Choreographic Programming

Fabrizio Montesi

A PhD Dissertation
Presented to the Faculty of the IT University of Copenhagen
in Partial Fulfillment of the Requirements of the PhD Degree

Advisor

Marco Carbone	IT University of Copenhagen, Denmark

Evaluation Committee

Luís Caires	New University of Lisbon, Portugal
Ilaria Castellani	INRIA - Sophia Antipolis, France
Peter Sestoft	IT University of Copenhagen, Denmark

IT University of Copenhagen August 2013

Abstract

Choreographies are descriptions of distributed systems where the developer gives a global view of how messages are exchanged by endpoint nodes (endpoints for short), instead of separately defining the behaviour of each endpoint. They have a significant impact on the quality of software, as they offer a concise view of the message flows enacted by a system. For this reason, in the last decade choreographies have been employed in the development of programming languages, giving rise to a programming paradigm that in this dissertation we refer to as Choreographic Programming.

Recent formal investigations of choreographies show that they have potential as foundations for the development of safe distributed software. The key idea is that since choreographies abstract from the single input/output actions of endpoints, they avoid typical safety problems such as deadlocks and race conditions; the concrete implementation of each endpoint described in a choreography can then be automatically obtained by compilation, ensuring that such implementations are also safe by construction from the originating choreography. However, current formal models for choreographies do not deal with critical aspects of distributed programming, such as asynchrony, mobility, modularity, and multiparty sessions; it remains thus unclear whether choreographies can still guarantee safety when dealing with such nontrivial features.

This PhD dissertation argues for the usefulness of choreographic programming as a paradigm for the development of safe distributed systems. We proceed by investigating its foundations and application. To this aim we provide three main contributions.

The first contribution is the development of a formal model and type theory for choreographic programming that support asynchrony, mobility, modular development, and multiparty sessions. We prove that our model guarantees safety by mapping choreographies to distributed implementations in terms of a variant of the π-calculus, the reference model for mobile processes. Our translation preserves the expected safety properties of choreographies, among which freedom from deadlocks and race conditions.

The second contribution is the development of Linear Connection Logic (LCL), a formal logic that captures the reasoning behind choreographic programming. We show that LCL is a conservative extension of Linear Logic. We then develop a Curry-Howard correspondence between LCL and a calculus of choreographies, proving that: (i) proofs in LCL correspond to choreographies; and (ii) the transformations between proofs in LCL to proofs in Linear Logic and vice versa correspond to compiling choreography programs to π-calculus terms and vice versa. The latter result, known as round-trip development, contributes to the open problem of extracting choreographies from existing endpoint programs.

The third contribution is the implementation of a prototype programming framework for choreographic programming, called Chor. Chor provides an Integrated Development Environment (IDE) for programming with choreographies, equipped with a type checker for verifying that choreographies respect protocol specifications given as session types. Programs in Chor can be compiled to executable endpoint implementation in the Jolie programming language, a general-purpose language for distributed computing, which we extend to support the development of multiparty asynchronous sessions. We use Chor for evaluating choreographic programming against a series of use cases.

Acknowledgements

I would like to dedicate this thesis to all who supported me in my passion for research, and to whom I owe the success of finishing this body of work.

I will start by thanking my family, for encouraging me to pursue my ambitions and for giving me all the serenity I needed in the difficult moments, both personal and professional.

My first thanks go to my wife, Maja. She is the one who encouraged me to start my PhD studies in the first place. Before meeting her, I was inclined to end my studies after getting a BSc Degree; and now, here I am with a PhD Degree. Maja kindled my interest for basic research, and it did not take me much time to discover that it is for me an inexhaustible source of fun and satisfaction! Even though I am a very pragmatic person at work[1], I have a tendency for demanding too much of myself. Maja has been my beacon of emotional and intellectual wisdom, keeping my enthusiasm in balance so that I would enjoy my PhD time, rather than becoming a burnout because of it. Maja, thank you for your brilliance and your patience.

