APPLIED CHOREOGRAPHIES

SAVERIO GIALLORENZO, FABRIZIO MONTESI, AND MAURIZIO GABBRIELLI

Università di Bologna, Italy and INRIA, France
e-mail address: saverio.giallorenzo@gmail.com

University of Southern Denmark, Denmark
e-mail address: fmontesi@imada.sdu.dk

Università di Bologna, Italy and INRIA, France
e-mail address: maurizio.gabbrielli@unibo.it

Abstract. Choreographic Programming is a correct-by-construction paradigm where a compilation procedure synthesises deadlock-free, concurrent, and distributed communicating processes from global, declarative descriptions of communications, called choreographies. Previous work used choreographies for the synthesis of programs. Alas, there is no formalisation that provides a chain of correctness from choreographies to their implementations. This problem originates from the gap between existing theoretical models, which abstract communications using channel names (à la CCS/π-calculus), and their implementations, which use low-level mechanisms for message routing. As a solution, we propose the theoretical framework of Applied Choreographies. In the framework, developers write choreographies in a language that follows the standard syntax and name-based communication semantics of previous works. Then, they use a compilation procedure to transform a choreography into a low-level, implementation-adherent calculus of Service-Oriented Computing (SOC). To manage the complexity of the compilation, we divide its formalisation and proof in three stages, respectively dealing with: *a*) the translation of name-based communications into their SOC equivalents (namely, using correlation mechanisms based on message data); *b*) the projection of a choreography into a composition of partial, single-participant choreographies (towards their translation into SOC processes); *c*) the translation of partial choreographies and the distribution of choreography-level state into SOC processes. We provide results of behavioural correspondence for each stage. Thus, given a choreography specification, we guarantee to synthesise its faithful and deadlock-free service-oriented implementation.

1. Introduction

Background. Concurrent, distributed software applications have become a crucial asset of our society: messaging, governance, healthcare, and transportation are just some of the contexts recently revolutionised by distributed applications. A hallmark of those applications is that their global behaviour, usually referred to as *protocol*, emerges from the interaction of programs, also called *endpoints*, that run in parallel and rely on message passing to communicate and coordinate their actions [CD88]. Developers strive to correctly implement

Key words and phrases: Correctness-by-construction, Endpoint Projection, Session Types, Global Types.
This work was partially supported by ...

Preprint submitted to
Logical Methods in Computer Science

separate endpoints that, when put together, will enact the expected protocols. If endpoints fail to follow their protocols, the distributed system can block or misbehave — e.g., due to deadlocks [CES71] or race conditions [NM92]. Ensuring that all endpoints play their respective parts correctly—i.e., that they follow their intended protocols—is difficult due to the inherent non-determinism of distributed programs running in parallel [O'H18].

Since the early days of distributed computing, designers and developers introduced and used tools to describe the order of interactions among the endpoints of a system, like security protocol notation [NS78], Message Sequence Charts [Int96] and UML Sequence Diagrams [OMG04]. The common denominator of these tools is to present a global description of the sequence of messages in the system, an information difficult to infer (due to the complexity of interleaved communications) from the specified behaviours of the endpoints. Even for simple systems with a fixed number of participants, algorithms for extracting this information have exponential complexity [CLM17].

Recognising the usefulness of these approaches, in the early 2000s the W3C assembled a Working Group tasked with the definition of a standard for describing interactions among Web Services. This resulted in the Web Services Choreography Description Language (WS-CDL) [W3C04]. A WS-CDL artefact is a "choreography", which specifies the observable behaviours of all the endpoints involved in the system of interest, formalising from a global viewpoint the ordering conditions and constraints that regulate the exchange of messages.

Example 1. We illustrate the choreographic approach with a representative example. We use the example to also introduce the syntax of choreographies used in the remainder of the paper. The example describes a simple business scenario among a client process `c`, a seller service located at $l_{\mathtt{S}}$ and a bank service located at $l_{\mathtt{B}}$. Locations (l) are abstractions of network addresses, or URIs, which identify where services can be contacted to interact with them.

```
1  start k : c[C] <-> l_S.s[S], l_B.b[B];
2  k:c[C].product -> s[S].buy( x );
3  k:s[S].mk_order( x ) -> b[B].reqPay( order );
4  k:c[C].cc -> b[B].sendCC( cc );
5  if b.confirm_pay( cc, order ){
6    k:b[B] -> c[C].ok(); k:b[B] -> s[S].ok()
7  } else {
8    k:b[B] -> c[C].ko(); k:b[B] -> s[S].ko()
9  }
```

At Line 1, the client `c` asks the seller and the bank services to create two new processes, respectively `s` and `b`. The three processes `c`, `s`, and `b` can communicate over a private multiparty session k, intended as in Multiparty Session Types [CDCYP15]: each process owns a statically-defined *role* in the session, which identifies a message queue that the process uses to receive messages asynchronously. For simplicity, at Line 1, we assign role `C` to process `c`, `S` to `s`, and `B` to `b`. As usual, processes have local states and run concurrently. All communications in the rest of the choreography now take place over session k, as indicated by the prefix "k :" in the other lines. At Line 2, the client `c` invokes operation *buy* of the seller `s` with the name of a *product* it wishes to buy, which the seller stores in its local variable x. At Line 3, the seller uses its internal function `mk_order` to prepare an order (e.g., compute the price of the product) and sends it to the bank on operation $openTx$, for opening a payment transaction. At Line 4, the client sends its credit card information *cc* to the bank on operation *pay*. Then, at Line 5, the bank makes an internal choice on whether

the payment can be performed (with the internal function `close_tx`, which takes the local variables *cc* and *order* as parameters). The bank then notifies the client and the seller of the final outcome, by invoking them both either on operation *ok* or *ko*.

The advantage of choreographies is their clarity: they specify the intended global behaviour of a communicating system unambiguously. For this reason, since the inception of WS-CDL, choreographies have been adopted also in other practical applications, like the Business Process Model and Notation by the Object Management Group [OMG11] and Testable Architecture [JBo13]. In general, choreographies come with the promise of enhancing the correctness of systems, since they equip programmers with precise specifications of the communications that a system should enact. This promise motivated a fruitful line of research in the areas of process calculi and programming languages, which rotates around the question: "*Can we use choreographies to prove that a concurrent program will execute the right communications?*"

Inspired by this question, two development methodologies have emerged based on choreographies. In the first, called Choreographic Programming [Mon13], programs are choreographies as that in our Example 1. The idea is that the choreography defines both the internal computation performed by processes and the communications among them. Then, a correct-by-construction implementation (typically given in terms of a process calculus) can be automatically synthesised [CHY12, CM13]. In the second methodology, choreographies are used to describe protocols, which abstract away from internal computation. The aim of this second methodology is to verify that each process, written manually (in contrast to being automatically synthesised, as in choreographic programming), implements correctly its role in the protocols that it participates in. Multiparty Session Types [HYC16] is representative of this methodology.

Both methodologies are based on the same general idea: for each endpoint described in a choreography, we can *project* a definition of its local behaviour using a procedure known as EndPoint Projection (EPP). In choreographic programming, this yields the local implementation of each endpoint. For multiparty session types, this yields a type, e.g., used to check that a process implements its role in a protocol correctly. The key technical result that one needs to prove then is that the projection yields a set of endpoint terms (programs or types) that, when executed in parallel, implement exactly the communications described in the original choreography. This is typically called the EndPoint Projection Theorem (or EPP Theorem, for short).

The model of Compositional Choreographies [MY13] unifies the two methodologies, with the aim of combining their advantages. In that model, programmers can describe parts of a system as in choreographic programming and other parts as independent process terms. The model uses multiparty session types to check that the composition of the independent process terms with the projections of choreographic programs will behave correctly. What made the unification of the two approaches possible is the strong operational correspondence guaranteed by the EPP.

Motivation. The main application area for choreographies so far is that of Service-Oriented Computing (SOC), as in web services [W3C04] or microservices [DGL$^+$17, New15]. Implementing communications in this setting is non-trivial, since services must be loosely coupled and thus we cannot assume the presence of any particular common middleware. However, in all previous definitions of EPP, both the choreography language and the target language abstract from how real-world frameworks support communications [QZCY07, LGMZ08, CHY12, CM13, CMS17], by modelling message exchange through synchronisations on *names*

(as in CCS and the π-calculus [Mil80, MPW92]). As a consequence, the implementations of choreographic frameworks [Cho16, AIO16, NY14] significantly depart from their respective formalisations [CM13, DGG$^+$15, HYC16] (a common aspect of implementing process calculi, cf. [CLM05, HYH08]). In particular, implementations realise the creation of new channels and message routing with additional data structures and message exchanges [Mon13, DGL$^+$14] that are absent in their formalisations. The specific communication mechanism used in these implementations is message correlation; correlation is the reference communication support in SOC, and is supported by mainstream technologies (e.g., WS-BPEL [OAS07], Java/JMS, C#/.NET). The gap between formalisations and implementations can compromise the correctness guarantee of choreographies. Thus we ask:

How can we formalise the implementation of communications in choreographies?

A satisfactory answer should preserve the correctness guarantees down to the level of how communications are concretely implemented. Defining such a model is challenging: we wish to retain the typical clarity of choreography languages, yet we need enough details to (formally) reason on how communications are realised at the lower level. Ideally, the complexity of implementing communications should not leak into the choreographic programming model exposed to programmers, and should just be a "detail" that we can forget about with confidence. Building this confidence is the main aim of this article.

Contributions and Outline. We tackle our question by developing the framework of Applied Choreographies. Our framework consists of three calculi, which enjoy a tight series of correspondences.

The first calculus, called Frontend Choreographies (FC), is meant to be the programming model exposed to programmers and is presented in § 2. FC is a straightforward reformulation of the standard calculus of Compositional Choreographies [MY13], which we adopt to show that our approach applies to both the methodology of choreographic programming and that of multiparty session types. In particular, communications are based on name synchronisation, as in standard process calculi.

The second calculus, called Backend Choreographies (BC), has the same syntax of FC but a different semantics: instead of using abstract name synchronisation, BC models and keeps track of the data structures needed to implement concrete correlation-based communications (§ 4.1). While more involved than FC, BC is agnostic wrt the specific technology used for correlation.

The third is a process calculus of distributed executable code, based on a standard formal model for Service-Oriented Computing [MC11], called Dynamic Correlation Calculus (DCC) (§ 5). DCC models both data distribution and how concrete communications are implemented. Given its low-level scope, DCC does not capture all the abstraction of choreographies.

Our main contribution is the definition of a behaviour-preserving compiler from Frontend Choreographies to DCC distributed services. The compiler uses Backend Choreographies as intermediate representation. This is the first correctness result of an end-to-end translation from standard choreographies to programs based on a real-world communication mechanism.

More concretely, our compiler includes two transitional stages (FC-to-BC and EPP) toward the Compilation:

FC-to-BC: generates the data structures needed to support the execution of a source FC program using message correlation (§ 4.3). Essentially, we obtain a Backend choreography which is operationally correspondent to its source Frontend;

EPP: transforms a choreography that describes the behaviours of many participants into a set of modules, called *endpoint choreographies*, each describing the behaviour of a single participant. More precisely, the procedure is an endomorphism (§ 6.1) that transforms a source choreography—whether FC or BC does not matter, since they share the same syntax—into a set of (endpoint) choreographies whose syntax is restricted to only partial actions (i.e., belonging to one of the two ends of a communication);

Compilation: takes in the BC data structures obtained from the FC-to-BC conversion and the endpoint choreographies obtained from the EPP transformation and synthesises a correct implementation of the source FC program as a distributed system of DCC services.

Starting from FC proves that our development is adequate. Programmers can use high-level programming primitives and semantics as found in previous works on choreographies—with state-of-the-art features like asynchronous communications [CM13] and modular development [MY13]—while our compilation procedure tackles the heavy-lifting of producing correct service-oriented implementations.

We conclude our proposal discussing related and future work in § 7 and report in the Appendix auxiliary technical material and the proofs of our results.

This paper integrates and extends material from [GMG18], which presented the main ideas behind the Applied Choreographies framework. The extensions in this work include: *a*) full formal definitions (syntax and semantics of all three calculi); *b*) detailed examples for each main component of the work (the three calculi, the typing system, the three stages of compilation) to illustrate their relevant characteristics and features; *c*) full proofs of the formal properties guaranteed by the framework (in appendix B, to avoid breaking the flow of the reader with details of the technical development). Besides the previous points, this version contains an extended, revised, and refined presentation of all the contents presented in [GMG18]

2. Frontend Choreographies

We present Frontend Choreographies (FC), the language model intended for programmers.

Before giving the formal syntax of FC, we first describe the intuition behind its key components. The following table displays the symbols that we are going to use, along with their names and domains.

Name	**Symbols**	**Domain**
Choreographies	C_1, C_2	$-$
Processes	p, q	$\mathcal{P}$
Operations	o_1, o_2	$\mathcal{O}$
Variables	x, y	Var
Sessions	k_1, k_2	$\mathcal{K}$
Roles	$\mathtt{A}, \mathtt{B}$	$\mathcal{A}$
Locations	l_1, l_2	$\mathcal{L}$

FC programs are choreographies, as in Example 1, denoted by C. A choreography describes the behaviour of some processes. Processes, denoted $\mathsf{p}, \mathsf{q} \in \mathcal{P}$, are intended as usual: they are independent execution units running concurrently and equipped with local variables, denoted $x \in \mathit{Var}$.

Processes communicate by exchanging messages. A message consists of two elements: *i*) a payload, representing the data exchanged between two processes; and *ii*) an operation, which is a label used by the receiver to determine what it should do with the message—in object-oriented programming, these labels are called method names [Pie02]; in service-oriented computing, labels are typically called operations as in here. Operations are denoted $o \in \mathcal{O}$.

Message exchanges happen through a session, denoted $k \in \mathcal{K}$, which acts as a communication channel. Sessions in FC are behaviourally typed [HLV+16]. Intuitively, a session is an instantiation of a protocol, where each process is responsible for implementing the actions of a role defined in the protocol. We denote roles with $\mathtt{A}, \mathtt{B} \in \mathcal{A}$.

A process can create new processes and sessions at runtime by invoking service processes (services for short). Services are always available at fixed locations, denoted $l \in \mathcal{L}$, meaning that they can be used multiple times (in process calculus terms, they act as replicated processes [SW01]).

FC supports modular development by allowing choreographies, say C and C', to be composed in parallel, written $C \mid C'$. A parallel composition of choreographies is also a choreography, which can thus be used in further parallel compositions. Composing two choreographies in parallel allows the processes in the two choreographies to interact over shared location and session names.

We distinguish between two kinds of statements inside of a choreography: complete and partial actions. A complete action is internal to the system defined by the choreography, and thus does not have any external dependency. By contrast, a partial action defines the behaviour of some processes that need to interact with another choreography in order to be executed. Therefore, a choreography containing partial actions needs to be composed with other choreographies that provide compatible partial actions.

To exemplify the distinction between complete and partial actions, we consider the case of a single communication between two processes.

Complete interaction	*Composed partial actions*
$k : \mathtt{c}[\mathtt{C}].\mathit{product} \longrightarrow \mathtt{s}[\mathtt{S}].\mathit{buy}(\ x\ )$	$k : \mathtt{c}[\mathtt{C}].\mathit{product} \longrightarrow \mathtt{S}.\mathit{buy}$ $\mid$ $k : \mathtt{C} \longrightarrow \mathtt{s}[\mathtt{S}].\mathit{buy}(\ x\ )$

Above, on the left we have the communication statement as seen at Line 2 of Example 1. This is a complete action: it defines exactly all the processes that should interact (`c` and `s`). On the right, we implement the same action as the parallel composition of two choreographies with partial actions: a send action by process `c` to role `S` over session k (left of the parallel) and a reception by process `s` from a role `C` (right of the parallel) over the same session k. More specifically, we read the send action (top of the parallel) as "process `c` sends a message as role `C` with payload *product* for operation *buy* to the process playing role `S` on session k". Dually, we read the receive action (bottom of the parallel) as "process `s` receives a message for role `S` and operation *buy* over session k and stores the payload in variable x". The compatible roles, session, and operation used in the two partial actions make them compliant. Thus, the choreography on the left is operationally equivalent to the one on the right. Observe that partial actions do not mention the name of the process on the other end—for example, the send action by process `c` does not specify that it wishes to

$$
\begin{array}{rlll}
C ::= & \eta; C & (\mathit{seq}) & \mid \quad \texttt{if p}.e\ \{C_1\}\ \texttt{else}\ \{C_2\} \quad (\mathit{cond}) \\
\mid & C_1 \mid C_2 & (\mathit{par}) & \mid \quad k : \texttt{A} \longrightarrow \texttt{q[B]}.\{o_i(x_i); C_i\}_{i \in I} \quad (\mathit{recv}) \\
\mid & \mathbf{0} & (\mathit{inact}) & \mid \quad \texttt{def}\ X = C'\ \texttt{in}\ C \quad (\mathit{rec}) \\
\mid & X & (\mathit{call}) & \mid \quad \texttt{acc}\ k : \widetilde{l.\texttt{q[B]}}; C \quad (\mathit{accept}) \\
\\
\eta ::= & k : \texttt{p[A]}.e \longrightarrow \texttt{B}.o & (\mathit{send}) & \mid \quad \texttt{start}\ k : \texttt{p[A]} \leftrightarrow \widetilde{l.\texttt{q[B]}} \quad (\mathit{start}) \\
\mid & k : \texttt{p[A]}.e \longrightarrow \texttt{q[B]}.o(x) & (\mathit{com}) & \mid \quad \texttt{req}\ k : \texttt{p[A]} \leftrightarrow \widetilde{l.\texttt{B}} \quad (\mathit{req})
\end{array}
$$

Figure 1: Frontend Choreographies — syntax.

communicate with process s precisely. This mechanism supports some information hiding: a partial action in a choreography can interact with partial actions in other choreographies independently of the process names used in the latter. Expressions and variables used by senders and receivers are also kept local to statements that define local actions.

2.1. **Syntax of Frontend Choreographies.** We present the formal syntax of FC, displayed in Figure 1. In the remainder, we use the symbol $\sim$ over an element to indicate an ordered set of elements of its kind, e.g., $\tilde{\texttt{p}}$ indicates an ordered set of processes $\texttt{p}_1, \ldots, \texttt{p}_n$.

Complete Actions. In term (*start*), process p creates a new session k together with processes $\tilde{\texttt{q}}$ ($\tilde{\texttt{q}}$ is assumed non-empty). Process p, called *active process*, is already running, whereas each process q in $\widetilde{l.\texttt{q}}$, called *service process*, is dynamically created at the respective service location l. Each process is annotated with the role it plays in the new session k. Term (*com*) reads: on session k, process p sends to process q a message for its operation o; the message carries the evaluation of expression e on the local state of p, whilst x is the variable where q will store the content of the message. We leave the guest language for writing local expressions (e) unspecified, and assume that it consists of terms for accessing local variables (x) and implementing standard computations based on those (e.g., arithmetics).

Partial Actions. A choreography can use partial actions to interact with other choreographies composed in parallel. Therefore, partial actions describe the behaviour of processes that wish to synchronise with "external" participants. Concretely, these external participants will be processes and/or services whose behaviours are defined in other choreographies composed in parallel. In term (*req*), process p requests some external services, respectively located at $\tilde{l}$, to create a new session k and some new external processes. Role annotations follow the same intuition as in term (*start*): in the new session k, p will play A and each new external process $\texttt{q}_i$ will play the respective role $\texttt{B}_i$.

Term (*acc*) is the dual of (*req*) and defines a choreography module that provides the implementation of some service processes. We assume that (*acc*) terms are always at the top level, to capture that choreography modules are always available. By top level, we mean that the term is not preceded by another term in a sequential composition (*seq*).

In term (*send*), process p sends a message to an external process that plays B in session k. Dually, in term (*recv*), process q receives a message for one of the operations o_i from an external process playing role A in session k, and then proceeds with the corresponding continuation. In the remainder, we omit curly brackets in (*recv*) terms when they have only one operation, i.e., $k : \texttt{A} \longrightarrow \texttt{q[B]}.o(x); C$ is an abbreviation of $k : \texttt{A} \longrightarrow \texttt{q[B]}.\{o(x); C\}$.

```
1  start k : c[C] ⟷ l_S.s[S], l_B.b[B];
2  k:c[C].buyReq ⟶ s[S].buy( x );
3  req k' : s[S] ⟷ l_D.D;
4  k':s[S].mk_shipping( x ) ⟶ D.quoteShipping;
5  k':D ⟶ s[S].shippingCosts( y );
6  k:s[S].mk_order( x, y ) ⟶ b[B].reqPay( order );
7  k:c[C].cc ⟶ b[B].sendPay( cc );
8  if b.confirm_pay( cc, order ){
9    k:b[B] ⟶ c[C].ok(); k:b[B] ⟶ q[S].ok();
10   k':s[S] ⟶ D.sendShipping
11 } else {
12   k:b[B] ⟶ c[C].ko(); k:b[B] ⟶ q[S].ko();
13   k':s[S] ⟶ D.abortShipping
14 }
```

Figure 2: Choreography C_1, extension of Example 1.

```
1  acc k' : l_D.d[D];
2  k':S ⟶ d[D].quoteShipping( pkg );
3  k':d[D].quote( pkg ) ⟶ S.shippingCosts;
4  k':S ⟶ d[D].{
5    sendShipping(),
6    abortShipping()  }
```

Figure 3: Choreography C_2, compliant choreography to Figure 2.

Other Terms. Term (*seq*) is sequential composition. In a conditional (*cond*), process $\mathtt{p}$ evaluates a condition e in its local state to choose between the continuations C_1 and C_2. Term (*par*) is standard parallel composition, which allows partial actions in two choreographies C_1 and C_2 to interact. Respectively, terms (*def*), (*call*), and (*inact*) model the definition of recursive procedures, procedure calls, and inaction.

Some terms bind identifiers in continuations—the choreography that follows them in a sequential composition. In terms (*start*) and (*acc*), the session identifier k and the process identifiers $\tilde{\mathtt{q}}$ are bound (as they are freshly created). In terms (*com*) and (*recv*), the variables used by the receiver to store the message are bound (x and all the x_i, respectively). In term (*req*), the session identifier k is bound. Finally, in term (*def*), the procedure identifier X is bound. In the remainder, we omit $\mathbf{0}$ or irrelevant variables (e.g., in communications with empty messages). Terms (*com*), (*send*), and (*recv*) include role annotations only for clarity reasons; roles in such terms can be inferred, as shown in [Mon13].

Example 2. In Figure 2, we extend (in blue) the behaviour of the seller of Example 1 to use an external module. In the updated code, the seller contacts an external service for the delivery of the product: the seller receives a request *buyReq* from the buyer. The request contains the wanted product and the delivery address (Line 2). Next, the seller creates a new session k' with an external delivery process (Line 3) and sends to the latter the shipping information of the product, e.g., the origin and destination addresses (Line 4). At Line 5, the seller receives the shipping costs, which it adds to the costs of the order at the bank (Line 6). At Lines 11 and 14, the seller notifies the delivery process if it shall ship the product or not. Let us call C_1 the code above. We report in Figure 3 the module C_2 of a compliant delivery service for C_1. We obtain a working system by composing the two choreographies in parallel: $C_1 \mid C_2$.

2.2. **Semantics of Frontend Choreographies.** We give an operational semantics for FC in terms of reductions of the form $D, C \to D', C'$, where D is a deployment. Deployments

keep track of: the local states of processes (the values of their local variables); and the messages in transit in sessions, which we use to model asynchronous communications. In the following, we formalise our notion of deployment and we present our reduction semantics.

2.2.1. *Frontend Deployments.* In the remainder, we adopt as a convention, when indicating a Frontend Choreographies program, its shortened form "Frontend choreography" (lowercase c) or simply "choreography" when the context clearly associates it to Frontend Choreographies. We also use the shortened form "Frontend deployment" to indicate a Frontend Choreographies deployment.

Each pair of roles in a session has two dedicated asynchronous message queues that they can use to exchange messages, one for each direction. Formally, let $\mathcal{Q} = \mathcal{K} \times \mathcal{A} \times \mathcal{A}$ be the set of all *queue identifiers*; we write $k[\mathtt{A}\rangle\mathtt{B}] \in \mathcal{Q}$ to identify the queue from role `A` to role `B` in session k.

A *deployment* D is an overloaded partial function defined by cases as the sum of two partial functions, $f_s : \mathcal{P} \rightharpoonup \mathit{Var} \rightharpoonup \mathit{Val}$ and $f_q : \mathcal{Q} \rightharpoonup \mathit{Seq}(\mathcal{O} \times \mathit{Val})$ (their domains and co-domains are disjoint):

$$D(z) = \begin{cases} f_s(z) & \text{if } z \in \mathcal{P} \\ f_q(z) & \text{if } z \in \mathcal{Q} \end{cases}$$

Function f_s maps a process p to its state. A state is a partial function from variables $x, y \in \mathit{Var}$ to values $v \in \mathit{Val}$. Function f_q stores the queues used in sessions. Each queue is a sequence of messages $\tilde{m} = m_1 :: \ldots :: m_n \mid \varepsilon$ (ε is the empty queue), where each message $m = (o, v) \in \mathcal{O} \times \mathit{Val}$ contains the operation o for which the message is intended and the payload v.

Deployments are a runtime concept: programmers do not need to define them, just as they normally do not explicitly give an initial state for their programs in other language models. Formally, we assume that choreographies without free session names start execution with a *default deployment* that contains empty process states. Let $\mathbf{fp}(C)$ return the set of free process names in C. Then, we formally define a default deployment as follows.

Definition 1 (Default Deployment)**.** Let C be a choreography without free session names. Then, the default deployment D for C is defined as the function that maps all free process names in C to empty states (we write $\varnothing$ for the empty partial function from *Var* to *Val*):

$$D = \left[\mathsf{p} \mapsto \varnothing \mid \mathsf{p} \in \mathbf{fp}(C)\right]$$

Intuitively, D is a default deployment for a choreography without free session names C if *i*) D is defined for all and only the processes that appear free in C and *ii*) the state of these processes is empty.

2.2.2. *Frontend Deployment Transitions.* In our semantics, choreographic actions have effects on the state of a system — deployments change during execution. At the same time, a deployment also determines which choreographic actions can be performed. For example, a communication from role `A` to role `B` over session k requires a queue $k[\mathtt{A}\rangle\mathtt{B}]$ to exist in the deployment of the system.

$$\frac{D' = D\big[\mathtt{q} \mapsto \varnothing \mid \mathtt{q} \in \tilde{\mathtt{q}}\big]\big[k[\mathtt{C}\rangle\mathtt{E}] \mapsto \varepsilon \mid \{\mathtt{C},\mathtt{E}\} \subseteq \{\mathtt{A},\tilde{\mathtt{B}}\}\big]}{D, \mathtt{start}\ k : \mathtt{p}[\mathtt{A}] \leftrightarrow \widetilde{l.\mathtt{q}[\mathtt{B}]} \blacktriangleright D'}\ {\scriptstyle [\mathsf{D}|_{\mathsf{Start}}]}$$

$$\frac{v = \mathbf{eval}(e, D(\mathtt{p})) \qquad D(k[\mathtt{A}\rangle\mathtt{B}]) = \tilde{m}}{D, k : \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o \blacktriangleright D\big[k[\mathtt{A}\rangle\mathtt{B}] \mapsto \tilde{m} :: (o, v)\big]}\ {\scriptstyle [\mathsf{D}|_{\mathsf{Send}}]}$$

$$\frac{D(k[\mathtt{A}\rangle\mathtt{B}]) = (o, v) :: \tilde{m}}{D, k : \mathtt{A} \longrightarrow \mathtt{q}[\mathtt{B}].o(x) \blacktriangleright D\big[k[\mathtt{A}\rangle\mathtt{B}] \mapsto \tilde{m}\big]\big[\mathtt{q} \mapsto D(\mathtt{q})[x \mapsto v]\big]}\ {\scriptstyle [\mathsf{D}|_{\mathsf{Recv}}]}$$

Figure 4: Frontend Choreographies — Deployment transitions.

We formalise the notion of which choreographic actions are allowed by a deployment and their effects using transitions of the form $D, \delta \blacktriangleright D'$, read "the deployment D allows for the execution of δ and becomes D' as result". The following grammar defines δ actions.

$$\begin{array}{rcll} \delta & ::= & \mathtt{start}\ k : \mathtt{p}[\mathtt{A}] \leftrightarrow \widetilde{l.\mathtt{q}[\mathtt{B}]} & \textit{(session start)} \\ & \mid & k : \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o & \textit{(send in session)} \\ & \mid & k : \mathtt{A} \longrightarrow \mathtt{q}[\mathtt{B}].o(x) & \textit{(receive in session)} \end{array}$$

The rules defining $D, \delta \blacktriangleright D'$ are given in Figure 4.

Rule $[\mathsf{D}|_{\mathsf{Start}}]$ states that the creation of a new session k between an existing process $\mathtt{p}$ and new processes $\tilde{\mathtt{q}}$ results in updating the deployment with: a new (empty) state for each of the new processes $\mathtt{q}$ in $\tilde{\mathtt{q}}$ ($\big[\mathtt{q} \mapsto \varnothing \mid \mathtt{q} \in \tilde{\mathtt{q}}\big]$); and a new (empty) queue between each pair of distinct roles in the session ($\big[k[\mathtt{C}\rangle\mathtt{E}] \mapsto \varepsilon \mid \{\mathtt{C},\mathtt{E}\} \subseteq \{\mathtt{A},\tilde{\mathtt{B}}\}\big]$).

Rule $[\mathsf{D}|_{\mathsf{Send}}]$ models the effect of a send action. In the first premise, we use the auxiliary function **eval** to evaluate the local expression e in the state of process $\mathtt{p}$, obtaining the value v to use as message payload. Then, in the conclusion, we add a message (o, v) — where o is the operation used to label the message — to the tail of the queue $k[\mathtt{A}\rangle\mathtt{B}]$, i.e., the queue expected to contain messages sent by $\mathtt{A}$ to $\mathtt{B}$ in session k. We assume that function **eval** always terminates — in practice, this can be obtained by using timeouts.

Rule $[\mathsf{D}|_{\mathsf{Recv}}]$ models the effect of a reception. First, in the premise, we look up the head of the message queue between sender and receiver, i.e., (o, v). Then, in the conclusion, we remove the message from the queue ($\big[k[\mathtt{A}\rangle\mathtt{B}] \mapsto \tilde{m}\big]$) and update the state of the receiver at the variable used to store the message ($\big[\mathtt{q} \mapsto D(\mathtt{q})[x \mapsto v]\big]$).

2.2.3. *Reductions.* Using deployment transitions, we can now define the rules for reductions $D, C \to D', C'$. We call a configuration D, C a *running choreography*. The reduction relation $\to$ for FC is the smallest relation closed under the rules given in Figure 5.

Rule $[\mathsf{C}|_{\mathsf{Start}}]$ creates a new session, by ensuring that both the new session name k' and new processes $\tilde{\mathtt{r}}$ are fresh wrt D ($D \# k', \tilde{\mathtt{r}}$). We use the fresh names in the continuation C, by using a standard substitution $C[k'/k][\tilde{\mathtt{r}}/\tilde{\mathtt{q}}]$.

Rule $[\mathsf{C}|_{\mathsf{Send}}]$ reduces a send action, if the deployment permits it: $D, k : \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o \blacktriangleright D'$.

$$\frac{D\#k',\tilde{\mathtt{r}} \quad \delta = \mathtt{start}\ k' : \mathtt{p}[\mathtt{A}] \leftrightarrows \widetilde{l.\mathtt{q}[\mathtt{B}]} \quad D,\delta \blacktriangleright D'}{D,\ \mathtt{start}\ k : \mathtt{p}[\mathtt{A}] \leftrightarrows \widetilde{l.\mathtt{q}[\mathtt{B}]}; C \quad \to \quad D',\ C[k'/k][\tilde{\mathtt{r}}/\tilde{\mathtt{q}}]}\ \lfloor^{\mathsf{C}}{}_{\mathsf{Start}}\rceil$$

$$\frac{\eta = k : \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o \quad D,\eta \blacktriangleright D'}{D,\ \eta; C \quad \to \quad D',\ C}\ \lfloor^{\mathsf{C}}{}_{\mathsf{Send}}\rceil$$

$$\frac{j \in I \quad D, k : \mathtt{A} \longrightarrow \mathtt{q}[\mathtt{B}].o_j(x_j) \blacktriangleright D'}{D,\ k : \mathtt{A} \longrightarrow \mathtt{q}[\mathtt{B}].\{o_i(x_i); C_i\}_{i\in I} \quad \to \quad D',\ C_j}\ \lfloor^{\mathsf{C}}{}_{\mathsf{Recv}}\rceil$$

$$\frac{i = 1 \text{ if } \mathbf{eval}(\ e, D(\mathtt{p})\ ) = \mathtt{true}, i = 2 \text{ otherwise}}{D,\ \mathtt{if}\ \mathtt{p}.e\ \{C_1\}\ \mathtt{else}\ \{C_2\} \quad \to \quad D,\ C_i}\ \lfloor^{\mathsf{C}}{}_{\mathsf{Cond}}\rceil$$

$$\frac{D, C_1 \quad \to \quad D', C_1'}{D, \mathtt{def}\ X = C_2\ \mathtt{in}\ C_1 \quad \to \quad D', \mathtt{def}\ X = C_2\ \mathtt{in}\ C_1'}\ \lfloor^{\mathsf{C}}{}_{\mathsf{Ctx}}\rceil$$

$$\frac{\mathcal{R} \in \{\equiv, \simeq_{\mathtt{C}}\} \quad C\,\mathcal{R}\,C_1 \quad D, C_1 \to D', C_1' \quad C_1'\,\mathcal{R}\,C'}{D, C \quad \to \quad D', C'}\ \lfloor^{\mathsf{C}}{}_{\mathsf{Eq}}\rceil$$

$$\frac{D, C_1 \quad \to \quad D', C_1'}{D, C_1 \mid C_2 \quad \to \quad D', C_1' \mid C_2}\ \lfloor^{\mathsf{C}}{}_{\mathsf{Par}}\rceil$$

$$\frac{\begin{array}{c} i \in \{1,\ldots,n\} \qquad D\#k',\tilde{\mathtt{r}} \qquad \{\widetilde{l.\mathtt{B}}\} = \biguplus_i \{\widetilde{l_i.\mathtt{B}_i}\}_i \qquad \{\tilde{\mathtt{r}}\} = \bigcup_i \{\tilde{\mathtt{r}}_i\} \\ \delta = \mathtt{start}\ k' : \mathtt{p}[\mathtt{A}] \leftrightarrows \widetilde{l_1.\mathtt{r}_1[\mathtt{B}_1]}, \ldots, \widetilde{l_n.\mathtt{r}_n[\mathtt{B}_n]} \qquad D,\delta \blacktriangleright D' \end{array}}{\begin{array}{l} D, \mathtt{req}\ k : \mathtt{p}[\mathtt{A}] \leftrightarrows \widetilde{l.\mathtt{B}}; C \mid \prod_i \left(\mathtt{acc}\ k : \widetilde{l_i.\mathtt{q}_i[\mathtt{B}_i]}; C_i\right) \quad \to \\ \qquad D',\ C[k'/k] \mid \prod_i \left(\ C_i[k'/k][\tilde{\mathtt{r}}_i/\tilde{\mathtt{q}}_i]\ \right) \mid \prod_i \left(\mathtt{acc}\ k : \widetilde{l_i.\mathtt{q}_i[\mathtt{B}_i]}; C_i\right) \end{array}}\ \lfloor^{\mathsf{C}}{}_{\mathsf{PStart}}\rceil$$

Figure 5: Frontend Choreographies—semantics.

$$\mathtt{def}\ X = C'\ \mathtt{in}\ \mathbf{0} \equiv_{\mathtt{C}} \mathbf{0} \qquad C \mid C' \equiv_{\mathtt{C}} C' \mid C \qquad (C_1 \mid C_2) \mid C_3 \equiv_{\mathtt{C}} C_1 \mid (C_2 \mid C_3)$$

$$\mathtt{def}\ X = C'\ \mathtt{in}\ C[X] \equiv_{\mathtt{C}} \mathtt{def}\ X = C'\ \mathtt{in}\ C[C']$$

$$k : \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{q}[\mathtt{B}].o(x); C \equiv_{\mathtt{C}} k : \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o; k : \mathtt{A} \longrightarrow \mathtt{q}[\mathtt{B}].\{o(x); C\}$$

Figure 6: Frontend Choreographies — structural congruence $\equiv_{\mathtt{C}}$

Rule $\lfloor^{\mathsf{C}}{}_{\mathsf{Recv}}\rceil$ reduces a message reception, if the deployment permits the reception of a message on one of the branches in the receive term ($j \in I$). Recalling the corresponding rule $\lfloor^{\mathsf{D}}{}_{\mathsf{Recv}}\rceil$, this can happen only if the deployment D has a message for operation o_j in the queue $k[\mathtt{A}\rangle\mathtt{B}]$. **Rule** $\lfloor^{\mathsf{C}}{}_{\mathsf{Eq}}\rceil$ closes $\to$ under the congruences $\equiv_{\mathtt{C}}$ and $\simeq_{\mathtt{C}}$. Structural congruence $\equiv_{\mathtt{C}}$, reported in Figure 6, is the smallest congruence supporting α-conversion, recursion unfolding, and commutativity and associativity of parallel composition. The swap relation $\simeq_{\mathtt{C}}$, reported in Figure 7, is the smallest congruence able to exchange the order of non-interfering concurrent actions. For example, provided **pn** returns the set of process names, Rule $\lfloor^{\mathsf{CS}}{}_{\mathsf{EtaEta}}\rceil$ swaps two communications respectively enacted by completely disjoint processes.

$$\frac{\mathbf{pn}(\eta) \cap \mathbf{pn}(\eta') = \varnothing}{\eta;\eta' \quad \simeq_{\mathsf{C}} \quad \eta';\eta} \; \lfloor^{\mathsf{CS}}|_{\mathsf{EtaEta}}\rceil \qquad \frac{\mathsf{p} \notin \mathbf{pn}(\eta)}{\begin{array}{l}\texttt{if}\ \mathsf{p}.e\ \{\eta; C_1\}\ \texttt{else}\ \{\eta; C_2\} \\ \quad \simeq_{\mathsf{C}} \quad \eta;\texttt{if}\ \mathsf{p}.e\ \{C_1\}\ \texttt{else}\ \{C_2\}\end{array}} \; \lfloor^{\mathsf{CS}}|_{\mathsf{EtaCnd}}\rceil$$

$$\frac{\mathsf{q} \notin \mathbf{pn}(\eta)}{k:\mathtt{A} \longrightarrow \mathsf{q}[\mathtt{B}].\{o_i(x_i);\eta;C_i\}_{i\in I} \quad \simeq_{\mathsf{C}} \quad \eta;k:\mathtt{A} \longrightarrow \mathsf{q}[\mathtt{B}].\{o_i(x_i);C_i\}_{i\in I}} \; \lfloor^{\mathsf{CS}}|_{\mathsf{EtaRcv}}\rceil$$

$$\frac{\mathsf{p} \neq \mathsf{q}}{\begin{array}{l}k:\mathtt{A} \longrightarrow \mathsf{p}[\mathtt{B}].\{o_i(x_i);k':\mathtt{C} \longrightarrow \mathsf{q}[\mathtt{D}].\{o'_{ij}(x'_{ij});C_{ij}\}_{j\in J}\}_{i\in I} \\ \quad \simeq_{\mathsf{C}} \quad k':\mathtt{C} \longrightarrow \mathsf{q}[\mathtt{D}].\{o'_j(x'_j);k:\mathtt{A} \longrightarrow \mathsf{p}[\mathtt{B}].\{o_{ij}(x_{ij});C_{ij}\}_{i\in I}\}_{j\in J}\end{array}} \; \lfloor^{\mathsf{CS}}|_{\mathsf{RcvRcv}}\rceil$$

$$\frac{\mathsf{p} \neq \mathsf{q}}{\begin{array}{l}\texttt{if}\ \mathsf{p}.e\ \{\texttt{if}\ \mathsf{q}.e'\ \{C_1\}\ \texttt{else}\ \{C_2\}\}\ \texttt{else}\ \{\texttt{if}\ \mathsf{q}.e'\ \{C'_1\}\ \texttt{else}\ \{C'_2\}\} \\ \quad \simeq_{\mathsf{C}} \quad \texttt{if}\ \mathsf{q}.e'\ \{\texttt{if}\ \mathsf{p}.e\ \{C_1\}\ \texttt{else}\ \{C'_1\}\}\ \texttt{else}\ \{\texttt{if}\ \mathsf{p}.e\ \{C_2\}\ \texttt{else}\ \{C\prime_2\}\}\end{array}} \; \lfloor^{\mathsf{CS}}|_{\mathsf{CndCnd}}\rceil$$

$$\frac{\mathsf{p} \neq \mathsf{q}}{\begin{array}{l}k:\mathtt{A} \longrightarrow \mathsf{p}[\mathtt{B}].\{o_i(x_i);\texttt{if}\ \mathsf{q}.e\ \{C_{i1}\}\ \texttt{else}\ \{C_{i2}\}\}_{i\in I} \\ \quad \simeq_{\mathsf{C}} \quad \texttt{if}\ \mathsf{q}.e\ \{k:\mathtt{A} \longrightarrow \mathsf{p}[\mathtt{B}].\{o_i(x_i);C_{i1}\}_{i\in I}\}\ \texttt{else}\ \{k:\mathtt{A} \longrightarrow \mathsf{p}[\mathtt{B}].\{o_i(x_i);C_{i2}\}_{i\in I}\}\end{array}} \; \lfloor^{\mathsf{CS}}|_{\mathsf{RcvCnd}}\rceil$$

Figure 7: Frontend Choreographies — swap relation $\simeq_{\mathsf{C}}$.

Rule $\lfloor^{\mathsf{C}}|_{\mathsf{Eq}}\rceil$ also enables the reduction of complete communications on (*com*) terms—see the last equivalence in Figure 6, which unfolds a complete communication term into the two corresponding send and receive terms.

Rule $\lfloor^{\mathsf{C}}|_{\mathsf{PStart}}\rceil$ starts a new session by synchronising a partial choreography that requests to start a session with other choreographies that can accept the request. The premise of the rule $\{\widetilde{l.\mathtt{B}}\} = \biguplus_i \{\widetilde{l_i.\mathtt{B}_i}\}_i$, where $\biguplus$ indicates the disjoint union of the list of located roles, requires that in the accepting choreographies the list of locations and their supported roles match the corresponding list of the request. The rest of the rule is similar to $\lfloor^{\mathsf{C}}|_{\mathsf{Start}}\rceil$. Here it is convenient that deployment transitions are specified by a separate set of rules, since the effect of starting a session using partial actions is equivalent to that of using a complete start term. The choreographies accepting the request remain available for subsequent reuses.

Finally, rules $\lfloor^{\mathsf{C}}|_{\mathsf{Cond}}\rceil$,$\lfloor^{\mathsf{C}}|_{\mathsf{Ctx}}\rceil$, and $\lfloor^{\mathsf{C}}|_{\mathsf{Par}}\rceil$ are standard and respectively model guarded conditionals, recursion, and parallel composition.

Example 3. The interplay between $\simeq_{\mathsf{C}}$ and rule $\lfloor^{\mathsf{C}}|_{\mathsf{Send}}\rceil$ yields an elegant formalisation of asynchronous behaviour for choreographies that, differently from previous work [CM13], does not require a labelled transition system and ad-hoc reduction rules. Consider Line 10 in Example 2, reported below.

$$C \stackrel{\text{def}}{=} k:\mathsf{b}[\mathtt{B}] \longrightarrow \mathsf{c}[\mathtt{C}].ok();\ k:\mathsf{b}[\mathtt{B}] \longrightarrow \mathsf{q}[\mathtt{S}].ok()$$

We can reduce C as follows (for brevity, we omit deployments):

$$\begin{array}{lll} C \to & k:\mathtt{B} \longrightarrow \mathsf{c}[\mathtt{C}].ok();\ k:\mathsf{b}[\mathtt{B}] \longrightarrow \mathsf{s}[\mathtt{S}].ok() & \text{by } \lfloor^{\mathsf{C}}|_{\mathsf{Eq}}\rceil \text{ with } \mathcal{R} = \equiv_{\mathsf{C}} \text{ and } \lfloor^{\mathsf{C}}|_{\mathsf{Send}}\rceil \\ \to & k:\mathtt{B} \longrightarrow \mathsf{s}[\mathtt{S}].ok();\ k:\mathtt{B} \longrightarrow \mathsf{c}[\mathtt{C}].ok() & \text{by } \lfloor^{\mathsf{C}}|_{\mathsf{Eq}}\rceil \text{ with } \mathcal{R} = \simeq_{\mathsf{C}} \text{ and } \lfloor^{\mathsf{C}}|_{\mathsf{Send}}\rceil \end{array}$$

In this case, process s may receive its message before process c, due to asynchronous message passing (the sending actions for process b are non-blocking).

$$
\begin{array}{lrcll}
\textit{Global Types} & G & ::= & \mathtt{A} \longrightarrow \mathtt{B}.\{o_i(U_i); G_i\}_i & (\textit{communication}) \\
& & \mid & \mathtt{rec}\ \mathbf{t}.G \quad \mid \quad \mathbf{t} & (\textit{recursion}) \\
& & \mid & \mathtt{end} & (\textit{end}) \\
\\
\textit{Local Types} & T & ::= & \oplus \mathtt{A}.\{o_i(U_i); T_i\}_i & (\textit{send}) \\
& & \mid & \& \mathtt{A}.\{o_i(U_i); T_i\}_i & (\textit{receive}) \\
& & \mid & \mathtt{rec}\ \mathbf{t}.T \quad \mid \quad \mathbf{t} & (\textit{recursion}) \\
& & \mid & \mathtt{end} & (\textit{end}) \\
\\
\textit{Sort Types} & U & ::= & \mathbf{unit} \mid \mathbf{int} \mid \mathbf{bool} \mid \mathbf{str} \mid \ldots &
\end{array}
$$

Figure 8: Frontend Choreographies — Syntax for Global and Local Types.

3. Typing

In this section, we define our typing discipline for the Frontend Choreographies. Our typing checks the behaviour of sessions against protocols, given as Multiparty Session Types [HYC08, CDCYP15]. Interestingly, we retain the same syntax of traditional Multiparty Session Types yet we ensure that correct initial deployments do not corrupt at runtime due to inconsistencies on states and message queues.

In § 3.1 we present the types that abstract choreographies, called global types. We define the syntax of global types and we introduce local types. The latter are abstract descriptions of the behaviour of single processes, used for type checking. We also formalise how, from a global type, we obtain a set of related local types by means of a projection procedure. In § 3.2 we formalise the environment and the rules of our type discipline. In § 3.3, we consider the typing of running choreographies. We illustrate why and how a choreography and its companion deployment can become inconsistent and we present a runtime typing extension to avoid inconsistencies. Finally, in § 3.4, we present two comprehensive examples to clarify the relationship between types and running choreographies, and in § 3.5 we formalise the properties guaranteed by our typing system.

3.1. Types and Type Projection.

Global and Local types. As in standard Multiparty Session Types, we use *global types* to represent protocols from a global viewpoint and *local types* to describe the behaviour of each participant. Our type system checks that a set of local types, each abstracting the behaviour of a process in a choreography, coherently follows a global type. We report in Figure 8 the syntax of global types G and local types T.

A global type $\mathtt{A} \longrightarrow \mathtt{B}.\{o_i(U_i); G_i\}_i$ abstracts a communication, where $\mathtt{A}$ can send to $\mathtt{B}$ a message on any of the operations o_i and continue with the respective continuation G_i. A carried type U types the value exchanged in the message. In local types, $!\mathtt{A}.\{o_i(U_i); T_i\}_i$ abstracts the sending of a message of type U_i to role $\mathtt{A}$ on one of the operations o_i, with continuation T_i. Dually, $?\mathtt{A}.\{o_i(U_i); T_i\}_i$ abstracts the offering of an input choice among the operations o_i, with continuation T_i. The other terms for recursion and end of types are standard. As done for FC, also in types we omit curly brackets when outputs and inputs comprise only one operation.

$$
\begin{aligned}
G_1 = \;& \mathtt{C} \longrightarrow \mathtt{S}.\mathit{buy}(\mathbf{str});\\
& \mathtt{S} \longrightarrow \mathtt{C}.\mathit{reqPay}(\mathbf{int});\\
& \mathtt{C} \longrightarrow \mathtt{B}.\mathit{sendPay}(\mathbf{str});\\
& \mathtt{B} \longrightarrow \mathtt{C}.\{\\
& \quad \mathit{ok}();\mathtt{B} \longrightarrow \mathtt{S}.\mathit{ok}(),\\
& \quad \mathit{ko}();\mathtt{B} \longrightarrow \mathtt{S}.\mathit{ko}()\\
& \};\mathtt{end}
\end{aligned}
\qquad
\begin{aligned}
G_2 = \;& \mathtt{S} \longrightarrow \mathtt{D}.\mathit{quoteShipping}(\mathbf{str});\\
& \mathtt{D} \longrightarrow \mathtt{S}.\mathit{shippingCost}(\mathbf{int});\\
& \mathtt{S} \longrightarrow \mathtt{D}.\{\\
& \quad \mathit{sendShipping}(),\\
& \quad \mathit{abortShipping}();\\
& \};\mathtt{end}
\end{aligned}
$$

Figure 9: Global types G_1 (left) and G_2 (right) abstract the respective choreographies presented in Figures 2 and 3.

$$
\begin{aligned}
\llbracket \mathtt{B} \longrightarrow \mathtt{C}.\{o_i(U_i);G_i\}_i \rrbracket_{\mathtt{A}} &= \begin{cases} \oplus\mathtt{C}.\{o_i(U_i);\llbracket G_i\rrbracket_{\mathtt{C}}\}_i & \text{if } \mathtt{A}=\mathtt{B}\\ \&\mathtt{B}.\{o_i(U_i);\llbracket G_i\rrbracket_{\mathtt{C}}\}_i & \text{if } \mathtt{A}=\mathtt{C}\\ \bigsqcup_i \llbracket G_i\rrbracket_{\mathtt{A}} & \text{otherwise}\end{cases}\\
\llbracket \mathtt{rec}\ \mathbf{t}.G\rrbracket_{\mathtt{A}} &= \begin{cases} \mathtt{rec}\ \mathbf{t}.\llbracket G\rrbracket_{\mathtt{A}} & \text{if } \mathtt{A}\in G\\ \mathtt{end} & \text{otherwise}\end{cases}\\
\llbracket \mathbf{t}\rrbracket_{\mathtt{A}} &= \mathbf{t}\\
\llbracket \mathtt{end}\rrbracket_{\mathtt{A}} &= \mathtt{end}
\end{aligned}
$$

Figure 10: Frontend Choreographies — Global Type Projection.

As an example, we report in Figure 9 two global types, G_1 and G_2, that abstract the choreographies presented in Figures 2 and 3. In particular, G_1 types session k, created at locations $(l_{\mathtt{S}}, l_{\mathtt{B}})$ — Line 1 of Figure 2 — and G_2 types session k', created at location $(l_{\mathtt{D}})$ — request at Line 3 of Figure 2, accept at Line 1 of Figure 3. We also write operations followed by empty parentheses when the type of their message U is **unit**.

Type Projection. To relate global types to the behaviour of processes in choreographies, we project a global type G onto a set of local types, each corresponding to the behaviour of a single role. We report in Figure 10 the projection of global types, defined following [MY13]. $\llbracket G\rrbracket_{\mathtt{A}}$ denotes the projection of G onto the role $\mathtt{A}$. Intuitively, $\llbracket G\rrbracket_{\mathtt{A}}$ gives an encoding of the local actions expected by role $\mathtt{A}$ in the global type G. When projecting a communication, we require the local behaviour of all roles not involved in it to be merged with the merging operator $\sqcup$. Like in [MY13], $T \sqcup T'$ is isomorphic to T and T' up to branching, where all branches of T or T' with distinct operations are also included, formally

$$
T \;\sqcup\; T' = \begin{cases} T & \text{if } T = T'\\ \&\mathtt{A}.\left\{\begin{array}{ll} \{\ o_h(U_h);T_h\ \}_{h\in I\setminus J} & \cup\\ \{\ o_h(U_h);T'_h\ \}_{h\in J\setminus I} & \cup\\ \{\ o_h(U_h);T_h \sqcup T'_h\ \}_{h\in J\cap I} & \end{array}\right\} & \begin{array}{l}\text{if } T = \&\mathtt{A}.\{o_i(U_i);T_i\}_{i\in I}\\ \text{and } T' = \&\mathtt{A}.\{o_j(U_j);T'_j\}_{j\in J}\end{array}\end{cases}
$$

$$
\begin{array}{rll}
\Gamma ::= & \varnothing & (\textit{empty environment}) \\
\mid & \Gamma, \mathtt{p}.x : U & (\textit{variable}) \\
\mid & \Gamma, X : \Gamma & (\textit{definition}) \\
\mid & \Gamma, k[\mathtt{A}] : T & (\textit{local session}) \\
\mid & \Gamma, \mathtt{p} : k[\mathtt{A}] & (\textit{ownership}) \\
\mid & \Gamma, \tilde{l} : G\langle \mathtt{A} | \tilde{\mathtt{B}} | \tilde{\mathtt{C}} \rangle & (\textit{service})
\end{array}
$$

Figure 11: Frontend Choreographies — Typing Environments.

3.2. **Type checking.** Now that we defined the relation between global and local types, we can proceed to present our system that guarantees that sessions in choreographies follow their types.

3.2.1. *Environments.* We define our typing environments $\Gamma, \Gamma', \ldots$ as reported in Figure 11.

The typing of *variables* denote that a process $\mathtt{p}$ has in its state a variable x of type U. We assume that we can write $\Gamma, \mathtt{p}.x : U$ only if either x has not been typed yet in Γ or it is already associated with the same type U (formally let $\{u, u'\} \in U$, if $u = u'$ then $\Gamma, \mathtt{p}.x : u, \mathtt{p}.x : u' = \Gamma, \mathtt{p}.x : u$). We assume a similar convention for all the identifiers in Γ except for *service* typings, whose rule for set inclusion is detailed at the end of this section. The typing of *definition* of recursive procedures associates a procedure identifier X to a typing environment Γ. A *local session* typing $k[\mathtt{A}] : T$ states that role $\mathtt{A}$ in session k follows the local type T. An *ownership* typing $\mathtt{p} : k[\mathtt{A}]$ states that process $\mathtt{p}$ owns the role $\mathtt{A}$ in session k. Hence, each process can participate in multiple sessions, but can play only one role in each session. A service typing $\tilde{l} : G\langle \mathtt{A} | \tilde{\mathtt{B}} | \tilde{\mathtt{C}} \rangle$ types with a global type G all sessions created by contacting the services at the locations $\tilde{l}$. In the typing,

- $\mathtt{A}$ is the role that the active process (the starter) should play;
- $\tilde{\mathtt{B}}$ are the roles respectively played by each service process at l in $\tilde{l}$ — we assume that each l plays a unique role, so the lengths of $\tilde{\mathtt{B}}$ and $\tilde{l}$ are the same;
- $\tilde{\mathtt{C}}$ are the roles implemented by the choreography that we are typing — we assume $\tilde{\mathtt{C}} \subseteq \tilde{\mathtt{B}}$, i.e., that $\tilde{\mathtt{C}}$ contain a subset of the roles in $\tilde{\mathtt{B}}$, ordered following the order in $\tilde{\mathtt{B}}$ (as of Definition 14).

Regarding set inclusion of service typings, when we write $\Gamma = \Gamma', \tilde{l} : G\langle \mathtt{A} | \tilde{\mathtt{B}} | \tilde{\mathtt{C}} \rangle$ we assume that:

- $\{\mathtt{A}, \tilde{\mathtt{B}}\} = \mathbf{roles}(G)$, where function **roles** returns the set of roles in G;
- the locations $\tilde{l}$ are ordered lexicographically;
- the locations in $\tilde{l}$ do not appear in any other service typing in Γ;
- that either:
  - $\tilde{l}$ does not appear in Γ' and the resulting Γ includes it, formally $\tilde{l} \notin \mathbf{dom}(\Gamma')$ and $\Gamma = \Gamma',\ \tilde{l} : G\langle \mathtt{A} | \tilde{\mathtt{B}} | \tilde{\mathtt{C}} \rangle$;
  - $\tilde{l}$ appears in Γ', such that $\Gamma' = \Gamma'', \tilde{l} : G\langle \mathtt{A} | \tilde{\mathtt{B}} | \tilde{\mathtt{D}} \rangle$, and $\{\tilde{\mathtt{C}}\} \cap \{\tilde{\mathtt{D}}\} = \varnothing$, i.e., the roles in $\tilde{\mathtt{C}}$ do not appear in $\tilde{\mathtt{D}}$. The resulting Γ includes in the service typing of $\tilde{l}$ the merged list of roles in $\tilde{\mathtt{C}}$ and $\tilde{\mathtt{D}}$, following the lexicographic order in $\tilde{\mathtt{B}}$. We write the merge as $\tilde{\mathtt{D}} \bowtie_{\tilde{\mathtt{B}}} \tilde{\mathtt{C}}$ (see Definition 15) and $\Gamma = \Gamma'', \tilde{l} : G\langle \mathtt{A} | \tilde{\mathtt{B}} | \tilde{\mathtt{D}} \bowtie_{\tilde{\mathtt{B}}} \tilde{\mathtt{C}} \rangle$.

$$\frac{\Gamma, \tilde{l} : G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{B}}\rangle, \mathbf{init}(\ \widetilde{\mathtt{r}[\mathtt{C}]}, k, G\ ) \vdash C \quad \widetilde{\mathtt{r}[\mathtt{C}]} = \mathtt{p}[\mathtt{A}], \widetilde{\mathtt{q}[\mathtt{B}]} \quad \tilde{\mathtt{q}} \notin \Gamma}{\Gamma, \tilde{l} : G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{B}}\rangle \vdash \mathtt{start}\ k : \mathtt{p}[\mathtt{A}] \leftrightarrows \widetilde{l.\mathtt{q}[\mathtt{B}]}; C}\ \llbracket \cdot |_{\mathsf{Start}} \rrbracket$$

$$\frac{\Gamma, \mathtt{p} : k[\mathtt{A}], k[\mathtt{A}] : [\![G]\!]_{\mathtt{A}} \vdash C \quad \tilde{l} : G\langle \mathtt{A}|\tilde{\mathtt{B}}|\varnothing\rangle \in \Gamma}{\Gamma \vdash \mathtt{req}\ k : \mathtt{p}[\mathtt{A}] \leftrightarrows \widetilde{l.\mathtt{B}}; C}\ \llbracket \cdot |_{\mathsf{Req}} \rrbracket$$

$$\frac{\tilde{l} \subseteq \tilde{l}' \quad \Gamma, \tilde{l}' : G\langle \mathtt{A}|\tilde{\mathtt{B}}|\varnothing\rangle, \mathbf{init}(\ \widetilde{\mathtt{q}[\mathtt{C}]}, k, G\ ) \vdash C \quad \tilde{\mathtt{q}} \notin \Gamma}{\Gamma, \tilde{l}' : G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{C}}\rangle \vdash \mathtt{acc}\ k : \widetilde{l.\mathtt{q}[\mathtt{C}]}; C}\ \llbracket \cdot |_{\mathsf{Acc}} \rrbracket$$

$$\frac{\Gamma \vdash \mathtt{p} : k[\mathtt{A}], \mathtt{q} : k[\mathtt{B}] \quad j \in I \quad \Gamma \vdash \mathtt{p}.e : U_j \quad \Gamma, \mathtt{q}.x : U_j, k[\mathtt{A}] : T_j, k[\mathtt{B}] : T'_j \vdash C}{\Gamma, k[\mathtt{A}] : \ \oplus\, \mathtt{B}.\{o_i(U_i); T_i\}_{i\in I}, k[\mathtt{B}] : \ \&\mathtt{A}.\{o_i(U_i); T'_i\}_{i\in I} \vdash k : \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{q}[\mathtt{B}].o_j(x); C}\ \llbracket \cdot |_{\mathsf{Com}} \rrbracket$$

$$\frac{j \in I \quad \Gamma \vdash \mathtt{p} : k[\mathtt{A}] \quad \Gamma \vdash \mathtt{p}.e : U_j \quad \Gamma, k[\mathtt{A}] : T_j \vdash C}{\Gamma, k[\mathtt{A}] : \ \oplus\, \mathtt{B}.\{o_i(U_i); T_i\}_{i\in I} \vdash k : \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o_j; C}\ \llbracket \cdot |_{\mathsf{Send}} \rrbracket$$

$$\frac{\Gamma \vdash \mathtt{q} : k[\mathtt{B}] \quad \forall j \in I.\ \Gamma, \mathtt{q}.x_j : U_j, k[\mathtt{B}] : T_j \vdash C_j}{\Gamma, k[\mathtt{B}] : \ \&\mathtt{A}.\{o_i(U_i); T_i\}_{i\in I} \vdash k : \mathtt{A} \longrightarrow \mathtt{q}[\mathtt{B}].\{o_j(x_j); C_j\}_{j\in I\cup J}}\ \llbracket \cdot |_{\mathsf{Recv}} \rrbracket$$

$$\frac{\Gamma \vdash \mathtt{p}.e : \mathbf{bool} \quad \Gamma \vdash C_1 \quad \Gamma \vdash C_2}{\Gamma \vdash \mathtt{if}\ \mathtt{p}.e\ \{C_1\}\ \mathtt{else}\ \{C_2\}}\ \llbracket \cdot |_{\mathsf{Cond}} \rrbracket$$

$$\frac{\Gamma, X : \Gamma' \vdash C \quad \Gamma', X : \Gamma' \vdash C' \quad \Gamma'|_{\mathbf{locs}} \subseteq \Gamma}{\Gamma \vdash \mathtt{def}\ X = C'\ \mathtt{in}\ C}\ \llbracket \cdot |_{\mathsf{Def}} \rrbracket$$

$$\frac{\Gamma_1 \vdash C_1 \quad \Gamma_2 \vdash C_2}{\Gamma_1, \Gamma_2 \vdash C_1 \mid C_2}\ \llbracket \cdot |_{\mathsf{Par}} \rrbracket \qquad \frac{\mathtt{end}(\Gamma)}{\Gamma \vdash \mathbf{0}}\ \llbracket \cdot |_{\mathsf{End}} \rrbracket \qquad \frac{\Gamma'' \subseteq \Gamma' \quad \mathtt{end}(\Gamma)}{\Gamma, \Gamma', X : \Gamma'' \vdash X}\ \llbracket \cdot |_{\mathsf{Call}} \rrbracket$$

Figure 12: Frontend Choreographies — Typing Rules.

