Multiparty Classical Choreographies

Marco Carbone[1], Luís Cruz-Filipe[2], Fabrizio Montesi[2], and Agata Murawska[1]

[1]IT University of Copenhagen [2]University of Southern Denmark

Abstract. We present Multiparty Classical Choreographies (MCC), a language model where global descriptions of communicating systems (choreographies) and processes can be modularly composed to implement typed multiparty sessions. Typing is achieved by generalising classical linear logic to judgements that explicitly record parallelism by means of hypersequents. Our approach unifies different lines of work on choreographies and processes with multiparty sessions, as well as their connection to linear logic. Thus, results developed in one context are carried over to the others. Key novelties of MCC include support for behavioural polymorphism in choreographies, as well as a translation from processes with replication to choreographies.

1 Introduction

Choreographic Programming [14] is a programming paradigm where programs (called choreographies) define the intended communication behaviour of a system based on message passing, using an "Alice and Bob" notation. Then, correct-by-construction endpoint implementations are synthesised automatically [4, 8].

Choreographic programming has a deep relationship with the proof theory of linear logic [5]. Specifically, choreographic programs can be seen as terms describing the reduction steps of cut elimination in linear logic (choreographies as cut reductions). The key advantage of this result is that it provides a logical reconstruction of two useful translations, one from choreographies to processes (projection, or synthesis) and another from processes to choreographies (extraction)—this is obtained by composition with the recently found correspondence between intuitionistic linear logic and a variant of the π-calculus [2]. These translations can be used to keep process implementations aligned with the desired communication flows given as choreographies, whenever code changes are applied to any of the two. This kind of alignment is a desirable property in practice, e.g., it is the basis of the Testable Architecture development lifecycle for web services [11].

Unfortunately, the logical reconstruction of choreographies in [5] covers only the multiplicative-additive fragment of intuitionistic linear logic, limiting its practical applicability to simple scenarios. The aim of this paper is to push the boundaries of this approach—and, crucially, its accompanying translations—towards more realistic scenarios with sophisticated features. Specifically, we cover: (i) replicated services, (ii) behavioural polymorphism, and (iii) multiparty

sessions. Reaching our aim is challenging for both design and technical reasons; we give an overview in the following.

In the multiplicative-additive fragment of linear logic considered in [5], all reductions intuitively match choreographic terms explored in previous works on choreographies, i.e., communication of a channel and branch selection [4]. This is not the case for the exponential fragment, which yields reductions never considered before in choreographies, like explicit garbage collection of services. To bridge this gap, we propose a choreographic syntax for the management of services as explicit resources that can be duplicated, used, or destroyed. We show that the reductions for these terms correspond to the principal cut reductions for exponentials. Typing guarantees that resource management is safe, e.g., no destroyed resource is ever used again. Behavioural polymorphism is addressed with the same idea: we find out that it corresponds to new choreographic terms for the transmission of types and explicit renaming of channels.

In [5], all sessions (protocols) have exactly two participants. This works well in intuitionistic linear logic, where sequents are two-sided: two processes can be connected if one "provides" a behaviour and the other "needs" it. This is verified by checking identity of types, respectively between a type on the right-hand side of the sequent of the first process and a type of the left-hand side of the sequent for the second. To date, it is still unclear how identity for two-sided sequents can be generalised to multiparty sessions, where a session can have multiple participants and thus we need to check compatibility of multiple types. Instead, this topic has been investigated in the setting of classical linear logic, where multiparty compatibility is captured by coherence, a generalisation of duality [6]. Therefore, our formulation of Multiparty Compositional Choreographies (MCC) is based on classical linear logic. In order to bridge choreographies to multiparty sessions, we introduce a new session environment, which records the types of multiparty communications performed by a choreography as *global types* [10]. The manipulation of the session environment reveals that typing a choreography with multiparty sessions corresponds to *building the coherence proofs for typing its sessions*. Since a proof of coherence is the type compatibility check required by the multiparty version of cut in classical linear logic, our result generalises the choreographies as cut reductions approach to the multiparty case as one would expect, providing further evidence of the robustness of this idea.

The final result of our efforts is an expressive calculus for programming choreographies with multiparty sessions, services, and polymorphism, which supports both projection and extraction *for all typable programs*.

2 Preview

We start by giving an informal introduction to MCC. We focus on modelling a variant of the OpenID protocol, a protocol where a client authenticates through a third-party identity provider. MCC offers a way of specifying protocols in terms of global types. For example, the OpenID protocol can be specified by the global

type G, defined as

$$u \to rp(\mathsf{String});\ u \to ip(\mathsf{String});\ u \to ip(\mathsf{PWD});$$
$$ip \to rp.\mathsf{case}(\quad u \to rp(\mathsf{String});\ G_1,\ G_2)$$

This protocol concerns three endpoints (often called roles in literature) denoted by u (user), rp (relaying party) and ip (identity provider). The user starts by sending its login string to both rp and ip. Then, it sends its password to ip which will either confirm or reject u's authentication to rp. If the authentication is successful then the user will send an evaluation of the authentication service to rp, and then complete as the unspecified protocol G_1. Otherwise, if the password is wrong, then the protocol continues as G_2. The specification given by the global type G can be used by a programmer when implementing a distributed system that uses it. In MCC, we could give an implementation on terms of the choreography:

$$\begin{array}{ll}
u\,\mathsf{starts}\,rp, ip; & \texttt{// } u \texttt{ starts protocol with rp and ip} \\
u(user_u) \to rp(user_{rp}); & \texttt{// } u \texttt{ sends its login to } rp \\
u(login_u) \to ip(login_{ip}); & \texttt{// } u \texttt{ sends its login to } ip \\
u(pwd_u) \ \to \ ip(pwd_{ip}); & \texttt{// } u \texttt{ authenticates with } ip \\
ip \ \to rp.\mathsf{inl} \left\{ \begin{array}{ll}
u'\,\mathsf{starts}\,s; & \texttt{// } u' \texttt{ starts protocol with } s \\
u'(rep_{u'}) \to s(rep_s); & \texttt{// } u' \texttt{ sends report to } s \\
s(ack_s) \to u'(ack_{u'}); & \texttt{// } s \texttt{ acknowledges to } u' \\
u(rep_u) \to rp(rep_{rp}); P, Q & \texttt{// } u \texttt{ sends report to } rp
\end{array} \right\}
\end{array}$$

Each line is commented with an explanation of the performed action. We observe that two different protocols are started. That on the first line starts the OpenID protocol between u, rp and ip described above. Moreover, after branching, the choreography starts another session between the user (named u') with a server s that is used for reviewing the authentication service given by ip. In this case, the protocol used is $G' = u' \to s(\mathsf{String}); s \to u'(\mathsf{String})$. We leave unspecified the case in which the identity provider receives a wrong password (term Q).

In this work, we show how a choreography like the one above, that follows a protocol such as G, can be expressed as a proof in a proof theory strictly related classical linear logic. Moreover, thanks to proof transformations, the choreography above can be projected into a parallel composition of endpoint processes, each running a different endpoint. As an example, the endpoint process for the user would correspond to the process P_u, defined as

$$\mathsf{use}\,u^{rp,ip}; \overline{u^{rp}}(user_u); \overline{u^{ip}}(login_u); \overline{u^{ip}}(pwd_u); \mathsf{use}\,u'^{s}; \overline{u'^{s}}(rep_{u'}); u'^{s}(ack_{u'}); \overline{u^{rp}}(rep_u); R$$

which mimics the behaviour of u and u' specified in the choreography. Operator use is used to start a session, while the other two operators utilised above are for in-session communication. Similarly, we can have the endpoint processes for rp, ip and s:

$$P_{rp} = \mathsf{srv}\,rp^u; rp^u(user_{rp}); rp^{ip}.\mathsf{case}(rp^u(rep_{rp}); R_1, Q_1)$$
$$P_{ip} = \mathsf{srv}\,ip^u; ip^u(login_{ip}); ip^u(pwd_{ip}); R_2 \qquad P_s = \mathsf{srv}\,s^{u'}; s^{u'}(rep_s); \overline{s}^{u'}(ack_s); R_3$$

$$
\begin{array}{lllll}
P ::= & x^A \to y^B & & & \text{link} \\
\mid & P \mid Q & & & \text{parallel comp} \\
\mid & (\boldsymbol{\nu} \tilde{x}^{\tilde{A}} : G)\, P & & & \text{restriction} \\
\mid & \overline{x^{\tilde{y}}}(x', y', \tilde{z}'); (P \mid Q) & \mid\ y^{\tilde{x}}(\tilde{x}'; y'); P & \mid\ \tilde{x}(\tilde{x}') \to y(y'); P & \text{communication} \\
\mid & \mathsf{close}[x^y] & \mid\ \mathsf{wait}[y^{\tilde{x}}]; P & \mid\ \tilde{x}\ \mathsf{closes}\ y; P & \text{session close} \\
\mid & x^{\tilde{y}}.\mathsf{inl}; P & \mid\ y^x.\mathsf{case}(P, Q) & \mid\ x \to \tilde{y}.\mathsf{inl}(P, Q) & \text{left selection} \\
\mid & x^{\tilde{y}}.\mathsf{inr}; Q & & \mid\ x \to \tilde{y}.\mathsf{inr}(P, Q) & \text{right selection} \\
\mid & \mathsf{use}\, x^{\tilde{y}}; P & \mid\ \mathsf{srv}\, y^x; P & \mid\ x\ \mathsf{starts}\ \tilde{y}; P & \text{server accept/request} \\
\mid & \mathsf{kill}\, x^{\tilde{y}}; P & & \mid\ x\ \mathsf{kills}\ \widetilde{y_i(Q_i)}; P & \text{server kill} \\
\mid & \mathsf{clone}\, x^{\tilde{y}}(x', \tilde{y}'); P & & \mid\ x\ \mathsf{clones}\ \tilde{y}(x', \tilde{y}'); P & \text{server clone} \\
\mid & x^{\tilde{y}}[A]; P & \mid\ y^x[X]; P & \mid\ x[A] \to \tilde{y}(X); P & \text{type communication} \\
\mid & x^y.\mathsf{case}() & & & \text{empty choice}
\end{array}
$$

$$
\begin{array}{llllll}
A ::= & A \otimes^z B & \text{output} & \mid\ A \parr^{\tilde{z}} B & \text{input} & (\otimes\parr) \\
\mid & A \oplus^{\tilde{z}} B & \text{selection} & \mid\ A \,\&^z\, B & \text{choice} & (\oplus\&) \\
\mid & !^z A & \text{server} & \mid\ ?^{\tilde{z}} A & \text{client} & (!?) \\
\mid & \mathbf{1}^z & \text{close} & \mid\ \bot^{\tilde{z}} & \text{wait} & (\mathbf{1}\bot) \\
\mid & 0^{\tilde{z}} & \text{false} & \mid\ \top^z & \text{empty} & (0\top) \\
\mid & \forall^z X.A & \text{send type} & \mid\ \exists^{\tilde{z}} X.A & \text{receive type} & (\exists\forall) \\
\mid & X & \text{variable} & \mid\ X^\perp & \text{dual variable} & (\text{AXIOM})
\end{array}
$$

$$
\begin{array}{lll}
G ::= & \tilde{x} \to y(G); H & (\otimes\parr) \\
\mid & x \to \tilde{y}.\mathsf{case}(G, H) & (\oplus\&) \\
\mid & !x \to \tilde{y}(G) & (!?) \\
\mid & \tilde{x} \to y & (\mathbf{1}\bot) \\
\mid & x \to \tilde{y}.\mathsf{case}() & (0\top) \\
\mid & x \to \tilde{y}.(X)G & (\exists\forall) \\
\mid & x^A \to y^B & (\text{AXIOM})
\end{array}
$$

Fig. 1. Syntax of MCC

3 Multiparty Classical Choreographies

We present Multiparty Compositional Choreographies (MCC), a language model allowing modular composition of choreographies and processes. We begin by presenting the syntax of programs, program types and global types of sessions. We continue by giving the proof theory of MCC, which includes rules for typing of processes and choreographies, as well as a judgment for verifying that types used in a communication are compatible (coherent).