I thank my parents, Paolo and Irene, for always showing to me such incredible examples of sensitivity, fiery determination, and candour. While I am still just an amateur on these qualities compared to them, I have already lost count of how many times they have been useful to me. I thank my brother and sister, Marco and Dania, for all their enormous patience and support, both important aspects for being my big brother and big sister. Both have greatly influenced the way I work, respectively by inspiring my interest for industrial application and my positive attitude towards trying out new things. Thank you for all the happy moments that we share together. For this last important point, my thanks go also to their respective spouses, Sara and Roberto.

I would like now to thank the people I met during my three years of PhD research, to whom I am indebted for their help and interesting discussions.

I had the good fortune and priviledge of having an outstanding PhD supervisor, Marco Carbone. He has been, and still is, also a great colleague and a friend. Marco has not only a profound technical knowledge of the theoretical tools I worked with, but also an appreciation for the practical implications of formal models. Sharing this appreciation with him made me very motivated, and discussing challenging topics with Marco has always been a real pleasure.

I extend my gratitude to Nobuko Yoshida and all the members of the Mobility Reading Group at Imperial College London, for hosting me during my two-month visit there. Nobuko has been a great source of encouragement and support for me. I have been impressed by how much honest effort and planning she puts in her team at Imperial, which consequently influenced me in how I deal with teamwork.

I would also like to thank Carsten Schürmann, for sharing his deep knowledge on formal logics with me[2]. Formal logics provided me with an additional perspective for understanding the underlying structures I was dealing with in my work, which has been tremendously useful both technically and for making my results accessible to an entirely new research community for me.

[1] Or at least I think I am. Curiously, in contrast, outside of work I have my head in the clouds.

[2] "It is all about cut elimination".

I thank my reviewers, Luís Caires, Ilaria Castellani, and Peter Sestoft, for their useful comments on this PhD dissertation. It has been a pleasure to discuss my work with them, going from looking at the tiny details up to analysing where my results stand in the big picture of computer science. Such discussions have been really inspiring and supportive, and are already influencing my work.

I thank all the colleagues I met in the 4C corridor (including our visitors, and some people from 4D) at the IT University of Copenhagen. For me, 4C is like an oasis full of experts on different formal techniques for dealing with programming languages. There is a great drive for inclusion, not only social but also professional. My colleagues were so enthusiastic about their work and so open to sharing it, that I had the possibility to learn so many more different techniques in just three years than I imagined possible. A heartfelt thank you also for making my working days feel so light. Special thanks to Jesper Bengtson for the great dinners, to Ornela Dardha for making work more fun, to Søren Debois for his help with my (still present) struggle with the Danish language, to Saverio Giallorenzo for the new shiny Jolie website (and the salame and Nutella!), to Thomas Hildebrandt for his gigantic professional support, and to Raghava Rao Mukkamala for being a great office mate for three years.

Before going to the IT University of Copenhagen for my PhD studies, I worked on the research and development of languages for concurrent computation at the University of Bologna. I owe a lot to my colleagues and mentors there, who gave me many opportunities for learning about research and for carrying out my work.

My thanks to Gianluigi Zavattaro, for spending his time to explore opportunities for my PhD studies. Gianluigi was the one who initially put me in contact with the people at the IT University; after a few discussions with them I grew enthusiastic to apply for a PhD position there. Not only is Gianluigi technically excellent, but he also has the fine quality of being very open-minded in his research. Seeing his attitude towards new ideas has been really precious for me, especially when considering new theories and techniques during my PhD.

I extend my gratitude to Davide Sangiorgi, for providing me with many opportunities for extending my network and discussing my ideas with world-leading researchers. Working with Davide has been really motivating. I owe him the most his encouragement to bring my research to a higher level by engaging with larger research communities. I now continuously try to improve on this.

Thanks to Ivan Lanese, for the time we spent together on developing formal models for service-oriented computing. Most people start studying and working with formal models by looking at simple problems first. In my case, chance had it that I started from something that turned out to be not so trivial (self-modifying code for error handling in distributed systems). Thanks to Ivan, dealing with the formal details was fun instead of traumatic.

