mathsf{CMV}^+})$$

$$D_{3,2} = \dfrac{}{x : T_1, y : \mathsf{end} \vdash \mathbf{0}}(\text{T-Inact}_{\mathsf{CMV}^+})$$

$$D_4 = \dfrac{D_{3,1} \quad \dfrac{\dfrac{}{x : T_1, y : \mathsf{end} \vdash \mathsf{false} : \mathsf{bool}}(\text{T-False}_{\mathsf{CMV}^+}) \quad D_{3,2}}{x : T_1, y : \mathsf{end} \vdash \mathsf{l!false}.\mathbf{0} : \mathsf{l!bool.end}}(\text{T-Out}_{\mathsf{CMV}^+})}{x : T_1, y : T_2 \vdash \mathsf{lin}\, y\, (\mathsf{l?}z.\mathbf{0} + \mathsf{l!false}.\mathbf{0})}(\text{T-Choice}_{\mathsf{CMV}^+})$$

We also inherit the definition of the synchronisation pattern $\star$ from [39] without distinguishing parallel and distributable steps.

Definition 4.3 *(Synchronisation Pattern $\star$).* Let $\langle \mathcal{P}, \longmapsto \rangle$ be a process calculus and $\mathsf{P}^\star \in \mathcal{P}$ such that:

- $\mathsf{P}^\star$ can perform at least five alternative reduction steps $i : \mathsf{P}^\star \longmapsto P_i$ for $i \in \{a, b, c, d, e\}$ such that the P_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 $\mathsf{P}^\star$.

In this case, we denote the process $\mathsf{P}^\star$ as $\star$.

In contrast to CMV^+ we do find $\star$ in the π-calculus.

Example 4.4 *(The $\star$ in the π-Calculus).* Consider the following $\star$ in the π-calculus:

$$\mathsf{S}_\pi^\star = \overline{a} + b.\overline{o_b} \mid \overline{b} + c.\overline{o_c} \mid \overline{c} + d.\overline{o_d} \mid \overline{d} + e.\overline{o_e} \mid \overline{e} + a.\overline{o_a}$$

The steps $a, \ldots, e$ of Definition 4.3 are the steps on the respective channels.

Step a: $\mathsf{S}_\pi^\star \longmapsto S_a$ with $S_a = \overline{b} + c.\overline{o_c} \mid \overline{c} + d.\overline{o_d} \mid \overline{d} + e.\overline{o_e} \mid \overline{o_a}$,
Step b: $\mathsf{S}_\pi^\star \longmapsto S_b$ with $S_b = \overline{o_b} \mid \overline{c} + d.\overline{o_d} \mid \overline{d} + e.\overline{o_e} \mid \overline{e} + a.\overline{o_a}$,
Step c: $\mathsf{S}_\pi^\star \longmapsto S_c$ with $S_c = \overline{a} + b.\overline{o_b} \mid \overline{o_c} \mid \overline{d} + e.\overline{o_e} \mid \overline{e} + a.\overline{o_a}$,
Step d: $\mathsf{S}_\pi^\star \longmapsto S_d$ with $S_d = \overline{a} + b.\overline{o_b} \mid \overline{b} + c.\overline{o_c} \mid \overline{o_d} \mid \overline{e} + a.\overline{o_a}$
Step e: $\mathsf{S}_\pi^\star \longmapsto S_e$ with $S_e = \overline{a} + b.\overline{o_b} \mid \overline{b} + c.\overline{o_c} \mid \overline{c} + d.\overline{o_d} \mid \overline{o_e}$

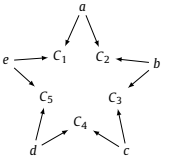
The different outputs $\overline{o_x}$ allow to distinguish between the different steps by their observables. □

We use the $\star$ $\mathsf{S}^{\star}_{\pi}$ as counterexample to show that there is no good encoding from the π-calculus into CMV^{+}. From Lemma 3.6 we learned that CMV^{+} cannot express certain electoral systems. Accordingly, we are not surprised that CMV^{+} cannot express the pattern $\star$.

Lemma 4.5. *There are no $\star$ in CMV^{+}.*

Proof. Assume the contrary, i.e., assume that there is a term $\mathsf{P}^{\star}_{\mathsf{CMV}^{+}}$ in CMV^{+} that is a $\star$. Then $\mathsf{P}^{\star}_{\mathsf{CMV}^{+}}$ can perform at least five alternative reduction steps a, b, c, d, e such that neighbouring steps in the sequence a, b, c, d, e, a are pairwise in conflict and non-neighbouring steps are distributable. Since steps reducing a conditional cannot be in conflict with any other step, none of the steps in $\{a, b, c, d, e\}$ reduces a conditional. Then all steps in $\{a, b, c, d, e\}$ are communication steps that reduce an output and an input that both are part of choices (with at least one summand). Because of the conflict between a and b, these two steps reduce the same choice but this choice is not reduced in c, because a and c are distributable.

By repeating this argument, we conclude that 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. By the reduction semantics of CMV^{+}, the two choices C_1 and C_2 that are reduced in step a need to use dual endpoints of the same channel. Without loss of generality, assume that C_1 is on channel endpoint x and C_2 is on channel endpoint y. Then the choice C_3 needs to be on channel endpoint x again, because step b reduces C_2 (on y) and C_3. By repeating this argument, then C_4 is on y and C_5 is on x. But then step e reduces two choices C_1 and C_5 that are both on channel endpoint x. Since the reduction semantics of CMV^{+} does not allow such a step, this is a contradiction.



We conclude that there are no $\star$ in CMV^{+}. □

The proof of the above lemma tells us more about why choice in CMV^{+} is limited. From the confluence property in CMV^{+} we get the hint that the problem is the restriction of choice to a single channel endpoint. A $\star$ is a circle of steps of odd degree, where neighbouring steps are in conflict. More precisely, the star with five points in $\star$ is the smallest cycle of steps where neighbouring steps are in conflict and that contains non-neighbouring distributable steps. The proof shows that the limitation of choice to a single channel endpoint and the requirement of the semantics that a channel endpoint can interact with exactly one other channel endpoint causes the problem. This also explains why Lemma 3.6 considers electoral systems of odd degree, because the odd degree does not allow to close the cycle as explained in the proof above. Indeed, if we change the syntax to allow mixed choice with summands on more than one channel, we obtain the mixed-choice-construct of the π-calculus. Similarly, we invalidate our separation result in the Theorems 3.7 and 4.8, if we change the semantics to allow two choices to communicate even if they are on the same channel. The latter may be more surprising, but indeed we do not need more than a single channel to solve leader election and build $\star$, e.g. $\mathsf{S}^{\star}_{\pi}$ remains a star if we choose $a = b = c = d = e$ (though we might want to pick different names $o_a, \ldots, o_e$ to be able to distinguish the steps).

We use $\mathsf{S}^{\star}_{\pi}$ in Example 4.4 as counterexample to separate the π-calculus from CMV^{+} in Theorem 4.8 below. We prove first that the conflicts in the source term $\mathsf{S}^{\star}_{\pi}$ have to be translated into conflicts of the corresponding emulations.

Lemma 4.6. *Any good encoding $[\![\cdot]\!]$ from the π-calculus into CMV^{+} has to translate the conflicts in $\mathsf{S}^{\star}_{\pi}$ given in Example 4.4 into conflicts of the corresponding emulations.*

Proof. By operational completeness, all five steps of $\mathsf{S}^{\star}_{\pi}$ have to be emulated in $[\![\mathsf{S}^{\star}_{\pi}]\!]$, i.e., there exist some $T_a, T_b, T_c, T_d, T_e \in \mathcal{P}_{\mathsf{CMV}^{+}}$ such that $[\![\mathsf{S}^{\star}_{\pi}]\!] \Longrightarrow T_x \asymp [\![S_x]\!]$ for all $x \in \{a, b, c, d, e\}$. Because $[\![\cdot]\!]$ preserves distributability, for each pair of steps x and y that are parallel in $\mathsf{S}^{\star}_{\pi}$, the emulations $X : [\![\mathsf{S}^{\star}_{\pi}]\!] \Longrightarrow T_x$ and $Y : [\![\mathsf{S}^{\star}_{\pi}]\!] \Longrightarrow T_y$ such that $T_x \asymp [\![S_x]\!]$ and $T_y \asymp [\![S_y]\!]$ are distributable. Note that X and Y refer to the upper case variants of x and y, respectively.

Consider each triple of steps $x, y, z \in \{a, b, c, d, e\}$ in $\mathsf{S}^{\star}_{\pi}$ such that y is in conflict with x and z but x and z are parallel. Since $[\![\cdot]\!]$ as well as $\asymp$ respect barbs, $T_x \Downarrow_{\overline{o_x}}$, $T_x \not\Downarrow_{\overline{o_y}}$, $T_y \not\Downarrow_{\overline{o_x}}$, $T_y \Downarrow_{\overline{o_y}}$, $T_y \not\Downarrow_{\overline{o_z}}$, $T_z \not\Downarrow_{\overline{o_y}}$, $T_z \Downarrow_{\overline{o_z}}$, and thus $T_x \not\asymp T_y \not\asymp T_z$. We conclude that, for all $T_x, T_y, T_z \in \mathcal{P}_{\mathsf{CMV}^{+}}$ such that $T_x \asymp [\![S_x]\!]$, $T_y \asymp [\![S_y]\!]$, and $T_z \asymp [\![S_z]\!]$ and for all sequences $X : [\![\mathsf{S}^{\star}_{\pi}]\!] \Longrightarrow T_x$, $Y : [\![\mathsf{S}^{\star}_{\pi}]\!] \Longrightarrow T_y$, and $Z : [\![\mathsf{S}^{\star}_{\pi}]\!] \Longrightarrow T_z$, there is a conflict between a step of X and a step of Y, and there is a conflict between a step of Y and a step of Z. □

Then we show that each good encoding of the counterexample $\mathsf{S}^{\star}_{\pi}$ has to distribute one of its conflicts.