3.1 Syntax

Programs in MCC can be composed of both processes and choreographies, resulting in a fully modular design. When writing a program in our language, we do not identify sessions via channel names, but rather we name sessions' *endpoints*. A process will then have ownership of only one endpoint of a session, but it will be aware of names of its other endpoints. By contrast, a choreography, which describes a global view of a communication within a session, will own all of such session's endpoints. The complete syntax of MCC is presented in Fig. 1.

The first constructs are structural: a process can be a link between two (compatible) endpoints, a parallel composition of two processes, or a private session among (typed) endpoints $\tilde{x}$ satisfying the global type G. All remaining constructs come in two flavours: *actions*, executed by a single endpoint of a process (the first two columns), or choreography *interactions* (third column), which specify several actions being executed simultaneously.

The actions $\overline{x^y}(x'; y', \tilde{z}'); (P \mid Q)$, $y^{\tilde{x}}(\tilde{x}'; y'); P$ and $\tilde{x}(\tilde{x}') \to y(y'); P$ regard session creation, with the process owning endpoint y acting as a coordinator. In $y^{\tilde{x}}(\tilde{x}'; y'); P$, the process owning y waits for the processes owning each endpoint x to join the session and send the endpoint x' they plan to use. It then shares the names of all endpoints $\tilde{x}'$, as well as its own y', to all participants, and continues as P (which knows $\tilde{x}$ and $\tilde{x}'$, and owns y and y'). Dually, in $\overline{x^y}(x'; y', \tilde{z}'); (P \mid Q)$, the process owning x notifies y of its intention to join the new session and sends its own endpoint x' for it. When the new session starts, the process knows the names $\tilde{z}'$ of all endpoints; in the continuation, P is the protocol for endpoint x' (which may also refer to $\tilde{z}'$ and y') and Q is the protocol for endpoint x (which may also refer to y). Choreographically, we have $\tilde{x}(\tilde{x}') \to y(y'); P$.

The next constructs model termination of a session. In $\mathsf{wait}[y^{\tilde{x}}]; P$, the owner of y waits for all other endpoints to execute their $\mathsf{close}[x^y]$ action (and terminate), and then proceeds as P. The corresponding choreography is $\tilde{x}\ \mathsf{closes}\ y; P$.

In the case of choice, $y^x.\mathsf{case}(P, Q)$, endpoint y offers a choice of two possible behaviours to x; these are selected by a dual selection actions: $x^{\tilde{y}}.\mathsf{inl}; P$ or $x^{\tilde{y}}.\mathsf{inr}; Q$. The selection is the same for all $\tilde{x}$ offering a choice. Again, there are corresponding choreographic actions $x \to \tilde{y}.\mathsf{inl}(P, Q)$ and $x \to \tilde{y}.\mathsf{inr}(P, Q)$.

In $\mathsf{srv}\, y^x; P$, the process owning y offers a service to x at endpoint y with protocol P. This server can react to three kinds of requests. In the action $\mathsf{use}\, x^{\tilde{y}}; P$, endpoint x engages all servers $\tilde{y}$ in a session with protocol P. Alternatively, x may kill all the servers by executing the action $\mathsf{kill}\, x^{\tilde{y}}; P$, or duplicate them by means of $\mathsf{clone}\, x^{\tilde{y}}(x', \tilde{y}'); P$. In the last case, the new copies of the servers are replicated at endpoints $\tilde{y}'$, and are ready to engage in a session with endpoint x'. All these behaviours can be invoked by the choreographic actions $x\, \mathsf{starts}\, \tilde{y}; P$, $x\, \mathsf{kills}\, \widetilde{y_i(Q_i)}; P$ and $x\, \mathsf{clones}\, \tilde{y}(x', \tilde{y}'); P$, respectively. Observe that the choreographic kill action includes the (local) behaviours of all the servers.

In the type send action $x^{\tilde{y}}[A]; P$, x sends type A to all processes in $\tilde{y}$ – matched by the dual actions $y^x[X]; P$ for each owner of endpoint y. This can again be expressed as a choreography $x[A] \to \tilde{y}(X); P$, where X is bound in all actions in P that have y as a subject.

Finally, the empty choice $x^y.\mathsf{case}()$ implements termination.

Types, also defined in Fig. 1, are used to ensure proper behaviour of participants in a session. In the multiparty setting, these can be split into *local types* A, which specify behaviours of a single process, and *global types* G, which describe interaction within sessions and choreography actions. Again most global types correspond to pairs of local types, the exception being the global axiom type, describing a linking session. Local type operators are based on connectives

from classical linear logic; however, each operator is annotated with the names of the endpoints that the process being typed interacts with. Thus, $A \otimes^{z} B$ is the type of a process that outputs to z an endpoint of type A and continues with type B; $A \parr^{\tilde{z}} B$ is the type of a process that receives endpoints of type A from all $\tilde{z}$ and is itself ready to continue as B. The corresponding global type $\tilde{x} \to y(G); H$ types the corresponding choreographic interaction. The exceptions are types 0 and $0\top$, which are not assigned to any process (0) or choreography ($0\top$): the inclusion of 0 is justified by the necessity of having a dual type to $\top$ (see below), while the type $0\top$ is essential for the definition of coherence (see below). We use type variables also to represent concrete datatypes.

Each local type A can be transformed into a formula in linear logic $|A|$ by erasing all process annotations. These logic formulas enjoy the usual notion of duality, where a formula's dual is obtained by recursively replacing each connector by the other one in the same row in Fig. 1. For example, the dual of $!(A \otimes 0)$ is $?(A^{\perp} \parr \top)$, where $A^{\perp}$ is the dual of formula A. The set of endpoints occurring as annotations in type A is denoted $\mathsf{ann}(A)$.

3.2 Typing

Terms of MCC are typed in judgements of the form $\Sigma \Vdash P \vdots \Psi$, where:

- Σ is a set of session typings of the form $(x_i)_i : G$;
- P is a proof term (a choreography or a process);
- Ψ is a hypersequent—a parallel composition of classical linear logic sequents.

Intuitively, $\Sigma \Vdash P \vdots \Psi$ reads as: Ψ types the program P under the sessions described in Σ.

Before presenting the typing rules of MCC, we make a few general remarks. The notion of hypersequent used in MCC is similar to the one presented in [5] for intuitionistic linear logic, with a variation on the treatment of endpoints participating in a session: instead of marking names that are engaged in a session, we store this information in Σ. Given a judgment $\Sigma \Vdash P \vdots \Psi \mid \vdash \Gamma, x : A$, checking whether x is available and not engaged in a session can be done by verifying that x does not occur in the domain of Σ. It should be noted that names cannot occur more than once in Σ: each endpoint x may only belong to (at most) one session G. Hypersequents Ψ_1, Ψ_2 and sets of sessions Σ_1, Σ_2 can only be joined if the names used in their domains do not repeat.

We divide the typing rules for MCC into three groups: structural rules, rules for the action fragment and rules for the interaction fragment. Fig. 2 details structural rules, which type sessions. Rule Ax types a communication between two processes with dual types. The remaining two rules, Conn and Scope, replace the Cut rule typically found in linear logic – and so are best understood as two complementary rules. Read bottom-to-top, rule Scope initiates a session in a choreography, storing its type in Σ, while rule Conn splits the choreography of processes involved in a session into parallel, independent processes. Rule Conn can alternatively be read top-to-bottom: it merges several independent processes

$$\frac{|A|^{\perp} = |B|}{\cdot \Vdash x^{A} \to y^{B} \;\vdots\; \vdash x:A, y:B}\ \mathsf{Ax}$$

$$\frac{(\Sigma_i \Vdash P_i \;\vdots\; \Psi_i \mid \vdash \Gamma_i, x_i:A_i)_i \qquad G \vDash (x_i:A_i)_i}{(\Sigma_i)_i, (x_i)_i : G \Vdash (P_i)_i \;\vdots\; (\Psi_i \mid \vdash \Gamma_i, x_i:A_i)_i}\ \mathsf{Conn}$$

$$\frac{\Sigma, (x_i)_i : G \Vdash P \;\vdots\; \Psi \mid (\vdash \Gamma_i, x_i:A_i)_i}{\Sigma \Vdash (\boldsymbol{\nu} \tilde{x}^{\tilde{A}} : G)\, P \;\vdots\; \Psi \mid \vdash (\Gamma_i)_i}\ \mathsf{Scope}$$

Fig. 2. Structural Rules

into a joint session, provided their types are provably coherent – a notion we introduce later in this section. When a typing derivation is built bottom-up, rule Scope is applied to store a type of a session in Σ. The choreography rules (presented below) then destruct the session type in parallel with the choreography, until the session is split among its participants by rule Conn. The types A_i in that rule are the part of the original session type that has not yet been "consumed".

$$\frac{\Sigma_1 \Vdash P \;\vdots\; \Psi_1 \mid \vdash \Gamma_1, x':A \qquad \Sigma_2 \Vdash Q \;\vdots\; \Psi_2 \mid \vdash \Gamma_2, x:B}{\Sigma_1, \Sigma_2 \Vdash \overline{x^y}(x'; y', \tilde{z}'); (P \mid Q) \;\vdots\; \Psi_1 \mid \Psi_2 \mid \vdash \Gamma_1, \Gamma_2, x:A \otimes^{y} B}\ \otimes$$

$$\frac{\Sigma \Vdash P \;\vdots\; \Psi \mid \vdash \Gamma, y':A, y:B}{\Sigma \Vdash y^{\tilde{x}}(\tilde{x}'; y'); P \;\vdots\; \Psi \mid \vdash \Gamma, y:A \parr^{\tilde{x}} B}\ \parr$$

$$\frac{}{\cdot \Vdash \mathsf{close}[x^y] \;\vdots\; \vdash x:\mathbf{1}^{y}}\ \mathbf{1} \qquad \frac{\Sigma \Vdash P \;\vdots\; \Psi \mid \vdash \Gamma}{\Sigma \Vdash \mathsf{wait}[y^{\tilde{x}}]; P \;\vdots\; \Psi \mid \vdash \Gamma, y:\perp^{\tilde{x}}}\ \perp$$

$$\frac{\Sigma \Vdash P \;\vdots\; \Psi \mid \vdash \Gamma, x:A}{\Sigma \Vdash x^{\tilde{y}}.\mathsf{inl}; P \;\vdots\; \Psi \mid \vdash \Gamma, x:A \oplus^{\tilde{y}} B}\ \oplus_1 \qquad \frac{\Sigma \Vdash P \;\vdots\; \Psi \mid \vdash \Gamma, x:B}{\Sigma \Vdash x^{\tilde{y}}.\mathsf{inr}; Q \;\vdots\; \Psi \mid \vdash \Gamma, x:A \oplus^{\tilde{y}} B}\ \oplus_2$$

$$\frac{\Sigma \Vdash P \;\vdots\; \Psi \mid \vdash \Gamma, y:A \qquad \Sigma \Vdash Q \;\vdots\; \Psi \mid \vdash \Gamma, y:B}{\Sigma \Vdash y^{x}.\mathsf{case}(P, Q) \;\vdots\; \Psi \mid \vdash \Gamma, y:A \,\&^{x}\, B}\ \&$$