We underline that the annotation $\tilde{\mathtt{C}}$ in service typings plays two important parts: it enables the composition of choreographies and it ensures that only one choreography implements a specific role. This is mirrored in the composition $\Gamma = \Gamma', \tilde{l} : G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{C}}\rangle$ where, if Γ' already contains the typing for some roles $\tilde{\mathtt{D}}$ in $\tilde{l}$, Γ will contain the additional roles defined in $\tilde{\mathtt{C}}$ (provided $\tilde{\mathtt{D}}$ and $\tilde{\mathtt{C}}$ contain distinct roles).

3.2.2. *Typing Judgements and Rules.* A judgement $\Gamma \vdash C$ states that the choreography C follows the specifications given in Γ. We comment the typing rules reported in Figure 12.

Rule $\llbracket \cdot |_{\mathsf{Start}} \rrbracket$ types a session start. In the first premise, the service typing $\tilde{l} : G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{B}}\rangle$ checks that the continuation implements all the roles in protocol G. The function **init** assembles the typing environment that correctly types — with the appropriate ownerships and local typings — the freshly-started session k, given the global type G and the processes in $\tilde{\mathtt{p}}$, each playing its corresponding role in $\tilde{\mathtt{B}}$. Formally,

$$\mathbf{init}(\ \widetilde{\mathtt{p}[\mathtt{A}]}, k, G\ ) = \left\{\ \mathtt{q} : k[\mathtt{B}],\ k[\mathtt{B}] : [\![G]\!]_{\mathtt{B}}\ \middle|\ \mathtt{q}[\mathtt{B}] \in \left\{\widetilde{\mathtt{p}[\mathtt{A}]}\right\}\ \right\}.$$

where the type of each process $\mathtt{p} \in \tilde{\mathtt{p}}$ playing role $\mathtt{B} \in \tilde{\mathtt{B}}$ is the local type projection $[\![G]\!]_{\mathtt{B}}$ of the global type G. In $[\![\mathsf{T}|_{\mathsf{Start}}]\!]$, we abuse the notation $\tilde{\mathtt{q}} \notin \Gamma$ to check that all freshly created processes in $\tilde{\mathtt{q}}$ do not appear in Γ (i.e., there is no variable or ownership typings in Γ associated with any process in $\tilde{\mathtt{q}}$).

Rule $[\![\mathsf{T}|_{\mathsf{Req}}]\!]$ types (*req*) terms and is similar to $[\![\mathsf{T}|_{\mathsf{Start}}]\!]$, although it only performs the checks for the process $\mathtt{p}$, playing role $\mathtt{A}$, that requests the creation of the new session k. Dually, $[\![\mathsf{T}|_{\mathsf{Acc}}]\!]$ mirrors Rule $[\![\mathsf{T}|_{\mathsf{Start}}]\!]$ and $\tilde{l} \subseteq \tilde{l}'$ checks that (following Definition 14) the list of locations of the service typing in Γ includes the locations in the (*acc*) term. In the premise, we type the continuation C with $\tilde{l}' : G\langle \mathtt{A} | \tilde{\mathtt{B}} | \varnothing \rangle$ since (*acc*) terms can only appear at top level in choreographies.

Rule $[\![\mathsf{T}|_{\mathsf{Com}}]\!]$ types a complete communication. From left to right the premises check that:

(1) the sender $\mathtt{p}$ and the receiver $\mathtt{q}$ own their respective roles in the session;
(2) since $j \in I$:
   - operation o_j can be effectively selected by the sender, according to its local type;
   - similarly, o_j is among the operations offered by the receiver, according to its local type;
(3) the expression of the sender (e) has the type[1] U_j, expected by the protocol;
(4) the resulting environment $\Gamma, \mathtt{q}.x : U_j, k[\mathtt{A}] : T_j, k[\mathtt{B}] : T'_j$ correctly types the continuation C, in particular that:
   - the receiver $\mathtt{q}$ correctly uses the reception variable x in C;
   - processes $\mathtt{p}$ and $\mathtt{q}$ proceed according to their local types, respectively T_j and T'_j.

Rules $[\![\mathsf{T}|_{\mathsf{Send}}]\!]$ and $[\![\mathsf{T}|_{\mathsf{Recv}}]\!]$ share part of the checks commented for $[\![\mathsf{T}|_{\mathsf{Com}}]\!]$ and judge the respective partial terms (*send*) and (*recv*). Note that, as in standard multiparty session types, the local typing of the branching process $\mathtt{q}$ is contravariant wrt the branches in the choreography, i.e., the rule $[\![\mathsf{T}|_{\mathsf{Recv}}]\!]$ checks that the operations supported by the typing $o_i, i \in I$ are at least a subset of the actual operations $o_j, j \in I \cup J$ provided in the (*recv*) term.

Rule $[\![\mathsf{T}|_{\mathsf{Cond}}]\!]$ checks that the expression of a conditional has a compatible type (**bool**) and that both branches C_1 and C_2 are correctly typed by Γ.

Rule $[\![\mathsf{T}|_{\mathsf{Def}}]\!]$ checks procedure definitions. Here, function $|_{\mathbf{locs}}$ applied to an environment Γ returns all service typings in it. In the rule we write $\Gamma'|_{\mathbf{locs}} \subseteq \Gamma$ to check that the body of the recursive procedure does not introduce unexpected services, i.e., services that are not present at top level.

In rule $[\![\mathsf{T}|_{\mathsf{Par}}]\!]$ we extend the set inclusion for Γ_1, Γ_2 point-wise to the identifiers in Γ to merge typings and to check that choreographies executing in parallel do not implement overlapping roles at locations.

In rule $[\![\mathsf{T}|_{\mathsf{End}}]\!]$, the predicate $\mathtt{end}(\Gamma)$ holds if the protocols for all sessions in Γ have terminated (i.e., all local typings have type $\mathtt{end}$).

$[\![\mathsf{T}|_{\mathsf{Call}}]\!]$ checks a procedure call. The premise $\Gamma'' \subseteq \Gamma$ checks that procedure X does not introduce unexpected typings (and, by extension, behaviours) wrt the active sessions contained in Γ'. The premise $\mathtt{end}(\Gamma)$ makes sure that the remaining sessions in the typing environment have all terminated.

3.3. **Runtime Typing.** To prove that well-typed FC programs never go wrong, we need to pay attention to how their deployments evolve at runtime. For example, in rule $[\![\mathsf{C}|_{\mathsf{Send}}]\!]$, the deployment D must contain the proper queue where the sender can deliver its message: a remarkable difference wrt previous works on choreographies, where such conditions do

[1] The judgement $\vdash v : U$ reads as "value v has type U".

not exist and choreographies can always continue execution (see, e.g., [QZCY07, BGG+06, CHY12, CM13]).

To guarantee that well-typed FC programs never go wrong, we must guarantee that their companion deployments evolve in a consistent way. We address this issue by extending our typing discipline to check runtime states.

Wrong Deployments. We want to rule out "wrong" deployments. Intuitively, we say that a deployment is wrong wrt a choreography if e.g., processes have undefined variables that are used in the choreography or a message queue does not contain messages as expected by the protocol of the session in which it is used.

Wrong deployments may cause unpredictable executions or faulty behaviours, e.g., deadlocks. We illustrate the consequences of having wrong deployments with this simple running choreography:

$$D, k : \mathtt{p}[\mathtt{A}].y \longrightarrow \mathtt{q}[\mathtt{B}].o(x); \mathbf{0}$$

- *(uninitialised variables)* assume that D is such that the state of process $\mathtt{p}$ in D, $D(\mathtt{p})$, does not contain a value for variable y; then the condition $\mathbf{eval}(\ y, D(\mathtt{p})\ )$ given in rule $[\mathsf{P}|_{\mathsf{Send}}]$ is undefined and rule $[\mathsf{C}|_{\mathsf{Com}}]$ cannot be applied, causing the choreography to get stuck.
- *(protocol violations)* assume that $D(\ k[\mathtt{A}\rangle\mathtt{B}]\ ) = (o', v)$ where $o \neq o'$. Namely, that i) in session k process $\mathtt{q}$ (playing role $\mathtt{B}$) has a message in its receiving queue from process $\mathtt{p}$ (playing role $\mathtt{A}$) and ii) the operation of the message is o', different from operation o expected in the choreography. If we let the choreography reduce following the previous point, it ends up deadlocked. After the reduction, the queue used by $\mathtt{p}$ contains in its head the message (o', v) and we cannot apply rule $[\mathsf{C}|_{\mathsf{Recv}}]$, as it expects to find a message for o at that position.

To avoid these outcomes, we extend our type system to prove that, given a well-typed choreography and a non-wrong companion deployment, our semantics never produces wrong deployments. Note that this development is transparent to programmers, since default deployments are trivially never wrong.

Runtime Global Types. To capture asynchrony and partial runtime states, we extend the syntax of global types with:

$$\begin{array}{rll} G ::= & \ldots & \\ \mid & \oplus \mathtt{AB}.\{o_i(U_i)\}; G & (\textit{global choice}) \\ \mid & \&\mathtt{AB}.\{o_i(U_i); G_i\}_{i \in I} & (\textit{global branch}) \\ \mid & \mathtt{A}\rangle\mathtt{B}.o(U); G & (\textit{global buffer}) \end{array}$$

Global choice and branch are the equivalent of a complete communication $\mathtt{A} \longrightarrow \mathtt{B}.o(U); G$ where: $\oplus \mathtt{AB}.\{o_i(U_i)\}; G$ means that role $\mathtt{A}$ can choose to send a message to role $\mathtt{B}$ on operation o_i with type U_i, proceeding with continuation G; while $\&\mathtt{AB}.\{o_i(U_i); G_i\}_{i \in I}$ means that $\mathtt{B}$ can receive a message from $\mathtt{A}$ on any operation $o_i, i \in I$, proceeding with the related continuation G_i.

When the choice performed by $\mathtt{A}$ is applied to the branch controlled by $\mathtt{B}$, we obtain term $\mathtt{A}\rangle\mathtt{B}.o(U)$, which marks that $\mathtt{A}$ has sent the message but $\mathtt{B}$ still has to consume it.

Semantics of Global Types. To express the (abstract) execution of protocols, we give a semantics for global types. Formally, $G \to G'$ is the smallest relation on the recursion-unfolding of global types satisfying the rules in Figure 13.

$$\frac{o \in \bigcup_i \{o_i\} \quad G' = {}^{\mathtt{A}\rangle\mathtt{B}}_{o}\!\downarrow G}{\oplus \mathtt{A}_{\mathtt{B}}.\{o_i(U_i)\}; G \quad \to \quad G'} \; \lfloor^{\mathtt{G}}_{\mathsf{Send}}\rceil \qquad \frac{}{\mathtt{A}\rangle\mathtt{B}.o(U); G \quad \to \quad G} \; \lfloor^{\mathtt{G}}_{\mathsf{Recv}}\rceil$$

$$\frac{\mathcal{R} \in \{\equiv_{\mathtt{G}}, \simeq_{\mathtt{G}}\} \quad G \; \mathcal{R} \; G_1 \quad G_1 \to G_1' \quad G_1' \; \mathcal{R} \; G'}{G \quad \to \quad G'} \; \lfloor^{\mathtt{G}}_{\mathsf{Eq}}\rceil$$

Reduction Rules.

$$\mathtt{A} \longrightarrow \mathtt{B}.\{o_i(U_i); G_i\} \quad \equiv_{\mathtt{G}} \quad \oplus \mathtt{A}_{\mathtt{B}}.\{o_i(U_i)\}; \&_{\mathtt{A}}\mathtt{B}.\{o_i(U_i); G_i\}$$

$$G[\mathtt{rec}\ \mathbf{t}.G'] \quad \equiv_{\mathtt{G}} \quad G\,[\; G'[\mathtt{rec}\ \mathbf{t}.G'/\mathbf{t}]\;]$$

Structural Congruence.

$$\frac{\mathtt{A} \neq \mathtt{C} \vee \mathtt{B} \neq \mathtt{D}}{\oplus \mathtt{A}_{\mathtt{B}}.\{o_i(U_i)\}; \oplus \mathtt{C}_{\mathtt{D}}.\{o_j(U_j)\} \simeq_{\mathtt{G}} \oplus \mathtt{C}_{\mathtt{D}}.\{o_j(U_j)\}; \oplus \mathtt{A}_{\mathtt{B}}.\{o_i(U_i)\}} \; \lfloor^{\mathtt{GS}}_{\mathsf{ChoCho}}\rceil$$

$$\frac{\mathtt{A} \neq \mathtt{C} \vee \mathtt{B} \neq \mathtt{D}}{\&_{\mathtt{A}}\mathtt{B}.\{o_i(U_i); \&_{\mathtt{C}}\mathtt{D}.\{o_j(U_j); G_{ij}\}\} \simeq_{\mathtt{G}} \&_{\mathtt{C}}\mathtt{D}.\{o_j(U_j); \&_{\mathtt{A}}\mathtt{B}.\{o_i(U_i); G_{ij}\}\}} \; \lfloor^{\mathtt{GS}}_{\mathsf{BrcBrc}}\rceil$$

$$\frac{\mathtt{A} \neq \mathtt{D}}{\oplus \mathtt{A}_{\mathtt{B}}.\{o_i(U_i)\}; \&_{\mathtt{C}}\mathtt{D}.\{o_j(U_j); G_j\} \simeq_{\mathtt{G}} \&_{\mathtt{C}}\mathtt{D}.\{o_j(U_j); \oplus \mathtt{A}_{\mathtt{B}}.\{o_i(U_i)\}; G_j\}} \; \lfloor^{\mathtt{GS}}_{\mathsf{ChoBrc}}\rceil$$

$$\frac{\mathtt{A} \neq \mathtt{C} \vee \mathtt{B} \neq \mathtt{D}}{\mathtt{A}\rangle\mathtt{B}.o(U); \&_{\mathtt{C}}\mathtt{D}.\{o_j(U_j); G_j\} \simeq_{\mathtt{G}} \&_{\mathtt{C}}\mathtt{D}.\{o_j(U_j); \mathtt{A}\rangle\mathtt{B}.o(U); G_j\}} \; \lfloor^{\mathtt{GS}}_{\mathsf{BufBrc}}\rceil$$

$$\frac{\mathtt{A} \neq \mathtt{C} \vee \mathtt{B} \neq \mathtt{D}}{\mathtt{A}\rangle\mathtt{B}.o(U); \mathtt{C}\rangle\mathtt{D}.o'(U') \simeq_{\mathtt{G}} \mathtt{C}\rangle\mathtt{D}.o'(U'); \mathtt{A}\rangle\mathtt{B}.o(U)} \; \lfloor^{\mathtt{GS}}_{\mathsf{BufBuf}}\rceil$$

$$\frac{\mathtt{A} \neq \mathtt{D}}{\oplus \mathtt{A}_{\mathtt{B}}.\{o_i(U_i)\}; \mathtt{C}\rangle\mathtt{D}.o(U) \simeq_{\mathtt{G}} \mathtt{C}\rangle\mathtt{D}.o(U); \oplus \mathtt{A}_{\mathtt{B}}.\{o_i(U_i)\}} \; \lfloor^{\mathtt{GS}}_{\mathsf{ChoBuf}}\rceil$$

Swap Relation.

$$\begin{array}{rcll} {}^{\mathtt{A}\rangle\mathtt{B}}_{o_j}\!\downarrow \&_{\mathtt{A}}\mathtt{B}.\{o_i(U_i); G_i\}_{i\in I} & = & \mathtt{A}\rangle\mathtt{B}.o_j(U_j); G_j & \text{if } j \in I \\ {}^{\mathtt{A}\rangle\mathtt{B}}_{o_j}\!\downarrow \&_{\mathtt{C}}\mathtt{D}.\{o_i(U_i); G_i\}_{i\in I} & = & \&_{\mathtt{C}}\mathtt{D}.\{o_i(U_i); {}^{\mathtt{A}\rangle\mathtt{B}}_{o_j}\!\downarrow G_i\} & \text{if } \mathtt{A} \neq \mathtt{C} \vee \mathtt{B} \neq \mathtt{D} \end{array}$$

$${}^{\mathtt{A}\rangle\mathtt{B}}_{o}\!\downarrow \mathtt{C}\rangle\mathtt{D}.o(U); G = \mathtt{C}\rangle\mathtt{D}.o(U); {}^{\mathtt{A}\rangle\mathtt{B}}_{o}\!\downarrow G \qquad {}^{\mathtt{A}\rangle\mathtt{B}}_{o}\!\downarrow \oplus\, \mathtt{C}_{\mathtt{D}}.\{o_i(U_i)\}; G = \oplus \mathtt{C}_{\mathtt{D}}.\{o_i(U_i)\}; {}^{\mathtt{A}\rangle\mathtt{B}}_{o}\!\downarrow G$$

$${}^{\mathtt{A}\rangle\mathtt{B}}_{o}\!\downarrow \mathtt{rec}\ \mathbf{t}.G = \mathtt{rec}\ \mathbf{t}.G \qquad {}^{\mathtt{A}\rangle\mathtt{B}}_{o}\!\downarrow \mathbf{t} = \mathbf{t} \qquad {}^{\mathtt{A}\rangle\mathtt{B}}_{o}\!\downarrow \mathtt{end} = \mathtt{end}$$

Application Function.

Figure 13: Global types — Semantics.

Rule $\lfloor^{\mathtt{G}}_{\mathsf{Send}}\rceil$ allows the sending of a message from a (*global choice*). The continuation G' is obtained from the application of the sending to the corresponding (*global branch*), with function ${}^{\mathtt{A}\rangle\mathtt{B}}_{o_i}\!\downarrow G$ that transforms the related branch in G into a (*global buffer*) on the selected operation o_i, followed by the respective continuation G_i.

The actual reception of the message is executed in rule $\lfloor^{\mathtt{G}}_{\mathsf{Recv}}\rceil$. In $\lfloor^{\mathtt{G}}_{\mathsf{Eq}}\rceil$ we model the splitting of complete communications and recursion unfolding with the structural equivalence $\equiv_{\mathtt{G}}$, the smallest congruence defined by the rules in Figure 13. To capture the semantics of asynchronous message delivery, we define the swap relation $\simeq_{\mathtt{G}}$ as the smallest congruence

$$
\begin{array}{lcl}
[\![\mathtt{C} \longrightarrow \mathtt{D}.\{o_i(U_i); G_i\}]\!]^{\mathtt{A}}_{\mathtt{B}} & = & \begin{cases} \mathtt{end} & \text{if } \mathtt{C} = \mathtt{A} \ \wedge \ \mathtt{D} = \mathtt{B} \\ \bigsqcup_i [\![G_i]\!]^{\mathtt{A}}_{\mathtt{B}} & \text{otherwise} \end{cases} \\
[\![\mathtt{C}\rangle\mathtt{D}.o(U); G]\!]^{\mathtt{A}}_{\mathtt{B}} & = & \begin{cases} \&\mathtt{A}.o(U); [\![G]\!]^{\mathtt{A}}_{\mathtt{B}} & \text{if } \mathtt{C} = \mathtt{A} \wedge \ \mathtt{D} = \mathtt{B} \\ [\![G]\!]^{\mathtt{A}}_{\mathtt{B}} & \text{otherwise} \end{cases} \\
[\![\mathbf{t}]\!]^{\mathtt{A}}_{\mathtt{B}} = [\![\mathtt{end}]\!]^{\mathtt{A}}_{\mathtt{B}} = [\![\mathtt{rec}\ \mathbf{t}.G]\!]^{\mathtt{A}}_{\mathtt{B}} & = & \mathtt{end}
\end{array}
$$

Figure 14: Frontend Choreographies — Buffer Type Projection.

defined by the rules in Figure 13. Both congruences are similar to what presented for choreographies in § 2.2. Note that rules $\lfloor^{\mathsf{GS}}|_{\mathsf{ChoBrc}}\rceil$ and $\lfloor^{\mathsf{GS}}|_{\mathsf{ChoBuf}}\rceil$ enable the swapping of choice terms with receptions, as long as the swap preserves the causal consistency between operations (i.e., we do not swap a sending that is causally dependent from a reception on the same role).

Runtime Type checking and Typing Rules. We extend the typing rules given in the previous section to check runtime terms. The extension consists in *i*) new terms for Γ, and *ii*) the introduction of rule $\lfloor^{\mathsf{T}}|_{\mathsf{DC}}\rceil$ to type runtime choreographies. We extend the grammar of typing environments with

$$
\begin{array}{rcll}
\Gamma ::= & \ldots & & \\
\mid & \Gamma, \mathtt{p}@l & & (\textit{location}) \\
\mid & \Gamma, k[\mathtt{A}\rangle\mathtt{B}] : T & & (\textit{buffer})
\end{array}
$$

where $\Gamma, \mathtt{p}@l$ states that process $\mathtt{p}$ runs at location l and a *buffer* typing $k[\mathtt{A}\rangle\mathtt{B}] : T$ types the messages in the queue where the process implementing role $\mathtt{B}$ in session k receives messages from role $\mathtt{A}$. We extend to buffer typings the assumption for set inclusion stated for standard elements in Γ. For location typings, we assume that we can write $\Gamma, \mathtt{p}@l$ only if $\mathtt{p}@l \notin \Gamma$. This formalises the requirement that a process can appear only in one choreography (e.g., given the choreography $C = C_1 \mid C_2$ process $\mathtt{p} \in \mathbf{pn}(C)$ appears either in C_1 or in C_2) and that it is associated only to one location.

To relate the typings of queues to the buffer types expected by the protocol of sessions, we define the *buffer type projection* $[\![G]\!]^{\mathtt{A}}_{\mathtt{B}}$, which follows the rules in Figure 14 and returns the expected buffer type of role $\mathtt{B}$ from $\mathtt{A}$ in G. $[\![G]\!]^{\mathtt{A}}_{\mathtt{B}}$ extracts from G the partial receptions of the form $\mathtt{A}\rangle\mathtt{B}.o(U)$, translating them to local types of the form $\&\mathtt{A}.o(U)$. Below, we report the rule that extends global type projection for global buffers.

$$
[\![\mathtt{A}\rangle\mathtt{B}.o(U); G]\!]_{\mathtt{C}} = \begin{cases} \&\mathtt{A}.o(U); [\![G]\!]_{\mathtt{C}} & \text{if } \mathtt{C} = \mathtt{B} \\ [\![G]\!]_{\mathtt{C}} & \text{otherwise} \end{cases}
$$

Note that we do not need to extend the projection to (*global choice*) and (*global branch*). Indeed, in our setting we consider only running global types that are evolution of a global type, hence global choices and branches are always balanced. Given a running global type G, we can always obtain an equivalent ($\simeq_{\mathtt{G}}$, $\equiv_{\mathtt{G}}$) global type G' which is absent from (*global choice*) and (*global branch*) terms. We call a running global type *canonic* if it contains no (*global choice*) and (*global branch*) terms. When writing projections of global types we assume G to be in canonic form.

Finally, we extend our typing discipline with a new rule $\lfloor^{\mathsf{T}}|_{\mathsf{DC}}\rceil$ that checks for coherence among types, choreographies, and deployments. To define $\lfloor^{\mathsf{T}}|_{\mathsf{DC}}\rceil$, we formalise a predicate,

called *partial coherence*[2] and denoted $\mathbf{pco}(\Gamma)$, that holds if and only if, for all sessions k, the local and buffer typings of k follow (are projections of) the same global type G.

Definition 2 (Partial Coherence)**.** We write $\mathbf{pco}(\Gamma)$ when, for all sessions k in Γ, there exists a global type G such that

$$\forall\, k[\mathtt{B}] \colon T \in \Gamma,\ T = [\![G]\!]_{\mathtt{B}} \quad \wedge \quad \forall\, \mathtt{A} \in \mathbf{roles}(G) \setminus \{\mathtt{B}\},\ \Gamma \vdash k[\mathtt{A}\rangle\mathtt{B}] \colon [\![G]\!]^{\mathtt{A}}_{\mathtt{B}}$$

Rule $[{}^{\mathsf{T}}|_{\mathsf{DC}}]$ is defined as:

$$\frac{\mathbf{pco}(\Gamma) \quad \Gamma \vdash D \quad \Gamma \vdash C}{\Gamma \vdash D, C}\ [{}^{\mathsf{T}}|_{\mathsf{DC}}]$$

where a judgement $\Gamma \vdash D, C$ states that C and D are coherent according to Γ and all sessions in Γ are coherent. Γ is an abstraction between D and C and guarantees that D cannot go wrong. Formally

Definition 3 (Deployment Judgements)**.**

$$\Gamma \vdash D \iff \begin{cases} (1)\ \forall\ \mathtt{p}.x \colon U \in \Gamma,\ D(\mathtt{p}).x : U \\ (2)\ \forall\ k[\mathtt{A}\rangle\mathtt{B}] : T \in \Gamma \wedge D(k[\mathtt{A}\rangle\mathtt{B}]) = \tilde{m},\ \mathtt{bte}(\mathtt{A}, \tilde{m}) = T \end{cases}$$

We comment the checks performed by $\Gamma \vdash D$: (1) checks that, for each typing $\mathtt{p}.x \colon U$ in Γ, D associates x, in the state of process $\mathtt{p}$, to a value of type U; (2) uses buffer types to check that the typing of a message queue in Γ is correct wrt to the actual sequence of messages stored by that queue in D. We extract the type of a queue $\tilde{m}$, i.e., the sequence of message receptions from a role $\mathtt{A}$, with function $\mathtt{bte}(\mathtt{A}, \tilde{m})$. Formally,

Definition 4 (Buffer Type Extraction)**.** Let $\vdash v_i \colon U_i$, $i \in [1, n]$ and $\tilde{m} = (o_1, v_1) :: \cdots :: (o_n, v_n)$ then $\mathtt{bte}(\ \mathtt{A}, \tilde{m}\ ) = \&\mathtt{A}.o_1(U_1);\ \cdots\ ;\ \&\mathtt{A}.o_n(U_n)$.

3.4. **Runtime Examples, Typing and Reductions.** In this section, we present two running examples that illustrate the relationship between global types and choreographies. First we report a basic case where a session starts and two processes exchange a message. Then we consider a started session and comment the asynchronous delivery of messages.

Example 4 (Start and Message Delivery)**.** We consider a running choreography C, D and a global type G such that D is a default deployment (cf. Definition 1) and

$$\begin{array}{rl} C = & \mathtt{start}\ k : \mathtt{a}[\mathtt{A}] \leftrightarrow l_{\mathtt{B}}.\mathtt{b}[\mathtt{B}], l_{\mathtt{C}}.\mathtt{c}[\mathtt{C}]; \\ & k \colon \mathtt{a}[\mathtt{A}].\texttt{"ok"} \longrightarrow \mathtt{b}[\mathtt{B}].pass(\ x\ ); \\ & k \colon \mathtt{b}[\mathtt{B}].x \longrightarrow \mathtt{c}[\mathtt{C}].fwd(\ x\ ) \end{array} \qquad \begin{array}{rl} G = & \mathtt{A} \longrightarrow \mathtt{B}.pass(\mathbf{str}); \\ & \mathtt{B} \longrightarrow \mathtt{C}.fwd(\mathbf{str}); \\ & \mathtt{end} \end{array}$$

The global type G is used in the typing environment Γ to check C, D, formally the service typing $l_{\mathtt{B}}, l_{\mathtt{C}} \colon G\langle\ \mathtt{A} \mid \mathtt{B}, \mathtt{C} \mid \mathtt{B}, \mathtt{C}\ \rangle$ belongs to Γ and $\Gamma \vdash C, D$.

Now, we let D, C reduce to D', C' following rules $[{}^{\mathsf{C}}|_{\mathsf{Start}}]$ and $[{}^{\mathsf{P}}|_{\mathsf{Start}}]$ so that D contains the data and queues needed to support interactions on session k. Finally, we report in Table 1:

- left column, the main elements in the typing environment Γ, i.e., the evolution of the type G. To show how partial coherence (Definition 2) holds, we report also the local and buffer types of $\mathtt{A}$, $\mathtt{B}$, and $\mathtt{C}$ projected from G following global type projection $[\![G]\!]_{\mathtt{A}}$ for local types (see Figure 10) and buffer type projection $[\![G]\!]^{\mathtt{B}}_{\mathtt{A}}$ (see Figure 14) for buffer types. For brevity, we omit to report empty buffer types such as $k[\mathtt{A}\rangle\mathtt{B}] = \mathtt{end}$;

[2]Partial because it accounts for missing typings of roles implemented by external partial choreographies.

	Typing Environment	Choreography	Deployment
①	$G = \texttt{A} \to \texttt{B}.pass(\mathbf{str});\ \texttt{B} \to \texttt{C}.fwd(\mathbf{str});\ \mathbf{end}$ $k[\texttt{A}] = \oplus\texttt{B}.pass(\mathbf{str}); \mathbf{end}$ $k[\texttt{B}] = \&\texttt{A}.pass(\mathbf{str});\ \oplus\texttt{C}.fwd(\mathbf{str}); \mathbf{end}$ $k[\texttt{C}] = \&\texttt{B}.fwd(\mathbf{str}); \mathbf{end}$	$C' = k{:}\texttt{a}[\texttt{A}]."\texttt{ok}" \to \texttt{b}[\texttt{B}].first(\ x\ );\ k{:}\texttt{b}[\texttt{B}].x \to \texttt{c}[\texttt{C}].second(\ x\ )$	D'
	$G \to G'$ by $\lfloor^{\texttt{G}}\vert_{\mathsf{Eq}}\rceil, \lfloor^{\texttt{G}}\vert_{\mathsf{Send}}\rceil$	$C' \to C''$ by $\lfloor^{\texttt{C}}\vert_{\mathsf{Eq}}\rceil, \lfloor^{\texttt{C}}\vert_{\mathsf{Send}}\rceil$	$\delta = k{:}\texttt{a}[\texttt{A}]."\texttt{ok}" \to \texttt{B}.pass$ $D', \delta \blacktriangleright D''$ by $\lfloor^{\texttt{D}}\vert_{\mathsf{Send}}\rceil$
②	$G' = \texttt{A}\rangle\texttt{B}.pass(\mathbf{str});\ \texttt{B} \to \texttt{C}.fwd(\mathbf{str});\ \mathbf{end}$ $k[\texttt{A}] = \mathbf{end}$ $k[\texttt{B}] = \&\texttt{A}.pass(\mathbf{str});\ \oplus\texttt{C}.fwd(\mathbf{str}); \mathbf{end}$ $k[\texttt{C}] = \&\texttt{B}.fwd(\mathbf{str}); \mathbf{end}$ $k[\texttt{A}\rangle\texttt{B}] = \&\texttt{A}.pass(\mathbf{str})$	$C'' = k{:}\texttt{A} \to \texttt{b}[\texttt{B}].pass(\ x\ );\ k{:}\texttt{b}[\texttt{B}].x \to \texttt{c}[\texttt{C}].fwd(\ x\ )$	$D''(k[\texttt{A}\rangle\texttt{B}]) = (pass, "\texttt{ok}")$
	$G' \to G''$ by $\lfloor^{\texttt{G}}\vert_{\mathsf{Recv}}\rceil$	$C'' \to C'''$ by $\lfloor^{\texttt{C}}\vert_{\mathsf{Recv}}\rceil$	$\delta' = k{:}\texttt{A} \to \texttt{b}[\texttt{B}].pass(x)$ $D'', \delta' \blacktriangleright D'''$ by $\lfloor^{\texttt{D}}\vert_{\mathsf{Recv}}\rceil$
③	$G'' = \texttt{B} \to \texttt{C}.fwd(\mathbf{str});\ \mathbf{end}$ $k[\texttt{A}] = \mathbf{end}$ $k[\texttt{B}] = \oplus\texttt{C}.fwd(\mathbf{str}); \mathbf{end}$ $k[\texttt{C}] = \&\texttt{B}.fwd(\mathbf{str}); \mathbf{end}$	$C''' = k{:}\texttt{b}[\texttt{B}].x \to \texttt{c}[\texttt{C}].fwd(\ x\ )$	$D'''(\texttt{b}).x = "\texttt{ok}"$

Table 1: Example of message delivery on elements of interest of choreography C' (second column), its companion deployment D' (third column), and their typing environment (first column).

- middle column, the reduction of choreography C;
- right column, the main changes in D.

To ease the reading of the example, we highlight in grey the elements that have been changed by the reduction. To keep our example brief, we only report the reduction (sending and reception) of the first interaction in C, namely $k{:}\texttt{a}[\texttt{A}]."\texttt{ok}" \to \texttt{b}[\texttt{B}].pass(\ x\ )$.

In Table 1, row ① shows on the left column the original type G and the global type projection onto the local types of roles `A`, `B`, and `C`; in the next two columns we reported for completeness the reductions C' and D'. Next, we let the running choreography reduce, applying rules $\lfloor^{\texttt{C}}\vert_{\mathsf{Eq}}\rceil$, $\lfloor^{\texttt{C}}\vert_{\mathsf{Send}}\rceil$, and $\lfloor^{\texttt{D}}\vert_{\mathsf{Send}}\rceil$ to let process `a` deliver its message in the queue $k[\texttt{A}\rangle\texttt{B}]$ of process `b`. We also let G reduce to G' with rule $\lfloor^{\texttt{G}}\vert_{\mathsf{Send}}\rceil$. In row ② we report the result of the reductions. In the left column, G' indicates that role `A` has sent a message to `B`, which should consume it in the next step. This is also mirrored by the buffer projection, where the buffer typing $k[\texttt{A}\rangle\texttt{B}]$ is $\&\texttt{A}.pass$. The deployment D'' contains the actual message sent by `a` in the queue owned by `b`. The reduced choreography is still well-typed as, applying function $\texttt{bte}(\ \texttt{A}, D''(k[\texttt{A}\rangle\texttt{B}])\ )$ on the interested queue, we obtain the same local type of the buffer typing $k[\texttt{A}\rangle\texttt{B}]$. Finally, we let the running choreography and the global type reduce again, allowing process `b` to consume the message. We show the result of the reductions in row ③,

	Typing Environment	Choreography	Deployment
①	$G = \mathtt{A} \longrightarrow \mathtt{B}.first;\ \mathtt{A} \longrightarrow \mathtt{B}.second;\ \mathtt{end}$ $k[\mathtt{A}] = \oplus\mathtt{B}.first;\ \oplus\mathtt{B}.second;\ \mathtt{end}$ $k[\mathtt{B}] = \&\mathtt{A}.first;\ \&\mathtt{A}.second;\ \mathtt{end}$	$C = k : \mathsf{a}[\mathtt{A}] \longrightarrow \mathsf{b}[\mathtt{B}].first;\ k : \mathsf{a}[\mathtt{A}] \longrightarrow \mathsf{b}[\mathtt{B}].second$	D
Ⓐ	$G \to G'$ by $\lfloor^{\mathsf{G}}\!/_{\mathsf{Eq}}\rceil, \lfloor^{\mathsf{G}}\!/_{\mathsf{Send}}\rceil$i.e., $G \equiv_{\mathtt{G}} G_1 = \oplus_{\mathtt{A}}\mathtt{B}.first;\ \&_{\mathtt{A}}\mathtt{B}.first;\ \mathtt{A} \longrightarrow \mathtt{B}.second;\ \mathtt{end}$ $G_1 \to G_1'$ and $G_1' \equiv_{\mathtt{G}} G'$	$C \to C'$ by $\lfloor^{\mathsf{C}}\!/_{\mathsf{Eq}}\rceil, \lfloor^{\mathsf{C}}\!/_{\mathsf{Send}}\rceil$	$\delta = k : \mathsf{a}[\mathtt{A}] \longrightarrow \mathtt{B}.first$ $D, \delta \blacktriangleright D'$ by $\lfloor^{\mathsf{D}}\!/_{\mathsf{Send}}\rceil$
②	$G' = \mathtt{A}\rangle\mathtt{B}.first;\ \oplus_{\mathtt{B}}.second;\ \&_{\mathtt{A}}\mathtt{B}.second;\ \mathtt{end}$ $k[\mathtt{A}] = \oplus\mathtt{B}.second;\ \mathtt{end}$ $k[\mathtt{B}] = \&\mathtt{A}.first;\ \&\mathtt{A}.second;\ \mathtt{end}$ $k[\mathtt{A}\rangle\mathtt{B}] = \&\mathtt{A}.first();\ \mathtt{end}$	$C' = k : \mathtt{A} \longrightarrow \mathsf{b}[\mathtt{B}].first;\ k : \mathsf{a}[\mathtt{A}] \longrightarrow \mathtt{B}.second;\ k : \mathtt{A} \longrightarrow \mathsf{b}[\mathtt{B}].second$	$D'(k[\mathtt{A}\rangle\mathtt{B}]) = (first, \_)$
Ⓑ	$G' \to G''$ by $\lfloor^{\mathsf{G}}\!/_{\mathsf{Eq}}\rceil, \lfloor^{\mathsf{G}}\!/_{\mathsf{Send}}\rceil$i.e., $G' \simeq_{\mathtt{G}} G_2 = \oplus_{\mathtt{A}}\mathtt{B}.second;\ \mathtt{A}\rangle\mathtt{B}.first;\ \&_{\mathtt{A}}\mathtt{B}.second;\ \mathtt{end}$ $G_2 \to G_2'$ and $G_2' \simeq_{\mathtt{G}} G''$	$C' \to C''$ by $\lfloor^{\mathsf{C}}\!/_{\mathsf{Eq}}\rceil, \lfloor^{\mathsf{C}}\!/_{\mathsf{Send}}\rceil$	$\delta' = k : \mathtt{A} \longrightarrow \mathsf{b}[\mathtt{B}].second$ $D', \delta' \blacktriangleright D''$ by $\lfloor^{\mathsf{D}}\!/_{\mathsf{Send}}\rceil$
③	$G'' = \mathtt{A}\rangle\mathtt{B}.first;\ \mathtt{A}\rangle\mathtt{B}.second;\ \mathtt{end}$ $k[\mathtt{A}] = \mathtt{end}$ $k[\mathtt{B}] = \&\mathtt{A}.first;\ \&\mathtt{A}.second;\ \mathtt{end}$ $k[\mathtt{A}\rangle\mathtt{B}] = \&\mathtt{A}.first\ \&\mathtt{A}.second\ \mathtt{end}$	$C'' = k : \mathtt{A} \longrightarrow \mathsf{b}[\mathtt{B}].first;\ k : \mathtt{A} \longrightarrow \mathsf{b}[\mathtt{B}].second$	$D''(k[\mathtt{A}\rangle\mathtt{B}]) = (first, \_) :: (second, \_)$

Table 2: Example of asynchrony and effects on elements of interest of choreography C (second column), its companion deployment D (third column), and their typing environment (first column).

where in deployment D''' we can find that the value of the message has been assigned to the receiving variable x of b.

Example 5 (Asynchronous Message Delivery)**.** In this example, we consider a well-typed running choreography $\Gamma \vdash D, C$ where C and its correspondent reduced global type G are:

$$C = \begin{array}{l} k : \mathtt{a}[\mathtt{A}] \longrightarrow \mathtt{b}[\mathtt{B}].first(); \\ k : \mathtt{a}[\mathtt{A}] \longrightarrow \mathtt{b}[\mathtt{B}].second() \end{array} \qquad G = \begin{array}{l} \mathtt{A} \longrightarrow \mathtt{B}.first(\mathbf{unit}); \\ \mathtt{A} \longrightarrow \mathtt{B}.second(\mathbf{unit}); \\ \mathtt{end} \end{array}$$

We keep the same conventions on notation defined in the previous example with the addition of omitting round parenthesis for void values. We report in Table 2 a possible sequence of reduction. Following the previous example, we use row ① to summarise the status of (from left to right) the typing environment Γ, the choreography C, and its companion deployment D.

In row Ⓐ we report the main elements involved in the reduction. In the left-most cell of the raw, the global type G_1 is structurally equivalent ($\equiv_{\mathsf{G}}$) to G and that appears in rule $\lfloor^{\mathsf{G}}_{\mathsf{Eq}}\rceil$ to split the complete communication $\mathtt{A} \longrightarrow \mathtt{B}.first()$ into its equivalent $\oplus\mathtt{AB}.first(); \&_{\mathtt{A}}\mathtt{B}.first()$. Then G_1 reduces to G_1' with rule $\lfloor^{\mathsf{G}}_{\mathsf{Send}}\rceil$ and, as of rule $\lfloor^{\mathsf{G}}_{\mathsf{Eq}}\rceil$, we take G' as structurally equivalent to G_1', as shown in row ②, G' splits the complete communication $\mathtt{A} \longrightarrow \mathtt{B}.second()$ into its equivalent $\oplus\mathtt{AB}.second(); \&_{\mathtt{A}}\mathtt{B}.second()$. The reduction of C mirrors that of G: it splits the complete communication on operation $first$, consumes the sending, and finally splits the other complete communication on operation *second*, resulting in C' (row ②). The sending is applied on D which contains the related message in queue $k[\mathtt{A}\rangle\mathtt{B}]$ in its reductum D'.

Then, in row Ⓑ we allow the delivery of operation *second*. This illustrates how asynchrony works at both levels of global types and choreographies. As before, we start from the left-most cell in the row. First we consider G_2, which is swap-equivalent to G', after applying to it rule $\lfloor^{\mathsf{GS}}_{\mathsf{ChoBuf}}\rceil$. This brings on top the (*global choice*) on operation *second*. Then G_2 reduces to G_2' with rule $\lfloor^{\mathsf{G}}_{\mathsf{Send}}\rceil$ and, as of rule $\lfloor^{\mathsf{G}}_{\mathsf{Eq}}\rceil$, we take $G'' = G_1'$. The reduction on C', D' is similar to that of G'.

3.5. **Properties.** We close this section with the main guarantees of our type system.

First, our semantics preserves well-typedness:

Theorem 1 (Subject Reduction)**.** $\Gamma \vdash D, C$ and $D, C \to D', C'$ imply $\Gamma' \vdash D', C'$ for some Γ'.

We report in appendix B.1 the proof of Theorem 1.

We now relate Γ and Γ' to prove that the behaviours of sessions in a well-typed choreography follow their respective types. We denote $[\![G]\!]_k$ the projection of a global type G for a session k and let $[\![G]\!]_k$ be the set of local and buffer typings as obtained by the projection of G on each of its roles:

Definition 5 (Global Type Projection)**.**

$$[\![G]\!]_k = \{\ k[\mathtt{A}] \colon [\![G]\!]_{\mathtt{A}} \mid \mathtt{A} \in \mathbf{roles}(G)\ \}, \{\ k[\mathtt{A}\rangle\mathtt{B}] \colon [\![G]\!]^{\mathtt{A}}_{\mathtt{B}} \mid \mathtt{A} \in \mathbf{roles}(G), \mathtt{B} \in \mathbf{roles}(G) \setminus \{\mathtt{A}\}\ \}$$

We say that a reduction is "at session k" if it is obtained by consuming a communication term for session k (as in [HYC08]), and we write $k \notin \Gamma$ when k does not appear in any local typing in Γ. Then we have:

Theorem 2 (Session Fidelity)**.** Let $\Gamma, \Gamma_k \vdash D, C$, $k \notin \Gamma$. Then, $D, C \to D', C'$ with a redex at session k implies that, for some G and Γ', $k \notin \Gamma'$, (i) $\Gamma_k \subseteq [\![G]\!]_k$, (ii) $G \to G'$, (iii) $\Gamma_k' \subseteq [\![G']\!]_k$, and (iv) $\Gamma', \Gamma_k' \vdash D', C'$.

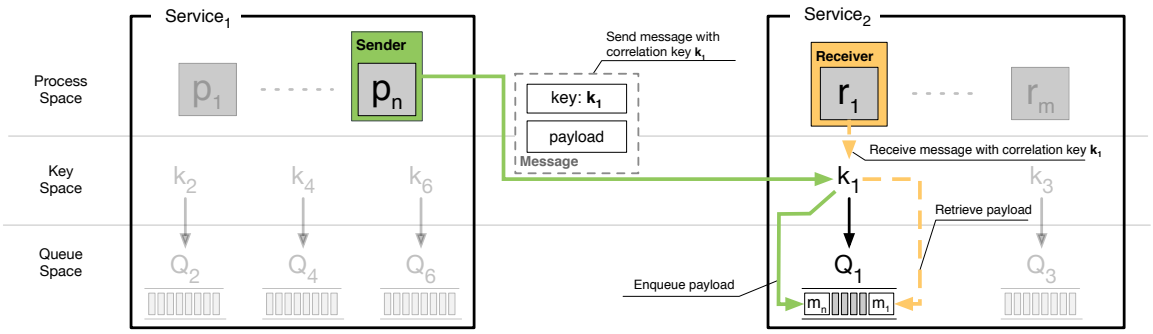


Figure 15: Depiction of correlation-based message exchange in SOC.

Theorem 2 states that all communications on sessions follow the expected protocols (Γ' may differ from Γ for the instantiation of a new variable). The proof of Theorem 2 is reported in appendix B.1.

Finally, we present the definition of the coherence predicate **co**:

Definition 6 (Coherence)**.** $\mathbf{co}(\Gamma)$ holds iff $\forall\, k \in \Gamma\ ,\ \exists\, G$ s.t.

- $\tilde{l} : G\langle \mathtt{A} | \tilde{\mathtt{B}} | \tilde{\mathtt{C}} \rangle \in \Gamma \ \wedge\ \tilde{\mathtt{C}} = \tilde{\mathtt{B}}$ and
- $\forall\, \mathtt{A} \in \mathbf{roles}(G),\ k[\mathtt{A}] : T \in \Gamma \quad \wedge \quad T = [\![G]\!]_{\mathtt{A}} \quad \wedge \quad \forall\, \mathtt{B} \in \mathbf{roles}(G) \setminus \{\mathtt{A}\},\ \Gamma \vdash k[\mathtt{B}\rangle \mathtt{A}] : [\![G]\!]^{\mathtt{B}}_{\mathtt{A}}$

Coherence extends partial coherence to check that *i*) all needed services to start new sessions are present and *ii*) all the roles in every open session are correctly implemented by some processes.

Coherent and well-typed systems are deadlock-free, as stated by Theorem 3.

Theorem 3 (Deadlock-freedom)**.** $\Gamma \vdash D, C$ and $\mathbf{co}(\Gamma)$ imply that either (i) $C \equiv_{\mathtt{C}} \mathbf{0}$ or (ii) there exist D' and C' such that $D, C \to D', C'$.

We report the proof of Theorem 3 in appendix B.2.

4. Backend Choreographies

We now present *Backend Choreographies* (BC). The syntax of programs in BC is the same as that of FC. Also the two semantics are close, except that FC models communication over named channels while BC formalises message exchange based on message correlation, as found in Service-Oriented Computing (SOC) [OAS07]. Formally, thanks to the separation between choreographic programs and deployments presented in FC, we can let FC and BC share a large fragment of semantics rules, while the significant differences between the two semantics of message exchange—name-based for FC, correlation-based for BC—are isolated within their specific deployments and deployment transitions.

The structure and semantics of the Backend deployments $\mathbb{D}$ is one of our major contributions: it formalises, at the level of choreographies, how to implement sessions using the communication mechanism of message correlation typical of SOC systems.

In the following, we first informally introduce correlation-based message exchange, then we formalise data and queues in (the deployment of the) Backend Choreographies, and finally we formalise correlation-based message exchange in the semantics of deployment transitions in BC.

4.1. **Correlation-based Communication.** Processes in SOC run within services and communicate asynchronously. To realise asynchronous communication, services provide an unbound number of first-in-first-out message queues that processes interact with. The interaction happens from processes that associate a message insertion/retrieval action with a *correlation key*, which uniquely identifies the queue subject of the action. Concretely, a correlation key corresponds to a set of data that the service associates to a specific queue.

Processes retrieve messages from the queues of their enclosing service. This is represented in (the right side of) Figure 15 by process r_1, which wants to consume a message received on queue Q_1, associated to the correlation key k_1. The request is satisfied by the service, which delivers message m_1 to r_1, also removing the interested message from the head of queue Q_1. The complement of the action above is message insertion. Any process (within the queue-enclosing service and remote) can insert data into a queue by sending a message to the service owning the queue. That message must associate the payload to be inserted with the correlation key that identifies the queue within the service. Concretely, when a service receives a message from the network, it inspects its content, looking for a valid correlation key, i.e., one that points to any of its queues. If a queue can be found, the message is enqueued in its tail. In Figure 15, this is represented by data k_1 marked by the attribute key in the message sent by process p_n (of $\mathsf{Service}_1$) to $\mathsf{Service}_2$. At reception, $\mathsf{Service}_2$:

(1) checks for the presence of the attribute key;
(2) extracts the corresponding key k_1;
(3) finds the queue Q_1, pointed by k_1;
(4) enqueues the received payload in Q_1 as message m_n.

As depicted in Figure 15, messages in SOC contain correlation keys as either part their payload or in some separate header. As in [MC11], also here we abstract from such details. To summarise, two processes can communicate over correlation-based messaging if: *i*) the sender knows the (location of the) service where the addressee is running and *ii*) the sender and the addressee know the key corresponding to a queue in the addressee service. After having presented the mechanism of correlation for message exchange, we can proceed to explain how we model SOC systems in BC.

Data and Process state. Data in SOC is structured following a tree-like format, e.g., XML [BPSM$^+$98] or JSON [B$^+$14]. In BC, we use trees to represent both the payload of messages and the state of running processes (as in, e.g., BPEL [OAS07] and Jolie [MGZ14]).

Formally, we consider rooted trees $t \in \mathcal{T}$, where $\mathcal{T} = \mathit{Val} \cup \mathcal{L} \cup \mathit{Set}(\mathit{Lab} \times \mathcal{T})$ and

$$t ::= \quad v \quad | \quad l \quad | \quad \{\ \underline{x_1} : t_1, \ldots, \underline{x_n} : t_n\ \}$$

i.e., a tree (node) is either a value v, a location l, or a set of ordered pairs of edge labels $\underline{x}, \underline{y} \in \mathit{Lab}$ and tree nodes. We assume tree nodes to be values or locations only in leaves. Now we can define BC variables as paths on trees (the latter, we remind, represents state of processes) as sequences of labels $x, y \in \mathit{Seq}(\mathit{Lab})$ such that $x ::= \underline{x}.x \mid \varepsilon$, where ε is the empty sequence, which we often omit for brevity. When writing paths in their extended form, e.g., $\underline{x}.\underline{y}.\underline{z}.\varepsilon$, we often use the abbreviation $\underline{x.y.z}$.

In addition, we define two operators to handle trees: path application and deep copy. The path-application operator $x(t)$ is used to access the sub-nodes pointed by path x in tree t. Intuitively, $x(t)$ returns either the value, the location or the sub-tree pointed by path x in

t. If x is not present in t, $x(t)$ returns an empty set of ordered pairs label-tree. Formally,

$$\underline{x}.x(t) = \begin{cases} x(\ \underline{x}.\varepsilon(t)\ ) & \text{if } x \neq \varepsilon \\ t' & \text{if } x = \varepsilon \text{ and } t = \{\ \underline{x}\colon t', \ldots, \underline{x_n}\colon t_n\ \} \\ \varnothing & \text{otherwise} \end{cases}$$

The deep-copy operator $t \triangleleft (\, x, t' \,)$ is a (total) replacement operator that returns the tree obtained by replacing in t the sub-tree rooted in $x(t)$ with t'. If x is not present in t, $t \triangleleft (\, x, t' \,)$ adds the smallest chain of empty nodes to t such that it stores t' under path x. Formally,

$$t \triangleleft (\, \underline{x}.x, t' \,) = \begin{cases} \varnothing \triangleleft (\, \underline{x}.x, t' \,) & \text{if } t \in \mathit{Val} \cup \mathcal{L} \\ (\ t \ \setminus\ \{\ \underline{x}\colon \underline{x}(t)\ \}\ ) \cup \{\ \underline{x}\colon t'\ \} & \text{if } t \notin \mathit{Val} \cup \mathcal{L} \text{ and } x = \varepsilon \\ (\ t \ \setminus\ \{\ \underline{x}\colon \underline{x}(t)\ \}\ ) \cup \{\ \underline{x}\colon \underline{x}(t) \triangleleft (\, x, t' \,)\ \} & \text{otherwise} \end{cases}$$

4.2. **Backend Deployments, Transition Rules, and BC Semantics.** In addition to the convention of using the terms "Frontend choreography/deployment" to indicate a Frontend Choreographies program/deployment, in the reminder we adopt the same convention for "Backend choreographies" and "Backend deployments". We use the term "choreography" alone, when the context makes it clear if we refer to Backend or Frontend ones.

We can now define the notion of deployment for BC, denoted $\mathbb{D}$, which includes:

- the locality of processes;
- queues, pointed by a combination of a location and a correlation key;
- the state of processes.

Formally, $\mathbb{D}$ is an overloaded partial function defined by cases as the sum of three partial functions $g_l : \mathcal{L} \rightharpoonup \mathit{Set}(\mathcal{P})$, $g_m : (\mathcal{L} \times \mathcal{T}) \rightharpoonup \mathit{Seq}(\mathcal{O} \times \mathcal{T})$, and $g_s : \mathcal{P} \rightharpoonup \mathcal{T}$. The domains and co-domains of the functions are disjoint, hence:

$$\mathbb{D}(z) = \begin{cases} g_l(z) & \text{if } z \in \mathcal{L}, \\ g_m(z) & \text{if } z \in (\mathcal{L} \times \mathcal{T}), \\ g_s(z) & \text{otherwise} \end{cases}$$

Function g_l maps a location to the set of processes running in the service at that location. Given a location l, we read $\mathbb{D}(l) = \{\mathsf{p}_1, \ldots, \mathsf{p}_n\}$ as "the processes $\mathsf{p}_1, \ldots, \mathsf{p}_n$ are running at the location l" (we assume each process p to run at most at one location). Function g_m maps a couple location-tree to a message queue. This reflects message correlation as informally described above, where a queue resides in a service, i.e., at its location, and is pointed by a correlation key. Given a couple $l : t$, we read $\mathbb{D}(l : t) = \tilde{m}$ as "the queue $\tilde{m}$ resides in a service at location l and is pointed by correlation key t". The queue $\tilde{m}$ is a sequence of messages $\tilde{m} ::= m_1 :: \cdots :: m_n \mid \varepsilon$ and a message of the queue is $m ::= (o, t)$, where t is the payload of the message and o is the operation on which the message was received. Pairing operation labels with message payloads is typical of SOC implementations in general. Indeed, while not essential for the correct delivery of messages, operation labels are used by processes to program external choices (for instance, a process expecting to receive a message on either of two mutually-exclusive operations, e.g., to continue or exit a loop). The case applies also to BC, where we preserve the association between payload and operations (o, t)—similarly to FC (o, v). Function g_s maps a process to its local state. Given a process p, the notation $\mathbb{D}(\mathsf{p}) = t$ means that p has local state t.

$$\frac{\mathtt{p} \in \mathbb{D}(l) \qquad \mathbb{D}, \mathbf{sup}(\ \{\ l.\mathtt{p}[\mathtt{A}], \widetilde{l.\mathtt{q}.[\mathtt{B}]}\ \}\ ) \blacktriangleright \mathbb{D}'}{\mathbb{D}, \mathtt{start}\ k : \mathtt{p}[\mathtt{A}] \leftrightarrow \widetilde{l.\mathtt{q}[\mathtt{B}]} \blacktriangleright \mathbb{D}'}\ \lfloor\!\lceil^{\mathbb{p}}|_{\mathsf{Start}}\rceil\!\rfloor$$

$$\frac{\begin{array}{c}\mathtt{q}_1 \in \mathbb{D}\ \textcircled{1} \qquad j \in I \setminus \{i\} \qquad \underline{B_i.l}(t) = l_i \textcircled{2} \qquad \underline{B_i.B_j}(t) = t_{ij} \textcircled{3} \qquad l_j : t_{ij} \notin \mathbb{D} \textcircled{4} \\ \mathbb{D}' = \mathbb{D}[\ l_i \mapsto \mathbb{D}(l_i) \cup \{\mathtt{q}_i\}\ ] \textcircled{5} \quad \mathbb{D}'' = \mathbb{D}'[\ l_i : t_{ij} \mapsto \varepsilon\ ] \textcircled{6} \quad \mathbb{D}''' = \mathbb{D}''[\ \mathtt{q}_1 \mapsto \mathbb{D}''(\mathtt{q}_1) \triangleleft (\underline{k}, t)\ ] \textcircled{7}\end{array}}{\mathbb{D}, \mathbf{sup}(\ \{\ l_i.\mathtt{q}_i[\mathtt{B}_i]\ \}_{i \in I}\ ) \blacktriangleright \mathbb{D}'''[\ \mathtt{q}_h \mapsto \{\underline{k} : t\}\ ]_{h \in \{2,\ldots,n\}} \textcircled{8}}\ \lfloor\!\lceil^{\mathbb{p}}|_{\mathsf{Sup}}\rceil\!\rfloor$$

$$\frac{l = \underline{k.B.l}(\ \mathbb{D}(\mathtt{p})\ ) \qquad t_c = \underline{k.A.B}(\ \mathbb{D}(\mathtt{p})\ ) \quad t_m = \mathbf{eval}(e, \mathbb{D}(\mathtt{p}))}{\mathbb{D}, k : \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o \blacktriangleright \mathbb{D}[\ l : t_c \mapsto \mathbb{D}(l : t_c) :: (o, t_m)\ ]}\ \lfloor\!\lceil^{\mathbb{p}}|_{\mathsf{Send}}\rceil\!\rfloor$$

$$\frac{t_c = \underline{k.A.B}(\ \mathbb{D}(\mathtt{q})\ ) \qquad \mathtt{q} \in \mathbb{D}(l) \qquad \mathbb{D}(l : t_c) = (o, t_m) :: \tilde{m} \qquad \mathbb{D}' = \mathbb{D}[\ l : t_c \mapsto \tilde{m}\ ]}{\mathbb{D}, k : \mathtt{A} \longrightarrow \mathtt{q}[\mathtt{B}].o(x) \blacktriangleright \mathbb{D}'[\ \mathtt{q} \mapsto \mathbb{D}'(\mathtt{q}) \triangleleft (\underline{x}, t_m)\ ]}\ \lfloor\!\lceil^{\mathbb{p}}|_{\mathsf{Recv}}\rceil\!\rfloor$$

Figure 16: Backend Choreographies — Deployment transitions.

Backend Deployment Transitions. In BC, we replace the deployment transitions of FC with the rules defining $\mathbb{D}, \delta \blacktriangleright \mathbb{D}'$, reported in Figure 16. We comment them in the following.

Rule $\lfloor\!\lceil^{\mathbb{p}}|_{\mathsf{Start}}\rceil\!\rfloor$ simply retrieves the location of process $\mathtt{p}$ (the one that requested the creation of session k) and uses rule $\lfloor\!\lceil^{\mathbb{p}}|_{\mathsf{Sup}}\rceil\!\rfloor$ to obtain the new deployment $\mathbb{D}'$ that supports interactions over session k. Namely, $\mathbb{D}'$ is an updated version of $\mathbb{D}$ with: *i*) the newly created processes for session k and *ii*) the queues used by the new processes and $\mathtt{p}$ to communicate over session k. In addition, in $\mathbb{D}'$, *iii*) the new processes and $\mathtt{p}$ contain in their states a structure, rooted in $\underline{k}$ and called *session descriptor*, that includes all the information (correlation keys and the locations of all involved processes) to support correlation-based communication in session k. Formally, this is done by rule $\lfloor\!\lceil^{\mathbb{p}}|_{\mathsf{Sup}}\rceil\!\rfloor$ where we ① retrieve the starter process, here called $\mathtt{q}_1$, which is the only process already present in $\mathbb{D}$. Then, given a tree t, we ensure it is a proper session descriptor for session k, i.e., that:

② t contains the location l_i of each process, represented by its role in the session $\mathtt{B}_i$, under path $\underline{B_i.l}$;
③ t contains a correlation key t_{ij} for each ordered couple of roles $\mathtt{B}_i$, $\mathtt{B}_j$ under path $\underline{B_i.B_j}$, such that ④ there is no queue in $\mathbb{D}$ at location l_j pointed by correlation key t_{ij};

Finally, we assemble the update of $\mathbb{D}$ in four steps:

⑤ first, we obtain $\mathbb{D}'$ by adding in $\mathbb{D}$ the processes $\mathtt{q}_2, \ldots, \mathtt{q}_n$ at their respective locations;
⑥ second, we obtain $\mathbb{D}''$ by adding to $\mathbb{D}'$ an empty queue ε for each couple $l_j : t_{ij}$;
⑦ third, we obtain $\mathbb{D}'''$ from $\mathbb{D}''$ by storing in the state of (the starter) process $\mathtt{q}_1$ the session descriptor t under path $\underline{k}$;
⑧ finally, we update $\mathbb{D}'''$ such that each new created process $(\mathtt{q}_2, \ldots, \mathtt{q}_n)$ has in its state just the session descriptor t rooted under path $\underline{k}$.