Last, but not least, among my colleagues from my times in Bologna, special thanks go to Claudio Guidi. Driven by our passion for technology, Claudio and I worked tirelessly together for bringing the results of our research to the industry, which has been an incredible experience. We share the enthusiasm for empowering developers with scientific tools, which makes working with him a real pleasure for me. Our adventure still continues in the development of the Jolie language and other projects.

I finally thank all my friends who I have met in my first three years in Denmark, for making my time in Copenhagen fantastic.

Thank you all. I had a lot of fun.

Fabrizio Montesi
Copenhagen, December 3, 2013

List of Publications

1. F. Montesi, C. Guidi, G. Zavattaro. Service-oriented Programming with Jolie. Book chapter in *Web Services Foundations*, Springer-Verlag, pages 81–107, 2014.

2. M. Carbone, F. Montesi, C. Schürmann. Choreographies, Logically. In *Proc. of the 25th International Conference on Concurrency Theory (CONCUR)*, pages 47–62, 2014.

3. F. Montesi, N. Yoshida. Compositional Choreographies. In *Proc. of the 24th International Conference on Concurrency Theory (CONCUR)*, pages 425–439, 2013.

4. M. Carbone, F. Montesi. Deadlock-freedom-by-design: Multiparty Asynchronous Global Programming. In *Proc. of 40th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 263–274, 2013.

5. F. Montesi. Process-aware Web Programming with Jolie. In *Proc. of 28th ACM SIGAPP Symposium on Applied Computing (SAC)*, pages 761–763, 2013.

6. I. Lanese, F. Montesi, G. Zavattaro. Amending Choreographies. In *Proc. of 9th International Workshop on Automated Specification and Verification of Web Systems (WWV)*, 2013.

7. M. Carbone, F. Montesi. Merging Multiparty Protocols in Multiparty Choreographies. In *Proc. of Programming Language Approaches to Concurrency and Communication-cEntric Software workshop (PLACES)*, pages 21–27, 2012.

8. C. Guidi, M. Dalla Preda, M. Gabbrielli, J. Mauro, F. Montesi. Service integration via target-transparent mediation. In *Proc. of IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 1–5, 2012.

9. C. Guidi, M. Dalla Preda, M. Gabbrielli, J. Mauro, F. Montesi. Interface-Based Service Composition with Aggregation. In *Proc. of European Conference on Service-Oriented and Cloud Computing (ESOCC)*, pages 48–63, 2012.

10. F. Montesi, M. Carbone. Programming services with correlation sets. In *Proc. of 9th International Conference on Service Oriented Computing (ICSOC)*, pages 125–141, 2011.

Contents

List of Figures

CHAPTER 1

Introduction

1.1 Problem Description

Distributed systems are ubiquitous and can be of vital importance in the modern developed society: they can be found in our information databases, mobile devices, transportation systems, and medical equipments. A distributed system is composed by individual endpoint nodes (endpoints for short) that execute concurrently and communicate with each other by sending and receiving messages. These communications are supported by channels, which can be implemented at the levels of software (e.g., TCP/IP sockets [24]) and hardware (e.g., network cables).

A major reason, if not the most important reason, that drove the wide adoption of distributed systems is their scalability: they can be formed by a single or a very large number of endpoints. For example, a personal computer is a distributed system in itself, in which the CPUs communicate with internal and external peripherals via standard buses (e.g., the Universal Serial Bus [97]). Different computers can be connected together in larger systems, by using wired or wireless networks. Such larger systems can be inter-connected once again, allowing endpoints in a system to communicate with those in other systems. By repeating this procedure, we can obtain huge "systems of systems" with billions (or more) of computer components. The currently best known system of systems of this kind is the Internet.

The scalability of distributed computing is not free of obstacles: bad programming of the sending/receiving actions of a single endpoint (also called input/output, or simply I/O actions) may compromise the functioning of a whole system. Typical consequences of programming errors are: (i) system freezes due to the indefinite waiting time for a message, known as deadlocks [35]; and (ii) faulty computations due to the concurrent access to a shared resource, known as race conditions [78]. More broadly, we refer to malfunctionings caused by bad communication programming as safety issues; we say that a distributed system is *safe* when it has no safety issues.