Lemma 4.7. *Any good encoding $[\![\cdot]\!]$ from the π-calculus into CMV^{+} has to split up at least one of the conflicts in $\mathsf{S}^{\star}_{\pi}$ given by Example 4.4 such that there exists a maximal execution in $[\![\mathsf{S}^{\star}_{\pi}]\!]$ that emulates only one source term step.*

The proof of Lemma 4.7 is similar to the respective proof in [33,39] and can be found for the current case in the Appendix A. Its main idea is as follows: Because of operational completeness, the preservation of distributability, and Lemma 4.6, the five steps of the $\star$ in $\mathsf{S}^{\star}_{\pi}$ has to be translated into five sequences of steps with the same requirements

on conflicts and distributability. By Lemma 4.5 the conflicts between these sequences cannot be ruled out by a single step in each sequence. Then at least one of these sequences, say Y, uses two different steps to build the conflicts with its respective neighbouring sequences X and Z. These two steps have to be distributable. Then there is a maximal execution that rules out all three sequences X, Y, Z and thus emulates only one source term step.

Since each maximal execution of $\mathsf{S}^{\star}_{\pi}$ given by Example 4.4 consists of exactly two distributable steps, Lemma 4.7 violates the requirements on a good encoding.

Theorem 4.8 *(Separate* CMV^{+} *and the* π*-Calculus via* $\star$*). There is no good and distributability preserving encoding from the* π*-calculus into* CMV^{+}*.*

Proof. Assume the opposite, i.e., there is a good encoding $[\![\cdot]\!]$ from the π-calculus into CMV^{+}, and, thus, also of $\mathsf{S}^{\star}_{\pi}$ given by Example 4.4. By Lemma 4.7 there exists a maximal execution in $[\![\mathsf{S}^{\star}_{\pi}]\!]$ in which only one source term step is emulated. Let us denote this step by $x \in \{a, b, c, d, e\}$, i.e., there is a maximal execution $X : [\![\mathsf{S}^{\star}_{\pi}]\!] \Longmapsto T_x \Longmapsto \ldots$ with $T_x \asymp [\![S_x]\!]$ in that only step x is emulated. Moreover, because $[\![\cdot]\!]$ is operationally corresponding and respects barbs and because no other source term step is emulated, $T_x \downarrow_{\overline{o_x}}$ but $T_x \not\downarrow_{\overline{o_y}}$ for any $y \in \{a, b, c, d, e\}$ with $x \neq y$. Since for every S' with $\mathsf{S}^{\star}_{\pi} \Longmapsto S'$ there are at least two $i \in \{a, b, c, d, e\}$ such that $S' \Downarrow_{\overline{o_i}}$, the execution X violates the combination of the criteria operational soundness and that $[\![\cdot]\!]$ respects barbs. We conclude that there cannot be such an encoding. □

5. Encoding mixed sessions into separate choice

In [7, §7] an encoding of mixed sessions (CMV^{+}) into the variant of this session type system CMV with only separate choice (branching and selection) is presented. The proof of soundness of this encoding is missing in [7]. They suggest to prove soundness modulo "a weak form of bisimulation". As discussed below, the soundness criterion used in [7] needs to be corrected first. Prior to this discussion, we present the encoding $[\![\cdot]\!]^{\mathsf{CMV}^{+}}_{\mathsf{CMV}}$ from CMV^{+} into CMV of [7].

To describe the encoding function $[\![\cdot]\!]^{\mathsf{CMV}^{+}}_{\mathsf{CMV}}$ we reorder choices $q\,y\sum_{h\in\mathrm{H}} M_h$ into their respective send and receive actions for the same label

$$q\,y\sum_{i\in\mathrm{I}}\left(\sum_{j\in\mathrm{J}_i}\mathrm{l}_i!v_{i,j}.P_{i,j}+\sum_{k\in\mathrm{K}_i}\mathrm{l}_i?x_{i,k}.P'_{i,k}\right)$$

where $i \in \mathrm{I}$ is used to range over labels and for each label l_i the indices $j \in \mathrm{J}_i$ iterate over send branches and $k \in \mathrm{K}_i$ iterate over receive branches with this label.

The paper [7] does not explicitly mention a renaming policy, but for the encoding to work properly, we need the names c, d and u, v to be fresh. To increase readability, we omit the renaming policy and instead assume that c, d, u, v are different from all source term names. A renaming policy can implement this freshness property. Therefore, assume a renaming policy $\varphi^{\mathsf{CMV}^{+}}_{\mathsf{CMV}}(\cdot)$ that does not split names, i.e., translates a source term name by a single target term name, but that reserves the names c, d, u, v such that $\varphi^{\mathsf{CMV}^{+}}_{\mathsf{CMV}}(y) \cap \{c, d, u, v\} = \emptyset$ for all source term names y. Then replace all names n in target terms except c, d, u, v by $\varphi^{\mathsf{CMV}^{+}}_{\mathsf{CMV}}(n).1$.

We call terms junk if they are stuck and do not emit barbs, i.e., we can ignore the junk. In particular, junk is invisible modulo $\approx_{\mathsf{CMV}}$.

The encoding $[\![\cdot]\!]^{\mathsf{CMV}^{+}}_{\mathsf{CMV}}$ of [7] is then given by the Figs. 6 and 7. It relies on the predicate NDC, i.e., a non-deterministic choice

$$\mathsf{NDC}\{P_i\}_{i\in\mathrm{I}} = (\nu st)\left(s \triangleright \{\mathrm{l}_i : P_i\}_{i\in\mathrm{I}} \mid \prod_{i\in\mathrm{I}} t \triangleleft \mathrm{l}_i.\mathbf{0}\right)$$

introduced in [7] for CMV to non-deterministically choose one process from the set $\{P_i\}_{i\in\mathrm{I}}$ in a single reduction step. Let $1 \leq j \leq n$. Then choosing option j we obtain

$$\mathsf{NDC}\{P_i\}_{i\in\mathrm{I}} \longmapsto P_j \mid (\nu st)\left(\prod_{i\in\mathrm{I}\setminus\{j\}} t \triangleleft \mathrm{l}_i.\mathbf{0}\right)$$

where $(\nu st)\left(\prod_{i\in\mathrm{I}\setminus\{j\}} t \triangleleft \mathrm{l}_i.\mathbf{0}\right)$ remains as junk. Then [7] extend structural congruence $\equiv$ of CMV to $\equiv^{\mathsf{gc}}$ by adding the rule

$$(\nu yz)\left(\prod_{i\in\mathrm{I}} y \triangleleft \mathrm{l}_i.\mathbf{0}\right) \equiv^{\mathsf{gc}} \mathbf{0}$$

$$\left[\!\left[\Gamma \vdash \mathsf{lin}\, y \sum_{i\in I}\left(\sum_{j\in J_i} \mathsf{l}_i!v_{i,j}.P_{i,j} + \sum_{k\in K_i} \mathsf{l}_i?x_{i,k}.P'_{i,k}\right)\right]\!\right]^{\mathsf{CMV}^+}_{\mathsf{CMV}} =$$

$$\mathsf{NDC}\left\{ y \triangleleft \mathsf{l}_{i,!}.\mathsf{NDC}\left\{ y!v_{i,j}.\left[\!\left[\Gamma_4 \vdash P_{i,j}\right]\!\right]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \right\}_{j\in J_i},\right.$$
$$\left. y \triangleleft \mathsf{l}_{i,?}.\mathsf{NDC}\left\{ \mathsf{lin}\, y?x_{i,k}.\left[\!\left[\left(\Gamma_2 + y : U'_i\right), x_{i,k} : T'_i \vdash P'_{i,k}\right]\!\right]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \right\}_{k\in K_i} \right\}_{i\in I}$$

where $\Gamma = \Gamma_1 \circ \Gamma_2$, $\Gamma_1 \vdash y : \mathsf{lin}\oplus\{\mathsf{l}_i!T_i.U_i, \mathsf{l}_i?T'_i.U'_i\}_{i\in I}$, $\Gamma_2 + y : U_i = \Gamma_3 \circ \Gamma_4$, and $\Gamma_3 \vdash v_{i,j} : T_i$.

$$\left[\!\left[\Gamma \vdash \mathsf{lin}\, y \sum_{i\in I}\left(\sum_{j\in J_i} \mathsf{l}_i!v_{i,j}.P_{i,j} + \sum_{k\in K_i} \mathsf{l}_i?x_{i,k}.P'_{i,k}\right)\right]\!\right]^{\mathsf{CMV}^+}_{\mathsf{CMV}} =$$

$$y \triangleright \left\{ \mathsf{l}_{i,?} : \mathsf{NDC}\left\{ y!v_{i,j}.\left[\!\left[\Gamma_4 \vdash P_{i,j}\right]\!\right]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \right\}_{j\in J_i},\right.$$
$$\left. \mathsf{l}_{i,!} : \mathsf{NDC}\left\{ \mathsf{lin}\, y?x_{i,k}.\left[\!\left[\left(\Gamma_2 + y : U_i\right), x_{i,k} : T_i \vdash P'_{i,k}\right]\!\right]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \right\}_{k\in K_i} \right\}_{i\in I}$$

where $\Gamma = \Gamma_1 \circ \Gamma_2$, $\Gamma_1 \vdash y : \mathsf{lin}\&\{\mathsf{l}_i!T_i.U_i, \mathsf{l}_i?T'_i.U'_i\}_{i\in I}$, $\Gamma_2 + y : U'_i = \Gamma_3 \circ \Gamma_4$, and $\Gamma_3 \vdash v_{i,j} : T'_i$.

