Noname manuscript No.
(will be inserted by the editor)

Choreographies, Logically

Marco Carbone · Fabrizio Montesi · Carsten Schürmann

Received: – / Accepted: –

Abstract In Choreographic Programming, a distributed system is programmed by giving a *choreography*, a global description of its interactions, instead of separately specifying the behaviour of each of its processes. Process implementations in terms of a distributed language can then be automatically projected from a choreography.

We present Linear Compositional Choreographies (LCC), a proof theory for reasoning about programs that modularly combine choreographies with processes. Using LCC, we logically reconstruct a semantics and a projection procedure for programs. For the first time, we also obtain a procedure for extracting choreographies from process terms.

Keywords Choreographies · Curry-Howard Isomorphism · Linear Logic · Programming Languages

1 Introduction

Choreographic Programming [17] is a programming paradigm for distributed systems inspired by the "Alice and Bob" notation, where programs, called *choreographies* [25,1], are global descriptions of how endpoint processes interact (exchange messages) during execution. The typical set of programs defining the actions performed by each process is then generated by means of *endpoint projection* (EPP) [21,14,9,4,10,18].

The key aspect of choreography languages is that process interactions are treated linearly, i.e., they are executed exactly once. Previous work [9,10,18] developed correct notions of EPP by using typing disciplines based on session types [13], linear types for communications inspired by linear logic [12]. Despite the deep connections between choreographies and linearity, the following question remains unanswered:

> *Is there a formal connection between choreographies and linear logic?*

Finding such a connection would contribute to a more precise understanding of choreographies, and possibly lead to answering open questions about them.

A good starting point for answering our question is a recent line of work on a Curry-Howard correspondence between the internal π-calculus [22] and linear logic [7,26]. In particular, proofs in Intuitionistic Linear Logic (ILL) correspond to π-calculus terms (proofs-as-programs) and ILL propositions correspond to session types [7]. An ILL judgement describes the interface of a process, for example:

$$P \triangleright x : A, y : B \vdash z : C$$

Above, P is a process and the judgement $P \triangleright x : A, y : B \vdash z : C$ reads as follows: process P needs to be composed with other processes that provide the behaviours (represented as types) A on channel x and B on channel y, in order to provide behaviour C on channel z. Note that process P may contain nested processes that

M. Carbone
IT University of Copenhagen
Rued Langgaards Vej 7, 2300 Copenhagen (Denmark)
Tel.: +45-72185067
E-mail: carbonem@itu.dk

F. Montesi (✉)
University of Southern Denmark
Campusvej 55, 5230 Odense (Denmark)
Tel.: +45-65507171
E-mail: fmontesi@imada.sdu.dk

C. Schürmann
IT University of Copenhagen
Rued Langgaards Vej 7, 2300 Copenhagen (Denmark)
Tel.: +45-72185282
E-mail: carsten@itu.dk

engage in internal communications that are not made visible at the level of session types. In contrast, choreographies are descriptions of the internal interactions among the processes inside a system, and therefore type systems for choreographies focus on checking such internal interactions [9,10]. It is thus unclear how the linear typing of ILL can be related to choreographies.

In this paper, we present Linear Compositional Choreographies (LCC), a proof theory inspired by linear logic that seamlessly integrates the typing of internal communications with that of interfaces for external composition, using session types. The programs typed in LCC modularly combine choreographies with processes in the internal π-calculus. The key aspect of LCC is that it extends ILL to capture interactions among internal processes in a system. Thanks to LCC, not only do we obtain a logical understanding of choreographic programming, but we also provide the foundations for tackling the open problem of extracting a choreography from a system of processes.

This article is an extended version of the conference paper that appeared in the proceedings of the 25th International Conference on Concurrency Theory (CONCUR 2014).

1.1 Main Contributions

We summarise our main contributions.

Linear Compositional Choreographies (LCC). We present LCC, a generalisation of ILL where judgements can also describe the internal interactions of a system (§ 3). LCC proofs are equipped with proof terms, called LCC programs, following the standard Curry-Howard interpretation of proofs-as-programs. LCC programs are in a language where choreographies and processes are modularly combined by following protocols given in the type language of LCC (à la session types [13]).

Logically-derived semantics. We derive a semantics for LCC programs from our proof theory (§ 4): (i) some rule applications in LCC proofs can be permuted, defining equivalences on LCC programs (§ 4.1); (ii) some proofs can be safely reduced to smaller proofs, which corresponds to executing communications (§ 4.2). By following our semantics, we prove that all internal communications in a system can be reduced (proof normalisation), i.e., LCC programs are deadlock-free by construction (§ 4.3).

Choreography Extraction and Endpoint Projection. LCC consists of two fragments: the *action fragment*, which manipulates the external interfaces of processes, and the *interaction fragment*, which handles internal communications. We derive automatic transformations from proofs in either fragment to proofs in the other, yielding procedures of endpoint projection and choreography extraction (§ 5) that preserve the semantics of LCC programs. To our knowledge, this is the first work addressing extraction for a fragment of the π-calculus, providing the foundations for a new development methodology where programmers can compose choreographies with existing process code (e.g., software libraries) and then obtain a choreography that describes the overall behaviour of the entire composition.

2 From ILL to LCC

We informally introduce processes and choreographies, and revisit the Curry-Howard correspondence between the internal π-calculus and ILL [7]. Building on ILL, we introduce the intuition behind the proof theory of LCC.

Processes and Choreographies. Consider the following processes:

$$\begin{array}{lcl} P_{\mathsf{client}} & = & \overline{x}(tea);\ x(tr);\ \overline{tr}(p) \\ P_{\mathsf{server}} & = & x(tea);\ \overline{x}(tr);\ tr(p);\ \overline{b}(m) \\ P_{\mathsf{bank}} & = & b(m) \end{array} \tag{1}$$

The three processes above, given as terms in the language of the internal π-calculus [22], denote a system composed by three endpoints: client, server, and bank. The parallel execution of these endpoints is such that: client sends to server a request for tea on a channel x; then, server replies to client on the same channel x with a new channel tr (for transaction); client uses tr for sending to server the payment p; after receiving the payment, server deposits some money m by sending it over channel b to bank.

Programming with processes is error-prone, since they do not give an explicit description of how endpoints interact [17]. By contrast, choreographies provide a clear specification of how messages flow during execution [25]. For example, consider the following choreography:

$$\begin{array}{l} 1.\ \mathsf{client} \to \mathsf{server} : x(tea);\ \mathsf{server} \to \mathsf{client} : x(tr); \\ 2.\ \mathsf{client} \to \mathsf{server} : tr(p);\ \mathsf{server} \to \mathsf{bank} : b(m) \end{array} \tag{2}$$

The choreography in (2) defines the communications that occur in (1). We read $\mathsf{client} \to \mathsf{server} : x(tea)$ as "process client sends tea to process server through channel x".

ILL and the π-calculus. The processes in (1) can be typed by ILL, using propositions as session types that describe the usage of channels. For example, channel x in P_{client} has type $\mathbf{string} \otimes (\mathbf{string} \multimap \mathsf{end}) \multimap \mathsf{end}$, meaning: send a string; then, receive a channel of type $\mathbf{string} \multimap \mathsf{end}$ and, finally, stop (end). Concretely, in process P_{client}, the channel of type $\mathbf{string} \multimap \mathsf{end}$ received through x is channel tr. The type of tr says that the process sending tr, i.e., P_{server}, will use it to receive a $\mathbf{string}$; therefore, process P_{client} must implement the dual operation of that implemented by P_{server}, i.e., the output $\overline{tr}(p)$. Similarly, channel b has type $\mathbf{int} \otimes \mathsf{end}$ in P_{server}. We can formalise this intuition with the following three ILL judgements, where $A = \mathbf{string} \otimes (\mathbf{string} \multimap \mathsf{end}) \multimap \mathsf{end}$ and $B = \mathbf{int} \otimes \mathsf{end}$:

$$P_{\mathsf{client}} \triangleright \cdot \vdash x : A$$
$$P_{\mathsf{server}} \triangleright x : A \vdash b : B$$
$$P_{\mathsf{bank}} \triangleright b : B \vdash z : \mathsf{end}$$

Recall that the judgement $P_{\mathsf{server}} \triangleright x : A \vdash b : B$ reads as "given a context that implements channel x with type A, process P_{server} implements channel b with type B". Given these judgements, we can compose the processes P_{client}, P_{server}, and P_{bank} using channels x and b as:

$$(\boldsymbol{\nu} x)\,(\; P_{\mathsf{client}} \;\;|_x\;\; (\boldsymbol{\nu} b)\,(\; P_{\mathsf{server}} \;\;|_b\;\; P_{\mathsf{bank}} \;)\;) \tag{3}$$

The compositions in (3) can be typed using the Cut rule of ILL:

$$\frac{P \triangleright \Delta_1 \vdash x : A \qquad Q \triangleright \Delta_2, x : A \vdash y : B}{(\boldsymbol{\nu} x)\,(P \mid Q) \triangleright \Delta_1, \Delta_2 \vdash y : B}\ \mathsf{Cut} \tag{4}$$

Above, Δ_1 and Δ_2 are sets of typing assignments $(z : D)$. We interpret rule Cut as "If a process provides A on channel x, and another requires A on channel x to provide B on channel y, their parallel execution provides B on channel y".

Proofs in ILL correspond to process terms in the internal π-calculus [7], and applications of rule Cut can always be eliminated by a proof normalisation procedure known as *cut elimination*. This procedure provides a model of computation for processes. We illustrate a *cut reduction*, a step of cut elimination, in the following (we omit process terms for readability):

$$\frac{\dfrac{C_1 \vdash A \quad C_2 \vdash B}{C_1, C_2 \vdash A \otimes B}\otimes\mathsf{R} \qquad \dfrac{A, B \vdash D}{A \otimes B \vdash D}\otimes\mathsf{L}}{C_1, C_2 \vdash D}\ \mathsf{Cut} \qquad \Longrightarrow$$

$$\frac{C_1 \vdash A \qquad \dfrac{C_2 \vdash B \quad A, B \vdash D}{C_2, A \vdash D}\ \mathsf{Cut}}{C_1, C_2 \vdash D}\ \mathsf{Cut}$$

The proof on the left-hand side applies a cut to two proofs, one providing $A \otimes B$, and the other providing D when provided with $A \otimes B$. The cut-reduction above ($\Longrightarrow$) shows how this proof can be simplified to a proof where the cut on $A \otimes B$ is reduced to two cuts on the smaller formulas A and B. A cut-reduction corresponds to executing a communication between two processes, one outputting on a channel of type $A \otimes B$, and another inputting from the same channel [7]. Executing the communication yields a new system corresponding to the proof on the right-hand side. Cut-free proofs correspond to systems that have successfully completed all their internal communications.

Towards LCC. Cut reductions in ILL model the interactions between the internal processes in a system, which is exactly what choreographies describe syntactically. Therefore, in order to capture choreographies, we wish our proof theory to reason about transformations such as the cut reduction above.

ILL judgements give us no information on the applications of rule Cut in a proof. In contrast, standard type systems for choreographies [9,10,18] have different judgements: instead of interfaces for later composition, they contain information about internal processes and their interactions. Following this observation, we make two important additions to ILL judgements. First, we extend them to describe multiple processes by using *hypersequents*, i.e., collections of multiple ILL sequents [2]. Second, we represent the *connections* between sequents in a hypersequent, since two processes need to share a common connection for interacting. The following is an LCC judgement:

$$P \triangleright \Delta_1 \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash y : B$$

Above, we composed two ILL sequents with the operator $|$, which captures the parallel composition of processes. The two sequents are *connected* through channel x, denoted by the marking $\bullet$. We will use hypersequents and marking let us reason about interactions by handling both ends of a connection.

LCC judgements can express cut elimination as a proof. For example,

$$Q \triangleright z_1 : C_1, z_2 : C_2 \vdash x : \bullet A \otimes B \mid x : \bullet A \otimes B \vdash w : D$$

represents the left-hand side of the cut reduction seen previously, where a process requires C_1 and C_2 to perform an interaction of type $A \otimes B$ with another process that can then provide D. Importantly, the connection of type $A \otimes B$ between the two sequents cannot be composed with external systems, because it is used for internal interactions. Using our judgements, we can capture cut reductions:

$$Q' \triangleright z_1 : C_1 \vdash y : \bullet A \mid z_2 : C_2 \vdash x : \bullet B \mid y : \bullet A, x : \bullet B \vdash w : D$$

The new judgement describes a system that still requires C_1 and C_2 in order to provide D, but now with three processes: one providing A from C_1, one providing B from C_2 and, finally, one using A and B for providing D. Also, the first two sequents are connected to the third one. This corresponds to the right-hand side of the cut reduction that we have seen previously, where process Q reduces to process Q'.