We deliberately define in $\lfloor\!\lceil^{\mathbb{p}}|_{\mathsf{Sup}}\rceil\!\rfloor$ the session descriptor t with a set of constrains on data, rather than with a procedure to obtain the data for correlation. In this way, our model is general enough to capture different methodologies for creating correlation keys (e.g., UUIDs or API keys).

Rule $\lfloor\!\lceil^{\mathbb{p}}|_{\mathsf{Send}}\rceil\!\rfloor$ models the sending of a message. We comment the premises. From left to right, the first gets the location l of the receiver $\mathtt{B}$ from the state of the sender $\mathtt{p}$; the

second retrieves the correlation key in the state of p (playing role A) to send messages to role B; the third evaluates the expression e of the sender p using its local state to get a value t_m. Function **eval** evaluates expressions in a process state, traversing its paths and performing local computation. We highlight that, since in BC we preserve the syntax of Fronted Choreographies, we make two assumptions: that expressions (e.g., e in $[\![\mathbb{p}|_{\mathsf{Send}}]\!]$) are defined on *Var*iables and that **eval** in BC automatically maps variables x, y, z into the respective paths $\underline{x}.\varepsilon$, $\underline{y}.\varepsilon$, and $\underline{z}.\varepsilon$, used to access process states in $\mathbb{D}$. Finally, in the conclusion of the rule, we add the message (o, t_m) in the queue pointed by $l : t_c$ that we found via correlation.

Rule $[\![\mathbb{p}|_{\mathsf{Recv}}]\!]$ models a reception. From left to right, the first premise finds the correlation key t_c for the queue that q (playing role B) should use to receive from A in session k. The second premise retrieves the location l of q. The third accesses the queue pointed by $l : t_c$ and retrieves message (o, t_m). The last premise updates $\mathbb{D}$ to $\mathbb{D}'$ removing (o, t_m) from the interested queue. Dually to rule $[\![\mathbb{p}|_{\mathsf{Send}}]\!]$, where **eval** maps variables into paths, in the conclusion of rule $[\![\mathbb{p}|_{\mathsf{Recv}}]\!]$ we map x, i.e., the intended variable that should store the payload t_m in the state of q, into path $\underline{x}.\varepsilon$.

4.3. **Encoding Frontend Choreographies to Backend Choreographies and Properties.** Now that we presented Backend Choreographies, we can proceed with our main intent of defining a compilation procedure from high-level FC programs to low-level services. Here, we tackle the transition from FC programs to their intermediate representation toward SOC systems as Backend Choreographies. Specifically, we translate FC programs that use the abstract mechanism of communication over names, into BC programs that use the concrete mechanism of correlation-based communication. We prove our translation correct, i.e., that our encoding guarantees an operational correspondence between the semantics of a Frontend choreography and its Backend encoding.

Formally, since choreographies in BC have the same syntax of FC ones, we can translate FC runtime terms D, C to BC runtime terms by encoding the FC deployment D to an appropriate Backend deployment. Notably, BC deployments contain more information wrt FC deployments. We extract these data from Γ, the typing environment of D, C.

Definition 7 (Encoding FC in BC)**.** Let $\Gamma \vdash D, C$ and $\langle\!\langle D \rangle\!\rangle^{\Gamma}$ be defined by the algorithm in Figure 17. Then, the Backend encoding of D, C is defined as $\langle\!\langle D \rangle\!\rangle^{\Gamma}, C$.

What the algorithm $\langle\!\langle D \rangle\!\rangle^{\Gamma}$ does is:

(1) include in $\mathbb{D}$ all (located) processes present in D (and typed in Γ);
(2) translate the state (i.e., the association *Var*iable-*Val*ue) of each process in D to its correspondent tree-shaped state in $\mathbb{D}$;
(3) for each ongoing session in D, set the proper correlation keys and queues in $\mathbb{D}$ and, for each queue, import and translate its related messages.

More precisely, in the algorithm defined in Figure 17 at Line 1, we create a new Backend deployment $\mathbb{D}$ and assign to it the totally undefined function ($\varnothing$); $\mathbb{D}$ is an empty Backend deployment that will be later refined via the updates on $\mathbb{D}$ at Lines 3–16. Then,

- *Lines 3–5*, for each located process $\mathsf{p}@l$ in Γ, we update the locations of $\mathbb{D}$ to contain p at location l (Line 4) and we include process p in $\mathbb{D}$, associating to it an empty state, i.e., the empty tree $\varnothing$ (Line 5);

```
1   ⟪D⟫^Γ  =  𝔻 := ∅

3             foreach p@l in Γ
4                 𝔻 := 𝔻 [ l ↦ 𝔻(l) ∪ {p} ]
5                 𝔻 := 𝔻 [ p ↦ ∅ ]

7             foreach p.x: U in Γ
8                 𝔻 := 𝔻[ p ↦ 𝔻(p) ◁ ( x, D(p)(x) ) ]

10            foreach { p: k[A] q: k[B], q@l } in Γ
11                t := fresh(𝔻, l)
12                𝔻 := 𝔻[ l: t ↦ D(k[A⟩B]) ]
13                𝔻 := 𝔻[ p ↦ 𝔻(p) ◁ ( k.A.B, t ) ]
14                𝔻 := 𝔻[ q ↦ 𝔻(q) ◁ ( k.A.B, t ) ]
15                𝔻 := 𝔻[ p ↦ 𝔻(p) ◁ ( k.B.l, l ) ]
16                𝔻 := 𝔻[ q ↦ 𝔻(q) ◁ ( k.B.l, l ) ]

18            return 𝔻
```

Figure 17: Encoding Algorithm from Frontend to Backend Deployments.

- *Lines 7–8*, for each variable x (typed in Γ) of a process $\mathtt{p}$, we update the state of process $\mathtt{p}$ in $\mathbb{D}$ to include the association of x to its value in the state $D(\mathtt{p})$. As done in rules $[\![^{\mathbb{P}}|_{\mathsf{Send}}]$ and $[\![^{\mathbb{P}}|_{\mathsf{Recv}}]$, we map FC variables $x \in \mathit{Var}$ into BC paths $\underline{x} \in \mathit{Seq}(\mathit{Lab})$;
- *Lines 10–16*, follow the same principles to support correlation-based exchanges as formalised in rule $[\![^{\mathbb{P}}|_{\mathsf{Sup}}]$; for each couple of processes $\mathtt{p}, \mathtt{q}$, respectively playing distinct roles $\mathtt{A}$ and $\mathtt{B}$ in a session k, with $\mathtt{q}$ located at l:
  - *Line 11*, we obtain a fresh correlation key t with auxiliary function **fresh**. The latter takes deployment $\mathbb{D}$ and location l as input and returns a correlation key which is fresh among the keys associated to location l in $\mathbb{D}$. Formally t is such that $l\colon t \notin \mathbf{dom}(\mathbb{D})$;
  - *Line 12*, we associate correlation key t with location l in $\mathbb{D}$ and make it point the corresponding queue of messages from role $\mathtt{A}$ to role $\mathtt{B}$ in D (accessed with triple $k[\mathtt{A}\rangle\mathtt{B}]$). Note that we can directly copy message queues from D into $\mathbb{D}$. Indeed, while message queues in D and $\mathbb{D}$ are respectively of type $\mathit{Seq}(\mathcal{O} \times \mathit{Val})$ and $\mathit{Seq}(\mathcal{O} \times \mathcal{T})$, by definition $\mathcal{T}$ subsumes Val;
  - *Line 13–14*, we include in the state of processes $\mathtt{p}$ (Line 13) and $\mathtt{q}$ (Line 14) correlation key t, storing it under path $\underline{k.A.B}$;
  - *Line 15–16*, we include in the state of processes $\mathtt{p}$ (Line 15) and $\mathtt{q}$ (Line 16) the location of role $\mathtt{B}$ under path $\underline{k.B.l}$.

  The encoding from FC to BC guarantees a strong operational correspondence.

Theorem 4 (Operational Correspondence (FC $\leftrightarrow$ BC))**.** Let $\Gamma \vdash D, C$. Then:

(1) (Completeness) $D, C \to D', C'$ implies $\langle\!\langle D \rangle\!\rangle^{\Gamma}, C \to \langle\!\langle D' \rangle\!\rangle^{\Gamma'}, C'$ for some Γ' s.t. $\Gamma' \vdash D', C'$;
(2) (Soundness) $\langle\!\langle D \rangle\!\rangle^{\Gamma}, C \to \mathbb{D}, C'$ implies $D, C \to D', C'$ and $\mathbb{D} = \langle\!\langle D' \rangle\!\rangle^{\Gamma'}$ for some Γ' s.t. $\Gamma' \vdash D', C'$.

Proof Sketch of Theorem 4. We sketch the proof of Theorem 4, analysing its two parts: *(Completeness)* and *(Soundness)*. The proof of Completeness is by induction on the derivation of D, C. The main observation is that the encoded system $\langle\!\langle D \rangle\!\rangle^{\Gamma}, C$ mimics D, C by applying the same semantic rules on C and the corresponding deployment transitions (e.g., respectively defied by rules $[\mathsf{P}|_{\mathsf{Send}}]$ and $[\mathbb{P}|_{\mathsf{Send}}]$). Let $\mathbb{D}'$ be the Backend environment obtained from the reduction $\langle\!\langle D \rangle\!\rangle^{\Gamma}, C \to \mathbb{D}, C'$ on rule $[\mathsf{C}|_{\mathsf{Start}}]$. Since Figure 17 and rule $[\mathbb{P}|_{\mathsf{Sup}}]$ (on which rule $[\mathbb{P}|_{\mathsf{Start}}]$ relies) implement the same principles of $\langle\!\langle D \rangle\!\rangle^{\Gamma}$, we know that $\underline{k.A.B}(\mathbb{D})$ and $\underline{k.A.B}(\ \langle\!\langle D' \rangle\!\rangle^{\Gamma'}\ )$ will be the same, except possibly for i) the location of processes and ii) trees of correlation keys corresponding to the same paths. Concretely, item i) derives from the fact that Γ and Γ' can disagree on the location of the same process p, and item ii) is caused by the random generation of correlation keys, for which, considering a correlation key rooted in $\underline{k.A.B}$ of a process p, the trees obtained from $\underline{k.A.B}(\ \mathbb{D}(\mathsf{p})\ )$ and $\underline{k.A.B}(\ \langle\!\langle D' \rangle\!\rangle^{\Gamma'}(\mathsf{p})\ )$ may differ. However, these discrepancies do not constitute a problem, since both locations and correlation keys are used consistently in their respective deployments, which are thus interchangeable.

We can extend the same observation also for Soundness, which is proved by induction on the derivation of $\langle\!\langle D \rangle\!\rangle^{\Gamma}, C$. □

5. Dynamic Correlation Calculus

In this section, we introduce the Dynamic Correlation Calculus (DCC), the target language of our compilation.

DCC is a straightforward extension of a previous proposal called Correlation Calculus [MC11], which is a process calculus that formalises service-oriented, correlation-based communications. Indeed, while we started this work considering CC as the target language of our compilation, we found it limited for our purposes: in CC each process receives from only one message queue, while we need processes to be able to select receptions from multiple queues (as in our Backend deployments). Hence, we defined DCC as an extension of CC with the support for the dynamic creation and selection of queues in processes.

We deem DCC a choice that fits the practical motivations of this work thanks to its closeness to the implementation languages/frameworks listed below, which casts a good outlook on the affordability of future implementations of our theoretical results. First, CC formalises the semantics of message exchange of Jolie, a service-oriented programming language [MGZ14]. Thus CC specifications are directly translatable into Jolie executable programs. This is not the case for our DCC code, as Jolie lacks the primitives to let processes create and select queues. Fortunately, the distance between CC and DCC is close enough so that supporting the extended features in Jolie would entail minimal change, i.e., the inclusion of the syntactic primitives for queue creation and selection[3] and the implementation of the associated semantics—a direct extension of the one-process-one-queue semantics of the current implementation. Second, the service-oriented language BPEL [OAS07] lets processes create and receive from multiple queues, making DCC a useful reference for BPEL-based implementations. Third, besides service-oriented languages, DCC abstracts real-world message-exchange models where processes can interact with multiple message queues. This is the case, e.g., for some versions of the actor model [Agh85] where one actor can be associated with many queues/mailboxes [HO07] and in some popular message-exchange

[3]The (*newque*) and `from` and `to` particles in Figure 18.

Services	S	$::=$	$\langle \mathfrak{B}, P, M \rangle_l$	(srv)
		$\mid$	$S \mid S'$	(net)
Start Behaviour	$\mathfrak{B}$	$::=$	$!(x); B$	$(acpt)$
		$\mid$	$\mathbf{0}$	$(inact)$
Processes	P	$::=$	$B \cdot t$	$(prcs)$
		$\mid$	$P \mid P'$	(par)

Behaviours

$B ::=$	$?@e_1(e_2); B$	$(reqst)$	$\mid$	$\sum_i [o_i(x_i)\ \texttt{from}\ e]\ \{B_i\}$	$(choice)$
$\mid$	$o(x)\ \texttt{from}\ e; B$	$(input)$	$\mid$	$o@e_1(e_2)\ \texttt{to}\ e_3; B$	$(output)$
$\mid$	$\texttt{def}\ X = B'\ \texttt{in}\ B$	(def)	$\mid$	$\texttt{if}\ e\ \{B_1\}\ \texttt{else}\ \{B_2\}$	$(cond)$
$\mid$	$\nu\rangle x; B$	$(newque)$	$\mid$	$\mathbf{0}$	$(inact)$
$\mid$	$x = e; B$	$(assign)$	$\mid$	X	$(call)$

Figure 18: Dynamic Correlation Calculus — Syntax.

middlewares [Vin06, VW12], which are suitable alternatives to the implementation targets above.

Syntax. We now introduce the syntax of DCC, which we report in Figure 18 and which comprises two layers: *Services*, ranged over by S, and *Processes*, ranged over by P.

In the syntax of services, term (srv) is a service, located at l, with a *Start Behaviour* $\mathfrak{B}$ and running processes P (both described later on) and a queue map M. The queue map is a partial function $M : \mathcal{T} \rightharpoonup Seq(\mathcal{O} \times \mathcal{T})$ that, similarly to function g_m in Backend deployments, associates a correlation key t to a message queue. We model messages like in BC where a message is a couple (o, t), o being the operation on which the message has been received, and t the payload of the message. Services are composed in parallel in term (net).

Concerning behaviours, in DCC we distinguish between start behaviours and process behaviours. Process behaviours define the general behaviour of processes in DCC, as described later on. Start behaviours use term $!(x)$ to indicate the availability of a service to generate new local processes on request. At runtime, the start behaviour $\mathfrak{B}$ of a service is activated by the reception of a dedicated message that triggers the creation of a new process. The new process has (process) behaviour B, which is defined in $\mathfrak{B}$ after the $!(x)$ term, and an empty state. The content of the request message is stored in the state of the newly created process, under the bound path x. As in Backend Choreographies, also in DCC paths are used to access process states.

Finally, processes (prc) in DCC consists of a behaviour B and a state t and can be composed in parallel (par). Process states t are trees and, in *Behaviours*, operations (o), procedures (X), paths (x), and expressions (e, evaluated at runtime on the state of the enclosing process) are all the same as defined for Backend Choreographies (§ 4.1). Terms $(input)$ and $(output)$ model communications. In $(input)$, the process stores under x a message `from` the head of the queue correlating with e and received on operation o. Dually, term $(output)$ sends a message on operation o. The three expressions in the term define: e_1, the location of the service where the addressee is running; e_2, the content of the message; e_3, the key that correlates with the receiving queue of the addressee. Term $(choice)$ is an $(input)$-choice: when one of the inputs can receive a message from the queue correlating with

$$\frac{t' = \mathbf{eval}(x,t)}{x = e\ ;B\cdot t \quad\to\quad B\cdot t \triangleleft (\,x,t'\,)}\ \lfloor^{\mathsf{DCC}}|_{\mathsf{Assign}}\rceil \qquad \frac{B\cdot t \to B'\cdot t'}{\mathtt{def}\ X = B_1\ \mathtt{in}\ B\cdot t \quad\to\quad \mathtt{def}\ X = B_1\ \mathtt{in}\ B'\cdot t'}\ \lfloor^{\mathsf{DCC}}|_{\mathsf{Ctx}}\rceil$$

$$\frac{i = 1 \text{ if } \mathbf{eval}(e,t) = \text{true}, i = 2 \text{ otherwise}}{\mathtt{if}\ e\ \{B_1\}\ \mathtt{else}\ \{B_2\}\cdot t \quad\to\quad B_i\cdot t}\ \lfloor^{\mathsf{DCC}}|_{\mathsf{Cond}}\rceil \qquad \frac{P \equiv_{\mathtt{D}} P_1 \mid P_2 \quad P_1 \to P_1' \quad P_1' \mid P_2 \equiv_{\mathtt{D}} P'}{\langle \mathfrak{B},\ P,\ M\rangle_l \quad\to\quad \langle \mathfrak{B},\ P',\ M\rangle_l}\ \lfloor^{\mathsf{DCC}}|_{\mathsf{PEq}}\rceil$$

$$\frac{B = \nu\rangle x; B \quad t_c \notin \mathbf{dom}(M) \quad M' = M[t_c \mapsto \varepsilon]}{\langle \mathfrak{B},\ B\cdot t \mid P,\ M\rangle_l \quad\to\quad \langle \mathfrak{B},\ B\cdot t \triangleleft (\,x,t_c\,) \mid P,\ M'\rangle_l}\ \lfloor^{\mathsf{DCC}}|_{\mathsf{Newque}}\rceil$$

$$\frac{\begin{array}{c} B \in \{\ o_j(x_j)\ \mathtt{from}\ e; B_j\ ,\ \sum_{i\in I}\,[o_i(x_i)\ \mathtt{from}\ e]\,\{B_i\}\ \} \\ j \in I \qquad t_c = \mathbf{eval}(e,t) \qquad M(t_c) = (o_j, t_m) :: \tilde{m} \end{array}}{\langle \mathfrak{B},\ B\cdot t \mid P,\ M\rangle_l \quad\to\quad \langle \mathfrak{B},\ B_j\cdot t \triangleleft (\,x_j, t_m\,) \mid P,\ M[t_c \mapsto \tilde{m}]\rangle_l}\ \lfloor^{\mathsf{DCC}}|_{\mathsf{Recv}}\rceil$$

$$\frac{\begin{array}{c} B = o@e_1(e_2)\ \mathtt{to}\ e_3; B' \qquad \mathbf{eval}(e_1,t) = l \\ \mathbf{eval}(e_3,t) = t_c \qquad \mathbf{eval}(e_2,t) = t_m \qquad t_c \in \mathbf{dom}(M) \end{array}}{\langle \mathfrak{B},\ B\cdot t \mid P, M\rangle_l \quad\to\quad \langle \mathfrak{B},\ B'\cdot t \mid P, M[t_c \mapsto M(t_c) :: (o, t_m)]\rangle_l}\ \lfloor^{\mathsf{DCC}}|_{\mathsf{InSend}}\rceil$$

$$\frac{B = ?@e_1(e_2); B'' \quad \mathbf{eval}(e_1,t) = l \quad Q = B'\cdot \varnothing \triangleleft (\,x, \mathbf{eval}(e_2,t)\,)}{\langle !(x); B',\ B\cdot t \mid P,\ M\rangle_l \quad\to\quad \langle !(x); B',\ Q \mid B''\cdot t \mid P,\ M\rangle_l}\ \lfloor^{\mathsf{DCC}}|_{\mathsf{InStart}}\rceil$$

$$\frac{\begin{array}{c} B = o@e_1(e_2)\ \mathtt{to}\ e_3; B'' \qquad \mathbf{eval}(e_1,t) = l' \qquad \mathbf{eval}(e_3,t) = t_c \\ \mathbf{eval}(e_2,t) = t_m \qquad t_c \in \mathbf{dom}(M') \qquad M'' = M'[t_c \mapsto M'(t_c) :: (o, t_m)] \end{array}}{\langle \mathfrak{B}, B\cdot t \mid P,\ M\rangle_l \mid \langle \mathfrak{B}', P',\ M'\rangle_{l'} \quad\to\quad \langle \mathfrak{B}, B''\cdot t \mid P,\ M\rangle_l \mid \langle \mathfrak{B}', P', M''\rangle_{l'}}\ \lfloor^{\mathsf{DCC}}|_{\mathsf{Send}}\rceil$$

$$\frac{B = ?@e_1(e_2); B'' \quad \mathfrak{B}' = !(x); B' \quad \mathbf{eval}(e_1,t) = l' \quad Q = B'\cdot \varnothing \triangleleft (\,x, \mathbf{eval}(e_2,t)\,)}{\langle \mathfrak{B},\ B\cdot t \mid P, M\rangle_l \mid \langle \mathfrak{B}',\ P',\ M'\rangle_{l'} \quad\to\quad \langle \mathfrak{B},\ B''\cdot t \mid P,\ M\rangle_l \mid \langle \mathfrak{B}', Q \mid P',\ M'\rangle_{l'}}\ \lfloor^{\mathsf{DCC}}|_{\mathsf{Start}}\rceil$$

$$\frac{S \to S'}{S \mid S_1 \quad\to\quad S' \mid S_1}\ \lfloor^{\mathsf{DCC}}|_{\mathsf{SPar}}\rceil \qquad \frac{S \equiv_{\mathtt{D}} S_1 \quad S_1 \to S_1' \quad S_1' \equiv_{\mathtt{D}} S'}{S \quad\to\quad S'}\ \lfloor^{\mathsf{DCC}}|_{\mathsf{SEq}}\rceil$$

$$\mathtt{def}\ X = B\ \mathtt{in}\ \mathbf{0}\cdot t \ \equiv_{\mathtt{D}}\ \mathbf{0}\cdot t \qquad P \mid P' \ \equiv_{\mathtt{D}}\ P' \mid P \qquad (P_1 \mid P_2) \mid P_3 \ \equiv_{\mathtt{D}}\ P_1 \mid (P_2 \mid P_3)$$
$$P \quad \equiv_{\mathtt{D}}\ P \mid \mathbf{0}\cdot t \qquad \mathtt{def}\ X = B\ \mathtt{in}\ X\cdot t \ \equiv_{\mathtt{D}}\ \mathtt{def}\ X = B\ \mathtt{in}\ B/X\cdot t \qquad S \mid S' \ \equiv_{\mathtt{D}}\ S' \mid S$$
$$(S_1 \mid S_2) \mid S_3 \ \equiv_{\mathtt{D}}\ S_1 \mid (S_2 \mid S_3)$$

Figure 19: Dynamic Correlation Calculus — Semantics.

e on operation o_i, it discards all other inputs and executes the continuation B_i. Term (*reqst*) is the dual of (*acpt*) and asks the service located at e_1 to spawn a new process, passing to it the message in e_2. Term (*newque*) models the creation of a new queue that correlates with a unique correlation key (in the service hosting the running process). The correlation key is stored under path x in the state of the process, for later access. The remaining terms are standard.

Semantics. In Figure 19, we report the rules defining the semantics of DCC, a relation $\rightarrow$ closed under a (standard) structural congruence $\equiv_{\mathtt{D}}$ that supports commutativity and associativity of parallel composition. We comment the rules.

Rules $\lfloor^{\mathsf{DCC}}{}_{\mathsf{Assign}}\rceil$, $\lfloor^{\mathsf{DCC}}{}_{\mathsf{Ctx}}\rceil$, and $\lfloor^{\mathsf{DCC}}{}_{\mathsf{Cond}}\rceil$ are standard for, respectively, assignments, procedure definition, and condition evaluation. Rule $\lfloor^{\mathsf{DCC}}{}_{\mathsf{PEq}}\rceil$ uses equivalence $\equiv_{\mathtt{D}}$ on DCC processes to describe parallel execution and recursion. The rules of $\equiv_{\mathtt{D}}$ are reported in the lower part of Figure 19.

Rule $\lfloor^{\mathsf{DCC}}{}_{\mathsf{Newque}}\rceil$ adds to M an empty queue (ε) correlating with a randomly generated key t_c. The key is stored under path x of the process that requested the creation of the queue. As in rule $\lfloor^{\mathbb{D}}{}_{\mathsf{Sup}}\rceil$ of Backend Choreographies (see § 4.1), we do not impose a structure for correlation keys, yet we require that they are distinct within their service.

Rule $\lfloor^{\mathsf{DCC}}{}_{\mathsf{Recv}}\rceil$ models message reception. Since both (*input*) and (*choice*) define receptions of messages, we consider both cases in the rule. Indeed, the first premise of the rule captures the presence of either an (*input*)—with shape $o_j(x)$ `from` e—or a (*choice*)—with shape $\sum_{i\in I}[o_i(x_i)\ \mathtt{from}\ e]\,\{B_i\}$. In both cases, we obtain the correlation key of the receiving queue from the evaluation of expression e against the state of the receiving process (t). Then, we inspect queue map M and check if it has a message in its head received on operation o_j. If this holds, the rule removes the message from the queue and stores the payload (t_m) under path x_j in the state of the process.

Regarding message delivery, in DCC, there are two output actions: *i*) (*output*) used by a process to communicate with another one and *ii*) (*reqst*) used by a process to require the creation of a new process in a service. Since in DCC communications can happen within the same service or between two services, we describe two sets of rules, either for internal and inter-service message delivery.

We start from the easier case of internal delivery, defined by rules $\lfloor^{\mathsf{DCC}}{}_{\mathsf{InSend}}\rceil$ and $\lfloor^{\mathsf{DCC}}{}_{\mathsf{InStart}}\rceil$. In rule $\lfloor^{\mathsf{DCC}}{}_{\mathsf{InSend}}\rceil$ a process $B \cdot t$ sends a message into a queue of its hosting service. This is illustrated by the second premise of the rule where the location l, corresponding to the evaluation of expression e_1 against the state of the sender process, is the same of its hosting service. As expected, correlation key t_c must point an actual queue of the service. This is checked by the last premise, which requires t_c to be in the domain of queue map M. In the conclusion of the rule, we update the content of the queue pointed by t_c including message (o, t_m) in its tail. In rule $\lfloor^{\mathsf{DCC}}{}_{\mathsf{InStart}}\rceil$ a service accepts the request to create a new process from one of its local processes. In the conclusion of the rule, we find the newly created process Q. The behaviour of the new process corresponds to the one associated with the (*acpt*) term of the service (B'). The state of the new process is empty ($\varnothing$) except for the inclusion of the payload of the request, stored under path x and obtained from the evaluation of e_2 against t.

Message delivery between two services is defined by rules $\lfloor^{\mathsf{DCC}}{}_{\mathsf{Send}}\rceil$ and $\lfloor^{\mathsf{DCC}}{}_{\mathsf{Start}}\rceil$. The two rules are similar to their respective internal cases, except for requiring the location defined by the sender (i.e., the one obtained from the evaluation of expression e_1 against the state t of the sender process) to match that of the receiving service.

The last two rules in Figure 19 are $\lfloor^{\mathsf{DCC}}{}_{\mathsf{SPar}}\rceil$ and $\lfloor^{\mathsf{DCC}}{}_{\mathsf{SEq}}\rceil$ and define the (parallel) execution of networks of services.

6. Compiling Frontend Choreographies into DCC processes

We now present our main result: the correct compilation of high-level Frontend Choreographies into low-level DCC networks of services (and processes). We depict in Figure 20 a schematic

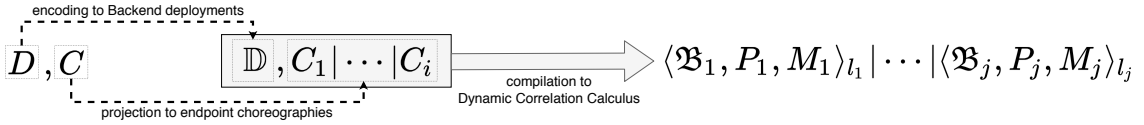


Figure 20: Scheme of compilation from Frontend Choreographies to Dynamic Correlation Calculus.

representation of the stages involved in the compilation from FC to DCC programs. Concretely, given an FC program D, C and its typing environment Γ, our compilation procedure consists of three stages:

FC-to-BC: the encoding, defined in § 4.3, of the Frontend deployment D to a correspondent Backend deployment $\mathbb{D} = \langle\!\langle D \rangle\!\rangle^{\Gamma}$. This stage is depicted in Figure 20 with the relation $D \dashrightarrow \mathbb{D}$;

EPP: the projection of the choreography C into a parallel composition of partial choreographies (i.e., containing only actions concerning one participant), each defining the behaviour of a single active or service process in C. This stage is called *Endpoint Projection*, it is presented in § 6.1, and it is depicted in Figure 20 with the relation $C \dashrightarrow C_1 | \cdots | C_i$;

Compilation: the compilation of the composition of the results of the previous stages—essentially, an endpoint Backend choreography—into a network of corresponding DCC services and their located processes. We present the compilation in § 6.3. This stage is depicted in Figure 20 with the relation $\mathbb{D}, C_1 | \cdots | C_i \Rightarrow \langle \mathfrak{B}_1, P_1, M_1 \rangle_{l_1} | \cdots | \langle \mathfrak{B}_j, P_j, M_j \rangle_{l_j}$ (the left part highlighted in grey to help readability);

The division in three stages makes the definition of the compilation process, and its related checks for correctness, simpler. In particular, they ease the extraction of the behaviour of a single process (**EPP**) from the source Frontend choreography and of its state (**FC-to-BC**) from the source Frontend deployment. In the remainder of this section, we detail the projection stage (**EPP**), we define how we pair the outputs of **FC-to-BC** and **EPP** and the properties of that pairing, and we present the **Compilation** stage and the properties of our main contribution.

6.1. **Endpoint Projection (EPP).** Given a choreography[4] C, its Endpoint Projection (EPP), denoted $[\![C]\!]$, returns an operationally-equivalent composition of *Endpoint choreographies*. Intuitively, an Endpoint choreography is a choreography that does not contain complete actions—i.e., terms (*start*) and (*com*)—and that describes the behaviour of a single process. We remind that a choreography can contain two kinds of processes: *active processes* which are already running, and *service processes* which accept requests to create new active processes at their respective associated location l. As detailed later on, our EPP procedure projects Endpoint choreographies on all processes, both active and service ones.

Our definition of EPP is an adaptation of that presented in [MY13] and it is divided into two components:

- a *process projection* that derives the Endpoint choreography of a single process p from a given choreography C, written $[\![C]\!]_{\mathsf{p}}$;
- the actual EPP of a given choreography C, which results into the parallel composition of:

[4]Since the EPP acts on the syntax and FC and BC share the same syntax, distinguishing between them here is irrelevant.

– the process projections of all active processes in C;
– the process projections of all service processes in C, with the exception that we merge into the same Endpoint choreography all process projections of service processes that accept requests at the same location.

In the next paragraphs, first we present the process projection and next the actual Endpoint Projection.

Process Projection. Let us start the definition of process projection by formalising Endpoint choreographies.

Definition 8 (Endpoint Choreographies)**.** Given a (Frontend/Backend) choreography C. If either:

- $C = \texttt{acc}\ k : l.\mathsf{q}[\mathsf{B}]; C'$, and q is the only free process name in C';
- C has only one free process name.

then C is an Endpoint choreography.

The process projection of a subject process p in a choreography C, written $[\![C]\!]_{\mathsf{p}}$, returns the Endpoint choreography obtained following the rules defined in Figure 21.

Process projection follows the structure of the source choreography. We briefly comment the rules in Figure 21, from top to bottom.

We start with the complete actions (*start*) and (*com*) which, if participated by the subject process, are projected into proper partial terms. When projecting a (*start*) action, if the subject process is the active process p, we project a (*req*). If otherwise the subject process is one of the service processes in $\tilde{\mathsf{q}}$, we project an (always-available) (*accept*). Similarly, when projecting a (*com*) action, if the subject process is the sender or the receiver in the interaction, we respectively project a (*send*) or a (*recv*). Partial actions (*accept*), (*req*), and (*send*) are projected verbatim, except for (*accept*) terms, which define the availability of only the subject process.

When projecting a (*rec*) term, we project both the body of the procedure (C') and the choreography C. This is safe even if r does not take part into the body of X, indeed, in that case, the projection of C' is just an (*inact*) term. As a consequence, we can safely project (*call*) terms verbatim.

The projections of conditionals and receptions are peculiar. Indeed, we project a conditional verbatim if the subject process evaluates the condition; for all other processes, we merge their behaviours with the merging (partial commutative) operator $\sqcup$, defined by the rules reported in Appendix (Figure 24). $C \sqcup C'$ is defined only for Endpoint choreographies and returns a choreography isomorphic to C and C' up to receptions, where all receptions with distinct operations are also included. We use $\sqcup$ also in the projection of (*recv*) terms, where we require the behaviour of all processes not receiving the message to be merged.

When projecting two choreographies in parallel we return the parallel composition of their respective projections, while (*inact*) is projected verbatim.

Finally, we draw attention on the definition of the rule of process projection for (*rec*) terms. Indeed, applying a naïve rule like

$$[\![\texttt{def}\ X = C'\ \texttt{in}\ C]\!]_{\mathsf{r}} \quad = \quad \texttt{def}\ X = [\![C']\!]_{\mathsf{r}}\ \texttt{in}\ [\![C]\!]_{\mathsf{r}}$$

in the EPP would yeld more than one procedure with the same identifier, which could prevent the obtained projection from being typable as, according to the typing rules defined in § 3, we cannot have in Γ two definition typings on the same identifier. To tackle the issue, the rule for (*rec*) terms in Figure 21 guarantees the coherent definition and usage of process-unique

$$
\begin{array}{lcl}
\llbracket \texttt{start}\ k : \mathtt{p}[\mathtt{A}] \leftrightarrow \widetilde{l.\mathtt{q}[\mathtt{B}]}; C \rrbracket_{\mathsf{r}} & = & \begin{cases} \texttt{req}\ k : \mathtt{p}[\mathtt{A}] \leftrightarrow \widetilde{l.\mathtt{B}}; \llbracket C \rrbracket_{\mathsf{r}} & \text{if } \mathsf{r} = \mathtt{p} \\ \texttt{acc}\ k : l.\mathtt{r}[\mathtt{C}]; \llbracket C \rrbracket_{\mathsf{r}} & \text{if } l.\mathtt{r}[\mathtt{C}] \in \{\widetilde{l.\mathtt{q}[\mathtt{B}]}\} \\ \llbracket C \rrbracket_{\mathsf{r}} & \text{otherwise} \end{cases} \\
\llbracket k : \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{q}[\mathtt{B}].o(x); C \rrbracket_{\mathsf{r}} & = & \begin{cases} k : \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o; \llbracket C \rrbracket_{\mathsf{r}} & \text{if } \mathsf{r} = \mathtt{p} \\ k : \mathtt{A} \longrightarrow \mathtt{q}[\mathtt{B}].o(x); \llbracket C \rrbracket_{\mathsf{r}} & \text{if } \mathsf{r} = \mathtt{q} \\ \llbracket C \rrbracket_{\mathsf{r}} & \text{otherwise} \end{cases} \\
\llbracket \texttt{acc}\ k : \widetilde{l.\mathtt{q}[\mathtt{B}]}; C \rrbracket_{\mathsf{r}} & = & \begin{cases} \texttt{acc}\ k : l.\mathtt{r}[\mathtt{C}]; \llbracket C \rrbracket_{\mathsf{r}} & \text{if } l.\mathtt{r}[\mathtt{C}] \in \{\widetilde{l.\mathtt{q}[\mathtt{B}]}\} \\ \llbracket C \rrbracket_{\mathsf{r}} & \text{otherwise} \end{cases} \\
\llbracket \texttt{req}\ k : \mathtt{p}[\mathtt{A}] \leftrightarrow \widetilde{l.\mathtt{B}}; C \rrbracket_{\mathsf{r}} & = & \begin{cases} \texttt{req}\ k : \mathtt{p}[\mathtt{A}] \leftrightarrow \widetilde{l.\mathtt{B}}; \llbracket C \rrbracket_{\mathsf{r}} & \text{if } \mathsf{r} = \mathtt{p} \\ \llbracket C \rrbracket_{\mathsf{r}} & \text{otherwise} \end{cases} \\
\llbracket k : \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o; C \rrbracket_{\mathsf{r}} & = & \begin{cases} k : \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o; \llbracket C \rrbracket_{\mathsf{r}} & \text{if } \mathsf{r} = \mathtt{p} \\ \llbracket C \rrbracket_{\mathsf{r}} & \text{otherwise} \end{cases} \\
\llbracket \texttt{def}\ X = C' \ \texttt{in}\ C \rrbracket_{\mathsf{r}} & = & \texttt{def}\ X_{\mathsf{r}} = \llbracket\ C'[X_{\mathsf{r}}/X]\ \rrbracket_{\mathsf{r}}\ \texttt{in}\ \llbracket\ C[X_{\mathsf{r}}/X]\ \rrbracket_{\mathsf{r}} \\
\llbracket X \rrbracket_{\mathsf{r}} & = & X \\
\llbracket \texttt{if}\ \mathtt{p}.e\ \{C_1\}\ \texttt{else}\ \{C_2\} \rrbracket_{\mathsf{r}} & = & \begin{cases} \texttt{if}\ \mathtt{p}.e\ \{\llbracket C_1 \rrbracket_{\mathsf{r}}\}\ \texttt{else}\ \{\llbracket C_2 \rrbracket_{\mathsf{r}}\} & \text{if } \mathsf{r} = \mathtt{p} \\ \llbracket C_1 \rrbracket_{\mathsf{r}} \sqcup \llbracket C_2 \rrbracket_{\mathsf{r}} & \text{otherwise} \end{cases} \\
\llbracket k : \mathtt{A} \longrightarrow \mathtt{q}[\mathtt{B}].\{o_i(x_i); C_i\}_{i \in I} \rrbracket_{\mathsf{r}} & = & \begin{cases} k : \mathtt{A} \longrightarrow \mathtt{q}[\mathtt{B}].\{o_i(x_i); \llbracket C_i \rrbracket_{\mathsf{r}}\}_{i \in I} & \text{if } \mathsf{r} = \mathtt{q} \\ \bigsqcup_{i \in I} \llbracket C_i \rrbracket_{\mathsf{r}} & \text{otherwise} \end{cases} \\
\llbracket C_1 \mid C_2 \rrbracket_{\mathsf{r}} & = & \llbracket C_1 \rrbracket_{\mathsf{r}} \mid \llbracket C_2 \rrbracket_{\mathsf{r}} \\
\llbracket \mathbf{0} \rrbracket_{\mathsf{r}} & = & \mathbf{0}
\end{array}
$$

Figure 21: Frontend Choreographies — process projection.

identifiers through renaming. The renaming is safe as, by assumption, we consider well-sorted choreographies where definitions always precede recursive calls.

We conclude the paragraph with the formal definition of process projection.

Definition 9 (Process Projection)**.** $\llbracket C \rrbracket_{\mathsf{r}}$ is a partial homomorphism from (Frontend/Backend) choreographies to Endpoint Choreographies, inductively defined by the rules in Figure 21.

Endpoint Projection. We can now proceed to define our Endpoint Projection.

In the definition below, we use the grouping operator $\lfloor C \rceil_l$, which returns the set of all service processes accepting requests at location l. We report in Appendix (Figure 25) the rules that inductively define $\lfloor C \rceil_l$.

$$
[\![C]\!]_{\mathsf{b}} = \begin{array}{l}
\texttt{if b.confirm\_pay}(cc,\ order)\ \{ \\
\quad k : \mathtt{b}[\mathtt{B}] \longrightarrow \mathtt{C}.ok;\ k : \mathtt{b}[\mathtt{B}] \longrightarrow \mathtt{S}.ok \\
\}\ \texttt{else}\ \{ \\
\quad k : \mathtt{b}[\mathtt{B}] \longrightarrow \mathtt{C}.ko;\ k : \mathtt{b}[\mathtt{B}] \longrightarrow \mathtt{S}.ko \\
\}
\end{array}
\qquad
\begin{array}{l}
[\![C]\!]_{\mathsf{c}} = k : \mathtt{B} \longrightarrow \mathtt{c}[\mathtt{C}].\{\ ok(),\ ko()\ \} \\
\\
[\![C]\!]_{\mathsf{s}} = k : \mathtt{B} \longrightarrow \mathtt{s}[\mathtt{S}].\{\ ok(),\ ko()\ \}
\end{array}
$$

$$
C = \begin{array}{l}
\texttt{if b.confirm\_pay}(\ cc,\ order\ )\{ \\
\quad k : \mathtt{b}[\mathtt{B}] \longrightarrow \mathtt{c}[\mathtt{C}].ok();\ k : \mathtt{b}[\mathtt{B}] \longrightarrow \mathtt{s}[\mathtt{S}].ok() \\
\}\ \texttt{else}\ \{ \\
\quad k : \mathtt{b}[\mathtt{B}] \longrightarrow \mathtt{c}[\mathtt{C}].ko();\ k : \mathtt{b}[\mathtt{B}] \longrightarrow \mathtt{s}[\mathtt{S}].ko() \\
\}
\end{array}
$$

Figure 22: Example of Endpoint Projection (top-half) of Lines 5–9 of Example 1 (lower-half).

Definition 10 (Endpoint Projection)**.** Let C be a (Frontend/Backend) choreography. The endpoint projection of C, denoted by $[\![C]\!]$, is defined as:

$$
[\![C]\!] = \underbrace{\prod_{\mathsf{p} \,\in\, \mathbf{fp}(C)} [\![C]\!]_{\mathsf{p}}}_{(i)} \mid \underbrace{\prod_{l} \left(\bigsqcup_{\mathsf{p} \,\in\, \lfloor C \rfloor_l} [\![C]\!]_{\mathsf{p}} \right)}_{(ii)}
$$

Commenting Definition 10, the EPP of a choreography is the parallel composition of two kinds of Endpoint choreographies: (i) Endpoint choreographies that are the process projection of active processes $\mathsf{p} \in \mathbf{fp}(C)$ and (ii) Endpoint choreographies that are the merge ($\sqcup$) of the process projections of all service processes available at the same location l, i.e., $\mathsf{p} \in \lfloor C \rfloor_l$.

Example 6. As an example of Endpoint Projection, let C be the choreography at Lines 5–9 of Example 1 (for convenience, we report the mentioned snippet of code grayed-out in the lower part of Figure 22). The EPP of C, $[\![C]\!]$, is the parallel composition of the process projections of processes c, s, and b, i.e., respectively $[\![C]\!]_{\mathsf{c}}$, $[\![C]\!]_{\mathsf{s}}$, and $[\![C]\!]_{\mathsf{b}}$. As per Definition 10, $[\![C]\!] = [\![C]\!]_{\mathsf{c}} \mid [\![C]\!]_{\mathsf{s}} \mid [\![C]\!]_{\mathsf{b}}$.

We report in the top half of Figure 22 the projections $[\![C]\!]_{\mathsf{c}}$, $[\![C]\!]_{\mathsf{s}}$, and $[\![C]\!]_{\mathsf{b}}$. The example is useful to illustrate that the projection of the conditional is homomorphic on the process (b) that evaluates it. The projection of a (*com*) term results into a partial (*send*) for the sender—as in the two branches of the conditional in $[\![C]\!]_{\mathsf{b}}$—and a partial (*recv*) for the receiver—as in $[\![C]\!]_{\mathsf{c}}$ and $[\![C]\!]_{\mathsf{s}}$. Note that the EPP merges branching behaviours: in $[\![C]\!]_{\mathsf{c}}$ and $[\![C]\!]_{\mathsf{s}}$ the two complete communications are merged into a partial reception on either operation *ok* or *ko*.

6.2. **Properties.** We conclude this section presenting the guarantees provided by the Endpoint Projection wrt to the source Frontend choreography, as formalised in Theorem 5. Before presenting Theorem 5, introduce the notion of *pruning* (as defined in [CHY12]), where $\prec$ specifies an asymmetric relation between two choreographies C and C', written $C \prec C'$,

in which C prunes some unused accepts and receptions of C'. To give a formal definition to our pruning relation, we present the two concepts of subtyping of typing environments and minimal typing system. Below we just give the intuition on both concepts, which are formalised in the Appendix:

- given two typing environments Γ and Γ', Γ is a subtype of Γ', written $\Gamma \prec \Gamma'$, if Γ is identical to Γ' up to *i*) some local and global types that are more constrained in Γ than in Γ' and *ii*) some service typings present in Γ' and not present in Γ. We report the formal definition of $\Gamma \prec \Gamma'$ in Definition 18,
- the minimal typing system $\Gamma \vdash_{\texttt{min}} C$ uses the minimal global and local types to type sessions and services in C. We report in appendix B.3.1 the formal definition of minimal typing.

We can finally formalise the pruning relation.

Definition 11 (Pruning)**.** Let $\Gamma \vdash_{\texttt{min}} C$ and $\Gamma' \vdash_{\texttt{min}} C'$, if $\Gamma \prec \Gamma'$ then C prunes C' under Γ, written $\Gamma \vdash_{\texttt{min}} C \prec C'$, or $C \prec C'$ for short.

The shortened form $C \prec C'$ is similar to [CHY12], where, as here, it does not lose precision since it is always possible to reconstruct appropriate typings. The pruning of C' by C means that C omits unused inputs and service processes present in C'. The $\prec$ relation is thus a strong bisimulation since $C \prec C'$ means that the two choreographies have precisely the same observable behaviours, except for the receive actions at pruned receptions and unused available service processes.

We can now write the statement of our EPP Theorem.

Theorem 5 (EPP Theorem)**.** Let D, C be a well-typed Frontend choreography. Then,

(1) (Well-typedness) $D, [\![C]\!]$ is well-typed.
(2) (Completeness) $D, C \to D', C'$ implies $D, [\![C]\!] \to D', C''$ and $[\![C']\!] \prec C''$.
(3) (Soundness) $D, [\![C]\!] \to D', C''$ implies $D, C \to D', C'$ and $[\![C']\!] \prec C''$.

We report in appendix B.3 the proof of Theorem 5.

6.3. **From Backend Endpoint Choreographies to DCC (Compilation).** This is the last stage of our compilation process, where, given a parallel composition of Backend Endpoint choreographies, we obtain a network of DCC services that faithfully follow the semantics of the source choreography.

Given a Backend deployment $\mathbb{D}$, a parallel composition of endpoint choreographies C, and a typing environment Γ, we write $\boxed{\mathbb{D}, C}^{\Gamma}$ to indicate the compilation of $\mathbb{D}, C$ under Γ into DCC.

To formally define $\boxed{\mathbb{D}, C}^{\Gamma}$, we use some auxiliary functions:

- $C|_l$ returns the endpoint choreography in C correspondent to the service process accepting requests at location l (e.g., $C|_l = \texttt{acc}\ k : l.\mathsf{p}[\texttt{A}]; C''$);
- $C|_{\mathsf{p}}$ returns the endpoint choreography in C correspondent to process p;
- $\boxed{C}^{\Gamma}$, given a single endpoint choreography C and a typing environment Γ, compiles C to DCC, using the rules in Figure 23;
- $l \in \Gamma$, a predicate satisfied if, according to Γ, location l contains or can spawn processes;
- $\mathbb{D}|_l$ returns the partial function of type $\mathcal{T} \rightharpoonup \mathit{Seq}(\mathcal{O} \times \mathcal{T})$ that corresponds to the projection of function g_m in $\mathbb{D}$ with location l fixed. Formally, for each t such that $\mathbb{D}(l : t) = \tilde{m}$, $\mathbb{D}|_l(t) = \tilde{m}$.

$$\text{Let } \mathtt{p}@l' \in \Gamma,\ \boxed{\mathtt{req}\ k : \mathtt{p}[\mathtt{A}] \leftarrow\!\!\rightarrow \widetilde{l.\mathtt{B}}; C}^{\Gamma} \quad = \quad \mathbf{start}(k, l'.\mathtt{A}, \widetilde{l.\mathtt{B}}); \boxed{C}^{\Gamma}$$

$$\mathbf{start}(k, l_{\mathtt{A}}.\mathtt{A}, \widetilde{l_{\mathtt{B}}.\mathtt{B}}) = \underbrace{\bigodot_{\mathtt{I} \in \{\mathtt{A}, \tilde{\mathtt{B}}\}} \underline{k.I.l} = l_{\mathtt{I}}\ ;}_{s_1}\ \underbrace{\bigodot_{\mathtt{I} \in \{\tilde{\mathtt{B}}\}} \left(\begin{array}{l} \nu\rangle \underline{k.I.A}\ ; \\ ?@\underline{k.I.l}(\underline{k})\ ; \\ sync(\underline{k})\ \mathtt{from}\ \underline{k.I.A} \end{array} \right);}_{s_2}\ \underbrace{\bigodot_{\mathtt{I} \in \{\tilde{\mathtt{B}}\}} start@\underline{k.I.l}(\underline{k})\ \mathtt{to}\ \underline{k.A.I}}_{s_3}$$

$$\text{Let } l \in \tilde{l}, \tilde{l} \in \Gamma,\ \boxed{\mathtt{acc}\ k : l.\mathtt{q}[\mathtt{B}]; C}^{\Gamma} \quad = \quad \mathbf{accept}(\ k,\ \mathtt{B}, \Gamma(\tilde{l})\ ); \boxed{C}^{\Gamma},$$

$$\mathbf{accept}(k, \mathtt{B}, G\langle \mathtt{A} | \tilde{\mathtt{C}} | \tilde{\mathtt{D}} \rangle) = \underbrace{!(\underline{k})\ ;}_{a_1}\ \underbrace{\bigodot_{\mathtt{I} \in \{\mathtt{A}, \tilde{\mathtt{C}}\} \setminus \{\mathtt{B}\}} \Big(\nu\rangle \underline{k.I.B}\Big);}_{a_2}\ \underbrace{sync@\underline{k.A.l}(\underline{k})\ \mathtt{to}\ \underline{k.B.A};}_{a_3}\ \underbrace{start(\underline{k})\ \mathtt{from}\ \underline{k.A.B}}_{a_4}$$

$$\boxed{k : \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o; C}^{\Gamma} \quad = \quad o@\underline{k.B.l}(e)\ \mathtt{to}\ \underline{k.A.B}; \boxed{C}^{\Gamma}$$

$$\boxed{k : \mathtt{A} \longrightarrow \mathtt{q}[\mathtt{B}].\{o_i(x_i); C_i\}_{i \in I}}^{\Gamma} \quad = \quad \sum_{i \in I} [o_i(x_i)\ \mathtt{from}\ \underline{k.A.B}]\ \{\boxed{C_i}^{\Gamma}\}$$

$$\boxed{\mathtt{if}\ \mathtt{p}.e\ \{C_1\}\ \mathtt{else}\ \{C_2\}}^{\Gamma} \quad = \quad \mathtt{if}\ e\ \{\boxed{C_1}^{\Gamma}\}\ \mathtt{else}\ \{\boxed{C_2}^{\Gamma}\}$$

$$\boxed{\mathtt{def}\ X = C'\ \mathtt{in}\ C}^{\Gamma} \quad = \quad \mathtt{def}\ X = \boxed{C'}^{\Gamma}\ \mathtt{in}\ \boxed{C}^{\Gamma}$$

$$\boxed{X}^{\Gamma} \quad = \quad X$$

$$\boxed{\mathbf{0}}^{\Gamma} \quad = \quad \mathbf{0}$$

Figure 23: Compiler from Endpoint Choreographies to DCC.

Definition 12 (Compilation)**.** Let $\mathbb{D}$ be a Backend deployment, C a parallel composition of endpoint choreographies, and given the typing environment Γ

$$\boxed{\mathbb{D}, C}^{\Gamma} = \prod_{l\ \in\ \Gamma} \left\langle \boxed{C|_l}^{\Gamma}, \prod_{\mathtt{p}\ \in\ \mathbb{D}(l)} \boxed{C|_{\mathtt{p}}}^{\Gamma} \cdot \mathbb{D}(\mathtt{p})\ ,\ \mathbb{D}|_l \right\rangle_l$$

Intuitively, for each service $\langle \mathfrak{B}, P, M \rangle_l$ in the compiled network: *i*) the start behaviour $\mathfrak{B}$ is the compilation of the endpoint choreography in C accepting the creation of processes at location l; *ii*) P is the parallel composition of the compilation of all active processes located at l, equipped with their respective states according to $\mathbb{D}$; *iii*) M is the set of queues in $\mathbb{D}$ corresponding to location l.

We comment the rules in Figure 23, where the notation $\bigodot$ is the sequence of behaviours $\bigodot_{i \in [1,n]}(B_i) = B_1; \ldots; B_n$.

Requests. Function `start` defines the compilation of (*req*) terms. Function `start` compiles (*req*) terms to create the queues and a part of the session descriptor for the starter (this is similar to what rule $[^{\mathbb{D}}|_{\mathsf{Sup}}]$ does in Backend deployment transitions, § 4.1). Given a session identifier k, the located role of the starter ($l_{\mathtt{A}}.\mathtt{A}$), and the other located roles in the session ($\widetilde{l_{\mathtt{B}}.\mathtt{B}}$), `start` returns the DCC code that:

s_1 includes in the session descriptor all the locations of the processes involved in the session;

s_2 for each role, except for the starter,

- creates the key and the correlated queue that the current role will use in the session to communicate with the starter;

– requests the creation of the service process that will play the current role in the session;
– waits on the reserved operation *sync* to receive the correlation data for the session defined by the newly created process.

s_3 sends to the newly created processes the complete session descriptor obtained after the reception (in the *sync* step) of all correlation keys.

Accepts. (*acc*) terms define the start behaviour of a spawned process at a location. Given a session identifier k, the role $\mathtt{B}$ of the service process, and the service typing $G\langle \mathtt{A}|\tilde{\mathtt{C}}|\tilde{\mathtt{D}}\rangle$ of the location, function `accept` compiles the code that: (a_1) accepts the request to spawn a process, (a_2) creates its queues and keys, updates the session descriptor received from the starter, and sends it back to the latter (a_3). Finally with (a_4) the new process waits to start the session.

Other terms. A (*send*) term compiles to a DCC (*output*) term. Notably, the compiled code contains the same elements used by the semantics of BC to implement correlation, i.e., the location of the receiver ($\underline{k.B.l}$) and the key that correlates with its queue ($\underline{k.A.B}$). Similarly, (*recv*) compiles to (*choice*), which defines the path ($\underline{k.A.B}$) of the key correlating with the receiving queue.

Example 7. As an example of compilation, we compile the first two lines of the choreography C in Example 1, considering a deployment $\mathbb{D}$ and a typing environment Γ such that $\Gamma \vdash \mathbb{D}, C$.

$$\boxed{\mathbb{D}, [\![C]\!]}^{\Gamma} = \langle \mathbf{0}, P_{\mathtt{c}}\rangle_{l_{\mathtt{c}}} \mid \langle \mathfrak{B}_{\mathtt{S}}, \mathbf{0}\rangle_{l_{\mathtt{S}}} \mid \langle \mathfrak{B}_{\mathtt{B}}, \mathbf{0}\rangle_{l_{\mathtt{B}}}$$

where

$$P_{\mathtt{c}} = \begin{cases} \underline{k.S.l} = l_{\mathtt{S}}; \quad \underline{k.B.l} = l_{\mathtt{B}}; \quad \nu\rangle\underline{k.S.C}; \quad ?@\underline{k.S.l}(\underline{k}); \quad \mathit{sync}(\underline{k})\ \mathtt{from}\ \underline{k.S.C}; \\ \nu\rangle\underline{k.B.C}; \quad ?@\underline{k.B.l}(\underline{k}); \quad \mathit{sync}(\underline{k})\ \mathtt{from}\ \underline{k.B.C}; \quad \mathit{start}@\underline{k.S.l}(\underline{k})\ \mathtt{to}\ \underline{k.C.S}; \\ \mathit{start}@\underline{k.B.l}(\underline{k})\ \mathtt{to}\ \underline{k.C.B}; \quad \texttt{/* end of start-request */} \\ \mathit{buy}@\underline{k.S.l}(\mathit{product})\ \mathtt{to}\ \underline{k.C.S}; \quad \ldots \end{cases}$$

and

$$\mathfrak{B}_{\mathtt{S}} = \begin{cases} !(\underline{k}); \quad \nu\rangle\underline{k.C.S}; \quad \nu\rangle\underline{k.B.S}; \quad \mathit{sync}@\underline{k.C.l}(\underline{k})\ \mathtt{to}\ \underline{k.S.C}; \\ \mathit{start}(\underline{k})\ \mathtt{from}\ \underline{k.C.S}; \quad \texttt{/* end of accept */} \quad \mathit{buy}(x)\ \mathtt{from}\ \underline{k.C.S}; \quad \ldots \end{cases}$$

We omit to report $\mathfrak{B}_{\mathtt{B}}$, which is similar to $\mathfrak{B}_{\mathtt{S}}$.

6.4. **Properties of Applied Choreographies.** We conclude this section by presenting our main result, i.e., a compiler from Frontend choreographies to DCC networks and its properties.

In our definition, we use the term *projectable* to indicate that, given a choreography C, we can obtain its projection $[\![C]\!]$. Formally

Definition 13 (Projectable Choreography)**.** Let C be a choreography, we call C *projectable* if there is a choreography C such that $C' = [\![C]\!]$.

Theorem 6 defines our result, for which, given a well-typed, projectable Frontend choreography, we can obtain its correct implementation as a DCC network. Such result is obtained by merging the properties of the stages **FC-to-BC**(§ 4.3), **EPP**(§ 6.1), and **Compilation**(§ 6.3).

Theorem 6 (Applied Choreographies)**.** Let D, C be a Frontend choreography where C is projectable and $\Gamma \vdash D, C$ for some Γ. Then:
(1) (Completeness) $D, C \to D', C'$ implies

$$\boxed{\langle\!\langle D \rangle\!\rangle^{\Gamma}, [\![C]\!]}^{\Gamma} \to^{+} \boxed{\langle\!\langle D' \rangle\!\rangle^{\Gamma'}, C''}^{\Gamma'} \quad \text{and} \quad [\![C']\!] \prec C'' \quad \text{and} \quad \text{for some } \Gamma',\ \Gamma' \vdash D', C'$$

(2) (Soundness) $\boxed{\langle\!\langle D \rangle\!\rangle^{\Gamma}, [\![C]\!]}^{\Gamma} \to^{*} S$ implies

$$D, C \to^{*} D', C' \quad \text{and} \quad S \to^{*} \boxed{\langle\!\langle D' \rangle\!\rangle^{\Gamma'}, C''}^{\Gamma'} \quad \text{and} \quad [\![C']\!] \prec C'' \quad \text{and} \quad \text{for some } \Gamma',\ \Gamma' \vdash D', C'$$

We report in appendix B.7 the proof of Theorem 6.

By Theorem 3 and Theorem 6, deadlock-freedom is preserved from well-typed choreographies to their final translation in DCC. We say that a network S in DCC is deadlock-free if it is either a composition of services with terminated running processes or it can reduce.

Corollary 1. $\Gamma \vdash D, C$ and $\mathbf{co}(\Gamma)$ imply that $\boxed{D, [\![C]\!]}^{\Gamma}$ is deadlock-free.

7. Related Work and Discussion

This is the first work that formalises how we can use choreographies in the setting of a practical communication mechanism used in Service-Oriented Computing (SOC), i.e., message correlation. Previous formal choreography languages specify only an EPP procedure towards a calculus based on name synchronisation, leaving the design of its concrete support to implementors. Chor [Cho16] and AIOCJ [AIO16] are the respective implementations of the models found in [CM13] and [DGG+15]. However, the implementations of their EPP depart significantly from their respective formalisations, since they are based on message correlation instead of name synchronisation. This means that there is no proof that the implementation strategies followed in these languages correctly supports synchronisation on names. Implementations of other frameworks based on sessions share similar issues [HYH08, HNY+13, NY14]. Our work gives the first correctness result for the compilation of choreographies to a language close to real-world implementations. More in general, our results are a useful reference to formalise the implementation of session-based languages. In the future, this line of work may pave the way to establishing certified choreography compilation.

We believe that our approach can be easily applied to many models that use choreographies and sessions (or channel-based communications), including those designed around (variants of) the π-calculus [CHY12, CM13, MY13, HYC16] and those based on linear logic [CMS17, CMSY17].

Our development shows that it is possible to keep a simple language model as frontend, allowing developers to abstract from how sessions are concretely implemented. Nevertheless, our Frontend Choreographies are expressive, as illustrated by our examples, and recent studies have shown that choreography languages such as this are Turing complete [CM16]. There are many works that investigate how to introduce different features to choreographies, which we have not studied here and leave to future work. Examples include nested protocols [DH12], asynchronous two-way exchanges [CMS17], and general recursion [CM17]. These features are orthogonal to our development, so their inclusion should be straightforward. A more interesting feature to add may be session delegation for choreographies [CM13, HYC16]. Delegation allows to transfer the responsibility to continue a session from a process to another. Introducing delegation in FC is straightforward, since we can just import the

development from [CM13, MY13]. Implementing it in BC and DCC would be more involved, but not difficult: delegating a role in a session translates to moving the content of a queue from a process to another, and ensuring that future messages reach the new process. The mechanisms to achieve the latter part have been investigated in [HYH08], which use retransmission protocols. Formalising these "middleware" protocols and proving that they preserve the intended semantics of FC could be an interesting future work.

In the semantics of BC, we abstract from how correlation keys are generated. With this loose definition we capture several implementations, provided they satisfy the requirement of uniqueness of keys (wrt to locations). As future work, we plan to implement a language, based on our framework, able to support custom procedures for the generation of correlation keys (e.g., from database queries, cookies, etc.).