$$\left[\!\left[\Gamma \vdash \mathsf{lin}\, y \sum_{i\in I}\left(\sum_{j\in J_i} \mathsf{l}_i!v_{i,j}.P_{i,j} + \sum_{k\in K_i} \mathsf{l}_i?x_{i,k}.P'_{i,k}\right)\right]\!\right]^{\mathsf{CMV}^+}_{\mathsf{CMV}} =$$

$$\mathsf{NDC}\left\{ (\nu cd)\left(y!c.d \triangleleft \mathsf{l}_{i,!}.\mathsf{NDC}\left\{ d!v_{i,j}.\left[\!\left[\Gamma \vdash P_{i,j}\right]\!\right]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \right\}_{j\in J_i}\right),\right.$$
$$\left. (\nu cd)\left(y!c.d \triangleleft \mathsf{l}_{i,?}.\mathsf{NDC}\left\{ \mathsf{lin}\, d?x_{i,k}.\left[\!\left[\Gamma, x_{i,k} : T'_i \vdash P'_{i,k}\right]\!\right]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \right\}_{k\in K_i}\right) \right\}_{i\in I}$$

where $\Gamma\,\mathsf{un}$, $\Gamma \vdash y : \mu t.\mathsf{un}\oplus\{\mathsf{l}_i!T_i.t, \mathsf{l}_i?T'_i.t\}_{i\in I}$, and $\Gamma \vdash v_{i,j} : T_i$.

$$\left[\!\left[\Gamma \vdash \mathsf{lin}\, y \sum_{i\in I}\left(\sum_{j\in J_i} \mathsf{l}_i!v_{i,j}.P_{i,j} + \sum_{k\in K_i} \mathsf{l}_i?x_{i,k}.P'_{i,k}\right)\right]\!\right]^{\mathsf{CMV}^+}_{\mathsf{CMV}} =$$

$$\mathsf{lin}\, y?c.c \triangleright \left\{ \mathsf{l}_{i,?} : \mathsf{NDC}\left\{ c!v_{i,j}.\left[\!\left[\Gamma \vdash P_{i,j}\right]\!\right]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \right\}_{j\in J_i},\right.$$
$$\left. \mathsf{l}_{i,!} : \mathsf{NDC}\left\{ \mathsf{lin}\, c?x_{i,k}.\left[\!\left[\Gamma, x_{i,k} : T_i \vdash P'_{i,k}\right]\!\right]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \right\}_{k\in K_i} \right\}_{i\in I}$$

where $\Gamma\,\mathsf{un}$, $\Gamma \vdash y : \mu t.\mathsf{un}\&\{\mathsf{l}_i!T_i.t, \mathsf{l}_i?T'_i.t\}_{i\in I}$, and $\Gamma \vdash v_{i,j} : T'_i$.

Fig. 6. The Encoding $[\![\cdot]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$ from CMV^+ into CMV from [7] (Part I).

to garbage collect this kind of junk. This relation is used in [7] to prove completeness of the encoding $[\![\cdot]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$. For soundness we need something less restrictive, because the encoding allows the translation of an unrestricted choice to perform a step even if the original unrestricted choice in the source cannot be reduced (see [7]). As suggested we use $\approx_{\mathsf{CMV}}$ (as definition in Definition 2.3), i.e., a form of weak reduction barbed bisimilarity that we simply call bisimilarity in the following.

To prepare for the soundness proof, we show that steps reducing a non-deterministic choice always yield modulo bisimilarity one of its options. Here we use bisimilarity to abstract from the junk produced by reducing non-deterministic choices and also show that a non-deterministic choice can do nothing but reduce to one of its options.

Lemma 5.1. *If* $\mathsf{NDC}\{P_i\}_{i\in I} \longmapsto Q$ *then there is some* $j \in I$ *such that* $Q \approx_{\mathsf{CMV}} P_j$.

$$\left[\!\left[\Gamma \vdash \mathsf{un}\, y \sum_{i\in I}\left(\sum_{j\in J_i} \mathsf{l}_i!v_{i,j}.P_{i,j} + \sum_{k\in K_i} \mathsf{l}_i?x_{i,k}.P'_{i,k}\right)\right]\!\right]^{\mathsf{CMV}^+}_{\mathsf{CMV}} =$$

$$(\nu u v)\Bigg(u!().\mathbf{0} \mid \mathsf{un}\, v?\_.\mathsf{NDC}\Big\{ (\nu c d)\Big(y!c.d \triangleleft \mathsf{l}_{i,!}.\mathsf{NDC}\Big\{d!v_{i,j}.\Big(u!().\mathbf{0} \mid [\![\Gamma \vdash P_{i,j}]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}\Big)\Big\}_{j\in J_i}\Big),$$
$$(\nu c d)\Bigg(y!c.d \triangleleft \mathsf{l}_{i,?}.\mathsf{NDC}\Big\{\mathsf{lin}\, d?x_{i,k}.\Big(u!().\mathbf{0} \mid \Big[\!\Big[\Gamma, x_{i,k} : T'_i \vdash P'_{i,k}\Big]\!\Big]^{\mathsf{CMV}^+}_{\mathsf{CMV}}\Big)\Big\}_{k\in K_i}\Bigg)\Big\}_{i\in I}\Bigg)$$

where $\Gamma\,\mathsf{un}$, $\Gamma \vdash y : \mu t.\mathsf{un}\oplus\{\mathsf{l}_i!T_i.t, \mathsf{l}_i?T'_i.t\}_{i\in I}$, and $\Gamma \vdash v_{i,j} : T_i$.

$$\left[\!\left[\Gamma \vdash \mathsf{un}\, y \sum_{i\in I}\left(\sum_{j\in J_i} \mathsf{l}_i!v_{i,j}.P_{i,j} + \sum_{k\in K_i} \mathsf{l}_i?x_{i,k}.P'_{i,k}\right)\right]\!\right]^{\mathsf{CMV}^+}_{\mathsf{CMV}} =$$

$$(\nu u v)\Bigg(u!().\mathbf{0} \mid \mathsf{un}\, v?\_.\mathsf{lin}\, y?c.c \triangleright \Big\{ \mathsf{l}_{i,?} : \mathsf{NDC}\Big\{c!v_{i,j}.\Big(u!().\mathbf{0} \mid [\![\Gamma \vdash P_{i,j}]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}\Big)\Big\}_{j\in J_i},$$
$$\mathsf{l}_{i,!} : \mathsf{NDC}\Big\{\mathsf{lin}\, c?x_{i,k}.\Big(u!().\mathbf{0} \mid \Big[\!\Big[\Gamma, x_{i,k} : T_i \vdash P'_{i,k}\Big]\!\Big]^{\mathsf{CMV}^+}_{\mathsf{CMV}}\Big)\Big\}_{k\in K_i}\Big\}_{i\in I}\Bigg)$$

where $\Gamma\,\mathsf{un}$, $\Gamma \vdash y : \mu t.\mathsf{un}\&\{\mathsf{l}_i!T_i.t, \mathsf{l}_i?T'_i.t\}_{i\in I}$, and $\Gamma \vdash v_{i,j} : T'_i$.

$$[\![\Gamma_1 \circ \Gamma_2 \vdash P_1 \mid P_2]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} = [\![\Gamma_1 \vdash P_1]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \mid [\![\Gamma_2 \vdash P_2]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$$
$$[\![\Gamma \vdash (\nu yz)P]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} = (\nu yz)\,[\![\Gamma, y : T, z : U \vdash P]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$$
$$[\![\Gamma_1 \circ \Gamma_2 \vdash \mathsf{if}\, v\, \mathsf{then}\, P_1\, \mathsf{else}\, P_2]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} = \mathsf{if}\, v\, \mathsf{then}\, [\![\Gamma_1 \vdash P_1]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}\, \mathsf{else}\, [\![\Gamma_2 \vdash P_2]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$$
$$[\![\Gamma \vdash \mathbf{0}]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} = \mathbf{0}$$

where $T \perp U$ and $\Gamma_1 \vdash v : \mathsf{bool}$.

Fig. 7. The Encoding $[\![\cdot]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$ from CMV^+ into CMV from [7] (Part II).

Proof. By the definition of NDC, there is some $j \in I$ such that $Q = P_j \mid J$ with $J = (\nu st)\Big(\prod_{i\in I\setminus\{j\}} t \triangleleft \mathsf{l}_i.\mathbf{0}\Big)$. Since J is junk, $Q \approx_{\mathsf{CMV}} P_j$. □

The main idea of $[\![\cdot]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$ is to encode the information about whether a summand is an output or an input into the label used in branching, where a label l_i used with polarity ! in a choice typed as internal becomes $\mathsf{l}_{i,!}$ and in a choice typed as external it becomes $\mathsf{l}_{i,?}$. The dual treatment of polarities w.r.t. the type ensures that the labels of matching communication partners are translated to the same label.

Example 5.2 *(Translation).* Consider for example the term $S \in \mathcal{P}_{\mathsf{CMV}^+}$:

$$S = (\nu xy)\,(\mathsf{lin}\, y\ (\mathsf{l}!\mathsf{false}.S_1 + \mathsf{l}?z.S_2) \mid \mathsf{lin}\, x\ (\mathsf{l}!\mathsf{true}.\mathbf{0} + \mathsf{l}?z.\mathbf{0}) \mid \mathsf{lin}\, y\ (\mathsf{l}!\mathsf{false}.S_3 + \mathsf{l}?z.S_4))$$

S is well-typed but the type system forces us to assign dual types to x and y. Because of that, the choices on one channel need to be internal and on the other external. Let us assume that we have external choices on y and that the choice on x is internal. Moreover, we assume that both channels are marked as linear but typed as unrestricted. Then the translation[1] yields $[\![S]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \Longmapsto T_1$ with

[1] Note that [7] introduces a typed encoding, thus $[\![P]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$ actually means $[\![\Gamma \vdash P]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$, where $\Gamma \vdash P$ is the type statement ensuring that P is well-typed.