We can now express the different internal states of a system before and after a cut reduction, by the structure of its connections in our judgements. This is the intuition behind the new rules in our proof theory for typing choreographies, which we present in § 3.

3 Linear Compositional Choreographies

We present Linear Compositional Choreographies (LCC), a proof theory for typing programs that can modularly combine choreographies and processes. We start by introducing LCC types and typing contexts; then, we define the syntax of LCC programs, which consists of process and choreography terms; and, finally, we give the rules of our proof theory.

3.1 Types and Hypersequents

Types. LCC propositions, or types, are defined as:

(Propositions)
$$A, B ::= \mathbf{1} \mid A \otimes B \mid A \multimap B \mid A \oplus B \mid A \& B$$

LCC propositions are the same as in ILL: $\otimes$ and $\multimap$ are the multiplicative connectives, while $\oplus$ and $\&$ are additives. The type $\mathbf{1}$ is the atomic proposition. A type $A \otimes B$ is interpreted as "output a channel of type A and then behave as specified by type B". On the other hand, $A \multimap B$, the linear implication, reads "receive a channel of type A and then continue as B". Proposition $A \oplus B$ selects a branch of type A or B, while $A \& B$ offers the choice of A or B.

Hypersequents. We introduce hypersequents, i.e., collections of ILL sequents, that we will use to type the behaviour of programs on channels. Their syntax is given below:

(Hypersequents)	Ψ	$::=$	$\Delta \vdash T \mid \Psi \mid \Psi$
(Contexts)	Δ, Θ	$::=$	$\cdot \mid \Delta, T$
(Element)	T	$::=$	$x : A \mid x : \bullet A$

In the syntax of hypersequents, an *element* assigns a type to a channel. Elements may be marked by $\bullet$, denoting that the channel is shared in a connection, as anticipated in § 2. *Contexts*, as in ILL, are sets of elements. A *hypersequent* is then a collection of some ILL sequents: $\Delta_1 \vdash T_1 \mid \ldots \mid \Delta_n \vdash T_n$.

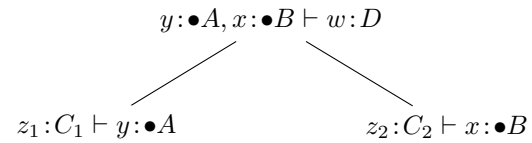
We consider contexts Δ and hypersequents Ψ equivalent up to associativity and commutativity. Given a sequent $\Delta \vdash T$, we call Δ its hypotheses and T its conclusion.

Following the ideas that we outlined in § 2, we intend to use hypersequents in order to represent the structure of connections among participants in a system. We assume that a channel name can appear at most once in any hypersequent, *unless* it is marked with $\bullet$. On the other hand, we assume that bulleted variables appear exactly twice in a hypersequent, once as a hypothesis and once as a conclusion of two respective sequents which we say are then "connected". In the proof theory that we will present, a provable hypersequent will always have exactly one sequent with a non-bulleted conclusion, which we call the conclusion of the hypersequent. Similarly, we call non-bulleted hypotheses the hypotheses of the hypersequent.

Graphically, we can think of a provable hypersequent as a tree of sequents, where the root is the only sequent with a non-bulleted conclusion. Then, a sequent has a child for each bulleted variable in its hypotheses, which is another sequent that has the same bulleted variable as conclusion. For example, consider the hypersequent from § 2:

$$z_1 : C_1 \vdash y : \bullet A \mid z_2 : C_2 \vdash x : \bullet B \mid y : \bullet A, x : \bullet B \vdash w : D$$

We graphically represent the hypersequent above as:



The root of this hypersequent is the sequent $y : \bullet A, x : \bullet B \vdash w : D$. Its children are $z_1 : C_1 \vdash y : \bullet A$ and $z_2 : C_2 \vdash x : \bullet B$, which are respectively connected to the root by their conclusions $y : \bullet A$ and $x : \bullet B$.

Remark 1 (Bullet annotations) For clarity, we chose to adopt bullet annotations in order to highlight connections among sequents in a hypersequent. However, for provable hypersequents, such connections can actually be inferred by looking at channel names, i.e., a channel that occurs twice in a hypersequent is always connected, whereas a channel that occurs once is not.

3.2 Programs

We give the syntax of LCC programs (to which we also refer as proof terms) in Fig. 1. The syntax is a variation

of that of the internal π-calculus, extended with choreographic primitives. The internal π-calculus allows us to focus on a simple, yet very expressive fragment of the π-calculus [23], as in [7]. Terms in LCC can be *processes* performing communication actions or *choreographies* of interactions. We explain them separately in the following.

Processes. An *(output)* $\overline{x}(y);(P|Q)$ sends a *fresh* name y over channel x and then proceeds with the parallel composition $P|Q$. Dually, *(input)* $x(y);P$ receives y over x and then proceeds as P. In a *(left sel)* $x.\mathsf{inl};P$, we send over channel x the choice of the left branch offered by the receiver. The term *(right sel)* $x.\mathsf{inr};P$ selects the right branch instead. Selections communicate with the term *(case)* $x.\mathsf{case}(P,Q)$, which offers a left branch P and a right branch Q. The term *(par)* $P \mid_x Q$ models parallel composition; here, differently from the output case, the two composed processes are not independent, but share the communication channel x. The term *(res)* is the standard restriction operator, as found in typical process calculi. Terms *(close)* and *(wait)* model, respectively, the request and acceptance for closing a channel, following real-world closing handshakes in communication protocols such as TCP. Closing channels explicitly is unnecessary for our development, but will make the presentation of our results clearer (cf. [24], where a similar notation is adopted).

Choreographies. The term *(res)* belongs also to the choreography fragment of LCC programs. A *(global com)* $\overrightarrow{x(y)};P$ describes a system where a fresh name y is communicated over a channel x, and then continues as P, where y is bound in P. Note that, in order to follow the standard choreography notation, we use an arrow above our choreographic terms: this is not to be confused with standard vector notation. The terms *(global left sel)* and *(global right sel)* model systems where, respectively, a left branch or a right branch is selected on channel x. Unused branches in global selections, e.g., Q in $\overrightarrow{x.\mathsf{l}}(P,Q)$, are unnecessary in our setting since they are never executed; however, their specification will be convenient for our technical development of endpoint projection, which will follow our concretisation transformation in LCC. Finally, term *(global close)* models the closure of a channel.

Note that, differently from § 2, we omit process identifiers in choreographies since our typing will make them redundant (cf. § 8).

3.3 Judgements and Rules

A *judgement* in LCC has the form $P \triangleright \Psi$, where Ψ is a hypersequent and P is a proof term. If we regard LCC as a type theory for our term language, we say that the hypersequent Ψ types the term P.

Valid judgements can be derived using the proof theory of LCC, which consists of two fragments: the *action fragment* and the *interaction fragment*. The action fragment reasons about processes, whereas the interaction fragment reasons about choreographies.

3.3.1 Action Fragment

The action fragment includes the left and right rules for the unit and the multiplicative and additive connectives of ILL, adapted to hypersequents. Left and right rules are reported in Fig. 2. The fragment includes also the structural rules Conn and Scope, which we introduce separately.

Unit. The rules for unit are standard. Rule $\mathsf{1R}$ is the only axiom of LCC; it types a process that requests to close channel x and terminates. Symmetrically, rule $\mathsf{1L}$ types a process that waits for a request to close x; after the channel is closed, it cannot be used anymore in the continuation P.

Tensor. Rule $\otimes\mathsf{R}$ types the output $\overline{x}(y);(P|Q)$, combining the conclusions of the hypersequents used to type the terms P and Q. The continuations P and Q will handle, respectively, the transmitted channel y and channel x. Ensuring that channels y and x are handled by different parallel processes avoids potential deadlocks caused by their interleaving [7,26]. Dually, rule $\otimes\mathsf{L}$ types an input $x(y);P$, by requiring the continuation P to use channels y and x following their respective types.

Linear Implication. The proof term for rule $\multimap\ \mathsf{R}$ is an input $x(y);P$, meaning that the process needs to receive a name of type A before offering behaviour B on channel x. Rule $\multimap\ \mathsf{L}$ types the dual term $\overline{x}(y);(P|Q)$. Note that the prefixes in the proof terms are the same as the ones for tensor rules. This does not introduce any ambiguity, since proof terms are typed differently in the premises and, thus, it is never the case that both connectives could be used for typing the same term [7]. In particular, in the right premises of $\otimes\mathsf{R}$ and $\multimap\ \mathsf{L}$, variable x appears on the right and on the left of $\vdash$, respectively.

$$
\begin{array}{lll}
P,Q,R ::= & \overline{x}(y);(P|Q) \ \textit{(output)} & \mid x(y);P \ \textit{(input)} \\
& \mid x.\mathsf{inl};P \ \textit{(left sel)} & \mid x.\mathsf{inr};P \ \textit{(right sel)} \\
& \mid x.\mathsf{case}(P,Q) \ \textit{(case)} & \mid P \mid_x Q \ \textit{(par)} \\
& \mid \mathsf{close}[x] \ \textit{(close)} & \mid \mathsf{wait}[x];P \ \textit{(wait)} \\
& \mid (\boldsymbol{\nu} x)\,P \ \textit{(res)} & \\
\textit{Choreographies} & \mid \overrightarrow{x(y)};P \ \textit{(global com)} & \mid \overrightarrow{\mathsf{close}}[x]\,;P \ \textit{(global close)} \\
& \mid \overrightarrow{x.\mathsf{l}}\,(P,Q) \ \textit{(global left sel)} & \mid \overrightarrow{x.\mathsf{r}}\,(P,Q) \ \textit{(global right sel)}
\end{array}
$$

Fig. 1 LCC programs.

$$
\frac{P \triangleright \Psi_1 | \Delta_1 \vdash y:A \quad Q \triangleright \Psi_2 | \Delta_2 \vdash x:B}{\overline{x}(y);(P|Q) \triangleright \Psi_1 | \Psi_2 | \Delta_1, \Delta_2 \vdash x:A \otimes B}\otimes\mathsf{R}
\qquad
\frac{P \triangleright \Psi | \Delta, y:A, x:B \vdash T}{x(y);P \triangleright \Psi | \Delta, x:A \otimes B \vdash T}\otimes\mathsf{L}
$$

$$
\frac{P \triangleright \Psi | \Delta, y:A \vdash x:B}{x(y);P \triangleright \Psi | \Delta \vdash x:A \multimap B}\multimap\mathsf{R}
\qquad
\frac{P \triangleright \Psi_1 | \Delta_1 \vdash y:A \quad Q \triangleright \Psi_2 | \Delta_2, x:B \vdash T}{\overline{x}(y);(P|Q) \triangleright \Psi_1 | \Psi_2 | \Delta_1, \Delta_2, x:A \multimap B \vdash T}\multimap\mathsf{L}
$$

$$
\frac{}{\mathsf{close}[x] \triangleright \cdot \vdash x:\mathbf{1}}\mathbf{1}\mathsf{R}
\qquad
\frac{P \triangleright \Psi | \Delta, x:A \vdash T}{x.\mathsf{inl};P \triangleright \Psi | \Delta, x:A\&B \vdash T}\&\mathsf{L}_1
\qquad
\frac{Q \triangleright \Psi | \Delta, x:B \vdash T}{x.\mathsf{inr};Q \triangleright \Psi | \Delta, x:A\&B \vdash T}\&\mathsf{L}_2
$$

$$
\frac{P \triangleright \Psi | \Delta \vdash T}{\mathsf{wait}[x];P \triangleright \Psi | \Delta, x:\mathbf{1} \vdash T}\mathbf{1}\mathsf{L}
\qquad
\frac{P \triangleright \Psi | \Delta \vdash x:A}{x.\mathsf{inl};P \triangleright \Psi | \Delta \vdash x:A \oplus B}\oplus\mathsf{R}_1
\qquad
\frac{Q \triangleright \Psi | \Delta \vdash x:B}{x.\mathsf{inr};Q \triangleright \Psi | \Delta \vdash x:A \oplus B}\oplus\mathsf{R}_2
$$

$$
\frac{P \triangleright \Psi | \Delta \vdash x:A \quad Q \triangleright \Psi | \Delta \vdash x:B}{x.\mathsf{case}(P,Q) \triangleright \Psi | \Delta \vdash x:A\&B}\&\mathsf{R}
\qquad
\frac{P \triangleright \Psi | \Delta, x:A \vdash T \quad Q \triangleright \Psi | \Delta, x:B \vdash T}{x.\mathsf{case}(P,Q) \triangleright \Psi | \Delta, x:A \oplus B \vdash T}\oplus\mathsf{L}
$$

Fig. 2 Left and Right Rules of the Action Fragment.