8. Conclusion

In this paper, we presented our framework of Applied Choreographies, which includes three calculi: a high-level choreographic language intended for developers, an intermediate-representation choreographic language, and a low-level, close-to-implementation distributed calculus. We equip our framework with a tight series of behavioural correspondences, so that we guarantee that low-level distributed programs compiled from high-level sources faithfully follow their source specifications. By pairing our compilation with a type system and static checks that guarantee the absence of deadlocks in high-level choreographies, we obtain that the compiled distributed systems are deadlock-free. Specifically, we target Service-Oriented distributed systems that communicate over correlation mechanisms.

Besides the contribution above, Applied Choreographies introduce a novel semantics for choreographies that abstracts the features of choreographies (message passing, creation of new sessions and processes) from their implementation (and the related complexity). To this end we *i*) equip choreographies with a global deployment and *ii*) define a separate semantics of effects on deployments. This separation allows us to compose our semantics of choreographies with other definitions of deployment and effects so that we have a straightforward way to capture different communication semantics (e.g., synchronous, asynchronous with buffers) and implementations (e.g., distributed objects [CC91]). The notion of deployments let us formalise how choreographies can go wrong (see § 3.3) and show that the theory of session types is useful not only to type communications on choreographies ([CM13, MY13]) but also to check the correctness of deployments. It is worth noting that, except for the declaration of locations, Applied Choreographies has the same types and syntax from previous works [CM13, MY13], hence developers have only to specify protocols and choreographies and do not need to deal with deployment information or correlation data.

We have already mentioned some short-term future work in the previous section. More long term projects include the investigation of compilation to other target languages/communication mechanisms besides correlation-based ones-orientation, for instance those found in Erlang and Scala+Akka. Clearly this would be a major development, since the actor-based concurrency and message passing of these languages are substantially different from that based on correlation, considered in this paper. Another ambitious goal is the application of our research to the Internet of Things (IoT) setting. IoT promotes the communication among heterogeneous entities—which use a wide range of communication media and data protocols–whose integration result in a cumbersome low level programming activity. Indeed,

to achieve a higher degree of interoperability [GGLZ18, GGLZ19] propose the use of high-level, service-oriented languages for communication technology integration in IoT systems. In particular, an extension of Jolie is introduced [GGLZ18, GGLZ19] which natively integrates the two most adopted protocols for IoT communication (CoAP and MQTT). We plan to take this approach further by developing a suitable version of Applied Choreographies, specifically designed for IoT applications, which can then be compiled to the Jolie extension mentioned above. This would allow one to import in the IoT field the correct-by-construction approach through the formal correctness of compilation that we have developed in this paper.

References

[Agh85] Gul A Agha. Actors: A model of concurrent computation in distributed systems. Technical report, MASSACHUSETTS INST OF TECH CAMBRIDGE ARTIFICIAL INTELLIGENCE LAB, 1985.

[AIO16] AIOCJ Team. AIOCJ framework, 2016. `http://www.cs.unibo.it/projects/jolie/aiocj.html`.

[B+14] Tim Bray et al. The javascript object notation (json) data interchange format, 2014.

[BGG+06] Nadia Busi, Roberto Gorrieri, Claudio Guidi, Roberto Lucchi, and Gianluigi Zavattaro. Choreography and orchestration conformance for system design. In *COORDINATION*, pages 63–81. Springer, 2006.

[BPSM+98] Tim Bray, Jean Paoli, C Michael Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible markup language (xml). *W3C Recommendation REC-xml-19980210*, 16, 1998.

[CC91] Roger S. Chin and Samuel T. Chanson. Distributed object-based programming systems. *ACM Computing Surveys*, 23(1):91–124, 1991.

[CD88] George F. Coulouris and Jean Dollimore. *Distributed Systems: Concepts and Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1988.

[CDCYP15] Mario Coppo, Mariangiola Dezani-Ciancaglini, Nobuko Yoshida, and Luca Padovani. Global progress for dynamically interleaved multiparty sessions. *MSCS*, 760:1–65, 2015.

[CES71] Edward G Coffman, Melanie Elphick, and Arie Shoshani. System deadlocks. *ACM Computing Surveys (CSUR)*, 3(2):67–78, 1971.

[Cho16] Chor Team. Chor Programming Language, 2016. `http://www.chor-lang.org/`.

[CHY12] Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured communication-centered programming for web services. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 34(2):1–78, 2012.

[CLM05] Samuele Carpineti, Cosimo Laneve, and Paolo Milazzo. Bopi - A distributed machine for experimenting web services technologies. In *ACSD*, pages 202–211. IEEE, 2005.

[CLM17] Luís Cruz-Filipe, Kim S. Larsen, and Fabrizio Montesi. The paths to choreography extraction. In *FoSSaCS*, volume 10203 of *Lecture Notes in Computer Science*, pages 424–440, 2017.

[CM13] Marco Carbone and Fabrizio Montesi. Deadlock-freedom-by-design: multiparty asynchronous global programming. In *POPL*, pages 263–274, 2013.

[CM16] Luís Cruz-Filipe and Fabrizio Montesi. A core model for choreographic programming. In *FACS*, volume 10231 of *Lecture Notes in Computer Science*, pages 17–35, 2016.

[CM17] Luís Cruz-Filipe and Fabrizio Montesi. Procedural choreographic programming. In *FORTE*, volume 10321 of *Lecture Notes in Computer Science*, pages 92–107. Springer, 2017.

[CMS17] Marco Carbone, Fabrizio Montesi, and Carsten Schürmann. Choreographies, logically. *Distributed Computing*, pages 1–17, 2017. Also: CONCUR, pages 47–62, 2014.

[CMSY17] Marco Carbone, Fabrizio Montesi, Carsten Schürmann, and Nobuko Yoshida. Multiparty session types as coherence proofs. *Acta Informatica*, 54(3):243–269, 2017.

[DGG+15] Mila Dalla Preda, Maurizio Gabbrielli, Saverio Giallorenzo, Ivan Lanese, and Jacopo Mauro. Dynamic choreographies. In *COORDINATION*, pages 67–82. Springer, 2015.

[DGL+14] Mila Dalla Preda, Saverio Giallorenzo, Ivan Lanese, Jacopo Mauro, and Maurizio Gabbrielli. AIOCJ: A choreographic framework for safe adaptive distributed applications. In *SLE*, pages 161–170, 2014.

[DGL+17] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. Microservices: yesterday, today, and tomorrow. In *Present and ulterior software engineering*, pages 195–216. Springer, 2017.

[DH12] Romain Demangeon and Kohei Honda. Nested protocols in session types. In *CONCUR*, pages 272–286, 2012.

[GGLZ18] Maurizio Gabbrielli, Saverio Giallorenzo, Ivan Lanese, and Stefano Pio Zingaro. A language-based approach for interoperability of iot platforms. In *51st Hawaii International Conference on System Sciences, HICSS 2018, Hilton Waikoloa Village, Hawaii, USA, January 4-7, 2018*, 2018. To appear.

[GGLZ19] Maurizio Gabbrielli, Saverio Giallorenzo, Ivan Lanese, and Stefano Pio Zingaro. Linguistic abstractions for interoperability of iot platforms. In *Towards Integrated Web, Mobile, and IoT Technology*, pages 83–114. Springer, 2019.

[GH05] Simon Gay and Malcolm Hole. Subtyping for session types in the pi calculus. *Acta Informatica*, 42(2-3):191–225, November 2005.

[GMG18] Saverio Giallorenzo, Fabrizio Montesi, and Maurizio Gabbrielli. Applied choreographies. In *Formal Techniques for Distributed Objects, Components, and Systems - 38th IFIP WG 6.1 International Conference, FORTE 2018, Held as Part of the 13th International Federated Conference on Distributed Computing Techniques, DisCoTec 2018, Madrid, Spain, June 18-21, 2018, Proceedings*, pages 21–40. Springer, 2018.

[HLV+16] Hans Hüttel, Ivan Lanese, Vasco T Vasconcelos, Luís Caires, Marco Carbone, Pierre-Malo Deniélou, Dimitris Mostrous, Luca Padovani, António Ravara, Emilio Tuosto, et al. Foundations of session types and behavioural contracts. *ACM Computing Surveys (CSUR)*, 49(1):1–36, 2016.

[HNY+13] Raymond Hu, Rumyana Neykova, Nobuko Yoshida, Romain Demangeon, and Kohei Honda. Practical interruptible conversations. In *RV*, pages 130–148, 2013.

[HO07] Philipp Haller and Martin Odersky. Actors that unify threads and events. In *Coordination Models and Languages*, pages 171–190. Springer, 2007.

[HYC08] K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *Proc. of POPL*, volume 43(1), pages 273–284. ACM, 2008.

[HYC16] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. *Journal of the ACM (JACM)*, 63(1):1–67, 2016.

[HYH08] Raymond Hu, Nobuko Yoshida, and Kohei Honda. Session-based distributed programming in java. In *ECOOP*, pages 516–541, 2008.

[Int96] International Telecommunication Union. Recommendation Z.120: Message sequence chart, 1996.

[JBo13] JBoss Community. Savara, 2013. `http://www.jboss.org/savara/`.

[LGMZ08] I. Lanese, C. Guidi, F. Montesi, and G. Zavattaro. Bridging the gap between interaction- and process-oriented choreographies. In *SEFM*, pages 323–332. IEEE, 2008.

[MC11] Fabrizio Montesi and Marco Carbone. Programming services with correlation sets. In *ICSOC*, pages 125–141, 2011.

[MGZ14] Fabrizio Montesi, Claudio Guidi, and Gianluigi Zavattaro. Service-oriented programming with Jolie. In *Web Services Foundations*, pages 81–107. 2014.

[Mil80] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer, 1980.

[Mon13] Fabrizio Montesi. *Choreographic Programming*. Ph.D. thesis, IT University of Copenhagen, 2013. `http://www.fabriziomontesi.com/files/choreographic_programming.pdf`.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–40,41–77, September 1992.

[MY13] Fabrizio Montesi and Nobuko Yoshida. Compositional choreographies. In *CONCUR*, pages 425–439, 2013.

[New15] Sam Newman. *Building microservices: designing fine-grained systems*, chapter 4. O'Reilly Media, Inc., 2015.

[NM92] Robert HB Netzer and Barton P Miller. What are race conditions? some issues and formalizations. *ACM Letters on Programming Languages and Systems (LOPLAS)*, 1(1):74–88, 1992.

[NS78] Roger M Needham and Michael D Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[NY14] Rumyana Neykova and Nobuko Yoshida. Multiparty session actors. In *COORDINATION*, pages 131–146, 2014.

[OAS07] OASIS. WS-BPEL. `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html`, 2007.

[O'H18] Peter O'Hearn. Experience developing and deploying concurrency analysis at facebook. In *International Static Analysis Symposium*, pages 56–70. Springer, 2018.

[OMG04] OMG. Unified modelling language, version 2.0, 2004.

[OMG11] OMG. Business Process Model and Notation. http://www.omg.org/spec/BPMN/2.0/, 2011.

[Pie02] Benjamin C. Pierce. *Types and Programming Languages*. MIT Press, MA, USA, 2002.

[QZCY07] Zongyan Qiu, Xiangpeng Zhao, Chao Cai, and Hongli Yang. Towards the theoretical foundation of choreography. In *WWW*, pages 973–982. IEEE Computer Society Press, 2007.

[SW01] D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[Vin06] Steve Vinoski. Advanced message queuing protocol. *IEEE Internet Computing*, 10(6), 2006.

[VW12] Alvaro Videla and Jason JW Williams. *RabbitMQ in action: distributed messaging for everyone*. Manning, 2012.

[W3C04] W3C WS-CDL Working Group. WS-CDL version 1.0, 2004. http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/.

Appendix A. Additional Material

A.1. **Typing.**

Definition 14 (List Subset)**.** Let ε be the empty list and $\tilde{N}$, $\tilde{M}$ be two lists of elements n of the kind $\tilde{N} ::= \varepsilon \mid n, \tilde{N'}$, the predicate $\tilde{N} \subseteq \tilde{M}$ holds if $\tilde{N} = \tilde{M} = \varepsilon$ or, assuming $\tilde{N} = n, \tilde{N'}$ and $\tilde{M} = m, \tilde{M'}$ either $n = m$ and $\tilde{N'} \subseteq \tilde{M'}$ or $\tilde{N} \subseteq \tilde{M'}$.

Definition 15 (Ordered Join Operator)**.** Let $\tilde{N}$, $\tilde{L}$, and $\tilde{M}$ be three lists of elements as defined in Definition 14, the ordered-join operator $\tilde{N} \bowtie_{\tilde{L}} \tilde{N}$ is defined as

$$\begin{array}{lcl} \tilde{N} \bowtie_{\varepsilon} \tilde{M} & = & \varepsilon \\ \tilde{N} \bowtie_{l,\tilde{L}} \tilde{M} & = & \begin{cases} \tilde{N} \bowtie_{\tilde{L}} \tilde{M} & \text{if } l \notin \tilde{N} \cup \tilde{M} \\ l, \tilde{N'} \bowtie_{\tilde{L}} \tilde{M} & \text{if } \tilde{N} = l, \tilde{N'} \\ l, \tilde{N} \bowtie_{\tilde{L}} \tilde{M'} & \text{if } \tilde{M} = l, \tilde{M'} \end{cases} \end{array}$$

A.2. **Compiling Frontend Choreographies into DCC Processes.**

$$\begin{array}{rcl} \texttt{acc}\ k : l.\texttt{p}[\texttt{A}]; C_1 \sqcup \\ \texttt{acc}\ k : l.\texttt{q}[\texttt{A}]; C_2 & = & \texttt{acc}\ k : l.\texttt{p}[\texttt{A}]; (C_1 \sqcup C_2) \end{array}$$

$$\begin{array}{rcl} \texttt{req}\ k : \texttt{p}[\texttt{A}] \leftrightarrow \widetilde{l.\texttt{B}}; C_1 \sqcup \\ \texttt{req}\ k : \texttt{q}[\texttt{A}] \leftrightarrow \widetilde{l.\texttt{B}}; C_2 & = & \texttt{req}\ k : \texttt{p}[\texttt{A}] \leftrightarrow \widetilde{l.\texttt{B}}; (C_1 \sqcup C_2) \end{array}$$

$$\begin{array}{rcl} k{:}\texttt{p}[\texttt{A}].e \longrightarrow \texttt{B}.o; C_1 \sqcup \\ k{:}\texttt{q}[\texttt{A}].e \longrightarrow \texttt{B}.o; C_2 & = & k{:}\texttt{p}[\texttt{A}].e \longrightarrow \texttt{B}.o; (C_1 \sqcup C_2) \end{array}$$

$$\begin{array}{rcl} k{:}\texttt{A} \longrightarrow \texttt{p}[\texttt{B}].\{\ o_i(x_i); C_i\ \}_{i \in I} \sqcup \\ k{:}\texttt{A} \longrightarrow \texttt{q}[\texttt{B}].\{\ o_j(x_j); C'_j\ \}_{j \in J} & = & k{:}\texttt{A} \longrightarrow \texttt{p}[\texttt{B}].\left\{ \begin{array}{ll} & \{\ o_i(x_i); C_i\ \}_{i \in I \setminus J} \\ \cup & \{\ o_i(x_i); C'_i\ \}_{i \in J \setminus I} \\ \cup & \{\ o_i(x_i); C_i \sqcup C'_i\ \}_{i \in I \cap J} \end{array} \right\} \end{array}$$

$$\begin{array}{rcl} \texttt{if}\ \texttt{p}.e\ \{C_1\}\ \texttt{else}\ \{C'_1\} \sqcup \\ \texttt{if}\ \texttt{q}.e\ \{C_2\}\ \texttt{else}\ \{C'_2\} & = & \texttt{if}\ \texttt{p}.e\ \{C_1 \sqcup C_2\}\ \texttt{else}\ \{C'_1 \sqcup C'_2\} \end{array}$$

$$\begin{array}{rcl} \texttt{def}\ X = C'_1\ \texttt{in}\ C_1 \sqcup \\ \texttt{def}\ Y = C'_2\ \texttt{in}\ C_2 & = & \texttt{def}\ X = C'_1 \sqcup C'_2\ \texttt{in}\ C_1 \sqcup C_2 \end{array}$$

$$X \sqcup Y = X$$

$$\mathbf{0} \sqcup \mathbf{0} = \mathbf{0}$$

Figure 24: Merging Function

$$
\begin{array}{lcl}
\left\lfloor \texttt{start}\ k : \texttt{p[D]} \leftrightarrow \widetilde{l.\texttt{q[B]}}; C \right\rfloor_l & = & \left\lfloor \texttt{acc}\ k : \widetilde{l.\texttt{q[B]}}; C \right\rfloor_l \\
\left\lfloor \texttt{acc}\ k : \widetilde{l.\texttt{q[B]}}; C \right\rfloor_l & = & \begin{cases} \{\texttt{r}\} \cup \lfloor C \rfloor_l & \text{if } l.\texttt{r[A]} \in \{\ \widetilde{l.\texttt{q[B]}}\ \} \\ \lfloor C \rfloor_l & \text{otherwise} \end{cases} \\
\lfloor \eta; C \rfloor_l & = & \lfloor C \rfloor_l \quad \text{if } \eta \neq (\mathit{start}) \\
\lfloor \texttt{if p}.e\ \{C_1\}\ \texttt{else}\ \{C_2\} \rfloor_l & = & \lfloor C_1 \rfloor_l \cup \lfloor C_2 \rfloor_l \\
\lfloor \texttt{def}\ X = C'\ \texttt{in}\ C \rfloor_l & = & \lfloor C' \rfloor_l \cup \lfloor C \rfloor_l \\
\lfloor X \rfloor_l & = & \varnothing \\
\lfloor \mathbf{0} \rfloor_l & = & \varnothing \\
\lfloor C_1 \mid C_2 \rfloor_l & = & \lfloor C_1 \rfloor_l \cup \lfloor C_2 \rfloor_l
\end{array}
$$

Figure 25: Service Grouping

Appendix B. Proofs

B.1. **Proofs of Subject Reduction and Session Fidelity.** In order to prove Subject Reduction (Theorem 1), we prove the stronger result of Typing Soundness, defined in Theorem 7. We use Theorem 7 to also prove Session Fidelity (Theorem 2).

In order to define and prove Theorem 7, we provide additional definitions and lemmas, in particular:

- we define an annotated semantics for FC (appendix B.1.1) to track reductions on sessions;
- we define subtyping (appendix B.1.2) for local types and for typing environments. On these definitions we prove lemmas used to relate evolutions of the typing environment wrt reductions in choreographies;
- we define an annotated semantics for global types (appendix B.1.3) and prove Lemma 4, guaranteeing that global types and local types in the typing environment evolve accordingly.

Finally, we proceed to prove Typing Soundness (appendix B.1.5) and consequently Subject Reduction and Session Fidelity.

B.1.1. *FC Annotated Semantics.* We define the semantics of annotated FCs by marking transitions with the name of the session whose term has reduced. We annotate other reductions as τ. We range over annotated labels with

$$\beta \quad ::= \quad k : \mathtt{A} \longrightarrow \mathtt{B}.o \quad | \quad k : \mathtt{A} \rangle \mathtt{B}.o(x) \quad | \quad \tau$$

We report the annotated semantics of FC in Figure 26. Intuitively, we mark reductions over a session k with $k : \mathtt{A} \longrightarrow \mathtt{B}.o$ for message sends ($\lfloor^{\mathsf{C}}_{\mathsf{Send}}\rceil$ and $\lfloor^{\mathsf{C}}_{\mathsf{Com}}\rceil$) and $k : \mathtt{A} \rangle \mathtt{B}.o(x)$ for receptions ($\lfloor^{\mathsf{C}}_{\mathsf{Recv}}\rceil$).

$$\frac{D\#k', \tilde{\mathtt{r}} \quad \delta = \mathtt{start}\ k' : \mathtt{p}[\mathtt{A}] \leftrightarrows \widetilde{l.\mathtt{q}[\mathtt{B}]} \quad D, \delta \blacktriangleright D'}{D,\ \mathtt{start}\ k : \mathtt{p}[\mathtt{A}] \leftrightarrows \widetilde{l.\mathtt{q}[\mathtt{B}]}; C \quad \xrightarrow{\tau} \quad D',\ C[k'/k][\tilde{\mathtt{r}}/\tilde{\mathtt{q}}]} \ \lfloor \mathsf{C} |_{\mathsf{Start}} \rceil$$

$$\frac{\eta = k : \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o \quad D, \eta \blacktriangleright D'}{D,\ \eta; C \quad \xrightarrow{k:\mathtt{A} \longrightarrow \mathtt{B}.o} \quad D',\ C} \ \lfloor \mathsf{C} |_{\mathsf{Send}} \rceil$$

$$\frac{j \in I \quad D, k : \mathtt{A} \longrightarrow \mathtt{q}[\mathtt{B}].o_j(x_j) \blacktriangleright D'}{D,\ k : \mathtt{A} \longrightarrow \mathtt{q}[\mathtt{B}].\{o_i(x_i); C_i\}_{i \in I} \quad \xrightarrow{k:\mathtt{A}\rangle\mathtt{B}.o_j(x_j)} \quad D',\ C_j} \ \lfloor \mathsf{C} |_{\mathsf{Recv}} \rceil$$

$$\frac{i = 1 \text{ if } \mathbf{eval}(\ e, D(\mathtt{p})\ ) = \mathtt{true}, i = 2 \text{ otherwise}}{D,\ \mathtt{if}\ \mathtt{p}.e\ \{C_1\}\ \mathtt{else}\ \{C_2\} \quad \xrightarrow{\tau} \quad D,\ C_i} \ \lfloor \mathsf{C} |_{\mathsf{Cond}} \rceil$$

$$\frac{D, C_1 \xrightarrow{\beta} D', C_1'}{D, \mathtt{def}\ X = C_2\ \mathtt{in}\ C_1 \quad \xrightarrow{\beta} \quad D', \mathtt{def}\ X = C_2\ \mathtt{in}\ C_1'} \ \lfloor \mathsf{C} |_{\mathsf{Ctx}} \rceil$$

$$\frac{\mathcal{R} \in \{ \equiv, \simeq_{\mathsf{C}} \} \quad C\,\mathcal{R}\,C_1 \quad D, C_1 \xrightarrow{\beta} D', C_1' \quad C_1\,\mathcal{R}\,C'}{D, C \quad \xrightarrow{\beta} \quad D', C'} \ \lfloor \mathsf{C} |_{\mathsf{Eq}} \rceil$$

$$\frac{D, C_1 \quad \xrightarrow{\beta} \quad D', C_1'}{D, C_1 \mid C_2 \quad \xrightarrow{\beta} \quad D', C_1' \mid C_2} \ \lfloor \mathsf{C} |_{\mathsf{Par}} \rceil$$

$$\frac{\begin{array}{c} i \in \{1, \ldots, n\} \quad D\#k', \tilde{\mathtt{r}} \quad \{\widetilde{l.\mathtt{B}}\} = \biguplus_i \{\widetilde{l_i.\mathtt{B}_i}\} \quad \{\tilde{\mathtt{r}}\} = \bigcup_i \{\tilde{\mathtt{r}}_i\} \\ \delta = \mathtt{start}\ k' : \mathtt{p}[\mathtt{A}] \leftrightarrows \widetilde{l_1.\mathtt{r}_1[\mathtt{B}_1]}, \ldots, \widetilde{l_n.\mathtt{r}_n[\mathtt{B}_n]} \qquad D, \delta \blacktriangleright D' \end{array}}{\begin{array}{l} D, \mathtt{req}\ k : \mathtt{p}[\mathtt{A}] \leftrightarrows \widetilde{l.\mathtt{B}}; C \mid \prod_i \left(\mathtt{acc}\ k : \widetilde{l_i.\mathtt{q}_i[\mathtt{B}_i]}; C_i\right) \xrightarrow{\tau} \\ \quad D',\ C[k'/k] \mid \prod_i \left(\ C_i[k'/k][\tilde{\mathtt{r}}_i/\tilde{\mathtt{q}}_i]\ \right) \mid \prod_i \left(\mathtt{acc}\ k : \widetilde{l_i.\mathtt{q}_i[\mathtt{B}_i]}; C_i\right) \end{array}} \ \lfloor \mathsf{C} |_{\mathsf{PStart}} \rceil$$

Figure 26: Fronted Choreographies — annotated semantics.

B.1.2. *Local Types and Typing Environment Subtyping.* We define a subtyping relation on local types following [GH05, CHY12, MY13]. We write the subtyping relation as $T' \prec T$, which intuitively indicates that T' is more constrained than T in its behaviour. Note that, like in [CHY12, MY13], the input type is covariant and the output type is contravariant for this relation.

Definition 16 (Local Subtyping)**.** We define the subtyping relation between local types as $T' \prec T$, which is the smallest relation over closed local types, satisfying the rules

$$\frac{T'' \prec T' \quad T \approx T''}{T \prec T'} \ \lfloor^{\mathsf{SubT}}|_{\mathsf{Eq}}\rceil \qquad \frac{J \subseteq I \quad \forall\, i \in J \mid T_i \prec T_i' \ \wedge\ U_i \prec U_i'}{!\mathtt{A}.\{o_i(U_i); T_i\}_{i\in I} \ \prec\ !\mathtt{A}.\{o_i(U_i'); T_i'\}_{i\in J}} \ \lfloor^{\mathsf{SubT}}|_{\mathsf{Send}}\rceil$$

$$\frac{I \subseteq J \quad \forall\, i \in I \mid T_i \prec T_i' \ \wedge\ U_i \prec U_i'}{?\mathtt{A}.\{o_i(U_i); T_i\}_{i\in I} \ \prec\ ?\mathtt{A}.\{o_i(U_i'); T_i'\}_{i\in J}} \ \lfloor^{\mathsf{SubT}}|_{\mathsf{Recv}}\rceil \qquad \frac{}{U \prec U} \ \lfloor^{\mathsf{SubT}}|_{\mathsf{Val}}\rceil \qquad \frac{\mathtt{end} \approx T}{\mathtt{end} \prec T} \ \lfloor^{\mathsf{SubT}}|_{\mathsf{End}}\rceil$$

In rule $\lfloor^{\mathsf{SubT}}|_{\mathsf{Eq}}\rceil$, $T \prec T'$ if there exists a local type T'', subtype of T', such that $T \approx T''$, i.e., T'' approximates T, $\approx$ being the standard tree isomorphism on recursive types.

Although not directly relevant in the current proof, we also define the subtyping for global types $G \prec G'$, which intuitively follows that of local ones. Subtyping for global types is used in the definition of Environment subtyping. The relation between subtyping of Environments and of global types (in service typings) will become relevant when proving properties of our Endpoint Projection (see appendix B.3). Our definition of subtyping for global types follows [MY13].

Definition 17 (Global Subtyping)**.** $G \prec G'$ is the smallest relation over closed global types satisfying the rules below

$$\frac{I \subseteq J \quad \forall\, i \in I,\ G_i \prec G_i' \ \wedge U_i \prec U_i'}{\mathtt{A} \longrightarrow \mathtt{B}.\{o_i(U_i); G_i\}_{i\in I} \prec \mathtt{A} \longrightarrow \mathtt{B}.\{o_j(U_j'); G_j'\}_{j\in J}} \ \lfloor^{\mathsf{SubG}}|_{\mathsf{Com}}\rceil$$

$$\frac{U \prec U' \quad G \prec G'}{\mathtt{A}\rangle\mathtt{B}.o(U); G \prec \mathtt{A}\rangle\mathtt{B}.o(U'); G'} \ \lfloor^{\mathsf{SubG}}|_{\mathsf{Recv}}\rceil$$

$$\frac{G'' \prec G' \quad (G'' \approx G \ \vee\ G'' \simeq_{\mathtt{G}} G)}{G \prec G'} \ \lfloor^{\mathsf{SubG}}|_{\mathsf{Eq}}\rceil \qquad \frac{\mathtt{end} \approx G}{\mathtt{end} \prec G} \ \lfloor^{\mathsf{SubG}}|_{\mathsf{End}}\rceil$$

Finally, we define a subtyping relation between Typing Environments. Intuitively $\Gamma \prec \Gamma'$ means that Γ' and Γ are identical Typing Environments up to *a*) some local and global types that are more constrained in Γ — i.e., subtypes of a correspondent global/local type — than in Γ' and *b*) some service typings not present in Γ.

Definition 18 (Typing Environment Subtyping)**.** Let Γ and Γ' be two typing environments, where $\Gamma' = \Gamma'', \Gamma_l$, for which $\mathbf{dom}(\Gamma) = \mathbf{dom}(\Gamma'')$ and Γ_l contains only service typings. Then, $\Gamma \prec \Gamma'$ if and only if

(*i*)	$\forall\ \mathsf{p}.x\colon U \in \Gamma,$	$\Gamma' \vdash \mathsf{p}.x\colon U$
(*ii*)	$\forall\ X\colon \Gamma_x \in \Gamma,$	$\Gamma' \vdash X\colon \Gamma_x$
(*iii*)	$\forall\ \mathsf{p}\colon k[\mathtt{A}] \in \Gamma,$	$\Gamma' \vdash \mathsf{p}\colon k[\mathtt{A}]$
(*iv*)	$\forall\ \mathsf{p}@l \in \Gamma,$	$\Gamma' \vdash \mathsf{p}@l$
(*v*)	$\forall\ k[\mathtt{A}\rangle\mathtt{B}]\colon T \in \Gamma,$	$\Gamma' \vdash k[\mathtt{A}\rangle\mathtt{B}]\colon T$
(*vi*)	$\forall\ k[\mathtt{A}]\colon T \in \Gamma,$	$\Gamma' \vdash k[\mathtt{A}]\colon T'$ and $T \prec T'$
(*vii*)	$\forall\ \tilde{l}\colon G\langle \mathtt{A}\vert\tilde{\mathtt{B}}\vert\tilde{\mathtt{C}}\rangle \in \Gamma,$	$\Gamma' \vdash \tilde{l}\colon G'\langle \mathtt{A}\vert\tilde{\mathtt{B}}\vert\tilde{\mathtt{C}}\rangle$ and $G \prec G'$

Commenting the definition, the subtyping relation for typing environments states that an environment Γ is a subtype of an environment Γ' if

- they type the same variables (i), procedure definitions (ii), role ownerships (iii), process locations (iv), and buffers (v) and they agree on their judgements;
- they type the same local sessions (vi) and the local type in Γ is a subtype of the local type in Γ';
- if they type the same service (vii) (note that Γ' is allowed to have additional service typings wrt Γ) and the global type in Γ is a subtype of the global type in Γ'.

In Lemma 1 we prove that if $\Gamma \prec \Gamma'$ and Γ types a running choreography D, C also Γ' types that choreography.

Lemma 1 (Subsumption)**.** Let $\Gamma \prec \Gamma'$ and $\Gamma \vdash D, C$ for some D, C then $\Gamma' \vdash D, C$.

Proof . The proof is immediate by Definition 16 and rules $[\![\mathsf{T}|_{\mathsf{Recv}}]\!]$, $[\![\mathsf{T}|_{\mathsf{Send}}]\!]$, and $[\![\mathsf{T}|_{\mathsf{Com}}]\!]$. Intuitively, the lemma holds since the local typings in Γ' allow for additional, unused actions in D, C. □

We also prove Lemma 2 which guarantees that the typing of choreographies (C) is invariant wrt buffer types.

Lemma 2 (Buffer types invariance)**.** Let $\Gamma = \Gamma', \Gamma_b$ where Γ_b contains only buffer typings. If $\Gamma' \vdash C$ then $\Gamma \vdash C$.

Proof . Trivial from the definition of rule $[\![\mathsf{T}|_{\mathsf{DC}}]\!]$ and $\Gamma \vdash C$ for which buffer typings affect only predicate **pco** and the typing of deployments. □

B.1.3. *Reductions for Global Types.* We annotate the reductions of global types with labels

$$\gamma \quad ::= \quad \mathtt{A} \longrightarrow \mathtt{B}.o \qquad | \qquad \mathtt{A}\rangle\mathtt{B}.o$$

and report below the correspondent annotated semantics.

$$\frac{o \in \bigcup_i \{o_i\} \qquad G' = {}^{\mathtt{A}\rangle\mathtt{B}}_{o}\!\downarrow G}{\oplus \mathtt{A}\mathtt{B}.\{o_i(U_i)\}; G \xrightarrow{\mathtt{A} \longrightarrow \mathtt{B}.o} G'}\ [\![\mathsf{G}|_{\mathsf{Send}}]\!] \qquad \frac{}{\mathtt{A}\rangle\mathtt{B}.o(U); G \xrightarrow{\mathtt{A}\rangle\mathtt{B}.o} G}\ [\![\mathsf{G}|_{\mathsf{Recv}}]\!]$$

$$\frac{\mathcal{R} \in \{\equiv_{\mathtt{G}}, \simeq_{\mathtt{G}}\} \qquad G\ \mathcal{R}\ G_1 \qquad G_1 \xrightarrow{\gamma} G_1' \qquad G_1'\ \mathcal{R}\ G'}{G \xrightarrow{\gamma} G'}\ [\![\mathsf{G}|_{\mathsf{Eq}}]\!]$$

In Lemma 3 we account for the fact that any output reduction at the level of global types can constrain the projected local types of the roles not involved in the reduction. Indeed, referring to rule $[\![\mathsf{G}|_{\mathsf{Send}}]\!]$, the output operation chooses one of the available continuations G' and discards all the others. Therefore the local types of the other roles not involved in the reduction can be constrained by the removal of the unused branches.

Lemma 3 (Projection Subtyping)**.** Let $T = [\![G]\!]_{\mathtt{C}}$, $T' = [\![G']\!]_{\mathtt{C}}$, and $\{\mathtt{A}, \mathtt{B}, \mathtt{C}\} \subseteq \mathbf{roles}(G)$, $\mathtt{C} \notin \{\mathtt{A}, \mathtt{B}\}$, then $G \xrightarrow{\mathtt{A} \longrightarrow \mathtt{B}.o} G'$ implies $T' \prec T$.

Proof . By induction on the derivation of $G \xrightarrow{\gamma} G'$. □

B.1.4. *Typing Environment Reductions.* We define a reduction relation for typing environments. To do so, we first formalise the writing $k \notin \Gamma$, which means that Γ has no local typing and buffer types for session k, formally, for some local types T and T'

$$k \notin \Gamma \iff \nexists\, \mathtt{A}, \mathtt{B} \text{ s.t. } k[\mathtt{A}] \colon T \in \Gamma \ \vee k[\mathtt{A}\langle\mathtt{B}] \colon T' \in \Gamma$$

Finally, we formalise the reduction relation for typing environments of the form $\Gamma \to \Gamma'$, $\to$ being the smallest closed under the rules below. Note that the annotation labels are a subset of the labels used to annotate the semantics of FC, ranged over by β.

$$\frac{k \notin \Gamma \quad \Gamma_k \subseteq [\![G]\!]_k \quad \{k[\mathtt{A}] \colon T, k[\mathtt{B}] \colon T'\} \in \Gamma_k \quad j \in I \quad G \xrightarrow{\mathtt{A} \longrightarrow \mathtt{B}.o_j} G'}{\Gamma, \Gamma_k \xrightarrow{k\,:\,\mathtt{A} \longrightarrow \mathtt{B}.o_j} \Gamma, \{k[\mathtt{C}] \colon [\![G']\!]_{\mathtt{C}} \mid k[\mathtt{C}] \in \Gamma_k\}, \{k[\mathtt{C}\langle\mathtt{D}] \colon [\![G']\!]^{\mathtt{D}}_{\mathtt{C}} \mid k[\mathtt{C}\langle\mathtt{D}] \in \Gamma_k\}} \lfloor\Gamma|_{\mathsf{Send}}\rceil$$

$$\frac{k \notin \Gamma \quad \Gamma_k \subseteq [\![G]\!]_k \quad \{k[\mathtt{A}] \colon T, k[\mathtt{B}] \colon T'\} \in \Gamma_k \quad \Gamma \vdash \mathtt{q} \colon k[\mathtt{B}] \quad G \xrightarrow{\mathtt{A}\langle\mathtt{B}.o_j} G'}{\Gamma, \Gamma_k \xrightarrow{k:\mathtt{A}\langle\mathtt{B}.o_j(x)} \Gamma, \{k[\mathtt{C}] \colon [\![G']\!]_{\mathtt{C}} \mid k[\mathtt{C}] \in \Gamma_k\}, \{k[\mathtt{C}\langle\mathtt{D}] \colon [\![G']\!]^{\mathtt{D}}_{\mathtt{C}} \mid k[\mathtt{C}\langle\mathtt{D}] \in \Gamma_k\}, \mathtt{q}.x \colon U_j} \lfloor\Gamma|_{\mathsf{Recv}}\rceil$$

With slight abuse of notation, we also write β_k to mark reductions of Γ on session k, i.e., $\beta_k \in \{k \colon \mathtt{A} \longrightarrow \mathtt{B}.o,\ k : \mathtt{A}\langle\mathtt{B}.o(x)\}$.

We define the correspondence operator $G_{\mathtt{act}}(\beta)$ between β and γ labels:

$$G_{\mathtt{act}}(\beta_k) = \begin{cases} \mathtt{A} \longrightarrow \mathtt{B}.o & \text{if } \beta_k = k \colon \mathtt{A} \longrightarrow \mathtt{B}.o \\ \mathtt{A}\langle\mathtt{B}.o & \text{if } \beta_k = k : \mathtt{A}\langle\mathtt{B}.o(x). \end{cases}$$

In Lemma 4 we prove that if a typing environment Γ includes local types that are projection of a global type G, then if the global type can reduce, also the typing environment can reduce. The reduction preserves the correspondence between the reduced global type and the reduced local types in Γ.

Lemma 4 (Type-Environment Fidelity)**.** Let $\Gamma = \Gamma_*, [\![G]\!]_k$ for some Γ_*, $k \notin \Gamma_*$, and $G \xrightarrow{G_{\mathtt{act}}(\beta_k)} G'$ then $\Gamma \xrightarrow{\beta_k} \Gamma'$ and for some Γ'_*, $k \notin \Gamma'_*$, $\Gamma' = \Gamma'_*, [\![G']\!]_k$.

Proof . Direct by cases on the derivation of Γ. □

B.1.5. *Proof of Typing Soundness.* We also report Lemmas 5 and 6 that prove that typing is invariant wrt structural equivalence and swapping.

Lemma 5 (Subject Congruence)**.** $\Gamma \vdash D, C$ and $C \equiv_{\mathtt{C}} C'$ imply $\Gamma \vdash D, C'$ (up to α-renaming)

Proof . By induction on the rules that define $\equiv_{\mathtt{C}}$. □

Lemma 6 (Subject Swap)**.** $\Gamma \vdash D, C$ and $C \simeq_{\mathtt{C}} C'$ imply $\Gamma \vdash D, C'$

Proof . By induction on the derivation of $C \simeq_{\mathtt{C}} C'$. □ Below we restate the definition of *Deployment Judgements* enriched with pointers of the kind (DX.Y) for a clearer referencing in the proofs.

Definition 3 (Deployment Judgements)
$\Gamma \vdash D \iff$

(D|3.1) $\forall\ \mathtt{p}.x \in \Gamma, D(\mathtt{p}).x : U$
(D|3.2) $\forall\ k[\mathtt{A}\rangle\mathtt{B}] : T \in \Gamma \wedge D(k[\mathtt{A}\rangle\mathtt{B}]) = \tilde{m},\ \mathtt{bte}(\mathtt{A}, \tilde{m}) = T$

Finally, we prove Theorem 1 by proving the stronger result Theorem 7.

In the proof, we use the context over global types $\mathcal{G}[\cdot]$, defined as

$$\begin{array}{rcl} \mathcal{G}[\cdot] ::= & \mathtt{A} \longrightarrow \mathtt{B}.\{o_i(U_i); \mathcal{G}[\cdot]\}_i \\ | & \oplus \mathtt{AB}.\{o_i(U_i)\}; \mathcal{G}[\cdot] \\ | & \&_{\mathtt{A}}\mathtt{B}.\{o_i(U_i); \mathcal{G}[\cdot]\}_{i\in I} \\ | & \mathtt{A}\rangle\mathtt{B}.o(U); \mathcal{G}[\cdot] \end{array}$$

We can now proceed to define and prove Theorem 7.

Theorem 7 (Typing Soundness)**.** Let D, C be an annotated FC and (T|7.1) $\Gamma \vdash D, C$ for some Γ:

: if (T|7.2) $\beta \neq \tau$ and $D, C \xrightarrow{\beta} D', C'$ then (T|7.3) $\Gamma \xrightarrow{\beta} \Gamma'$ and (T|7.4) $\Gamma' \vdash D', C'$;
: if (T|7.5) $D, C \xrightarrow{\tau} D', C'$ then, for some Γ', (T|7.6) $\Gamma' \vdash D', C'$.

Proof . Proof by induction on the derivation of $D, C \xrightarrow{\beta} D', C'$.

Case $\lfloor^{\mathsf{C}}_{\mathsf{Send}}\rceil$

The case is:

$$\frac{\eta = k : \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o_j \qquad D, \eta \blacktriangleright D'}{D,\ \eta; C \quad \xrightarrow{k:\ \mathtt{A} \longrightarrow \mathtt{B}.o_j} \quad D',\ C}\ \lfloor^{\mathsf{C}}_{\mathsf{Send}}\rceil$$

Where (T|7.2) has the reductum $C' = C$ and, let $v = \mathbf{eval}(e, D(\mathtt{p}))$ and $\tilde{m} = D(k[\mathtt{A}\rangle\mathtt{B}])$, $D' = D\big[\ k[\mathtt{A}\rangle\mathtt{B}] \mapsto \tilde{m} :: (o_j, v)\ \big]$ by rule $\lfloor^{\mathsf{D}}_{\mathsf{Send}}\rceil$.

To prove (T|7.3) we must prove rule $\lfloor^{\mathsf{\Gamma}}_{\mathsf{Send}}\rceil$ to be applicable.

From (T|7.1) we know that there exists a global type G for session k such that $\mathbf{pco}(\Gamma)$ holds. We can partition $\Gamma = \Gamma_*, \Gamma_k$ such that $\Gamma_* = \Gamma \setminus [\![G]\!]_k$ and $\Gamma_k = \Gamma \setminus \Gamma_*$.

From (T|7.1) we can write the derivation (with $\Gamma = \Gamma_1, k[\mathtt{A}] :\ \oplus\, \mathtt{B}.\{o_i(U_i); [\![G_i]\!]_{\mathtt{A}}\}_{i\in I}$)

$$\frac{\mathbf{pco}(\Gamma) \qquad \Gamma \vdash D \qquad \dfrac{j \in I \qquad \Gamma_1 \vdash \mathtt{p} :\ k[\mathtt{A}] \qquad \Gamma_1 \vdash \mathtt{p}.e :\ U_j \qquad \Gamma_1, k[\mathtt{A}] :\ [\![G_j]\!]_{\mathtt{A}} \vdash C}{\Gamma_1, k[\mathtt{A}] :\ \oplus\, \mathtt{B}.\{o_i(U_i); [\![G_i]\!]_{\mathtt{A}}\}_{i\in I} \vdash k : \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o_j; C}\ \lfloor^{\mathsf{\Gamma}}_{\mathsf{Send}}\rceil}{\Gamma \vdash D, k : \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o_j; C}\ \lfloor^{\mathsf{\Gamma}}_{\mathsf{DC}}\rceil$$

Since $\Gamma \vdash k[\mathtt{A}] :\ \oplus\, \mathtt{B}.\{o_i(U_i); T_i\}_{i\in I}$, we can write $G = \mathcal{G}[\mathtt{A} \longrightarrow \mathtt{B}.o_i(U_i); G_i]$ where $\forall\ i \in I$, $[\![G_i]\!]_{\mathtt{A}} = T_i$. Let π be the reduction of G with rules $\lfloor^{\mathsf{G}}_{\mathsf{Eq}}\rceil$ and $\lfloor^{\mathsf{G}}_{\mathsf{Send}}\rceil$, we observe the following derivation:

$$\pi = \left\{ \begin{array}{l} \dfrac{\dfrac{o_i \in U_i\{o_i\} \qquad G' = \Delta}{G_2 \xrightarrow{\gamma} G'}\ \lfloor^{\mathsf{G}}_{\mathsf{Send}}\rceil \qquad G' \simeq_{\mathtt{G}} G'}{\vdots\ \lfloor^{\mathsf{G}}_{\mathsf{Eq}}\rceil}\ G_1 \simeq_{\mathtt{G}} G_2 \\ \dfrac{G \equiv_{\mathtt{G}} G_1 \qquad G_1 \xrightarrow{\gamma} G' \qquad G \equiv_{\mathtt{G}} G'}{G \xrightarrow{\gamma} G'}\ \lfloor^{\mathsf{G}}_{\mathsf{Eq}}\rceil \end{array} \right. \quad \begin{array}{l} \Delta = {}^{\mathtt{A}\rangle\mathtt{B}}_{o_i}\!\downarrow \mathcal{G}[\&_{\mathtt{AB}}.\{o_i(U_i\}; G_i] \\ \\ G_2 = \oplus \mathtt{AB}.\{o_i(U_i)\}; \mathcal{G}[\&_{\mathtt{AB}}.\{o_i(U_i\}; G_i] \\ G_1 = \mathcal{G}[\oplus \mathtt{AB}.\{o_i(U_i\}; \&_{\mathtt{AB}}.\{o_i(U_i\}; G_i] \\ \\ \\ G' = \mathcal{G}[\mathtt{A}\rangle\mathtt{B}.o_j; G_j] \\ \gamma = \mathtt{A} \longrightarrow \mathtt{B}.o_j \end{array}$$

In the reductions, since C, D reduces with $\beta = k : \mathtt{A} \longrightarrow \mathtt{B}.o_j$ and G types C, D in Γ, there are no other exchanges from $\mathtt{A}$ to $\mathtt{B}$ in G that could prevent from obtaining, after a finite number of derivations on rule $\lfloor^{\mathsf{G}}_{\mathsf{Eq}}\rceil$, the swap-equivalence $G_1 \simeq_{\mathtt{G}} G_2$. Following a

similar reasoning, the application Δ targets the global branching in the context, which reduces the continuation $\mathcal{G}[\&_{\mathtt{A}\mathtt{B}}.\{o_i(U_i); G_i]$ after the global choice $\oplus_{\mathtt{A}\mathtt{B}}.\{o_i(U_i)\}$ to G'.

Given π, we can use it to write the reduction at the level the typing environment Γ, applying rule $\lfloor\Gamma|_{\mathsf{Send}}\rceil$. Below, we consider $\Gamma = \Gamma_*, \Gamma_k$ where Γ_k contains all and only typings of session k in Γ.

$$\frac{\begin{array}{c}\pi\\ \vdots\end{array}}{\dfrac{k \notin \Gamma_* \quad \Gamma_k \subseteq [\![G]\!]_k \quad \{k[\mathtt{A}]\colon T, k[\mathtt{B}]\colon T'\} \in \Gamma_k \quad j \in I \quad G \xrightarrow{\mathtt{A} \longrightarrow \mathtt{B}.o_j} G'}{\Gamma_*, \Gamma_k \xrightarrow{k\ \mathtt{A} \longrightarrow \mathtt{B}.o_j} \Gamma_*, \{k[\mathtt{C}]\colon [\![G']\!]_{\mathtt{C}} \mid k[\mathtt{C}] \in \Gamma_k\}, \{k[\mathtt{C}\langle\mathtt{D}]\colon [\![G']\!]^{\mathtt{D}}_{\mathtt{C}} \mid k[\mathtt{C}\langle\mathtt{D}] \in \Gamma_k\}}\ \lfloor\Gamma|_{\mathsf{Send}}\rceil}$$

Hence (T|7.3) holds and $\Gamma' = \Gamma_*, \{k[\mathtt{C}]\colon [\![G']\!]_{\mathtt{C}} \mid k[\mathtt{C}] \in \Gamma_k\}, \{k[\mathtt{C}\langle\mathtt{D}]\colon [\![G']\!]^{\mathtt{D}}_{\mathtt{C}} \mid k[\mathtt{C}\langle\mathtt{D}] \in \Gamma_k\}$. We now prove (T|7.4) by proving that rule $\lfloor\mathsf{T}|_{\mathsf{DC}}\rceil$ applies to $\Gamma' \vdash D', C'$.

$$\frac{\mathbf{pco}(\Gamma') \quad \Gamma' \vdash C' \quad \Gamma' \vdash D'}{\Gamma' \vdash D', C'}\ \lfloor\mathsf{T}|_{\mathsf{DC}}\rceil$$

Hence we need to prove ① $\mathbf{pco}(\Gamma')$, ② $\Gamma' \vdash C'$, and ③ $\Gamma' \vdash D'$

Proof of ①.

For all sessions $k' \in \Gamma_*$, $\mathbf{pco}(\Gamma')$ holds as $\mathbf{pco}(\Gamma)$ holds by (T|7.1). For session k, $\mathbf{pco}(\Gamma')$ holds by construction.

□

Proof of ②.

From the derivation on $\Gamma \vdash D, k\colon \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o_j; C$ we know that $\Gamma_1, k[\mathtt{A}]\colon [\![G_j]\!]_{\mathtt{A}} \vdash C$. Let $\Gamma'' = \Gamma_1, k[\mathtt{A}]\colon [\![G_j]\!]_{\mathtt{A}}$ and $\Gamma'_k = \Gamma_1 \setminus \Gamma_* = \Gamma_k \setminus \{k[\mathtt{A}]\colon [\![G]\!]_{\mathtt{A}}\}$. We can write $\Gamma'' = \Gamma_*, \Gamma'_k, k[\mathtt{A}]\colon [\![G_j]\!]_{\mathtt{A}}$. Note that in the premise of rule $\lfloor\mathsf{T}|_{\mathsf{Send}}\rceil$ that types the continuation C, the buffer types in Γ (i.e., those in Γ_1) are unaffected. Therefore $\Gamma''(k[\mathtt{A}\langle\mathtt{B}]) \neq \Gamma'(k[\mathtt{A}\langle\mathtt{B}])$, however from Lemma 2 we know that we can omit to consider buffer types as they are irrelevant for the typing of choreographies. For all sessions $k' \neq k$ in Γ'' their local typings are the same in Γ'. For session k, the typing $\Gamma''(k[\mathtt{A}]) = \Gamma'(k[\mathtt{A}]) = [\![G_j]\!]_{\mathtt{A}}$. From Lemma 3, for all other $k[\mathtt{C}] \in \Gamma'', \mathtt{C} \neq \mathtt{A}$ it holds that $\Gamma''(k[\mathtt{C}]) = [\![G]\!]_{\mathtt{C}}$, $\Gamma'(k[\mathtt{C}]) = [\![G']\!]_{\mathtt{C}}$, and $[\![G']\!]_{\mathtt{C}} \prec [\![G]\!]_{\mathtt{C}}$. Therefore $\Gamma' \prec \Gamma''$ and ② holds by Lemma 1.

□

Proof of ③.

To prove $\Gamma' \vdash D'$ we need to prove that the conditions of Definition 3 hold. (D|3.1) holds by the application of rule $\lfloor\mathsf{P}|_{\mathsf{Send}}\rceil$, by construction of Γ', and by (T|7.1). (D|3.2) holds for all sessions $k' \neq k$ by application of rule $\lfloor\mathsf{P}|_{\mathsf{Send}}\rceil$ and the construction of Γ'. The same holds true for session k and any process $\mathtt{q}\colon k[\mathtt{C}] \in \Gamma' \mid \mathtt{C} \neq \mathtt{B}$.

Finally, we need to prove that $\Gamma'(\ k[\mathtt{A}\langle\mathtt{B}]\ ) = \mathtt{bte}(\ \mathtt{A}, D'(k[\mathtt{A}\langle\mathtt{B}])\ )$. From (T|7.1) we know that *i)* $\Gamma(k[\mathtt{A}\langle\mathtt{B}]) = T$ and *ii)* let $D(k[\mathtt{A}\langle\mathtt{B}]) = \tilde{m}$, that $\mathtt{bte}(\mathtt{A}, \tilde{m}) = T$. From Definition 4 we have a direct proof that $\mathtt{bte}(\ \mathtt{A},\ m_1 :: \cdots :: m_n\ ) = \mathtt{bte}(\mathtt{A}, m_1)\ ;\ \ldots\ ;\ \mathtt{bte}(\mathtt{A}, m_n)$.

Now, from the reduction on $\lfloor\mathsf{C}|_{\mathsf{Send}}\rceil$ we know that

$$D'(\ k[\mathtt{A}\langle\mathtt{B}]\ ) = m' = \tilde{m} :: (o_j, v)$$

And therefore, $\mathtt{bte}(\mathtt{A}, m') = T;\ \mathtt{bte}(\mathtt{A},\ (o_j, v))$. From the reductions on Γ and G, we observe that the reduction on G do not affect the context $\mathcal{G}$ (which contains local type T), thus, by the rules of the definition of the Buffer Type Projection (Figure 14), we have

$$[\![G']\!]^{\mathtt{A}}_{\mathtt{B}} = T; \&\mathtt{A}.o_j(U_j)$$

Hence, from the reduction on rule $[\Gamma|_{\mathsf{Send}}]$, we know that $\Gamma'(\ k[\mathtt{B}\rangle\mathtt{A}]\ ) = \Gamma'(\ [\![G']\!]^{\mathtt{A}}_{\mathtt{B}}\ ) = T; \&\mathtt{A}.o_j(U_j)$. Finally, from the typing rule $[\mathsf{T}|_{\mathsf{Send}}]$ we know that $\mathtt{p}.e \vdash U_j$ and from reduction rule $[\mathsf{C}|_{\mathsf{Send}}]$ that $v = \mathbf{eval}(\ e, D(\mathtt{p})\ )$, thus v has type U_j. Hence, $\mathtt{bte}(\ \mathtt{A}, (o_j, v)\ ) = \&\mathtt{A}.o_j(U_j)$ and

$$\Gamma'(\ k[\mathtt{A}\rangle\mathtt{B}]\ ) = T; \&\mathtt{A}.o_j(U_j) = \mathtt{bte}(\ \mathtt{A}, D'(k[\mathtt{A}\rangle\mathtt{B}])\ )$$

□

Case $[\mathsf{C}|_{\mathsf{Recv}}]$

The case is:

$$\frac{j \in I \quad D, k\colon \mathtt{A} \longrightarrow \mathtt{q}[\mathtt{B}].o_j(x_j) \blacktriangleright D'}{D,\ k\colon \mathtt{A} \longrightarrow \mathtt{q}[\mathtt{B}].\{o_i(x_i); C_i\}_{i\in I} \xrightarrow{k:\mathtt{A}\rangle\mathtt{B}.o_j(x_j)} D',\ C_j}\ [\mathsf{C}|_{\mathsf{Recv}}]$$

(T|7.2) has reductum $C' = C_j$. Since we could apply $[\mathsf{C}|_{\mathsf{Recv}}]$, we know that $D(k[\mathtt{A}\rangle\mathtt{B}]) = (o_j, v) :: \tilde{m}$. Let $D_1 = D\big[\mathtt{q} \mapsto D(\mathtt{q})[x \mapsto v]\big]$, from the application of rule $[\mathsf{P}|_{\mathsf{Recv}}]$, we know that $D' = D_1\big[k[\mathtt{A}\rangle\mathtt{B}] \mapsto \tilde{m}\big]$. To prove (T|7.3) we must prove that rule $[\Gamma|_{\mathsf{Recv}}]$ is applicable.

Since (T|7.1) holds $\mathbf{pco}(\Gamma)$ and $\Gamma \vdash D$ hold and therefore we know that, by (D|3.2), $\Gamma(k[\mathtt{A}\rangle\mathtt{B}]) = \mathtt{bte}(\mathtt{A},\ (o_j, v) ::\ \tilde{m})$.

Let $\vdash v : U_j$, then $\mathtt{bte}(\mathtt{A},\ (o_j, v) ::\ \tilde{m}) = \&\mathtt{A}.o_j(U_j); T$ where $T = \mathtt{bte}(\mathtt{A},\ \tilde{m})$ by Definition 4 and $\Gamma(k[\mathtt{A}\rangle\mathtt{B}]) = \&\mathtt{A}.o_j(U_j); T$. Since $\mathbf{pco}(\Gamma)$ holds, there exists a global type G for session k such that $G = \mathcal{G}[\mathtt{A}\rangle\mathtt{B}.o_j(U_j); G_j]$. Let π be the reduction of G with rules $[\mathsf{G}|_{\mathsf{Eq}}]$ and $[\mathsf{G}|_{\mathsf{Recv}}]$, we observe the following derivation:

$$\pi = \left\{ \begin{array}{l|l} G \simeq_{\mathtt{G}} G_1 \quad \dfrac{}{G_1 \xrightarrow{\gamma} G'}\ [\mathsf{G}|_{\mathsf{Recv}}] \quad G \simeq_{\mathtt{G}} G' & G_1 = \mathtt{A}\rangle\mathtt{B}.o_j(U_j); \mathcal{G}[G_j] \\ \qquad\qquad \vdots\ [\mathsf{G}|_{\mathsf{Eq}}] & \\ \qquad\qquad G \xrightarrow{\gamma} G' & G' = \mathcal{G}[G_j] \\ & \gamma = \mathtt{A}\rangle\mathtt{B}.o_j(U_j) \end{array} \right.$$

In the reductions, since C, D reduces with $\beta = k : \mathtt{A}\rangle\mathtt{B}.o_j(x_j)$ and G types C, D in Γ, there are no other exchanges from $\mathtt{A}$ to $\mathtt{B}$ in G that could prevent from obtaining, after a finite number of derivations on rule $[\mathsf{G}|_{\mathsf{Eq}}]$, the swap-equivalence $G \simeq_{\mathtt{G}} G_1$. Then, applying rule $[\mathsf{G}|_{\mathsf{Send}}]$, G_1 can reduce to G'.

Given π, we can use it to write the reduction at the level the typing environment Γ, applying rule $[\Gamma|_{\mathsf{Recv}}]$. Below, we consider $\Gamma = \Gamma_*, \Gamma_k$ where Γ_k contains all and only typings of session k in Γ.

$$\frac{k \notin \Gamma_* \quad \Gamma_k \subseteq [\![G]\!]_k \quad \{k[\mathtt{A}]\colon T, k[\mathtt{B}]\colon T'\} \in \Gamma_k \quad \Gamma_* \vdash \mathtt{q}\colon k[\mathtt{B}] \quad G \overset{\pi}{\vdots} \xrightarrow{\mathtt{A}\rangle\mathtt{B}.o_j} G'}{\Gamma_*, \Gamma_k \xrightarrow{k:\mathtt{A}\rangle\mathtt{B}.o_j(x)} \Gamma_*, \{k[\mathtt{C}]\colon [\![G']\!]_{\mathtt{C}} \mid k[\mathtt{C}] \in \Gamma_k\}, \{k[\mathtt{C}\rangle\mathtt{D}]\colon [\![G']\!]^{\mathtt{D}}_{\mathtt{C}} \mid k[\mathtt{C}\rangle\mathtt{D}] \in \Gamma_k\}, \mathtt{q}.x\colon U_j}\ [\Gamma|_{\mathsf{Recv}}]$$

Hence (T|7.3) holds and $\Gamma' = \Gamma_*, \{[\![G']\!]_{\mathtt{C}} \mid k[\mathtt{C}] \in \Gamma_k\}, \mathtt{q}.x\colon U_j$.

(T|7.4) holds if we can apply rule $[\mathsf{T}|_{\mathsf{DC}}]$ on $\Gamma' \vdash D', C'$

$$\frac{\mathbf{pco}(\Gamma') \quad \Gamma' \vdash C' \quad \Gamma' \vdash D'}{\Gamma' \vdash D', C'} \; \lfloor \mathsf{T} | \mathsf{DC} \rceil$$

and we need to prove ① $\mathbf{pco}(\Gamma')$, ② $\Gamma' \vdash C'$, and ③ $\Gamma' \vdash D'$

The proof of ① for this case is similar to that of ① for case $\lfloor \mathsf{C} | \mathsf{Send} \rceil$.

Proof of ②. From (T|7.1), partitioning $\Gamma = \Gamma_1, k[\mathtt{B}] : \&\mathtt{A}.o_j(U_j); [\![G_j]\!]_{\mathtt{B}}$ and since $j \in I$ from rule $\lfloor \mathsf{C} | \mathsf{Recv} \rceil$, we can write the derivation

$$\frac{\mathbf{pco}(\Gamma) \quad \Gamma \vdash D \quad \dfrac{j \in I \quad \Gamma_1 \vdash \mathtt{q} : k[\mathtt{B}] \quad \Gamma_1, \mathtt{q}.x_j : U_j, k[\mathtt{B}] : [\![G_j]\!]_{\mathtt{B}} \vdash C_j}{\Gamma_1, k[\mathtt{B}] : \&\mathtt{A}.o_j(U_j); [\![G_j]\!]_{\mathtt{B}} \vdash k : \mathtt{A} \longrightarrow \mathtt{q}[\mathtt{B}].\{o_i(x_i); C_i\}_{i \in I}} \; \lfloor \mathsf{T} | \mathsf{Recv} \rceil}{\Gamma \vdash D, k : \mathtt{A} \longrightarrow \mathtt{q}[\mathtt{B}].\{o_i(x_i); C_i\}_{i \in I}} \; \lfloor \mathsf{T} | \mathsf{DC} \rceil$$

hence we know that $\Gamma_1, \mathtt{q}.x_j : U_j, k[\mathtt{B}] : [\![G_j]\!]_{\mathtt{B}} \vdash C_j$.