$$
\begin{aligned}
T_1 = (\nu xy)\big(&\mathsf{lin}\, y?c.c \triangleright \Big\{ \mathsf{l}_? : \Big(c!\mathsf{false}.[\![S_1]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \mid J_1\Big),\quad \mathsf{l}_! : \Big(\mathsf{lin}\, c?z.[\![S_2]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \mid J_2\Big)\Big\} \\
&\mid (\nu st)\big(s \triangleright \{\mathsf{l}_1 : (\nu cd)\,(x!c.d \triangleleft \mathsf{l}_!.\,(d!\mathsf{true}.\mathbf{0} \mid J_3)),\\
&\qquad\qquad \mathsf{l}_2 : (\nu cd)\,(x!c.d \triangleleft \mathsf{l}_?.\,(\mathsf{lin}\, d?z.\mathbf{0} \mid J_4))\} \\
&\qquad \mid t \triangleleft \mathsf{l}_1.\mathbf{0} \mid t \triangleleft \mathsf{l}_2.\mathbf{0}\big) \\
&\mid \mathsf{lin}\, y?c.c \triangleright \Big\{ \mathsf{l}_? : \Big(c!\mathsf{false}.\,[\![S_3]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \mid J_5\Big),\quad \mathsf{l}_! : \Big(\mathsf{lin}\, c?z.\,[\![S_4]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \mid J_6\Big)\Big\}\big)
\end{aligned}
$$

where we already performed a few steps to hide some technical details of the encoding function $[\![\cdot]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$ that are not relevant for this explanation and where the $J_1, \ldots, J_6$ remain as junk from performing these steps. We observe that in the translation of the first $\mathsf{lin}\, y$ $(\mathsf{l}!\mathsf{false}.S_1 + \mathsf{l}?z.S_2)$ in the first line of T_1 the output with label l is translated to the label $\mathsf{l}_?$ and the input with label l is translated to the label $\mathsf{l}_!$, whereas in the translation of its dual $\mathsf{lin}\, x$ $(\mathsf{l}!\mathsf{true}.\mathbf{0} + \mathsf{l}?z.\mathbf{0})$ in the second line of T_1 we obtain $\mathsf{l}_!$ for the output and $\mathsf{l}_?$ for the input. To emulate the step $S \longmapsto S_2' = (\nu xy)\,(S_2\{^{\mathsf{true}}/_z\} \mid \mathsf{lin}\, y\ (\mathsf{l}!\mathsf{false}.S_3 + \mathsf{l}?z.S_4))$ of S in which true is transmitted to S_2, we start by picking the corresponding alternative, namely l_1 for sending, in the second and third line of T_1

$$
\begin{aligned}
T_1 \longmapsto T_2 = (\nu xy)\big(&\mathsf{lin}\, y?c.c \triangleright \Big\{ \mathsf{l}_? : \Big(c!\mathsf{false}.[\![S_1]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \mid J_1\Big),\quad \mathsf{l}_! : \Big(\mathsf{lin}\, c?z.[\![S_2]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \mid J_2\Big)\Big\} \\
&\mid (\nu cd)\,(x!c.d \triangleleft \mathsf{l}_!.\,(d!\mathsf{true}.\mathbf{0} \mid J_3)) \mid J_7 \\
&\mid \mathsf{lin}\, y?c.c \triangleright \Big\{ \mathsf{l}_? : \Big(c!\mathsf{false}.\,[\![S_3]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \mid J_5\Big),\quad \mathsf{l}_! : \Big(\mathsf{lin}\, c?z.\,[\![S_4]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \mid J_6\Big)\Big\}\big)
\end{aligned}
$$

where J_7 again remains as junk. Then we perform a communication on xy, where we chose the input on y in the first line:

$$
\begin{aligned}
T_2 \longmapsto T_3 = (\nu xy)\big((\nu cd)\big(&c \triangleright \Big\{ \mathsf{l}_? : \Big(c!\mathsf{false}.[\![S_1]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \mid J_1\Big),\quad \mathsf{l}_! : \Big(\mathsf{lin}\, c?z.[\![S_2]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \mid J_2\Big)\Big\} \\
&\qquad \mid d \triangleleft \mathsf{l}_!.\,(d!\mathsf{true}.\mathbf{0} \mid J_3)\big) \mid J_7 \\
&\mid \mathsf{lin}\, y?c.c \triangleright \Big\{ \mathsf{l}_? : \Big(c!\mathsf{false}.\,[\![S_3]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \mid J_5\Big),\quad \mathsf{l}_! : \Big(\mathsf{lin}\, c?z.\,[\![S_4]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \mid J_6\Big)\Big\}\big)
\end{aligned}
$$

Finally, two more steps on cd resolve the branching and transmit true:

$$
\begin{aligned}
T_3 \longmapsto\longmapsto T_4 = (\nu xy)\big(&[\![S_2]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}\{^{\mathsf{true}}/_z\} \mid J_2 \mid J_3 \mid J_7 \mid J_8 \\
&\mid \mathsf{lin}\, y?c.c \triangleright \Big\{ \mathsf{l}_? : \Big(c!\mathsf{false}.\,[\![S_3]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \mid J_5\Big),\quad \mathsf{l}_! : \Big(\mathsf{lin}\, c?z.\,[\![S_4]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \mid J_6\Big)\Big\}\big)
\end{aligned}
$$

This completes the emulation of $S \longmapsto S_2'$, i.e., the emulation of the single source term step $S \longmapsto S_2'$ required a sequence of target term steps $[\![S]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \Longmapsto T_1 \longmapsto T_2 \longmapsto T_3 \longmapsto\longmapsto T_4$. □

The operational soundness is defined in [7] as (adapting the notation):

$$
\text{If } [\![S]\!] \longmapsto_{\mathrm{T}} T \text{ then } S \longmapsto_{\mathrm{S}} S' \text{ and } T \Longmapsto_{\mathrm{T}} \asymp [\![S']\!]. \tag{1}
$$

As visualised above, the encoding translates a single source term step into a sequence of target term steps. Unfortunately, for such encodings the statement in (1) is not strong enough: with (1), we check only that the first step on a literal translation does not introduce new behaviour. The requirement $T \Longmapsto_{\mathrm{T}} \asymp [\![S']\!]$ additionally checks that the emulation started with $[\![S]\!] \longmapsto_{\mathrm{T}} T$ can be completed, but not that there are no alternative steps introducing new behaviour. Hence we prove a correct version of soundness as defined in [17] (see Definition 2.18).

We denote the steps that reduce the first non-deterministic choice of the translation of a choice typed as internal and steps reducing a conditional as *starting-steps*. The emulation of a step that reduces a conditional in the source is a single starting-step that also reduces a conditional in the target. The emulation of a communication starts with a single starting-step followed by some other steps to complete the emulation (that might be interleaved with steps from other emulations). Similarly, for branching we have again a single starting-step in the beginning.

For soundness we have to show that all steps of encoded terms belong modulo bisimilarity to the emulation of a source term step. In the proof we analyse the sequence of steps $[\![S]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \Longmapsto T$ and identify all source term steps $S \Longmapsto S'$ whose emulation is started within $[\![S]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \Longmapsto T$ and the target term steps $T \Longmapsto \approx_{\mathsf{CMV}} [\![S']\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$ that are necessary to complete all started emulations modulo bisimulation. Therefore, we use an induction on the number of steps in the sequence $[\![S]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \Longmapsto T$ and analyse the encoding function in order to distinguish between different kinds of target term steps and the emulations of source term steps to that they belong. Note that, as it is typical for many encodability results, the proof of operational soundness is more elaborate than the proof of operational completeness presented in [7].

Lemma 5.3 *(Soundness, $[\![\cdot]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$). The encoding $[\![\cdot]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$ is operationally sound modulo $\approx_{\mathsf{CMV}}$, i.e., $[\![S]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \Longmapsto T$ implies $S \Longmapsto S'$ and $T \Longmapsto \approx_{\mathsf{CMV}} [\![S']\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$.*

Proof. We have to prove that for all $S \in \mathcal{P}_{\mathsf{CMV}^+}$ and all $T \in \mathcal{P}_{\mathsf{CMV}}$ such that $[\![S]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \Longmapsto T$ there is some $S' \in \mathcal{P}_{\mathsf{CMV}^+}$ and some $T' \in \mathcal{P}_{\mathsf{CMV}}$ such that $S \Longmapsto S'$, $T \Longmapsto T'$, and $T' \approx_{\mathsf{CMV}} [\![S']\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$. Since $[\![\cdot]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$ is a typed encoding, S is well-typed. We further strengthen this goal by requiring that the sequence $T \Longmapsto T'$ contains no starting steps. This ensures that the steps $T \Longmapsto T'$ can only complete already started emulations instead of starting new emulations. We start with an induction on the number of steps in $[\![S]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \Longmapsto T$. For the base case, i.e., if $T = [\![S]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$ is reached in zero steps, we choose $S' = S$ and $T' = T$ and obtain $S \Longmapsto S'$, $T \Longmapsto T'$, and $T' \approx_{\mathsf{CMV}} [\![S']\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$ as required. For the induction step we have $[\![S]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \Longmapsto T_1 \longmapsto T$. By the induction hypothesis, there are S_2, T_2 such that $S \Longmapsto S_2$, $T_1 \Longmapsto T_2$, and $T_2 \approx_{\mathsf{CMV}} [\![S_2]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$, where $T_1 \Longmapsto T_2$ does not contain starting-steps.