Programming safe distributed systems is known to be hard, due to the complex nondeterministic behaviour that arises from the concurrent execution of many endpoint programs. Such behaviour makes detecting bugs with static analyses more difficult, because it may generate an exponential growth of the possible execution traces of programs; notably, the decidability of static analyses for concurrent programs depends on the expressive power of the language primitives that endpoint programs can use to communicate, which elevates the importance of formal language design in this setting [91, 42]. Moreover, the usual diagnostic techniques for detecting software bugs during execution, such as testing and debugging, are more unreliable than in the sequential setting because bugs appear nondeterministically. In general, the concurrent executions of many endpoints and their interactions makes the global behaviour of a distributed system hard to predict; consequently, the programming of

distributed systems is notoriously error-prone.

In a recent study, Lu et al. analyse the bugs found in some major software projects (e.g., MySQL [83] and Mozilla web browsers [76]) and reveal that in most cases fixing a concurrency-related bug is hard even after it has been identified in the source code [66]. The study shows that most of the investigated concurrency bugs (97%), which include deadlocks and races, are due to violations of the programmers' order or atomicity intentions, i.e., systems executing actions in an order or a number not intended by the designer. These actions, in distributed systems, are the sendings and receivings of messages performed by the endpoints; therefore, the critical factor in achieving a safe distributed system is the correct implementation of the communications between the endpoints. The importance of this factor is also reflected in the efforts made by the software engineering community, where global descriptions of communication flows, such as Message Sequence Charts (MSC), are used to document the intended order of communications in distributed systems [59].

From the observations above, we can conclude that the programming of safe distributed software is error-prone because of the difficulty in programming the communication flows between endpoints.

Problem Statement

Programming safe distributed software is error-prone, because it is difficult to program correctly the communication flows between endpoints.

1.2 Aim and Hypothesis

The goal of this dissertation is to contribute to the scientific foundations of safe distributed programming. Since the safety of a distributed system depends on the correct programming of its communication flows, our aim is in particular to support programmers in the writing of safe communication flows, i.e., communication flows that do not make a distributed system unsafe.

Aim

To contribute to lay the scientific foundations of safe distributed programming, by developing methods that support programmers in the writing of safe communication flows.

Our methodology is to develop language models providing language abstractions that support the programming of safe communication flows. Language abstractions are known to play a major role when dealing with the writing and analysis of safe concurrent programs [66, 91, 94]. Our work starts from the following observation: the key problem in programming safe communication flows is that expressing them by defining the sending/receiving actions of each endpoint is difficult. A communication occurs when an endpoint executes a sending action and another endpoint executes a matching receiving action; communications are thus effects of matching endpoint actions. However, since the behaviour of each endpoint is defined in a separate program that will execute concurrently with the others, predicting whether the combination of endpoint programs will enact the intended communications is hard. Clearly, it would be much easier if programmers could just write the communications they want to take place, rather than focus on endpoint actions; therefore, we ask the following question:

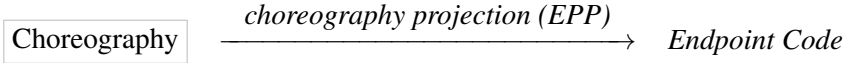
Is it possible to develop languages for programming distributed systems in which communications are the objects of discourse?

Our hypothesis is that the research question above can be answered positively by building on the emerging paradigm of Choreographic Programming. Choreographic programming is a programming paradigm for designing communication-based systems, where programmers write a *choreography* that describes how two or more endpoints exchange messages during execution, instead of a collection of programs that define individually the behaviour of each endpoint. As an example of a choreography, consider the following code (whose syntax is derived from the "Alice and Bob" notation by Needham and Schroeder [77]):

1. Alice –> Bob : $book$;
2. Bob –> Alice : $money$

The choreography above describes the behaviour of two endpoints, Alice and Bob. In Line 1, Alice sends to Bob a $book$; then, in Line 2, Bob replies to Alice with some $money$ for paying the book he received. Observe that a choreography is "safe by design", since it describes directly the intended communications in a system: a choreography can be seen as the formalisation of the communication flow intended by the programmer. Moreover, each communication is treated as atomic: the sending and receiving actions of the respective sender and receiver endpoints cannot be seen separately.