Let $\Gamma'' = \Gamma_1, \mathtt{q}.x_j : U_j, k[\mathtt{B}] : [\![G_j]\!]_{\mathtt{B}} \vdash C_j$ and $\Gamma'_k = \Gamma_1 \setminus \Gamma_* = \Gamma_k \setminus \{k[\mathtt{B}] : [\![G]\!]_{\mathtt{B}}\}$. We can write $\Gamma'' = \Gamma_*, \Gamma'_k, k[\mathtt{B}] : [\![G_j]\!]_{\mathtt{B}}$. Similarly to ② for case $\lfloor \mathsf{C} | \mathsf{Send} \rceil$, $\Gamma''(k[\mathtt{A}\rangle\mathtt{B}]) \neq \Gamma'(k[\mathtt{A}\rangle\mathtt{B}])$, but we omit to consider buffer types as they are irrelevant for the typing of choreographies by Lemma 2. For all sessions in Γ'', their local typings are the same as in Γ'. We consider in particular k on which we applied the reduction for this case for which it holds

$$\forall\ k[\mathtt{C}] \in \Gamma'',\ \Gamma''(k[\mathtt{C}]) = \Gamma'(k[\mathtt{C}]) = [\![G']\!]_{\mathtt{C}}$$

□

Proof of ③. To prove $\Gamma' \vdash D'$ we prove the conditions in Definition 3. (D|3.1) holds from the application of rule $\lfloor \mathsf{P} | \mathsf{Recv} \rceil$, (T|7.1), and the construction of Γ'. (D|3.2) holds for all $\mathtt{p}.x$ from the application of rule $\lfloor \mathsf{P} | \mathsf{Recv} \rceil$, (T|7.1), and the construction of Γ', except for $\mathtt{q}.x_j$ which is not defined in Γ. However the condition holds by construction of $\Gamma' = \Gamma_1, \mathtt{q}.x_j : U_j, k[\mathtt{B}] : [\![G_j]\!]_{\mathtt{B}}$. (D|3.2) holds for all sessions $k' \neq k$ by the application of rule $\lfloor \mathsf{P} | \mathsf{Recv} \rceil$ and the construction of Γ'. The same holds true for session k and any process $\mathtt{p} : k[\mathtt{C}] \in \Gamma \mid \mathtt{C} \neq \mathtt{B}$.

For $\mathtt{q} : k[\mathtt{B}]$ and role $\mathtt{A}$ we know from the application of $\lfloor \mathsf{C} | \mathsf{Send} \rceil$ that $D'(k[\mathtt{A}\rangle\mathtt{B}]) = \tilde{m}$. Since we took G such that $[\![G]\!]^{\mathtt{A}}_{\mathtt{B}} = \&\mathtt{A}.o_j(U_j); T$, where $T = \mathtt{bte}(\mathtt{A}, \tilde{m})$, then $[\![G']\!]^{\mathtt{A}}_{\mathtt{B}} = T$.

□

Case $\lfloor \mathsf{C} | \mathsf{Start} \rceil$

The case is:

$$\frac{D \# k', \tilde{\mathtt{r}} \quad \delta = \mathtt{start}\ k' :\ l.\mathtt{p}[\mathtt{A}], \widetilde{l.\mathtt{r}[\mathtt{B}]} \quad D, \delta \blacktriangleright D'}{D,\ \mathtt{start}\ k : \mathtt{p}[\mathtt{A}] \leftrightarrow \widetilde{l.\mathtt{q}[\mathtt{B}]}; C \quad \xrightarrow{\tau} \quad D',\ C[k'/k][\tilde{\mathtt{r}}/\tilde{\mathtt{q}}]} \; \lfloor \mathsf{C} | \mathsf{Start} \rceil$$

Where (T|7.5) has $C' = C[k'/k][\tilde{\mathtt{r}}/\tilde{\mathtt{q}}]$. D' is defined non-deterministically but abides the requirements defined in rule $\lfloor \mathsf{P} | \mathsf{Start} \rceil$. Let $\widetilde{\mathtt{s}[\mathtt{C}]} = \mathtt{p}[\mathtt{A}], \widetilde{\mathtt{r}[\mathtt{B}]}$. Since (T|7.1) holds, we can apply rule $\lfloor \mathsf{T} | \mathsf{Start} \rceil$. We partition $\Gamma = \Gamma_1, \tilde{l} : G\langle \mathtt{A} | \tilde{\mathtt{B}} | \tilde{\mathtt{B}} \rangle$

$$\frac{\Gamma_1, \tilde{l} : G\langle \mathtt{A} | \tilde{\mathtt{B}} | \tilde{\mathtt{B}} \rangle, \mathbf{init}(\ \widetilde{\mathtt{s}'[\mathtt{C}]}\ ,\ k,\ G\ ) \vdash C \quad \widetilde{\mathtt{s}'[\mathtt{C}]} = \mathtt{p}[\mathtt{A}], \widetilde{\mathtt{q}[\mathtt{B}]} \quad \tilde{\mathtt{q}} \notin \Gamma_1}{\Gamma_1, \tilde{l} : G\langle \mathtt{A} | \tilde{\mathtt{B}} | \tilde{\mathtt{B}} \rangle \vdash \mathtt{start}\ k : \mathtt{p}[\mathtt{A}] \leftrightarrow \widetilde{l.\mathtt{q}[\mathtt{B}]}; C} \; \lfloor \mathsf{T} | \mathsf{Start} \rceil$$

Coherently with the semantics of rule $\lfloor \mathsf{C} | \mathsf{Start} \rceil$, we take $\Gamma' = \Gamma, \mathbf{init}(\ \widetilde{\mathtt{s}[\mathtt{C}]},\ k',\ G\ )$ — obtainable from the typing environment in the left-most premise of rule $\lfloor \mathsf{T} | \mathsf{Start} \rceil$, α-renaming: *i*) typings on session k to session k' and *ii*) process identifies $\tilde{\mathtt{q}}$ to $\tilde{\mathtt{r}}$ in $\widetilde{\mathtt{s}'[\mathtt{C}]}$ (i.e., such that

$\widetilde{\mathtt{s}[\mathtt{C}]} = [\ \widetilde{\mathtt{r}[\mathtt{C}]}\ /\ \widetilde{\mathtt{q}[\mathtt{C}]}\ ]\ \widetilde{\mathtt{s}'[\mathtt{C}]}$) — and we prove the case by proving that we can apply rule $[\mathsf{T}|_{\mathsf{DC}}]$ on $\Gamma' \vdash D', C'$, i.e, that the following hold: ① $\mathbf{pco}(\Gamma')$, ② $\Gamma' \vdash C'$, and ③ $\Gamma' \vdash D'$.

Proof of ①. ① holds for all session $k'' \in \Gamma', k'' \neq k'$ by (T|7.1). For session k' ① holds by construction. □

Proof of ②. By (T|7.1) we could apply $[\mathsf{T}|_{\mathsf{Start}}]$ where $\Gamma, \mathbf{init}(\ \widetilde{\mathtt{s}[\mathtt{C}]},\ k,\ G\ ) \vdash C$. Since Γ' is obtained by α-renaming of the left-most premise of Rule $[\mathsf{T}|_{\mathsf{Start}}]$, which types the continuation C, Γ' types $C[k'/k][\tilde{\mathtt{r}}/\tilde{\mathtt{q}}]$ and ② holds by construction. □

Proof of ③. To prove ③ we prove the conditions in Definition 3. (D|3.1–D|3.2) hold by the application of rule $[\mathsf{D}|_{\mathsf{Start}}]$ and the construction of Γ'. □

Case $[\mathsf{C}|_{\mathsf{PStart}}]$

The case is:

$$\frac{\begin{array}{l} i \in \{1,\dots,n\} \qquad D \# k', \tilde{\mathtt{r}} \qquad \{\ \widetilde{l.\mathtt{B}}\ \} = \biguplus_i \{\ \widetilde{l_i.\mathtt{B}_i}\ \} \qquad \{\tilde{\mathtt{r}}\} = \bigcup_i \{\tilde{\mathtt{r}}_i\} \\ \mathtt{p} \in D(l) \qquad \delta = \mathtt{start}\ k' :\ l.\mathtt{p}[\mathtt{A}], \widetilde{l_1.\mathtt{r}_1[\mathtt{B}_1]}, \dots, \widetilde{l_n.\mathtt{r}_n[\mathtt{B}_n]} \qquad D, \delta \blacktriangleright D' \end{array}}{\begin{array}{l} D, \mathtt{req}\ k : \mathtt{p}[\mathtt{A}] \leftrightarrow \widetilde{l.\mathtt{B}}; C \mid \prod_i \left(\mathtt{acc}\ k :\ \widetilde{l_i.\mathtt{q}_i[\mathtt{B}_i]}; C_i\right) \xrightarrow{\tau} \\ \qquad D',\ C[k'/k] \mid \prod_i \left(\ C_i[k'/k][\tilde{\mathtt{r}}_i/\tilde{\mathtt{q}}_i]\ \right) \mid \prod_i \left(\mathtt{acc}\ k :\ \widetilde{l_i.\mathtt{q}_i[\mathtt{B}_i]}; C_i\right) \end{array}}\ [\mathsf{C}|_{\mathsf{PStart}}]$$

Where (T|7.5) has $C' = C[k'/k] \mid \prod_i \left(\ C_i[k'/k][\tilde{\mathtt{r}}_i/\tilde{\mathtt{q}}_i]\ \right) \mid \prod_i \left(\mathtt{acc}\ k :\ \widetilde{l_i.\mathtt{q}_i[\mathtt{B}_i]}; C_i\right)$. D' is defined non-deterministically but abides the requirements defined in rule $[\mathsf{D}|_{\mathsf{Start}}]$.

We partition Γ such that:

- $\Gamma = \Gamma_r, \Gamma_a$
- $\Gamma_r \vdash \tilde{l} \colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\varnothing\rangle$
- $\Gamma_a \vdash \tilde{l} \colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{B}}\rangle$
- $\Gamma_a = \Gamma_1, \tilde{l} \colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\widetilde{\mathtt{B}_1}\rangle, \cdots, \Gamma_n, \tilde{l} \colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\widetilde{\mathtt{B}_n}\rangle$
- $\Gamma_a^i = \Gamma_i, \tilde{l} \colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\widetilde{\mathtt{B}_i}\rangle, \cdots, \Gamma_n, \tilde{l} \colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\widetilde{\mathtt{B}_n}\rangle$

and we can write the derivation

$$\frac{\mathbf{pco}(\Gamma) \quad \Gamma \vdash D \qquad \dfrac{\dfrac{\dfrac{\Gamma_r, \mathtt{p} \colon k[\mathtt{A}], k[\mathtt{A}] \colon [\![G]\!]_{\mathtt{A}} \vdash C \quad \Gamma_r \vdash \tilde{l} \colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\varnothing\rangle}{\Gamma_r \vdash \mathtt{req}\ k : \mathtt{p}[\mathtt{A}] \leftrightarrow \widetilde{l.\mathtt{B}}; C}\ [\mathsf{T}|_{\mathsf{Req}}] \quad \Delta_1}{\Gamma \vdash \mathtt{req}\ k : \mathtt{p}[\mathtt{A}] \leftrightarrow \widetilde{l.\mathtt{B}}; C \mid \prod_{i \in I} \left(\mathtt{acc}\ k :\ \widetilde{l_i.\mathtt{q}_i[\mathtt{B}_i]}; C_i\right)}\ [\mathsf{T}|_{\mathsf{Par}}]}{\Gamma \vdash D, \mathtt{req}\ k : \mathtt{p}[\mathtt{A}] \leftrightarrow \widetilde{l.\mathtt{B}}; C \mid \prod_{i \in I} \left(\mathtt{acc}\ k :\ \widetilde{l_i.\mathtt{q}_i[\mathtt{B}_i]}; C_i\right)}\ [\mathsf{T}|_{\mathsf{DC}}]$$

$$\Delta_i = \left\{ \frac{\tilde{l}_i \subseteq \tilde{l} \quad \Gamma_i, \tilde{l} \colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\varnothing\rangle, \mathbf{init}(\ \widetilde{\mathtt{q}_i[\mathtt{B}_i]},\ k,\ G\ ) \vdash C_i \quad \tilde{\mathtt{q}}_i \notin \Gamma}{\dfrac{\Gamma_i, \tilde{l} \colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\widetilde{\mathtt{B}_i}\rangle \vdash \mathtt{acc}\ k :\ \widetilde{l_i.\mathtt{q}_i[\mathtt{B}_i]}; C_i \quad \Delta_{i+1}}{\Gamma_a^i \vdash \mathtt{acc}\ k :\ \widetilde{l_i.\mathtt{q}_i[\mathtt{B}_i]}; C_i \mid \prod_{j \in I \setminus \{1,\cdots,i\}} \left(\mathtt{acc}\ k :\ \widetilde{l_j.\mathtt{q}_j[\mathtt{B}_j]}; C_j\right)}\ [\mathsf{T}|_{\mathsf{Par}}]}\ [\mathsf{T}|_{\mathsf{Acc}}] \right.$$

Let $\widetilde{\mathtt{s}[\mathtt{C}]} = \mathtt{p}[\mathtt{A}], \widetilde{\mathtt{r}_1[\mathtt{B}_1]}, \cdots, \widetilde{\mathtt{r}_n[\mathtt{B}_n]}$.

To prove (T|7.6) we take

$$\Gamma' = \Gamma, \mathbf{init}(\ \widetilde{\mathtt{s}[\mathtt{C}]},\ k',\ G\ ) = \Gamma_r, \Gamma_a, \mathbf{init}(\ \widetilde{\mathtt{s}[\mathtt{C}]},\ k',\ G\ )$$

and we partition $\mathbf{init}(\ \widetilde{\mathtt{s}[\mathtt{C}]},\ k',\ G\ )$ such that

$$\Gamma' = \Gamma'_r, \Gamma'_a, \Gamma_a$$

Where

- $\Gamma'_r = \Gamma_r, \mathbf{init}(\ \mathtt{p}[\mathtt{A}],\ k',\ G\ )$

– $\Gamma'_a = \Gamma'_1, \cdots, \Gamma'_n$
– $\Gamma'_i = \Gamma_i, \tilde{l}\colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\varnothing\rangle, \mathbf{init}(\ \widetilde{\mathsf{r}_i[\mathtt{B}_i]},\ k',\ G\ )$ where $i \in \{1, \ldots, n\}$

To prove (T|7.6) we must prove we can apply rule $[\mathsf{T}|_{\mathsf{DC}}]$ on $\Gamma' \vdash D', C'$.

$$\dfrac{\textcircled{1}\ \mathbf{pco}(\Gamma') \qquad \textcircled{3}\ \Gamma' \vdash D' \qquad \textcircled{2}\left\{\dfrac{\textcircled{2a}\ \Gamma'_r \vdash C[k'/k] \qquad \dfrac{\textcircled{2b}\ \Gamma'_a \vdash \prod_i (\ C_i[k'/k][\tilde{\mathsf{r}}_i/\tilde{\mathsf{q}}_i]\ ) \quad \textcircled{2c}\ \Gamma_a \vdash \prod_i (\mathtt{acc}\ k : \widetilde{l_i.\mathsf{q}_i[\mathtt{B}_i]}; C_i)}{\Gamma'_a, \Gamma_a \vdash \prod_i (\ C_i[k'/k][\tilde{\mathsf{r}}_i/\tilde{\mathsf{q}}_i]\ ) \mid \prod_i (\mathtt{acc}\ k : \widetilde{l_i.\mathsf{q}_i[\mathtt{B}_i]}; C_i)}\ [\mathsf{T}|_{\mathsf{Par}}]}{\Gamma' \vdash C[k'/k] \mid \prod_i (\ C_i[k'/k][\tilde{\mathsf{r}}_i/\tilde{\mathsf{q}}_i]\ ) \mid \prod_i (\mathtt{acc}\ k : \widetilde{l_i.\mathsf{q}_i[\mathtt{B}_i]}; C_i)}\ [\mathsf{T}|_{\mathsf{Par}}]\right.}{\Gamma' \vdash D', C'}\ [\mathsf{T}|_{\mathsf{DC}}]$$

Proof of ①. ① holds by construction. □

Proof of ②. ② holds as

– ②a holds by α-renaming $(\Gamma_r, \mathsf{p}\colon k[\mathtt{A}], k[\mathtt{A}]\colon [\![G]\!]_{\mathtt{A}})[k'/k] \vdash C[k'/k]$ and by omitting to consider buffer types as of Lemma 2;
– similarly to ②a, ②b holds by α-renaming on the derivation of

$$(\ \Gamma_i, \tilde{l}\colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\varnothing\rangle, \mathbf{init}(\ \widetilde{\mathsf{q}_i[\mathtt{B}_i]},\ k,\ G\ )\ )[k'/k][\tilde{\mathsf{r}}_i/\tilde{\mathsf{q}}_i] \vdash C_i[k'/k][\tilde{\mathsf{r}}_i/\tilde{\mathsf{q}}_i]$$

and by Lemma 2;
– ②c holds by (T|7.1).

□

Proof of ③. The proof of ③ of this case is similar to the of ③ for Case $[\mathsf{C}|_{\mathsf{Start}}]$. □

Case $[\mathsf{C}|_{\mathsf{Cond}}]$

The case is:

$$\dfrac{i = 1 \text{ if } \mathbf{eval}(\ e, D(\mathsf{p})\ ) = \mathtt{true}, i = 2 \text{ otherwise}}{D,\ \mathtt{if}\ \mathsf{p}.e\ \{C_1\}\ \mathtt{else}\ \{C_2\} \quad \xrightarrow{\tau} \quad D,\ C_i}\ [\mathsf{C}|_{\mathsf{Cond}}]$$

In (T|7.5) $D' = D$ and we have two cases for $C' = C_1$ or $C' = C_2$.

From (T|7.1) we can write

$$\dfrac{\Gamma \vdash \mathsf{p}.e\colon \mathbf{bool} \qquad \Gamma \vdash C_1 \qquad \Gamma \vdash C_2}{\Gamma \vdash \mathtt{if}\ \mathsf{p}.e\ \{C_1\}\ \mathtt{else}\ \{C_2\}}\ [\mathsf{T}|_{\mathsf{Cond}}]$$

The proof of (T|7.6) follows directly from the premises of the typing derivation as $\Gamma \vdash D = D'$ and in both cases that $C' = C_1$ or $C' = C_2$ it holds that $\Gamma \vdash C'$ from the premises of $[\mathsf{T}|_{\mathsf{Cond}}]$.

Case $[\mathsf{C}|_{\mathsf{Ctx}}]$

The case is:

$$\dfrac{D, C_1 \xrightarrow{\beta} D', C'_1}{D, \mathtt{def}\ X = C_2\ \mathtt{in}\ C_1 \quad \xrightarrow{\beta} \quad D', \mathtt{def}\ X = C_2\ \mathtt{in}\ C'_1}\ [\mathsf{C}|_{\mathsf{Ctx}}]$$

From (T|7.1) we know that, $\Gamma = \Gamma_1, X\colon \Gamma_x$

$$\dfrac{\mathbf{pco}(\Gamma) \qquad \dfrac{\Gamma_1, X\colon \Gamma_x \vdash C_1 \qquad \Gamma_x, X\colon \Gamma_x \vdash C_2 \qquad \Gamma_x|_{\mathbf{locs}} \subseteq \Gamma}{\Gamma \vdash \mathtt{def}\ X = C_2\ \mathtt{in}\ C_1}\ [\mathsf{T}|_{\mathsf{Def}}] \qquad \Gamma \vdash D}{\Gamma \vdash D, \mathtt{def}\ X = C_2\ \mathtt{in}\ C_1}\ [\mathsf{T}|_{\mathsf{DC}}]$$

The proof is divided in two cases on the type of β.

Case $\beta \neq \tau$

D, C_1 reduces on some session k. By the induction hypothesis since $\Gamma \vdash D, C_1$ we can find Γ' such that (T|7.3) holds. We prove (T|7.4) by proving that we can apply $\lfloor\!\lceil \mathsf{T}|_{\mathsf{DC}} \rceil\!\rfloor$ on $\Gamma' \vdash D', \mathtt{def}\ X = C_2\ \mathtt{in}\ C_1'$ and therefore that ① $\mathbf{pco}(\Gamma')$ holds, ② $\Gamma' \vdash \mathtt{def}\ X = C_2\ \mathtt{in}\ C_1$ and ③ $\Gamma' \vdash D'$.

① holds by the construction of Γ' and ③ holds by the induction hypothesis.

To prove ② we have to prove that $\Gamma' \vdash X \colon C_2$ and $\Gamma_x|_{\mathbf{locs}} \subseteq \Gamma'$.

From the induction hypothesis we have that $\Gamma \xrightarrow{\beta} \Gamma'$ and $\Gamma' \vdash D', C_1'$. By construction of Γ' it holds that $\Gamma' = \Gamma'_*, \Gamma'_k$ where $\Gamma' \cap \Gamma = \Gamma_*$ such that $k \notin \Gamma_*$ and $\Gamma = \Gamma_*, \Gamma_k$ where $\Gamma_k \subseteq [\![G]\!]_k$ for some G. Therefore it holds that $\Gamma_* \vdash X \colon \Gamma_x$ and thus that $\Gamma' \vdash X \colon \Gamma_x$. The same applies to $\Gamma_x|_{\mathbf{locs}} \subseteq \Gamma_*$ which proves $\Gamma_x|_{\mathbf{locs}} \subseteq \Gamma'$.

Case $\beta = \tau$

from the induction hypothesis, for any considered derivation we have $\Gamma \subseteq \Gamma'$. We prove (T|7.6) by proving that we can apply $\lfloor\!\lceil \mathsf{T}|_{\mathsf{DC}} \rceil\!\rfloor$ on $\Gamma' \vdash D', \mathtt{def}\ X = C_2\ \mathtt{in}\ C_1'$. ①, ②, and ③ hold by construction of Γ'.

Case $\lfloor\!\lceil \mathsf{C}|_{\mathsf{Par}} \rceil\!\rfloor$

The case is:

$$\frac{D, C_1 \quad \xrightarrow{\beta} \quad D', C_1'}{D, C_1 \mid C_2 \quad \xrightarrow{\beta} \quad D', C_1' \mid C_2}\ \lfloor\!\lceil \mathsf{C}|_{\mathsf{Par}} \rceil\!\rfloor$$

From (T|7.1) we have the derivation below, with Γ partitioned as $\Gamma = \Gamma_1, \Gamma_2$

$$\frac{\mathbf{pco}(\Gamma) \qquad \dfrac{\Gamma_1 \vdash C_1 \quad \Gamma_2 \vdash C_2}{\Gamma \vdash C_1 \mid C_2}\ \lfloor\!\lceil \mathsf{T}|_{\mathsf{Par}} \rceil\!\rfloor \qquad \Gamma \vdash D}{\Gamma \vdash D, C_1 \mid C_2}\ \lfloor\!\lceil \mathsf{T}|_{\mathsf{DC}} \rceil\!\rfloor$$

The proof is divided in two cases on the type of β.

Case $\beta \neq \tau$

D, C_1 reduces on some session k. By the induction hypothesis and since $\Gamma_1 \vdash D, C_1$ we can find Γ_1' such that $\Gamma_1 \xrightarrow{\beta} \Gamma_1'$ and $\Gamma_1' \vdash D', C_1'$. Then we take $\Gamma' = \Gamma_1', \Gamma_2$ which proves (T|7.3) to hold. We prove (T|7.4) by proving that we can apply $\lfloor\!\lceil \mathsf{T}|_{\mathsf{DC}} \rceil\!\rfloor$ on $\Gamma' \vdash D', C_1' \mid C_2$ and therefore that ① $\mathbf{pco}(\Gamma')$, ② $\Gamma' \vdash C_1' \mid C_2$ and ③ $\Gamma' \vdash D'$ hold. ①, ②, and ③ hold by construction and the induction hypothesis.

Case $\beta = \tau$

from the induction hypothesis, for any derivation we have that $\Gamma_1' \vdash D', C_1'$ and $\Gamma_1 \subseteq \Gamma_1'$. Also in this case we take $\Gamma' = \Gamma_1', \Gamma_2$ and prove (T|7.6) by proving that we can apply $\lfloor\!\lceil \mathsf{T}|_{\mathsf{DC}} \rceil\!\rfloor$ on $\Gamma' \vdash D', C_1' \mid C_2$. ①, ②, and ③ hold by construction of Γ' and the induction hypothesis.

Case $\lfloor\!\lceil \mathsf{C}|_{\mathsf{Eq}} \rceil\!\rfloor$

The case is:

$$\frac{\mathcal{R} \in \{\, \equiv_{\mathtt{C}}, \simeq_{\mathtt{C}} \,\} \quad C\,\mathcal{R}\,C_1 \quad D, C_1 \xrightarrow{\beta} D', C_1' \quad C_1'\,\mathcal{R}\,C'}{D, C \quad \xrightarrow{\beta} \quad D', C'}\ \lfloor\!\lceil \mathsf{C}|_{\mathsf{Eq}} \rceil\!\rfloor$$

The proof is divided into two subcases on the type of $\mathcal{R}$.

Case $\mathcal{R} = \equiv_{\mathtt{C}}$

The case is proved by induction hypothesis and Lemma 5.

Case $\mathcal{R} = \simeq_{\mathsf{C}}$
The case is proved by induction hypothesis and Lemma 6.

□

The proof of Theorem 2 follows directly from the proof of Theorem 7 and Lemma 4.

B.2. **Proof of Deadlock Freedom.** We report below the statement of Theorem 3 enriched with pointers for clearer referencing in the proof.

Theorem 3 (Deadlock-freedom)
(D3.1) $\Gamma \vdash D, C$ and (D3.2) $\mathbf{co}(\Gamma)$ imply that either (D3.3) $C \equiv_{\mathsf{C}} \mathbf{0}$ or (D3.4) there exist D' and C' such that $D, C \to D', C'$.

Like in [CM13, MY13], frontend choreographies enjoy deadlock freedom, provided that they *i*) do not contain free variable names and *ii*) are *well-sorted*, i.e., have no undefined procedure calls. Notably, well-sortedness is guaranteed by the type system.

Proof . Proof by induction on the structure of C.

Case $C \equiv_{\mathsf{C}} \mathbf{0}$
trivial.

Case $C = k : \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o; C_1$
from (D3.1) and (D3.2) we know that the requirements of $\lfloor^{\mathsf{P}}_{\mathsf{Send}}$ hold and we can find D' such that $D, k : \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o \blacktriangleright D'$. We can apply Rule $\lfloor^{\mathsf{C}}_{\mathsf{Send}}$ for which $C' = C_1$.

Case $C = k : \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{q}[\mathtt{B}].o(x); C_1$
since (D3.1) holds both receiver and sender are typed by Γ. We apply rule $\lfloor^{\mathsf{C}}_{\mathsf{Eq}}\rceil$ to split the complete term into respectively a send and a receive partial terms, and similarly to the previous case, we apply rule $\lfloor^{\mathsf{C}}_{\mathsf{Send}}\rceil$, for which $C' = k : \mathtt{A} \longrightarrow \mathtt{q}[\mathtt{B}].o(x); C_1$.

Case $C = k : \mathtt{A} \longrightarrow \mathtt{q}[\mathtt{B}].\{o_i(x_i); C_i\}_{i \in I}$
from (D3.1) and (D3.2) we know that the requirements of Rule $\lfloor^{\mathsf{P}}_{\mathsf{Recv}}\rceil$ hold and $D(k[\mathtt{A}\rangle\mathtt{B}]) = (o_j, t_m) :: \tilde{m}$ for some $j \in I$. We can find D' such that $D, k : \mathtt{A} \longrightarrow \mathtt{q}[\mathtt{B}].o_j(x_j) \blacktriangleright D'$ and apply Rule $\lfloor^{\mathsf{C}}_{\mathsf{Recv}}\rceil$ for which $C' = C_j$.

Case $C = \mathtt{start}\ k : \mathtt{p}[\mathtt{A}] \leftarrow\!\!\rightarrow \widetilde{l.\mathtt{q}[\mathtt{B}]}; C_1$
from (D3.1) and (D3.2) $\lfloor^{\mathsf{P}}_{\mathsf{Start}}\rceil$ applies and we can find D' such that $D, \mathtt{start}\ k' : l.\mathtt{p}[\mathtt{A}], \widetilde{l.\mathtt{r}[\mathtt{B}]} \blacktriangleright D'$ for some $k', \tilde{\mathtt{r}}$ fresh. We can apply Rule $\lfloor^{\mathsf{C}}_{\mathsf{Start}}\rceil$ for which $C' = C_1[k'/k][\tilde{\mathtt{r}}/\tilde{\mathtt{q}}]$.

Case $C = \mathtt{req}\ k : \mathtt{p}[\mathtt{A}] \leftarrow\!\!\rightarrow \widetilde{l.\mathtt{B}}; C \mid \prod_{i=1}^{n} \left(\mathtt{acc}\ k : \widetilde{l_i.\mathtt{q}_i[\mathtt{B}_i]}; C_i\right)$
similarly to the previous case, the requirements of $\lfloor^{\mathsf{P}}_{\mathsf{Start}}\rceil$ hold and we can find D' such that $D, \mathtt{start}\ k' : l.\mathtt{p}[\mathtt{A}], \widetilde{l_1.\mathtt{r}_1[\mathtt{B}_1]}, \ldots, \widetilde{l_n.\mathtt{r}_n[\mathtt{B}_n]} \blacktriangleright D'$ for some k' and $\tilde{\mathtt{r}}_1, \cdots, \tilde{\mathtt{r}}_n$ fresh. We can apply Rule $\lfloor^{\mathsf{C}}_{\mathsf{PStart}}\rceil$ for which
$C' = C[k'/k] \mid \prod_{i=1}^{n} C_i[k'/k][\tilde{\mathtt{r}}_1/\tilde{\mathtt{q}}_1] \mid \prod_{i=1}^{n} \left(\mathtt{acc}\ k : \widetilde{l_i.\mathtt{q}_i[\mathtt{B}_i]}; C_i\right)$.

Case $C = C_1 \mid C_2$
we can apply the induction hypothesis and Rule $\lfloor^{\mathsf{C}}_{\mathsf{Par}}\rceil$ such that $D, C_1 \to D_1, C_1'$ and in (D3.4) $D' = D_1$ and $C' = C_1' \mid C_2$.

Case $C = \mathtt{def}\ X = C_2\ \mathtt{in}\ C_1$
applies the induction hypothesis and Rule $\lfloor^{\mathsf{C}}_{\mathsf{Ctx}}\rceil$ for which $D, C_1 \to D', C_1'$, where $C' = \mathtt{def}\ X = C_2\ \mathtt{in}\ C_1'$.

Case $\mathtt{def}\ X = C_2\ \mathtt{in}\ X; C_1$
applies Rule $\lfloor^{\mathsf{C}}_{\mathsf{Eq}}\rceil$ for $\mathtt{def}\ X = C_2\ \mathtt{in}\ X; C_1 \equiv_{\mathsf{C}} \mathtt{def}\ X = C_2\ \mathtt{in}\ C_2; C_1$ and by the induction hypothesis $D, C_2 \to D', C_2'$ and $C' = \mathtt{def}\ X = C_2\ \mathtt{in}\ C_2'; C_1$.

Case $C = \texttt{if p}.e\ \{C_1\}\ \texttt{else}\ \{C_2\}$
from (D3.1) we know that $\Gamma \vdash \texttt{p}.e\colon \textbf{bool}$ and therefore we can apply Rule $[\![\mathsf{C}|_{\mathsf{Cond}}]\!]$ and, according to the evaluation of e, we have $C' = C_1$ or $C' = C_2$.

□

B.3. **Proof of Endpoint Projection.** To prove our result on the Endpoint Projection we first define the minimal typing system $\vdash_{\mathtt{min}}$ for FC.

B.3.1. *Minimal Typing.* We recall the definition of subtyping for local and global types (see Definitions 16 and 17), which we extend to set inclusion and point-wise to *i*) the typing of services (i.e., of kind $\tilde{l}\colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{C}}\rangle$) and *ii*) the typing of sessions, respectively. Given two types G and G', we denote their least upper bound wrt $\prec$ with $G \triangledown G'$ (the same for local types and typing environments).

We define the minimal typing system $\vdash_{\mathtt{min}}$ on this notion of subtyping. The minimal typing uses the minimal global and local types for typing sessions and services such that the projection of the choreography is still typable. We report the rules for minimal typing in Figure 27.

Proposition 1 (Existence of Minimal Typing)**.** Let $\Gamma \vdash D, C$, then there exists Γ_0 such that $\Gamma_0 \vdash D, C$ and for each $\Gamma' \vdash D, C$ we have that $\Gamma_0 \prec \Gamma'$. The environment Γ_0 can be algorithmically calculated from C and is called the minimal typing of C.

Proof of Existence of Minimal Typing. The proof is standard and proceeds by induction on the rules in Figure 27, defining the minimal typing system $\Gamma \vdash_{\mathtt{min}} D, C$.

As in [CM13, MY13], our focus is on the reconstruction of global/local types, thus we leave the reconstruction of variable types undefined (which it is entirely standard, e.g., see [Pie02]).

We give the intuition behind each case corresponding to the derivation on the rules. $\lfloor^{\mathsf{Min}}_{\mathsf{Start1}}\rceil$ and $\lfloor^{\mathsf{Min}}_{\mathsf{Start2}}\rceil$ type the starting of sessions. The difference between $\lfloor^{\mathsf{Min}}_{\mathsf{Start1}}\rceil$ and $\lfloor^{\mathsf{Min}}_{\mathsf{Start2}}\rceil$ is that, when $\lfloor^{\mathsf{Min}}_{\mathsf{Start1}}\rceil$ applies, the service typing of $\tilde{l}$ is not used any more in C, and thus its typing is dropped to guarantee minimality. Contrarily, in $\lfloor^{\mathsf{Min}}_{\mathsf{Start2}}\rceil$ the service typing of $\tilde{l}$ is used in the continuation C. In the rule, we consider the minimal global type $G \triangledown G'$ where G' is minimal in session k and G is minimal in the typing of the continuation C.

Rules $\lfloor^{\mathsf{Min}}_{\mathsf{Req1}}\rceil$ and $\lfloor^{\mathsf{Min}}_{\mathsf{Req2}}\rceil$ mirror a similar relationship, where in the first rule we drop the typing of $\tilde{l}$, not used in the continuation C, while in the second we consider $G \triangledown G'$. Note that Rule $\lfloor^{\mathsf{Min}}_{\mathsf{Acc}}\rceil$ directly drops the typing of $\tilde{l}$ in the typing of the continuation. We do this because we assumed (see § 2.1) that *i*) (*acc*) terms can only be at the top level (not guarded by other actions) and *ii*) by rule $\lceil^{\Gamma}_{\mathsf{Acc}}\rceil$ no subsequent term (*start*) on the same locations $\tilde{l}$ is typable (and hence cannot be present in C, well-typed). The same holds for subsequent (*req*) terms on $\tilde{l}$, which could not be paired with a complementary (*acc*).

In $\lfloor^{\mathsf{Min}}_{\mathsf{Cond}}\rceil$ we consider $\Gamma_1 \triangledown \Gamma_2$ to determine the least upper bound of receive types. Rules $\lfloor^{\mathsf{Min}}_{\mathsf{Com}}\rceil$, $\lfloor^{\mathsf{Min}}_{\mathsf{Send}}\rceil$, and $\lfloor^{\mathsf{Min}}_{\mathsf{Recv}}\rceil$ type receptions with a singleton branching local type. Rule $\lfloor^{\mathsf{Min}}_{\mathsf{Par}}\rceil$ is standard.

Also in rule $\lfloor^{\mathsf{Min}}_{\mathsf{Def}}\rceil$ we consider the least upper bound of Γ and Γ' respectively typing the continuation C and the body of procedure X. In addition, we also consider the least upper bound of the local typings T and T', on which we apply function solve. Function solve is standard (cf. [CHY12, CM13]) and solves the equations $\mathbf{t}_X = T$ for each T in $\widetilde{k[\mathtt{A}]\colon T}$ where, if $\mathbf{t}_X$ appears in T, the corresponding component is $\mathtt{rec}\ \mathbf{t}.T_X$, or T otherwise. Rule $\lfloor^{\mathsf{Min}}_{\mathsf{Def}}\rceil$ uses rules $\lfloor^{\mathsf{Min}}_{\mathsf{D1}}\rceil$ and $\lfloor^{\mathsf{Min}}_{\mathsf{D2}}\rceil$ to determine the content of Γ_x and Γ'_x to respectively minimally type the continuation C and the body of procedure X. Indeed, when rule $\lfloor^{\mathsf{Min}}_{\mathsf{D1}}\rceil$ applies, the choreography C uses the typing $X\colon \Gamma_x$, otherwise $\lfloor^{\mathsf{Min}}_{\mathsf{D2}}\rceil$ applies and the minimal type does not contain the typing for X. Finally, in case both the typing of C and of C' type X (i.e., X in $\mathbf{dom}(\Gamma_x) \cap \mathbf{dom}(\Gamma'_x)$), their judgements coincide.

$$\frac{\Gamma, \mathbf{init}(\ \widetilde{\mathtt{r}[\mathtt{C}]}, k, G\ ) \vdash_{\mathtt{min}} C \quad \widetilde{\mathtt{r}[\mathtt{C}]} = \mathtt{p}[\mathtt{A}], \widetilde{\mathtt{q}[\mathtt{B}]} \quad \tilde{\mathtt{q}} \notin \Gamma \quad \tilde{l} \notin \Gamma}{\Gamma, \tilde{l} \colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{B}}\rangle \vdash_{\mathtt{min}} \mathtt{start}\ k : \mathtt{p}[\mathtt{A}] \leftarrow\!\!\rightarrow \widetilde{l.\mathtt{q}[\mathtt{B}]}; C} \ \lfloor^{\mathsf{Min}}|_{\mathsf{Start1}}\rceil$$

$$\frac{\Gamma, \tilde{l} \colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{B}}\rangle, \mathbf{init}(\ \widetilde{\mathtt{r}[\mathtt{C}]}, k, G'\ ) \vdash_{\mathtt{min}} C \quad \widetilde{\mathtt{r}[\mathtt{C}]} = \mathtt{p}[\mathtt{A}], \widetilde{\mathtt{q}[\mathtt{B}]} \quad \tilde{\mathtt{q}} \notin \Gamma}{\Gamma, \tilde{l} \colon G \triangledown G'\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{B}}\rangle \vdash_{\mathtt{min}} \mathtt{start}\ k : \mathtt{p}[\mathtt{A}] \leftarrow\!\!\rightarrow \widetilde{l.\mathtt{q}[\mathtt{B}]}; C} \ \lfloor^{\mathsf{Min}}|_{\mathsf{Start2}}\rceil$$

$$\frac{\Gamma, \mathtt{p} : k[\mathtt{A}], k[\mathtt{A}] : [\![G]\!]_{\mathtt{A}} \vdash_{\mathtt{min}} C \quad \tilde{l} \notin \Gamma}{\Gamma, \tilde{l} \colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\varnothing\rangle \vdash_{\mathtt{min}} \mathtt{req}\ k : \mathtt{p}[\mathtt{A}] \leftarrow\!\!\rightarrow \widetilde{l.\mathtt{B}}; C} \ \lfloor^{\mathsf{Min}}|_{\mathsf{Req1}}\rceil \qquad \frac{\Gamma, \tilde{l} \colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\varnothing\rangle, \mathtt{p} : k[\mathtt{A}], k[\mathtt{A}] : [\![G']\!]_{\mathtt{A}} \vdash_{\mathtt{min}} C}{\Gamma, \tilde{l} \colon G \triangledown G'\langle \mathtt{A}|\tilde{\mathtt{B}}|\varnothing\rangle \vdash_{\mathtt{min}} \mathtt{req}\ k : \mathtt{p}[\mathtt{A}] \leftarrow\!\!\rightarrow \widetilde{l.\mathtt{B}}; C} \ \lfloor^{\mathsf{Min}}|_{\mathsf{Req2}}\rceil$$

$$\frac{\tilde{l} \subseteq \tilde{l}' \quad \Gamma, \mathbf{init}(\ \widetilde{\mathtt{q}[\mathtt{C}]}, k, G\ ) \vdash_{\mathtt{min}} C \quad \tilde{\mathtt{q}} \notin \Gamma \quad \tilde{l} \notin \Gamma}{\Gamma, \tilde{l}' \colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{C}}\rangle \vdash_{\mathtt{min}} \mathtt{acc}\ k : \widetilde{l.\mathtt{q}[\mathtt{C}]}; C} \ \lfloor^{\mathsf{Min}}|_{\mathsf{Acc}}\rceil$$

$$\frac{\Gamma_1 \triangledown \Gamma_2 \vdash \mathtt{p}.e \colon \mathbf{bool} \quad \Gamma_1 \vdash_{\mathtt{min}} C_1 \quad \Gamma_2 \vdash_{\mathtt{min}} C_2}{\Gamma_1 \triangledown \Gamma_2 \vdash_{\mathtt{min}} \mathtt{if}\ \mathtt{p}.e\ \{C_1\}\ \mathtt{else}\ \{C_2\}} \ \lfloor^{\mathsf{Min}}|_{\mathsf{Cond}}\rceil$$

$$\frac{\Gamma \vdash \mathtt{p} \colon k[\mathtt{A}], \mathtt{q} \colon k[\mathtt{B}] \quad \Gamma \vdash \mathtt{p}.e \colon U \quad \Gamma, \mathtt{q}.x \colon U, k[\mathtt{A}] \colon T, k[\mathtt{B}] \colon T' \vdash_{\mathtt{min}} C}{\Gamma, k[\mathtt{A}] \colon \oplus\, \mathtt{B}.\{o(U); T\}, k[\mathtt{B}] \colon \&\mathtt{A}.\{o(U); T'\} \vdash_{\mathtt{min}} k \colon \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{q}[\mathtt{B}].o(x); C} \ \lfloor^{\mathsf{Min}}|_{\mathsf{Com}}\rceil$$

$$\frac{\Gamma \vdash \mathtt{p} \colon k[\mathtt{A}] \quad \mathtt{q} \colon k[\mathtt{B}] \notin \Gamma \quad \Gamma \vdash \mathtt{p}.e \colon U \quad \Gamma, k[\mathtt{A}] \colon T \vdash_{\mathtt{min}} C}{\Gamma, k[\mathtt{A}] \colon \oplus\, \mathtt{B}.\{o(U); T\} \vdash_{\mathtt{min}} k \colon \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o; C} \ \lfloor^{\mathsf{Min}}|_{\mathsf{Send}}\rceil$$

$$\frac{\Gamma \vdash \mathtt{q} \colon k[\mathtt{B}] \quad \mathtt{p} \colon k[\mathtt{A}] \notin \Gamma \quad \Gamma, \mathtt{q}.x \colon U, k[\mathtt{B}] \colon T \vdash_{\mathtt{min}} C}{\Gamma, k[\mathtt{B}] \colon \&\mathtt{A}.\{o(U); T\} \vdash_{\mathtt{min}} k \colon \mathtt{A} \longrightarrow \mathtt{q}[\mathtt{B}].o(x); C} \ \lfloor^{\mathsf{Min}}|_{\mathsf{Recv}}\rceil$$

$$\frac{\begin{array}{c}\Gamma_x(X) = \Gamma'_x(X) \text{ if } X \in \mathbf{dom}(\Gamma_x) \cap \mathbf{dom}(\Gamma'_x) \qquad \nexists\ k''[\mathtt{A}''] \in \mathbf{dom}(\Gamma \triangledown \Gamma') \\ X \notin \mathbf{dom}(\Gamma \triangledown \Gamma') \quad \Gamma'_x \rhd_X (\Gamma', \widetilde{k'[\mathtt{A}'] : T'}), C' \qquad \Gamma_x \rhd_X (\Gamma, \widetilde{k[\mathtt{A}] : T}), C \qquad \Gamma'|_{\mathbf{locs}} \subseteq \Gamma\end{array}}{(\Gamma \triangledown \Gamma'), \mathsf{solve}(\widetilde{k[\mathtt{A}] : T} \triangledown \widetilde{k'[\mathtt{A}'] : T'}, \mathbf{t}_X) \vdash_{\mathtt{min}} \mathtt{def}\ X = C'\ \mathtt{in}\ C} \ \lfloor^{\mathsf{Min}}|_{\mathsf{Def}}\rceil$$

$$\frac{\Gamma_1 \vdash_{\mathtt{min}} C_1 \quad \Gamma_2 \vdash_{\mathtt{min}} C_2}{\Gamma_1, \Gamma_2 \vdash_{\mathtt{min}} C_1 \mid C_2} \ \lfloor^{\mathsf{Min}}|_{\mathsf{Par}}\rceil \qquad \frac{\Gamma, \Gamma_x, X \colon \Gamma_x \vdash_{\mathtt{min}} C}{\Gamma_x, X \colon \Gamma_x \rhd_X \Gamma, C} \ \lfloor^{\mathsf{Min}}|_{\mathsf{D1}}\rceil \qquad \frac{X \notin \mathbf{dom}(\Gamma_x) \quad \Gamma, \Gamma_x \vdash_{\mathtt{min}} C}{\Gamma_x \rhd_X \Gamma, C} \ \lfloor^{\mathsf{Min}}|_{\mathsf{D2}}\rceil$$

$$\frac{\Gamma = \mathsf{ownerships} \cup \mathsf{sessions} \cup \mathsf{vars} \quad k[\mathtt{A}] \in \mathsf{sessions} \quad k[\mathtt{A}] : \mathtt{end}}{\Gamma \vdash_{\mathtt{min}} \mathbf{0}} \ \lfloor^{\mathsf{Min}}|_{\mathsf{End}}\rceil$$

$$\frac{\begin{array}{c}\Gamma = \mathsf{vars} \cup \mathsf{ownerships} \qquad \widetilde{k'[\mathtt{A}']} = \mathsf{sessions} \setminus \{\widetilde{k[\mathtt{A}]}\} \\ \Gamma' = \mathsf{vars}(X) \cup \mathsf{ownerships}(X) \quad \widetilde{k[\mathtt{A}]} = \mathsf{sessions}(\mathsf{X}) \qquad \Gamma' \subseteq \Gamma\end{array}}{\Gamma, \widetilde{k[\mathtt{A}]} : \mathbf{t}_X, \widetilde{k'[\mathtt{A}']} : \mathsf{end}, X \colon (\Gamma', \widetilde{k[\mathtt{A}]} : \mathbf{t}_X) \vdash_{\mathtt{min}} X} \ \lfloor^{\mathsf{Min}}|_{\mathsf{Call}}\rceil$$

$$\frac{\mathbf{pco}(\Gamma, \Gamma') \quad \Gamma \vdash D \quad \Gamma' \vdash_{\mathtt{min}} C}{\Gamma, \Gamma' \vdash_{\mathtt{min}} D, C} \ \lfloor^{\mathsf{Min}}|_{\mathsf{DC}}\rceil$$

Figure 27: Frontend Choreographies — Minimal typing rules

Rules $\lfloor^{\mathsf{Min}}_{\mathsf{End}}\rceil$ and $\lfloor^{\mathsf{Min}}_{\mathsf{Call}}\rceil$ use some auxiliary information, obtainable by a preliminary top-down visit of the choreography syntax tree (cf. [CHY12, CM13]). Specifically, vars, $\mathsf{ownerships}$, and $\mathsf{sessions}$ are respectively the variable, the ownership, and the session typings of the choreography whose type is being inferred. Similarly, $\mathsf{vars}(X)$, $\mathsf{ownerships}(X)$, and $\mathsf{sessions}(X)$ yield respectively the same kind of information regarding the body of procedure X (i.e., obtained inspecting the body of the inner-most recursive procedure X). In the rules, in $\lfloor^{\mathsf{Min}}_{\mathsf{End}}\rceil$ we check that in Γ reside only those ownership, variable, and session typings present in the typed choreography and that all sessions (i.e., their local types) are terminated. In rule $\lfloor^{\mathsf{Min}}_{\mathsf{Call}}\rceil$, i) all sessions outside X must be terminated and those inside X agree on $\mathbf{t}_X$ and ii) Γ and Γ' contain only appropriate variable and ownership typings and agree on their judgement ($\Gamma' \subseteq \Gamma$).

Rule $\lfloor^{\mathsf{Min}}_{\mathsf{DC}}\rceil$ defines minimal typing for running choreographies.

□

B.3.2. *Typing Projection.* Here we define the projection of typing environments, which is used to prove that, given the minimal typing environment Γ of a choreography C, from Γ we can build the minimal typing environment for the EPP of C.

To do that, we have to account for two peculiarities (as defined in § 6.2) of our EPP:

- it merges in the output choreography the behaviours of many service processes into one process. Hence, to guarantee typing and minimality we have to merge typings related to service processes on the same location into the same (and only) service process present in $[\![C]\!]$;
- it projects recursive definitions of the same procedure on different processes, e.g., if in C there are processes $\mathsf{p}_1, \ldots, \mathsf{p}_n$ and procedure X, in the EPP we will find procedures $X_{\mathsf{p}_1}, \ldots, X_{\mathsf{p}_n}$. Thus, we replace the definition typing of any procedure X in $\mathbf{dom}(\Gamma)$ with the typings of its projections X_{p_1}, …, X_{p_n}.

To indicate the projection of a typing environment Γ wrt to its typed choreography C, we write $[\![\Gamma]\!]^C$. To define $[\![\Gamma]\!]^C$ (and also later in this proof) we use the typing environment filtering operator $\Gamma|_{\mathsf{p}}$ defined as

$$\Gamma|_{\mathsf{p}} = \begin{cases} \{\ \mathsf{p}.x : U \mid \mathsf{p}.x : U \in \Gamma\ \} & \cup \\ \{\ \mathsf{p} : k[\mathtt{A}], k[\mathtt{A}] : T\} \mid \{\ \mathsf{p} : k[\mathtt{A}], k[\mathtt{A}] : T\ \} \subseteq \Gamma\ \} & \cup \\ \{\ \tilde{l} : G\langle \mathtt{A} | \tilde{\mathtt{B}} | \tilde{\mathtt{C}} \rangle \mid \tilde{l} : G\langle \mathtt{A} | \tilde{\mathtt{B}} | \tilde{\mathtt{C}} \rangle \in \Gamma\ \} & \cup \\ \{\ X_{\mathsf{p}} : \Gamma_x \mid X_{\mathsf{p}} : \Gamma_x \in \Gamma\ \} \end{cases}$$

Definition 19 (Typing Projection)**.** Let $\Gamma \vdash C$, the projection of Γ wrt to C, written $[\![\Gamma]\!]^C$, is defined as:

$$[\![\Gamma]\!]^C = \underbrace{\left\{ \bigcup_{\mathsf{q} \in \lfloor C \rfloor_l} [\![\Gamma]\!]_{\mathsf{q}} \underbrace{[\mathsf{p}/\mathsf{q}]}_{i.i)} \mid \underbrace{\mathsf{p} \in \lfloor C \rfloor_l \cap \mathbf{pn}([\![C]\!])}_{i.ii)} \wedge l \in \{\tilde{l}\} \wedge \tilde{l} \in \mathbf{dom}(\Gamma) \right\}}_{i)}, \underbrace{\{\ [\![\Gamma]\!]_{\mathsf{r}} \mid \mathsf{r} \in \mathbf{fp}([\![C]\!])\}}_{ii)}$$

$$[\![\Gamma]\!]_{\mathsf{p}} = \underbrace{\left(\Gamma|_{\mathsf{p}} \setminus \{X : \Gamma_x \mid \Gamma \vdash X : \Gamma_x\}\right)}_{iii)}, \underbrace{\left\{X_{\mathsf{p}} : [\![\Gamma_x]\!]_{\mathsf{p}} \mid \Gamma \vdash X : \Gamma_x\right\}}_{iv)}$$

As mentioned above, in the definition of $[\![\Gamma]\!]^C$ we distinguish two kinds of projections: the one on service processes i) and the one on active processes ii). In the first case, we unify

the projection on service processes at the same location in C (i.e., in $\lfloor C \rfloor_l$). To do that in a consistent way, wrt to the EPP of C we:

- obtain the identifier of process p *i.i*), the only service process at location l that is present in $[\![C]\!]$ (and hence the one that merges the behaviours of all service processes in C at l);
- get the projection of Γ on a service process q ($[\![G]\!]_{\mathsf{q}}$) in $\lfloor C \rfloor_l$;
- we rename all process-related typings in $[\![\Gamma]\!]_{\mathsf{q}}$ to correspond to process p (by abusing the notation $[\![\Gamma]\!]_{\mathsf{q}}[\mathsf{p}/\mathsf{q}]$) *i.ii*);
- we merge all the resulting, renamed typing environments into a single typing environment for process p.

Finally, the projection of typing environment Γ on process p, written $[\![\Gamma]\!]_{\mathsf{p}}$ corresponds to the union of *iii*) the typing in Γ related to process p, from which we remove the typings of definitions, and *iv*) the projection of the typings of definitions, renamed for process p.

Note the definition of $[\![\Gamma]\!]^C$ is coherent with the definition of process projection (see Definition 9) in which the rule for projecting (*rec*) terms is defined as:

$$[\![\mathtt{def}\ X = C'\ \mathtt{in}\ C]\!]_{\mathsf{r}} \quad = \quad \mathtt{def}\ X_{\mathsf{r}} = [\![\ C'[X_{\mathsf{r}}/X]\ ]\!]_{\mathsf{r}}\ \mathtt{in}\ [\![\ C[X_{\mathsf{r}}/X]\ ]\!]_{\mathsf{r}}$$

Similarly, $[\![\Gamma]\!]^C$ generates definition typings for each procedure corresponding to each process in the choreography (assumed to be C). The typings of definitions are guaranteed minimal (as required in Theorem 8).

The only remark regards service typings, which are present in all projected environments, although they might not be used. While having additional, unused service typings does not compromise type checking, we must consider a weakened form of minimality of typing where some unused service typings are allowed. This fact is clearly stated in the definition of the Theorem 8.

B.4. **Proof of the Well-Typedness property of Theorem 5.** To prove the property of well-typedness of Theorem 5 we prove the stronger result of Theorem 8.

Theorem 8 (EPP Typing Preservation)**.** Let D, C be a well-typed running choreography such that $\Gamma \vdash_{\mathtt{min}} D, C$, where $\Gamma = \Gamma_d, \Gamma_c$ such that $\Gamma_d \vdash D$, then $[\![\Gamma_c]\!]^C, \Gamma_d \vdash_{\mathtt{min}} D, [\![C]\!]$ up to service typings.

Intuitively, Theorem 8 subsumes the well-typedness property (1) of Theorem 5, using the environment projection defined above to provide a minimal typing environment for $[\![C]\!]$ up to some unused service typings.

We define some auxiliary lemmas used in the proof of Theorem 8.

Lemma 7 (Composability of Typing Projections)**.** Let $\Gamma \vdash C$ and $\Gamma = \Gamma', \Gamma''$ then $[\![\Gamma]\!]^C = [\![\Gamma']\!]^C, [\![\Gamma'']\!]^C$.

Proof . The proof is by contradiction. The projection $[\![\Gamma]\!]^C$ returns exactly Γ except for the projection of the typings of the procedures, as defined in Definition 19. Hence the projection $[\![\Gamma]\!]^C$ can differ from $[\![\Gamma']\!]^C, [\![\Gamma'']\!]^C$ only on definition typings. However, it is impossible that $[\![\Gamma]\!]^C \neq [\![\Gamma']\!]^C, [\![\Gamma'']\!]^C$. Indeed, there could be only two cases for the partitioning of Γ wrt any definition typing $X \in \mathbf{dom}(\Gamma)$, either:

- *i*) both Γ' and Γ'' type X, in which case, since $\Gamma = \Gamma', \Gamma''$, they must agree on their judgement on X;

- *ii*) the judgement on X is contained only in Γ' or Γ''.

in both cases the projections obtained from X remain the same wrt the one in Γ. □

We prove Lemma 8 that states that given a well-typed choreography C and a typing environment Γ for which $\Gamma \vdash_{\texttt{min}} C$ then the projection of Γ, $[\![\Gamma]\!]^C$, types minimally the projection of C, $[\![C]\!]$.

Lemma 8 (Choreography EPP Typing Preservation)**.** Let C be a well-typed choreography and let $\Gamma \vdash_{\texttt{min}} C$ then $[\![\Gamma]\!]^C \vdash_{\texttt{min}} [\![C]\!]$.

Proof . Like for the proof of Theorem 3, we assume our choreographies to be well-sorted. The proof is by induction on the typing derivation of $\Gamma \vdash_{\texttt{min}} C$.

Case $\lfloor^{\mathsf{Min}}|_{\mathsf{Start1}}\rceil$

From the premises we have $C = \texttt{start}\ k : \mathtt{p}[\mathtt{A}] \leftrightarrow \widetilde{l.\mathtt{q}[\mathtt{B}]}; C'$. We can partition $\Gamma = \tilde{l}\colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{B}}\rangle, \Gamma'$ and we can write the derivation

$$\frac{\Gamma', \mathbf{init}(\widetilde{\mathtt{r}[\mathtt{C}]}, k, G) \vdash_{\texttt{min}} C' \quad \widetilde{\mathtt{r}[\mathtt{C}]} = \mathtt{p}[\mathtt{A}], \widetilde{\mathtt{q}[\mathtt{B}]} \quad \tilde{\mathtt{q}} \notin \Gamma' \quad \tilde{l} \notin \Gamma'}{\Gamma', \tilde{l}\colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{B}}\rangle \vdash_{\texttt{min}} \texttt{start}\ k : \mathtt{p}[\mathtt{A}] \leftrightarrow \widetilde{l.\mathtt{q}[\mathtt{B}]}; C'}\ \lfloor^{\mathsf{Min}}|_{\mathsf{Start1}}\rceil$$

Let $\widetilde{l.\mathtt{q}[\mathtt{B}]} = l_1.\mathtt{q}_1[\mathtt{B}_1], \cdots, l_n.\mathtt{q}_n[\mathtt{B}_n]$.

Let $\Gamma_c = \Gamma', \mathbf{init}(\widetilde{\mathtt{r}[\mathtt{C}]}, k, G)$, from the induction hypothesis we have that $\Gamma_c \vdash_{\texttt{min}} C'$ and therefore $[\![\Gamma_c]\!]^{C'} \vdash_{\texttt{min}} [\![C']\!]$.

By its definition $[\![C']\!] \equiv_{\mathtt{C}} C'_s \mid C''$ where

$$C'_s = [\![C']\!]_{\mathtt{p}} \mid [\![C']\!]_{\mathtt{q}_1} \mid \ldots \mid [\![C']\!]_{\mathtt{q}_n}$$

and

$$C'' = \prod_{\mathtt{r} \in \mathbf{fp}(C')\setminus\{\mathtt{p},\tilde{\mathtt{q}}\}} [\![C']\!]_{\mathtt{r}} \quad \mid \quad \prod_{l} \left(\bigsqcup_{\mathtt{s} \in \lfloor C' \rfloor_l} [\![C']\!]_{\mathtt{s}} \right)$$

We partition $[\![\Gamma_c]\!]^{C'}$ (as per Lemma 7) as

$$[\![\Gamma_c]\!]^{C'} = \Gamma'_{\mathtt{p}}\ ,\ \Gamma'_{\tilde{\mathtt{q}}}\ ,\ \Gamma''$$

where

$$\Gamma'_{\mathtt{p}} = \Gamma''_{\mathtt{p}}, \mathtt{p}\colon k[\mathtt{A}], k[\mathtt{A}]\colon [\![G]\!]_{\mathtt{p}}$$

and

$$\Gamma'_{\tilde{\mathtt{q}}} = \Gamma'_{\mathtt{q}_1}\ ,\ \cdots\ ,\ \Gamma'_{\mathtt{q}_n}$$

where

$$\Gamma'_{\mathtt{q}_i} = \Gamma''_{\mathtt{q}_i}, \mathtt{q}_i\colon k[\mathtt{A}], k[\mathtt{A}]\colon [\![G]\!]_{\mathtt{q}_i}$$

such that we can write the derivation

$$\dfrac{\Gamma'' \vdash_{\texttt{min}} C'' \qquad \dfrac{\Gamma'_{\mathtt{p}} \vdash_{\texttt{min}} [\![C']\!]_{\mathtt{p}} \qquad \dfrac{\Gamma'_{\mathtt{q}_1} \vdash_{\texttt{min}} [\![C']\!]_{\mathtt{q}_1} \qquad \dfrac{\vdots}{\Gamma'_{\mathtt{q}_2}, \ldots, \Gamma'_{\mathtt{q}_n} \vdash_{\texttt{min}} [\![C']\!]_{\mathtt{q}_2} \mid \ldots \mid [\![C']\!]_{\mathtt{q}_n}}\ \lfloor^{\mathsf{Min}}|_{\mathsf{Par}}\rceil}{\Gamma'_{\mathtt{q}_1}, \Gamma'_{\mathtt{q}_2}, \ldots, \Gamma'_{\mathtt{q}_n} \vdash_{\texttt{min}} [\![C']\!]_{\mathtt{q}_1} \mid \ldots \mid [\![C']\!]_{\mathtt{q}_n}}\ \lfloor^{\mathsf{Min}}|_{\mathsf{Par}}\rceil}{\Gamma'_{\mathtt{p}}, \Gamma'_{\tilde{\mathtt{q}}} \vdash_{\texttt{min}} [\![C']\!]_{\mathtt{p}} \mid [\![C']\!]_{\mathtt{q}_1} \mid \ldots \mid [\![C']\!]_{\mathtt{q}_n}}\ \lfloor^{\mathsf{Min}}|_{\mathsf{Par}}\rceil}{\Gamma'', \Gamma'_{\mathtt{p}}, \Gamma'_{\tilde{\mathtt{q}}} \vdash_{\texttt{min}} C'' \mid C'_s}\ \lfloor^{\mathsf{Min}}|_{\mathsf{Par}}\rceil$$

Since the ownership and session typings for k in Γ_c belong to $\mathbf{init}(\widetilde{\mathtt{r}[\mathtt{C}]}, k, G)$ we can write $\Gamma'_{\mathtt{p}} = \Gamma''_{\mathtt{p}}, \mathtt{p}\colon k[\mathtt{A}], k[\mathtt{A}]\colon T$ where $\Gamma''_{\mathtt{p}}$ contains those and only typings (services, ownerships, sessions, etc.) that type minimally the projection of continuation C' for process $\mathtt{p}$.