- If the sequence $T_1 \Longmapsto T_2$ reduces in one step the same conditional or the same input and output or selection and branching constructs as reduced in $T_1 \longmapsto T$ then no step in the sequence $T_1 \Longmapsto T_2$ prior to the step that resembles $T_1 \longmapsto T$ can reduce the same capabilities as reduced in $T_1 \longmapsto T$, because they are reduced in $T_1 \longmapsto T$. Then all steps in $T_1 \Longmapsto T_2$ prior to the step that reduces the same capabilities as $T_1 \longmapsto T$ are pairwise distributable to $T_1 \longmapsto T$. By repeatedly applying Lemma 2.19, then we can reorder the sequence such that $T_1 \longmapsto T \Longmapsto T_2$. Then we can choose $S' = S_2$ and $T' = T_2$ such that $S \Longmapsto S'$, $T \Longmapsto T'$, and $T' \approx_{\mathsf{CMV}} [\![S']\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$ as required.
- Else, consider the case that $T_1 \longmapsto T$ is not in conflict with any step in $T_1 \Longmapsto T_2$. Note that if $T_1 \longmapsto T$ is a part of an emulation but not a starting-step the corresponding emulation that was started before or by reaching T_1 was finished in $T_1 \Longmapsto T_2$ modulo $\approx_{\mathsf{CMV}}$ to ensure $T_2 \approx_{\mathsf{CMV}} [\![S_2]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$. Since $T_1 \longmapsto T$ is not in conflict with any step in $T_1 \longmapsto T_2$, then either $T_1 \approx_{\mathsf{CMV}} T$ or the step $T_1 \longmapsto T$ is a starting-step. Note that $T_1 \approx_{\mathsf{CMV}} T$ may result from a communication on the channel endpoints u, v introduced in the Cases 5 or 6 of the encoding function, but also e.g. from a non-deterministic choice with a single option.
  - If $T_1 \approx_{\mathsf{CMV}} T$ then $T \Longmapsto T_2$, i.e., we can choose $S' = S_2$ and $T' = T_2$ such that $S \Longmapsto S'$, $T \Longmapsto T'$, and $T' \approx_{\mathsf{CMV}} [\![S']\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$ as required.
  - Otherwise, if $T_1 \longmapsto T$ is a starting step we complete this emulation with a sequence $T_1 \longmapsto T \Longmapsto T''$ as described in the completeness proof in [7], where $T \Longmapsto T''$ does not contain starting steps. Since $T_1 \longmapsto T$ is a starting-step, no step of the sequence $T_1 \longmapsto T \Longmapsto T''$ is in conflict with any step of $T_1 \Longmapsto T_2$. By repeatedly applying Lemma 2.19, then there is some T' such that $T \Longmapsto T'' \Longmapsto T'$, where the sequence $T'' \Longmapsto T'$ performs the steps of $T_1 \Longmapsto T_2$ starting in T'' instead of T_1. Moreover, there is some S' such that $S \Longmapsto S_2 \longmapsto S'$, where the step $S_2 \longmapsto S'$ is the step that is emulated in $T_1 \longmapsto T \Longmapsto T''$. Since $T_2 \approx_{\mathsf{CMV}} [\![S_2]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$ and by the construction of T' and S', we have $T' \approx_{\mathsf{CMV}} [\![S']\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$.
- Otherwise, there is exactly one step in the sequence $T_1 \Longmapsto T_2$ that is in conflict with the step $T_1 \longmapsto T$. Every such conflict marks a decision in the emulation of one or another source term step. To conclude, we have to show that the sequences $T_1 \Longmapsto T_2$ and $S \Longmapsto S_2$ can be adapted to the alternative decisions in $T_1 \longmapsto T$, i.e., that all decisions of an encoded term lead to the emulation of a source term step. We consider the ten cases of translations in Fig. 6 and 7. The procedure is similar for all decision points, we give a detailed proof for the first case. The other cases are similar or simpler.

Case 1 (linear choice typed as linear and internal): The translation of a choice

$$\left[\!\!\left[\Gamma \vdash \mathsf{lin}\, y \sum_{i\in I} \left(\sum_{j\in J_i} \mathrm{l}_i!v_{i,j}.P_{i,j} + \sum_{k\in K_i} \mathrm{l}_i?x_{i,k}.P'_{i,k} \right) \right]\!\!\right]^{\mathsf{CMV}^+}_{\mathsf{CMV}} =$$

$$\mathsf{NDC}\left\{ y \triangleleft \mathrm{l}_{i,!}.\mathsf{NDC}\left\{ y!v_{i,j}.\, [\![\Gamma_4 \vdash P_{i,j}]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \right\}_{j\in J_i}, \right.$$

$$\left. y \triangleleft \mathrm{l}_{i,?}.\mathsf{NDC}\left\{ \mathsf{lin}\, y?x_{i,k}.\, \left[\!\!\left[(\Gamma_2 + y : U'_i), x_{i,k} : T'_i \vdash P'_{i,k} \right]\!\!\right]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \right\}_{k\in K_i} \right\}_{i\in I}$$

that is typed as linear and internal starts with a non-deterministic choice that has exactly one option for each summand of the source term choice. The first NDC construct non-deterministically picks the translation of one of these summands. A conflict between $T_1 \longmapsto T$ and one step in $T_1 \Longmapsto T_2$ competing for this NDC then means that they both reduce this NDC construct but pick different source term summands. Since S is well-typed, the source term choice can be reduced only with a communication partner that is typed as linear external choice and encoded by the second case and there is at most one such choice on the respective other channel endpoint.

- If there is no such choice on the other channel endpoint then the source term choice is stuck. Since S well-typed, this can happen only for channel names that are free. Because the encoding does use source term channels only to encode a choice that is already on this source term channel, if the source term choice is stuck, so is its translation. In this case, the difference between $T_1 \longmapsto T$ and its conflicting step

in $T_1 \Longrightarrow T_2$ cannot be observed modulo $\approx_{\mathsf{CMV}}$, since both translations of summands emit the same barb. Since all steps of $T_1 \Longrightarrow T_2$ prior to the step that is in conflict to $T_1 \longmapsto T$ cannot reduce this NDC, they are pairwise distributable to $T_1 \longmapsto T$. Then we can simply replace the conflicting step in $T_1 \longmapsto T_2$ by the step $T_1 \longmapsto T$ and (by repeatedly applying Lemma 2.19) reorder the sequence, i.e., we have $T_1 \longmapsto T \Longrightarrow T'$, where T' is obtained from T_2 by exchanging the translations of the two summands. Then we choose $S' = S_2$ and have $S \Longrightarrow S'$, $T \Longrightarrow T'$, and $T' \approx_{\mathsf{CMV}} [\![S']\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$ as required.

– Otherwise, the type system ensures that for each combination of label and polarity requested by the internal choice the external choice offers a matching summand. The translation of choice typed as linear and internal introduces primitives for four consecutive steps: the outer non-deterministic choice, a selection construct, another non-deterministic choice, and an output or input. The translation of choice typed as linear and external introduces a branching, then a non-deterministic choice, and an output or input. Since we reason modulo $\approx_{\mathsf{CMV}}$, the sequence $T_1 \Longrightarrow T_2$ might not contain all of these steps. But, since we forbid for starting-steps in $T_1 \Longrightarrow T_2$, there are no steps that rely on the emulation of this source term communication. Then we remove the conflicting step in $T_1 \Longrightarrow T_2$ as well as all steps that also belong to this emulation attempt. Instead let $T_1 \longmapsto T \Longrightarrow T''$ be the steps necessary to fully emulate a step with the summand picked in $T_1 \longmapsto T$. That such a sequence of steps can be found was shown in the completeness theorem in [7]. Since $T \Longrightarrow T''$ completes the already started emulation of a source term step but does not start any other emulation, $T \Longrightarrow T''$ does not contain starting steps. The remaining steps of $T_1 \Longrightarrow T_2$, i.e., the sequence after removing the conflicting step and the other steps that belong to this emulation attempt, and $T_1 \longmapsto T \Longrightarrow T''$ are distributable. By repeatedly applying Lemma 2.19, then there is some T' such that $T \Longrightarrow T'' \Longrightarrow T'$, where the sequence $T'' \Longrightarrow T'$ executes exactly the steps performed in $T_1 \Longrightarrow T_2$ after removing the steps on the conflicting emulation. Then in $S \Longrightarrow S'$, S' is obtained from S_2 by exchanging the source term step whose emulation we removed by the source term step that is emulated in $T \Longrightarrow T''$. Since $T_2 \approx_{\mathsf{CMV}} [\![S_2]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$ and by the construction of T' and S', then $T' \approx_{\mathsf{CMV}} [\![S']\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$.

The second primitive introduced by Case 1 of the encoding is a selection construct. This step cannot be in conflict with any other step, because the matching branching construct in Case 2 provides exactly one branch for each combination of label and polarity. The third primitive is again a non-deterministic choice that allows to pick one value for transmission and matching continuation if there are several summands with the same label and polarity. The proof for this case is similar to the non-deterministic choice above. Finally, there is an output or input. Again this step cannot be in conflict with a step in $T_1 \Longrightarrow T_2$, because it is not possible to unguard more than one input or output for a choice that is typed as linear.

Case 2 (linear choice typed as linear and external): This case is dual to the case above, but simpler since the first non-deterministic choice is missing.

Case 3 (linear choice typed as unrestricted and internal): The translation of a

$$\left[\!\!\left[\Gamma \vdash \mathsf{lin}\, y \sum_{i\in I} \left(\sum_{j\in J_i} \mathsf{l}_i!v_{i,j}.P_{i,j} + \sum_{k\in K_i} \mathsf{l}_i?x_{i,k}.P'_{i,k} \right) \right]\!\!\right]^{\mathsf{CMV}^+}_{\mathsf{CMV}} =$$

$$\mathsf{NDC}\left\{ (\nu cd)\left(y!c.d \triangleleft \mathsf{l}_{i,!}.\mathsf{NDC}\left\{ d!v_{i,j}.\ [\![\Gamma \vdash P_{i,j}]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \right\}_{j\in J_i} \right), \right.$$

$$\left. (\nu cd)\left(y!c.d \triangleleft \mathsf{l}_{i,?}.\mathsf{NDC}\left\{ \mathsf{lin}\, d?x_{i,k}.\ \left[\!\!\left[\Gamma, x_{i,k} : T'_i \vdash P'_{i,k} \right]\!\!\right]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \right\}_{k\in K_i} \right) \right\}_{i\in I}$$

linear choice that is typed as unrestricted and internal starts again with a non-deterministic choice that has exactly one option for each summand of the source term choice. A conflict between $T_1 \longmapsto T$ and one step in $T_1 \Longrightarrow T_2$ competing for the first NDC then means that they both reduce this NDC construct but pick different source term summands. We proceed as with the non-deterministic choice in Case 1. Here, the translation of a linear choice typed as unrestricted and internal introduces primitives for five consecutive steps: the outer non-deterministic choice, an output, a selection construct, another non-deterministic choice, and an output or input. The translation of choice typed as unrestricted and external introduces an output and matching input in Case 6 (but not Case 4), an input, a branching, then a non-deterministic choice, and an output or input. Accordingly, we might need to remove more steps from $T_1 \Longrightarrow T_2$.