In [31], Carbone et al. show that it is theoretically possible to use choreographies for the programming of safe distributed systems, by projecting a choreography to the executable code for each endpoint (which we call endpoint code) using a procedure known as Endpoint Projection (EPP). The authors formally prove that their definition of EPP is correct, i.e., it preserves the intended behaviour of a projected choreography in the produced endpoint code; in other words, executing the endpoint code produced by EPP leads exactly to the communications defined in the originating choreography. This property enables a development methodology in which developers write a choreography and then distributed software implementing the choreography is automatically generated. We depict such methodology in the following:



The key aspect of the methodology above is that the produced endpoint code is safe by construction: since the EPP procedure is correct, it follows that the communications defined by the programmer are implemented faithfully and without errors. Carbone et al. prove that, as a consequence, the code produced by their EPP procedure is always free from deadlocks and race conditions [31]. Such results have been shown in other works in different, but similar, settings [89, 61]. In [67], the authors informally discuss how to project choreographies to endpoint code using real-world languages, i.e., the choreography language WS-CDL [102] and the endpoint process language WS-BPEL [79].

The safety properties that can be obtained by adopting choreographies make choreographic programming a candidate for supporting the development of safe distributed systems. However, this potential remains untapped because choreographies are currently impractical for other reasons. In their own investigation on theoretical foundations for choreographies and their EPP, Qiu et al. write that "it seems that a lot of work should be done

before a widely accepted result in this field" [89]; Aalst et al. also report that a "real" choreography language still has to be proposed [99]. We discuss some of the major issues in the following:

- **Compositionality.** Choreographies are not compositional: a choreography program cannot reuse other separately developed systems, nor be reused by them. Choreographies are thus unable to scale, as they cannot modularly build the typical systems of systems of distributed computing.

- **Multiparty Sessions.** Sessions are abstractions capturing private conversations between some endpoints; the concept is a hallmark of modern methodologies for distributed programming (found, e.g., in most web programming technologies [86, 82]). Currently, the only choreography model supporting the implementation of sessions is that presented in [31]. However, such model supports only the programming of binary sessions, i.e., sessions with two participants, and therefore it cannot be used to define the behaviour of sessions with multiple participants, called multiparty sessions.

- **Asynchrony.** In distributed systems communications can be asynchronous, i.e., a sender endpoint can send a message and proceed in its execution before the intended receiver has actually received the message. It is unclear how to represent asynchronous communications in choreography languages, because communications are given in choreographies as atomic actions and asynchrony is inherently linked to the separation of sending and receiving actions.

- **Mobility.** Choreographies do not offer any mobility mechanisms. For example, it is not possible for an endpoint to make another endpoint join a session during execution.

- **Round-trip Development.** In practice, software developers make use of choreographies in combination with the typical programming of each endpoint [1, 79]. The idea is to use the choreographic view to check that a system follows the expected flow of communications and to use the endpoint view to program the internal computations of each endpoint. Current formal choreography models do not support round-trip development; in general, supporting the editing of endpoint code in such a way that it is always possible to reconstruct the choreography that the endpoints implement is a hard problem [18, 63].

- **Implementation.** Despite a significant amount of research has already been conducted on formal choreography models, there are still no concrete implementations of applied programming languages based on them; this prevents the evaluation of such models in real-world scenarios.