Additives. The rules for the additive connectives are also straightforward. The proof terms are from [7] and are inspired by standard session typing [13]. We recall that $A \oplus B$ denotes the selection of either A or B from a choice that offers both options, whereas $A\&B$ denotes the offering of a choice with options A and B. For example, rule $\oplus\mathsf{R}_1$ types a left selection $x.\mathsf{inl};P$ on a channel x that appears as the conclusion of a sequent with $A \oplus B$, meaning that the term *provides* the behaviour of selecting from A or B. Rule $\oplus\mathsf{R}_2$ is similar, for the right selection of a branch. Dually, rule $\oplus\mathsf{L}$ types the offering of a choice $x.\mathsf{case}(P,Q)$ on a channel x that appears as a hypothesis with $A \oplus B$, meaning that the term *requires* the environment to provide another process that will select from this choice. As for the connectives $\otimes$ and $\multimap$, the rules for the connective & are similar to those for the connective $\oplus$, with no introduction of ambiguity for the same reasons.

Connection and Scoping. We pull apart the standard Cut rule of ILL, as (4) in § 2, and obtain two rules that depend on hypersequents as an interim place to store information. The first rule, Conn, merges two hypersequents by forming a connection:

$$
\frac{P \triangleright \Psi_1 \mid \Delta_1 \vdash x:A \qquad Q \triangleright \Psi_2 \mid \Delta_2, x:A \vdash T}{P \mid_x Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1 \vdash x:\bullet A \mid \Delta_2, x:\bullet A \vdash T}\ \mathsf{Conn}
$$

The proof term for Conn is parallel composition: in the conclusion, the two terms P and Q are composed in parallel and share channel x. Note that the creation of the connection on x results in annotating the typing of x in both sequents with $\bullet$.

While rule Conn connects the conclusion of a sequent to the compatible hypothesis of another sequent, the second rule, called Scope, scopes a connection by merging two connected sequents:

$$
\frac{P \triangleright \Psi \mid \Delta_1 \vdash x:\bullet A \mid \Delta_2, x:\bullet A \vdash T}{(\boldsymbol{\nu} x)\,P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash T}\ \mathsf{Scope}
$$

The proof term for Scope is the restriction of the channel corresponding to the connection that has been scoped.

3.3.2 Interaction Fragment

Connections are first-class citizens in LCC and are subject to logical reasoning. We give rules for manipulating connections, one for each connective, which correspond to choreographies. Such rules form, together with rule Scope, the interaction fragment of LCC.

Unit. A connection of type $\mathbf{1}$ between two sequents can always be introduced:

$$
\frac{P \triangleright \Psi \mid \Delta \vdash T}{\overrightarrow{\mathsf{close}}[x];P \triangleright \Psi \mid \cdot \vdash x:\bullet\mathbf{1} \mid \Delta, x:\bullet\mathbf{1} \vdash T}\ \mathbf{1}\mathsf{C}
$$

Observe that the choreography $\overrightarrow{\mathsf{close}}[x]; P$ describes the same behaviour as the process $\mathsf{close}[x] \mid_x \mathsf{wait}[x]; P$, and indeed their typing is the same. In general, in LCC the typing of process and choreographic terms describing equivalent behaviour is the same. We will formalise this intuition in § 5.

Tensor. The connection rule for $\otimes$ combines two connections between three sequents. Technically, when two sequents $\Delta_1 \vdash y : \bullet A$ and $\Delta_2 \vdash x : \bullet B$ are connected to a third sequent $\Delta_3, y : \bullet A, x : \bullet B \vdash T$, we can merge the two connections into a single one, obtaining the sequents $\Delta_1, \Delta_2 \vdash x : \bullet A \otimes B$ and $\Delta_3, x : \bullet A \otimes B \vdash T$:

$$\frac{P \triangleright \Psi \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2 \vdash x : \bullet B \mid \Delta_3, y : \bullet A, x : \bullet B \vdash T}{\overrightarrow{x(y)}; P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash x : \bullet A \otimes B \mid \Delta_3, x : \bullet A \otimes B \vdash T} \otimes \mathsf{C}$$

Rule $\otimes\mathsf{C}$ corresponds to typing a choreographic communication $\overrightarrow{x(y)}; P$. This rule is the formalisation in LCC of the cut reduction discussed in § 2. Term P will then perform communications on channel y with type A and channel x with type B.

Linear Implication. The rule for $\multimap$ manipulates connections with a causal dependency: if $\Delta_1 \vdash y : \bullet A$ is connected to $\Delta_2, y : \bullet A \vdash x : \bullet B$, which is connected to $\Delta_3, x : \bullet B \vdash T$, then $\Delta_2 \vdash x : \bullet A \multimap B$ is connected to $\Delta_1, \Delta_3, x : \bullet A \multimap B \vdash T$.

$$\frac{P \triangleright \Psi \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, y : \bullet A \vdash x : \bullet B \mid \Delta_3, x : \bullet B \vdash T}{\overrightarrow{x(y)}; P \triangleright \Psi \mid \Delta_2 \vdash x : \bullet A \multimap B \mid \Delta_1, \Delta_3, x : \bullet A \multimap B \vdash T} \multimap \mathsf{C}$$

Rule $\multimap\,\mathsf{C}$ types a communication $\overrightarrow{x(y)}; P$. The prefix $\overrightarrow{x(y)}$ is the same as that of rule $\otimes\mathsf{C}$, similarly to the action fragment for the connectives $\otimes$ and $\multimap$. Differently from rule $\otimes\mathsf{C}$, the usage of channel x in the continuation P has a causal dependency on y, whereas in $\otimes\mathsf{C}$ the two channels are independent.

Additives. The rules for the additive connectives follow similar reasoning and are reported in Fig. 3. Rule $\&\mathsf{C}_1$ types a choreography that selects the left branch on x and then proceeds as P, provided that x is not used in Q since the latter is unused.

We call C-rules the interaction rules for manipulating connections. C-rules represent cut reductions in ILL, following the intuition presented in § 2.

Example 1 In Fig. 4, we formalise and extend our example from § 2 as follows: Process $P_{\mathsf{client}'}$ implements a new version of the client, which selects the right choice of a branching on channel x and then asks for some tea; then, it proceeds as P_{client} from § 2. Note that we have enhanced the processes with all expected closing of channels. The server $P_{\mathsf{server}'}$, instead, now offers to the client a choice between buying a tea (as in § 2) and getting a free glass of water. Since the water is free, the payment to the bank is not performed in this case. In either case, the bank is notified of whether a payment will occur or not, respectively right and left branch in P_{bank}. The processes are composed as a system in P.

Term C is the equivalent choreographic representation of P. We can type channel x as $(\mathbf{string} \otimes \mathsf{end}) \oplus (\mathbf{string} \otimes (\mathbf{string} \multimap \mathsf{end}) \multimap \mathsf{end})$ in both C and P. The type of channel b is: $\mathsf{end} \oplus (\mathbf{string} \otimes \mathsf{end})$. For clarity, we have used concrete data types instead of the abstract basic type $\mathbf{1}$. □

4 Semantics

We now derive an operational semantics for LCC programs from our proof theory, by obtaining the standard relations of structural equivalence $\equiv$ and reduction $\rightarrow$ as theorems of LCC. For example, the π-calculus rule $(\boldsymbol{\nu} w)\,(P \mid_x Q) \equiv (\boldsymbol{\nu} w)\,P \mid_x Q$ (for $w \notin \mathsf{fn}(Q)$) can be derived as a proof transformation, as shown in Fig. 5.

4.1 Commuting Conversions ($\equiv$)

The structural equivalence of LCC ($\equiv$) is defined in terms of *commuting conversions*, i.e., admissible permutations of rule applications in proofs. In ILL, commuting conversions concern the cut rule. However, since in LCC the cut rule has been split into Scope and Conn, we need to introduce two sets of commuting conversions, one for rule Scope, and one for rule Conn. In the sequel, we report commuting conversions between proofs by giving the corresponding process and choreography terms (the complete LCC proofs that justify these conversions are given in [17, Appendix C]).

Commuting Conversions for Scope. Commuting conversions for Scope correspond to permuting restriction with other operators in LCC programs. We report them in Fig. 6, where we assume variables to be distinct. For example, $[\mathsf{Scope}/\otimes\,\mathsf{R}/\mathsf{L}]$ says that an application of rule Scope to the conclusion of rule $\otimes\mathsf{R}$ can be commuted so that we can apply $\otimes\mathsf{R}$ such that the conclusion of Scope becomes its left (L) premise. Note that the top-level LCC terms of some cases are identical, e.g., $[\mathsf{Scope}/\otimes\,\mathsf{R}/\mathsf{L}]$ and $[\mathsf{Scope}/\multimap\,\mathsf{L}/\mathsf{L}]$, but the subterms are different since they have different typing.

$$\frac{P \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x{:}\bullet A \mid \Delta_2, x{:}\bullet A \vdash T \qquad Q \triangleright \Psi' \mid \Delta_1 \vdash x{:}B}{\overrightarrow{x.\mathsf{l}}\,(P,Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x{:}\bullet A \& B \mid \Delta_2, x{:}\bullet A \& B \vdash T}\ \&\mathsf{C}_1$$

$$\frac{P \triangleright \Psi \mid \Delta_1 \vdash x{:}A \qquad Q \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x{:}\bullet B \mid \Delta_2, x{:}\bullet B \vdash T}{\overrightarrow{x.\mathsf{r}}\,(P,Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x{:}\bullet A \& B \mid \Delta_2, x{:}\bullet A \& B \vdash T}\ \&\mathsf{C}_2$$

$$\frac{P \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x{:}\bullet A \mid \Delta_2, x{:}\bullet A \vdash T \qquad Q \triangleright \Psi' \mid \Delta_2, x{:}B \vdash T}{\overrightarrow{x.\mathsf{l}}\,(P,Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x{:}\bullet A \oplus B \mid \Delta_2, x{:}\bullet A \oplus B \vdash T}\ \oplus\mathsf{C}_1$$

$$\frac{P \triangleright \Psi \mid \Delta_2, x{:}A \vdash T \qquad Q \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x{:}\bullet B \mid \Delta_2, x{:}\bullet B \vdash T}{\overrightarrow{x.\mathsf{r}}\,(P,Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x{:}\bullet A \oplus B \mid \Delta_2, x{:}\bullet A \oplus B \vdash T}\ \oplus\mathsf{C}_2$$

Fig. 3 Interaction Fragment, Rules for the Additives.

$$\begin{array}{rcl}
P_{\mathsf{client}'} & = & x.\mathsf{inr};\ \overline{x}(tea);\ \Big(\ \mathsf{close}[tea] \mid x(tr);\ \overline{tr}(p);\ (\mathsf{close}[p] | \mathsf{wait}[tr];\ \mathsf{close}[x]\ )\ \Big) \\
P_{\mathsf{server}'} & = & x.\mathsf{case}\left(\begin{array}{l} x(water);\ b.\mathsf{inl};\ \mathsf{wait}[water];\ \mathsf{wait}[x];\ \mathsf{close}[b], \\ x(tea);\ \overline{x}(tr);\ \left(\begin{array}{l} tr(p);\ \mathsf{wait}[tea];\ \mathsf{wait}[p];\ \mathsf{close}[tr]\ \mid \\ b.\mathsf{inr};\ \overline{b}(m);\Big(\mathsf{close}[m] \mid \mathsf{wait}[x];\ \mathsf{close}[b]\Big) \end{array}\right) \end{array}\right) \\
P_{\mathsf{bank}'} & = & b.\mathsf{case}(\quad \mathsf{wait}[b];\ \mathsf{close}[z],\quad b(m);\ \mathsf{wait}[m];\ \mathsf{wait}[b];\ \mathsf{close}[z]\quad) \\
P & = & (\boldsymbol{\nu} x)\,(P_{\mathsf{client}'} \mid_x (\boldsymbol{\nu} b)\,(P_{\mathsf{server}'} \mid_b P_{\mathsf{bank}'}))
\end{array}$$

$$C \quad = \quad (\boldsymbol{\nu} x)\,(\boldsymbol{\nu} b)\ \overrightarrow{x.\mathsf{r}}\left(\begin{array}{l} x(water);\ b.\mathsf{inl};\ \mathsf{wait}[water];\ \mathsf{wait}[x];\ \mathsf{close}[b], \\ \overrightarrow{x(tea)};\ \overrightarrow{x(tr)};\ \overrightarrow{tr(p)};\ \overrightarrow{b.\mathsf{r}}\left(\begin{array}{l} \mathsf{wait}[b];\ \mathsf{close}[z], \\ \overrightarrow{b(m)};\ \overrightarrow{\mathsf{close}[tea, p, tr, m, x, b]} \end{array}\right) \end{array}\right)$$

Fig. 4 Beverage Dispenser Example.