The second primitive introduced by Case 3 is an output. Since in Case 3 a choice typed as unrestricted is translated that is matched by the type system with another choice typed as unrestricted, there can be several outputs on this channel endpoint or several inputs on the other channel endpoint. Since we forbid for starting-steps in $T_1 \Longrightarrow T_2$, there are no steps that rely on the emulation of the source term communication. Again we remove the conflicting step in $T_1 \Longrightarrow T_2$ as well as all steps that also belong to this emulation attempt. Instead let $T_1 \longmapsto T \Longrightarrow T''$ be the steps necessary to fully emulate a step with the input and output picked in $T_1 \longmapsto T$, where $T \Longrightarrow T''$ does not contain starting steps. That such a sequence of steps can be found

was shown in the completeness theorem in [7]. The remaining steps of $T_1 \Longmapsto T_2$, i.e., the sequence after removing the conflicting step and the other steps that belong to this emulation attempt, and $T_1 \longmapsto T \Longmapsto T''$ are distributable. By repeatedly applying Lemma 2.19, then there is some T' such that $T \Longmapsto T'' \Longmapsto T'$, where the sequence $T'' \Longmapsto T'$ executes exactly the steps performed in $T_1 \Longmapsto T_2$ after removing the steps on the conflicting emulation. Then in $S \Longmapsto S'$, S' is obtained from S_2 by exchanging the source term step whose emulation we removed by the source term step that is emulated in $T \Longmapsto T''$. Since $T_2 \approx_{\mathsf{CMV}} [\![S_2]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$ and by the construction of T' and S', then $T' \approx_{\mathsf{CMV}} [\![S']\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$.

Next there is a selection construct matched by a branching construct in the Cases 4 or 6. The restriction on the channel endpoints c and d in Case 3 ensures that this step cannot be in conflict with any other step. The fourth primitive is again a non-deterministic choice that allows to pick one value for transmission and matching continuation if there are several summands with the same label and polarity. The proof for this case is similar to the non-deterministic choice in Case 1. Finally, there is an output or input. Again the restriction on the channel endpoints c and d in Case 3 ensures that this step cannot be in conflict with any other step.

Case 4 (linear choice typed as unrestricted and external): This case is dual to the case above, but simpler since the first non-deterministic choice is missing.

Case 5 (unrestricted choice typed as unrestricted and internal): In comparison to the Case 3 there is only one additional internal step on the restricted channels u, v. Because of the restriction, this step cannot be in conflict with any other step. The proof is then as in Case 3.

Case 6 (unrestricted choice typed as unrestricted and external): Case 6 is dual to the case above, but simpler since the first non-deterministic choice is missing.

Case 7 (parallel composition): The translation of parallel composition does not introduce any step, i.e., there are no conflicts to be considered in this case.

Case 8 (restriction): The translation of restriction does not introduce any step, i.e., there are no conflicts to be considered in this case.

Case 9 (conditional): The translation of a conditional in CMV^+ yields a conditional in CMV. Since steps reducing a conditional in CMV^+ (as well as CMV) cannot be in conflict with any other step, there are no conflicts to be considered in this case.

Case 10 (inaction): The translation of $\mathbf{0}$ cannot perform steps, i.e., there are no conflicts to be considered in this case. □

In Example 5.2 we have $T_4 \approx_{\mathsf{CMV}} [\![S_2']\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$, because all differences between T_4 and $[\![S_2']\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$ are due to junk that cannot be observed modulo $\approx_{\mathsf{CMV}}$. In fact, we have already $T_3 \approx_{\mathsf{CMV}} [\![S_2']\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$, since we consider a weak form of bisimulation here.

In the above variant of soundness T can catch up with the source term S' by the steps $T \Longmapsto\approx_{\mathsf{CMV}} [\![S']\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$. This allows for so-called *intermediate states*: target terms that are strictly in between the translation of two source terms, i.e., T such that $S \longmapsto S'$, $[\![S]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \Longmapsto T \Longmapsto\approx_{\mathsf{CMV}} [\![S']\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$, but neither $[\![S]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \approx_{\mathsf{CMV}} T$ nor $[\![S']\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \approx_{\mathsf{CMV}} T$ (see [32,39]). In $[\![\cdot]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$ such intermediate states are caused by mapping the task of finding matching communication partners of a single source term step onto several steps in the target. Consider the term T_2 in the above emulation of $S \longmapsto S_2'$. By picking the branch with label l_1, we discarded the branch with label l_2. Because of that, the emulation starting with $[\![S]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \Longmapsto T_2$ can no longer emulate source term steps of S that use channel x for receiving, i.e., $T_2 \not\approx_{\mathsf{CMV}} [\![S]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$. But, since we have not yet decided whether we emulate a communication with the first or second choice on y, we also have $T_2 \not\approx_{\mathsf{CMV}} [\![S_2']\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$ whenever $S_2 \not\approx_{\mathsf{CMV}^+} S_4$. Indeed, if we assume that S_1, S_2, S_3, S_4 are pairwise not bisimilar, then $T_2 \not\approx_{\mathsf{CMV}} [\![S']\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$ for all $S \longmapsto S'$, i.e., T_2 is an intermediate state.

The existence of intermediate states prevents us from using stronger versions of soundness, i.e., with $T \asymp [\![S']\!]$ instead of the requirement $T \Longmapsto_{\mathrm{T}} \asymp [\![S']\!]$ in soundness. The encoding $[\![\cdot]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$ needs the steps in $T \Longmapsto\approx_{\mathsf{CMV}} [\![S']\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$ to complete the emulation of source term steps started in $[\![S]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}} \Longmapsto T$. With the soundness result we can complete the proof of [7] that $[\![\cdot]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$ presented in [7, §7] is good.

Theorem 5.4 *(Encoding from* CMV^+ *into* CMV*). The encoding* $[\![\cdot]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$ *from* CMV^+ *into* CMV *presented in [7] is good. By this encoding source terms in* CMV^+ *and their literal translations in* CMV *are related by coupled similarity.*

Proof. Compositionality follows from the encoding function in the Figs. 6 and 7. Under the assumption that c, d, u, v are different from all source term names, we can translate source term names by themselves. This ensures name invariance. If instead of this assumption the renaming policy $\varphi^{\mathsf{CMV}^+}_{\mathsf{CMV}}(\cdot)$ is used, name invariance follows from the consequent use of this renaming policy. Operational completeness was shown in [7] w.r.t. $\equiv^{\mathsf{gc}}$. Since $\equiv^{\mathsf{gc}}$ is contained in $\approx_{\mathsf{CMV}}$, we can inherit this completeness result. Operational soundness follows from Lemma 5.3. By the Figs. 6 and 7, all literal translations of source terms have the same barbs as the respective source term and the encoding does not introduce free names. Barb

sensitiveness then follows from operational correspondence, since $\approx_{\mathsf{CMV}}$ respects barbs. Divergence reflection follows from operational correspondence, since every sequence of target term steps eventually emulates a source term step and since all emulations of a single source term step are finite. Distributability preservation follows from the homomorphic translation of the parallel operator. We conclude that the encoding $[\![\cdot]\!]^{\mathsf{CMV}^+}_{\mathsf{CMV}}$ is good.

As shown in [35] the combination of operational correspondence, divergence reflection, and barb sensitiveness induces a (weak reduction, barbed) coupled similarity that relates all source terms and their literal translations. □

Note that the translation of choice and in particular the non-deterministic choices distribute the decision made by a single source term step into several smaller decisions on the target: first a label and polarity on the internal choice is chosen, then a matching summand in the external choice, ... As explained in [36,33,2], splitting decisions leads to intermediate states and prevents from a tighter connection between source and target, i.e., this encoding relates source terms and their literal translations by coupled similarity and not bisimilarity as shown in [35]. To obtain a tighter connection such as the bisimilarity, we would need the stronger version of soundness with $T \asymp [\![S']\!]$ instead of $T \Longmapsto_{\mathrm{T}} \asymp [\![S']\!]$ (see [35]).

As mentioned, a key feature of the encoding is to translate the nature of its summands, i.e., whether they are send or receive actions, into the label used by the target term. That this is possible, i.e., that the prefixes for send and receive in a choice of CMV^+ can be translated to labels in a separate choice of CMV such that the difference is not observable modulo the criteria in Definition 2.18, gives us the last piece of evidence that we need. CMV^+ does not allow to solve problems such as leader election (Theorem 3.7) that are standard problems for mixed choice; CMV^+ cannot express the synchronisation pattern $\star$ either that we associate with mixed choice (Theorem 4.8). Yet, CMV^+ can express the pattern **M** which is associated with *separate choice*, and is encoded by a language with only separate choice (Theorem 5.4). We conclude that choice in CMV^+ is semantically rather a separate choice.

Corollary 5.5. *Concerning its expressive power the extension of* CMV *given by* CMV^+ *introduces a form of separate choice rather than mixed choice.*

6. Related work and outlook

We conclude by discussing related work, summing up our results, and briefly discussing our next steps.

6.1. Related work

Encodings or the proof of their absence are the main way to compare process calculi [3,30,15,17,16,10,33,12,31,9,13]. See [34] for an overview and discussion on encodings. We used this methodology to compare different variants of choice in session types.