Since the only difference between Γ and Γ_c are the typings for session k, we have that $\Gamma''_{\mathtt{p}} \subseteq [\![\Gamma]\!]^C$ and also $\Gamma'' \subseteq [\![\Gamma]\!]^C$. The same argument holds for typings $\Gamma'_{\mathtt{q}_i}$. Indeed, we can partition $[\![\Gamma]\!]^C = \Gamma'', \Gamma''_{\mathtt{p}}, \Gamma''_{\mathtt{q}_1}, \ldots, \Gamma''_{\mathtt{q}_n}, \tilde{l}\colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{B}}\rangle$ (as of Lemma 7).

Finally, by the definition of inclusion of service typings in Γ (cf § 3.2.1), we can write judgement $\tilde{l}\colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{B}}\rangle$ as the sequence of judgements $\tilde{l}\colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\varnothing\rangle, \tilde{l}\colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\mathtt{B}_1\rangle, \ldots, \tilde{l}\colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\mathtt{B}_n\rangle$.

Therefore we write $[\![\Gamma']\!]^C$ as

$$[\![\Gamma']\!]_C = \Gamma'', \Gamma''_{\mathtt{p}}, \Gamma''_{\mathtt{q}_1}, \ldots, \Gamma''_{\mathtt{q}_n}, \tilde{l}\colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\varnothing\rangle, \tilde{l}\colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\mathtt{B}_1\rangle, \ldots, \tilde{l}\colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\mathtt{B}_n\rangle$$

Let $\widetilde{l.\mathtt{q}[\mathtt{B}]}\big|_i = \{l_i.\mathtt{q}_i[\mathtt{B}_i], \ldots, l_n.\mathtt{q}_n[\mathtt{B}_n]\}$, we prove the case by proving the typing derivation for $[\![\Gamma]\!]^C \vdash_{\mathtt{min}} [\![C]\!]$.

From the definition of EPP (Definition 10) we can write

$$[\![C]\!] \equiv C_s \mid C''$$

where, given the shape of C, we know that C'' is the same as the one generated from $[\![C']\!]$, as seen above. C_s is

$$C_s = \mathtt{req}\ k : \mathtt{p}[\mathtt{A}] \leftrightarrows \widetilde{l.\mathtt{B}};\ [\![C']\!]_{\mathtt{p}} \quad \mid \prod_{l.\mathtt{r}[\mathtt{C}]\ \in\ \{\widetilde{l.\mathtt{q}[\mathtt{B}]}\}} \mathtt{acc}\ k : l.\mathtt{r}[\mathtt{C}];\ [\![C']\!]_{\mathtt{r}}$$

We now prove we can derive the typing of $[\![\Gamma]\!]^C \vdash_{\mathtt{min}} [\![C]\!]$

$$\dfrac{\Gamma'' \vdash_{\mathtt{min}} C'' \qquad \dfrac{\dfrac{\Gamma''_{\mathtt{p}}, \mathtt{p}\colon k[\mathtt{A}], k[\mathtt{A}]\colon [\![G]\!]_{\mathtt{A}} \vdash_{\mathtt{min}} [\![C']\!]_{\mathtt{p}} \quad \tilde{l} \notin \Gamma''_{\mathtt{p}}}{\Gamma''_{\mathtt{p}}, \tilde{l}\colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\varnothing\rangle \vdash_{\mathtt{min}} \mathtt{req}\ k : \mathtt{p}[\mathtt{A}] \leftrightarrows \widetilde{l.\mathtt{B}};\ [\![C']\!]_{\mathtt{p}}}\ \lfloor^{\mathsf{Min}}|_{\mathsf{Req1}}\rceil \qquad \Delta_1}{\Gamma''_{\mathtt{p}}, \tilde{l}\colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\varnothing\rangle, \Gamma''_{\mathtt{q}_1}, \tilde{l}\colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\mathtt{B}_1\rangle, \ldots, \Gamma''_{\mathtt{q}_n}, \tilde{l}\colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\mathtt{B}_n\rangle \vdash_{\mathtt{min}} C_s}\ \lfloor^{\mathsf{Min}}|_{\mathsf{Par}}\rceil}{\Gamma'', \Gamma''_{\mathtt{p}}, \tilde{l}\colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\varnothing\rangle, \Gamma''_{\mathtt{q}_1}, \tilde{l}\colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\mathtt{B}_1\rangle, \ldots, \Gamma''_{\mathtt{q}_n}, \tilde{l}\colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\mathtt{B}_n\rangle \vdash_{\mathtt{min}} C_s \mid C''}\ \lfloor^{\mathsf{Min}}|_{\mathsf{Par}}\rceil$$

where

$$\Delta_i = \dfrac{\Delta_{i+1} \qquad \dfrac{l_i \subseteq \tilde{l} \qquad \Gamma''_{\mathtt{q}_i}, \mathbf{init}(\mathtt{q}_i[\mathtt{B}_i], k, G) \vdash_{\mathtt{min}} [\![C']\!]_{\mathtt{q}_i} \qquad \mathtt{q}_i \notin \Gamma''_{\mathtt{q}_i} \qquad \tilde{l} \notin \Gamma''_{\mathtt{q}_i}}{\Gamma''_{\mathtt{q}_i}, \tilde{l}\colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\mathtt{B}_i\rangle \vdash_{\mathtt{min}} \mathtt{acc}\ k : l_i.\mathtt{q}_i[\mathtt{B}_i];\ [\![C']\!]_{\mathtt{q}_i}}\ \lfloor^{\mathsf{Min}}|_{\mathsf{Acc}}\rceil}{\Gamma''_{\mathtt{q}_i}, \tilde{l}\colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\mathtt{B}_i\rangle, \ldots, \Gamma''_{\mathtt{q}_n}, \tilde{l}\colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\mathtt{B}_n\rangle \vdash_{\mathtt{min}} \mathtt{acc}\ k : l_i.\mathtt{q}_i[\mathtt{B}_i];\ [\![C']\!]_{\mathtt{q}_i} \mid \prod_{l.\mathtt{r}[\mathtt{C}]\ \in\ \widetilde{l.\mathtt{q}[\mathtt{B}]}\big|_{i+1}} \mathtt{acc}\ k : l.\mathtt{r}[\mathtt{C}];\ [\![C']\!]_{\mathtt{r}}}\ \lfloor^{\mathsf{Min}}|_{\mathsf{Par}}\rceil$$

Note that we are reporting only the derivation terminating with $\lfloor^{\mathsf{Min}}|_{\mathsf{Req1}}\rceil$, i.e., the one that applies when $\Gamma''_{\mathtt{p}}$ does not contain the typing of $\tilde{l}$. The other case is similar and it applies rule $\lfloor^{\mathsf{Min}}|_{\mathsf{Req2}}\rceil$.

– $\Gamma'' \vdash_{\mathtt{min}} C''$;
– $\Gamma''_{\mathtt{p}}, \mathtt{p}\colon k[\mathtt{A}], k[\mathtt{A}]\colon [\![G]\!]_{\mathtt{A}} \vdash_{\mathtt{min}} [\![C']\!]_{\mathtt{p}}$;
– $\Gamma''_{\mathtt{q}_i}, \mathbf{init}(\mathtt{q}_i[\mathtt{B}_i], k, G) \vdash_{\mathtt{min}} [\![C']\!]_{\mathtt{q}_i}$.

hold by the induction hypothesis.

Case $\lfloor^{\mathsf{Min}}|_{\mathsf{Start2}}\rceil$

Similar to case $\lfloor^{\mathsf{Min}}|_{\mathsf{Start1}}\rceil$.

Case $\lfloor^{\mathsf{Min}}|_{\mathsf{Req1}}\rceil$
and ***Case.*** $\lfloor^{\mathsf{Min}}|_{\mathsf{Req2}}\rceil$ follow the proof of case $\lfloor^{\mathsf{Min}}|_{\mathsf{Start1}}\rceil$, focussing on the request branch.
Case $\lfloor^{\mathsf{Min}}|_{\mathsf{Acc}}\rceil$
Follows the proof of case $\lfloor^{\mathsf{Min}}|_{\mathsf{Start1}}\rceil$, following the accept branch.
Case $\lfloor^{\mathsf{Min}}|_{\mathsf{Cond}}\rceil$
By induction hypothesis on C_1 or C_2.
Case $\lfloor^{\mathsf{Min}}|_{\mathsf{Com}}\rceil$
From the premises we have $C = k\colon \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{q}[\mathtt{B}].o(x); C'$ on which we can apply the typing derivation

$$\frac{\Gamma' \vdash \mathtt{p}\colon k[\mathtt{A}], \mathtt{q}\colon k[\mathtt{B}] \quad \Gamma' \vdash \mathtt{p}.e\colon U \quad \Gamma', \mathtt{q}.x\colon U, k[\mathtt{A}]\colon T, k[\mathtt{B}]\colon T' \vdash_{\mathtt{min}} C'}{\Gamma', k[\mathtt{A}]\colon \oplus \mathtt{B}.o(U); T, k[\mathtt{B}]\colon \&\mathtt{A}.o(U); T' \vdash_{\mathtt{min}} k\colon \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{q}[\mathtt{B}].o(x); C'} \lfloor^{\mathsf{Min}}|_{\mathsf{Com}}\rceil$$

Hence we consider $\Gamma = \Gamma', k[\mathtt{A}]\colon \oplus \mathtt{B}.o(U); T, k[\mathtt{B}]\colon \&\mathtt{A}.o(U); T'$. From the definition of EPP (Definition 10) we have $[\![C]\!] \equiv C_c \mid C''$ where

$$C_c = k\colon \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o; [\![C']\!]_{\mathtt{p}} \quad \mid \quad k\colon \mathtt{A} \longrightarrow \mathtt{q}[\mathtt{B}].o(x); [\![C']\!]_{\mathtt{q}}$$

$$C'' = \prod_{\mathtt{r} \,\in\, \{\mathbf{fp}(C')\setminus\{\mathtt{p},\mathtt{q}\}\}} [\![C']\!]_{\mathtt{r}} \quad \mid \quad \prod_{l} \left(\bigsqcup_{\mathtt{s} \,\in\, \lfloor C' \rfloor_l} [\![C']\!]_{\mathtt{s}} \right)$$

From the definition of $[\![\Gamma]\!]^C$ we can write

$$[\![\Gamma]\!]^C = [\![\Gamma']\!]^C, k[\mathtt{A}]\colon \oplus \mathtt{B}.o(U); T, k[\mathtt{A}]\colon \&\mathtt{A}.o(U); T'$$

from the induction hypothesis we have that, let $\Gamma_c = \Gamma', \mathtt{q}.x\colon U, k[\mathtt{A}]\colon T, k[\mathtt{B}]\colon T'$, $\Gamma_c \vdash_{\mathtt{min}} C'$ and therefore $[\![\Gamma_c]\!]^{C'} \vdash_{\mathtt{min}} [\![C']\!]$. We can partition $[\![\Gamma_c]\!]^{C'}$ as

$$[\![\Gamma_c]\!]^{C'} = \Gamma'', \Gamma_{\mathtt{p}}, k[\mathtt{A}]\colon T, \Gamma_{\mathtt{q}}, \mathtt{q}.x\colon U, k[\mathtt{B}]\colon T'$$

such that

$$\frac{\Gamma'' \vdash_{\mathtt{min}} C'' \qquad \dfrac{\Gamma_{\mathtt{p}}, k[\mathtt{A}]\colon T \vdash_{\mathtt{min}} [\![C']\!]_{\mathtt{p}} \quad \Gamma_{\mathtt{q}}, \mathtt{q}.x\colon U, k[\mathtt{B}]\colon T' \vdash_{\mathtt{min}} [\![C']\!]_{\mathtt{q}}}{\Gamma_{\mathtt{p}}, k[\mathtt{A}]\colon T, \Gamma_{\mathtt{q}}, \mathtt{q}.x\colon U, k[\mathtt{B}]\colon T' \vdash_{\mathtt{min}} [\![C']\!]_{\mathtt{p}} \mid [\![C']\!]_{\mathtt{q}}} \lfloor^{\mathsf{Min}}|_{\mathsf{Par}}\rceil}{\Gamma'', \Gamma_{\mathtt{p}}, k[\mathtt{A}]\colon T, \Gamma_{\mathtt{q}}, \mathtt{q}.x\colon U, k[\mathtt{B}]\colon T' \vdash_{\mathtt{min}} [\![C']\!]_{\mathtt{p}} \mid [\![C']\!]_{\mathtt{q}} \mid C''} \lfloor^{\mathsf{Min}}|_{\mathsf{Par}}\rceil$$

From the derivation on rule $\lfloor^{\mathsf{Min}}|_{\mathsf{Com}}\rceil$ we know that

$$[\![\Gamma']\!]^{C'} = \Gamma'', \Gamma_{\mathtt{p}}, \Gamma_{\mathtt{q}}$$

and therefore that

$$[\![\Gamma]\!]^C = \Gamma'', \Gamma_{\mathtt{p}}, k[\mathtt{A}]\colon \oplus \mathtt{B}.o(U); T, \Gamma_{\mathtt{q}}, k[\mathtt{B}]\colon \oplus \mathtt{A}.o(U); T'$$

To prove $[\![\Gamma]\!]^C \vdash_{\mathtt{min}} [\![C]\!]$ we prove that we can apply rule $\lfloor^{\mathsf{Min}}|_{\mathsf{Par}}\rceil$.

$$\dfrac{\dfrac{\dfrac{\Gamma_{\mathtt{p}} \vdash \mathtt{p}\colon k[\mathtt{A}] \quad \mathtt{q}\colon k[\mathtt{B}] \notin \Gamma_{\mathtt{p}} \quad \Gamma_{\mathtt{p}} \vdash \mathtt{p}.e\colon U \quad \Gamma_{\mathtt{p}}, k[\mathtt{A}]\colon T \vdash_{\mathtt{min}} [\![C']\!]_{\mathtt{p}}}{\Gamma_{\mathtt{p}}, k[\mathtt{A}]\colon \oplus\, \mathtt{B}.o(U); T \vdash_{\mathtt{min}} k\colon \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o; [\![C']\!]_{\mathtt{p}}}\ \lfloor^{\mathsf{Min}}|_{\mathsf{Send}}\rceil \quad \dfrac{\Gamma_{\mathtt{q}} \vdash \mathtt{p}\colon k[\mathtt{B}] \qquad \mathtt{p}\colon k[\mathtt{A}] \notin \Gamma_{\mathtt{q}} \quad \Gamma_{\mathtt{q}}, \mathtt{q}.x\colon U, k[\mathtt{B}]\colon T' \vdash_{\mathtt{min}} [\![C']\!]_{\mathtt{q}}}{\Gamma_{\mathtt{p}}, k[\mathtt{B}]\colon \&\mathtt{A}.o(U); T' \vdash_{\mathtt{min}} k\colon \mathtt{A} \longrightarrow \mathtt{q}[\mathtt{B}].o(x); [\![C']\!]_{\mathtt{p}}}\ \lfloor^{\mathsf{Min}}|_{\mathsf{Recv}}\rceil}{\Gamma_{\mathtt{p}}, k[\mathtt{A}]\colon \oplus\, \mathtt{B}.o(U); T, \Gamma_{\mathtt{q}}, k[\mathtt{B}]\colon \oplus\, \mathtt{A}.o(U); T' \vdash_{\mathtt{min}} k\colon \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o; [\![C']\!]_{\mathtt{p}} \mid k\colon \mathtt{A} \longrightarrow \mathtt{q}[\mathtt{B}].o(x); [\![C']\!]_{\mathtt{q}}}\ \lfloor^{\mathsf{Min}}|_{\mathsf{Par}}\rceil \qquad \Gamma'' \vdash_{\mathtt{min}} C''}{\Gamma'', \Gamma_{\mathtt{p}}, k[\mathtt{A}]\colon \oplus\, \mathtt{B}.o(U); T, \Gamma_{\mathtt{q}}, k[\mathtt{B}]\colon \oplus\, \mathtt{A}.o(U); T' \vdash_{\mathtt{min}} k\colon \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o; [\![C']\!]_{\mathtt{p}} \mid k\colon \mathtt{A} \longrightarrow \mathtt{q}[\mathtt{B}].o(x); [\![C']\!]_{\mathtt{q}} \mid C''}\ \lfloor^{\mathsf{Min}}|_{\mathsf{Par}}\rceil$$

Case $\lfloor^{\mathsf{Min}}|_{\mathsf{Send}}\rceil$
Analogous to case $\lfloor^{\mathsf{Min}}|_{\mathsf{Com}}\rceil$
Case $\lfloor^{\mathsf{Min}}|_{\mathsf{Recv}}\rceil$
Analogous to case $\lfloor^{\mathsf{Min}}|_{\mathsf{Com}}\rceil$.
Case $\lfloor^{\mathsf{Min}}|_{\mathsf{Par}}\rceil$
From the premises we know that $C = C_1 \mid C_2$ on which we can apply the typing derivation

$$\dfrac{\Gamma_1 \vdash_{\mathtt{min}} C_1 \qquad \Gamma_2 \vdash_{\mathtt{min}} C_2}{\Gamma_1, \Gamma_2 \vdash_{\mathtt{min}} C_1 \mid C_2}\ \lfloor^{\mathsf{Min}}|_{\mathsf{Par}}\rceil$$

the case is proved applying the induction hypothesis.
Case $\lfloor^{\mathsf{Min}}|_{\mathsf{Def}}\rceil$
From the premises we know that $C = \mathtt{def}\ X = C''\ \mathtt{in}\ C'$ on which we can apply the typing derivation, with $\Gamma = (\Gamma' \triangledown \Gamma''), \mathsf{solve}(\widetilde{k[\mathtt{A}] : T} \triangledown \widetilde{k'[\mathtt{A}'] : T'}, \mathbf{t}_X)$

$$\dfrac{\Gamma_x(X) = \Gamma'_x(X) \text{ if } X \in \mathbf{dom}(\Gamma_x) \cap \mathbf{dom}(\Gamma'_x) \qquad \nexists\, k''[\mathtt{A}''] \in \mathbf{dom}(\Gamma \triangledown \Gamma') \quad X \notin \mathbf{dom}(\Gamma \triangledown \Gamma') \quad \Gamma'_x \rhd_X (\Gamma'', \widetilde{k'[\mathtt{A}'] : T'}), C'' \qquad \Gamma_x \rhd_X (\Gamma', \widetilde{k[\mathtt{A}] : T}), C' \qquad \Gamma'|_{\mathbf{locs}} \subseteq \Gamma'}{(\Gamma' \triangledown \Gamma''), \mathsf{solve}(\widetilde{k[\mathtt{A}] : T} \triangledown \widetilde{k'[\mathtt{A}'] : T'}, \mathbf{t}_X) \vdash_{\mathtt{min}} \mathtt{def}\ X = C''\ \mathtt{in}\ C'}\ \lfloor^{\mathsf{Min}}|_{\mathsf{Def}}\rceil$$

To prove $[\![\Gamma]\!]^C \vdash_{\mathtt{min}} [\![C]\!]$, we consider the processes $\mathtt{p} \in \tilde{\mathtt{p}} = \mathbf{pn}([\![C]\!])$ with cardinality $[1, n]$ and we let
– $[\![C]\!] = \prod_{\mathtt{p}} C_{\mathtt{p}}$
– $C_{\mathtt{p}} = \mathtt{def}\ X_{\mathtt{p}} = [\![C''[X_{\mathtt{p}}/X]]\!]_{\mathtt{p}}\ \mathtt{in}\ [\![C'[X_{\mathtt{p}}/X]]\!]_{\mathtt{p}}$
– $\Gamma_c = [\![\Gamma]\!]^C$
– $\widetilde{k_{\mathtt{p}}[\mathtt{A}] : T} = \left\{k[\mathtt{A}] : T \mid \{\mathtt{p}\colon k[\mathtt{A}], k[\mathtt{A}]\colon T\} \subseteq \Gamma_c \wedge k[\mathtt{A}]\colon T \in \widetilde{k[\mathtt{A}] : T}\right\}$
– $\widetilde{k'_{\mathtt{p}}[\mathtt{A}'] : T'} = \left\{k'[\mathtt{A}'] : T' \mid \{\mathtt{p}\colon k'[\mathtt{A}'], k'[\mathtt{A}']\colon T'\} \subseteq \Gamma_c \wedge k'[\mathtt{A}']\colon T' \in \widetilde{k'[\mathtt{A}'] : T'}\right\}$
The case is proved by the derivation Δ_1 where

$$\Delta_i = \dfrac{\pi_{\mathtt{p}_i} \quad \dfrac{\Delta_{i+1}}{\bigcup_{\mathtt{p} \in \{\mathtt{p}_{i+1}, \ldots, \mathtt{p}_n\}} \Gamma_c|_{\mathtt{p}} \vdash_{\mathtt{min}} \prod_{\mathtt{p} \in \{\mathtt{p}_{i+1}, \ldots, \mathtt{p}_n\}} C_{\mathtt{p}}}\ \lfloor^{\mathsf{Min}}|_{\mathsf{Par}}\rceil}{\Gamma_c|_{\mathtt{p}_i}, \bigcup_{\mathtt{p} \in \{\mathtt{p}_{i+1}, \ldots, \mathtt{p}_n\}} \Gamma_c|_{\mathtt{p}} \vdash_{\mathtt{min}} C_{\mathtt{p}_i} \mid \prod_{\mathtt{p} \in \{\mathtt{p}_{i+1}, \ldots, \mathtt{p}_n\}} C_{\mathtt{p}}}\ \lfloor^{\mathsf{Min}}|_{\mathsf{Par}}\rceil$$

and

$$\pi_{\mathsf{p}} = \frac{\begin{array}{c}\Gamma_x(X_{\mathsf{p}}) = \Gamma'_x(X_{\mathsf{p}}) \text{ if } X_{\mathsf{p}} \in \mathbf{dom}(\Gamma_x) \cap \mathbf{dom}(\Gamma'_x) \\ \nexists k''[\mathtt{A}'']\colon \in \mathbf{dom}\left(\llbracket\Gamma'\rrbracket^C\big|_{\mathsf{p}} \triangledown \llbracket\Gamma''\rrbracket^C\big|_{\mathsf{p}}\right) \qquad \Gamma'_x \rhd \left(\llbracket\Gamma''\rrbracket^C\big|_{\mathsf{p}}, \widetilde{k'_{\mathsf{p}}[\mathtt{A}']\colon T'}\right), \llbracket C''[X_{\mathsf{p}}/X]\rrbracket_{\mathsf{p}} \\ \Gamma_x \rhd \left(\llbracket\Gamma'\rrbracket^C\big|_{\mathsf{p}}, \widetilde{k_{\mathsf{p}}[\mathtt{A}]\colon T}\right), \llbracket C''[X_{\mathsf{p}}/X]\rrbracket_{\mathsf{p}} \qquad \llbracket\Gamma''\rrbracket^C\big|_{\mathsf{p}}\Big|_{\mathsf{locs}} \subseteq \llbracket\Gamma'\rrbracket^C\big|_{\mathsf{p}}\end{array}}{\llbracket\Gamma'\rrbracket^C\big|_{\mathsf{p}} \triangledown \llbracket\Gamma''\rrbracket^C\big|_{\mathsf{p}}, \mathsf{solve}\left(\widetilde{k_{\mathsf{p}}[\mathtt{A}]\colon T} \triangledown \widetilde{k'_{\mathsf{p}}[\mathtt{A}']\colon T'}, \mathbf{t}_{X_{\mathsf{p}}}\right) \vdash_{\mathtt{min}} \mathtt{def}\ X_{\mathsf{p}} = \llbracket C''[X_{\mathsf{p}}/X]\rrbracket_{\mathsf{p}}\ \mathtt{in}\ \llbracket C'[X_{\mathsf{p}}/X]\rrbracket_{\mathsf{p}}}\ \lfloor^{\mathsf{Min}}|_{\mathsf{Def}}\rceil$$

Essentially, using the filtrations $\Gamma|_{\mathsf{p}}$ and the partitions $\widetilde{k_{\mathsf{p}}[\mathtt{A}]\colon T}$ and $\widetilde{k'_{\mathsf{p}}[\mathtt{A}']\colon T'}$ in Δ_i, we shape $\llbracket\Gamma\rrbracket^C$ in such a way that its partitions contain all and only the typings (variable, ownership, definitions) that minimally type the endpoint choreography C_{p}, with the exception of service typings, which are duplicated in all filtrations (as per its definition). However, this is not a problem, as we consider a weakened form of minimal typing that allows for additional, unused service typings.

Such a partitioning of $\llbracket\Gamma\rrbracket^C$ is possible by the definitions of $\llbracket\Gamma\rrbracket^C$ and $\triangledown$ (and $\prec$ by extension):

$$\begin{array}{rcl}\llbracket\Gamma\rrbracket^C & = & \left[\!\!\left[(\Gamma'\triangledown\Gamma''), \mathsf{solve}(\widetilde{k[\mathtt{A}] : T} \triangledown \widetilde{k'[\mathtt{A}'] : T'}, \mathbf{t}_X)\right]\!\!\right]^C = \\ & = & \bigcup\limits_{\mathsf{p}\in\tilde{\mathsf{p}}} \left(\left(\llbracket\Gamma'\rrbracket^C \triangledown \llbracket\Gamma''\rrbracket^C\right), \left[\!\!\left[\mathsf{solve}(\widetilde{k[\mathtt{A}] : T} \triangledown \widetilde{k'[\mathtt{A}'] : T'}, \mathbf{t}_X)\right]\!\!\right]^C\right)\Big|_{\mathsf{p}} = \\ & = & \bigcup\limits_{\mathsf{p}\in\tilde{\mathsf{p}}} \left(\left(\llbracket\Gamma'\rrbracket^C \triangledown \llbracket\Gamma''\rrbracket^C\right), \mathsf{solve}(\widetilde{k[\mathtt{A}] : T} \triangledown \widetilde{k'[\mathtt{A}'] : T'}, \mathbf{t}_X)\right)\Big|_{\mathsf{p}}\end{array}$$

Finally, we simply rename $\mathbf{t}_X$ to $\mathbf{t}_{X_{\mathsf{p}}}$ (in each filtration $\mathsf{p} \in \tilde{\mathsf{p}}$).

Then in π_{p} we prove the partition $\Gamma_c|_{\mathsf{p}}$ to minimally type the endpoint choreography C_{p}. All preconditions in π_{p} hold as the environments $\llbracket\Gamma'\rrbracket^C$, $\llbracket\Gamma''\rrbracket^C$, $\widetilde{k_{\mathsf{p}}[\mathtt{A}]\colon T}$, and $\widetilde{k'_{\mathsf{p}}[\mathtt{A}']\colon T'}$, contain those and only definition, ownership, variable, and session types related to process p (with the exception of duplicated service typings) and originally contained in Γ', Γ'',$\widetilde{k[\mathtt{A}]\colon T}$, and $\widetilde{k'[\mathtt{A}']\colon T'}$. Definition typing identifiers are properly renamed to be unique for p (i.e., from X to X_{p}).

Case $\lfloor^{\mathsf{Min}}|_{\mathsf{End}}\rceil$

Trivial.

Case $\lfloor^{\mathsf{Min}}|_{\mathsf{Call}}\rceil$

From the premises we know that $C = X$, on which we can apply the typing derivation

$$\frac{\begin{array}{c}\Gamma' = \mathsf{vars} \cup \mathsf{ownerships} \qquad \widetilde{k'[\mathtt{A}']} = \mathsf{sessions} \setminus \{\widetilde{k[\mathtt{A}]}\} \\ \Gamma'' = \mathsf{vars}(X) \cup \mathsf{ownerships}(X) \quad \widetilde{k[\mathtt{A}]} = \mathsf{sessions}(\mathsf{X}) \qquad \Gamma'' \subseteq \Gamma'\end{array}}{\Gamma', \widetilde{k[\mathtt{A}]} : \mathbf{t}_X, \widetilde{k'[\mathtt{A}']} : \mathtt{end}, X\colon (\Gamma'', \widetilde{k[\mathtt{A}]} : \mathbf{t}_X) \vdash_{\mathtt{min}} X}\ \lfloor^{\mathsf{Min}}|_{\mathsf{Call}}\rceil$$

Thus, in the case, $\Gamma = \Gamma', \widetilde{k[\mathtt{A}]} : \mathbf{t}_X, \widetilde{k'[\mathtt{A}']} : \mathtt{end}, X\colon (\Gamma'', \widetilde{k[\mathtt{A}]} : \mathbf{t}_X)$. Given our assumption of well sortedness, we can consider as EPP of X the composition

$$\llbracket X\rrbracket = \prod_{\mathsf{p}\in\tilde{\mathsf{p}}} X_{\mathsf{p}}$$

Where processes $\tilde{\mathsf{p}}$ are a subset of the processes present both in the prefix of procedure call X in C and in the typing environment Γ (we recall, Γ contains typings that are coalesced in $[\![\Gamma]\!]^C$). From the definition of $[\![\Gamma]\!]^C$, we can write

$$\Gamma_c = [\![\Gamma]\!]^C = [\![\Gamma']\!]^C, \widetilde{k_{\mathsf{p}}[\mathtt{A}]} : \mathbf{t}_X, \widetilde{k'_{\mathsf{p}}[\mathtt{A}']} : \mathtt{end}, \bigcup_{\mathsf{p}\in\tilde{\mathsf{p}}} X_{\mathsf{p}} \colon ([\![\Gamma'']\!]_{\mathsf{p}}, \widetilde{k_{\mathsf{p}}[\mathtt{A}]} : \mathbf{t}_{X_{\mathsf{p}}})$$

where

– $\widetilde{k_{\mathsf{p}}[\mathtt{A}]} \colon \mathbf{t}_{X_{\mathsf{p}}} = \left\{k_{\mathsf{p}}[\mathtt{A}] \colon \mathbf{t}_{X_{\mathsf{p}}} \mid \{\mathsf{p} \colon k[\mathtt{A}], k[\mathtt{A}] \colon \mathbf{t}_X\} \subseteq \Gamma\right\}$

– $\widetilde{k'_{\mathsf{p}}[\mathtt{A}']} \colon \mathtt{end} = \left\{k'_{\mathsf{p}}[\mathtt{A}'] \colon \mathtt{end} \mid \{\mathsf{p} \colon k'[\mathtt{A}'], k'[\mathtt{A}'] \colon \mathtt{end}\} \subseteq \Gamma\right\}$

Finally, let the cardinality of $\tilde{\mathsf{p}}$ be $[1, n]$. The case is proved by the derivation Δ_1 where

$$\Delta_i = \dfrac{\pi_{\mathsf{p}_i} \quad \dfrac{\Delta_{i+1}}{\bigcup_{\mathsf{p}\,\in\,\{\mathsf{p}_{i+1},\ldots,\mathsf{p}_n\}} \Gamma_c|_{\mathsf{p}} \vdash_{\mathtt{min}} \prod_{\mathsf{p}\in\{\mathsf{p}_{i+1},\ldots,\mathsf{p}_n\}} X_{\mathsf{p}}}\ \lfloor^{\mathsf{Min}}|_{\mathsf{Par}}\rceil}{\Gamma_c|_{\mathsf{p}_i}, \bigcup_{\mathsf{p}\,\in\,\{\mathsf{p}_{i+1},\ldots,\mathsf{p}_n\}} \Gamma_c|_{\mathsf{p}} \vdash_{\mathtt{min}} X_{\mathsf{p}_i} \mid \prod_{\mathsf{p}\in\{\mathsf{p}_{i+1},\ldots,\mathsf{p}_n\}} X_{\mathsf{p}}}\ \lfloor^{\mathsf{Min}}|_{\mathsf{Par}}\rceil$$

and

$$\pi_{\mathsf{p}} = \dfrac{[\![\Gamma']\!]^C\Big|_{\mathsf{p}} = \mathsf{vars} \cup \mathsf{ownerships} \qquad \widetilde{k'_{\mathsf{p}}[\mathtt{A}']} = \mathsf{sessions} \setminus \{\widetilde{k_{\mathsf{p}}[\mathtt{A}]}\} \quad [\![\Gamma']\!]^C\Big|_{\mathsf{p}} = \mathsf{vars}(X_{\mathsf{p}}) \cup \mathsf{ownerships}(X_{\mathsf{p}}) \quad \widetilde{k_{\mathsf{p}}[\mathtt{A}]} = \mathsf{sessions}(\mathsf{X}_{\mathsf{p}}) \qquad [\![\Gamma'']\!]_{\mathsf{p}} \subseteq [\![\Gamma']\!]^C\Big|_{\mathsf{p}}}{[\![\Gamma']\!]^C\Big|_{\mathsf{p}}, \widetilde{k_{\mathsf{p}}[\mathtt{A}]} \colon \mathbf{t}_{X_{\mathsf{p}}}, \widetilde{k'_{\mathsf{p}}[\mathtt{A}']} \colon \mathtt{end}, X_{\mathsf{p}} \colon ([\![\Gamma'']\!]_{\mathsf{p}}, \widetilde{k_{\mathsf{p}}[\mathtt{A}]} : \mathbf{t}_{X_{\mathsf{p}}}) \vdash_{\mathtt{min}} X_{\mathsf{p}}}\ \lfloor^{\mathsf{Min}}|_{\mathsf{Call}}\rceil$$

Where in π_p we consider the usage of auxiliary functions vars, owenerships, and sessions on the projection $[\![C]\!]_{\mathsf{p}}$.

□

We finally prove Theorem 8.

Proof of EPP Typing Preservation. From Theorem 8, we have that $\Gamma = \Gamma_d, \Gamma_c$ and we need to prove that we can apply rule $\lfloor^{\mathsf{Min}}|_{\mathsf{DC}}\rceil$ on $\Gamma_d, [\![\Gamma_c]\!]^C \vdash_{\mathtt{min}} D, [\![C]\!]$

$$\dfrac{\mathbf{pco}(\Gamma_d, [\![\Gamma_c]\!]^C) \quad \Gamma_d \vdash D \quad [\![\Gamma_c]\!]^C \vdash_{\mathtt{min}} [\![C]\!]}{\Gamma_d, [\![\Gamma_c]\!]^C \vdash_{\mathtt{min}} D, [\![C]\!]}\ \lfloor^{\mathsf{Min}}|_{\mathsf{DC}}\rceil$$

where

- $\mathbf{pco}(\Gamma_d, [\![\Gamma]\!]^C)$ holds as, regarding session typings, $[\![\Gamma]\!]^C$ just coalesces session typings and their related ownerships of service processes;
- $\Gamma_d \vdash_{\mathtt{min}} D$ holds as per premises of Theorem 8;
- $[\![\Gamma]\!] \vdash_{\mathtt{min}} [\![C]\!]$ holds from Lemma 8 and the assumption of well-sortedness on C (if C is well-sorted also $[\![C]\!]$ is well-sorted and typable by $[\![\Gamma]\!]^C$).

□

B.5. **EPP Theorem.** Before proving Theorem 5 we define some auxiliary concepts to establish a correspondence between a choreography and its projection.

Lemma 9 (EPP Swap Invariance)**.** Let $C \simeq_{\mathsf{C}} C'$ then $[\![C]\!] \simeq_{\mathsf{C}} [\![C']\!]$.

Proof Sketch. In the proof we show that the projection is invariant under the rules for the swapping relation $\simeq_{\mathsf{C}}$ defined in Figure 7. $\lfloor^{\mathsf{CS}}_{\mathsf{EtaEta}}\rceil$ is trivial. For rule $\lfloor^{\mathsf{CS}}_{\mathsf{EtaCnd}}\rceil$ we need to check that the projections of the processes in the swapped interaction η do not change, which holds by the definition of EPP for (*cond*) terms and the merging operator (merging the same η returns η). The same reasoning on the EPP and the merging operator applies to all other cases. □

Lemma 10 (EPP under $\equiv$)**.** Let $C \equiv_{\mathsf{C}} C'$ then $[\![C]\!] \equiv_{\mathsf{C}} [\![C']\!]$.

Proof . Easy by cases on the rules of $\equiv_{\mathsf{C}}$. □

Lemma 11 (Compositional EPP)**.** Let C be well-typed and $C = C_1 \mid C_2$ then $[\![C]\!] \equiv_{\mathsf{C}} [\![C_1]\!] \mid [\![C_2]\!]$.

Proof . By definition of EPP

$$[\![C]\!] = \prod_{\mathsf{p} \,\in\, \mathbf{fp}(C)} [\![C]\!]_{\mathsf{p}} \mid \prod_{l} \left(\bigsqcup_{\mathsf{s} \in \lfloor C \rfloor_l} [\![C]\!]_{\mathsf{s}} \right)$$

Since C is well-typed and $C = C_1 \mid C_2$, rule $\lfloor^{\Gamma}_{\mathsf{Par}}\rceil$ applies and by definition of Γ_1, Γ_2 there cannot be a process p such that $\mathsf{p} \in \mathbf{fp}(C_1) \cap \mathbf{fp}(C_2)$. Therefore we can write

$$[\![C]\!] \equiv_{\mathsf{C}} \prod_{\mathsf{p} \,\in\, \mathbf{fp}(C_1)} [\![C_1]\!]_{\mathsf{p}} \mid \prod_{\mathsf{q} \,\in\, \mathbf{fp}(C_2)} [\![C_2]\!]_{\mathsf{q}} \mid \prod_{l} \left(\bigsqcup_{\mathsf{s} \in \lfloor C \rfloor_l} [\![C]\!]_{\mathsf{s}} \right)$$

By the definition of service typing we know that *i*) locations can implement only one role in a choreography and *ii*) a location can appear only in one service typing. Therefore there cannot be two service processes at the same location in C_1 and C_2. Thus we can write

$$[\![C]\!] \equiv_{\mathsf{C}} \underbrace{\prod_{\mathsf{p} \,\in\, \mathbf{fp}(C_1)} [\![C_1]\!]_{\mathsf{p}}}_{C_1^a} \mid \underbrace{\prod_{\mathsf{q} \,\in\, \mathbf{fp}(C_2)} [\![C_2]\!]_{\mathsf{q}}}_{C_2^a} \mid \underbrace{\prod_{l} \left(\bigsqcup_{\mathsf{r} \,\in\, \lfloor C_1 \rfloor_l} [\![C_1]\!]_{\mathsf{r}} \right)}_{C_1^s} \mid \underbrace{\prod_{l'} \left(\bigsqcup_{\mathsf{s} \,\in\, \lfloor C_2 \rfloor_{l'}} [\![C_2]\!]_{\mathsf{s}} \right)}_{C_2^s}$$

where $[\![C_1]\!] = C_1^a \mid C_1^s$ and $[\![C_2]\!] = C_2^a \mid C_2^s$ by definition of EPP. □

B.5.1. *Pruning.* Following our definition of EPP, the projection of (*start*) terms on service processes yield a parallel composition of (*acc*) terms on the locations subject of the (*start*). However, the reduction of a (*start*) term might remove the availability to start new processes on the locations subject of the (*start*) (i.e., if the reductum does not contain another (*start*) term on the same locations). Contrarily, (*acc*) terms remain always available.

A similar observation can be drawn between conditional branches that contain (*com*) terms whose projection merges all possible communications into (*recv*) and (*send*) terms. Also in this case, reducing the condition and projecting the result we obtain a subset of all possible branches for the considered communication.

Similarly to [MY13] and [CHY12], we deal with these asymmetries by introducing the *pruning relation* (see Definition 11), which allows us to ignore unused *i*) endpoint services and *ii*) input branches.

Before continuing with the last auxiliary results and the proof of Theorem 5 we need to extend the labels of the semantics of annotated Frontend Choreographies (see appendix B.1.1) with the identifiers of the processes involved in a reduction

$$\beta ::= k : \mathtt{p}[\mathtt{A}] \longrightarrow \mathtt{B}.o \mid \mathtt{A}\rangle\mathtt{q}[\mathtt{B}].o(x) \mid \tau@\mathtt{p} \mid \tau$$

and the annotation of the reduction with rule $\lfloor \mathsf{C} |_{\mathsf{Cond}} \rceil$ as

$$\frac{i = 1 \text{ if } \mathbf{eval}(e, D(\mathtt{p})) = \text{ true}, i = 2 \text{ otherwise}}{D,\ \mathtt{if}\ \mathtt{p}.e\ \{C_1\}\ \mathtt{else}\ \{C_2\} \quad \xrightarrow{\tau@\mathtt{p}} \quad D,\ C_i}\ \lfloor \mathsf{C} |_{\mathsf{Cond}} \rceil$$

Let also $\mathbf{pn}(k : \mathtt{p}[\mathtt{A}] \longrightarrow \mathtt{B}.o) = \{\mathtt{p}\}$, $\mathbf{pn}(\mathtt{A}\rangle\mathtt{q}[\mathtt{B}].o(x)) = \{\mathtt{q}\}$, $\mathbf{pn}(\tau@\mathtt{p}) = \{\mathtt{p}\}$, and $\mathbf{pn}(\tau) = \varnothing$

Lemma 12 (Passive Processes Pruning Invariance)**.** $D, C \xrightarrow{\beta} D', C'$ implies that for all $\mathtt{p} \in \mathbf{fp}(C) \setminus \mathbf{pn}(\beta)$, $[\![C']\!]_{\mathtt{p}} \prec [\![C]\!]_{\mathtt{p}}$.

Proof Sketch. By cases on the derivation of C. The only interesting case is $\lfloor \mathsf{C} |_{\mathsf{Cond}} \rceil$ in which the projection of the processes receiving selections are merged. The thesis follows directly from Definition 11 and Lemmas 9 and 10.

□

B.6. **Proof of Theorem 5.** We restate items (2) and (3) of Theorem 5 to include annotated reductions.

Theorem 5 (EPP Operational Correspondence)
Let D, C be well-typed and well-annotated. Then,

(1) *(Completeness) $D, C \xrightarrow{\beta} D', C'$ implies $D, [\![C]\!] \xrightarrow{\beta} D', C''$ and $[\![C']\!] \prec C''$.*
(2) *(Soundness) $D, [\![C]\!] \xrightarrow{\beta} D', C''$ implies $D, C \xrightarrow{\beta} D', C'$ and $[\![C']\!] \prec C''$.*

We report below the respective proofs of *(Completeness)* and *(Soundness)* separately.
Proof (Completeness).

Proof by induction on the derivation of $D, C \xrightarrow{\beta} D', C'$.

: Case $\lfloor \mathsf{C} |_{\mathsf{Send}} \rceil$
we know that $C = k : \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o; C_c$ and we can write the derivation

$$\frac{\eta = k : \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o \qquad D, k : \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o \blacktriangleright D'}{D,\ \eta; C \quad \xrightarrow{k\ \mathtt{p}[\mathtt{A}] \longrightarrow \mathtt{B}.o} \quad D',\ C_c}\ \lfloor \mathsf{C} |_{\mathsf{Send}} \rceil$$

and $C' = C_c$.
From the definition of EPP we have that $[\![C]\!] = C_{act} \mid C_s$ such that

$$C_{act} = k : \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o; [\![C_c]\!]_{\mathtt{p}} \mid \prod_{\mathtt{r} \in \mathbf{fp}(C) \setminus \{\mathtt{p}\}} [\![C_c]\!]_{\mathtt{r}}$$

and

$$C_s = \prod_{l} \left(\bigsqcup_{\mathtt{s} \in \lfloor C \rfloor_l} [\![C_c]\!]_{\mathtt{s}} \right)$$

While $[\![C']\!] \equiv_{\mathsf{C}} C'_{act} \mid C_s$

$$C'_{act} = [\![C_c]\!]_{\mathsf{p}} \mid \prod_{\mathsf{r} \in \mathbf{fp}(C')\setminus\{\mathsf{p}\}} [\![C_c]\!]_{\mathsf{r}}$$

We can apply Rules $\lfloor^{\mathsf{C}}_{\mathsf{Par}}\rceil$, $\lfloor^{\mathsf{C}}_{\mathsf{Eq}}\rceil$, and $\lfloor^{\mathsf{C}}_{\mathsf{Send}}\rceil$ on $D, [\![C]\!]$ such that

$$\begin{array}{c} \eta = k : \mathsf{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o \qquad D, \eta \blacktriangleright D'' \\ \vdots \ \lfloor^{\mathsf{C}}_{\mathsf{Par}}\rceil \\ D,\ [\![C]\!] \xrightarrow{k\ \mathsf{p}[\mathtt{A}] \longrightarrow \mathtt{B}.o} D'',\ C'' \end{array}$$

for which it holds that $D' = D''$ by rule $\lfloor^{\mathsf{D}}_{\mathsf{Send}}\rceil$.

$$C'' = [\![C_c]\!]_{\mathsf{p}} \mid \prod_{\mathsf{r} \in \mathbf{fp}(C')\setminus\{\mathsf{p}\}} [\![C_c]\!]_{\mathsf{r}} \mid C_s$$

for which it holds that $[\![C']\!] \prec C''$.

: Case $\lfloor^{\mathsf{C}}_{\mathsf{Recv}}\rceil$

we know that $D, C = D,\ k : \mathtt{A} \longrightarrow \mathsf{q}[\mathtt{B}].\{o_i(x_i); C_i\}_{i\in I}$ and we can write the derivation

$$\frac{j \in I \qquad D, k : \mathtt{A} \longrightarrow \mathsf{q}[\mathtt{B}].o_j(x_j) \blacktriangleright D'}{D,\ k : \mathtt{A} \longrightarrow \mathsf{q}[\mathtt{B}].\{o_i(x_i); C_i\}_{i\in I} \xrightarrow{k:\mathtt{A}\rangle\mathsf{q}[\mathtt{B}].o_j(x_j)} D',\ C_j}\ \lfloor^{\mathsf{C}}_{\mathsf{Recv}}\rceil$$

for $\beta = k : \mathtt{A}\rangle\mathsf{q}[\mathtt{B}].o_j(x_j)$ and $C' = C_j$.

By the definition of EPP we have

$$[\![C]\!] \equiv_{\mathsf{C}} k : \mathtt{A} \longrightarrow \mathsf{q}[\mathtt{B}].\left\{o_i(x_i); [\![C_i]\!]_{\mathsf{q}}\right\}_{i\in I} \mid \prod_{\mathsf{p} \in \mathbf{fp}(C)\setminus\{\mathsf{q}\}} \left(\bigsqcup_{i\ \in\ I} [\![C_i]\!]_{\mathsf{p}}\right) \mid \prod_{l} \left(\bigsqcup_{\mathsf{r}\ \in\ \lfloor C\rfloor_l} [\![C]\!]_{\mathsf{r}}\right)$$

Then we can apply rules $\lfloor^{\mathsf{C}}_{\mathsf{Par}}\rceil$, $\lfloor^{\mathsf{C}}_{\mathsf{Eq}}\rceil$, and $\lfloor^{\mathsf{C}}_{\mathsf{Recv}}\rceil$ such that

$$\begin{array}{c} j \in I \qquad D, k : \mathtt{A} \longrightarrow \mathsf{q}[\mathtt{B}].o_j(x_j) \blacktriangleright D'' \\ \vdots \ \lfloor^{\mathsf{C}}_{\mathsf{Par}}\rceil \\ D,\ [\![C]\!] \xrightarrow{k:\mathtt{A}\rangle\mathsf{q}[\mathtt{B}].o_j(x_j)} D'',\ [\![C_j]\!]_{\mathsf{q}} \mid \prod_{\mathsf{p} \in \mathbf{fp}(C)\setminus\{\mathsf{q}\}} \left(\bigsqcup_{i\ \in\ I} [\![C_i]\!]_{\mathsf{p}}\right) \mid \prod_{l} \left(\bigsqcup_{\mathsf{r}\ \in\ \lfloor C\rfloor_l} [\![C]\!]_{\mathsf{r}}\right) \end{array}$$

and

$$C'' = [\![C_j]\!]_{\mathsf{q}} \mid \prod_{\mathsf{p} \in \mathbf{fp}(C)\setminus\{\mathsf{q}\}} \left(\bigsqcup_{i\ \in\ I} [\![C_i]\!]_{\mathsf{p}}\right) \mid \prod_{l} \left(\bigsqcup_{\mathsf{r}\ \in\ \lfloor C\rfloor_l} [\![C]\!]_{\mathsf{r}}\right)$$

From rule $\lfloor^{\mathsf{D}}_{\mathsf{Recv}}\rceil$ we know that $D'' = D'$. Finally $[\![C']\!] \prec C''$ by Definition 11 and Lemma 12.

: Case $\lfloor^{\mathsf{C}}_{\mathsf{Start}}\rceil$

we know that $C = \mathtt{start}\ k : \mathsf{p}[\mathtt{A}] \leftrightarrow \widetilde{l.\mathsf{q}[\mathtt{B}]}; C_c$ and we can write the derivation

$$\frac{D\#k', \tilde{\mathsf{r}} \quad \delta = \mathtt{start}\ k' : \mathsf{p}[\mathtt{A}] \leftarrow\!\!\rightarrow \widetilde{l.\mathsf{q}[\mathtt{B}]} \quad D, \delta \blacktriangleright D'}{D,\ \mathtt{start}\ k : \mathsf{p}[\mathtt{A}] \leftarrow\!\!\rightarrow \widetilde{l.\mathsf{q}[\mathtt{B}]}; C \quad \to \quad D',\ C[k'/k][\tilde{\mathsf{r}}/\tilde{\mathsf{q}}]}\ \lfloor^{\mathsf{C}}|_{\mathsf{Start}}\rceil$$

and $C' = C_c[k'/k][\tilde{\mathsf{r}}/\tilde{\mathsf{q}}]$.

From the definition of EPP we have

$$[\![C']\!] = \prod_{\mathsf{q}\ \in\ \mathbf{fp}(C')} [\![C']\!]_{\mathsf{q}} \mid \prod_{l} \left(\bigsqcup_{\mathsf{s}\ \in\ \lfloor C' \rfloor_l} [\![C']\!]_{\mathsf{s}} \right)$$

and

$$[\![C]\!] \equiv_{\mathtt{C}} \begin{cases} & \mathtt{req}\ k : \mathsf{p}[\mathtt{A}] \leftarrow\!\!\rightarrow \widetilde{l.\mathtt{B}}; [\![C_c]\!]_{\mathsf{p}} \\ \mid & \prod_{l.\mathsf{q}[\mathtt{B}]\ \in\ \widetilde{l.\mathsf{q}[\mathtt{B}]}} \mathtt{acc}\ k : l.\mathsf{q}[\mathtt{B}]; [\![C_c]\!]_{\mathsf{q}} \\ \mid & \prod_{\mathsf{r}\ \in\ \mathbf{fp}(C)\setminus\{\mathsf{p}\}} [\![C]\!]_{\mathsf{r}} \\ \mid & \prod_{l' \notin\ \tilde{l}} \left(\prod_{\mathsf{s}\ \in\ \lfloor C \rfloor_{l'}} [\![C]\!]_{\mathsf{s}} \right) \end{cases}$$

we can apply rules $\lfloor^{\mathsf{C}}|_{\mathsf{Par}}\rceil$, $\lfloor^{\mathsf{C}}|_{\mathsf{Eq}}\rceil$, $\lfloor^{\mathsf{C}}|_{\mathsf{PStart}}\rceil$ such that

$$\begin{array}{c} i \in \{1, \ldots, n\} \qquad D\#k'', \tilde{\mathsf{r}}' \qquad \{\widetilde{l.\mathtt{B}}\} = \biguplus_i \{\widetilde{l_i.\mathtt{B}_i}\}_i \qquad \{\tilde{\mathsf{r}}'\} = \bigcup_i \{\tilde{\mathsf{r}}'_i\} \\ \delta = \mathtt{start}\ k'' : \mathsf{p}[\mathtt{A}] \leftarrow\!\!\rightarrow \widetilde{l_1.\mathsf{r}'_1[\mathtt{B}_1]}, \ldots, \widetilde{l_n.\mathsf{r}'_n[\mathtt{B}_n]} \qquad D, \delta \blacktriangleright D'' \\ \vdots\ \lfloor^{\mathsf{C}}|_{\mathsf{Par}}\rceil \\ D, [\![C]\!] \xrightarrow{\tau} D'', C'' \end{array}$$

where

$$C'' \equiv_{\mathtt{C}} \begin{cases} & [\![C_c]\!]_{\mathsf{p}}\,[k''/k] \\ \mid & \prod_{(\mathsf{q},\mathsf{r}')\ \in\ \left\{(\mathsf{q}_1,\mathsf{r}'_1),\ldots,(\mathsf{q}_n,\mathsf{r}'_n)\right\}} [\![C_c]\!]_{\mathsf{q}}\,[k''/k][\mathsf{q}/\mathsf{r}'] \\ \mid & \prod_{\mathsf{r}\ \in\ \mathbf{fp}(C_c)\setminus\{\mathsf{p},\tilde{\mathsf{q}}\}} [\![C_c]\!]_{\mathsf{r}} \\ \mid & \prod_{l.\mathsf{q}[\mathtt{B}]\ \in\ \widetilde{l.\mathsf{q}[\mathtt{B}]}} \mathtt{acc}\ k : l.\mathsf{q}[\mathtt{B}]; [\![C_c]\!]_{\mathsf{q}} \\ \mid & \prod_{l' \notin\ \tilde{l}} \left(\prod_{\mathsf{s}\ \in\ \lfloor C_c \rfloor_{l'}} [\![C_c]\!]_{\mathsf{s}} \right) \end{cases}$$

Observe that we can α-rename k'' to k' and $\tilde{\mathsf{r}}'$ to $\tilde{\mathsf{r}}$ as k'', k', $\tilde{\mathsf{r}}'$, and $\tilde{\mathsf{r}}$ are all fresh wrt D, C.

From the application of rule $\lfloor^{\mathsf{D}}|_{\mathsf{Start}}\rceil$ we can find Γ such that

$$\Gamma \vdash_{\mathtt{min}} (\ D'', C''\ )[k'/k''][\tilde{\mathsf{r}}/\tilde{\mathsf{r}}']$$

and

$$\Gamma \vdash_{\mathtt{min}} (D', C'')[k'/k''][\tilde{\mathsf{r}}/\tilde{\mathsf{r}}']$$

and by α-renaming we have that

$$D, [\![C]\!] \xrightarrow{\tau} D', C''[k'/k''][\tilde{\mathsf{r}}/\tilde{\mathsf{r}}']$$

Finally $[\![C']\!] \prec C''[k'/k''][\tilde{\mathsf{r}}/\tilde{\mathsf{r}}']$ by Lemma 12.

: Case $\lfloor \mathsf{C}|_{\mathsf{PStart}} \rceil$

Similar to (in particular the second part of) the proof of case $\lfloor \mathsf{C}|_{\mathsf{Start}} \rceil$.

: Case $\lfloor \mathsf{C}|_{\mathsf{Cond}} \rceil$

we know that $C \equiv_{\mathsf{C}} \texttt{if } \mathsf{p}.e\ \{C_1\}\ \texttt{else}\ \{C_2\}$ and we can write the derivation

$$\frac{i = 1 \text{ if } \mathbf{eval}(e, D(\mathsf{p})) = \text{ true}, i = 2 \text{ otherwise}}{D,\ \texttt{if } \mathsf{p}.e\ \{C_1\}\ \texttt{else}\ \{C_2\} \quad \xrightarrow{\tau@\mathsf{p}} \quad D,\ C_i}\ \lfloor \mathsf{C}|_{\mathsf{Cond}} \rceil$$

We only consider the case for $\mathbf{eval}(e, D(\mathsf{p})) = \text{true}$ as $\mathbf{eval}(e, D(\mathsf{p})) = \text{false}$ is similar. $C' = C_1$ and by the definition of EPP

$$[\![C]\!] \equiv_{\mathsf{C}} \texttt{if } \mathsf{p}.e\ \{[\![C_1]\!]_{\mathsf{p}}\}\ \texttt{else}\ \{[\![C_2]\!]_{\mathsf{p}}\} \mid \prod_{\mathsf{q} \in \mathbf{fp}(C')\setminus\{\mathsf{p}\}} [\![C_1]\!]_{\mathsf{q}} \sqcup [\![C_2]\!]_{\mathsf{q}} \mid \prod_{l} \left(\bigsqcup_{\mathsf{r} \in \lfloor C \rfloor_l} [\![C]\!]_{\mathsf{r}} \right)$$

and

$$[\![C']\!] \equiv_{\mathsf{C}} [\![C_1]\!]_{\mathsf{p}} \mid \prod_{\mathsf{q} \in \mathbf{fp}(C')\setminus\{\mathsf{p}\}} [\![C_1]\!]_{\mathsf{q}} \mid \prod_{l} \left(\bigsqcup_{\mathsf{r} \in \lfloor C_1 \rfloor_l} [\![C_1]\!]_{\mathsf{r}} \right)$$

We can apply rules $\lfloor \mathsf{C}|_{\mathsf{Par}} \rceil$, $\lfloor \mathsf{C}|_{\mathsf{Eq}} \rceil$, and $\lfloor \mathsf{C}|_{\mathsf{Cond}} \rceil$ such that $D, [\![C]\!] \xrightarrow{\tau@\mathsf{p}} D, C''$ where

$$C'' = [\![C_1]\!]_{\mathsf{p}} \mid \prod_{\mathsf{q} \in \mathbf{fp}(C')\setminus\{\mathsf{p}\}} [\![C_1]\!]_{\mathsf{q}} \sqcup [\![C_2]\!]_{\mathsf{q}} \mid \prod_{l} \left(\bigsqcup_{\mathsf{r} \in \lfloor C \rfloor_l} [\![C]\!]_{\mathsf{r}} \right)$$

and $[\![C']\!] \prec C''$ by Lemma 12.

: Case $\lfloor \mathsf{C}|_{\mathsf{Ctx}} \rceil$ and **Case** $\lfloor \mathsf{C}|_{\mathsf{Par}} \rceil$

proved by the definition of EPP and the induction hypothesis.

: Case $\lfloor \mathsf{C}|_{\mathsf{Eq}} \rceil$

We can write the derivation

$$\frac{\mathcal{R} \in \{ \equiv_{\mathsf{C}}, \simeq_{\mathsf{C}} \} \quad C_1 \,\mathcal{R}\, C_1' \quad D, C_1' \xrightarrow{\beta} D', C_2' \quad C_2' \,\mathcal{R}\, C_2}{D, C_1 \quad \xrightarrow{\beta} \quad D', C_2}\ \lfloor \mathsf{C}|_{\mathsf{Eq}} \rceil$$

For $\mathcal{R} = \equiv_{\mathsf{C}}$, proved by the definition of EPP, Lemma 10, and the induction hypothesis.
For $\mathcal{R} = \simeq_{\mathsf{C}}$, proved by the definition of EPP, Lemma 9, and the induction hypothesis.

□

Proof (Soundness). Proof by induction on the structure of C.

: **Case** $C = k : \mathsf{p}[\mathtt{A}].e \longrightarrow \mathsf{q}[\mathtt{B}].o(x); C_c$
From the definition of EPP we have

$$[\![C]\!] \equiv_{\mathsf{C}} k : \mathsf{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o; [\![C_c]\!]_{\mathsf{p}} \mid k : \mathtt{A} \longrightarrow \mathsf{q}[\mathtt{B}].o(x); [\![C_c]\!]_{\mathsf{q}} \mid \prod_{\mathsf{r} \,\in\, \mathbf{fp}(C)} [\![C_c]\!]_{\mathsf{r}} \mid \prod_{l} \left(\bigsqcup_{\mathsf{s} \,\in\, \lfloor C \rfloor_l} [\![C]\!]_{\mathsf{s}} \right)$$

we proceed by subcases on the last applied rule in the derivation of $D, [\![C]\!] \xrightarrow{\beta} D', C''$.

: **Case** $\lfloor \mathsf{C} | \mathsf{Send} \rceil$

Divided into subcases whether $\beta = k : \mathsf{p}[\mathtt{A}] \longrightarrow \mathtt{B}.o$ holds or not.

: **Case** $\beta = k : \mathsf{p}[\mathtt{A}] \longrightarrow \mathtt{B}.o$
$D, [\![C]\!]$ reduces to D', C'' with rules $\lfloor \mathsf{C} | \mathsf{Par} \rceil$, $\lfloor \mathsf{C} | \mathsf{Eq} \rceil$, ending with rule $\lfloor \mathsf{C} | \mathsf{Send} \rceil$ such that

$$C'' = [\![C_c]\!]_{\mathsf{p}} \mid k : \mathtt{A} \longrightarrow \mathsf{q}[\mathtt{B}].o(x); [\![C_c]\!]_{\mathsf{q}} \mid \prod_{\mathsf{r} \,\in\, \mathbf{fp}(C) \setminus \{\mathsf{p},\mathsf{q}\}} [\![C_c]\!]_{\mathsf{r}} \mid \prod_{l} \left(\bigsqcup_{\mathsf{s} \,\in\, \lfloor C \rfloor_l} [\![C]\!]_{\mathsf{s}} \right)$$

D, C mimics $D, [\![C]\!]$ with rules $\lfloor \mathsf{C} | \mathsf{Eq} \rceil$ and $\lfloor \mathsf{C} | \mathsf{Send} \rceil$ for which $D, C \xrightarrow{\beta} D'', C'$, $D' = D''$ by rule $\lfloor \mathsf{D} | \mathsf{Send} \rceil$,

$$[\![C']\!] \equiv_{\mathsf{C}} [\![C_c]\!]_{\mathsf{p}} \mid k : \mathtt{A} \longrightarrow \mathsf{q}[\mathtt{B}].o(x); [\![C_c]\!]_{\mathsf{q}} \mid \prod_{\mathsf{r} \,\in\, \mathbf{fp}(C) \setminus \{\mathsf{p},\mathsf{q}\}} [\![C_c]\!]_{\mathsf{r}} \mid \prod_{l} \left(\bigsqcup_{\mathsf{s} \,\in\, \lfloor C \rfloor_l} [\![C]\!]_{\mathsf{s}} \right)$$

and $[\![C']\!] \prec C''$.