$$\frac{\cdot \Vdash P \;\vdots\; ?\Psi \mid \vdash ?\Gamma, y:A}{\cdot \Vdash \mathsf{srv}\, y^{x}; P \;\vdots\; ?\Psi \mid \vdash ?\Gamma, y:!^{x}A}\ ! \qquad \frac{\Sigma \Vdash P \;\vdots\; \Psi \mid \vdash \Gamma, x:A}{\Sigma \Vdash \mathsf{use}\, x^{\tilde{y}}; P \;\vdots\; \Psi \mid \vdash \Gamma, x:?^{\tilde{y}}A}\ ?$$

$$\frac{\Sigma \Vdash P \;\vdots\; \Psi \mid \vdash \Gamma}{\Sigma \Vdash \mathsf{kill}\, x^{\tilde{y}}; P \;\vdots\; \Psi \mid \vdash \Gamma, x:?^{\tilde{y}}A}\ \mathsf{Weaken} \qquad \frac{\Sigma \Vdash P \;\vdots\; \Psi \mid \vdash \Gamma, x:?^{\tilde{y}}A, x':?^{\tilde{y}'}A}{\Sigma \Vdash \mathsf{clone}\, x^{\tilde{y}}(x', \tilde{y}'); P \;\vdots\; \Psi \mid \vdash \Gamma, x:?^{\tilde{y}}A}\ \mathsf{Contract}$$

$$\frac{\Sigma \Vdash P \;\vdots\; \Psi \mid \vdash \Gamma, x:B\{A/X\}}{\Sigma \Vdash x^{\tilde{y}}[A]; P \;\vdots\; \Psi \mid \vdash \Gamma, x:\exists^{\tilde{y}} X.B}\ \exists \qquad \frac{\Sigma \Vdash P \;\vdots\; \Psi \mid \vdash \Gamma, y:B \qquad X \notin \mathsf{fv}(\Gamma)}{\Sigma \Vdash y^{x}[X]; P \;\vdots\; \Psi \mid \vdash \Gamma, y:\forall^{x} X.B}\ \forall$$

$$\text{no rule for } \mathbf{0} \qquad \frac{}{\cdot \Vdash x^{y}.\mathsf{case}() \;\vdots\; \vdash \Gamma, x:\top^{y}}\ \top$$

Fig. 3. Rules for the Action Fragment

Fig. 3 details the rules for typing the action fragment of MCC, containing its process terms (the first two columns in Fig. 1). In this fragment, we work with processes not involved in any session – the endpoints explicitly mentioned in proof terms can not occur in the domain of Σ. These rules do not affect ongoing sessions, except for $\otimes$, where existing sessions have to be split among the two

processes, combined in the conclusion of the rule. The types for the different actions follow the intuitions given earlier, the rules themselves being similar to ones found e.g. in [3, 18]. Notation $?\Gamma$ means that all endpoint types in Γ begin with ?, similarly $?\Psi$ requires that all sequents in Ψ are of the form $?\Gamma$. Processes can only terminate after all their sessions are closed (so Σ is empty).

Linear Fragment:

$$\frac{\Sigma, \boxed{(\tilde{x}, y, \tilde{u}) : H, (\tilde{x}', y') : G\{\tilde{x}'/\tilde{x}, y'/y\}} \Vdash P \ \vdots\ \Psi \mid \underline{(\vdash \Gamma_{i1}, x_i' : A_i)_i} \mid (\vdash \Gamma_{i2}, x_i : B_i)_i \mid \vdash \Gamma, \underline{y' : C, y : D}}{\Sigma, \boxed{(\tilde{x}, y, \tilde{u}) : \tilde{x} \to y(G); H} \Vdash \tilde{x}(\tilde{x}') \to y(y'); P \ \vdots\ \Psi \mid \underline{(\vdash \Gamma_{i1}, \Gamma_{i2}, x_i : A_i \otimes^y B_i)_i} \mid \vdash \Gamma, \underline{y : C \parr^{\tilde{x}} D}} \ \mathsf{C}_{\otimes\parr}$$

$$\frac{\Sigma \Vdash P \ \vdots\ \Psi \mid \vdash \Gamma}{\Sigma, \boxed{(\tilde{x}, y) : \tilde{x} \to y} \Vdash \tilde{x} \text{ closes } y; P \ \vdots\ \Psi \mid \underline{(\vdash x_i : \mathbf{1}^y)_i} \mid \vdash \Gamma, \underline{y : \bot^{\tilde{x}}}} \ \mathsf{C}_{\mathbf{1}\bot}$$

$$\frac{\Sigma_1, \Sigma_2, \boxed{(x, \tilde{y}, \tilde{u}) : G} \Vdash P \ \vdots\ \Psi_1 \mid \Psi_2 \mid \underline{\vdash \Gamma, x : A} \mid \underline{(\vdash \Gamma_i, y_i : C_i)_i} \quad \Sigma_2, \boxed{(x, \tilde{y}, \tilde{u}) : H} \Vdash Q \ \vdots\ \Psi_2 \mid \underline{(\vdash \Gamma_i, y_i : D_i)_i}}{\Sigma_1, \Sigma_2, \boxed{(x, \tilde{y}, \tilde{u}) : x \to \tilde{y}.\mathsf{case}(G, H)} \Vdash x \to \tilde{y}.\mathsf{inl}(P, Q) \ \vdots\ \Psi_1 \mid \Psi_2 \mid \vdash \Gamma, \underline{x : A \oplus^{\tilde{y}} B} \mid \underline{(\vdash \Gamma_i, y_i : C_i \ \&^x \ D_i)_i}} \ \mathsf{C}^1_{\oplus\&}$$

$$\frac{\Sigma_1, \boxed{(x, \tilde{y}, \tilde{u}) : G} \Vdash P \ \vdots\ \Psi_1 \mid \underline{(\vdash \Gamma_i, y_i : C_i)_i} \quad \Sigma_1, \Sigma_2, \boxed{(x, \tilde{y}, \tilde{u}) : H} \Vdash Q \ \vdots\ \Psi_1 \mid \Psi_2 \mid \vdash \Gamma, \underline{x : B} \mid \underline{(\vdash \Gamma_i, y_i : D_i)_i}}{\Sigma_1, \Sigma_2, \boxed{(x, \tilde{y}, \tilde{u}) : x \to \tilde{y}.\mathsf{case}(G, H)} \Vdash x \to \tilde{y}.\mathsf{inr}(P, Q) \ \vdots\ \Psi_1 \mid \Psi_2 \mid \vdash \Gamma, \underline{x : A \oplus^{\tilde{y}} B} \mid \underline{(\vdash \Gamma_i, y_i : C_i \ \&^x \ D_i)_i}} \ \mathsf{C}^2_{\oplus\&}$$

Exponential Fragment:

$$\frac{\Sigma, \boxed{(x, \tilde{y}) : G} \Vdash P \ \vdots\ \Psi \mid \vdash \Gamma, \underline{x : A} \mid \underline{(\vdash ?\Gamma_i, y_i : B_i)_i}}{\Sigma, \boxed{(x, \tilde{y}) : !x \to \tilde{y}(G)} \Vdash x \text{ starts } \tilde{y}; P \ \vdots\ \Psi \mid \vdash \Gamma, \underline{x : ?^{\tilde{y}} A} \mid \underline{(\vdash ?\Gamma_i, \ y_i : !^x B_i \ )_i}} \ \mathsf{C}!?$$

$$\frac{\Sigma \Vdash P \ \vdots\ \Psi \mid \vdash \Gamma \quad \Big(\cdot \Vdash Q_i \ \vdots\ ?\Psi_i \mid \underline{\vdash ?\Gamma_i, y_i : B_i} \Big)_i}{\Sigma, \boxed{(x, \tilde{y}) : !x \to \tilde{y}(G)} \Vdash x \text{ kills } \widetilde{y_i(Q_i)}; P \ \vdots\ \Psi \mid (?\Psi_i)_i \mid \vdash \Gamma, \underline{x : ?^{\tilde{y}} A} \mid \underline{(\vdash ?\Gamma_i, \ y_i : !^x B_i \ )_i}} \ \mathsf{C}_{!\mathsf{w}}$$

$$\frac{\Sigma, \begin{array}{l}\boxed{(x, \tilde{y}) : !x \to \tilde{y}(G)}, \\ \boxed{(x', \tilde{y}') : !x' \to \tilde{y}'(G\sigma)}\end{array} \Vdash P \ \vdots\ \Psi \mid \begin{array}{l}(?\Psi_i \mid ?\Psi_i\sigma)_i \mid \vdash \Gamma, \underline{x : ?^{\tilde{y}} A, x' : ?^{\tilde{y}'} A} \\ \underline{\Big(\vdash ?\Gamma_i, \ y_i : !^x B_i \mid \vdash ?\Gamma_i\sigma, \ y_i' : !^{x'} B_i\sigma \Big)_i}\end{array} \quad \sigma = \{x'/x, \tilde{y}'/\tilde{y}\}}{\Sigma, \boxed{(x, \tilde{y}) : !x \to \tilde{y}(G)} \Vdash x \text{ clones } \tilde{y}(x', \tilde{y}'); P \ \vdots\ \Psi \mid \ \mid (?\Psi_i)_i \vdash \Gamma, \underline{x : ?^{\tilde{y}} A} \mid \underline{(\vdash ?\Gamma_i, \ y_i : !^x B_i \ )_i}} \ \mathsf{C}_{!\mathsf{C}}$$

Quantifiers:

$$\frac{\Sigma, \boxed{(x, \tilde{y}) : G\{A/X\}} \Vdash P\{A/X\} \ \vdots\ \Psi \mid \vdash \Gamma, \underline{x : B\{A/X\}} \mid \underline{(\vdash \Gamma_i, y_i : B_i\{A/X\})_i}}{\Sigma, \boxed{(x, \tilde{y}) : x \to y.(X)G} \Vdash x[A] \to \tilde{y}(X); P \ \vdots\ \Psi \mid \vdash \Gamma, \underline{x : \exists^{\tilde{y}} X.B} \mid \underline{(\vdash \Gamma_i, y_i : \forall^x X.B_i)_i}} \ \mathsf{C}_{\exists\forall}$$

Fig. 4. Rules for the Interaction Fragment

Fig. 4 details the rules for typing choreography terms. At the process level, each of these rules combines two rules from the action fragment of MCC. Differently from the previous set, these rules now also look at Σ to check that the interactions described conform to the types of the ongoing sessions. In rule $\mathsf{C}_{\mathbf{1}\bot}$, we close a session (removed from Σ) and terminate all processes involved in it. Rule $\mathsf{C}_{\otimes\parr}$ types the creation of a new session with protocol G, created among endpoints $\tilde{z}$ and w; this session is stored in Σ, while the process types are updated as in rules $\otimes$ and $\parr$ above. The remaining rules in the linear fragment are similarly understood. Exponentials give rise to three rules, all of them combining

! with another rule. In rule C!?, process x invokes the services provided by $\tilde{y}$, creating a new session among these processes with type G. Rule $\mathsf{C}_{!\mathsf{w}}$ combines **!** with Weaken: here the processes providing the service are simply removed from the context. Rule **!** combines **!** with Contract, allowing a service to be duplicated. Finally, rule $\mathsf{C}_{\exists\forall}$ allows x to pass values to parameters in $\tilde{y}$, which are then instantiated by a fresh name A throughout the whole judgement.