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash y : \bullet D \mid \Delta, y : \bullet D \vdash x : A \qquad Q \triangleright \Psi' \mid \Delta', x : A \vdash T}{P \mid_x Q \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash y : \bullet D \mid \Delta, y : \bullet D \vdash x : \bullet A \mid \Delta', x : \bullet A \vdash T}\ \mathsf{Conn}}{(\boldsymbol{\nu} y)\,(P \mid_x Q) \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta \vdash x : \bullet A \mid \Delta', x : \bullet A \vdash T}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash y{:}\bullet D \mid \Delta, y{:}\bullet D \vdash x : A}{(\boldsymbol{\nu} y)\,P \triangleright \Psi \mid \Delta_1, \Delta \vdash x : A}\ \mathsf{Scope} \qquad Q \triangleright \Psi' \mid \Delta', x : A \vdash T}{(\boldsymbol{\nu} y)\,P \mid_x Q \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta \vdash x : \bullet A \mid \Delta', x : \bullet A \vdash T}\ \mathsf{Conn}$$

Fig. 5 A Proof Transformation, Structural Equivalence.

Commuting Conversions for Conn. The commuting conversions for rule Conn, reported in Fig. 7, correspond to commuting the parallel operator with other terms. For example, rule [Conn/Conn] is the standard associativity of parallel in the π-calculus. Also, [Conn/ $\otimes$ C/L] says that $\overrightarrow{x(y)}$ in $\overrightarrow{x(y)}; P \mid_w Q$ can always be executed before Q as far as x and y do not occur in Q. This captures the concurrent behaviour of choreographies in [10]. Note that some of the rules are not standard for the π-calculus, e.g., [Conn/ $\multimap$ R/R], but this does not alter the intended semantics of parallel (cf. § 8, Semantics).

Since conversions preserve the concluding judgement of a proof, we have:

Theorem 1 (Subject Congruence) *$P \triangleright \Psi$ and $P \equiv Q$ implies that $Q \triangleright \Psi$.*

4.2 Reductions ($\rightarrow$)

Similarly to structural equivalence, we derive the reduction semantics for LCC programs from proof transfor-

[Scope/Conn/L]	$(\boldsymbol{\nu} y)\,(P \mid_x Q) \equiv (\boldsymbol{\nu} y)\,P \mid_x Q$	$\big(y \notin \mathsf{fn}(Q)\big)$
[Scope/Conn/R]	$(\boldsymbol{\nu} y)\,(P \mid_x Q) \equiv P \mid_x (\boldsymbol{\nu} y)\,Q$	$\big(y \notin \mathsf{fn}(P)\big)$
[Scope/Scope]	$(\boldsymbol{\nu} y)\,(\boldsymbol{\nu} x)\,P \equiv (\boldsymbol{\nu} x)\,(\boldsymbol{\nu} y)\,P$	
[Scope/1L]	$(\boldsymbol{\nu} x)\,\mathsf{wait}[y]; P \equiv \mathsf{wait}[y]; (\boldsymbol{\nu} x)\,P$	
[Scope/$\otimes$ R/L], [Scope/$\multimap$ L/L]	$(\boldsymbol{\nu} w)\,\overline{x}(y); (P \mid Q) \equiv \overline{x}(y); ((\boldsymbol{\nu} w)\,P \mid Q)$	$\big(w \notin \mathsf{fn}(Q)\big)$
[Scope/$\otimes$ R/R], [Scope/$\multimap$ L/R]	$(\boldsymbol{\nu} w)\,\overline{x}(y); (P \mid Q) \equiv \overline{x}(y); (P \mid (\boldsymbol{\nu} w)\,Q)$	$\big(w \notin \mathsf{fn}(P)\big)$
[Scope/$\otimes$ L], [Scope/$\multimap$ R]	$(\boldsymbol{\nu} w)\,x(y); P \equiv x(y); (\boldsymbol{\nu} w)\,P$	
[Scope/$\oplus$ R$_1$], [Scope/&L$_1$]	$(\boldsymbol{\nu} w)\,x.\mathsf{inl}; P \equiv x.\mathsf{inl}; (\boldsymbol{\nu} w)\,P$	
[Scope/$\oplus$ R$_2$], [Scope/&L$_2$]	$(\boldsymbol{\nu} w)\,x.\mathsf{inr}; P \equiv x.\mathsf{inr}; (\boldsymbol{\nu} w)\,P$	
[Scope/$\oplus$ L], [Scope/&R]	$(\boldsymbol{\nu} w)\,x.\mathsf{case}(P, Q) \equiv x.\mathsf{case}((\boldsymbol{\nu} w)\,P\ ,\ (\boldsymbol{\nu} w)\,Q)$	
[Scope/1C]	$(\boldsymbol{\nu} w)\ \overrightarrow{\mathsf{close}}[x]; P \equiv \overrightarrow{\mathsf{close}}[x]; (\boldsymbol{\nu} w)\,P$	
[Scope/$\otimes$ C], [Scope/$\multimap$ C]	$(\boldsymbol{\nu} w)\ \overrightarrow{x(y)}; P \equiv \overrightarrow{x(y)}; (\boldsymbol{\nu} w)\,P$	
[Scope/$\oplus$ C$_1$/L], [Scope/&C$_1$/L]	$(\boldsymbol{\nu} w)\ \overrightarrow{x.\mathsf{l}}\,(P, Q) \equiv \overrightarrow{x.\mathsf{l}}\,((\boldsymbol{\nu} w)\,P, Q)$	$\big(w \notin \mathsf{fn}(Q)\big)$
[Scope/$\oplus$ C$_1$/L/R], [Scope/&C$_1$/L/R]	$(\boldsymbol{\nu} w)\ \overrightarrow{x.\mathsf{l}}\,(P, Q) \equiv \overrightarrow{x.\mathsf{l}}\,((\boldsymbol{\nu} w)\,P, (\boldsymbol{\nu} w)\,Q)$	$\big(w \in \mathsf{fn}(Q)\big)$
[Scope/$\oplus$ C$_2$/R], [Scope/&C$_2$/R]	$(\boldsymbol{\nu} w)\ \overrightarrow{x.\mathsf{r}}\,(P, Q) \equiv \overrightarrow{x.\mathsf{r}}\,(P, (\boldsymbol{\nu} w)\,Q)$	$\big(w \notin \mathsf{fn}(P)\big)$
[Scope/$\oplus$ C$_2$/L/R], [Scope/&C$_2$/L/R]	$(\boldsymbol{\nu} w)\ \overrightarrow{x.\mathsf{r}}\,(P, Q) \equiv \overrightarrow{x.\mathsf{r}}\,((\boldsymbol{\nu} w)\,P, (\boldsymbol{\nu} w)\,Q)$	$\big(w \in \mathsf{fn}(P)\big)$

Fig. 6 Commuting Conversions ($\equiv$) for Scope (Restriction).

mations. Formally, the reduction relation $\rightarrow$ is defined by the transformations reported in Fig. 8, closed up to structural equivalence $\equiv$. Reductions are standard for both processes and choreographies (cf. [23,10]): processes are reduced when they are the parallel composition of compatible actions, while choreographies can always be reduced. The LCC proof transformations for reductions are reported in [17, Appendix C]. Using a slight abuse of notation, we label each reduction with the channel it uses. Choreography reductions are also annotated with $\bullet$. We use t to range over labels of the form x or $\bullet x$. As for commuting conversions, reductions preserve judgements:

Theorem 2 (Subject Reduction) *$P \triangleright \Psi$ and $P \xrightarrow{t} Q$ implies that $Q \triangleright \Psi$.*

4.3 Scope Elimination (Normalisation)

We can use commuting conversions and reductions to permute and reduce all applications of Scope in a proof until the proof is Scope-free. Since applications of Scope correspond to restrictions in LCC programs, the latter can always progress until all communications on restricted channels are executed. We denote with $P \xrightarrow{\tilde{t}} Q$ a sequence of reductions $P \xrightarrow{t_1} \ldots \xrightarrow{t_n} Q$, where $\tilde{t} = t_1, \ldots, t_n$.

Theorem 3 (Termination/Deadlock-freedom) *$P \triangleright \Psi$ implies there exist Q restriction-free and $\tilde{t}$ such that $P \xrightarrow{\tilde{t}} Q$ and $Q \triangleright \Psi$.*

Proof The proof follows the same intuition of that of cut elimination in ILL. We proceed by induction on the size of the proof for $P \triangleright \Psi$ and the sizes of the types in Ψ. If a reduction from Fig. 8 is applicable, then we apply it. For all such reductions, we observe that the size of the proof and/or the size of the types decreases and therefore the thesis follows by induction hypothesis. Otherwise, we can apply one of the structural equivalences from Fig. 6. In this case, the proof gets smaller while the sizes of the types stay the same. Again, we obtain the thesis by applying the induction hypothesis. □

5 Choreography Extraction and Endpoint Projection

In LCC, a judgement containing connections can be derived by either (i) using the action fragment, corresponding to processes, or (ii) using the interaction fragment, corresponding to choreographies. Let us consider the two following proofs:

$$\dfrac{\dfrac{\dfrac{}{\mathsf{close}[x] \triangleright \cdot \vdash x : \mathbf{1}}\,1\mathsf{R} \qquad \dfrac{\dfrac{}{\mathsf{close}[y] \triangleright \cdot \vdash y : \mathbf{1}}\,1\mathsf{R}}{\mathsf{wait}[x]; \mathsf{close}[y] \triangleright x : \mathbf{1} \vdash y : \mathbf{1}}\,1\mathsf{L}}{\mathsf{close}[x] \mid_x \mathsf{wait}[x]; \mathsf{close}[y] \triangleright \cdot \vdash x : \bullet\mathbf{1} \mid x : \bullet\mathbf{1} \vdash y : \mathbf{1}}\,\mathsf{Conn}}{(\boldsymbol{\nu} x)\,(\mathsf{close}[x] \mid_x \mathsf{wait}[x]; \mathsf{close}[y]) \triangleright \cdot \vdash y : \mathbf{1}}\,\mathsf{Scope}$$