The relevance of mixed choice for the expressive power of the π-calculus was extensively studied. An important encodability result on choices is the existence of a good encoding from the choice-free synchronous π-calculus into its asynchronous variant [4,19], since it proves the relevance of choice. As for the separation result, [29,17,37] have shown that there is no good encoding from the full π-calculus, i.e., the synchronous π-calculus including mixed choice, into its asynchronous variant if an encoding should preserve the distribution of systems. Palamidessi in [28] was the first to point out that mixed choice strictly raises the expressive power of the π-calculus. Later work studies the criteria under that this separation result holds and alternative ways to prove this result: [27] studies the relevance of divergence reflection for this result and considers separate choice. [17,30] discuss how to reprove this result if the rather strict criterion on the homomorphic translation of the parallel operator is replaced by compositionality. [33,36] show that compositionality itself is not strong enough to replace the homomorphic translation of the parallel operator by presenting an encoding and then propose the preservation of distributability as criterion to regain the result of Palamidessi. [37] uses the more fundamental problem of breaking symmetries instead of leader election. [39] further simplifies this separation result by introducing synchronisation patterns to distinguish the languages. [40] shows that instead of the preservation of distributability or the homomorphic translation of the parallel operator also the preservation of causality can be used as criterion.

While there are a vast amount of theories [22], programming languages [1], and tools [43] of session types, as far as we know, the CMV^+-calculus is the only session π-calculus which extends external and internal choices to their mixtures with full constructs, i.e. delegation, shared (or unlimited) name passing, value passing, and recursion in its process syntax, proposes its typing system and proves type-safety. In the context of *multiparty session types* [21], there are several works that extend the original form of global types where choice is fixed (from one sender to one receiver) with more flexible forms of choices: Recent work in [24] e.g. allows the global type to specify a choice of one sender to transmit to one of several receivers. In [23] flexible choices are discussed but their well-formedness (which ensures deadlock-freedom of local types) needs to be checked by bisimulation. These works focus on gaining expressiveness of behaviours of a set of local types (or a simple form of CCS-like processes which are equivalent to local types [24]) which correspond to *a single multiparty session*, without delegations, interleaved sessions, restrictions nor name passing.

More recently, [14] compares the expressive power of a variant of the π-calculus (with implicit matching) and the variant of CCS where the result of a synchronisation of two actions is itself an action subject to relabelling or restriction.

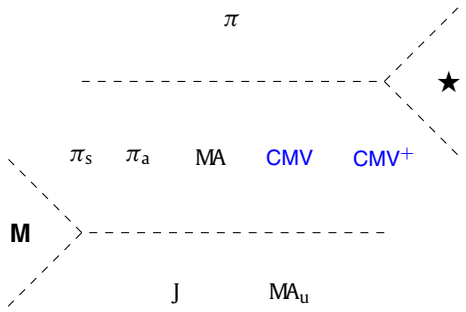


Fig. 8. Hierarchy of Pi-like Calculi.

Because of the connection between CCS-like languages and local types, it may be interesting to compare the expressiveness results in [14] with (variants of) multiparty session types.

6.2. Summary and outlook

We proved that CMV$^+$ is strictly less expressive than the π-calculus in two different ways: by showing that CMV$^+$ cannot solve leader election in symmetric networks of odd degree and that CMV$^+$ cannot express the synchronisation pattern $\star$. Then we provide the missing soundness proof for the encoding presented in [7]. From these results and the insights on the reasons of these results, we conclude that the choice primitive added to CMV in [7] is rather a separate choice and not a mixed choice at least with respect to its expressive power.

With these results we can extend the hierarchy of pi-like calculi obtained in [39,38] by two more languages as depicted in Fig. 8. This hierarchy orders languages according to their ability to express certain synchronisation patterns. At the top we have the π-calculus (π), because it can express the synchronisation pattern $\star$. In the middle are languages that can express **M** but not $\star$: the π-calculus with separate choice (π_s) [27], the asynchronous π-calculus without choice (π_a) [19,4], Mobile Ambients (MA) [6], CMV, and CMV$^+$. In the bottom we have the join-calculus (J) [8] and Mobile Ambients with unique Ambient names (MA$_u$) [38], i.e., the languages that cannot express $\star$ or **M**. That π, π_s, π_a, MA, J, and MA$_u$ can or cannot express the respective pattern was shown in [39,38].

Linearity as enforced by the type system of CMV/CMV$^+$ restricts the possible structures of communication protocols. In particular, the type system ensures that it is impossible to unguard two competing inputs or outputs on the same linear channel at the same time. Accordingly, it is not surprising that adding choice, even mixed choice, towards communication primitives under a type discipline that enforces linearity does not significantly increase the expressive power of the respective language (though it still might increase flexibility). However, that adding mixed choice between unrestricted communication primitives does not significantly increase the expressive power of the language, did surprise us. Unrestricted channels allow to have several in- or outputs on these channels in parallel, because the type system only ensures the absence of certain communication mismatches as e.g. that the sort of a transmitted value is as expected by the receiver; but not linearity (compare also to shared channels as e.g. in [20]). So, there is no obvious reason why the type system should limit the expressive power of unrestricted channels within a mixed choice. Indeed, it turns out that the problem lies not in the type system. In both ways to prove the separation result in §3 and §4 we completely ignore the type system and carry out the proof on the untyped version of the language, i.e., it is already the untyped version of CMV$^+$ that cannot express mixed choice despite a mixed-choice-like primitive. This limitation of the language definition, i.e., in its syntax and semantics, is not obvious and indeed it was very hard to spot the problem.

The two separation results in Section 3 and 4 reveal the reasons for this limitation. The expressive power of choice in CMV$^+$ is limited by the fact that syntactically the choice construct is fixed on a single channel endpoint and the requirement of the semantics that a channel endpoint can interact with exactly one other channel endpoint. These insights will help us to extend other variants of session types with stronger versions of mixed choice.

We expect that adding mixed choice to the non-linear parts of other session type systems will instead significantly increase the expressive power. Accordingly, as the next step, we want to add a primitive for mixed choice between shared channels in session types such as described e.g. in [20,45] and analyse the expressiveness of the resulting language.

CRediT authorship contribution statement

Kirstin Peters: Conceptualization, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing. **Nobuko Yoshida:** Conceptualization, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The work is partially supported by EPSRC (EP/T006544/2, EP/Y005244/1, EP/K034413/1, EP/L00058X/1, EP/N027833/2, EP/T014709/2, EP/V000462/1 and EP/X015955/1) and Horizon EU TaRDIS 101093006. We thank the anonymous reviewers for their comments and suggestions.

Appendix A. Proof of Lemma 4.7

Proof. By operational completeness, all five steps of $\mathsf{S}^\star_\pi$ have to be emulated in $[\![\mathsf{S}^\star_\pi]\!]$, i.e., there exist some $T_a, T_b, T_c, T_d, T_e \in \mathcal{P}_{\mathsf{CMV}^+}$ such that $X : [\![\mathsf{S}^\star_\pi]\!] \Longmapsto T_x \asymp [\![S_x]\!]$ for all $x \in \{a, b, c, d, e\}$, where X is the upper case variant of x. By Lemma 4.6, for all $T_a, T_b, T_c, T_d, T_e \in \mathcal{P}_{\mathsf{CMV}^+}$ and all $x \in \{a, b, c, d, e\}$ such that $T_x \asymp [\![S_x]\!]$, there is a conflict between a step of the following pairs of emulations: A and B, B and C, C and D, D and E, and E and A.

Since $[\![\cdot]\!]$ preserves distributability and by Lemma 2.22, each pair of distributable steps in $\mathsf{S}^\star_\pi$ has to be translated into emulations that are distributable within $[\![\mathsf{S}^\star_\pi]\!]$. Let $X, Y, Z \in \{A, B, C, D, E\}$ be such that X and Z are distributable within $[\![\mathsf{S}^\star_\pi]\!]$ but Y is in conflict with X as well as Z. By Lemma 2.21, this implies that $[\![\mathsf{S}^\star_\pi]\!]$ is distributable into $T_1, T_2 \in \mathcal{P}_{\mathsf{CMV}^+}$ such that X is an execution of T_1 and Z is an execution of T_2. Since Y is in conflict with X and Z and because all three emulations are executions of $[\![\mathsf{S}^\star_\pi]\!]$, there is one step of Y that is in conflict with one step of X and there is one (possibly the same) step of Y that is in conflict with one step of Z. Moreover, since X and Z are distributable, if a single step of Y is in conflict with X as well as Z then this step is a communication between T_1 and T_2.

Assume that for all such combinations X, Y, and Z, the conflicts between Y and X or Z are ruled out by a single step of Y, i.e., both conflicts are ruled out by a communication step between some choice of X and some choice of Z. Then this step reduces one endpoint in one of the executions X and Z and the respective other endpoint in the respective other execution, i.e., X and Y compete for one endpoint and Y and Z compete for the respective other endpoint (compare to Lemma 4.5). Without loss of generality let us assume that A and B compete for the channel endpoint x and, thus, B and C compete for the channel endpoint y, C and D compete for x, D and E compete for y, E and A compete for x, and A and B compete for y. This is a contradiction, because A and B cannot compete for both channel endpoints x and y.

We conclude that there is at least one triple of emulations X, Y, and Z such that the conflict of Y with X and with Z results from two different steps in Y. Because X and Z are distributable, the reduction steps of X that lead to the conflicting step with Y and the reduction steps of Z that lead to the conflicting step with Y are distributable. We conclude that there is at least one emulation of y, i.e., one execution $Y : [\![\mathsf{S}^\star_\pi]\!] \Longmapsto T_y \asymp [\![S_y]\!]$, starting with two distributable executions such that one is (in its last step) in conflict with the emulation of x in $X : [\![\mathsf{S}^\star_\pi]\!] \Longmapsto T_x \asymp [\![S_x]\!]$ and the other one is in conflict with the emulation of z in $Z : [\![\mathsf{S}^\star_\pi]\!] \Longmapsto T_z \asymp [\![S_z]\!]$. In particular this means that also the two steps of Y that are in conflict with a step in X and a step in Z are distributable. Hence, it is impossible to ensure that these two conflicts are decided consistently, i.e., there is a maximal execution of $[\![\mathsf{S}^\star_\pi]\!]$ that emulates X but neither Y nor Z.