3.3 Coherence

$$\frac{G \vDash (x_i : A_i)_i,\ y : C \qquad H \vDash \Gamma,\ (x_i : B_i)_i,\ y : D}{\tilde{x} \to y(G); H \vDash \Gamma,\ (x_i : A_i \otimes^{y} B_i)_i,\ y : C \parr^{\tilde{x}} D}\ \otimes\parr \qquad \frac{}{\tilde{x} \to y \vDash (x_i : \mathbf{1}^{y})_i, y : \bot^{\tilde{x}}}\ \mathbf{1}\bot$$

$$\frac{G_1 \vDash \Gamma, x : A, (y_i : C_i)_i \qquad G_2 \vDash \Gamma, x : B, (y_i : D_i)_i}{x \to \tilde{y}.\mathsf{case}(G_1, G_2) \vDash \Gamma, x : A \oplus^{\tilde{y}} B, (y_i : C_i \,\&^{x}\, D_i)_i}\ \oplus\& \qquad \frac{G \vDash x : A, (y_i : B_i)_i}{!x \to \tilde{y}(G) \vDash x : ?^{\tilde{y}} A, (y_i : !^{x} B_i)_i}\ !?$$

$$\frac{}{x \to \tilde{y}.\mathsf{case}() \vDash \Gamma, x : 0^{\tilde{y}}, (y_i : \top^{x})_i}\ 0\top \qquad \frac{G \vDash \Gamma, x : A, (y_i : B_i)_i \qquad X \notin \mathsf{ftv}(\Gamma)}{x \to \tilde{y}.(X)G \vDash \Gamma, x : \exists^{\tilde{y}} X.A, (y_i : \forall^{x} X.B_i)_i}\ \exists\forall$$

$$\frac{|A|^{\perp} = |B| \qquad \mathsf{ann}(A) = \{y\} \qquad \mathsf{ann}(B) = \{x\}}{x^{A} \to y^{B} \vDash x : A, y : B}\ \textsc{Axiom}$$

Fig. 5. Coherence Rules

In Fig. 5, we report the rules defining the coherence relation $\vDash$. These rules follow the intuitions given above, relating global types to their matching choreography actions, when viewed as sets of individual process actions. In the rule dealing with axioms, endpoints x and y can communicate only with each other.

3.4 Examples

OpenID. The OpenID example from § 2 is typable by the global type G (also in § 2). In particular, we observe that the main endpoints can be typed as:

$$u : ?^{rp,ip}(\mathsf{String}^{\perp} \otimes^{rp} (\mathsf{String}^{\perp} \otimes^{ip} (\mathsf{Pwd}^{\perp} \otimes^{ip} (\mathsf{String}^{\perp} \otimes^{rp} B))))$$
$$rp : !^{u}(\mathsf{String} \parr^{u} ((\mathsf{String} \parr^{u} C) \&^{ip} D_1)) \qquad ip : !^{u}(\mathsf{String} \parr^{u} (\mathsf{Pwd} \parr^{u} (A \oplus^{rp} D_2)))$$
$$u' : ?^{s}(\mathsf{String}^{\perp} \otimes^{s} (\mathsf{String} \parr^{s} \perp)) \qquad s : !^{u'}(\mathsf{String} \parr (\mathsf{String}^{\perp} \otimes^{u'} \mathbf{1}))$$

Polymorphism. In order to show how polymorphic behaviour looks at choreographic level, we recall an example with binary sessions by Wadler [18], which implements Church numerals as polymorphic processes. The parallel composition of a Church numeral with a counter process will return the corresponding natural number, while the composition of a Church numeral with a pinger process will ping the number of times corresponding to such number. We can write such

compositions in terms of choreographies. E.g., composing the Church numeral corresponding to one with a counter can be written choreographically as:

$$(\boldsymbol{\nu} xy)\begin{pmatrix} x[\mathsf{Nat}] \to y(X);\ x(s_x) \to y(s_y);\ x(z_x) \to y(z_y);\ s_y \,\mathsf{starts}\, s_x; \\ s_x(a_x) \to s_y(a_y);\ \left(\mathsf{incr}_{a_y,s_y} \mid \mathsf{nought}_{z_x} \mid P\right) \end{pmatrix}$$

where incr_{a_y,s_y} receives a natural number on a_y and sends its successor on s_y, while nought_{z_x} repeatedly outputs zero over z_x. Instead, in the case of ping:

$$(\boldsymbol{\nu} xy)\begin{pmatrix} x[\mathbf{1}] \to y(X);\ x(s_x) \to y(s_y);\ x(z_x) \to y(z_y);\ s_y \,\mathsf{starts}\, s_x; \\ s_x(a_x) \to s_y(a_y);\ \left(a_y \,\mathsf{closes}\, a_x;\ \mathsf{use}\, w^v;\ \mathsf{wait}[w^v];\ s_x \,\mathsf{closes}\, s_y; \right. \\ \left. z_x \,\mathsf{closes}\, z_y;\ y \,\mathsf{closes}\, x;\ \mathsf{close}[u^{u'}]\right) \end{pmatrix}$$

where w and u are endpoints used for transmitting the ping to v and u'.

4 Semantics

We describe the operational semantics of MCC. As usual, this semantics is obtained from cases of the proof of cut elimination: the principal cases describe reductions ($\longrightarrow$), while the permutations of rule applications give rise to the rules for structural equivalence ($\equiv$). We present these rules below, and show that they conform to the informal semantics described in § 3. Following [3, 5], in MCC we split the usual Cut rule into two components, Conn and Scope, resulting in slightly more complex semantics for the $\equiv$ fragment.

4.1 Reductions

We begin by presenting reductions for both processes and choreographies.

Processes. The semantics of the action fragment of our calculus is presented in Fig. 6, and follows the intuitions about process behaviours given in § 3.1. For the sake of readability, we omit the type annotations from the restrictions. Notice that the β-reductions are coordinated by a global type, as they correspond to multiple parties communicating. In the first reduction, one for new session creation, the notation $\tilde{x}'_{\setminus i}$ denotes $\tilde{x}'$ without the component x'_i.[1]

The final two rules in the action fragment make use of the axiom case of coherence. Note that the reduction can only be applied directly when the axiom rule is used with type variables. This is because when the axiom rule uses general types, annotations can be different. In particular, they may refer to sessions with different number of endpoints, e.g., $A_1 \otimes^x A_2 \parr^y$ and $A_1 \otimes^z A_2 \parr^z$. Therefore, allowing a substitution such as that of the final two rules may make the resulting

[1] It may be surprising that some of the rules also include a restriction also to a vector $\tilde{z}$, and a session using a vector of processes $\tilde{S}$, whose shape we do not inspect. This follows from the shape of coherence rules, described in previous section: rules such as $\otimes\parr$, $\oplus\&$, $0\top$ and $\exists\forall$ contain an additional context Γ, captured here by $\tilde{z}$.

$$
\begin{array}{ll}
(\boldsymbol{\nu}\tilde{x}, y, \tilde{z} : \tilde{x} \to y(G); H)\,(\overline{x_i^y}(x_i'; y', \tilde{x}'_{\setminus i}); (P_i \mid Q_i))_i \mid y^{\tilde{x}}(\tilde{x}'; y'); R \mid \tilde{S} & \\
\qquad \longrightarrow & \\
\qquad (\boldsymbol{\nu}\tilde{x}', y' : G\{\tilde{x}'/\tilde{x}, y'/y\})\,\tilde{P} \mid (\boldsymbol{\nu}\tilde{x}, y, \tilde{z} : H)\,\tilde{Q} \mid R \mid \tilde{S} & \\
(\boldsymbol{\nu}\tilde{x}, y : \tilde{x} \to y)\,(\mathsf{close}[x_i^y])_i \mid \mathsf{wait}[y^{\tilde{x}}]; P & \longrightarrow P \\
(\boldsymbol{\nu}x, \tilde{y}, \tilde{z} : x \to \tilde{y}.\mathsf{case}(G, H))\,x^{\tilde{y}}.\mathsf{inl}; P \mid (y_i^x.\mathsf{case}(Q_i, R_i))_i \mid \tilde{S} & \\
\qquad \longrightarrow & (\boldsymbol{\nu}x, \tilde{y}, \tilde{z} : G)\,P \mid \tilde{Q} \mid \tilde{S} \\
(\boldsymbol{\nu}x, \tilde{y}, \tilde{z} : x \to \tilde{y}.\mathsf{case}(G, H))\,x^{\tilde{y}}.\mathsf{inr}; Q \mid (y_i^x.\mathsf{case}(Q_i, R_i))_i \mid \tilde{S} & \\
\qquad \longrightarrow & (\boldsymbol{\nu}x, \tilde{y}, \tilde{z} : H)\,P \mid \tilde{R} \mid \tilde{S} \\
(\boldsymbol{\nu}x, \tilde{y} : !x \to \tilde{y}(G))\,\mathsf{use}\,x^{\tilde{y}}; P \mid (\mathsf{srv}\,y_i{}^x; Q_i)_i & \longrightarrow (\boldsymbol{\nu}x, \tilde{y} : G)\,P \mid \tilde{Q} \\
(\boldsymbol{\nu}x, \tilde{y} : !x \to \tilde{y}(G))\,\mathsf{kill}\,x^{\tilde{y}}; P \mid (\mathsf{srv}\,y_i{}^x; Q_i)_i & \longrightarrow P \\
(\boldsymbol{\nu}x, \tilde{y} : !x \to \tilde{y}(G))\,\mathsf{clone}\,x^{\tilde{y}}(x', \tilde{y}'); P \mid (\mathsf{srv}\,y_i{}^x; Q_i)_i & \longrightarrow \\
\quad (\boldsymbol{\nu}x, \tilde{y} : !x \to \tilde{y}(G))\,(\boldsymbol{\nu}x', \tilde{y}' : !x' \to \tilde{y}'(G\{x'/x, \tilde{y}'/\tilde{y}\}))\,P \mid (\mathsf{srv}\,y_i{}^x; Q_i)_i \mid (\mathsf{srv}\,y_i'{}^{x'}; Q_i\{x'/x, \tilde{y}'/\tilde{y}\})_i & \\
(\boldsymbol{\nu}x, \tilde{y}, \tilde{z} : x \to \tilde{y}.(X)G)\,x^{\tilde{y}}[A]; P \mid (y_i{}^x[X]; Q_i)_i \mid \tilde{S} & \longrightarrow (\boldsymbol{\nu}x, \tilde{y}, \tilde{z} : G\{A/X\})\,P \mid \tilde{Q}\{A/X\} \mid \tilde{S}\{A/X\} \\
(\boldsymbol{\nu}x, y : x^X \to y^{X^\perp})\,w^{X^\perp} \to x^X \mid Q & \longrightarrow Q\{w/y\} \\
(\boldsymbol{\nu}x, y : y^X \to x^{X^\perp})\,x^{X^\perp} \to w^X \mid Q & \longrightarrow Q\{w/y\}
\end{array}
$$

Fig. 6. Semantics for the Action Fragment

term untypable. As a consequence, in order to reduce processes we require both the global type and the linking process to be η-expanded. Rules for η-expanding proof terms in MCC using standard expansion techniques and for expanding global types are presented in Fig. 7. An alternative way of handling the axiom cases is to allow both the structural typing rule Ax and the coherence axiom, to operate only on type variables, and show admissibility of the generalized rules as given in Fig. 2 and Fig. 5.