[Conn/Conn]	$(P \mid_y Q) \mid_x R \equiv P \mid_y (Q \mid_x R)$	
[Conn/1L/L]	$\mathsf{wait}[x]; P \mid_y Q \equiv \mathsf{wait}[x]; (P \mid_y Q)$	
[Conn/1L/R]	$P \mid_y \mathsf{wait}[x]; Q \equiv \mathsf{wait}[x]; (P \mid_y Q)$	
[Conn/ $\otimes$R/R/L], [Conn/$\multimap$L/R/L]	$P \mid_w \overline{x}(y); (Q \mid R) \equiv \overline{x}(y); ((P \mid_w Q) \mid R)$	
[Conn/ $\otimes$R/R/R], [Conn/$\multimap$L/R/R]	$P \mid_w \overline{x}(y); (Q \mid R) \equiv \overline{x}(y); (Q \mid (P \mid_w R))$	
[Conn/ $\otimes$ L/L]	$x(y); P \mid_w Q \equiv x(y); (P \mid_w Q)$	
[Conn/ $\otimes$ L/R], [Conn/ $\multimap$ R/R]	$P \mid_w x(y); Q \equiv x(y); (P \mid_w Q)$	
[Conn/ $\multimap$ L/L/R]	$\overline{x}(y); (P \mid Q) \mid_w R \equiv \overline{x}(y); (P \mid (Q \mid_w R))$	
[Conn/ $\oplus$ R$_1$/R], [Conn/&L$_1$/R]	$P \mid_w x.\mathsf{inl}; Q \equiv x.\mathsf{inl}; (P \mid_w Q)$	
[Conn/ $\oplus$ R$_2$/R], [Conn/&L$_2$/R]	$P \mid_w x.\mathsf{inr}; Q \equiv x.\mathsf{inr}; (P \mid_w Q)$	
[Conn/ $\oplus$ L/L]	$x.\mathsf{case}(P,Q) \mid_w R \equiv x.\mathsf{case}((P \mid_w R), (Q \mid_w R))$	
[Conn/ $\oplus$ L/R], [Conn/&R/R]	$P \mid_w x.\mathsf{case}(Q,R) \equiv x.\mathsf{case}((P \mid_w Q), (P \mid_w R))$	
[Conn/&L$_1$/L]	$x.\mathsf{inl}; P \mid_w Q \equiv x.\mathsf{inl}; (P \mid_w Q)$	
[Conn/&L$_2$/L]	$x.\mathsf{inr}; P \mid_w Q \equiv x.\mathsf{inr}; (P \mid_w Q)$	
[Conn/1C/L]	$\overrightarrow{\mathsf{close}}[x]; P \mid_w Q \equiv \overrightarrow{\mathsf{close}}[x]; (P \mid_w Q)$	
[Conn/1C/R]	$P \mid_w \overrightarrow{\mathsf{close}}[x]; Q \equiv \overrightarrow{\mathsf{close}}[x]; (P \mid_w Q)$	
[Conn/$\otimes$ C/L], [Conn/ $\multimap$C/L]	$\overrightarrow{x(y)}; P \mid_w Q \equiv \overrightarrow{x(y)}; (P \mid_w Q)$	$\big(y \notin \mathsf{fn}(Q)\big)$
[Conn/$\otimes$C/R], [Conn/ $\multimap$C/R]	$P \mid_w \overrightarrow{x(y)}; Q \equiv \overrightarrow{x(y)}; (P \mid_w Q)$	$\big(y \notin \mathsf{fn}(P)\big)$
[Conn/$\oplus$ C$_1$/L]	$\overrightarrow{x.\mathsf{l}}(P,Q) \mid_w R \equiv \overrightarrow{x.\mathsf{l}}((P \mid_w R), (Q \mid_w R))$	$\big(w \in \mathsf{fn}(P) \cap \mathsf{fn}(Q)\big)$
[Conn/$\oplus$ C$_1$/R], [Conn/&C$_1$/R]	$P \mid_w \overrightarrow{x.\mathsf{l}}(Q,R) \equiv \overrightarrow{x.\mathsf{l}}((P \mid_w Q), (P \mid_w R))$	$\big(w \in \mathsf{fn}(Q) \cap \mathsf{fn}(R)\big)$
[Conn/$\oplus$C$_1$/R/L],[Conn/&C$_1$/R/L]	$P \mid_w \overrightarrow{x.\mathsf{l}}(Q,R) \equiv \overrightarrow{x.\mathsf{l}}((P \mid_w Q), R)$	$\big(w \in \mathsf{fn}(Q) \setminus \mathsf{fn}(R)\big)$
[Conn/$\oplus$ C$_2$/L]	$\overrightarrow{x.\mathsf{r}}(P,Q) \mid_w R \equiv \overrightarrow{x.\mathsf{r}}((P \mid_w R), (Q \mid_w R))$	$\big(w \in \mathsf{fn}(P) \cap \mathsf{fn}(Q)\big)$
[Conn/$\oplus$ C$_2$/R], [Conn/&C$_2$/R]	$P \mid_w \overrightarrow{x.\mathsf{r}}(Q,R) \equiv \overrightarrow{x.\mathsf{r}}((P \mid_w Q), (P \mid_w R))$	$\big(w \in \mathsf{fn}(Q) \cap \mathsf{fn}(R)\big)$
[Conn/$\oplus$C$_2$/R/R],[Conn/&C$_2$/R/L]	$P \mid_w \overrightarrow{x.\mathsf{r}}(Q,R) \equiv \overrightarrow{x.\mathsf{r}}(Q, (P \mid_w R))$	$\big(w \in \mathsf{fn}(R) \setminus \mathsf{fn}(Q)\big)$
[Conn/&C$_1$/L]	$\overrightarrow{x.\mathsf{l}}(P,Q) \mid_w R \equiv \overrightarrow{x.\mathsf{l}}((P \mid_w R), Q)$	$\big(w \in \mathsf{fn}(P) \setminus \mathsf{fn}(Q)\big)$
[Conn/&C$_2$/L]	$\overrightarrow{x.\mathsf{r}}(P,Q) \mid_w R \equiv \overrightarrow{x.\mathsf{r}}(P, (Q \mid_w R))$	$\big(w \in \mathsf{fn}(Q) \setminus \mathsf{fn}(P)\big)$

Fig. 7 Commuting Conversions ($\equiv$) for Conn (Parallel Composition).

and

$$\dfrac{\dfrac{}{\mathsf{close}[y] \triangleright \cdot \vdash y : \mathbf{1}}\ 1\mathsf{R}}{\overrightarrow{\mathsf{close}}[x]\ ; \mathsf{close}[y] \triangleright \cdot \vdash x : \bullet\mathbf{1} \mid x : \bullet\mathbf{1} \vdash y : \mathbf{1}}\ 1\mathsf{C}}{(\boldsymbol{\nu} x)\,(\overrightarrow{\mathsf{close}}[x]\ ; \mathsf{close}[y]) \triangleright \cdot \vdash y : \mathbf{1}}\ \mathsf{Scope}$$

The two proofs above reach the same hypersequent by the respective methodologies (i) and (ii). In this section, we formally relate the two methodologies, deriving procedures of choreography extraction and endpoint projection from proof equivalences.

As an example, consider the equivalence $[\alpha\gamma_{\otimes}]$ reported in Fig. 9. The equivalence $[\alpha\gamma_{\otimes}]$ transforms a proof of a connection of type $A \otimes B$ from the action fragment into an equivalent proof in the interaction fragment, and vice versa. We report the equivalences for extraction and projection in Fig. 10, presenting their proof terms. The full LCC proof transformations can be found in [17, Appendix C]. We read these equivalences from left to right for extraction, denoted by $\overset{x}{\rightarrowtail_{\mathsf{e}}}$, and from right to left for projection, denoted by $\overset{x}{\twoheadrightarrow_{\mathsf{p}}}$, where x is the name of the connection used for the transformation. Note how a choreography term corresponds to the parallel composition of two processes with the same behaviour. It is also clear why the unselected process Q in $\overrightarrow{x.\mathsf{l}}(P,Q)$ is necessary for projecting the corresponding case process. As for reductions ($\overset{x}{\longrightarrow}$), we consider the transformations $\overset{x}{\rightarrowtail_{\mathsf{e}}}$ and $\overset{x}{\twoheadrightarrow_{\mathsf{p}}}$ closed up to structural equivalence $\equiv$, and we denote sequences of their applications respectively with $\overset{\tilde{x}}{\rightarrowtail_{\mathsf{e}}}$ and $\overset{\tilde{x}}{\twoheadrightarrow_{\mathsf{p}}}$.

In the remainder of this section, we proceed as follows. First, we use our equivalences to prove that rule Conn and C-rules are admissible in a mutually exclusive way. This means that proofs containing applications of Conn can always be transformed into logically-equivalent proofs with applications of C-rules. Vice versa, proofs containing applications of C-rules can always be transformed into logically-equivalent proofs with appli-

$$
\begin{array}{rl}
[\beta_{\mathbf{1}}] & (\boldsymbol{\nu} x)\,(\mathsf{close}[x] \mid_x \mathsf{wait}[x]; Q) \xrightarrow{x} Q \\
[\beta_{\otimes}] & (\boldsymbol{\nu} x)\,(\overline{x}(y); (P|Q) \mid_x x(y); R) \xrightarrow{x} (\boldsymbol{\nu} y)\,(\boldsymbol{\nu} x)\,\big(P \mid_y (Q \mid_x R)\big) \\
[\beta_{\multimap}] & (\boldsymbol{\nu} x)\,(x(y); P \mid_x \overline{x}(y); (Q|R)) \xrightarrow{x} (\boldsymbol{\nu} x)\,(\boldsymbol{\nu} y)\,\big((Q \mid_y P) \mid_x R\big) \\
[\beta_{\oplus_1}] & (\boldsymbol{\nu} x)\,(x.\mathsf{inl}; P \mid_x x.\mathsf{case}(Q, R)) \xrightarrow{x} (\boldsymbol{\nu} x)\,(P \mid_w Q) \\
[\beta_{\oplus_2}] & (\boldsymbol{\nu} x)\,(x.\mathsf{inr}; P \mid_x x.\mathsf{case}(Q, R)) \xrightarrow{x} (\boldsymbol{\nu} x)\,(P \mid_x R) \\
[\beta_{\&_1}] & (\boldsymbol{\nu} x)\,(x.\mathsf{case}(P, Q) \mid_x x.\mathsf{inl}; R) \xrightarrow{x} (\boldsymbol{\nu} x)\,(P \mid_x R) \\
[\beta_{\&_2}] & (\boldsymbol{\nu} x)\,(x.\mathsf{case}(P, Q) \mid_x x.\mathsf{inr}; R) \xrightarrow{x} (\boldsymbol{\nu} x)\,(Q \mid_x R) \\
[\beta_{1\mathsf{C}}] & (\boldsymbol{\nu} x)\ \overrightarrow{\mathsf{close}}[x]; P \xrightarrow{\bullet x} P \\
[\beta_{\otimes\mathsf{C}}], [\beta_{\multimap\mathsf{C}}] & (\boldsymbol{\nu} x)\ \overrightarrow{x(y)}; P \xrightarrow{\bullet x} (\boldsymbol{\nu} y)\,(\boldsymbol{\nu} x)\,P \\
[\beta_{\&\mathsf{C}_1}], [\beta_{\oplus\mathsf{C}_1}] & (\boldsymbol{\nu} x)\ \overrightarrow{x.\mathsf{l}}\,(P, Q) \xrightarrow{\bullet x} (\boldsymbol{\nu} x)\,P \\
[\beta_{\&\mathsf{C}_2}], [\beta_{\oplus\mathsf{C}_2}] & (\boldsymbol{\nu} x)\ \overrightarrow{x.\mathsf{r}}\,(P, Q) \xrightarrow{\bullet x} (\boldsymbol{\nu} x)\,Q
\end{array}
$$

Fig. 8 Reductions.

$$
\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta_1 \vdash y{:}A \qquad Q \triangleright \Psi_2 \mid \Delta_2 \vdash x{:}B}{\overline{x}(y); (P|Q) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1, \Delta_2 \vdash x{:}A \otimes B}\otimes\mathsf{R} \qquad \dfrac{R \triangleright \Psi_3 \mid \Delta_3, y{:}A, x{:}B \vdash T}{x(y); R \triangleright \Delta_3, x{:}A \otimes B \vdash T}\otimes\mathsf{L}}{\overline{x}(y); (P|Q) \mid_x x(y); R \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1, \Delta_2 \vdash x{:}\bullet A \otimes B \mid \Delta_3, x{:}\bullet A \otimes B \vdash T}\,\mathsf{Conn}
$$

can be extracted to (denoted by $\overset{x}{\rightarrowtail_{\mathsf{e}}}$), can be projected from (denoted by $\overset{x}{\twoheadleftarrow_{\mathsf{p}}}$)

$$
\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta_1 \vdash y{:}A \qquad \dfrac{Q \triangleright \Psi_2 \mid \Delta_2 \vdash x{:}B \qquad R \triangleright \Psi_3 \mid \Delta_3, y{:}A, x{:}B \vdash T}{Q \mid_x R \triangleright \Psi_2 \mid \Psi_3 \mid \Delta_2 \vdash x{:}\bullet B | \Delta_3, y{:}A, x{:}\bullet B \vdash T}\,\mathsf{Conn}^x}{P \mid_y (Q \mid_x R) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash y{:}\bullet A \mid \Delta_2 \vdash x{:}\bullet B \mid \Delta_3, y{:}\bullet A, x{:}\bullet B \vdash T}\,\mathsf{Conn}^y}{\overrightarrow{x(y)}; (P \mid_y (Q \mid_x R)) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1, \Delta_2 \vdash x{:}\bullet A \otimes B \mid \Delta_3, x{:}\bullet A \otimes B \vdash T}\otimes\mathsf{C}^x
$$

Fig. 9 A Proof Transformation, Extraction/Projection.

cations of rule Conn. Second, we use our admissibility results to derive procedures of choreography extraction and endpoint projection for LCC programs. We finally show that the extraction and the projection procedures yield terms that are operationally equivalent according to our semantics.

5.1 Admissibility of Conn

In the remainder, we say that a program is *(par)*-free if it does not contain subterms of the form $P \mid_x Q$. We can now formally state our Theorem for the admissibility of Conn.