In the set $\{A, B, C, D, E\}$ there are—apart from X, Y, and Z—two remaining executions. One of them, say X', is in conflict with X and the other one, say Z', is in conflict with Z. Since X is emulated successfully, X' cannot be emulated. Moreover, note that Y and Z' are distributable. Thus, also Z' and the partial execution of Y that leads to the conflict with Z are distributable. Moreover, also the step of Y that already rules out Z cannot be in conflict with a step of Z'. Thus, although the successful completion of Z is already ruled out by the conflict with Y, there is some step of Z left, that is in conflict with one step in Z'. Hence, the conflict between Z and Z' cannot be ruled out by the partial execution described so far that leads to the emulation of X but forbids to complete the emulations of X', Y, and Z. Thus, it cannot be avoided that Z wins this conflict, i.e., that also Z' cannot be completed. We conclude that there is a maximal execution of $[\![\mathsf{S}^\star_\pi]\!]$ such that only one of the five source term steps of $\mathsf{S}^\star_\pi$ is emulated. □

References

[1] D. Ancona, V. Bono, M. Bravetti, J. Campos, G. Castagna, P. Deniélou, S.J. Gay, N. Gesbert, E. Giachino, R. Hu, E.B. Johnsen, F. Martins, V. Mascardi, F. Montesi, R. Neykova, N. Ng, L. Padovani, V.T. Vasconcelos, N. Yoshida, Behavioral Types in Programming Languages, Foundations and Trends in Programming Languages, vol. 3, 2016, pp. 95–230.
[2] B. Bisping, U. Nestmann, K. Peters, Coupled similarity: the first 32 years, Acta Inform. 57 (2019) 439–463, https://doi.org/10.1007/s00236-019-00356-4.
[3] F.S. Boer, C. Palamidessi, Embedding as a tool for language comparison: on the CSP hierarchy, in: Proc. of CONCUR, Springer, 1991, pp. 127–141.
[4] G. Boudol, Asynchrony and the π-calculus (Note), Rapport de Recherche 1702, 1992, https://hal.inria.fr/inria-00076939/document.
[5] D. Cacciagrano, F. Corradini, C. Palamidessi, Explicit fairness in testing semantics, Log. Methods Comput. Sci. 5 (2009) 1–27, https://doi.org/10.2168/LMCS-5(2:15)2009.
[6] L. Cardelli, A.D. Gordon, Mobile ambients, Theor. Comput. Sci. 240 (2000) 177–213, https://doi.org/10.1016/S0304-3975(99)00231-5.
[7] F. Casal, A. Mordido, V.T. Vasconcelos, Mixed sessions, Theor. Comput. Sci. 897 (2022) 23–48, https://doi.org/10.1016/j.tcs.2021.08.005.
[8] C. Fournet, G. Gonthier, The reflexive chemical abstract machine and the join-calculus, in: J.G. Steele (Ed.), Proc. of POPL, ACM, 1996, pp. 372–385.
[9] Y. Fu, Theory of interaction, Theor. Comput. Sci. 611 (2016) 1–49, https://doi.org/10.1016/j.tcs.2015.07.043.

[10] Y. Fu, H. Lu, On the expressiveness of interaction, Theor. Comput. Sci. 411 (2010) 1387–1451, https://doi.org/10.1016/j.tcs.2009.11.011.
[11] R. van Glabbeek, U. Goltz, J.W. Schicke, On synchronous and asynchronous interaction in distributed systems, in: Proc. of MFCS, 2008, pp. 16–35.
[12] R.J. van Glabbeek, Musings on encodings and expressiveness, in: Proc. of EXPRESS/SOS, 2012, pp. 81–98.
[13] R.J. van Glabbeek, A theory of encodings and expressiveness (extended abstract), in: Proc. of FoSSaCS, 2018, pp. 183–202.
[14] R.J. van Glabbeek, Comparing the expressiveness of the π-calculus and CCS, in: I. Sergey (Ed.), Proc. of ETAPS, Springer, 2022, pp. 548–574.
[15] D. Gorla, Comparing communication primitives via their relative expressive power, Inf. Comput. 206 (2008) 931–952, https://doi.org/10.1016/J.IC.2008.05.001.
[16] D. Gorla, A taxonomy of process calculi for distribution and mobility, Distrib. Comput. 23 (2010) 273–299, https://doi.org/10.1007/S00446-010-0120-6.
[17] D. Gorla, Towards a unified approach to encodability and separation results for process calculi, Inf. Comput. 208 (2010) 1031–1053, https://doi.org/10.1016/j.ic.2010.05.002.
[18] K. Honda, Types for dyadic interaction, in: E. Best (Ed.), Proc. of CONCUR, Springer, 1993, pp. 509–523.
[19] K. Honda, M. Tokoro, An object calculus for asynchronous communication, in: M. Tokoro, O. Nierstrasz, P. Wegner (Eds.), Proc. of ECOOP, Springer, 1992, pp. 133–147.
[20] K. Honda, V.T. Vasconcelos, M. Kubo, Language primitives and type discipline for structured communication-based programming, in: Proc. of ESOP, Springer, 1998, pp. 122–138.
[21] K. Honda, N. Yoshida, M. Carbone, Multiparty asynchronous session types, J. ACM 63 (2016) 1–67, https://doi.org/10.1145/1328438.1328472.
[22] H. Hüttel, I. Lanese, V.T. Vasconcelos, L. Caires, M. Carbone, P. Deniélou, D. Mostrous, L. Padovani, A. Ravara, E. Tuosto, H.T. Vieira, G. Zavattaro, Foundations of session types and behavioural contracts, ACM Comput. Surv. 49 (2016) 3:1–3:36, https://doi.org/10.1145/2873052.
[23] S.S. Jongmans, N. Yoshida, Exploring type-level bisimilarity towards more expressive multiparty session types, in: Proc. of ESOP, Springer, 2020, pp. 251–279.
[24] R. Majumdar, M. Mukund, F. Stutz, D. Zufferey, Generalising projection in asynchronous multiparty session types, in: S. Haddad, D. Varacca (Eds.), Proc. of CONCUR, 2021, pp. 35:1–35:24.
[25] R. Milner, J. Parrow, D. Walker, A calculus of mobile processes, Part I and II, Inf. Comput. 100 (1992) 1–77, https://doi.org/10.1016/0890-5401(92)90008-4.
[26] R. Milner, D. Sangiorgi, Barbed bisimulation, in: Proc. of ICALP, 1992, pp. 685–695.
[27] U. Nestmann, What is a "good" encoding of guarded choice?, Inf. Comput. 156 (2000) 287–319, https://doi.org/10.1006/inco.1999.2822.
[28] C. Palamidessi, Comparing the expressive power of the synchronous and the asynchronous π-calculus, in: Proc. of POPL, 1997, pp. 256–265.
[29] C. Palamidessi, Comparing the expressive power of the synchronous and the asynchronous π-calculus, Math. Struct. Comput. Sci. 13 (2003) 685–719, https://doi.org/10.1017/S0960129503004043.
[30] J. Parrow, Expressiveness of process algebras, Electron. Notes Theor. Comput. Sci. 209 (2008) 173–186, https://doi.org/10.1016/j.entcs.2008.04.011.
[31] J. Parrow, General conditions for full abstraction, Math. Struct. Comput. Sci. 26 (2014) 655–657, https://doi.org/10.1017/S0960129514000280.
[32] J. Parrow, P. Sjödin, Multiway synchronization verified with coupled simulation, in: W. Cleaveland (Ed.), Proc. of CONCUR, Springer Berlin Heidelberg, 1992, pp. 518–533.
[33] K. Peters, Translational Expressiveness, Ph.D. thesis, TU Berlin, 2012, http://opus.kobv.de/tuberlin/volltexte/2012/3749/.
[34] K. Peters, Comparing process calculi using encodings, in: Proc. of EXPRESS/SOS, 2019, pp. 19–38.
[35] K. Peters, R. van Glabbeek, Analysing and comparing encodability criteria, in: S. Crafa, D. Gebler (Eds.), Proc. of EXPRESS/SOS, 2015, pp. 46–60.
[36] K. Peters, U. Nestmann, Is it a "good" encoding of mixed choice?, in: Proc. of FoSSaCS, 2012, pp. 210–224.
[37] K. Peters, U. Nestmann, Breaking Symmetries, Mathematical Structures in Computer Science, vol. 26, 2016, pp. 1054–1106.
[38] K. Peters, U. Nestmann, Distributability of mobile ambients, Inf. Comput. 275 (2020) 104608, https://doi.org/10.1016/j.ic.2020.104608.
[39] K. Peters, U. Nestmann, U. Goltz, On distributability in process calculi, in: Proc. of ESOP, 2013, pp. 310–329.
[40] K. Peters, J.W. Schicke-Uffmann, U. Goltz, U. Nestmann, Synchrony versus causality in distributed systems, Math. Struct. Comput. Sci. 26 (2016) 1459–1498, https://doi.org/10.1017/S0960129514000644.
[41] K. Peters, N. Yoshida, On the expressiveness of mixed choice sessions, in: Proc. of EXPRESS/SOS, 2022, pp. 113–130.
[42] G.D. Plotkin, The origins of structural operational semantics, J. Log. Algebraic Program. 60 (2004) 17–140, https://doi.org/10.1016/j.jlap.2004.03.009, An earlier version of this paper was published as technical report at Aarhus University in 1981.
[43] A.R. Simon Gay (Ed.), Behavioural Types: from Theory to Tools, River Publisher, 2017, https://www.riverpublishers.com/research_details.php?book_id=439.
[44] K. Takeuchi, K. Honda, M. Kubo, An interaction-based language and its typing system, in: Proc. of PARLE, 1994, pp. 398–413.
[45] N. Yoshida, V.T. Vasconcelos, Language primitives and type discipline for structured communication-based programming revisited: two systems for higher-order session communication, in: Proc. of SecReT, 2006, pp. 73–93.