The limitations described above need to be addressed if we want to employ choreographic programming for the development of real-world distributed systems. Of course, these are not the only features that will be needed in the future: as for the π-calculus [69], the seminal formal model of concurrent and mobile systems, many extensions may be needed depending on the application domain (cf. [20]). However, the features we listed

are of particular relevance; for example, compositionality and mobility are crucial for capturing the development of scalable systems and are hallmarks of successful languages and models for distributed computing [79, 1, 69]. Therefore, showing that choreographies can deal with such issues would be a strong sign that they can be successfully employed as a future basis for the development of safe distributed systems. The hypothesis of this dissertation is that this is possible, and that choreography models can be successfully applied to many real-world scenarios.

Hypothesis
It is possible to address the current limitations of choreography models and adopt choreographies for the programming of real-world distributed systems.

1.3 Thesis Statement

In this dissertation we investigate the paradigm of choreographic programming for demonstrating how it can capture the programming of real-world distributed systems, guaranteeing that the systems obtained through such methodology are safe:

Thesis Statement
Choreographic Programming provides a solid foundation for the development of safe distributed systems.

More specifically, we test our hypothesis by developing models for choreography-based programming and by investigating how such models can be implemented in real technologies.

1.4 Contributions

This dissertation provides three main contributions.

The first contribution is the development of formal models and type theories for choreographic programming that support asynchrony, compositionality, mobility, and multiparty sessions. Choreographies are mapped to endpoint programs through Endpoint Projection (EPP); EPP is formally proven to guarantee the expected safety properties of choreographies, among which freedom from deadlocks and race conditions.

The second contribution is the development of a formal logic, called Linear Connection Logic (LCL), that is a logical reconstruction of choreographies and their semantics. LCL is a generalisation of Linear Logic [46]. Based on previous results on a Curry-Howard correspondence between Linear Logic and the π-calculus [27], we show that LCL is in a Curry-Howard correspondence with a model of compositional choreographies that supports round-trip development, i.e., choreographies can always be translated to π-calculus terms and vice versa by following proof transformations in LCL.

The third contribution is the implementation of a prototype programming framework for choreographic programming, called Chor. Chor offers an Integrated Development Environment (IDE) for programming with choreographies, equipped with a type checker for

verifying that choreographies respect protocol specifications given as session types. Programs in Chor can be compiled to executable endpoint implementations in the Jolie programming language, a general-purpose language for distributed computing [60], which we extend to support the programming of multiparty sessions as required by our framework. We finally use Chor for evaluating choreographic programming against a series of use cases.

1.5 Structure of the Dissertation

The structure of the rest of this dissertation is outlined in the following. Each Chapter is an extended version of material produced in the course of the PhD studies, and is organised to be relatively self-contained.

- Part I: Models
  In this part we develop formal models for Choreographic Programming, and exploit the structure of choreographies to guarantee safety properties in distributed systems.
  - Chapter 2: Global Programming
    We present the Choreography Calculus, a language model for the programming of distributed systems using only global descriptions for both the specification of protocols and their implementation as multiparty asynchronous sessions. We exploit the global nature of choreographies to guarantee, by construction, the deadlock-freedom property of systems developed with choreographies.
  - Chapter 3: Compositional Choreographies
    We extend choreographies to handle the composition of systems developed independently. Our results yield a fresh view on how to deal with the notion of deadlock-freedom when some participants in a system are still missing (progress) and in the presence of mobility features.
  - Chapter 4: Round-Trip Choreographic Programming
    We propose a new logic, called Linear Connection Logic (LCL), obtained by conservatively extending Linear Logic. LCL enjoys a Curry-Howard correspondence with a fragment of our compositional choreographies: choreographies correspond to proofs in LCL and vice versa. By following our Curry-Howard correspondence, we derive a language model for round-trip development in which choreographies can always be transformed to equivalent processes and vice versa.
- Part II: Implementation
  In this part we develop the tools necessary to support the programming of multiparty sessions and then use them to build a prototype programming framework for choreographies.
  - Chapter 5: Programming Sessions with Correlation Sets
    We present a model for programming multiparty sessions by routing messages depending on their data content. We implement our model by extending the Jolie language.