Theorem 4 (Conn Admissibility) *Let P and Q be (par)-free and such that:*

$P \triangleright \Psi_1 \mid \Delta_1 \vdash x{:}A$
$Q \triangleright \Psi_2 \mid \Delta_2, x{:}A \vdash T$

Then, there exists a (par)-free R such that:

$R \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1 \vdash x{:}\bullet A \mid \Delta_2, x{:}\bullet A \vdash T$

Proof The proof follows the same intuition of that of Cut admissibility in ILL. We proceed by mutual induction on (i) the respective sizes of the proof derivations for the typings of P and Q, and (ii) the sizes of the types in such proofs. If one of the equivalences in Fig. 10 can be applied, we apply it. The equivalence $[\alpha\gamma_{\mathbf{1}}]$ is the base case. The other equivalences reduce the sizes of the principal types. We then apply the induction hypothesis. Otherwise, we use one of the commuting conversions for rule Conn from Fig. 7 to push the application of Conn up in the derivation. In all such cases, the sizes of the proofs get smaller. We then apply the induction hypothesis. □

5.2 Admissibility of C-rules

We now state an auxiliary Lemma that will be useful later on for reasoning about the admissibility of rules that correspond to choreography terms. We remind the reader that, by convention, we say that a proof term is a process term if it contains no choreographic subterms, i.e., there are no applications of C-rules in its

$$[\alpha\gamma_{\mathbf{1}}] \quad \mathsf{close}[x] \mid_x \mathsf{wait}[x]; P \quad \overset{x}{\underset{x}{\rightleftarrows}}_{\mathsf{e}}{}^{\mathsf{p}} \quad \overrightarrow{\mathsf{close}}[x]; P$$

$$[\alpha\gamma_{\otimes}] \quad \overline{x}(y); (P|Q) \mid_x x(y); R \quad \overset{x}{\underset{x}{\rightleftarrows}}_{\mathsf{e}}{}^{\mathsf{p}} \quad \overrightarrow{x(y)}; \big(P \mid_y (Q \mid_x R)\big)$$

$$[\alpha\gamma_{\multimap}] \quad x(y); P \mid_x \overline{x}(y); (Q|R) \quad \overset{x}{\underset{x}{\rightleftarrows}}_{\mathsf{e}}{}^{\mathsf{p}} \quad \overrightarrow{x(y)}; \big((Q \mid_y P) \mid_x R\big)$$

$$[\alpha\gamma_{\&_1}] \quad x.\mathsf{case}(P, Q) \mid_x x.\mathsf{inl}; R \quad \overset{x}{\underset{x}{\rightleftarrows}}_{\mathsf{e}}{}^{\mathsf{p}} \quad \overrightarrow{x.\mathsf{l}}\,((P \mid_x R), Q)$$

$$[\alpha\gamma_{\&_2}] \quad x.\mathsf{case}(P, Q) \mid_x x.\mathsf{inr}; R \quad \overset{x}{\underset{x}{\rightleftarrows}}_{\mathsf{e}}{}^{\mathsf{p}} \quad \overrightarrow{x.\mathsf{r}}(P\ ,\ Q \mid_x R)$$

$$[\alpha\gamma_{\oplus_1}] \quad x.\mathsf{inl}; P \mid_x x.\mathsf{case}(Q, R) \quad \overset{x}{\underset{x}{\rightleftarrows}}_{\mathsf{e}}{}^{\mathsf{p}} \quad \overrightarrow{x.\mathsf{l}}\,((P \mid_x Q)\ ,\ R)$$

$$[\alpha\gamma_{\oplus_2}] \quad x.\mathsf{inr}; P \mid_x x.\mathsf{case}(Q, R) \quad \overset{x}{\underset{x}{\rightleftarrows}}_{\mathsf{e}}{}^{\mathsf{p}} \quad \overrightarrow{x.\mathsf{r}}(Q\ ,\ (P \mid_x R))$$

Fig. 10 Extraction and Projection.

type derivations. The Lemma below states that a parallel composition inside a process can always be brought up to the top level preserving its typing.

Lemma 1 *Let* P *be a process such that:*

$$P \triangleright \Psi \mid \Delta_1 \vdash x{:}\bullet A \mid \Delta_2, x{:}\bullet A \vdash T$$

Then, there exist two processes Q_1 *and* Q_2 *such that:*

$$Q_1 \mid_x Q_2 \triangleright \Psi \mid \Delta_1 \vdash x{:}\bullet A \mid \Delta_2, x{:}\bullet A \vdash T$$

Proof Since P is a process, we observe that the connection on x must have been obtained by applying rule Conn in the typing derivation of P. We proceed by induction on the distance between this application of Conn and the root of the typing derivation of P. The base case corresponds to when the application of Conn on x is already at the root. Otherwise, we use one of the commuting conversions for rule Conn from Fig. 7 to push the application of Conn towards the root of the typing derivation, therefore reducing the distance to the root by one. Note that we can always apply one of such commuting conversions: since P is a process, its proof contains no applications of C-rules; hence, we know that all the other rule applications beneath the application of Conn do not manipulate the same channel x. We then apply the induction hypothesis. □

Using Lemma 1, we can prove the admissibility of all our C-rules, formally stated below.

Theorem 5 (C-rules Admissibility) *Let* P *and* Q *be processes (*Q *is used only in the additive cases). Then, there exists a process* R *such that:*

(1C)
$P \triangleright \Psi | \Delta \vdash T$
implies
$R \triangleright \Psi | \cdot \vdash x{:}\bullet\mathbf{1} | \Delta, x{:}\bullet\mathbf{1} \vdash T$

(⊗C)
$P \triangleright \Psi \mid \Delta_1 \vdash y{:}\bullet A \mid \Delta_2 \vdash x{:}\bullet B \mid \Delta_3, y{:}\bullet A, x{:}\bullet B \vdash T$
implies
$R \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash x{:}\bullet A \otimes B \mid \Delta_3, x{:}\bullet A \otimes B \vdash T$

(⊸C)
$P \triangleright \Psi | \Delta_1 \vdash y{:}\bullet A | \Delta_2, y{:}\bullet A \vdash x{:}\bullet B | \Delta_3, x{:}\bullet B \vdash T$
implies
$R \triangleright \Psi | \Delta_2 \vdash x{:}\bullet A \multimap B | \Delta_1, \Delta_3, x{:}\bullet A \multimap B \vdash T$

(&C$_1$)
$P \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x{:}\bullet A \mid \Delta_2, x{:}\bullet A \vdash T$ *and*
$Q \triangleright \Psi' \mid \Delta_1 \vdash x{:}B$
implies
$R \triangleright \Psi | \Psi' | \Delta_1 \vdash x{:}\bullet A \& B | \Delta_2, x{:}\bullet A \& B \vdash T$

(&C$_2$)
$P \triangleright \Psi \mid \Delta_1 \vdash x{:}A$ *and*
$Q \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x{:}\bullet B \mid \Delta_2, x{:}\bullet B \vdash T$
implies
$R \triangleright \Psi | \Psi' | \Delta_1 \vdash x{:}\bullet A \& B | \Delta_2, x{:}\bullet A \& B \vdash T$

(⊕C$_1$)
$P \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x{:}\bullet A \mid \Delta_2, x{:}\bullet A \vdash T$ *and*
$Q \triangleright \Psi' \mid \Delta_2, x{:}B \vdash T$
implies
$R \triangleright \Psi | \Psi' | \Delta_1 \vdash x{:}\bullet A \oplus B | \Delta_2, x{:}\bullet A \oplus B \vdash T$

(⊕C$_2$)
$P \triangleright \Psi \mid \Delta_2, x{:}A \vdash T$ *and*
$Q \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x{:}\bullet B \mid \Delta_2, x{:}\bullet B \vdash T$
implies
$R \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x{:}\bullet A \oplus B \mid \Delta_2, x{:}\bullet A \oplus B \vdash T$

Proof The first case, (1C), is obtained by applying rule 1L to P and then by applying Conn to the result and the axiom 1R.

In the case for ($\otimes$C), from Lemma 1, we know that we can rewrite the proof derivation of P into an equivalent proof where the application of the Conn rule on y is at the root. Formally, for some Ψ_1, Ψ_2 such that $\Psi = \Psi_1, \Psi_2$:

$$\frac{P_1 \triangleright \Psi_1 | \Delta_1 \vdash y : A \qquad P_2 \triangleright \Psi_2 | \Delta_2 \vdash x : \bullet B | \Delta_3, y : A, x : \bullet B \vdash T}{P_1 \mid_y P_2 \triangleright \Psi_1, \Psi_2 | \Delta_1 \vdash y : \bullet A | \Delta_2 \vdash x : \bullet B | \Delta_3, y : \bullet A, x : \bullet B \vdash T}$$

Above, we know that the connection on x must be on the right premise, because both hypotheses for x and y are in the same sequent $\Delta_3, y : A, x : \bullet B \vdash T$. Now, we apply again Lemma 1, in order to bring the application of Conn on x to the root of the proof of the right premise, obtaining the proof in Fig. 11(a). We can then construct the proof reported in Fig. 11(b). We have thus proven the thesis, for $R = \overline{x}(y); (P_1 | P_3) \mid_x x(y); P_4$.

All the other cases follow by similar reasoning to that for ($\otimes$C). □

5.3 The Correspondence Theorem

As in linear logic, where the result of cut admissibility is used to define a procedure for eliminating cuts from a proof, we can use our admissibility results for rule Conn and C-rules to derive respective elimination procedures.

In terms of LCC programs, eliminating applications of Conn corresponds to transforming programs containing subterms of the form $P \mid_x Q$ to programs containing only choreography terms.

Corollary 1 (Choreography Extraction) *Let $P \triangleright \Psi$. Then $P \overset{\tilde{x}}{\rightarrowtail}_{\mathsf{e}} Q$ for some $\tilde{x}$ and Q such that $Q \triangleright \Psi$ and Q does not contain subterms of the form $R \mid_x R'$.*

Proof By interpreting the proof of Theorem 4 as an algorithm. We start by transforming the inner-most applications of rule Conn, and then proceeding towards the root of the proof until all applications of Conn are eliminated. □

Opposite to Corollary 1, we can define a procedure for eliminating applications of C-rules, corresponding to transforming choreography terms inside of an LCC program into equivalent process terms.

Corollary 2 (Endpoint Projection) *Let $P \triangleright \Psi$. Then, $P \xrightarrow{\tilde{x}}\!\!\!\!\rightarrow_{\mathsf{p}} Q$ for some $\tilde{x}$ and Q such that $Q \triangleright \Psi$ and Q does not contain choreography terms.*

Proof Similar to the proof for Corollary 1, where we use the proof for Theorem 5 instead of that for Theorem 4. □

Example 2 Using the equivalences in Fig. 10 and $\equiv$, we can transform the processes to the choreography in Example 1 and vice versa. □

We can now present the main property guaranteed by LCC: the extraction and projection procedures preserve the semantics of the transformed programs.

Theorem 6 (Correspondence) *Let $P \triangleright \Psi$. Then:*
(choreography extraction) $P \xrightarrow{\tilde{x}} P'$ implies $P \overset{\tilde{x}}{\rightarrowtail}_{\mathsf{e}} Q$ such that $Q \xrightarrow{\bullet\tilde{x}} P'$.
(endpoint projection) $P \xrightarrow{\bullet\tilde{x}} P'$ implies $P \xrightarrow{\tilde{x}}\!\!\!\!\rightarrow_{\mathsf{p}} Q$ such that $Q \xrightarrow{\tilde{x}} P'$.

Proof For simplicity, we omit proof terms and focus directly on pure logic proofs, ranged over by $\mathcal{D}$, $\mathcal{E}$, $\mathcal{F}$, ...

We proceed by proving the following property:

Let $\tilde{x} = x_1, \ldots, x_n$ and $\tilde{y} = x_n, \ldots, x_1$. Then,

1. $\mathcal{D} \xrightarrow{\tilde{x}} \mathcal{F}$ for some $\mathcal{F}$ implies that $\mathcal{D} \overset{\tilde{x}}{\rightarrowtail}_{\mathsf{e}} \mathcal{E}$ for some $\mathcal{E}$ such that $\mathcal{E} \xrightarrow{\bullet\tilde{x}} \mathcal{F}$.
2. $\mathcal{D} \xrightarrow{\bullet\tilde{x}} \mathcal{F}$ for some $\mathcal{F}$ implies that $\mathcal{D} \xrightarrow{\tilde{y}}\!\!\!\!\rightarrow_{\mathsf{p}} \mathcal{E}$ for some $\mathcal{E}$ such that $\mathcal{E} \xrightarrow{\tilde{x}} \mathcal{F}$.

We show the proof for (1); the proof for (2) is similar. We can depict point (1) with the following diagram:

$$\begin{array}{ccc} \mathcal{D} & \xrightarrow{\tilde{x}} & \mathcal{F} \\ {\scriptstyle \tilde{x}}\big\downarrow_{\mathsf{e}} & \nearrow_{\bullet\tilde{x}} & \\ \mathcal{E} & & \end{array}$$

The proof proceeds now by induction on the length of $\tilde{x}$. In the sequel, $\equiv^y_{\mathsf{c,s}}$ is an abbreviation for $\equiv^y_{\mathsf{c}} \equiv^y_{\mathsf{s}}$, where $\equiv^y_{\mathsf{c}}$ and $\equiv^y_{\mathsf{s}}$ are, respectively, the commuting conversions for Conn and Scope applied to the connection y.