Choreographies. Fig. 8 defines the semantics for the interaction fragment of MCC, which again respect the intuitions from § 3.1. From a proof-theoretical perspective, these reductions correspond to proof transformations of C-rules from Fig. 4 followed by a structural Scope rule; the transformation removes the C rule and pushes the Scope rule higher up in the proof tree. For instance, in a choreography for new session creation, the transformation is the following:

$$
\dfrac{\dfrac{\dfrac{\mathcal{D}}{\Sigma, (\tilde{x}, y, \tilde{u}) : H, (\tilde{x}', y') : G\{\tilde{x}'/\tilde{x}, y'/y\} \Vdash P \circ \Psi \mid (\vdash \Gamma_{i1}, x_i' : A_i)_i \mid (\vdash \Gamma_{i2}, x_i : B_i)_i \mid \vdash \Gamma, y' : C, y : D \mid (\vdash \Gamma_j', u_j : E_j)_j}\ \mathsf{C}_{\otimes\parr}}{\Sigma, (\tilde{x}, y, \tilde{u}) : \tilde{x} \to y(G); H \Vdash \tilde{x}(\tilde{x}') \to y(y'); P \circ \Psi \mid (\vdash \Gamma_{i1}, \Gamma_{i2}, x_i : A_i \otimes^y B_i)_i \mid \vdash \Gamma, y : C \parr^{\tilde{x}} D \mid (\vdash \Gamma_j', u_j : E_j)_j}\ \mathsf{Scope}}{\Sigma \Vdash (\boldsymbol{\nu}\tilde{x}^{\bar{A} \otimes \bar{B}} y^{C \parr^{\tilde{x}} D} \tilde{u}^{\bar{E}} : \tilde{x} \to y(G); H)\,\tilde{x}(\tilde{x}') \to y(y'); P \circ \Psi \mid \vdash (\Gamma_{i1}, \Gamma_{i2})_i, \Gamma, (\Gamma_j')_j}
$$

$$
\longrightarrow
$$

$$
\dfrac{\dfrac{\dfrac{\mathcal{D}}{\Sigma, (\tilde{x}, y, \tilde{u}) : H, (\tilde{x}', y') : G\{\tilde{x}'/\tilde{x}, y'/y\} \Vdash P \circ \Psi \mid (\vdash \Gamma_{i1}, x_i' : A_i)_i \mid (\vdash \Gamma_{i2}, x_i : B_i)_i \mid \vdash \Gamma, y' : C, y : D \mid (\vdash \Gamma_j', u_j : E_j)_j}\ \mathsf{Scope}}{\Sigma, (\tilde{x}', y') : G\{\tilde{x}'/\tilde{x}, y'/y\} \Vdash (\boldsymbol{\nu}\tilde{x}^{\bar{B}}, y^D, \tilde{u}^{\bar{E}} : H)\,P \circ \Psi \mid (\vdash \Gamma_{i1}, x_i' : A_i)_i \mid \vdash \Gamma, y' : C, (\Gamma_{i2})_i, (\Gamma_j')_j}\ \mathsf{Scope}}{\Sigma \Vdash (\boldsymbol{\nu}\tilde{x}'^{\bar{A}} y'^{C} : G\{\tilde{x}'/\tilde{x}, y'/y\})\,(\boldsymbol{\nu}\tilde{x}^{\bar{B}} y^D \tilde{u}^{\bar{E}} : H)\,P \circ \Psi \mid \vdash (\Gamma_{i1}, \Gamma_{i2})_i, \Gamma, (\Gamma_j')_j}
$$

$$
\begin{array}{ll}
\eta_{\mathsf{exp}}(x^{A \otimes^{v} B} \to y^{C \parr^{\tilde{z}} D}) & = y^{\tilde{z}}(x';y');\overline{x^{v}}(x';y');(\,\eta_{\mathsf{exp}}(x'^{A} \to y'^{C}) \mid \eta_{\mathsf{exp}}(x^{B} \to y^{D})) \\
\eta_{\mathsf{exp}}(x^{\mathbf{1}^{v}} \to y^{\bot^{\tilde{z}}}) & = \mathsf{wait}[y^{\tilde{z}}];\mathsf{close}[x^{v}] \\
\eta_{\mathsf{exp}}(x^{A \oplus^{\tilde{z}} B} \to y^{C \& ^{v} D}) & = y^{v}.\mathsf{case}(x^{\tilde{z}}.\mathsf{inl};\eta_{\mathsf{exp}}(x^{A} \to y^{C}), x^{\tilde{z}}.\mathsf{inr};\eta_{\mathsf{exp}}(x^{B} \to y^{D})) \\
\eta_{\mathsf{exp}}(x^{!^{v} A} \to y^{?^{\tilde{z}} B}) & = \mathsf{srv}\,x^{v};\mathsf{use}\,y^{\tilde{z}};\eta_{\mathsf{exp}}(x^{A} \to y^{B}) \\
\eta_{\mathsf{exp}}(x^{0^{\tilde{z}}} \to y^{\top^{v}}) & = y^{v}.\mathsf{case}() \\
\eta_{\mathsf{exp}}(x^{\exists^{\tilde{z}} X.A} \to y^{\forall^{v} X.B}) & = y^{v}[X];x^{\tilde{z}}[X];\eta_{\mathsf{exp}}(x^{A} \to y^{B}) \\
\eta_{\mathsf{exp}}(x^{X} \to y^{X^{\bot}}) & = x^{X} \to y^{X^{\bot}} \\
\eta_{\mathsf{exp}}(x^{X^{\bot}} \to y^{X}) & = x^{X^{\bot}} \to y^{X}
\end{array}
$$

$$
\begin{array}{ll}
\eta_{\mathsf{exp}}(x^{A \otimes^{y} B} \to y^{C \parr^{x} D}) & = x \to y(\eta_{\mathsf{exp}}(x^{A} \to y^{C}));\eta_{\mathsf{exp}}(x^{B} \to y^{D}) \\
\eta_{\mathsf{exp}}(x^{\mathbf{1}^{y}} \to y^{\bot^{x}}) & = x \to y \\
\eta_{\mathsf{exp}}(x^{A \oplus^{y} B} \to y^{C \& ^{x} D}) & = x \to y.\mathsf{case}(\eta_{\mathsf{exp}}(x^{A} \to y^{C}), \eta_{\mathsf{exp}}(x^{B} \to y^{D})) \\
\eta_{\mathsf{exp}}(x^{!^{y} A} \to y^{?^{x} B}) & = !x \to y(\eta_{\mathsf{exp}}(x^{A} \to y^{B})) \\
\eta_{\mathsf{exp}}(x^{0^{y}} \to y^{\top^{x}}) & = x \to y.\mathsf{case}() \\
\eta_{\mathsf{exp}}(x^{\exists^{y} X.A} \to y^{\forall^{x} X.B}) & = x \to y.(X)\,\eta_{\mathsf{exp}}(x^{A} \to y^{B}) \\
\eta_{\mathsf{exp}}(x^{X} \to y^{X^{\bot}}) & = x^{X} \to y^{X^{\bot}} \\
\eta_{\mathsf{exp}}(x^{X^{\bot}} \to y^{X}) & = x^{X^{\bot}} \to y^{X}
\end{array}
$$

Fig. 7. η-expansion for Local Types and Global Types

$$
\begin{array}{lll}
(\boldsymbol{\nu}\tilde{x}, y, \tilde{z} : \tilde{x} \to y(G); H)\,(\tilde{x}(\tilde{x}') \to y(y'); P) & \longrightarrow & (\boldsymbol{\nu}\tilde{x}', y' : G\{\tilde{x}'/\tilde{x}, y'/y\})\,(\boldsymbol{\nu}\tilde{x}, y, \tilde{z} : H)\,P \\
(\boldsymbol{\nu}\tilde{x}, y : \tilde{x} \to y)\,(\tilde{x}\ \mathsf{closes}\ y; P) & \longrightarrow & P \\
(\boldsymbol{\nu}x, \tilde{y}, \tilde{z} : x \to \tilde{y}.\mathsf{case}(G, H))\,(x \to \tilde{y}.\mathsf{inl}(P, Q)) & \longrightarrow & (\boldsymbol{\nu}x, \tilde{y}, \tilde{z} : G)\,P \\
(\boldsymbol{\nu}x, \tilde{y}, \tilde{z} : x \to \tilde{y}.\mathsf{case}(G, H))\,(x \to \tilde{y}.\mathsf{inr}(P, Q)) & \longrightarrow & (\boldsymbol{\nu}x, \tilde{y}, \tilde{z} : G)\,Q \\
(\boldsymbol{\nu}x, \tilde{y} : !x \to \tilde{y}(G))\,(x\ \mathsf{starts}\ \tilde{y}; P) & \longrightarrow & (\boldsymbol{\nu}x, \tilde{y} : G)\,P \\
(\boldsymbol{\nu}x, \tilde{y} : !x \to \tilde{y}(G))\,\left(x\ \mathsf{kills}\ \widetilde{y_i(Q_i)}; P\right) & \longrightarrow & P \\
(\boldsymbol{\nu}x, \tilde{y} : !x \to \tilde{y}(G))\,x\ \mathsf{clones}\ \tilde{y}(x', \tilde{y}'); P & \longrightarrow & \\
\multicolumn{3}{r}{(\boldsymbol{\nu}x, \tilde{y} : !x \to \tilde{y}(G))\,(\boldsymbol{\nu}x', \tilde{y}' : !x' \to \tilde{y}'(G\{x'/x, \tilde{y}'/\tilde{y}\}))\,P} \\
(\boldsymbol{\nu}x, \tilde{y} : x \to \tilde{y}.(X)G)\,(x[A] \to \tilde{y}(X); P) & \longrightarrow & (\boldsymbol{\nu}x, \tilde{y} : G\{A/X\})\,(P\{A/X\})
\end{array}
$$

Fig. 8. Semantics for the Interaction Frament

4.2 Structural equivalence

The reductions given earlier require that programs can be written in the very specific form given in their left-hand side. Formally, this is achieved by closing $\longrightarrow$ under structural equivalence, which we define below: if $P \equiv P'$, $P' \longrightarrow Q'$ and $Q' \equiv Q$, then $P \longrightarrow Q$.