: **Case** $\beta \neq k : \mathsf{p}[\mathtt{A}] \longrightarrow \mathtt{B}.o$

In this case D, C can mimic $D, [\![C]\!]$ with the application of rules $\lfloor \mathsf{C} | \mathsf{Eq} \rceil$, $\lfloor \mathsf{C} | \mathsf{Par} \rceil$, and $\lfloor \mathsf{C} | \mathsf{Send} \rceil$ and the thesis follows by the induction hypothesis.

: **Case** $\lfloor \mathsf{C} | \mathsf{Recv} \rceil$, $\lfloor \mathsf{C} | \mathsf{PStart} \rceil$, or $\lfloor \mathsf{C} | \mathsf{Cond} \rceil$
In this case $D, [\![C]\!]$ reduces with rules $\lfloor \mathsf{C} | \mathsf{Eq} \rceil$, $\lfloor \mathsf{C} | \mathsf{Par} \rceil$, and respectively ends the derivation with either $\lfloor \mathsf{C} | \mathsf{Recv} \rceil$, $\lfloor \mathsf{C} | \mathsf{PStart} \rceil$, or $\lfloor \mathsf{C} | \mathsf{Cond} \rceil$, i.e., some process $\mathsf{r} \in \mathbf{fp}(C)$ (p and q included) either receives a message, starts a new session with some service processes, or reduces to some branch. D, C can mimic $D, [\![C]\!]$ applying rules $\lfloor \mathsf{C} | \mathsf{Eq} \rceil$, $\lfloor \mathsf{C} | \mathsf{Par} \rceil$ and terminates the derivation with either rules $\lfloor \mathsf{C} | \mathsf{Recv} \rceil$, $\lfloor \mathsf{C} | \mathsf{PStart} \rceil$ (or $\lfloor \mathsf{C} | \mathsf{Start} \rceil$, depending on the form of C) or $\lfloor \mathsf{C} | \mathsf{Cond} \rceil$. The thesis follows by the induction hypothesis.

: **Case** $C = k : \mathsf{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o; C_c$

Similar to case $C = k : \mathsf{p}[\mathtt{A}].e \longrightarrow \mathsf{q}[\mathtt{B}].o(x); C_c$.

: **Case** $C = k : \mathtt{A} \longrightarrow \mathsf{q}[\mathtt{B}].\{o_i(x_i); C_i\}_{i \in I}$

From the definition of EPP we have

$$[\![C]\!] \equiv_{\mathsf{C}} k : \mathtt{A} \longrightarrow \mathsf{q}[\mathtt{B}].\{o_i(x_i); [\![C_i]\!]_{\mathsf{q}}\}_{i \in I} \mid \prod_{i \,\in\, I} \left(\bigsqcup_{\mathsf{p} \,\in\, \mathbf{fp}(C_i)} [\![C_i]\!]_{\mathsf{p}} \right) \mid \prod_{k} \left(\bigsqcup_{\mathsf{r} \,\in\, \lfloor C \rfloor_l} [\![C]\!]_{\mathsf{r}} \right)$$

we proceed by subcases on the last applied rule in the derivation of $D, [\![C]\!] \xrightarrow{\beta} D', C''$.

: Case $\lfloor \mathsf{C} | \mathsf{Recv} \rceil$

Divided into subcases whether $\beta = k\colon \mathtt{A}\rangle \mathtt{q}[\mathtt{B}].o_j$, $j \in I$ or not.

: Case $\beta = k\colon \mathtt{A}\rangle \mathtt{q}[\mathtt{B}].o_j$, $j \in I$

$D, [\![C]\!]$ reduces to D', C'' with rules $\lfloor \mathsf{C} | \mathsf{Par} \rceil$, $\lfloor \mathsf{C} | \mathsf{Eq} \rceil$, and terminates with rule $\lfloor \mathsf{C} | \mathsf{Recv} \rceil$ such that

$$C'' = [\![C_j]\!]_{\mathtt{q}} \mid \prod_{i \in I} \left(\bigsqcup_{\mathtt{p} \in \mathbf{fp}(C_i) \setminus \{\mathtt{q}\}} [\![C_i]\!]_{\mathtt{p}} \right) \mid \prod_{k} \left(\bigsqcup_{\mathtt{r} \in \lfloor C \rfloor_l} [\![C]\!]_{\mathtt{r}} \right)$$

D, C mimics $D, [\![C]\!]$ with rule $\lfloor \mathsf{C} | \mathsf{Recv} \rceil$ for which $D, C \xrightarrow{\beta} D'', C'$ where $D'' = D'$ by rule $\lfloor \mathsf{D} | \mathsf{Recv} \rceil$ and

$$[\![C']\!] = [\![C_j]\!]_{\mathtt{q}} \mid \prod_{\mathtt{p} \in \mathbf{fp}(C_j) \setminus \{\mathtt{q}\}} [\![C_j]\!]_{\mathtt{p}} \mid \prod_{k} \left(\bigsqcup_{\mathtt{r} \in \lfloor C_j \rfloor_l} [\![C_j]\!]_{\mathtt{r}} \right)$$

and $[\![C']\!] \prec C''$ by Lemma 12.

: Case $\beta \neq k\colon \mathtt{A}\rangle \mathtt{q}[\mathtt{B}].o_j$

For any β of this case D, C can mimic $D, [\![C]\!]$ with the application of rules $\lfloor \mathsf{C} | \mathsf{Eq} \rceil$ and $\lfloor \mathsf{C} | \mathsf{Par} \rceil$, terminating with rule $\lfloor \mathsf{C} | \mathsf{Recv} \rceil$ and the thesis follows by the induction hypothesis.

: Case $\lfloor \mathsf{C} | \mathsf{Send} \rceil$, $\lfloor \mathsf{C} | \mathsf{PStart} \rceil$, or $\lfloor \mathsf{C} | \mathsf{Cond} \rceil$

is similar to subcase **Case** $\lfloor \mathsf{C} | \mathsf{Recv} \rceil$, $\lfloor \mathsf{C} | \mathsf{PStart} \rceil$, or $\lfloor \mathsf{C} | \mathsf{Cond} \rceil$ of **Case** $C = k\colon \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{q}[\mathtt{B}].o(x); C_c$.

: Case $C = \mathtt{start}\ k : \mathtt{p}[\mathtt{A}] \leftrightarrow \widetilde{l.\mathtt{q}[\mathtt{B}]}; C_c$

$$[\![C]\!] \equiv_{\mathsf{C}} \mathtt{req}\ k : \mathtt{p}[\mathtt{A}] \leftrightarrow \widetilde{l.\mathtt{B}}; C_c \mid \prod_{\mathtt{r} \in \mathbf{fp}(C_c) \setminus \{\mathtt{p}\}} [\![C_c]\!]_{\mathtt{r}} \mid \prod_{l} \left(\bigsqcup_{\mathtt{s} \in \lfloor C \rfloor_l} [\![C]\!]_{\mathtt{s}} \right)$$

we proceed by subcases on the last applied rule in the derivation of $D, [\![C]\!] \xrightarrow{\beta} D, C''$.

: Case $\lfloor \mathsf{C} | \mathsf{PStart} \rceil$

$D, [\![C]\!]$ can reduce to D', C'' with a process $\mathtt{r}$ (including $\mathtt{p}$) that starts a new session with some service processes. D, C can reduce to D'', C' mimicking $D, [\![C]\!]$ by applying rules $\lfloor \mathsf{C} | \mathsf{Eq} \rceil$, $\lfloor \mathsf{C} | \mathsf{Par} \rceil$, terminating with either rule $\lfloor \mathsf{C} | \mathsf{PStart} \rceil$ or $\lfloor \mathsf{C} | \mathsf{Start} \rceil$.

: Case $\lfloor \mathsf{C} | \mathsf{Send} \rceil$, $\lfloor \mathsf{C} | \mathsf{Recv} \rceil$, and $\lfloor \mathsf{C} | \mathsf{Cond} \rceil$

are similar to the corresponding proof for the previous cases.

: Case $C = \mathtt{if}\ \mathtt{p}.e\ \{C_1\}\ \mathtt{else}\ \{C_2\}$

From the definition of EPP we have

$$[\![C]\!] \equiv_{\mathsf{C}} \mathtt{if}\ \mathtt{p}.e\ \{[\![C_1]\!]_{\mathtt{p}}\}\ \mathtt{else}\ \{[\![C_2]\!]_{\mathtt{p}}\} \mid \prod_{\mathtt{q} \in \mathbf{fp}(C_1) \cup \mathbf{fp}(C_2) \setminus \{\mathtt{p}\}} [\![C_1]\!]_{\mathtt{q}} \sqcup [\![C_2]\!]_{\mathtt{q}} \mid \mid \prod_{l} \left(\bigsqcup_{\mathtt{r} \in \lfloor C \rfloor_l} [\![C]\!]_{\mathtt{r}} \right)$$

we proceed by subcases on the derivation of $D, [\![C]\!] \xrightarrow{\beta} D', C''$.

: Case $\lfloor \mathsf{C} | \mathsf{Cond} \rceil$
$D, [\![C]\!]$ can reduce to D', C'' with:
: Case $\beta = \tau @ \mathtt{p}$
that reduces to a branch. D, C can mimic $D, [\![C]\!]$ applying rules $\lfloor \mathsf{C} | \mathsf{Eq} \rceil$, $\lfloor \mathsf{C} | \mathsf{Par} \rceil$, and terminating the derivation with rule $\lfloor \mathsf{C} | \mathsf{Cond} \rceil$. The case is proved by Lemma 12.
: Case $\beta = \tau @ \mathtt{r},\ \mathtt{r} \neq \mathtt{p}$
where process $\mathtt{r}$ reduced to a branch. The case follows the proof of the previous case and the thesis follows by the induction hypothesis.
: Case $\lfloor \mathsf{C} | \mathsf{Recv} \rceil$, $\lfloor \mathsf{C} | \mathsf{Send} \rceil$, $\lfloor \mathsf{C} | \mathsf{PStart} \rceil$
are similar to the corresponding proof for the previous cases.

: Case $C = \mathtt{req}\ k : \mathtt{p}[\mathtt{A}] \leftrightarrow \widetilde{l.\mathtt{B}}; C_c$
Case not allowed by the hypothesis that $D, [\![C]\!] \xrightarrow{\beta} D, C''$.

: Case $C = \mathtt{acc}\ k : \widetilde{l.\mathtt{q}[\mathtt{B}]}; C_c$
Case not allowed by the hypothesis that $D, [\![C]\!] \xrightarrow{\beta} D, C''$.

: Case $C = \mathtt{def}\ X = C''\ \mathtt{in}\ C'$
proved by Lemma 10 and the induction hypothesis.

: Case $C = X$
Case not allowed by the hypothesis that C is well-sorted.

: Case $C = C_1 \mid C_2$

$[\![C]\!] \equiv_{\mathsf{C}} [\![C_1]\!] \mid [\![C_2]\!]$ by Lemma 11.
we proceed by subcases for n equal to the length of the derivation of $D, [\![C]\!] \xrightarrow{\beta} D', C''$

: Case $n = 1$
In this case the only applicable rule is $\lfloor \mathsf{C} | \mathsf{PStart} \rceil$ where, Since both $[\![C_1]\!]$ and $[\![C_2]\!]$ reduce, we can infer, let

$$\widetilde{l.\mathtt{q}[\mathtt{B}]} = l_1.\mathtt{q}_1[\mathtt{B}_1], \ldots, l_i.\mathtt{q}_i[\mathtt{B}_i], l_{i+1}.\mathtt{q}_{i+1}[\mathtt{B}_{i+1}] \ldots, l_n.\mathtt{q}_n[\mathtt{B}_n]$$

that

$$C_1 \equiv_{\mathsf{C}} \mathtt{req}\ k : \mathtt{p}[\mathtt{A}] \leftrightarrow \widetilde{l.\mathtt{B}}; C_1^r \mid \prod_{j=1}^{i} \mathtt{acc}\ k : l_j.\mathtt{q}_j[\mathtt{B}_j]; C_1^j \mid C_c^1$$

$$C_2 \equiv_{\mathsf{C}} \prod_{j=i+1}^{n} \mathtt{acc}\ k : l_j.\mathtt{q}_j[\mathtt{B}_j]; C_2^j \mid C_c^2$$

and by the definition of EPP that

$$[\![C_1]\!] \equiv_{\mathsf{C}} \mathtt{req}\ k : \mathtt{p}[\mathtt{A}] \leftrightarrow \widetilde{l.\mathtt{B}}; [\![C_1^r]\!]_{\mathtt{p}} \mid \prod_{j=1}^{i} \mathtt{acc}\ k : l_j.\mathtt{q}_j[\mathtt{B}_j]; \left[\!\!\left[C_1^j \right]\!\!\right]_{\mathtt{q}_j} \mid [\![C_c^1]\!]$$

$$[\![C_2]\!] \equiv_{\mathsf{C}} \prod_{j=i+1}^{n} \mathtt{acc}\ k : l_j.\mathtt{q}_j[\mathtt{B}_j]; \left[\!\!\left[C_2^j \right]\!\!\right]_{\mathtt{q}_j} \mid [\![C_c^2]\!]$$

Observe that we can proceed without loss of generality as the symmetric case (with $\mathtt{p} \in \mathbf{fp}(C_2)$) follows the same structure.

$$\frac{\begin{array}{c} i \in \{1,\dots,n\} \qquad D\#k', \tilde{\mathsf{r}} \qquad \{\widetilde{l.\mathtt{B}}\} = \biguplus_i \{\widetilde{l_i.\mathtt{B}_i}\}_i \qquad \{\tilde{\mathsf{r}}\} = \bigcup_i \{\tilde{\mathsf{r}}_i\} \\ \delta = \mathtt{start}\ k' : \mathtt{p}[\mathtt{A}] \leftrightarrows \widetilde{l_1.\mathsf{r}_1[\mathtt{B}_1]}, \dots, \widetilde{l_n.\mathsf{r}_n[\mathtt{B}_n]} \qquad D, \delta \blacktriangleright D'' \end{array}}{D, [\![C_1]\!] \mid [\![C_2]\!] \quad \xrightarrow{\tau} \quad D'', C''} \lfloor \mathsf{C} |_{\mathsf{PStart}} \rceil$$

where

$$C'' \equiv_{\mathsf{C}} \left\{ \begin{array}{l} [\![C_1^r]\!]_{\mathsf{p}}\,[k'/k] \mid \left(\begin{array}{c} \prod_{j=1}^{i} \left[\!\left[C_1^j \right]\!\right]_{\mathsf{q}_j} \\ \mid \prod_{j=i+1}^{n} \left[\!\left[C_2^j \right]\!\right]_{\mathsf{q}_j} \end{array} \right) [k'/k][\tilde{\mathsf{r}}/\tilde{\mathsf{q}}] \\ \mid \left(\begin{array}{c} \prod_{j=1}^{i} \mathtt{acc}\ k : l_j.\mathsf{q}_j[\mathtt{B}_j]; \left[\!\left[C_1^j \right]\!\right]_{\mathsf{q}_j} \\ \mid \prod_{j=i+1}^{n} \mathtt{acc}\ k : l_j.\mathsf{q}_j[\mathtt{B}_j]; \left[\!\left[C_2^j \right]\!\right]_{\mathsf{q}_j} \end{array} \right) \mid [\![C_c^1]\!] \mid [\![C_c^2]\!] \end{array} \right.$$

Then D, C can mimic $D, [\![C]\!]$ applying rule $\lfloor \mathsf{C} |_{\mathsf{PStart}} \rceil$ with reduction

$$\frac{\begin{array}{c} i \in \{1,\dots,n\} \qquad D\#k'', \tilde{\mathsf{r}}' \qquad \{\widetilde{l.\mathtt{B}}\} = \biguplus_i \{\widetilde{l_i.\mathtt{B}_i}\}_i \qquad \{\tilde{\mathsf{r}}'\} = \bigcup_i \{\tilde{\mathsf{r}}'_i\} \\ \delta = \mathtt{start}\ k'' : \mathtt{p}[\mathtt{A}] \leftrightarrows \widetilde{l_1.\mathsf{r}'_1[\mathtt{B}_1]}, \dots, \widetilde{l_n.\mathsf{r}'_n[\mathtt{B}_n]} \qquad D, \delta \blacktriangleright D'' \end{array}}{D, C_1 \mid C_2 \quad \xrightarrow{\tau} \quad D'', C'} \lfloor \mathsf{C} |_{\mathsf{PStart}} \rceil$$

where

$$C' \equiv_{\mathsf{C}} C_1^r[k''/k] \mid \left(\begin{array}{c} \prod_{j=1}^{i} C_1^j \mid \\ \prod_{j=i+1}^{n} C_2^j \end{array} \right) [k''/k][\tilde{\mathsf{r}}'/\tilde{\mathsf{q}}] \mid \left(\begin{array}{c} \prod_{j=1}^{i} \mathtt{acc}\ k : l_j.\mathsf{q}_j[\mathtt{B}_j]; C_1^j \\ \mid \prod_{j=i+1}^{n} \mathtt{acc}\ k : l_j.\mathsf{q}_j[\mathtt{B}_j]; C_2^j \end{array} \right) \mid [\![C_c^1]\!] \mid [\![C_c^2]\!]$$

Following the structure of the second part of the proof of **Case** $\lfloor \mathsf{C} |_{\mathsf{Start}} \rceil$ for the proof of *Completeness* of Theorem 5, by α-renaming we have $D'' = D'$ and $[\![C']\!] \prec C''$.

: Case $n > 1$

For $n > 1$ we have a derivation similar to

$$\frac{\begin{array}{c} R \\ \vdots \quad \begin{array}{l} n-1 \text{ times, each either} \\ \lfloor \mathsf{C} |_{\mathsf{Par}} \rceil \text{ or } \lfloor \mathsf{C} |_{\mathsf{Eq}} \rceil \end{array} \end{array}}{D, [\![C_1]\!] \mid [\![C_2]\!] \quad \xrightarrow{\beta} \quad D', C_1'' \mid [\![C_2]\!]} \lfloor \mathsf{C} |_{\mathsf{Par}} \rceil$$

where R is the last applied rule, $R \in \{\lfloor \mathsf{C} |_{\mathsf{Send}} \rceil, \lfloor \mathsf{C} |_{\mathsf{Recv}} \rceil, \lfloor \mathsf{C} |_{\mathsf{PStart}} \rceil, \lfloor \mathsf{C} |_{\mathsf{Cond}} \rceil\}$. The thesis follows from the induction hypothesis.

The proof for the mirror case $D, [\![C_1]\!] \mid [\![C_2]\!] \quad \xrightarrow{\beta} \quad D', [\![C_1]\!] \mid C_2''$ follows the same structure.

: Case $C = \mathbf{0}$

trivial.

□

B.7. **Proof of Compilation from Frontend Choreographies to DCC Networks.** We first define some auxiliary results used in the proof of Theorem 6.

We provide some results on DCC variable substitution. We remind that the only bound names in *DCC* are the variables in (*accept*) terms (e.g., x in $!(x); B$). However, the following lemmas prove that renaming free variables with fresh names in processes (and, by extension, in services) preservers bisimilarity.

In the following, we abuse the notation for α-renaming to denote variable renaming in running processes. We define the variable renaming operator for DCC processes $P[x'/x]$.

Definition 20 (DCC Variable Renaming Operator)**.** Let $B \cdot t$ be a DCC process, then $(B \cdot t)[x'/x] = B[x'/x] \cdot t \triangleleft (\, x', x(t)\,) \triangleleft (\, x, \varnothing\,)$ where $B[x'/x]$ substitutes every occurrence of x with x'.

Lemma 13 (DCC Process Variable Renaming)**.** Let $\langle \mathfrak{B}, P \mid P_c, M\rangle_l$ be a DCC service where $P = B \cdot t$. Let $P' = P[x'/x]$ where x' is fresh in B. Then $\langle \mathfrak{B}, P \mid P_c, M\rangle_l \to \langle \mathfrak{B}, P'' \mid P_c, M\rangle_l \iff \langle \mathfrak{B}, P' \mid P_c, M\rangle_l \to \langle \mathfrak{B}, P''[x'/x] \mid P_c, M\rangle_l$.

Proof . The proof is by induction on the form of P. We report the most interesting cases. Below we consider $t' = t \triangleleft (\, x', x(t)\,) \triangleleft (\, x, \varnothing\,)$.

Case $P = o(y)\ \texttt{from}\ e; B' \cdot t$
The only applicable rule is $\lfloor^{\mathsf{DCC}}{}_{\mathsf{Recv}}\rceil$, hence we consider the interesting case in which M contains a message for the queue defined by e. In the other case the Lemma trivially holds as services cannot reduce on P and P'. The case unfolds on the combinations of whether i) $y \neq x$ and ii) expression e contains x. Below we consider the comprehensive case for $y = x$ and e that contains x. The proof of the other cases is either trivial or a slight modification of the reported one.

Since we assume we can apply rule $\lfloor^{\mathsf{DCC}}{}_{\mathsf{Recv}}\rceil$ we take $t_c = \mathbf{eval}(e, t)$ and $M(t_c) = (o, t') :: \tilde{m}$. From Definition 20 we have that $t_c = \mathbf{eval}(e[x'/x], t')$.

Meaningful reductions on P and P' are of the form $P \to B' \cdot t \triangleleft (\, x, t_m\,)$ and $P' \to B'[x'/x] \cdot t' \triangleleft (\, x', t_m\,)$ and the thesis follow by induction hypothesis.

Case $P = \sum_{i\in I} [o_i(x_i)\ \texttt{from}\ e]\,\{B_i\} \cdot t$
The only applicable rule on both P and P' is $\lfloor^{\mathsf{DCC}}{}_{\mathsf{Recv}}\rceil$. The most comprehensive case is for M that contains a message for operation o_j, $j \in I$ where $x_j = x$ and expression e contains x. The remainder of the proof follows that of the previous case.

Case $P = \texttt{if}\ e\ \{B_1\}\ \texttt{else}\ \{B_2\} \cdot t$
Trivial by Definition 20 for which $\mathbf{eval}(e, t) = \mathbf{eval}(e[x'/x], t')$.

Case $P = y = e; B \cdot t$
The only applicable rule on both P and P' is $\lfloor^{\mathsf{DCC}}{}_{\mathsf{Assign}}\rceil$. The most comprehensive case is for $y = x$ and expression e that contains x. The case is proved considering that, by Definition 20, it holds that $\mathbf{eval}(e, t) = \mathbf{eval}(e[x'/x], t')$.

Case $P = \texttt{def}\ X = B_1\ \texttt{in}\ B \cdot t$
The thesis follows from the application of rule $\lfloor^{\mathsf{DCC}}{}_{\mathsf{Ctx}}\rceil$ and the induction hypothesis.

Case $P = \nu\rangle x; B' \cdot t$
Let $t_c \notin M$. We have the reduction on rule $\lfloor^{\mathsf{DCC}}{}_{\mathsf{Newque}}\rceil$

$$S \to \left\langle \mathfrak{B}, B' \cdot t \triangleleft (\, x, t_c\,) \mid P_c, M[t_c \mapsto \varepsilon]\right\rangle_l$$

Let service S' be equal to S with P replaced with P'. S' can mimic the behaviour of S by taking the fresh value $t'_c = t_c$, obtaining the reduction

$$S' \to \left\langle \mathfrak{B}, B'[x'/x] \cdot t' \triangleleft (\, x', t'_c \,) \mid P_c, M[t'_c \mapsto \varepsilon] \right\rangle_l$$

The same holds if we let S' reduce and prove that S can mimic it.

Case $P = o@e_1(e_2)\ \texttt{to}\ e_3; B' \cdot t$

We consider the comprehensive case in which expressions e_1, e_2 and e_3 contain x. From Definition 20 we know that $\mathbf{eval}(e_1, t) = \mathbf{eval}(e_1[x'/x], t')$. Similarly the couples e_2 and $e_2[x'/x]$ and e_3 and $e_3[x'/x]$ enjoy the same property when evaluated respectively on t and t'.

We analyse the case in which P moves and $P[x'/x]$ mimics it. The other case, for $P[x'/x]$ that reduces and P that mimics it, follows the same structure.

$$\frac{\begin{array}{c} B = o@e_1(e_2)\ \texttt{to}\ e_3; B' \qquad \mathbf{eval}(e_1, t) = l \\ \mathbf{eval}(e_3, t) = t_c \qquad \mathbf{eval}(e_2, t) = t_m \qquad t_c \in \mathbf{dom}(M) \end{array}}{\langle \mathfrak{B},\ B \cdot t \mid P, M \rangle_l \quad \to \quad \langle \mathfrak{B},\ B' \cdot t \mid P, M[t_c \mapsto M(t_c) :: (o, t_m)] \rangle_l} \ \lfloor \mathsf{DCC}|_{\mathsf{InSend}} \rceil$$

and

$$\frac{\begin{array}{c} B[x'/x] = o@e_1[x'/x](e_2[x'/x])\ \texttt{to}\ e_3[x'/x]; B'[x'/x] \qquad \mathbf{eval}(e_1[x'/x], t') = l \\ \mathbf{eval}(e_3[x'/x], t') = t_c \qquad \mathbf{eval}(e_2[x'/x], t') = t_m \qquad t_c \in \mathbf{dom}(M) \end{array}}{\langle \mathfrak{B},\ B[x'/x] \cdot t' \mid P, M \rangle_l \quad \to \quad \langle \mathfrak{B},\ B'[x'/x] \cdot t' \mid P, M[t_c \mapsto M(t_c) :: (o, t_m)] \rangle_l} \ \lfloor \mathsf{DCC}|_{\mathsf{InSend}} \rceil$$

Case $?@e_1(e_2); B'' \cdot t$

We consider the comprehensive case where expressions e_1 and e_2 contain x. From Definition 20 we know that $\mathbf{eval}(e_1, t) = \mathbf{eval}(e_1[x'/x], t')$. Similarly e_2 and $e_2[x'/x]$ enjoy the same property when evaluated respectively on t and t'.

Below we describe the case in which P moves and $P[x'/x]$ mimics it. The other case, for $P[x'/x]$ that reduces and P that mimics it, follows the same structure. We assume the start behaviour $\mathfrak{B} = !(y); B'$.

$$\frac{B = ?@e_1(e_2); B'' \qquad Q = B' \cdot \varnothing \triangleleft (\, y, \mathbf{eval}(e_2, t) \,)}{\langle !(y); B',\ B \cdot t \mid P_c,\ M \rangle_l \quad \to \quad \langle !(y); B',\ Q \mid B'' \cdot t \mid P_c,\ M \rangle_l} \ \lfloor \mathsf{DCC}|_{\mathsf{InStart}} \rceil$$

and

$$\frac{B[x'/x] = ?@e_1[x'/x](e_2[x'/x]); B''[x'/x] \qquad Q = B' \cdot \varnothing \triangleleft (\, y, \mathbf{eval}(e_2[x'/x], t') \,)}{\langle !(y); B',\ B[x'/x] \cdot t' \mid P_c,\ M \rangle_l \quad \to \quad \langle !(y); B',\ Q \mid B''[x'/x] \cdot t' \mid P_c,\ M \rangle_l} \ \lfloor \mathsf{DCC}|_{\mathsf{InStart}} \rceil$$

□

Lemma 14 (DCC Network Variable Renaming)**.** Let S and S' be two DCC networks such that $S = \langle \mathfrak{B}, P \mid Q, M \rangle_l \mid S_*$ and $S' = \langle \mathfrak{B}, P[x'/x] \mid Q, M \rangle_l \mid S_*$ then

$$S \to \left\langle \mathfrak{B}, P' \mid Q', M' \right\rangle_l \mid S'_* \iff S' \to \left\langle \mathfrak{B}, P'[x'/x] \mid Q', M' \right\rangle_l \mid S'_*$$

.

Proof Sketch. The proof is by induction on the derivation of S. The main observation is that the most part of cases are already considered in Lemma 13. The cases not considered in Lemma 13 regard derivations on rules:

- $\lfloor^{\mathsf{DCC}}|_{\mathsf{Send}}\rceil$ whose proof follows the same steps of case $P = o@e_1(e_2)\ \texttt{to}\ e_3; B' \cdot t$ in Lemma 13;
- $\lfloor^{\mathsf{DCC}}|_{\mathsf{Start}}\rceil$ proved following the same steps of case $P = ?@e_1(e_2); B'' \cdot t$ in Lemma 13;
- $\lfloor^{\mathsf{DCC}}|_{\mathsf{Eq}}\rceil$ and $\lfloor^{\mathsf{DCC}}|_{\mathsf{Par}}\rceil$ where the thesis follows from the application of the induction hypothesis.

□

We report below the statement of Theorem 6, enriched with annotation on the transitions of D, C.

Theorem 6 *(Applied Choreographies)*
Let D, C be a Frontend choreography where C is projectable and $\Gamma \vdash D, C$ for some Γ. Then:

(1) (Completeness) $D, C \xrightarrow{\beta} D', C'$ implies

(a) $\boxed{\langle\!\langle D \rangle\!\rangle^{\Gamma}, [\![C]\!]}^{\Gamma} \to^{+} \boxed{\langle\!\langle D' \rangle\!\rangle^{\Gamma'}, C''}^{\Gamma'}$

(b) $[\![C']\!] \prec C''$

(c) for some Γ', $\Gamma' \vdash D', C'$

(2) (Soundness) $\boxed{\langle\!\langle D \rangle\!\rangle^{\Gamma}, [\![C]\!]}^{\Gamma} \to^{*} S$ implies

(a) $D, C \to^{*} D', C'$

(b) $S \to^{*} \boxed{\langle\!\langle D' \rangle\!\rangle^{\Gamma'}, C''}^{\Gamma'}$

(c) $[\![C']\!] \prec C''$

(d) for some Γ', $\Gamma' \vdash D', C'$

Proof (Completeness). We proceed by induction on the derivation of $D, C \xrightarrow{\beta} D', C'$. The general strategy is to:

- apply Theorem 4 from which, let $\mathbb{D} = \langle\!\langle D \rangle\!\rangle^{\Gamma}$, we have that $\mathbb{D}, C \xrightarrow{\beta} \mathbb{D}', C'$, $\mathbb{D}' = \langle\!\langle D \rangle\!\rangle^{\Gamma'}$;
- since C is *projectable*, we can always apply Theorem 5, from which, $D, [\![C]\!] \xrightarrow{\beta} D', C''$ and $[\![C']\!] \prec C''$;
- we compile the Backend Endpoint choreography $\mathbb{D}, [\![C]\!]$ into the DCC network $\boxed{\mathbb{D}, [\![C]\!]}^{\Gamma}$ and prove that we can reduce it in such a way that its reductum is $\equiv_{\mathtt{D}}$-equivalent to the compilation of the reductum $\boxed{\langle\!\langle D' \rangle\!\rangle^{\Gamma'}, C''}^{\Gamma'}$.

  Case $\lfloor^{\mathsf{C}}|_{\mathsf{Send}}\rceil$
  We know that
  – $[\![C]\!] \equiv_{\mathtt{C}} C_{\mathtt{p}} \mid C_c$ with $C_{\mathtt{p}} = k : \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o; C'_p$;
  – $D, [\![C]\!] \xrightarrow{\beta} D', C''$ with $\lfloor^{\mathsf{C}}|_{\mathsf{Send}}\rceil$ being the last applied rule, where $\beta = k : \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o$ and $C'' = C'_{\mathtt{p}} \mid C_c$;
  – let $\tilde{m} = D(k[\mathtt{A}\rangle\mathtt{B}])$ and $v = \mathbf{eval}(e, D(\mathtt{p}))$ we have, by rule $\lfloor^{\mathsf{D}}|_{\mathsf{Send}}\rceil$,

$$D' = D\big[\ k[\mathtt{A}\rangle\mathtt{B}] \mapsto \tilde{m} :: (o, v)\ \big]$$

  which, by Theorem 4, corresponds to $\mathbb{D}' = \mathbb{D}\big[\ l^* : t_c \mapsto \mathbb{D}(l^* : t_c) :: (o, t_m)\ \big]$ by $\lfloor^{\mathbb{P}}|_{\mathsf{Send}}\rceil$ where l^* is the location of the receiving process playing role $\mathtt{B}$ and t_c is the correlation

key used by the process playing $\mathtt{A}$ to send to the process playing role $\mathtt{B}$. The tree t_m corresponds to value v exchanged in rule $\lfloor^{\mathsf{D}}|_{\mathsf{Send}}\rceil$.

We have two cases, whether the receiving process q is in the same location of the sender p or not. Formally, let $\mathsf{p} \in \mathbb{D}(l)$ we consider the exhaustive cases:

Case $\mathsf{q} \in \mathbb{D}(l)$

From Definition 12 we have that $\boxed{\mathbb{D}, [\![C]\!]}^{\Gamma} \equiv_{\mathtt{D}} S \mid S_c$ where, let $t_{\mathsf{p}} = \mathbb{D}(\mathsf{p})$ and $M = \mathbb{D}|_l$

- $S = \left\langle \boxed{C_c|_l}^{\Gamma},\ P \mid Q, M \right\rangle_l$
- $P = o@\underline{k.B.l}(e)\ \mathtt{to}\ \underline{k.A.B}; \boxed{C'_{\mathsf{p}}}^{\Gamma} \cdot t_{\mathsf{p}}$
- $Q = \prod_{\mathsf{q} \in \mathbb{D}(l) \setminus \{\mathsf{p}\}} \boxed{C_c|_{\mathsf{q}}}^{\Gamma} \cdot \mathbb{D}(\mathsf{q})$
- $S_c = \prod_{l' \in \Gamma \setminus \{l\}} \left\langle \boxed{C_c|_{l'}}^{\Gamma},\ \prod_{\mathsf{r} \in \mathbb{D}(l')} \boxed{C_c|_{\mathsf{r}}}^{\Gamma} \cdot \mathbb{D}(\mathsf{s}), \mathbb{D}|_{l'} \right\rangle_{l'}$

In this case $\boxed{\mathbb{D}, [\![C]\!]}^{\Gamma}$ can mimic D, C applying rules $\lfloor^{\mathsf{DCC}}|_{\mathsf{Eq}}\rceil$, $\lfloor^{\mathsf{DCC}}|_{\mathsf{SPar}}\rceil$, and $\lfloor^{\mathsf{DCC}}|_{\mathsf{InSend}}\rceil$ where $S \mid S_c \to S' \mid S_c$ with $\lfloor^{\mathsf{DCC}}|_{\mathsf{SPar}}\rceil$ and $S \to S'$ with

$$\frac{P = o@\underline{k.B.l}(e)\ \mathtt{to}\ \underline{k.A.B}; \boxed{C'_{\mathsf{p}}}^{\Gamma} \cdot t_{\mathsf{p}} \qquad \mathbf{eval}(\underline{k.B.l}, t_{\mathsf{p}}) = l \qquad \mathbf{eval}(\underline{k.A.B}, t_{\mathsf{p}}) = t_c \qquad \mathbf{eval}(e, t_{\mathsf{p}}) = t_m \qquad t_c \in \mathbf{dom}(M)}{\langle \mathfrak{B},\ P \mid Q, M \rangle_l \quad \to \quad \langle \mathfrak{B},\ P' \mid Q, M[t_c \mapsto M(t_c) :: (o, t_m)] \rangle_l} \ \lfloor^{\mathsf{DCC}}|_{\mathsf{InSend}}\rceil$$

where $P' = \boxed{C'_{\mathsf{p}}}^{\Gamma} \cdot t_{\mathsf{p}}$. Since by Definition 12 l, t_c, and t_m result from the evaluation of the state of process p, $\mathbb{D}(\mathsf{p})$, we have that $M[t_c \mapsto M(t_c) :: (o, t_m)] = \mathbb{D}'|_l$.

This corresponds to the compilation of the reduction D', C', i.e,

$$\boxed{\langle\!\langle D' \rangle\!\rangle^{\Gamma'}, C''}^{\Gamma'} \equiv_{\mathtt{D}} \begin{array}{l} \left\langle \boxed{C_c|_l}^{\Gamma'}, \overbrace{\boxed{C'_{\mathsf{p}}}^{\Gamma'} \cdot \mathbb{D}'(\mathsf{p})}^{P'} \mid \overbrace{\prod_{\mathsf{q} \in \mathbb{D}'(l) \setminus \{\mathsf{p}\}} \boxed{C_c|_{\mathsf{q}}}^{\Gamma'} \cdot \mathbb{D}'(\mathsf{q})}^{Q}, \mathbb{D}'|_l \right\rangle_l \Bigg\} S' \\ \mid \\ \prod_{l' \in \Gamma' \setminus \{l\}} \left\langle \boxed{C_c|_{l'}}^{\Gamma'},\ \prod_{\mathsf{r} \in \mathbb{D}'(l')} \boxed{C_c|_{\mathsf{r}}}^{\Gamma'} \cdot \mathbb{D}'(\mathsf{r}), \mathbb{D}'|_{l'} \right\rangle_{l'} \Bigg\} S_c \end{array}$$

Where the changes in D' and Γ' affect only the compilation of the queue in $\mathbb{D}'|_l$ identified by t_c, while for all other terms $\boxdot^{\Gamma} = \boxdot^{\Gamma'}$ and $\mathbb{D}'|_{l'} = \mathbb{D}|_{l'}$.

Case $\mathsf{q} \notin \mathbb{D}(l)$

Similar to **Case** $\mathsf{q} \in \mathbb{D}(l)$ except the last applied rule in the reduction of $\boxed{\mathbb{D}, [\![C]\!]}^{\Gamma}$ is $\lfloor^{\mathsf{DCC}}|_{\mathsf{Send}}\rceil$.

Case $\lfloor^{\mathsf{C}}|_{\mathsf{Recv}}\rceil$

We know that

- $[\![C]\!] \equiv_{\mathtt{C}} C_{\mathsf{q}} \mid C_c$ with $C_{\mathsf{q}} = k : \mathtt{A} \longrightarrow \mathsf{q}[\mathtt{B}].\{o_i(x_i); C_i\}_{i \in I}$
- $D, [\![C]\!] \xrightarrow{\beta} D', C''$ with rule $\lfloor^{\mathsf{C}}|_{\mathsf{Recv}}\rceil$ where $\beta = k : \mathtt{A} \rangle \mathsf{q}[\mathtt{B}].o_j(x_j)$, $C' \equiv_{\mathtt{C}} C_j \mid C_c$. Let $\mathbb{D} = \langle\!\langle D \rangle\!\rangle^{\Gamma}$ and $D(k[\mathtt{A} \rangle \mathtt{B}]) = (o_j, v) :: \tilde{m}$, we have

$$D' = D\left[\ \mathsf{q} \mapsto D(\mathsf{q})[x_j \mapsto v]\ \right]\left[\ k[\mathtt{A} \rangle \mathtt{B}] \mapsto \tilde{m}\ \right]$$

By Theorem 4, let $\mathbb{D}(t_c : l^*) = (o_j, t_m) :: \tilde{m}^*$ we have

$$\mathbb{D}' = \mathbb{D}\big[\ \mathsf{q} \mapsto \mathbb{D}(\mathsf{q})[x_j \to t_m]\big]\big[\ l^* : t_c \mapsto \tilde{m}^*\ \big]$$

by $[\mathbb{P}|_{\mathsf{Recv}}]$ where l^* is the location of the receiving process playing role $\mathtt{B}$ and t_c is the correlation key used by the process playing $\mathtt{A}$ to send to the process playing role $\mathtt{B}$. The tree t_m corresponds to the encoding of value v in the queue.

Let $\mathsf{q}@l \in \Gamma$, $t_\mathsf{q} = \mathbb{D}(\mathsf{q})$, and $M = \mathbb{D}|_l$, from Definition 12 we have $\boxed{\mathbb{D}, [\![C]\!]}^\Gamma \equiv_\mathtt{D} S \mid S_c$ where

- $S = \Big\langle \boxed{C_c|_l}^\Gamma, Q \mid R, M \Big\rangle_l$
- $Q = \sum_{i\in I} [o_i(x_i)\ \mathtt{from}\ \underline{k.A.B}]\ \{\ \boxed{C_i}^\Gamma\ \} \cdot t_\mathsf{q}$
- $R = \prod\limits_{\mathsf{r}\in\mathbb{D}(l)\setminus\{\mathsf{q}\}} \boxed{C_c|_\mathsf{r}}^\Gamma \cdot \mathbb{D}(\mathsf{p})$
- $S_c = \prod\limits_{l'\ \in\ \Gamma\setminus\{l\}} \left\langle \boxed{C_c|_{l'}}^\Gamma, \prod\limits_{\mathsf{s}\in\mathbb{D}(l')} \boxed{C_c|_\mathsf{s}}^\Gamma \cdot t_\mathsf{s},\ \mathbb{D}|_{l'} \right\rangle_{l'}$

In this case $\boxed{\mathbb{D}, [\![C]\!]}^\Gamma$ can mimic D, C applying rules $\lfloor^{\mathsf{DCC}}|_{\mathsf{Eq}}\rceil$, $\lfloor^{\mathsf{DCC}}|_{\mathsf{SPar}}\rceil$, and $\lfloor^{\mathsf{DCC}}|_{\mathsf{Recv}}\rceil$.

$$\dfrac{\dfrac{Q = \sum_{i\in I} [o_i(x_i)\ \mathtt{from}\ \underline{k.A.B}]\ \{\ \boxed{C_i}^\Gamma\ \} \cdot t_\mathsf{q} \qquad j \in I \qquad t_c = \mathbf{eval}(e, t_\mathsf{q}) \qquad M(t_c) = (o_j, t_m) :: \tilde{m}^*}{\Big\langle \boxed{C_c|_l}^\Gamma, Q \mid R, M \Big\rangle_l \ \to\ \Big\langle \boxed{C_c|_l}^\Gamma, \boxed{C_j}^\Gamma \cdot t_\mathsf{q} \triangleleft (x_j, t_m) \mid R,\ M[t_c \mapsto \tilde{m}^*] \Big\rangle_l}\ \lfloor^{\mathsf{DCC}}|_{\mathsf{Recv}}\rceil}{S \mid S_c \ \to\ S' \mid S_c}\ \lfloor^{\mathsf{DCC}}|_{\mathsf{SPar}}\rceil$$

Where $S' = \Big\langle \boxed{C_c|_l}^\Gamma, \boxed{C_j}^\Gamma \cdot t_\mathsf{q} \triangleleft (x_j, t_m) \mid R,\ M[t_c \mapsto \tilde{m}^*] \Big\rangle_l$. Let $t'_\mathsf{q} = t_\mathsf{q} \triangleleft (x_j, t_m)$, $Q' = \boxed{C_j}^\Gamma \cdot t'_\mathsf{q}$, and $M' = M[t_c \mapsto \tilde{m}]$.

Since by Definition 12 t_c and t_m respectively result from the evaluation of the state of process q, $\mathbb{D}(\mathsf{q})$ and the encoding of value v, we have that $M' = \mathbb{D}'|_l$ and $t'_\mathsf{q} = \mathbb{D}'(\mathsf{q})$.

This corresponds to the compilation of the reduction D', C'', i.e,

$$\boxed{\langle\!\langle D' \rangle\!\rangle^{\Gamma'}, C''}^{\Gamma'} \equiv_\mathtt{D} \overbrace{\left\langle \boxed{C_c|_l}^{\Gamma'}, \overbrace{\boxed{C_j|_\mathsf{q}}^{\Gamma'} \cdot t'_\mathsf{q}}^{Q'} \mid \overbrace{\prod_{\mathsf{r}\ \in\ \mathbb{D}'(l)\setminus\{\mathsf{q}\}} \boxed{C_c|_\mathsf{r}}^{\Gamma'} \cdot \mathbb{D}'(\mathsf{r})}^{R}, M' \right\rangle_l}^{S'} \mid \underbrace{\prod_{l'\ \in\ \Gamma\setminus\{l\}} \left\langle \boxed{C_c|_{l'}}^{\Gamma'}, \prod_{\mathsf{s}\ \in\ D(l')} \boxed{C_c|_\mathsf{s}}^{\Gamma'} \cdot t_\mathsf{s},\ \mathbb{D}'|_l \right\rangle_{l'}}_{S_c}$$

Where the changes in D' and Γ' affect only the compilation of the queue in $\mathbb{D}'|_l$ identified by t_c and the state of q; while for all other terms $\boxdot^\Gamma = \boxdot^{\Gamma'}$ and $\mathbb{D}'|_{l'} = \mathbb{D}|_{l'}$.

Case $\lfloor^{\mathsf{C}}|_{\mathsf{PStart}}\rceil$

We know that

- $[\![C]\!] \equiv_\mathtt{C} C_r \mid C_a \mid C_c$ where, let $\tilde{l}\colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{B}}\rangle \in \Gamma$
- $C_r = \mathtt{req}\ k : \mathsf{p}[\mathtt{A}] \leftrightarrows \widetilde{l.\mathtt{B}}; C'_r$
- let $l_1.\mathtt{B}_1, \ldots, l_n.\mathtt{B}_n = \widetilde{l.\mathtt{B}}$, $C_a = \prod\limits_{i=1}^{n} \mathtt{acc}\ k : l_i.\mathsf{q}_i[\mathtt{B}_i]; C_{\mathsf{q}_i}$

We can apply rules $\lfloor^{\mathsf{C}}|_{\mathsf{Par}}\rceil$ and $\lfloor^{\mathsf{C}}|_{\mathsf{Eq}}\rceil$ and lastly rule $\lfloor^{\mathsf{C}}|_{\mathsf{PStart}}\rceil$ such that

$$\dfrac{\begin{array}{c} i \in \{1, \ldots, n\} \qquad D\#k', \tilde{\mathsf{r}} \qquad \{\widetilde{l.\mathtt{B}}\} = \biguplus_i \{\widetilde{l_i.\mathtt{B}_i}\}_i \qquad \{\tilde{\mathsf{r}}\} = \bigcup_i \{\tilde{\mathsf{r}}_i\} \\ \delta = \mathtt{start}\ k' : \mathsf{p}[\mathtt{A}] \leftrightarrows \widetilde{l_1.\mathsf{r}_1[\mathtt{B}_1]}, \ldots, \widetilde{l_n.\mathsf{r}_n[\mathtt{B}_n]} \qquad D, \delta \blacktriangleright D' \end{array}}{D, C_r \mid C_a \to D',\ C'_r[k'/k] \mid \prod_i \left(C'_{\mathsf{q}_i}[k'/k][\mathsf{r}_i/\mathsf{q}_i] \right) \mid C_a}\ \lfloor^{\mathsf{C}}|_{\mathsf{PStart}}\rceil$$

and

$$D, C_r \mid C_a \mid C_c \quad \xrightarrow{\tau} \quad D', C'_r[k'/k] \mid \prod_i \left(C_{\mathsf{q}_i}[k'/k][\mathsf{r}_i/\mathsf{q}_i] \right) \mid C_a \mid C_c$$

thus $C'' = C'_r[k'/k] \mid \prod_i \left(C_{\mathsf{q}_i}[k'/k][\mathsf{r}_i/\mathsf{q}_i] \right) \mid C_a \mid C_c$
We can find $\Gamma' = \Gamma, \mathbf{init}(k', (\mathsf{p}[\mathtt{A}], \widetilde{\mathsf{q}[\mathtt{B}]}), G)$ and $\Gamma' \vdash D', C'$.

Remark 1. We have two cases for, let $\mathsf{p}@l \in \Gamma$, whether $l \in \{\tilde{l}\}$ or not. For a clearer treatment of the case we proceed considering that $l \notin \{\tilde{l}\}$ (i.e., no service process is created in the same location — service — of the requester p). The other case follows the same structure of $l \notin \{\tilde{l}\}$ although the service located at l has $\boxed{C_a|_l}^\Gamma$ as start behaviour and $\boxed{\mathbb{D}, [\![C]\!]}^\Gamma$ applies rule $[^{\mathsf{DCC}}|_{\mathsf{InStart}}]$ in place of the $[^{\mathsf{DCC}}|_{\mathsf{Start}}]$ for starting the DCC process located at l.

Henceforth we proceed analysing the case for $l \notin \{\tilde{l}\}$.

From Definition 12 we have, let $\mathbb{D}^* = \langle\!\langle D' \rangle\!\rangle^{\Gamma'}$ and $M^* = \mathbb{D}^*|_l$ and $M^*_i = \mathbb{D}^*|_{l_i}$

$$\boxed{\mathbb{D}^*, C''}^{\Gamma'} = \left\langle \boxed{C_c|_l}^{\Gamma'}, P'' \mid R', M^* \right\rangle_l \mid \prod_{i=1}^{n} \langle Q''_i, Q^*_i \mid R'_{l_i}, M^*_i \rangle_{l_i} \mid S'_c$$

In the following, we use the abbreviation $t^*_{\mathsf{s}} = \mathbb{D}^*(\mathsf{s})$ for process s in $\mathbb{D}^*$.

- $P'' = \boxed{C'_r[k'/k]}^{\Gamma'} \cdot t^*_{\mathsf{p}}$
- $R' = \prod_{\mathsf{p}' \in \mathbb{D}^*(l) \setminus \{\mathsf{p}\}} \boxed{C_c|_{\mathsf{p}'}}^{\Gamma'} \cdot t^*_{\mathsf{p}'}$
- $Q''_i = \mathtt{accept}(k, \mathtt{B}_i, G\langle \mathtt{A} | \tilde{\mathtt{B}} | \tilde{\mathtt{B}} \rangle); \boxed{C_{\mathsf{q}_i}}^{\Gamma'}$
- $Q^*_i = \boxed{C_{\mathsf{q}_i}[k'/k][\mathsf{r}_i/\mathsf{q}_i]}^{\Gamma'} \cdot t^*_{q_i}$
- $R'_{l_i} = \prod_{\mathsf{s} \in \mathbb{D}^*(l_i)} \boxed{C_c|_{\mathsf{s}}}^{\Gamma'} \cdot t^*_{\mathsf{s}}$
- $S'_c = \prod_{l' \in \Gamma \setminus \{l, \tilde{l}\}} \left\langle \boxed{C_c|_{l'}}^{\Gamma'}, \prod_{\mathsf{s}' \in \mathbb{D}^*(l')} \boxed{C_c|_{\mathsf{s}'}}^{\Gamma'} \cdot t^*_{\mathsf{s}'}, \mathbb{D}^*|_{l'} \right\rangle_{l'}$

From Theorem 4 we can apply rule $[^{\mathsf{P}}|_{\mathsf{Start}}]$ on $\mathbb{D}, [\![C]\!] \to \mathbb{D}^*, C''$ such that we know that

$$\underline{k'}(t^*_{\mathsf{p}}) = \underline{k'}(t^*_{\mathsf{q}_1}) = \ldots = \underline{k'}(t^*_{\mathsf{q}_n}) = t_{k'}$$

for some $t_{k'}$ session descriptor of session k'.
We proceed by proving that we can reduce $\boxed{\mathbb{D}, [\![C]\!]}^\Gamma \to^+ S$.
From Definition 12 we have, let $t_{\mathsf{p}} = \mathbb{D}(\mathsf{p})$, $M = \mathbb{D}|_l$, and $M_i = \mathbb{D}|_{l_i}$

$$\boxed{\mathbb{D}, [\![C]\!]}^\Gamma \equiv_{\mathtt{D}} \left\langle \boxed{C_c|_l}^\Gamma, P \mid R, M \right\rangle_l \mid \prod_{i=1}^{n} \langle Q_i, R_{l_i}, M_i \rangle_{l_i} \mid S_c$$

where

$$P = \mathtt{start}(\ k,\ (l.\mathtt{A}, \widetilde{l.\mathtt{B}})\ ); \boxed{C'_r}^\Gamma \cdot t_{\mathsf{p}} =$$

$$= \left(\begin{array}{l} \bigodot_{\underline{I} \in \{\mathtt{A}, \tilde{\mathtt{B}}\}} \underline{k.I.l} = l_{\mathtt{I}}\ ; \\ \bigodot_{\underline{I} \in \{\tilde{\mathtt{B}}\}} \left(\nu \rangle \underline{k.I.A}; ?@\underline{k.I.l}(\underline{k}); \mathit{sync}(\underline{k})\ \mathtt{from}\ \underline{k.I.A} \right); \\ \bigodot_{\underline{I} \in \{\tilde{\mathtt{B}}\}} \mathit{start}@\underline{k.I.l}(\underline{k})\ \mathtt{to}\ \underline{k.A.I}; \boxed{C'_r}^\Gamma \end{array} \right) \cdot t_{\mathsf{p}}$$

– $Q_i = \texttt{accept}(k, \mathtt{B}_i, G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{B}}\rangle); \boxed{C_{\mathtt{q}_i}}^{\Gamma} =$ $!(\underline{k}); \bigodot_{\underline{I}\in\{\mathtt{A},\tilde{\mathtt{B}}\}\setminus\{\mathtt{B}_i\}} \nu\rangle \underline{k.I.B_i}$; $sync@\underline{k.A.l}(\underline{k})$ to $\underline{k.B_i.A}$; $start(\underline{k})$ from $\underline{k.A.B_i}$; $\boxed{C_{\mathtt{q}_i}}^{\Gamma}$

– $R = \prod_{\mathtt{p}' \in \mathbb{D}(l)\setminus\{\mathtt{p}\}} \boxed{C_c|_{\mathtt{p}'}}^{\Gamma} \cdot t_{\mathtt{p}'}$

– $R_{l_i} = \prod_{\mathtt{s} \in \mathbb{D}(l_i)} \boxed{C_c|_{\mathtt{s}}}^{\Gamma} \cdot t_{\mathtt{s}}$

– $S_c = \prod_{l' \in \Gamma\setminus\{l,\tilde{l}\}} \left\langle \boxed{C_c|_{l'}}^{\Gamma}, \prod_{\mathtt{s}' \in \mathbb{D}(l')} \boxed{C_c|_{\mathtt{s}'}}^{\Gamma} \cdot t_{\mathtt{s}'}, \mathbb{D}|_{l'} \right\rangle_{l'}$

$\boxed{\mathbb{D}, [\![C]\!]}^{\Gamma}$ can mimic D, C with the following sequence of reductions. Note that we make use of renaming on (*accept*) terms in $Q_1, \ldots, Q_n$ and variable renaming on P (as of Definition 20) to align the evolution of $\boxed{\mathbb{D}, [\![C]\!]}^{\Gamma}$ with the evolution of D, C, in which k has been renamed with the fresh name k'. Since the renamed DCC network and the original one are bisimilar, as per Lemma 14, we can proceed to prove our results on the original DCC network using the DCC renamed network as a proxy.

Therefore we take $S_0^* \sim \boxed{\mathbb{D}, [\![C]\!]}^{\Gamma}$

$$S_0^* = \left\langle \boxed{C_c|_l}^{\Gamma}, P[\underline{k'}/\underline{k}] \mid R, M \right\rangle_l \mid \prod_{i=1}^{n} \langle Q_i[\underline{k'}/\underline{k}], R_{l_i}, M_i \rangle_{l_i} \mid S_c$$

$$S_0^* \to \xrightarrow{\lfloor \mathsf{DCC}|_{\mathsf{SEq}} \rceil \quad \lfloor \mathsf{DCC}|_{\mathsf{SPar}} \rceil \quad \lfloor \mathsf{DCC}|_{\mathsf{PPar}} \rceil \quad \lfloor \mathsf{DCC}|_{\mathsf{Assign}} \rceil} \Big\}\ (1)\ n+1 \text{ times}$$

$$\Big\{ \begin{array}{l} (2.1) \xrightarrow{\lfloor \mathsf{DCC}|_{\mathsf{SEq}} \rceil \quad \lfloor \mathsf{DCC}|_{\mathsf{SPar}} \rceil \quad \lfloor \mathsf{DCC}|_{\mathsf{Newque}} \rceil} \\ (2.2) \xrightarrow{\lfloor \mathsf{DCC}|_{\mathsf{SEq}} \rceil \quad \lfloor \mathsf{DCC}|_{\mathsf{SPar}} \rceil \quad \lfloor \mathsf{DCC}|_{\mathsf{Start}} \rceil} \\ (2.3) \xrightarrow{\lfloor \mathsf{DCC}|_{\mathsf{SEq}} \rceil \quad \lfloor \mathsf{DCC}|_{\mathsf{SPar}} \rceil \quad \lfloor \mathsf{DCC}|_{\mathsf{Newque}} \rceil} \Big\}\ n \text{ times} \\ (2.4) \xrightarrow{\lfloor \mathsf{DCC}|_{\mathsf{SEq}} \rceil \quad \lfloor \mathsf{DCC}|_{\mathsf{SPar}} \rceil \quad \lfloor \mathsf{DCC}|_{\mathsf{Send}} \rceil} \\ (2.5) \xrightarrow{\lfloor \mathsf{DCC}|_{\mathsf{SEq}} \rceil \quad \lfloor \mathsf{DCC}|_{\mathsf{SPar}} \rceil \quad \lfloor \mathsf{DCC}|_{\mathsf{Recv}} \rceil} \end{array} \Big\}\ (2)\ \text{n times} \to S_1^*$$

$$\Big\{ \begin{array}{l} (3.1) \xrightarrow{\lfloor \mathsf{DCC}|_{\mathsf{SEq}} \rceil \quad \lfloor \mathsf{DCC}|_{\mathsf{SPar}} \rceil \quad \lfloor \mathsf{DCC}|_{\mathsf{Send}} \rceil} \\ (3.2) \xrightarrow{\lfloor \mathsf{DCC}|_{\mathsf{SEq}} \rceil \quad \lfloor \mathsf{DCC}|_{\mathsf{SPar}} \rceil \quad \lfloor \mathsf{DCC}|_{\mathsf{Recv}} \rceil} \end{array} \Big\}\ (3)\ n \text{ times}$$

We briefly comment the numbered transitions.

– In (1) $P[k'/k]$ proceeds to store (for $n+1$ times, l plus $l_i, i \in \{1, \ldots, n\}$) the locations of all roles under $\underline{k'}$.

– In (2), for each location $l_i, i \in \{1, \ldots, n\}$ (for each service process):
  * P creates its receiving queue for the service process (2.1);
  * in (2.2) P synchronises with the service at location l_i starting ($\lfloor \mathsf{DCC}|_{\mathsf{Start}} \rceil$) a new service process;
  * in (2.3) the service process creates its own queues for all other roles in the session (hence n times);
  * in (2.4) the service process sends the correlation values to P;

$*$ finally P receives the message in (2.5).

– In (3) for each service process (n times) (3.1) the starter sends a message to the service process to start the session and (3.2) the addressee receives it.

Finally we have

$$S_1^* = \left\langle \boxed{C_c|_l}^{\Gamma} \mid P' \mid R, M' \right\rangle_l \mid \prod_{i=1}^{n} \langle Q_i[\underline{k'}/\underline{k}], Q_i' \mid R_{l_i}, M_i' \rangle_{l_i} \mid S_c$$

where

– $P' = \boxed{C_r'}^{\Gamma}[\underline{k'}/\underline{k}] \cdot t_p'$, and
– $Q_i' = \boxed{C_{\mathsf{q}_i}}^{\Gamma}[\underline{k'}/\underline{k}] \cdot t_{k'}$

From the transitions presented above we know that there exists $t'_{k'}$ such that $t'_{\mathsf{p}} = t_{\mathsf{p}} \triangleleft (\underline{k'}, t'_{k'})$, where $t'_{k'}$ is a session descriptor for session k' (i.e., it contains all the locations and correlation keys used by the processes in session k'). In this case, we take $t'_k = t_{k'}$ obtained from the derivation $\mathbb{D}, C \to \mathbb{D}^*, C'$.

Similarly, M' and $M'_1, \ldots, M'_n$ contain the necessary (empty) queues to support communication in session k'.

$$M' = M[\underline{k'.B_1.A}(t_{k'}) \mapsto \varepsilon] \ \ldots \ [\underline{k'.B_n.A}(t_{k'}) \mapsto \varepsilon]$$

and ($\varnothing$ being a totally undefined function on $Val \rightharpoonup \mathcal{M}$)

$$M_i = \varnothing \begin{array}{l} [\underline{k'.A.B_i}(t_k) \mapsto \varepsilon][\underline{k'.B_1.B_i}(t_k) \mapsto \varepsilon] \ \ldots \ [\underline{k'.B_{i-1}.B_i}(t_k) \mapsto \varepsilon] \ \ldots \\ \ldots \ [\underline{k'.B_{i+1}.B_i}(t_k) \mapsto \varepsilon] \ \ldots \ [\underline{k'.B_n.B_i}(t_k) \mapsto \varepsilon] \end{array}$$

We proceed with the proof taking $S \sim S_1^*$ as S is simply the renaming of $\underline{k'}$ to $\underline{k}$ on start behaviours $Q_i, i \in \{1, \ldots, n\}$ (trivially $Q_i[\underline{k'}/\underline{k}][\underline{k}/\underline{k'}] = Q_i$)

$$S = \left\langle \boxed{C_c|_l}^{\Gamma} \mid P' \mid R, M' \right\rangle_l \mid \prod_{i=1}^{n} \langle Q_i, Q_i' \mid R_{l_i}, M_i' \rangle_{l_i} \mid S_c$$

We now proceed to prove that $\boxed{\mathbb{D}, [\![C]\!]}^{\Gamma} \to^+ \boxed{\mathbb{D}^*, C''}^{\Gamma'}$, i.e. that $\boxed{\mathbb{D}^*, C''}^{\Gamma'} = S$ with $\Gamma' \vdash D', C'$.