Case $\tilde{x}$ is empty. The thesis holds for $\mathcal{D} = \mathcal{E} = \mathcal{F}$.

Case $\tilde{x} = y, \tilde{z}$. In this case we know that $\mathcal{D} \equiv^y_{\mathsf{c,s}} \xrightarrow{y} \mathcal{D}'$. Our induction hypothesis is then that there exist $\mathcal{E}'$ and $\mathcal{F}$ such that:

$$\begin{array}{ccc} \mathcal{D}' & \xrightarrow{\tilde{z}} & \mathcal{F} \\ {\scriptstyle \tilde{z}}\big\downarrow_{\mathsf{e}} & \nearrow_{\bullet\tilde{z}} & \\ \mathcal{E}' & & \end{array}$$

We construct the thesis from the induction hypothesis by proving that the following diagram commutes:

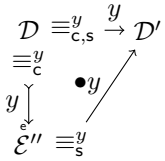
$$\begin{array}{ccccc} \mathcal{D} & \equiv^y_{\mathsf{c,s}} & \xrightarrow{y} \mathcal{D}' & \xrightarrow{\tilde{z}} & \mathcal{F} \\ \equiv^y_{\mathsf{c}} & & \nearrow_{\bullet y} & & \nearrow_{\bullet\tilde{z}} \\ {\scriptstyle y}\big\downarrow_{\mathsf{e}} & & {\scriptstyle \tilde{z}}\big\downarrow_{\mathsf{e}} & & \\ \mathcal{E}'' & \equiv^y_{\mathsf{s}} & \mathcal{E}' & & \\ {\scriptstyle \tilde{z}}\big\downarrow_{\mathsf{e}} & & \nearrow_{\bullet y} & & \\ \mathcal{E} & \equiv^y_{\mathsf{s}} & & & \end{array}$$

$$(a)\quad \dfrac{P_1 \triangleright \Psi_1 | \Delta_1 \vdash y{:}A \qquad \dfrac{P_3 \triangleright \Psi_3 | \Delta_2 \vdash x{:}B \qquad P_4 \triangleright \Psi_4 | \Delta_3, y{:}A, x{:}B \vdash T}{P_3 \mid_x P_4 \triangleright \Psi_3, \Psi_4 | \Delta_2 \vdash x{:}\bullet B \mid \Delta_3, y{:}A, x{:}\bullet B \vdash T}\,\mathsf{Conn}}{P_1 \mid_y (P_3 \mid_x P_4) \triangleright \Psi_1, \Psi_3, \Psi_4 \mid \Delta_1 \vdash y{:}\bullet A \mid \Delta_2 \vdash x{:}\bullet B \mid \Delta_3, y{:}\bullet A, x{:}\bullet B \vdash T}\,\mathsf{Conn}$$

$$(b)\quad \dfrac{\dfrac{P_1 \triangleright \Psi_1 | \Delta_1 \vdash y{:}A \qquad P_3 \triangleright \Psi_3 | \Delta_2 \vdash x{:}B}{\overline{x}(y); (P_1 | P_3) \triangleright \Psi_1 | \Psi_3 | \Delta_1, \Delta_2 \vdash x{:}A \otimes B}\otimes\mathsf{R} \qquad \dfrac{P_4 \triangleright \Delta_3, y{:}A, x{:}B \vdash T}{x(y); P_4 \triangleright \Delta_3, x{:}A \otimes B \vdash T}\otimes\mathsf{L}}{\overline{x}(y); (P_1 | P_3) \mid_x x(y); P_4 \triangleright \Psi_1, \Psi_3, \Psi_4 \mid \Delta_1, \Delta_2 \vdash x{:}\bullet A \otimes B \mid \Delta_3, x{:}\bullet A \otimes B \vdash T}\,\mathsf{Conn}$$

Fig. 11 Elements of the Proof of Theorem 5.

We can split the diagram in two parts. The first part is the upper-left triangle that instantiates every arrow to its one-time application:

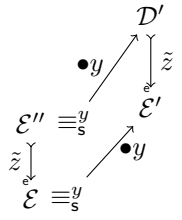


This diagram commutes because, from the definitions of $\equiv$ and $\to$, we can derive:

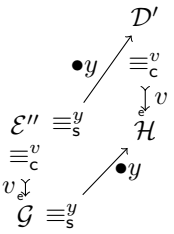
$$\equiv^y_{\mathsf{c}} \equiv^y_{\mathsf{s}} \xrightarrow{y} \quad = \quad \equiv^y_{\mathsf{c}} \overset{y}{\rightarrowtail}_{\mathsf{e}} \equiv^y_{\mathsf{s}} \xrightarrow{\bullet y}$$

The equation above holds because the shapes of the proofs required by $\xrightarrow{y}$ are the same as those required by $\overset{y}{\rightarrowtail}_{\mathsf{e}}$, with the only difference being that $\xrightarrow{y}$ also requires an application of rule Scope on y at the end. Therefore, before commuting such Scope application using $\equiv^y_{\mathsf{s}}$, we can immediately apply $\overset{y}{\rightarrowtail}_{\mathsf{e}}$ and apply $\equiv^y_{\mathsf{s}}$ afterwards, obtaining one of the required shapes for applying $\xrightarrow{\bullet y}$; this is easily verified by case analysis on the definitions of $\equiv$ and $\to$.

Let us now consider the second part of the diagram. We need to prove that the following part commutes, for $y \notin \tilde{z}$:



We observe that this diagram can be obtained by iteratively applying the following, for some v and some proofs $\mathcal{G}$ and $\mathcal{H}$:



This diagram commutes because we can derive the equality (where the left-hand side is what we know from our induction hypothesis):

$$\equiv^y_{\mathsf{s}} \xrightarrow{\bullet y} \equiv^v_{\mathsf{c}} \overset{v}{\rightarrowtail}_{\mathsf{e}} \quad = \quad \equiv^v_{\mathsf{c}} \overset{v}{\rightarrowtail}_{\mathsf{e}} \equiv^y_{\mathsf{s}} \xrightarrow{\bullet y}$$

The equation above holds because commuting conversions on different channels can always be applied one after the other or vice versa. □

6 Relation to Intuitionistic Linear Logic

In this section, we discuss the relationship between the proof theory of LCC and standard (multiplicative-additive) ILL. We recall, from § 3.1, *Hypersequents*, that the hypotheses of a hypersequent Ψ are the non-bulleted hypotheses that appear in the sequents inside of Ψ, and that the conclusion T of Ψ is the only non-bulleted conclusion that appears in Ψ (a valid hypersequent always has exactly one non-bulleted conclusion). At the end of this section, we will prove that if a judgement $P \triangleright \Psi$ with hypotheses $\Delta_1, \ldots, \Delta_n$ and conclusion T is valid in LCC, then the judgement $Q \triangleright \Delta_1, \ldots, \Delta_n \vdash T$ is valid in ILL for some Q.

The rules defining the proof theory of ILL are reported in Fig. 12, where for presentational convenience we use the same proof terms and variable assignments as for LCC. These terms correspond to the ones originally presented in [7,26]. The rules defining ILL are very similar to a subset of those forming the action fragment of LCC. There are two overall differences: all judgements contain only one sequent, and rule Cut, which is not present in LCC, is added. It is immediate to see that ILL is a restriction of the action fragment of LCC

$$\frac{P \triangleright \Delta_1 \vdash y:A \quad Q \triangleright \Delta_2 \vdash x:B}{\overline{x}(y);(P|Q) \triangleright \Delta_1, \Delta_2 \vdash x:A \otimes B}\otimes\mathsf{R} \qquad \frac{P \triangleright \Delta, y:A, x:B \vdash T}{x(y);P \triangleright \Delta, x:A \otimes B \vdash T}\otimes\mathsf{L}$$

$$\frac{P \triangleright \Delta, y:A \vdash x:B}{x(y);P \triangleright \Delta \vdash x:A \multimap B}\multimap\mathsf{R} \qquad \frac{P \triangleright \Delta_1 \vdash y:A \quad Q \triangleright \Delta_2, x:B \vdash T}{\overline{x}(y);(P|Q) \triangleright \Delta_1, \Delta_2, x:A \multimap B \vdash T}\multimap\mathsf{L}$$

$$\frac{}{\mathsf{close}[x] \triangleright \cdot \vdash x:\mathbf{1}}\mathsf{1R} \qquad \frac{P \triangleright \Delta, x:A \vdash T}{x.\mathsf{inl};P \triangleright \Delta, x:A\&B \vdash T}\&\mathsf{L}_1 \qquad \frac{Q \triangleright \Delta, x:B \vdash T}{x.\mathsf{inr};Q \triangleright \Delta, x:A\&B \vdash T}\&\mathsf{L}_2$$

$$\frac{P \triangleright \Delta \vdash T}{\mathsf{wait}[x];P \triangleright \Delta, x:\mathbf{1} \vdash T}\mathsf{1L} \qquad \frac{P \triangleright \Delta \vdash x:A}{x.\mathsf{inl};P \triangleright \Delta \vdash x:A \oplus B}\oplus\mathsf{R}_1 \qquad \frac{Q \triangleright \Delta \vdash x:B}{x.\mathsf{inr};Q \triangleright \Delta \vdash x:A \oplus B}\oplus\mathsf{R}_2$$

$$\frac{P \triangleright \Delta \vdash x:A \quad Q \triangleright \Delta \vdash x:B}{x.\mathsf{case}(P,Q) \triangleright \Delta \vdash x:A\&B}\&\mathsf{R} \qquad \frac{P \triangleright \Delta, x:A \vdash T \quad Q \triangleright \Delta, x:B \vdash T}{x.\mathsf{case}(P,Q) \triangleright \Delta, x:A \oplus B \vdash T}\oplus\mathsf{L}$$

$$\frac{P \triangleright \Delta_1 \vdash x:A \qquad Q \triangleright \Delta_2, x:A \vdash T}{(\boldsymbol{\nu} x)\,(P \mid_x Q) \triangleright \Delta_1, \Delta_2 \vdash T}\mathsf{Cut}$$

Fig. 12 Multiplicative-additive Intuitionistic Linear Logic.

(Fig. 2) to singleton hypersequents. For rule Cut, even though it is not present in LCC we can derive a more general rule by combining rules Conn and Scope:

Proposition 1 (Derivability of Cut in LCC) *The following rule is derivable in LCC:*

$$\frac{P \triangleright \Psi_1 \mid \Delta_1 \vdash x:A \qquad Q \triangleright \Psi_2 \mid \Delta_2, x:A \vdash T}{(\boldsymbol{\nu} x)\,(P \mid_x Q) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1, \Delta_2 \vdash T}\mathsf{Cut}$$

Proof We construct a closed-form proof of the conclusion of Cut starting from its premises:

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta_1 \vdash x:A \qquad Q \triangleright \Psi_2 \mid \Delta_2, x:A \vdash T}{P \mid_x Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1 \vdash x:\bullet A \mid \Delta_2, x:\bullet A \vdash T}\mathsf{Conn}}{(\boldsymbol{\nu} x)\,(P \mid_x Q) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1, \Delta_2 \vdash T}\mathsf{Scope}$$

□

Using Proposition 1 we can prove the following Theorem, which states that all judgements in ILL can be derived also in LCC.

Theorem 7 (From ILL to LCC) *If $P \triangleright \Delta \vdash T$ is a valid judgement in ILL, then it is also a valid judgement in LCC.*

Proof We proceed by induction on the structure of the ILL proof for $P \triangleright \Delta \vdash T$. For each rule that is not Cut in the ILL proof, we simply apply the corresponding rule in the action fragment of LCC. For rule Cut, we apply the construction from Proposition 1. □

We can also prove the opposite, namely that given any LCC proof for a judgement with a singleton hypersequent. We construct a corresponding ILL proof by using endpoint projection.

Lemma 2 (From LCC singletons to ILL) *If $P \triangleright \Delta \vdash T$ is a derivable judgement in LCC, then there exists a process Q such that $Q \triangleright \Delta \vdash T$ is a derivable judgement in ILL.*

Proof By the assumption, Corollary 2 and Theorem 3 we know that there exist $\tilde{x}$, $\tilde{t}$, and Q such that:

$$P \xrightarrow{\tilde{x}}\!\!\!\!\rightarrow_{\mathsf{p}} \xrightarrow{\tilde{t}} Q \qquad \text{and} \qquad Q \triangleright \Delta \vdash T$$

□

As ILL judgements are special cases of LCC judgements where hypersequents are singletons, those LCC judgements in which hypersequents contain more than one sequent cannot be derived in ILL. Nevertheless, we can define a procedure that transforms LCC into ILL proofs by erasing connection related information. Intuitively, we apply rule Scope until all connections are hidden, collapsing our hypersequent of interest and leaving us with its hypotheses and conclusion in a single sequent. We first prove the following auxiliary Lemma. Below, $(\boldsymbol{\nu}\tilde{x})$ is a shortcut for $(\boldsymbol{\nu} x_1) \ldots (\boldsymbol{\nu} x_n)$.