The majority of rules for structural equivalence are derived from the proof of cut-elimination for linear logic. The deconstruction of the Cut rule into Conn and Scope results in an altered structural equivalence relation: instead of permuting applications of Cut with rules applicable directly before (or after) it in the proof tree, we have two sets of such permutations—one for Conn and one for Scope. Intuitively, applications of Conn and Scope have to meet before they can be eliminated. A selection of the resulting structural equivalences, induced by com-

$$
\begin{array}{lll}
[\mathsf{Conn}/\mathsf{C}_{\otimes\parr}] & \tilde{x}(\tilde{x}') \to y(y'); P \mid \tilde{Q} \equiv \tilde{x}(\tilde{x}') \to y(y'); (P \mid \tilde{Q}) & (\tilde{x}', y' \notin \mathsf{fn}\,\tilde{Q}) \\
[\mathsf{Conn}/\mathsf{C}_{\mathbf{1}\bot}] & \tilde{x}\ \mathsf{closes}\ y; P \mid \tilde{Q} \equiv \tilde{x}\ \mathsf{closes}\ y; (P \mid \tilde{Q}) & \\
[\mathsf{Conn}/\mathsf{C}^{l1}_{\oplus\&}] & x \to \tilde{y}.\mathsf{inl}(P, Q) \mid \tilde{S} \equiv x \to \tilde{y}.\mathsf{inl}((P \mid \tilde{S}), Q) & (*) \\
[\mathsf{Conn}/\mathsf{C}^{l2}_{\oplus\&}] & x \to \tilde{y}.\mathsf{inl}(P, Q) \mid \tilde{S} \equiv x \to \tilde{y}.\mathsf{inl}((P \mid \tilde{S}), (Q \mid \tilde{S})) & (*) \\
[\mathsf{Conn}/\mathsf{C}^{r1}_{\oplus\&}] & x \to \tilde{y}.\mathsf{inr}(P, Q) \mid \tilde{S} \equiv x \to \tilde{y}.\mathsf{inr}(P, (Q \mid \tilde{S})) & (*) \\
[\mathsf{Conn}/\mathsf{C}^{r2}_{\oplus\&}] & x \to \tilde{y}.\mathsf{inr}(P, Q) \mid \tilde{S} \equiv x \to \tilde{y}.\mathsf{inr}((P \mid \tilde{S}), (Q \mid \tilde{S})) & (*) \\
[\mathsf{Conn}/\mathsf{C}!?] & x\ \mathsf{starts}\ \tilde{y}; P \mid \tilde{Q} \equiv x\ \mathsf{starts}\ \tilde{y}; (P \mid \tilde{Q}) & \\
[\mathsf{Conn}/\mathsf{C}_{!\mathsf{w}}] & x\ \mathsf{kills}\ \widetilde{y_i(Q_i)}; P \mid \tilde{Q} \equiv x\ \mathsf{kills}\ \widetilde{y_i(Q_i)}; (P \mid \tilde{Q}) & \\
[\mathsf{Conn}/\mathsf{C}_{!\mathsf{C}}] & x\ \mathsf{clones}\ \tilde{y}(x', \tilde{y}'); P \mid \tilde{Q} \equiv x\ \mathsf{clones}\ \tilde{y}(x', \tilde{y}'); (P \mid \tilde{Q}) & (x', \tilde{y}' \notin \mathsf{fn}\,\tilde{Q}) \\
[\mathsf{Conn}/\mathsf{C}_{\exists\forall}] & x[A] \to \tilde{y}(X); P \mid \tilde{Q} \equiv x[A] \to \tilde{y}(X); (P \mid \tilde{Q}) &
\end{array}
$$

Fig. 9. Selected commuting conversions, Conn fragment

$$
\begin{array}{lll}
[\mathsf{Scope}/\mathsf{Conn}] & (\boldsymbol{\nu}\tilde{x}^{\tilde{A}} : G)\,(P \mid \tilde{Q}) \equiv (\boldsymbol{\nu}\tilde{x}^{\tilde{A}} : G)\,P \mid \tilde{Q} & \\
[\mathsf{Scope}/\mathsf{Scope}] & (\boldsymbol{\nu}\tilde{x}^{\tilde{A}} : G)\,(\boldsymbol{\nu}\tilde{y}^{\tilde{B}} : H)\,P \equiv (\boldsymbol{\nu}\tilde{y}^{\tilde{B}} : H)\,(\boldsymbol{\nu}\tilde{x}^{\tilde{A}} : G)\,P & \\
[\mathsf{Scope}/\otimes_l] & (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,(\overline{x^y}(x'; y', \tilde{z}'); (P \mid Q)) \equiv \overline{x^y}(x'; y', \tilde{z}'); ((\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,P \mid Q) & (*) \\
[\mathsf{Scope}/\otimes_r] & (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,(\overline{x^y}(x'; y', \tilde{z}'); (P \mid Q)) \equiv \overline{x^y}(x'; y', \tilde{z}'); (P \mid (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,Q) & (*) \\
[\mathsf{Scope}/\parr] & (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,(y^{\tilde{x}}(\tilde{x}'; y'); P) \equiv y^{\tilde{x}}(\tilde{x}'; y'); (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,P & \\
[\mathsf{Scope}/\oplus_1] & (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,(x^{\tilde{y}}.\mathsf{inl}; P) \equiv x^{\tilde{y}}.\mathsf{inl}; (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,P & \\
[\mathsf{Scope}/\oplus_2] & (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,(x^{\tilde{y}}.\mathsf{inr}; Q) \equiv x^{\tilde{y}}.\mathsf{inr}; (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,Q & \\
[\mathsf{Scope}/\&] & (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,(y^x.\mathsf{case}(P, Q)) \equiv y^x.\mathsf{case}((\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,P, (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,Q) & \\
[\mathsf{Scope}/!] & (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,(\mathsf{srv}\ y^x; P) \equiv \mathsf{srv}\ y^x; (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,P & \\
[\mathsf{Scope}/?] & (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,(\mathsf{use}\ x^{\tilde{y}}; P) \equiv \mathsf{use}\ x^{\tilde{y}}; (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,P & \\
[\mathsf{Scope}/\mathsf{Weaken}] & (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,(\mathsf{kill}\ x^{\tilde{y}}; P) \equiv \mathsf{kill}\ x^{\tilde{y}}; (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,P & \\
[\mathsf{Scope}/\mathsf{Contract}] & (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,(\mathsf{clone}\ x^{\tilde{y}}(x', \tilde{y}'); P) \equiv \mathsf{clone}\ x^{\tilde{y}}(x', \tilde{y}'); (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,P & \\
[\mathsf{Scope}/\exists] & (\boldsymbol{\nu}\tilde{w}^{\tilde{B}} : G)\,(x^{\tilde{y}}[A]; P) \equiv x^{\tilde{y}}[A]; (\boldsymbol{\nu}\tilde{w}^{\tilde{B}} : G)\,P & \\
[\mathsf{Scope}/\forall] & (\boldsymbol{\nu}\tilde{w}^{\tilde{B}} : G)\,(y^x[X]; P) \equiv y^x[X]; (\boldsymbol{\nu}\tilde{w}^{\tilde{B}} : G)\,P & \\
[\mathsf{Scope}/\mathsf{C}_{\otimes\parr}] & (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,(\tilde{x}(\tilde{x}') \to y(y'); P) \equiv \tilde{x}(\tilde{x}') \to y(y'); (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,P & \\
[\mathsf{Scope}/\mathsf{C}_{\mathbf{1}\bot}] & (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,(\tilde{x}\ \mathsf{closes}\ y; P) \equiv \tilde{x}\ \mathsf{closes}\ y; (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,P & \\
[\mathsf{Scope}/\mathsf{C}^{l1}_{\oplus\&}] & (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,(x \to \tilde{y}.\mathsf{inl}(P, Q)) \equiv x \to \tilde{y}.\mathsf{inl}((\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,P, Q) & (*) \\
[\mathsf{Scope}/\mathsf{C}^{l2}_{\oplus\&}] & (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,(x \to \tilde{y}.\mathsf{inl}(P, Q)) \equiv x \to \tilde{y}.\mathsf{inl}((\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,P, (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,Q) & (*) \\
[\mathsf{Scope}/\mathsf{C}^{r1}_{\oplus\&}] & (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,(x \to \tilde{y}.\mathsf{inr}(P, Q)) \equiv x \to \tilde{y}.\mathsf{inr}(P, (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,Q) & (*) \\
[\mathsf{Scope}/\mathsf{C}^{r1}_{\oplus\&}] & (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,(x \to \tilde{y}.\mathsf{inr}(P, Q)) \equiv x \to \tilde{y}.\mathsf{inr}((\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,P, (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,Q) & (*) \\
[\mathsf{Scope}/\mathsf{C}!?] & (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,(x\ \mathsf{starts}\ \tilde{y}; P) \equiv x\ \mathsf{starts}\ \tilde{y}; (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,P & \\
[\mathsf{Scope}/\mathsf{C}_{!\mathsf{w}}] & (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,(x\ \mathsf{kills}\ \widetilde{y_i(Q_i)}; P) \equiv x\ \mathsf{kills}\ \widetilde{y_i(Q_i)}; (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,P & \\
[\mathsf{Scope}/\mathsf{C}_{!\mathsf{C}}] & (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,(x\ \mathsf{clones}\ \tilde{y}(x', \tilde{y}'); P) \equiv x\ \mathsf{clones}\ \tilde{y}(x', \tilde{y}'); (\boldsymbol{\nu}\tilde{w}^{\tilde{A}} : G)\,P & \\
[\mathsf{Scope}/\mathsf{C}_{\exists\forall}] & (\boldsymbol{\nu}\tilde{w}^{\tilde{B}} : G)\,(x[A] \to \tilde{y}(X); P) \equiv x[A] \to \tilde{y}(X); (\boldsymbol{\nu}\tilde{w}^{\tilde{B}} : G)\,P &
\end{array}
$$

Fig. 10. Commuting conversions, Scope fragment

muting conversions, is presented in Fig. 9 and 10 (refer to the Appendix for the complete set of equivalences). As we are interested only in commuting conver-

sions of typeable programs, there are certain cases where the correct equivalence can be found only by looking at the typing derivation. For instance, a choice between rules $\mathsf{Conn}/\mathsf{C}^{l_1}_{\oplus\&}$ and $\mathsf{Conn}/\mathsf{C}^{l_2}_{\oplus\&}$ follows from two possible origins of the endpoint getting connected via Conn. There are two possibilities of handling this ambiguity: either annotate parallel compositions introduced by a Conn rule with names of endpoints getting connected, allowing to perform a syntactic check on the subprograms, or assert – as we do here – that we define these conversions between typeable programs. The $(*)$ side condition in Fig. 9 and Fig. 10 therefore reads "provided both sides are typeable". Similar ambiguity can be noticed in the $\mathsf{Scope}/\otimes_l$ and $\mathsf{Scope}/\otimes_r$ cases, since in the presence of $\top$ type, we cannot count on all endpoint names being mentioned in the proof term. Again for a syntactic check to be possible, we would need to pollute the $x^y.\mathsf{case}()$ proof term with the whole context that is being dropped.

Besides the commuting conversions above, we also have the usual structural equivalence rules, making the parallel composition under restriction and the linking process symmetric. Furthermore, the order of restrictions can be swapped.

$$x^A \to y^B \equiv y^B \to x^A$$
$$(\boldsymbol{\nu}\tilde{w}, y, x, \tilde{z} : G)\, \tilde{P} \mid R \mid Q \mid \tilde{S} \equiv (\boldsymbol{\nu}\tilde{w}, x, y, \tilde{z} : G)\, \tilde{P} \mid Q \mid R \mid \tilde{S}$$
$$(\boldsymbol{\nu}z, \tilde{w} : H)\, (\boldsymbol{\nu}x, \tilde{y} : G)\, P \mid \tilde{R} \mid \tilde{Q} \equiv (\boldsymbol{\nu}x, \tilde{y} : G)\, (\boldsymbol{\nu}z, \tilde{w} : H)\, P \mid \tilde{Q} \mid \tilde{R}$$

4.3 Properties

Since structural congruence is typing-based, with occasional side conditions requiring type preservation ($*$ side condition), we can establish subject congruence.

Theorem 1 (Subject Congruence).
$\Sigma \Vdash P \mathbin{::} \Psi$ and $P \equiv Q$ implies that $\Sigma \Vdash Q \mathbin{::} \Psi$.

Furthermore, since the reductions in our system are based on proof transformations, they preserve typing. In the next result, $\xrightarrow{*}$ is the reflexive transitive closure of the reduction relation $\longrightarrow$, which may use commuting conversions as defined on Fig. 9 and Fig. 10.

Theorem 2 (Subject Reduction).
$\Sigma \Vdash P \mathbin{::} \Psi$ and $P \xrightarrow{*} Q$ implies $\Sigma \Vdash Q \mathbin{::} \Psi$.

Finally, by persistently applying commuting conversions in order to allow triggering reductions, we can eventually reach a proof term which is free of applications of Scope rule. Proof theoretically, this procedure can be viewed as MCC's equivalent of a cut elimination theorem: Scope elimination.