We prove that

$$\overbrace{\left\langle \boxed{C_c|_l}^{\Gamma'}, P'' \mid R', M^* \right\rangle_l \mid \prod_{i=1}^{n} \langle Q_i'', Q_i^* \mid R'_{l_i}, M_i^* \rangle_{l_i} \mid S_c'}^{\boxed{\mathbb{D}^*, C''}^{\Gamma'}} = \overbrace{\left\langle \boxed{C_c|_l}^{\Gamma} \mid P' \mid R, M' \right\rangle_l \mid \prod_{i=1}^{n} \langle Q_i, Q_i' \mid R_{l_i}, M_i' \rangle_{l_i} \mid S_c}^{S}$$

– M^* and M' are equal and similarly M_i^* and M_i are pair-wise equal by construction and rule $[\mathbb{P}|_{\mathsf{Start}}]$;
– $\boxed{C_c|_l}^{\Gamma} = \boxed{C_c|_l}^{\Gamma'}$ as $\Gamma|_{\mathtt{locs}} = \Gamma'|_{\mathtt{locs}}$ by construction;
– $P'' = P'$ is proved by

$$\boxed{C_r'[k'/k]}^{\Gamma'} \cdot t_{\mathsf{p}}^* = \boxed{C_r'}^{\Gamma}[\underline{k'}/\underline{k}] \cdot t_p'$$

which holds as

i) $\boxed{C_r'[k'/k]}^{\Gamma'} = \boxed{C_r'}^{\Gamma}[\underline{k'}/\underline{k}]$ since
  (a) Γ' does not contain any new process used in C_r';

(b) by renaming, and Lemma 14.

ii) $t^*_{\mathsf{p}} = t'_{\mathsf{p}}$ by construction and rule $[\![\mathsf{P}|_{\mathsf{Start}}]\!]$.

– $Q'^*_i = Q'_i$ proved by

$$\boxed{C_{\mathsf{q}_i}[k'/k][\mathsf{r}_i/\mathsf{q}_i]}^{\Gamma'} \cdot t^*_{q_i} = \boxed{C_{\mathsf{q}_i}}^{\Gamma}[\underline{k'}/\underline{k}] \cdot t_{k'}$$

whose proof of equivalence follows that of $P'' = P'$, except that Γ' contains the location of the process (r_i) used in $C_{\mathsf{q}_i}[k'/k][\mathsf{r}_i/\mathsf{q}_i]$.

– $Q''_i = Q_i$ proved by

$$\mathtt{accept}(k, \mathtt{B}_i, G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{B}}\rangle); \boxed{C_{\mathsf{q}_i}}^{\Gamma'} = \mathtt{accept}(k, \mathtt{B}_i, G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{B}}\rangle); \boxed{C_{\mathsf{q}_i}}^{\Gamma}$$

which holds as $\boxed{C_{\mathsf{q}_i}}^{\Gamma'} = \boxed{C_{\mathsf{q}_i}}^{\Gamma}$ because Γ and Γ' contain the same service typings.

– $R' = R$ is proved by

$$\prod_{\mathsf{p}' \in \mathbb{D}^*(l)\setminus\{\mathsf{p}\}} \boxed{C_c|_{\mathsf{p}'}}^{\Gamma'} \cdot t^*_{\mathsf{p}'} = \prod_{\mathsf{p}' \in \mathbb{D}(l)\setminus\{\mathsf{p}\}} \boxed{C_c|_{\mathsf{p}'}}^{\Gamma} \cdot t_{\mathsf{p}'}$$

for which

i) $\boxed{C_c|_{\mathsf{p}'}}^{\Gamma'} = \boxed{C_c|_{\mathsf{p}'}}^{\Gamma}$ as Γ' does not contain any new process used in C_c.

ii) $t^*_{\mathsf{p}'} = t_{\mathsf{p}'}$ unchanged by the reductions of $\mathbb{D}, C$ and $\boxed{\mathbb{D}, [\![C]\!]}^{\Gamma}$.

– $R'_{l_i} = R_{l_i}$ whose proof follows that of $R' = R$.

– $S'_c = S_c$ following the proof of $\boxed{C_c|_l}^{\Gamma} = \boxed{C_c|_l}^{\Gamma'}$ and $R'_{l_i} = R_{l_i}$.

Case $[\![\mathsf{C}|_{\mathsf{Start}}]\!]$

While the original FC program reduces applying rule $[\![\mathsf{C}|_{\mathsf{Start}}]\!]$, the endpoint projection $D, [\![C]\!]$ will mimic it applying rule $[\![\mathsf{C}|_{\mathsf{PStart}}]\!]$, as per Theorem 5. Hence, to prove this case, we can follow the same proof of case $[\![\mathsf{C}|_{\mathsf{PStart}}]\!]$.

Case $[\![\mathsf{C}|_{\mathsf{Cond}}]\!]$

We have $[\![C]\!] = C_{\mathsf{p}} \mid C_c$ where $C_{\mathsf{p}} = \mathtt{if}\ \mathsf{p}.e\ \{C_1\}\ \mathtt{else}\ \{C_2\}$. Let $\mathsf{p}@l \in \Gamma$ and

– $t_{\mathsf{p}} = \mathbb{D}(\mathsf{p})$;

– $P = \mathtt{if}\ e\ \{\ \boxed{C_1}^{\Gamma}\ \}\ \mathtt{else}\ \{\ \boxed{C_2}^{\Gamma}\ \} \cdot t_{\mathsf{p}}$;

– $R = \prod_{\mathsf{r} \in \mathbb{D}(l)\setminus\{\mathsf{p}\}} \boxed{C_c|_{\mathsf{r}}}^{\Gamma} \cdot t_{\mathsf{r}}$

– $S_c = \prod_{l' \in \Gamma\setminus\{l\}} \left\langle \boxed{C_c|_{l'}}^{\Gamma}, \prod_{\mathsf{r} \in \mathbb{D}(l')} \boxed{C_c|_{\mathsf{r}}}^{\Gamma} \cdot t_{\mathsf{r}}, \mathbb{D}|_{l'} \right\rangle_{l'}$

From Definition 12 we have, let $M = \mathbb{D}|_l$

$$\boxed{\mathbb{D}, [\![C]\!]}^{\Gamma} \equiv_{\mathtt{D}} \left\langle \boxed{C_c|_l}^{\Gamma}, P \mid R, M \right\rangle_l \mid S_c$$

we reduce $\mathbb{D}, [\![C]\!]$ applying rules $[\![\mathsf{C}|_{\mathsf{Par}}]\!]$, $[\![\mathsf{C}|_{\mathsf{Eq}}]\!]$ and lastly rule $[\![\mathsf{C}|_{\mathsf{Cond}}]\!]$. We analyse only the case for $\mathbf{eval}(e, t_{\mathsf{p}}) = \mathtt{true}$ as the other case for $\mathbf{eval}(e, t_{\mathsf{p}}) = \mathtt{false}$ follows the same structure.

$$\mathbb{D}, [\![C]\!] \quad \xrightarrow{\tau} \quad \mathbb{D}', C''$$

and $C'' = C_1 \mid C_c$ and $\mathbb{D}' = \mathbb{D}$ by the definition of $[\![\mathsf{C}|_{\mathsf{Cond}}]\!]$. We can choose $\Gamma = \Gamma'$, for which it holds that $\Gamma \vdash D', C'$.

From Definition 12 we have

$$\boxed{\mathbb{D}', C''}^{\Gamma'} = \boxed{\mathbb{D}, C''}^{\Gamma} = \left\langle \boxed{C_c|_l}^{\Gamma}, \boxed{C_1}^{\Gamma} \cdot t_{\mathsf{p}} \mid R, M \right\rangle_l \mid S_c$$

$\boxed{\mathbb{D}, [\![C]\!]}^{\Gamma}$ can mimic D, C applying rules $\lfloor^{\mathsf{DCC}}|_{\mathsf{Eq}}\rceil$, $\lfloor^{\mathsf{DCC}}|_{\mathsf{SPar}}\rceil$, $\lfloor^{\mathsf{DCC}}|_{\mathsf{PPar}}\rceil$, and lastly $\lfloor^{\mathsf{DCC}}|_{\mathsf{Cond}}\rceil$ for which

$$\boxed{\mathbb{D}, [\![C]\!]}^{\Gamma} \to \left\langle \boxed{C_c|_l}^{\Gamma}, \boxed{C_1}^{\Gamma} \cdot t_{\mathsf{p}} \mid R, M \right\rangle_l \mid S_c$$

Case $\lfloor^{\mathsf{C}}|_{\mathsf{Ctx}}\rceil$

The thesis follows from the induction hypothesis as D, C applies rule $\lfloor^{\mathsf{C}}|_{\mathsf{Ctx}}\rceil$ and $\boxed{\mathbb{D}, [\![C]\!]}^{\Gamma}$ can mimic it with rule $\lfloor^{\mathsf{DCC}}|_{\mathsf{Ctx}}\rceil$.

Case $\lfloor^{\mathsf{C}}|_{\mathsf{Par}}\rceil$

The thesis follows from the induction hypothesis.

Case $\lfloor^{\mathsf{C}}|_{\mathsf{Eq}}\rceil$

The thesis follows from the induction hypothesis. Starting from any configuration of D, C, $\boxed{\mathbb{D}, [\![C]\!]}^{\Gamma}$ can always mimic the evolution of D, C when it applies rule $\lfloor^{\mathsf{C}}|_{\mathsf{Eq}}\rceil$: in both cases that $\mathcal{R} = \equiv$ or $\mathcal{R} = \simeq_{\mathsf{C}}$, $\boxed{\mathbb{D}, [\![C]\!]}^{\Gamma}$ can apply $\lfloor^{\mathsf{DCC}}|_{\mathsf{Eq}}\rceil$, $\lfloor^{\mathsf{DCC}}|_{\mathsf{SPar}}\rceil$, and $\lfloor^{\mathsf{DCC}}|_{\mathsf{PPar}}\rceil$ to mimic D, C.

□

$$\frac{t' = \mathbf{eval}(x,t)}{x = e\ ; B \cdot t \quad \xrightarrow{x} \quad B \cdot t \triangleleft (\, x, t' \,)} \ \lfloor \text{DCC}|_{\mathsf{Assign}} \rceil \qquad \frac{B \cdot t \xrightarrow{\lambda} B' \cdot t'}{\texttt{def}\ X = B_1\ \texttt{in}\ B \cdot t \quad \xrightarrow{\lambda} \quad \texttt{def}\ X = B_1\ \texttt{in}\ B' \cdot t'} \ \lfloor \text{DCC}|_{\mathsf{Ctx}} \rceil$$

$$\frac{i = 1 \text{ if } \mathbf{eval}(e,t) = \text{true}, i = 2 \text{ otherwise}}{\texttt{if}\ e\ \{B_1\}\ \texttt{else}\ \{B_2\} \cdot t \quad \xrightarrow{\tau} \quad B_i \cdot t} \ \lfloor \text{DCC}|_{\mathsf{Cond}} \rceil \qquad \frac{P \equiv_{\mathtt{D}} P_1 \mid P_2 \quad P_1 \xrightarrow{\lambda} P_1' \quad P_1' \mid P_2 \equiv_{\mathtt{D}} P'}{\langle \mathfrak{B},\ P,\ M \rangle_l \quad \xrightarrow{\lambda} \quad \langle \mathfrak{B},\ P',\ M \rangle_l} \ \lfloor \text{DCC}|_{\mathsf{PEq}} \rceil$$

$$\frac{B = \nu\rangle x; B \quad t_c \notin \mathbf{dom}(M) \quad M' = M[t_c \mapsto \varepsilon]}{\langle \mathfrak{B},\ B \cdot t \mid P,\ M \rangle_l \quad \xrightarrow{\nu\rangle x} \quad \langle \mathfrak{B},\ B \cdot t \triangleleft (\, x, t_c \,) \mid P,\ M' \rangle_l} \ \lfloor \text{DCC}|_{\mathsf{Newque}} \rceil$$

$$\frac{\begin{array}{c} B \in \{\ o_j(x_j)\ \texttt{from}\ e; B_j\ ,\ \sum_{i \in I} [o_i(x_i)\ \texttt{from}\ e]\ \{B_i\}\ \} \\ j \in I \qquad t_c = \mathbf{eval}(e,t) \qquad M(t_c) = (o_j, t_m) :: \tilde{m} \end{array}}{\langle \mathfrak{B},\ B \cdot t \mid P,\ M \rangle_l \quad \xrightarrow{o_j\ \texttt{from}\ e} \quad \langle \mathfrak{B},\ B_j \cdot t \triangleleft (\, x_j, t_m \,) \mid P,\ M[t_c \mapsto \tilde{m}] \rangle_l} \ \lfloor \text{DCC}|_{\mathsf{Recv}} \rceil$$

$$\frac{\begin{array}{c} B = o@e_1(e_2)\ \texttt{to}\ e_3; B' \qquad \mathbf{eval}(e_1,t) = l \\ \mathbf{eval}(e_3,t) = t_c \qquad \mathbf{eval}(e_2,t) = t_m \qquad t_c \in \mathbf{dom}(M) \end{array}}{\langle \mathfrak{B},\ B \cdot t \mid P, M \rangle_l \quad \xrightarrow{o\ \texttt{to}\ e_3} \quad \langle \mathfrak{B},\ B' \cdot t \mid P, M[t_c \mapsto M(t_c) :: (o, t_m)] \rangle_l} \ \lfloor \text{DCC}|_{\mathsf{InSend}} \rceil$$

$$\frac{B = ?@e_1(e_2); B'' \quad Q = B' \cdot \varnothing \triangleleft (\, x, \mathbf{eval}(e_2,t) \,)}{\langle !(x); B',\ B \cdot t \mid P,\ M \rangle_l \quad \xrightarrow{?(e_2)} \quad \langle !(x); B',\ Q \mid B'' \cdot t \mid P,\ M \rangle_l} \ \lfloor \text{DCC}|_{\mathsf{InStart}} \rceil$$

$$\frac{\begin{array}{c} B = o@e_1(e_2)\ \texttt{to}\ e_3; B'' \qquad \mathbf{eval}(e_1,t) = l' \qquad \mathbf{eval}(e_3,t) = t_c \\ \mathbf{eval}(e_2,t) = t_m \qquad t_c \in \mathbf{dom}(M') \qquad M'' = M'[t_c \mapsto M'(t_c) :: (o, t_m)] \end{array}}{\langle \mathfrak{B}, B \cdot t \mid P,\ M \rangle_l \mid \langle \mathfrak{B}', P',\ M' \rangle_{l'} \quad \xrightarrow{o\ \texttt{to}\ e_3} \quad \langle \mathfrak{B}, B'' \cdot t \mid P,\ M \rangle_l \mid \langle \mathfrak{B}', P', M'' \rangle_{l'}} \ \lfloor \text{DCC}|_{\mathsf{Send}} \rceil$$

$$\frac{B = ?@e_1(e_2); B'' \quad \mathfrak{B}' = !(x); B' \quad \mathbf{eval}(e_1,t) = l' \quad Q = B' \cdot \varnothing \triangleleft (\, x, \mathbf{eval}(e_2,t) \,)}{\langle \mathfrak{B},\ B \cdot t \mid P, M \rangle_l \mid \langle \mathfrak{B}',\ P',\ M' \rangle_{l'} \quad \xrightarrow{?(e_2)} \quad \langle \mathfrak{B},\ B'' \cdot t \mid P,\ M \rangle_l \mid \langle \mathfrak{B}', Q \mid P',\ M' \rangle_{l'}} \ \lfloor \text{DCC}|_{\mathsf{Start}} \rceil$$

$$\frac{S \xrightarrow{\lambda} S'}{S \mid S_1 \quad \xrightarrow{\lambda} \quad S' \mid S_1} \ \lfloor \text{DCC}|_{\mathsf{SPar}} \rceil \qquad \frac{S \equiv_{\mathtt{D}} S_1 \quad S_1 \xrightarrow{\lambda} S_1' \quad S_1' \equiv_{\mathtt{D}} S'}{S \quad \xrightarrow{\lambda} \quad S'} \ \lfloor \text{DCC}|_{\mathsf{SEq}} \rceil$$

Figure 28: Dynamic Correlation Calculus, annotated semantics.

Before proceeding with the proof of (Soundness) of Theorem 6, we extend the semantics of DCC by annotating its transitions with the variable paths (of the kind $x = \underline{x.y.z}$) on which DCC operations execute. We range over DCC transition labels with λ.

$$\lambda ::= x \mid \nu\rangle x \mid ?(x) \mid o\ \texttt{from}\ x \mid o\ \texttt{to}\ x \mid \tau$$

We report in Figure 28 the annotated semantics of DCC.

We also introduce two operators on sequences of DCC transition labels. Let $\lambda, \tilde{\lambda}$ be a sequence of DCC labels, the filtering of $\lambda, \tilde{\lambda}$ on k , written $(\lambda, \tilde{\lambda})\Big|_k$ is defined as

$$(\lambda, \tilde{\lambda})\Big|_k = \begin{cases} \lambda, (\tilde{\lambda}\Big|_k) & \text{if } \lambda \in \left\{ \begin{array}{c} \underline{k.x.y},\ \nu\rangle\underline{k.x.y},\ ?(\underline{k}), \\ \mathit{sync}\ \texttt{from}\ \underline{k.x.y},\ \mathit{sync}@\underline{k.x.y}, \\ \mathit{start}\ \texttt{from}\ \underline{k.x.y},\ \mathit{start}@\underline{k.x.y} \end{array} \right\} \\ \tilde{\lambda}\Big|_k & \text{otherwise} \end{cases}$$

Let $\lambda_1, \tilde{\lambda}_1$ and $\lambda_2, \tilde{\lambda}_2$ be two sequences of DCC labels, the complement of $\lambda_1, \tilde{\lambda}_1$ on $\lambda_2, \tilde{\lambda}_2$, written $(\lambda_1, \tilde{\lambda}_1) \setminus (\lambda_2, \tilde{\lambda}_2)$ is defined as

$$(\lambda_1, \varepsilon) \setminus (\lambda_2, \tilde{\lambda}_2) = \begin{cases} \varepsilon & \text{if } \lambda_1 = \lambda_2 \\ \lambda_1 & \text{otherwise} \end{cases}$$

$$(\lambda_1, \tilde{\lambda}_1) \setminus (\lambda_2, \tilde{\lambda}_2) = \begin{cases} \tilde{\lambda}_1 \setminus \tilde{\lambda}_2 & \text{if } \lambda_1 = \lambda_2 \\ \lambda_1, (\tilde{\lambda}_1 \setminus (\lambda_2, \tilde{\lambda}_2)) & \text{otherwise} \end{cases}$$

Below we state Lemma 15 that proves that, given a DCC system S and a sequence of reductions $\tilde{\lambda}$ for which $S \xrightarrow{\tilde{\lambda}} S'$, if the first action is the initiation of a session k, then we can reorder the execution of the subsequent actions in $\tilde{\lambda}$ such that we first execute all transitions related to the start of k and then all the remaining actions, obtaining the same final system S'.

Lemma 15 (DCC Start Permutation)**.** Let S be a composition of DCC services such that $S \xrightarrow{\tilde{\lambda}} S'$ where $\tilde{\lambda} = \underline{k.C.l}, \tilde{\lambda}'$, then let $\tilde{\lambda}_k = \tilde{\lambda}\Big|_k$ and $\tilde{\lambda}_* = \tilde{\lambda} \setminus \tilde{\lambda}_k$ we have $S \xrightarrow{\tilde{\lambda}_k} S_1$ and $S_1 \xrightarrow{\tilde{\lambda}_*} S'$.

Proof Sketch. The proof is by induction on the length of $\tilde{\lambda}$. The main intuition is that, since the first action is the start of the new session k, all other actions in $\tilde{\lambda}'$ either are related to the initiation of k or do not affect it. Hence, we can reorder the execution of actions in $\tilde{\lambda}$ such that first we execute all actions regarding the start of the session[5] contained in $\tilde{\lambda}_k$ and then all the other actions in $\tilde{\lambda}_*$. □

Next we state Lemma 16 that proves that, given

- a well-typed FC endpoint choreography D, C
- its DCC compilation S
- the DCC system S' that results from an arbitrary number of steps of reduction belonging to the start of a session k in S

we can execute the remaining steps of reduction in S' to complete the start of session k, obtaining the final system S'' and prove that S'' is the same DCC system as the one obtained from the compilation of D', C', the reductum of the source FC choreography D, C after the step of reduction to start session k.

Lemma 16 (DCC Start Completion)**.** Let $\Gamma \vdash D, C$, C a endpoint choreography

$$C = \texttt{req}\ k : \mathtt{p}[\mathtt{A}] \leftrightarrows l_1.[\mathtt{B}_1], \ldots, l_n.[\mathtt{B}_n]; C_r \mid \prod_{i=1}^{n} \texttt{acc}\ k : l_i.\mathtt{q}_i[\mathtt{B}_i]; C_{\mathtt{q}_i}$$

[5]Note, this does not imply nor require that λ contains all actions needed to start session k.

and $\boxed{\langle\!\langle D \rangle\!\rangle^{\Gamma}, C}^{\Gamma} = S$ such that $S \xrightarrow{\tilde{\lambda}} S'$ where $\tilde{\lambda}\Big|_k = \tilde{\lambda}$ then *i*) $S' \xrightarrow{\tilde{\lambda}'} S''$, *ii*) $D, C \to D', C'$, and *iii*) there exists some Γ' s.t. $\Gamma' \vdash D', C'$ and $\boxed{\langle\!\langle D' \rangle\!\rangle^{\Gamma'}, C'}^{\Gamma'} = S''$.

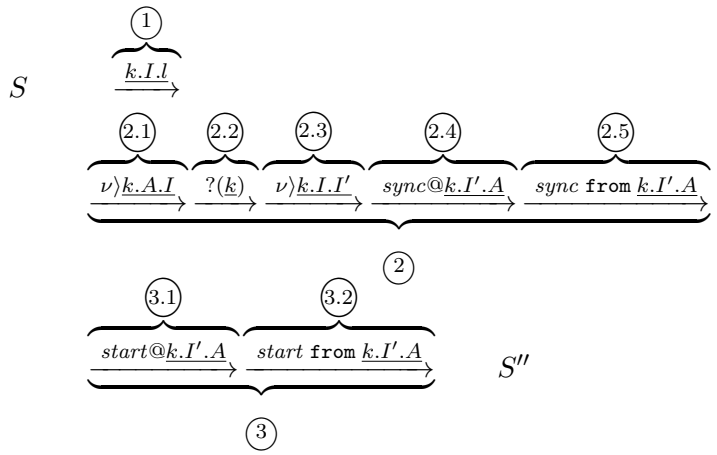
Proof . Proof by case analysis on the length of $\tilde{\lambda}$.

Let $\mathsf{p}@l \in \Gamma$. To proceed, we have two subcases whether $l \in \{l_1, \ldots, l_n\}$, i.e., whether one of the service processes is at the same location of p. Since the subcases follow the same structure, we detail only the proof for $l \notin \{l_1, \ldots, l_n\}$ which allows for a uniform treatment. In the other case, *i*) we should account for transitions on the same service of p with rules $\lfloor^{\mathsf{DCC}}|_{\mathsf{InStart}}\rceil$ and $\lfloor^{\mathsf{DCC}}|_{\mathsf{InSend}}\rceil$ and *ii*) we would have a newly created process in parallel with p in D, C and in the correspondent DCC system S''.

Provided n is the number of service processes involved in the start of the session k, from Definition 12 we can count the number of transitions needed to complete the start of a session. Indeed, given a D, C with

$$C = \mathtt{req}\ k : \mathsf{p}[\mathtt{A}] \leftrightarrows l_1.[\mathtt{B}_1], \ldots, l_n.[\mathtt{B}_n]; C_r \mid \prod_{i=1}^{n} \mathtt{acc}\ k : l_i.\mathsf{q}_i[\mathtt{B}_i]; C_{\mathsf{q}_i}$$

and $\boxed{\langle\!\langle D \rangle\!\rangle^{\Gamma}, C}^{\Gamma} = S$ then we can write the sequence of transitions of the compiled DCC system



and count the number of all the transitions to complete the start, let it be m, as the sum of:

(1): n+1 times, for $\underline{I} \in \{\mathtt{A}, \tilde{\mathtt{B}}\}$, with last Rule $\lfloor^{\mathsf{DCC}}|_{\mathsf{Assign}}\rceil$;

(2): n times, for $\underline{I} \in \tilde{\mathtt{B}}$:

- **(2.1):** reduces with last applied rule $\lfloor^{\mathsf{DCC}}|_{\mathsf{Newque}}\rceil$;
- **(2.2):** reduces with last applied rule $\lfloor^{\mathsf{DCC}}|_{\mathsf{Start}}\rceil$;
- **(2.3):** n times for $\underline{I'} \in \{\mathtt{A}, \tilde{\mathtt{B}}\} \setminus \{\underline{I}\}$, reduces with last applied rule $\lfloor^{\mathsf{DCC}}|_{\mathsf{Newque}}\rceil$;
- **(2.4):** reduces with last applied rule $\lfloor^{\mathsf{DCC}}|_{\mathsf{Send}}\rceil$;
- **(2.5):** reduces with last applied rule $\lfloor^{\mathsf{DCC}}|_{\mathsf{Recv}}\rceil$;

(3): n times, for $\underline{I} \in \tilde{\mathtt{B}}$:

- **(3.1):** reduces with last applied rule $\lfloor^{\mathsf{DCC}}|_{\mathsf{Send}}\rceil$;
- **(3.2):** reduces with last applied rule $\lfloor^{\mathsf{DCC}}|_{\mathsf{Recv}}\rceil$;

and $m = n^2 + 7n + 1$. We proceed unfolding the proof on the length of $\tilde{\lambda}$.

Case $|\{\tilde{\lambda}\}| = 1$

Since the cardinality of $\tilde{\lambda}$ is one and that from the premises we know that $\tilde{\lambda}$ contains only transitions belonging to the start of session k, we can infer that $\tilde{\lambda} = \underline{k.C.l}$ where $\mathtt{C} \in \{\mathtt{A}, \tilde{\mathtt{B}}\}$.

To prove the thesis we let S' do all the remaining transitions to start the session and show that D, C can mimic it. Let $\widetilde{l.\mathtt{B}} = l_1.\mathtt{B}_1, \ldots, l_n.\mathtt{B}_n$ and $\tilde{l} \colon G\langle \mathtt{A} | \tilde{\mathtt{B}} | \tilde{\mathtt{B}} \rangle \in \Gamma$.

From Definition 12 and Theorem 4 we have, let $\mathbb{D} = \langle\!\langle D \rangle\!\rangle^{\Gamma}$, $M = \mathbb{D}(l)$, $M_i = \mathbb{D}(l_i)$, and $t_{\mathtt{p}} = \mathbb{D}(\mathtt{p})$

$$\boxed{\langle\!\langle D \rangle\!\rangle^{\Gamma}, C}^{\Gamma} \equiv_{\mathtt{D}} \left\langle \boxed{C_c|_l}^{\Gamma}, P \mid R, M \right\rangle_l \mid \prod_{i=1}^{n} \langle Q_i, R_{l_i}, M_i \rangle_{l_i} \mid S_c$$

where

$P = \quad \mathtt{start}(\ k,\ (l.\mathtt{A}, \widetilde{l.\mathtt{B}})\ ); \boxed{C_r}^{\Gamma} \cdot t_{\mathtt{p}} =$

$$- \quad = \left(\begin{array}{l} \bigodot_{\underline{I} \in \{\mathtt{A}, \tilde{\mathtt{B}}\}} \underline{k.I.l} = l_{\mathtt{I}}\ ; \\ \bigodot_{\underline{I} \in \{\tilde{\mathtt{B}}\}} \left(\nu\langle \underline{k.I.A}; ?@\underline{k.I.l}(\underline{k}); \mathit{sync}(\underline{k})\ \mathtt{from}\ \underline{k.I.A} \right); \\ \bigodot_{\underline{I} \in \{\tilde{\mathtt{B}}\}} \mathit{start}@\underline{k.I.l}(\underline{k})\ \mathtt{to}\ \underline{k.A.I}; \end{array} \right) ; \boxed{C_r}^{\Gamma} \cdot t_{\mathtt{p}}$$

$$- \ Q_i = \mathtt{accept}(k, \mathtt{B}_i, G\langle \mathtt{A} | \tilde{\mathtt{B}} | \tilde{\mathtt{B}} \rangle); \boxed{C_{\mathtt{q}_i}}^{\Gamma} = \begin{array}{l} !(\underline{k});\ \bigodot_{\underline{I} \in \{\mathtt{A}, \tilde{\mathtt{B}}\} \setminus \{\mathtt{B}_i\}} \nu\langle \underline{k.I.B_i}\ ; \\ \mathit{sync}@\underline{k.A.l}(\underline{k})\ \mathtt{to}\ \underline{k.B_i.A}\ ; \\ \mathit{start}(\underline{k})\ \mathtt{from}\ \underline{k.A.B_i}\ ;\ \boxed{C_{\mathtt{q}_i}}^{\Gamma} \end{array}$$

$$- \ R = \prod_{\mathtt{p}' \in \mathbb{D}(l) \setminus \{\mathtt{p}\}} \boxed{C_c|_{\mathtt{p}'}}^{\Gamma} \cdot t_{\mathtt{p}'}$$

$$- \ R_{l_i} = \prod_{\mathtt{s} \in \mathbb{D}(l_i)} \boxed{C_c|_{\mathtt{s}}}^{\Gamma} \cdot t_{\mathtt{s}}$$

$$- \ S_c = \prod_{l' \in \Gamma \setminus \{l, \tilde{l}\}} \left\langle \boxed{C_c|_{l'}}^{\Gamma}, \prod_{\mathtt{s}' \in D(l')} \boxed{C_c|_{\mathtt{s}'}}^{\Gamma} \cdot t_{\mathtt{s}'}, \mathbb{D}|_{l'} \right\rangle_{l'}$$

The first transition, $\lambda = \underline{k.C.l}$ consumed the first assignment of location and assigned the location of role $\mathtt{C}$ to $\underline{k.C.l}$ in the state of the starter $t_{\mathtt{p}}$.

Let us suppose, without loss of generality, that $\mathtt{C} = \mathtt{A}$, then we have

$$P' = \left(\begin{array}{l} \bigodot_{\underline{I} \in \{\tilde{\mathtt{B}}\}} \underline{k.I.l} = l_{\mathtt{I}}\ ; \\ \bigodot_{\underline{I} \in \{\tilde{\mathtt{B}}\}} \left(\nu\langle \underline{k.I.A}; ?@\underline{k.I.l}(\underline{k}); \mathit{sync}(\underline{k})\ \mathtt{from}\ \underline{k.I.A} \right); \\ \bigodot_{\underline{I} \in \{\tilde{\mathtt{B}}\}} \mathit{start}@\underline{k.I.l}(\underline{k})\ \mathtt{to}\ \underline{k.A.I}; \end{array} \right) ; \boxed{C_r}^{\Gamma} \cdot t_{\mathtt{p}} \triangleleft (\ \underline{k.A.l}, l\ )$$

and $\boxed{\langle\!\langle D \rangle\!\rangle^{\Gamma}, C}^{\Gamma} \xrightarrow{\underline{k.A.l}} S'$ where

$$S' = \left\langle \boxed{C_c|_l}^{\Gamma}, P' \mid R, M \right\rangle_l \mid \prod_{i=1}^{n} \langle Q_i, R_{l_i}, M_i \rangle_{l_i} \mid S_c$$

Since in its reduction D, C renames the new session with a fresh name, we first rename session k, in P and the service processes Q_i, to k', which is fresh. We take

$$P'' = P'[k'/k] = \left(\begin{array}{l} \bigodot_{\underline{I}\in\{\tilde{\mathtt{B}}\}} \underline{k'.I.l} = l_1 \ ; \\ \bigodot_{\underline{I}\in\{\tilde{\mathtt{B}}\}} \big(\nu\rangle \underline{k'.I.A}; ?@\underline{k'.I.l}(\underline{k'}); ?@\underline{k'.I.l}(\underline{k'}); \big); \\ \bigodot_{\underline{I}\in\{\tilde{\mathtt{B}}\}} \mathit{sync}(\underline{k'}) \ \mathtt{from} \ \underline{k'.I.A} \end{array} \right) ; \boxed{C_r}^{\Gamma}[\underline{k'}/\underline{k}] \cdot t''_{\mathsf{p}}$$

where, let $t'_{\mathsf{p}} = t_{\mathsf{p}} \triangleleft (\, \underline{k.A.l}, l \,)$, $t''_{\mathsf{p}} = t'_{\mathsf{p}} \triangleleft (\, \underline{k'}, \underline{k}(t'_{\mathsf{p}}) \,) \triangleleft (\, \underline{k}, \varnothing \,)$.
We take

$$S_0^* = \left\langle \boxed{C_c|_l}^{\Gamma}, P'' \mid R, M \right\rangle_l \mid \prod_{i=1}^{n} \langle Q_i[\underline{k'}/\underline{k}], R_{l_i}, M_i \rangle_{l_i} \mid S_c$$

and by Lemma 14 we have $S_0^* \sim S'$.

Now we can proceed with the rest of the transitions of the start procedure as defined at the beginning of the proof, so that $S_0^* \to^+ S_1^*$. Finally we have

$$S_1^* \sim S'' = \left\langle \boxed{C_c|_l}^{\Gamma}, P''' \mid R, M' \right\rangle_l \mid \prod_{i=1}^{n} \langle Q_i[\underline{k'}/\underline{k}], Q'_i \mid R_{l_i}, M'_i \rangle_{l_i} \mid S_c$$

where $P''' = \boxed{C_r}^{\Gamma}[\underline{k'}/\underline{k}] \cdot t'_p$ and $Q'_i = \boxed{C_{\mathsf{q}_i}}^{\Gamma}[\underline{k'}/\underline{k}] \cdot t_{k'}$

From the transitions presented above we know that there exists $t'_{k'}$ such that $t'_{\mathsf{p}} = t_{\mathsf{p}} \triangleleft (\, \underline{k'}, t'_{k'} \,)$, where $t'_{k'}$ is a session descriptor for session k' (i.e., it contains all the locations and correlations keys used by the processes in session k').

We proceed by proving that D, C can mimic $\boxed{\langle\!\langle D \rangle\!\rangle^{\Gamma}, C}^{\Gamma}$.

We can apply rules $\lfloor^{\mathsf{C}}|_{\mathsf{Par}}\rceil$ and $\lfloor^{\mathsf{C}}|_{\mathsf{Eq}}\rceil$ and lastly rule $\lfloor^{\mathsf{C}}|_{\mathsf{PStart}}\rceil$ such that

$$\frac{\begin{array}{c} i \in \{1,\dots,n\} \qquad D \# k', \tilde{\mathsf{r}} \qquad \{\widetilde{l.\mathtt{B}}\} = \biguplus_i \{\widetilde{l_i.\mathtt{B}_i}\}_i \qquad \{\tilde{\mathsf{r}}\} = \bigcup_i \{\tilde{\mathsf{r}}_i\} \\ \delta = \mathtt{start}\ k' : \mathsf{p}[\mathtt{A}] \leftrightarrow \widetilde{l_1.\mathsf{r}_1[\mathtt{B}_1]}, \dots, \widetilde{l_n.\mathsf{r}_n[\mathtt{B}_n]} \qquad D, \delta \blacktriangleright D' \end{array}}{\begin{array}{l} D, \mathtt{req}\ k : \mathsf{p}[\mathtt{A}] \leftrightarrow \widetilde{l.\mathtt{B}}; C \mid \prod_i \big(\mathtt{acc}\ k : \widetilde{l_i.\mathsf{q}_i[\mathtt{B}_i]}; C_i\big) \quad \to \\ \qquad D',\ C[k'/k] \mid \prod_i \big(\ C_i[k'/k][\tilde{\mathsf{r}}_i/\tilde{\mathsf{q}}_i]\ \big) \mid \prod_i \big(\mathtt{acc}\ k : \widetilde{l_i.\mathsf{q}_i[\mathtt{B}_i]}; C_i\big) \end{array}} \lfloor^{\mathsf{C}}|_{\mathsf{PStart}}\rceil$$

and

$$D, C \mid C_c \quad \to \quad D', C_r[k'/k] \mid \prod_i \big(\ C_{\mathsf{q}_i}[k'/k][\mathsf{r}_i/\mathsf{q}_i]\ \big) \mid \prod_{i=1}^{n} \mathtt{acc}\ k : l_i.\mathsf{q}_i[\mathtt{B}_i]; C_{\mathsf{q}_i} \mid C_c$$

thus $C' = C_r[k'/k] \mid \prod_{i=1}^{n} \big(\ C_{\mathsf{q}_i}[k'/k][\mathsf{r}_i/\mathsf{q}_i]\ \big) \mid \prod_{i=1}^{n} \mathtt{acc}\ k : l_i.\mathsf{q}_i[\mathtt{B}_i]; C_{\mathsf{q}_i} \mid C_c$.

From the hypothesis we know that $\Gamma \vdash D, C$ and therefore that $\Gamma = \Gamma_1, \tilde{l} \colon G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{B}}\rangle$. We can find $\Gamma' = \Gamma, \mathbf{init}(k', (\mathsf{p}[\mathtt{A}], \widetilde{\mathsf{q}[\mathtt{B}]}), G)$ and $\Gamma' \vdash D', C'$.

Finally, we need to prove that $S_1^* = \boxed{\langle\!\langle D' \rangle\!\rangle^{\Gamma'}, C'}^{\Gamma'}$.

From Definition 12 we have

$$\boxed{\langle\!\langle D' \rangle\!\rangle^{\Gamma'}, C'}^{\Gamma'} = \left\langle \boxed{C_c|_l}^{\Gamma'}, P^* \mid R', M^* \right\rangle_l \mid \prod_{i=1}^{n} \langle Q''_i, Q^*_i \mid R'_{l_i}, M^*_i \rangle_{l_i} \mid S'_c$$

Let $\mathbb{D}^* = \langle\!\langle D' \rangle\!\rangle^{\Gamma'}$ we use the abbreviations $t^*_{\mathsf{s}} = \mathbb{D}^*(\mathsf{s})$, for s process in $\mathbb{D}^*$, and $M^* = \mathbb{D}|_l$, and $M^*_i = \mathbb{D}|_{l_i}$, in $\boxed{\mathbb{D}^*, C'}^{\Gamma'}$ we have

– $P^* = \boxed{C_r[k'/k]}^{\Gamma'} \cdot t^*_{\mathsf{p}}$
– $R' = \prod_{\mathsf{p}' \in \mathbb{D}^*(l)\setminus\{\mathsf{p}\}} \boxed{C_c|_{\mathsf{p}'}}^{\Gamma'} \cdot t^*_{\mathsf{p}'}$
– $Q''_i = \mathtt{accept}(k, \mathtt{B}_i, G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{B}}\rangle); \boxed{C_{\mathsf{q}_i}}^{\Gamma'}$
– $Q^*_i = \boxed{C_{\mathsf{q}_i}[k'/k][\mathsf{r}_i/\mathsf{q}_i]}^{\Gamma'} \cdot t^*_{q_i}$
– $R'_{l_i} = \prod_{\mathsf{s} \in \mathbb{D}^*(l_i)} \boxed{C_c|_{\mathsf{s}}}^{\Gamma'} \cdot t^*_{\mathsf{s}}$
– $S'_c = \prod_{l' \in \Gamma\setminus\{l,\tilde{l}\}} \left\langle \boxed{C_c|_{l'}}^{\Gamma'}, \prod_{\mathsf{s}' \in \mathbb{D}^*(l')} \boxed{C_c|_{\mathsf{s}'}}^{\Gamma'} \cdot t^*_{\mathsf{s}'}, \mathbb{D}^*|_{l'} \right\rangle_{l'}$

From Rule $[\mathsf{P}|_{\mathsf{Start}}]$ we know that

$$\underline{k'}(t^*_{\mathsf{p}}) = \underline{k'}(t^*_{\mathsf{q}_1}) = \ldots = \underline{k'}(t^*_{\mathsf{q}_n}) = t_{k'}$$

for some $t_{k'}$ session descriptor of session k'.

We prove the case by taking $t_{k'} = t'_{k'}$, $t'_{k'}$ obtained from the derivation of $\boxed{\langle\!\langle D \rangle\!\rangle^{\Gamma}, C}^{\Gamma}$ and $M^* = M'$ and $M^*_i = M'_i$, $i \in \{1, \ldots, n\}$.

Case $1 < |\{\tilde{\lambda}\}| < m - 1$

The case follows the same structure of the previous case. We rename k to k' on p and all the newly created service processes. Then we let the system complete all the transitions and prove that the reductum corresponds to the compilation of D', C'.

Case $|\{\tilde{\lambda}\}| = m$

Since $|\{\tilde{\lambda}\}| = m$ then $S = S'$ where S' has terminated all the transitions to start the session. Here we only have to rename k to k', as per Lemma 14, for all the involved processes, proving $S' = \boxed{\langle\!\langle D' \rangle\!\rangle^{\Gamma'}, C'}^{\Gamma'}$.

□

We now proceed to prove the *(Soundness)* of Theorem 6, restated here below to consider annotated transitions:

- *(Soundness)* $\boxed{D, C}^{\Gamma} \xrightarrow{\tilde{\lambda}} S$ implies *i)* $D, C \to^* D', C'$ and *ii)* $S \to^* \boxed{D', C'}^{\Gamma'}$ for some D',C', and Γ' such that *iii)* $\Gamma' \vdash D', C'$

In the following we use the shortcut

$$C_{start} = \mathtt{req}\ k : \mathsf{p}[\mathtt{A}] \triangleleft\triangleright l_1.[\mathtt{B}_1], \ldots, l_n.[\mathtt{B}_n]; C_r \mid \prod_{i=1}^{n} \mathtt{acc}\ k : l_i.\mathsf{q}_i[\mathtt{B}_i]; C_{\mathsf{q}_i}$$

Proof (Soundness).

We proceed by induction on the cardinality of $\tilde{\lambda}$. Then we consider sub-cases on the shape of C and the shape of $\tilde{\lambda}$.

Case $|\{\tilde{\lambda}\}| = 0$

Trivial, $\boxed{D, C}^{\Gamma} = S = \boxed{D', C'}^{\Gamma'}$, $D, C = D', C'$, and $\Gamma \vdash D', C'$.

Case $|\{\tilde{\lambda}\}| = 1$

Since the cardinality of $\tilde{\lambda}$ is one, we can directly consider the single annotated transition $\lambda = \tilde{\lambda}$. In the sub-cases of this case we omit to consider impossible cases for $\lambda = \nu\rangle x$ and $\lambda = ?(x)$ since these transitions (corresponding respectively to rules $\lfloor^{\mathsf{DCC}}|_{\mathsf{Newque}}\rceil$, and

$\lfloor^{\mathsf{DCC}}|_{\mathsf{InStart}}\rceil$ or $\lfloor^{\mathsf{DCC}}|_{\mathsf{Start}}\rceil$) can happen only within of a start session sequence (i.e., not at the first position).

In the following we use the abbreviation *follows* (#) to indicate that the case unfolds following the proof of **Case** # for the same subcase for λ, with the thesis following by applying the induction hypothesis.

Case $C = k\colon \mathtt{A} \longrightarrow \mathtt{q}[\mathtt{B}].\{o_i(x_i); C_i\}_{i\in I}; C_{\mathtt{q}} \mid C_c$

Case $\lambda = x$ *follows* ($C = C_{start} \mid C_c$).
Case $\lambda = o\ \mathtt{to}\ x$ *follows* ($C = k\colon \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o; C_{\mathtt{p}} \mid C_c$).
Case $\lambda = \tau$ *follows* ($C = \mathtt{if}\ \mathtt{p}.e\ \{C_1\}\ \mathtt{else}\ \{C_2\} \mid C_c$).
Case $\lambda = o\ \mathtt{from}\ x$
Since receptions in compiled DCC systems can only happen on correlating queues within sessions, without loss of generality we can assume that $\lambda = o\ \mathtt{from}\ \underline{k.A.B}$ where $o_j \notin \{start, sync\}$, indeed these operation names are reserved for session initiation and cannot appear as first (in this case, only) reduction of a compiled system.
Let $\mathtt{q}@l \in \Gamma$, from Definition 12 and Theorem 4 we have

$$\boxed{D, C}^{\Gamma} \equiv_{\mathtt{D}} \left\langle \boxed{C_c|_l}^{\Gamma}, Q \mid R, M \right\rangle_l \mid S_c$$

where, let $\mathbb{D} = \langle\!\langle D \rangle\!\rangle^{\Gamma}$, $M = \mathbb{D}|_l$ and $t_{\mathtt{q}} = \mathbb{D}(\mathtt{q})$,
– $Q = \sum_{i\in I} [o_i(x_i)\ \mathtt{from}\ \underline{k.A.B}]\ \{\boxed{C_i}^{\Gamma}\}\ \cdot t_{\mathtt{q}}$
– $R = \prod_{\mathtt{r}\ \in\ \mathbb{D}(l)\setminus\{\mathtt{q}\}} \boxed{C_c|_{\mathtt{r}}}^{\Gamma} \cdot t_{\mathtt{r}}$
– $S_c = \prod_{l'\in\Gamma\setminus\{l\}} \left\langle \boxed{C_c|_{l'}}^{\Gamma}, \prod_{\mathtt{s}\ \in\ D(l')} \boxed{C_c|_{\mathtt{s}}}^{\Gamma} \cdot t_{\mathtt{s}} \right\rangle_{l'}$
and we can apply rules $\lfloor^{\mathsf{DCC}}|_{\mathsf{Eq}}\rceil$, $\lfloor^{\mathsf{DCC}}|_{\mathsf{SPar}}\rceil$ and $\lfloor^{\mathsf{DCC}}|_{\mathsf{Recv}}\rceil$ such that, let $t_c = \mathbf{eval}(\underline{k.A.B}, t_{\mathtt{q}})$, $t_m = \mathbf{eval}(e, t_{\mathtt{q}})$, and $M(t_c) = (o_j, t_m) :: \tilde{m}$

$$\boxed{D, C}^{\Gamma} \xrightarrow{o_j\ \mathtt{from}\ \underline{k.A.B}} S$$

where

$$S = S' \mid S_c$$

and $S' = \left\langle \boxed{C_c|_l}^{\Gamma}, \boxed{C_j}^{\Gamma} \cdot t_{\mathtt{q}} \triangleleft (\, x_j, t_m \,) \mid R, M[t_c \mapsto \tilde{m}] \right\rangle_l$.
D, C can mimic $\boxed{D, C}^{\Gamma}$ with rules $\lfloor^{\mathsf{C}}|_{\mathsf{Eq}}\rceil$, $\lfloor^{\mathsf{C}}|_{\mathsf{Par}}\rceil$, and $\lfloor^{\mathsf{C}}|_{\mathsf{Recv}}\rceil$ for which

$$D, C \quad \rightarrow \quad D', C_{\mathtt{p}} \mid C_c$$

where, let $D(k[\mathtt{A}\rangle\mathtt{B}]) = (o_j, v_m) :: \tilde{m}'$, we have

$$D' = D[\mathtt{q} \mapsto t_{\mathtt{q}} \triangleleft (\, x_j, v_m \,)][k[\mathtt{A}\rangle\mathtt{B}] \mapsto \tilde{m}']$$

Since from the premises $\Gamma \vdash D, C$ then
$\Gamma = \Gamma_1, k[\mathtt{A}]\colon \&\mathtt{A}.\{o_i(U_i); T_i\}_{i\in I}, k[\mathtt{A}\rangle\mathtt{B}]\colon \&\mathtt{A}.o_j(U_j); T'$ and we can find
$\Gamma' = \Gamma_1, k[\mathtt{A}]\colon T_j, k[\mathtt{A}\rangle\mathtt{B}]\colon T'$ such that $\Gamma' \vdash D', C'$.
At the level of choreographies, since the changes in D' and Γ' and the related $\mathbb{D}' = \boxed{D'}^{\Gamma'}$ affect only the queue related to $\mathbb{D}'|_l$ and the state of $\mathtt{q}$, for all other terms $\boxdot^{\Gamma} = \boxdot^{\Gamma'}$ and $\mathbb{D}'|_{l'} = \mathbb{D}|_{l'}$.

Hence we can write $\boxed{D', C'}^{\Gamma'} = S'' \mid S_c$ where $S'' = S'$ by Theorem 4.

Case $C = k : \mathtt{p}[\mathtt{A}].e \longrightarrow \mathtt{B}.o; C_{\mathtt{p}} \mid C_c$

Case $\lambda = x$ *follows* ($C = C_{start} \mid C_c$).
Case $\lambda = o\ \mathtt{from}\ x$ *follows* ($C = k : \mathtt{A} \longrightarrow \mathtt{q}[\mathtt{B}].\{o_i(x_i); C_i\}_{i \in I}; C_{\mathtt{q}} \mid C_c$).
Case $\lambda = \tau$ *follows* ($C = \mathtt{if}\ \mathtt{p}.e\ \{C_1\}\ \mathtt{else}\ \{C_2\} \mid C_c$).
Case $\lambda = o\ \mathtt{to}\ x$

As for Case $C = k : \mathtt{A} \longrightarrow \mathtt{q}[\mathtt{B}].\{o_i(x_i); C_i\}_{i \in I}; C_{\mathtt{q}} \mid C_c$, we know that all send actions in DCC systems compiled from FC programs happen on a session-related queues, hence we can assume $\lambda = o@\underline{k.A.B}$. Also, we know that $o \notin \{start, sync\}$ for the reasons explained in Case $C = k : \mathtt{A} \longrightarrow \mathtt{q}[\mathtt{B}].\{o_i(x_i); C_i\}_{i \in I}; C_{\mathtt{q}} \mid C_c$.
From Theorem 4, let $\mathbb{D} = \langle\!\langle D \rangle\!\rangle^{\Gamma}$, $t_{\mathtt{p}} = \mathbb{D}(\mathtt{p})$, and $M = \mathbb{D}|_l$. Now we consider two cases for which, let $\mathtt{p}@l \in \Gamma$, whether the location of the receiving process (stored under path $\underline{k.B.l}$ in the state of $\mathtt{p}$) equals l, we either reduce the compiled DCC system by means of rule $\lfloor^{\mathsf{DCC}}|_{\mathsf{InSend}}\rceil$ or rule $\lfloor^{\mathsf{DCC}}|_{\mathsf{Send}}\rceil$. For brevity we just consider the case for $\lfloor^{\mathsf{DCC}}|_{\mathsf{InSend}}\rceil$ as the other case follows similarly.
Since $\lfloor^{\mathsf{DCC}}|_{\mathsf{InSend}}\rceil$ applies, we can infer that

$$\boxed{D, C}^{\Gamma} \equiv_{\mathtt{D}} \left\langle \boxed{C_c|_l}^{\Gamma}, P \mid Q \mid R, M \right\rangle_l \mid S_c$$

where
– $P = o@\underline{k.B.l}\ \mathtt{to}\ \underline{k.A.B}; \boxed{C_{\mathtt{p}}}^{\Gamma} \cdot t_{\mathtt{p}}$
– $Q = \boxed{C_c|_{\mathtt{q}}}^{\Gamma} \cdot t_{\mathtt{q}}$
– $R = \prod_{\mathtt{r} \in D(l) \setminus \{\mathtt{p},\mathtt{q}\}} \boxed{C_c|_{\mathtt{r}}}^{\Gamma} \cdot t_{\mathtt{r}}$
– $S_c = \prod_{l' \in \Gamma \setminus \{l\}} \left\langle \boxed{C_c|_{l'}}^{\Gamma}, \prod_{\mathtt{s} \in \mathbb{D}(l')} \boxed{C_c|_{\mathtt{s}}}^{\Gamma} \cdot t_{\mathtt{s}}, \mathbb{D}|_{l'} \right\rangle_{l'}$

Let $t_c = \mathbf{eval}(\underline{k.A.B}, t_{\mathtt{p}})$, $t_m = \mathbf{eval}(e, t_{\mathtt{p}})$, and $M(t_c) = \tilde{m}$

$$\boxed{D, C}^{\Gamma} \xrightarrow{o@\underline{k.A.B}} S$$

where

$$S = S' \mid S_c$$

and $S' = \left\langle \boxed{C_c|_l}^{\Gamma}, \boxed{C_{\mathtt{p}}}^{\Gamma} \cdot t_{\mathtt{p}} \mid \boxed{C_c|_{\mathtt{q}}}^{\Gamma} \cdot t_{\mathtt{q}} \mid R, M[t_c \mapsto \tilde{m} :: (o, t_m)] \right\rangle_l$
D, C can mimic $\boxed{D, C}^{\Gamma}$ with rules $\lfloor^{\mathsf{C}}|_{\mathsf{Eq}}\rceil$, $\lfloor^{\mathsf{C}}|_{\mathsf{Par}}\rceil$, and $\lfloor^{\mathsf{C}}|_{\mathsf{Send}}\rceil$ for which

$$D, C \quad \to \quad D', C_{\mathtt{p}} \mid C_c$$

where, let $v_m = \mathbf{eval}(e, D(\mathtt{p}))$ and $\tilde{m}' = D(k[\mathtt{A}\rangle\mathtt{B}])$, $D' = D[k[\mathtt{A}\rangle\mathtt{B}] \mapsto \tilde{m}' :: (o, v_m)]$.
Since from the premises $\Gamma \vdash D, C$ then $\Gamma = \Gamma_1, k[\mathtt{A}] : \oplus\, \mathtt{B}.o(U); T, k[\mathtt{A}\rangle\mathtt{B}] : T'$ and we can find $\Gamma' = \Gamma_1, k[\mathtt{A}] : T, k[\mathtt{A}\rangle\mathtt{B}] : T'; \&\mathtt{A}.o(U)$ such that $\Gamma' \vdash D', C'$.
At the level of choreographies, since the changes in D' and Γ' and the related $\mathbb{D}' = \boxed{D'}^{\Gamma'}$ affect only the queue related to $\mathbb{D}'|_l$, for all other terms $\boxdot^{\Gamma} = \boxdot^{\Gamma'}$ and $\mathbb{D}'|_{l'} = \mathbb{D}|_{l'}$.

Hence we can write $\boxed{D', C'}^{\Gamma'} = S'' \mid S_c$ where $S'' = S'$ by Theorem 4.

Case $C = C_{start} \mid C_c$

Case $\lambda = o\,\texttt{from}\,x$ *follows* $(C = k{:}\texttt{A} \longrightarrow \texttt{q}[\texttt{B}].\{o_i(x_i); C_i\}_{i\in I}; C_\texttt{q} \mid C_c)$.
Case $\lambda = o\,\texttt{to}\,x$ *follows* $(C = k{:}\texttt{p}[\texttt{A}].e \longrightarrow \texttt{B}.o; C_\texttt{p} \mid C_c)$
Case $\lambda = \tau$ *follows* $(C = \texttt{if p}.e\ \{C_1\}\ \texttt{else}\ \{C_2\} \mid C_c)$.
Case $\lambda = x$
From Definition 12 we know that assignments in DCC systems that are compiled from FC programs appear only within the starting of a session. In this case, since $\tilde{\lambda}$ contains only one action which corresponds to the first reduction of the compiled DCC system, it must be the first assignment for the creation of the session descriptor for some session k' in C.
Let $\texttt{C} \in \{\texttt{A}, \tilde{\texttt{B}}\}$, we have two subcases whether $\tilde{\lambda} = \lambda = \underline{k.C.l}$ or $\tilde{\lambda} = \lambda = \underline{k''.C.l}$, i.e., whether we are starting session k or we are starting another session k''.

Case $\lambda = \underline{k.C.l}$
In this case $\boxed{D, C}^{\Gamma}$ is starting a new session on k. The case is proved applying Lemma 16.

Case $\lambda \neq \underline{k'.C.l}$
In this case we are starting a session on $k'' \neq k$. The case unfolds following the proof of case $C = C_{start}|C_c$ where C_c contains the endpoint choreographies for the starter process and the service processes for session k''. The thesis follows by applying the induction hypothesis.

Case $C = \texttt{if p}.e\ \{C_1\}\ \texttt{else}\ \{C_2\} \mid C_c$

Case $\lambda = x$ *follows* $(C = C_{start} \mid C_c)$.
Case $\lambda = o\,\texttt{from}\,x$ *follows* $(C = k{:}\texttt{A} \longrightarrow \texttt{q}[\texttt{B}].\{o_i(x_i); C_i\}_{i\in I}; C_\texttt{q} \mid C_c)$.
Case $\lambda = o\,\texttt{to}\,x$ *follows* $(C = k{:}\texttt{p}[\texttt{A}].e \longrightarrow \texttt{B}.o; C_\texttt{p} \mid C_c)$.
Case $\lambda = \tau$

In this case, the label does not allow us to establish a correspondence between the considered shape of C and the actual reduction annotated by the label.
However, since τ labels only correspond to the reduction of conditionals, without loss of generality, we can consider here only the case where the reduction acts on the considered term. The other case follows the unfolding of this case for the term reduced by the action and the induction hypothesis.
Let $\texttt{p@}l \in \Gamma$. From Definition 12 we have

$$\boxed{D, C}^{\Gamma} \equiv_{\texttt{D}} \left\langle \boxed{C_c|_l}^{\Gamma}, P \mid R, M \right\rangle_l \mid S_c$$

where, let $\mathbb{D} = \langle\!\langle D \rangle\!\rangle^{\Gamma}$, $t_\texttt{p} = \mathbb{D}(\texttt{p})$, and $M = \mathbb{D}|_l$
– $P = \texttt{if p}.e\ \{\boxed{C_1}^{\Gamma}\}\ \texttt{else}\ \{\boxed{C_2}^{\Gamma}\} \cdot t_\texttt{p}$
– $R = \prod_{\texttt{r}\in\mathbb{D}(l)\setminus\{\texttt{p}\}} \boxed{C_c|_\texttt{r}}^{\Gamma} \cdot t_\texttt{r}$
– $S_c = \prod_{l'\in\Gamma\setminus\{l\}} \left\langle \boxed{C_c|_{l'}}^{\Gamma}, \prod_{\texttt{s}\in\mathbb{D}(l')} \boxed{C_c|_\texttt{s}}^{\Gamma} \cdot t_\texttt{s}, \mathbb{D}|_{l'} \right\rangle_{l'}$

The case unfolds into two cases, on whether $\textbf{eval}(e, \mathbb{D}(\texttt{p})) = \texttt{true}$. Here we proceed with the positive case. The other case follows the same structure.

We proceed considering that $\mathbf{eval}(e, \mathbb{D}(\mathsf{p})) = \texttt{true}$. $\boxed{D, C}^{\Gamma}$ reduces with rules $\lfloor^{\mathsf{DCC}}|_{\mathsf{Eq}}\rceil$, $\lfloor^{\mathsf{DCC}}|_{\mathsf{SPar}}\rceil$, $\lfloor^{\mathsf{DCC}}|_{\mathsf{Par}}\rceil$, and $\lfloor^{\mathsf{DCC}}|_{\mathsf{Cond}}\rceil$ such that

$$\boxed{D, C}^{\Gamma} \quad \rightarrow \quad \left\langle \boxed{C_c|_l}^{\Gamma}, \boxed{C_c|_{\mathsf{p}}}^{\Gamma} \cdot t_{\mathsf{p}} \mid R, M \right\rangle_l \mid S_c$$

where $S = \left\langle \boxed{C_c|_l}^{\Gamma}, \boxed{C_c|_l}^{\Gamma} \cdot t_{\mathsf{p}} \mid R, M \right\rangle_l \mid S_c$. D, C can mimic $\boxed{D, C}^{\Gamma}$ with rules $\lfloor^{\mathsf{C}}|_{\mathsf{Eq}}\rceil$, $\lfloor^{\mathsf{C}}|_{\mathsf{Par}}\rceil$, and $\lfloor^{\mathsf{C}}|_{\mathsf{Cond}}\rceil$ such that

$$D, C \quad \rightarrow \quad D, C_1 \mid C_c$$

We choose $\Gamma' = \Gamma$ for which it holds that $\Gamma \vdash D, C_1 \mid C_c$.

Finally, $\boxed{D, C_1 \mid C_c}^{\Gamma} = S$ by Definition 12.

Finally, **Case**s $C = C_1 \mid C_2$, $C = \texttt{def}\ X = C' \ \texttt{in}\ C_{\mathsf{p}} \mid C_c$, $C = X \mid C_c$ and $C = \mathbf{0} \mid C_c$ unfold applying the induction hypothesis on the respective sub-cases **Case** $\lambda = x$ *follows* ($C = C_{start} \mid C_c$), **Case** $\lambda = o\ \texttt{from}\ x$ *follows* ($C = k{:}\texttt{A} \longrightarrow \mathsf{q}[\texttt{B}].\{o_i(x_i); C_i\}_{i \in I}; C_{\mathsf{q}} \mid C_c$), **Case** $\lambda = o\ \texttt{to}\ x$ *follows* ($C = k{:}\ \mathsf{p}[\texttt{A}].e \longrightarrow \texttt{B}.o; C_{\mathsf{p}} \mid C_c$), **Case** $\lambda = \tau$ *follows* ($C = \texttt{if}\ \mathsf{p}.e\ \{C_1\}\ \texttt{else}\ \{C_2\} \mid C_c$).

Case $|\{\tilde{\lambda}\}| > 1$

The case unfolds considering λ as the first action in $\tilde{\lambda} = \lambda, \tilde{\lambda}'$. For any shape of C and label $\lambda \neq x$ we can i) apply the same steps followed in the related case for the same C with $|\{\tilde{\lambda}\}| = 1$, $\tilde{\lambda} = \lambda$ and ii) inductively unfold the case on the remaining part $\tilde{\lambda}'$.

For $\lambda = x$ and C of shape $C_{start} \mid C_c$ (the case for other shapes of C can be re-conducted to this case), let $x = \underline{k.A.l}$ (other cases for $x = \underline{k'.B.l}$ are similar) and the thesis follows by applying Lemma 15, Lemma 16 and the induction hypothesis on the remaining transitions in $\tilde{\lambda} \setminus \tilde{\lambda}\Big|_k$.

□