Lemma 3 *Let $P \triangleright \Psi$. Then, there exists $\tilde{x}$ such that $(\boldsymbol{\nu}\tilde{x})\,P \triangleright \Delta \vdash T$, where Δ is the union of all the hypotheses of Ψ and T its conclusion.*

Proof We derive $(\boldsymbol{\nu}\tilde{x})\,P \triangleright \Delta \vdash T$ by applying rule Scope for each connection in Ψ, obtaining by construction that Δ is the union of all the hypotheses of Ψ and T its conclusion. □

By combining Lemma 2 with Lemma 3, we can finally show how to reconduce a general LCC proof to ILL.

Theorem 8 (From LCC to ILL) *Let $P \triangleright \Psi$. Then, there exists Q such that $Q \triangleright \Delta \vdash T$ is a valid judgement in ILL, where Δ is the union of all the hypotheses of Ψ and T its conclusion.*

Proof We first apply the construction from Lemma 3 to obtain $(\boldsymbol{\nu}\tilde{x})\,P \triangleright \Delta \vdash T$. Then, the thesis follows by applying Lemma 2. □

7 Related Work

Linear Logic. Linear logic was first described by Girard as a logic of resources [12]. The connection between linear logic and processes was first formally explored by Bellin and Scott [5]. However, it was not until much later that Caires and Pfenning discovered a Curry-Howard correspondence based on session types [7]. In parallel to the work on linear logic, Pottinger [20] and Avron [2] independently discovered and studied hypersequents, i.e., generalisations of ordinary sequent systems operating with sets of sequents instead of single sequents. This generalisation increases the expressive power of the sequent calculus by allowing additional transfer of information among sequents. This is precisely what we exploit in our proof theory for typing choreographies, where we use hypersequents to capture the structure of a system of processes and their connections.

Our action fragment is inspired by π-DILL [7]. The key difference is that we split rule Cut into rules Conn and Scope, which allows us to (i) reason about choreographies and (ii) type processes where restriction and parallel are used separately. LCC and ILL are essentially equivalent in terms of what conclusions can be derived from a set of hypotheses (§ 6). The hallmark characteristic of LCC is thus the ability to type choreographies and to yield the transformations of extraction and projection. In Classical Processes [26], processes correspond to proofs in classical linear logic, following the line of [7]. We conjecture that LCC can be adapted to the classical setting. Both works [7] and [**?**] can type shared channels that can be used multiple times using exponentials from linear logic. We discuss how LCC could be extended in a similar direction in § 8.

Choreographies. Compositional Choreographies [18] is a language model where choreographies can be mixed with processes to achieve compositionality. The proof theory of LCC can be seen as a foundational model for compositional choreographies, logically reconstructed using linear logic. As a consequence, our work yields an extraction procedure and normalisation results that are not provided in [18]. Our commuting conversions can be seen as a logical characterisation of *swapping* [10], which permutes independent communications in a choreography. For example, using the rules in Fig. 7, we can derive the following equivalences.

$$\overrightarrow{x(y)};(P \mid_u \overrightarrow{z(w)};Q) \equiv (\overrightarrow{x(y)};P) \mid_u (\overrightarrow{z(w)};Q)$$
$$(\overrightarrow{x(y)};P) \mid_u (\overrightarrow{z(w)};Q) \equiv \overrightarrow{z(w)};(\overrightarrow{x(y)};P \mid_u Q)$$

Previous works [14,9,10,18] have formally addressed choreographies and EPP but without providing choreography extraction. Choreography extraction is a known hard problem [3,16,11], and our work is the first to address it for a language supporting channel passing. Probably, the work closest to ours with respect to extraction is the model in [15], where global types are extracted from session types; choreographies are more expressive than global types, since they capture the interleaving of different sessions. In the future, we plan to address standard features supported by other choreography calculi, such as multiparty sessions, asynchrony, replicated services, and nondeterminism [9,10,18].

Our mixing of choreographies with processes is similar to that developed for conversation types [8] and choreography programs [18]. Conversation types deal with the simpler setting of protocols, whereas we handle programs supporting name passing and session interleaving, both nontrivial problems [6,10,18]. The type system of compositional choreographies [18] does not keep information on where the endpoints of connections are actually located as in our hypersequents, which enables choreography extraction in our setting.

8 Discussion

We discuss some aspects of this work and some future extensions.

Process identifiers. In typical choreography calculi, choreography terms identify the processes involved in a communication explicitly using the "Alice and Bob" notation from security protocols [14,9,10,18]. To illustrate the difference, a communication of a fresh name y over a channel x would be typically written as $\mathsf{p} \to \mathsf{q} : x(y)$, where p and q are process identifiers representing respectively the sender and the receiver for the communication. In LCC, process identifiers are implicit: for each communication $\overrightarrow{x(y)}$, there are a sender and a receiver that can be automatically inferred from the connections inside of the hypersequent used for typing the communication. Indeed, the separation between the sender and the receiver processes of each communication becomes evident when applying our projection procedure. Omitting process identifiers is thus just a matter of presentational convenience: a way of retaining them would be to annotate each sequent in a hypersequent with a process identifier (cf. [17]). For example, the rule $\otimes\mathsf{C}$ could be rewritten as follows.

$$\frac{P \triangleright \Psi \mid \Delta_1 \vdash_{\mathsf{r}} y{:}\bullet A \mid \Delta_2 \vdash_{\mathsf{p}} x{:}\bullet B \mid \Delta_3, y{:}\bullet A, x{:}\bullet B \vdash_{\mathsf{q}} T}{\mathsf{p} \to \mathsf{q} : x(\mathsf{r}{:}y);P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash_{\mathsf{p}} x{:}\bullet A \otimes B \mid \Delta_3, x{:}\bullet A \otimes B \vdash_{\mathsf{q}} T}$$

In such new rule, we read the term $\mathsf{p} \to \mathsf{q} : x(\mathsf{r}{:}y);P$ as "process p sends channel y to process q over channel

x and starts the new process r; the system then proceeds as P". The notation $\mathsf{r}:y$ indicates that process p transfers its ownership of the fresh channel y to the newly created process r, as checked by the rule.

Exponentials. Previous work has shown that the exponential fragment of linear logic can be used to type processes that offer reusable (non-linear) channels [5, 7, 26]. We conjecture that our development of LCC can be extended by following a similar direction. For example, consider the cut rule for exponentials found in π-DILL [7] (reformulated in our syntax for sequents):

$$\frac{P \triangleright \Gamma; \cdot \vdash y : A \qquad Q \triangleright \Gamma, u : A; \Delta \vdash T}{(\boldsymbol{\nu} u)\,(!u(y); P \mid Q) \triangleright \Gamma; \Delta \vdash T}\ \mathsf{Cut!}$$

In order to capture exponentials in choreographies, the key aspect will be to understand how the Cut! rule above can be properly split into connection and scope rules, as done for the standard Cut rule in this paper. We leave an exploration of this aspect to future work.

Semantics. Due to the commuting conversions supported by LCC, our proof theory includes more term equivalences than those defined in the standard π-calculus. For example, the equivalence [Conn / $\multimap$ L/L/R] in Fig. 7 allows to move a parallel process under a prefix. This kind of commuting conversions arise naturally from the correspondence with linear logic [26]. However, the extra equivalences do not produce any new reductions in well-typed systems, and are therefore redundant [19].

Multiparty Sessions. Another interesting extension to our work would be to generalise the proof theory of LCC to multiparty sessions, i.e., sessions with more than two participants. This would be in line with the development of compositional choreographies [18], where the added benefit of using LCC would be to obtain an extraction procedure in a multiparty context. We conjecture that the first step to achieve this goal would be to generalise our Conn and Scope rules to multiparty versions where multiple connections are respectively formed and scoped.

Acknowledgements We thank the anonymous reviewers for their helpful comments on this paper. Montesi was supported by the Danish Council for Independent Research (Technology and Production), grant no. DFF–4005-00304, and by the Open Data Framework project at the University of Southern Denmark. This publication was made possible in part by the DemTech grant 10-092309 from the Danish Council for Strategic Research, Program Commission on Strategic Growth Technologies and in part by NPRP Grant #7-1393-5-209 from the Qatar National Research Fund (a member of Qatar Foundation). The statements made herein are solely the responsibility of the authors.

References

1. Business Process Model and Notation. http://www.omg.org/spec/BPMN/2.0/
2. Avron, A.: Hypersequents, logical consequence and intermediate logics for concurrency. Ann. Math. Artif. Intell. **4**, 225–248 (1991)
3. Basu, S., Bultan, T.: Choreography conformance via synchronizability. In: WWW, pp. 795–804 (2011)
4. Basu, S., Bultan, T., Ouederni, M.: Deciding choreography realizability. In: POPL, pp. 191–202 (2012)
5. Bellin, G., Scott, P.J.: On the pi-calculus and linear logic. Theor. Comput. Sci. **135**(1), 11–65 (1994)
6. Bettini, L., Coppo, M., D'Antoni, L., Luca, M.D., Dezani-Ciancaglini, M., Yoshida, N.: Global progress in dynamically interleaved multiparty sessions. In: CONCUR, *LNCS*, vol. 5201, pp. 418–433. Springer (2008)
7. Caires, L., Pfenning, F.: Session types as intuitionistic linear propositions. In: CONCUR, pp. 222–236 (2010)
8. Caires, L., Vieira, H.T.: Conversation types. In: ESOP'09, *LNCS*, vol. 5502, pp. 285–300. Springer-Verlag, Heidelberg, Germany (2009)
9. Carbone, M., Honda, K., Yoshida, N.: Structured communication-centered programming for web services. ACM TOPLAS **34**(2), 8 (2012)
10. Carbone, M., Montesi, F.: Deadlock-freedom-by-design: multiparty asynchronous global programming. In: POPL, pp. 263–274 (2013)
11. Cruz-Filipe, L., Larsen, K.S., Montesi, F.: The paths to choreography extraction. In: FoSSaCS, LNCS. Springer-Verlag (2017). Accepted for publication.
12. Girard, J.Y.: Linear logic. Theor. Comput. Sci. **50**, 1–102 (1987)
13. Honda, K., Vasconcelos, V., Kubo, M.: Language primitives and type disciplines for structured communication-based programming. In: ESOP'98, *LNCS*, vol. 1381, pp. 22–138. Springer-Verlag, Heidelberg, Germany (1998)
14. Lanese, I., Guidi, C., Montesi, F., Zavattaro, G.: Bridging the gap between interaction- and process-oriented choreographies. In: Proc. of SEFM, pp. 323–332. IEEE (2008)
15. Lange, J., Tuosto, E.: Synthesising choreographies from local session types. In: CONCUR, pp. 225–239 (2012)
16. Lange, J., Tuosto, E., Yoshida, N.: From communicating machines to graphical choreographies. In: Proc. of POPL 2015, pp. 221–232 (2015). URL http://doi.acm.org/10.1145/2676726.2676964
17. Montesi, F.: Choreographic Programming. Ph.D. thesis, IT University of Copenhagen (2013). http://www.fabriziomontesi.com/files/choreographic_programming.pdf
18. Montesi, F., Yoshida, N.: Compositional choreographies. In: CONCUR, pp. 425–439 (2013)
19. Pérez, J.A., Caires, L., Pfenning, F., Toninho, B.: Linear logical relations for session-based concurrency. In: ESOP, pp. 539–558 (2012)
20. Pottinger, G.: Uniform cut-free formulations of T, S4 and S5 (abstract). Journal of Symbolic Logic **48**, 900 (1983)
21. Qiu, Z., Zhao, X., Cai, C., Yang, H.: Towards the theoretical foundation of choreography. In: WWW, pp. 973–982. IEEE (2007)
22. Sangiorgi, D.: pi-calculus, internal mobility, and agent-passing calculi. Theor. Comput. Sci. **167**(1&2), 235–274 (1996)
23. Sangiorgi, D., Walker, D.: The π-calculus: a Theory of Mobile Processes. Cambridge University Press (2001)
24. Toninho, B., Caires, L., Pfenning, F.: Higher-order processes, functions, and sessions: A monadic integration. In: ESOP, pp. 350–369 (2013)
25. W3C WS-CDL Working Group: Web services choreography description language version 1.0. http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/ (2004)
26. Wadler, P.: Propositions as sessions. J. Funct. Program. **24**(2-3), 384–418 (2014). DOI 10.1017/S095679681400001X. URL http://dx.doi.org/10.1017/S095679681400001X