Theorem 3 (Termination/Deadlock-freedom).
$\Sigma \Vdash P \mathbin{::} \Psi$ implies that there exists Q which is restriction-free *and such that $P \xrightarrow{*} Q$ and $\Sigma \Vdash Q \mathbin{::} \Psi$.*

5 Projection and Extraction

As suggested in Fig. 1 and made precise by the semantics of MCC (Figs. 6 and 8), most interactions can be implemented in two ways: as a single choreography term, or as multiple process terms appearing in different behaviours composed in parallel. Choreography interactions can be projected to process implementations, and symmetrically process implementations can be extracted to choreographies, by applying the transformations defined in Fig. 11 to subterms of a program. Note that to extract a choreography we will sometimes have to rewrite processes in accordance with the commuting conversions from Figs. 9 and 10, to reach a shape expected by the extraction rules.

$$
\begin{aligned}
\left(\overline{x_i^y}(x_i'; y', \tilde{x}_{\setminus i}'); (P_i \mid Q_i)\right)_i \mid y^{\tilde{x}}(\tilde{x}'; y'); R &\rightleftharpoons \tilde{x}(\tilde{x}') \to y(y'); ((P_i \mid Q_i)_i \mid R) \\
(\mathsf{close}[x_i^y])_i \mid \mathsf{wait}[y^{\tilde{x}}]; P &\rightleftharpoons \tilde{x}\ \mathsf{closes}\ y; P \\
x^{\tilde{y}}.\mathsf{inl}; P \mid (y_i^x.\mathsf{case}(Q_i, R_i))_i &\rightleftharpoons x \to \tilde{y}.\mathsf{inl}(P \mid (Q_i)_i, (R_i)_i) \\
x^{\tilde{y}}.\mathsf{inr}; P \mid (y_i^x.\mathsf{case}(Q_i, R_i))_i &\rightleftharpoons x \to \tilde{y}.\mathsf{inr}((Q_i)_i, P \mid (R_i)_i) \\
\mathsf{use}\, x^{\tilde{y}}; P \mid (\mathsf{srv}\, {y_i}^x; Q_i)_i &\rightleftharpoons x\ \mathsf{starts}\ \tilde{y}; (P \mid (Q_i)_i) \\
\mathsf{kill}\, x^{\tilde{y}}; P \mid (\mathsf{srv}\, {y_i}^x; Q_i)_i &\rightleftharpoons x\ \mathsf{kills}\ \widetilde{y_i(Q_i)}; P \\
\mathsf{clone}\, x^{\tilde{y}}(x', \tilde{y}'); P \mid (\mathsf{srv}\, {y_i}^x; Q_i)_i &\rightleftharpoons \\
x\ \mathsf{clones}\ \tilde{y}(x', \tilde{y}'); \left(P \mid (\mathsf{srv}\, {y_i}^x; Q_i)_i \mid (\mathsf{srv}\, {y_i'}^{x'}; Q_i\{x'/x, \tilde{y}'/\tilde{y}\})_i\right) & \\
x^{\tilde{y}}[A]; P \mid ({y_i}^x[X]; Q_i)_i &\rightleftharpoons x[A] \to \tilde{y}(X); (P \mid (Q_i)_i)
\end{aligned}
$$

Fig. 11. Extraction (left-to-right) and projection (right-to-left)

These transformation rules are derived by applying the same proof transformation techniques described in [3, 5, 14] (Conn elimination, which is an adaptation of standard cut-elimination to our setting). We omit the concrete derivations, since they are straightforward and obtained by previously established methods (cf. Appendix). As a consequence of Conn elimination, every program can be rewritten into a (non-unique) pure process form containing only process terms by applying the rules in Fig. 11 from right to left until no longer possible. Conversely, every program can be rewritten into a maximal choreographic form by applying the same rules from left to right until no longer possible.

In order to express that this transformation does not change the semantics of the program, we need to annotate the reductions in Figs. 6 and 8. As label for a transition, we take the set of endpoints in the subterm performing the reduction. For example, the reductions $(\boldsymbol{\nu}\tilde{x}, y : \tilde{x} \to y)\,(\mathsf{close}[x_i^y])_i \mid \mathsf{wait}[y^{\tilde{x}}]; P \longrightarrow P$ and $(\boldsymbol{\nu}\tilde{x}, y : \tilde{x} \to y)\,(\tilde{x}\ \mathsf{closes}\ y; P) \longrightarrow P$ are both annotated by $\tilde{x}, y$. In particular, reductions affecting the subterms on the left- and right-hand side of the transformations in Fig. 11 have the same label.

Definition 1. *The relation $\triangle$ between MCC programs is defined as follows: $P \triangle Q$ iff P can be transformed into Q by a finite sequence of applications of the transformations in Fig. 11 (in any direction) to its subterms.*

Theorem 4. *If $P \triangle Q$, then P and Q are trace equivalent.*

6 Related Work and Discussion

The principle of choreographies as cut reductions was introduced in [5]. As discussed in § 1, they cannot capture services, polymorphism, and multiparty sessions. Another difference is that it is based on intuitionistic linear logic, whereas ours on classical linear logic—in particular, on Classical Processes [18].

Switching to classical linear logic is not a mere change of appearance. It is what allows us to reuse the logical understanding of multiparty sessions in linear logic as *coherence proofs*, introduced in [6] and later extended to polymorphism in [3]. These works did not consider choreographic programs, and thus do not offer a global view on how different sessions are composed, as we do in this paper.

Extracting choreographies from compositions of process code is well-known to be a hard problem. In [12], choreographies that abstract from the exchanged values and computation are extracted from communicating finite-state machines. The authors of [7] present an efficient algorithm for extracting concrete choreographic programs with asynchronous messaging. These works do not consider the composition of multiple sessions, multiparty sessions, services, and polymorphism, as in MCC. However, they can both deal with infinite behaviour (through loops or recursion), which we do not address. An interesting direction for this feature would be to integrate structural recursion for classical linear logic [13].

Our approach can be seen as a principled reconstruction of previous works on choreographic programming. The first work that typed choreographies using multiparty session types is [4]. The idea of integrating choreographies with processes using multiparty session types is from [16]. None of these consider extraction or polymorphism.

Since the seminal paper that identified a correspondence between session-typed processes and linear logic [2], there have been a number of developments in this research line. Some of these are particularly interesting for MCC; we list a few examples. By importing the functional stratification from [17], we could obtain a monadic integration of choreographies with functions. The calculus of classical higher-order processes [15] could be of inspiration for adding code mobility to MCC, by adding higher-order types. The non-deterministic linear types from [1] might offer insight on how we could extract choreographies from processes with non-deterministic behaviour. Types for manifest sharing may lead us to global specifications of sharing in choreographies. And the asynchronous interpretation of cut reductions in [9] might give us an asynchronous implementation of choreographies in MCC. We leave an exploration of these extensions to future work. Hopefully, the shared foundations of linear logic will make it possible to build on these pre-existing technical developments following the same idea of choreographies as cut reductions.

References

1. L. Caires and J. A. Pérez. Linearity, control effects, and behavioral types. In *ESOP*, volume 10201 of *Lecture Notes in Computer Science*, pages 229–259. Springer, 2017.
2. L. Caires and F. Pfenning. Session types as intuitionistic linear propositions. In *CONCUR*, pages 222–236, 2010.
3. M. Carbone, S. Lindley, F. Montesi, C. Schürmann, and P. Wadler. Coherence generalises duality: A logical explanation of multiparty session types. In *CONCUR*, volume 59 of *LIPIcs*, pages 33:1–33:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.
4. M. Carbone and F. Montesi. Deadlock-freedom-by-design: multiparty asynchronous global programming. In *POPL*, pages 263–274, 2013.
5. M. Carbone, F. Montesi, and C. Schürmann. Choreographies, logically. *Distributed Computing*, 31(1):51–67, 2018.
6. M. Carbone, F. Montesi, C. Schürmann, and N. Yoshida. Multiparty session types as coherence proofs. *Acta Inf.*, 54(3):243–269, 2017. Also: CONCUR 2015.
7. L. Cruz-Filipe, K. S. Larsen, and F. Montesi. The paths to choreography extraction. In *FoSSaCS*, volume 10203 of *Lecture Notes in Computer Science*, pages 424–440, 2017.
8. M. Dalla Preda, M. Gabbrielli, S. Giallorenzo, I. Lanese, and J. Mauro. Dynamic choreographies: Theory and implementation. *Logical Methods in Computer Science*, 13(2), 2017.
9. H. DeYoung, L. Caires, F. Pfenning, and B. Toninho. Cut reduction in linear logic as asynchronous session-typed communication. In *CSL*, volume 16 of *LIPIcs*, pages 228–242. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.
10. K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. *J. ACM*, 63(1):9:1–9:67, 2016.
11. JBoss Community and Red Hat. Testable Architecture. http://www.jboss.org/savara/.
12. J. Lange, E. Tuosto, and N. Yoshida. From communicating machines to graphical choreographies. In *POPL*, pages 221–232. ACM, 2015.
13. S. Lindley and J. G. Morris. Talking bananas: structural recursion for session types. In *ICFP*, pages 434–447. ACM, 2016.
14. F. Montesi. *Choreographic Programming*. Ph.D. thesis, IT University of Copenhagen, 2013. http://www.itu.dk/people/fabr/papers/phd/thesis.pdf.
15. F. Montesi. Classical higher-order processes - (short paper). In *FORTE*, volume 10321 of *Lecture Notes in Computer Science*, pages 171–178. Springer, 2017.
16. F. Montesi and N. Yoshida. Compositional choreographies. In *CONCUR*, pages 425–439, 2013.
17. B. Toninho, L. Caires, and F. Pfenning. Higher-order processes, functions, and sessions: A monadic integration. In *ESOP*, pages 350–369, 2013.
18. P. Wadler. Propositions as sessions. *J. Funct. Program.*, 24(2-3):384–418, 2014.

A Commuting Conversions

[Conn/Conn]	$(\tilde{P} \mid Q) \mid \tilde{S} \equiv \tilde{P} \mid (Q \mid \tilde{S})$	
[Conn/$\otimes_l$]	$(\overline{x^{\tilde{y}}}(x'; y', \tilde{z}'); (P \mid Q)) \mid \tilde{S} \equiv \overline{x^{\tilde{y}}}(x'; y', \tilde{z}'); ((P \mid \tilde{S}) \mid Q)$	
[Conn/$\otimes_r$]	$(\overline{x^{\tilde{y}}}(x'; y', \tilde{z}'); (P \mid Q)) \mid \tilde{S} \equiv \overline{x^{\tilde{y}}}(x'; y', \tilde{z}'); (P \mid (Q \mid \tilde{S}))$	
[Conn/⅋]	$y^{\tilde{x}}(\tilde{x}'; y'); P \mid \tilde{Q} \equiv y^{\tilde{x}}(\tilde{x}'; y'); (P \mid \tilde{Q})$	
[Conn/$\bot$]	$\mathsf{wait}[y^{\tilde{x}}]; P \mid \tilde{Q} \equiv \mathsf{wait}[y^{\tilde{x}}]; (P \mid \tilde{Q})$	
[Conn/$\oplus_1$]	$x^{\tilde{y}}.\mathsf{inl}; P \mid \tilde{Q} \equiv x^{\tilde{y}}.\mathsf{inl}; (P \mid \tilde{Q})$	
[Conn/$\oplus_2$]	$x^{\tilde{y}}.\mathsf{inr}; Q \mid \tilde{Q} \equiv x^{\tilde{y}}.\mathsf{inr}; (P \mid \tilde{Q})$	
[Conn/&]	$y^x.\mathsf{case}(P, Q) \mid \tilde{S} \equiv y^x.\mathsf{case}((P \mid \tilde{S}), (Q \mid \tilde{S}))$	
[Conn/!]	$\mathsf{srv}\, y^x; P \mid \tilde{Q} \equiv \mathsf{srv}\, y^x; (P \mid \tilde{Q})$	
[Conn/?]	$\mathsf{use}\, x^{\tilde{y}}; P \mid \tilde{Q} \equiv \mathsf{use}\, x^{\tilde{y}}; (P \mid \tilde{Q})$	
[Conn/Weaken]	$\mathsf{kill}\, x^{\tilde{y}}; P \mid \tilde{Q} \equiv \mathsf{kill}\, x^{\tilde{y}}; (P \mid \tilde{Q})$	
[Conn/Contract]	$\mathsf{clone}\, x^{\tilde{y}}(x', \tilde{y}'); P \mid \tilde{Q} \equiv \mathsf{clone}\, x^{\tilde{y}}(x', \tilde{y}'); (P \mid \tilde{Q})$	
[Conn/$\exists$]	$x^{\tilde{y}}[A]; P \mid \tilde{Q} \equiv x^{\tilde{y}}[A]; (P \mid \tilde{Q})$	
[Conn/$\forall$]	$y^x[X]; P \mid \tilde{Q} \equiv y^x[X]; (P \mid \tilde{Q})$	
[Conn/$\mathsf{C}_{\otimes⅋}$]	$\tilde{x}(\tilde{x}') \to y(y'); P \mid \tilde{Q} \equiv \tilde{x}(\tilde{x}') \to y(y'); (P \mid \tilde{Q})$	$(\tilde{x}', y' \notin \mathsf{fn}\,\tilde{Q})$
[Conn/$\mathsf{C}_{\mathbf{1}\bot}$]	$\tilde{x}\ \mathsf{closes}\ y; P \mid \tilde{Q} \equiv \tilde{x}\ \mathsf{closes}\ y; (P \mid \tilde{Q})$	
[Conn/$\mathsf{C}^{l_1}_{\oplus\&}$]	$x \to \tilde{y}.\mathsf{inl}(P, Q) \mid \tilde{S} \equiv x \to \tilde{y}.\mathsf{inl}((P \mid \tilde{S}), Q)$	$(*)$
[Conn/$\mathsf{C}^{l_2}_{\oplus\&}$]	$x \to \tilde{y}.\mathsf{inl}(P, Q) \mid \tilde{S} \equiv x \to \tilde{y}.\mathsf{inl}((P \mid \tilde{S}), (Q \mid \tilde{S}))$	$(*)$
[Conn/$\mathsf{C}^{r_1}_{\oplus\&}$]	$x \to \tilde{y}.\mathsf{inr}(P, Q) \mid \tilde{S} \equiv x \to \tilde{y}.\mathsf{inr}(P, (Q \mid \tilde{S}))$	$(*)$
[Conn/$\mathsf{C}^{r_2}_{\oplus\&}$]	$x \to \tilde{y}.\mathsf{inr}(P, Q) \mid \tilde{S} \equiv x \to \tilde{y}.\mathsf{inr}((P \mid \tilde{S}), (Q \mid \tilde{S}))$	$(*)$
[Conn/C!?]	$x\ \mathsf{starts}\ \tilde{y}; P \mid \tilde{Q} \equiv x\ \mathsf{starts}\ \tilde{y}; (P \mid \tilde{Q})$	
[Conn/$\mathsf{C}_{!\mathsf{w}}$]	$x\ \mathsf{kills}\ \widetilde{y_i(Q_i)}; P \mid \tilde{Q} \equiv x\ \mathsf{kills}\ \widetilde{y_i(Q_i)}; (P \mid \tilde{Q})$	
[Conn/$\mathsf{C}_{!\mathsf{C}}$]	$x\ \mathsf{clones}\ \tilde{y}(x', \tilde{y}'); P \mid \tilde{Q} \equiv x\ \mathsf{clones}\ \tilde{y}(x', \tilde{y}'); (P \mid \tilde{Q})$	$(x', \tilde{y}' \notin \mathsf{fn}\,\tilde{Q})$
[Conn/$\mathsf{C}_{\exists\forall}$]	$x[A] \to \tilde{y}(X); P \mid \tilde{Q} \equiv x[A] \to \tilde{y}(X); (P \mid \tilde{Q})$	

Fig. 12. Commuting conversions, Conn fragment

$$\dfrac{\dfrac{\dfrac{\mathcal{D}_{i1}}{\left(\Sigma_{i1} \;\Vdash\; P_i \;\wr\; \Psi_{i1} \mid \vdash \Gamma_{i1}, x'_i : A_i\right)_i} \quad \dfrac{\mathcal{D}_{i2}}{\left(\Sigma_{i2} \;\Vdash\; Q_i \;\wr\; \Psi_{i2} \mid \vdash \Gamma_{i2}, x_i : B_i\right)_i}}{\left((\Sigma_{i1}, \Sigma_{i2})_i \;\Vdash\; \left(\overline{x_i^{\,y}}(x'_i; y', \tilde{x}'_{\setminus i}); (P_i \mid Q_i)\right)_i \;\wr\; (\Psi_{i1} \mid \Psi_{i2})_i \mid (\vdash \Gamma_{i1}, \Gamma_{i2}, x_i : A_i \otimes^y B_i)_i\right)} \otimes \quad \dfrac{\dfrac{\mathcal{D}}{\Sigma_y \;\Vdash\; R \;\wr\; \vdash \Gamma_y, y' : C, y : D}}{\Sigma_y \;\Vdash\; y^{\tilde{x}}(\tilde{x}'; y'); R \;\wr\; \Psi_y \mid \vdash \Gamma_y, y : C \parr^{\tilde{x}} D} \parr}{(\Sigma_{i1}, \Sigma_{i2})_i, \Sigma_y, (\tilde{x}, y) : \tilde{x} \to y(G); H \;\Vdash\; \left(\overline{x_i^{\,y}}(x'_i; y', \tilde{x}'_{\setminus i}); (P_i \mid Q_i)\right)_i \mid y^{\tilde{x}}(\tilde{x}'; y'); R \;\wr\; (\Psi_{i1} \mid \Psi_{i2})_i \mid \Psi_y \mid (\vdash \Gamma_{i1}, \Gamma_{i2}, x_i : A_i \otimes^y B_i)_i \mid \vdash \Gamma_y, y : C \parr^{\tilde{x}} D} \;\mathsf{Conn}$$

extracts to / projects from

$$\dfrac{\dfrac{\dfrac{\mathcal{D}_{i1}}{\left(\Sigma_{i1} \;\Vdash\; P_i \;\wr\; \Psi_{i1} \mid \vdash \Gamma_{i1}, x'_i : A_i\right)_i} \quad \dfrac{\dfrac{\mathcal{D}_{i2}}{\left(\Sigma_{i2} \;\Vdash\; Q_i \;\wr\; \Psi_{i2} \mid \vdash \Gamma_{i2}, x_i : B_i\right)_i} \quad \dfrac{\mathcal{D}}{\Sigma_y \;\Vdash\; R \;\wr\; \vdash \Gamma_y, y' : C, y : D}}{(\Sigma_{i2})_i, \Sigma_y, (\tilde{x}', y') : G\{\tilde{x}'/\tilde{x}, y'/y\} \;\Vdash\; (Q_i)_i \mid R \;\wr\; (\Psi_{i2})_i \mid (\vdash \Gamma_{i2}, x_i : B_i)_i \mid \vdash \Gamma_y, y' : C, y : D} \;\mathsf{Conn}}{(\Sigma_{i1}, \Sigma_{i2})_i, \Sigma_y, (\tilde{x}, y, \tilde{u}) : H, (\tilde{x}', y') : G\{\tilde{x}'/\tilde{x}, y'/y\} \;\Vdash\; ((P_i \mid Q_i)_i \mid R) \;\wr\; (\Psi_{i1} \mid \Psi_{i2})_i \mid \Psi_y \mid \left(\vdash \Gamma_{i1}, x'_i : A_i\right)_i \mid (\vdash \Gamma_{i2}, x_i : B_i)_i \mid \vdash \Gamma_y, y' : C, y : D} \;\mathsf{Conn}}{(\Sigma_{i1}, \Sigma_{i2})_i, \Sigma_y, (\tilde{x}, y) : \tilde{x} \to y(G); H \;\Vdash\; \tilde{x}(\tilde{x}') \to y(y'); ((P_i \mid Q_i)_i \mid R) \;\wr\; (\Psi_{i1} \mid \Psi_{i2})_i \mid \Psi_y \mid (\vdash \Gamma_{i1}, \Gamma_{i2}, x_i : A_i \otimes^y B_i)_i \mid \vdash \Gamma_y, y : C \parr^{\tilde{x}} D} \;\mathsf{C}_{\otimes\parr}$$

$$\dfrac{\dfrac{\dfrac{\mathcal{D}}{\Sigma_x \;\Vdash\; P \;\wr\; \Psi_x \mid \vdash \Gamma_x, x : A}}{\Sigma_x \;\Vdash\; \mathsf{use}\, x^{\tilde{y}}; P \;\wr\; \Psi_x \mid \vdash \Gamma_x, x : ?^{\tilde{y}} A} \;? \quad \dfrac{\dfrac{\mathcal{D}_i}{\left(\Sigma_i \;\Vdash\; Q_i \;\wr\; ?\Psi_i \mid \vdash ?\Gamma_i,\; y_i : B_i\right)_i}}{\left(\Sigma_i \;\Vdash\; \mathsf{srv}\, y_i{}^x; Q_i \;\wr\; ?\Psi_i \mid \vdash ?\Gamma_i,\; y_i : !^x B_i\right)} \;!}{\Sigma_x, (\Sigma_i)_i, (x, \tilde{y}) : !x \to \tilde{y}(G) \;\Vdash\; \mathsf{use}\, x^{\tilde{y}}; P \mid (\mathsf{srv}\, y_i{}^x; Q_i)_i \;\wr\; \Psi_x \mid (?\Psi_i)_i \mid \vdash \Gamma_x, x : ?^{\tilde{y}} A \mid \left(\vdash ?\Gamma_i,\; y_i : !^x B_i\right)_i} \;\mathsf{Conn}$$

extracts to / projects from

$$\dfrac{\dfrac{\dfrac{\mathcal{D}}{\Sigma_x \;\Vdash\; P \;\wr\; \Psi_x \mid \vdash \Gamma_x, x : A} \quad \dfrac{\mathcal{D}_i}{\left(\Sigma_i \;\Vdash\; Q_i \;\wr\; ?\Psi_i \mid \vdash ?\Gamma_i,\; y_i : B_i\right)_i}}{\Sigma_x, (\Sigma_i)_i, (x, \tilde{y}) : G \;\Vdash\; P \mid (Q_i)_i \;\wr\; \Psi_x \mid (?\Psi_i)_i \mid \vdash \Gamma_x, x : A \mid \left(\vdash ?\Gamma_i,\; y_i : B_i\right)_i} \;\mathsf{Conn}}{\Sigma_x, (\Sigma_i)_i, (x, \tilde{y}) : !x \to \tilde{y}(G) \;\Vdash\; x\;\mathsf{starts}\;\tilde{y}; (P \mid (Q_i)_i) \;\wr\; \Psi_x \mid (?\Psi_i)_i \mid \vdash \Gamma_x, x : ?^{\tilde{y}} A \mid \left(\vdash ?\Gamma_i,\; y_i : !^x B_i\right)_i} \;\mathsf{C!?}$$

Fig. 13. Example derivations of projection/extraction (omitting side conditions on coherence).