  - Chapter 6: Process-aware Web Programming
    This chapter is dedicated to investigating how our extension of the Jolie language can be applied to the broad field of web programming. We extend Jolie to deal with HTTP messages natively, by abstracting them as messages that can be sent and received through sessions; then, we evaluate our approach through web-based use cases.
  - Chapter 7: Chor: a Framework for Choreographic Programming
    We present Chor, a language implementation of the Choreography Calculus from Chapter 2. Chor comes with an IDE for developing safe choreographies and automatically project them to executable code. We execute systems developed with Chor by using our extension of Jolie from Chapter 5.

- Part III: Appendices
  This part includes additional material and technical details relative to the first two parts.

  - Appendix A: Global Programming: Additional Material
  - Appendix B: Compositional Choreographies: Additional Material
  - Appendix C: Round-Trip Choreographic Programming: Additional Material

Part I

Models

CHAPTER 2

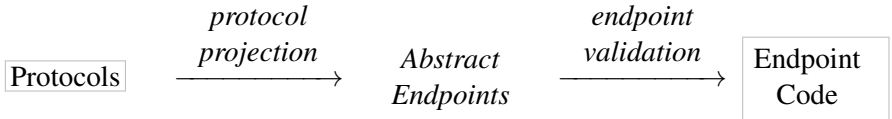
Global Programming

2.1 Introduction

The idea of using *global descriptions* of the communications in a system is not confined to choreographies. Another particularly fruitful application is that of using a global view for defining protocol specifications [55, 25]. Protocol specifications differ from choreographies in that a choreography concretely defines the implementation of a system, whereas a protocol defines requirements on the communication flow that a system implementation has to follow. In this chapter we develop a calculus for choreographic programming by bringing these two practices together, allowing choreographies to be checked for safety through globally-specified protocols. The outcome of this endeavour is the formal definition of a programming model that developers can use to program distributed systems by reasoning only at the global level, i.e., without having to deal with the programming of separate endpoint programs.

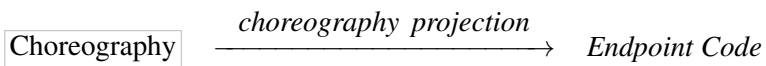
Global descriptions have been studied in formal models [26, 61, 55, 31], standards [102, 1], and language implementations [53, 87, 95]. Such descriptions have a great impact on the quality of software, as they represent "formal blueprints" of how communicating systems should behave and offer a concise view of the message flows enacted by a system. As mentioned above, global descriptions can be used at different levels of abstraction, ranging from abstract descriptions of *protocols* to descriptions of concrete system implementations, called *choreographies*. These different incarnations respectively underpin two recent and successful development methodologies.

In the first methodology, programmers design abstract protocols using global descriptions [55, 53]. These are automatically projected onto *abstract endpoint* specifications which are finally used for the static verification of manually written endpoint code:



The approach depicted above has the benefit of producing very clear protocol specifications. However, it deprives the programmer from a global view of the system when dealing with its implementation. A major consequence is that programming becomes error-prone when dealing with the actual interleaving of different protocol instances. For example, it can easily lead to deadlocked systems [21].

The second methodology deals with system implementations using choreographies [102, 95]. Programmers can write a choreography and then automatically project an executable system from it:



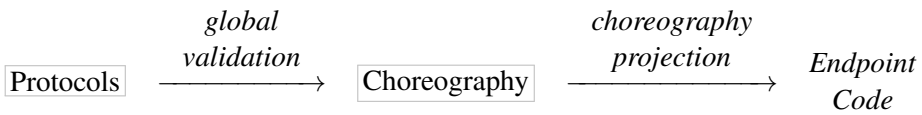
Here, the main advantage is the precise view given by choreographies on the possible system executions. However, choreographies lack in abstraction wrt global protocol descriptions and their programming needs to be disciplined with additional tools. Current disciplines for choreographic programming are based on writing abstract endpoint descriptions and then using them to check the behaviour of each endpoint, directly on the choreography or its EPP [31, 95]. Hence, in these models, we lose a global view of the system when describing its protocol specifications.

Inspired by these observations and by private conversations with our industry collaborators [102, 90, 6, 80, 41], we ask:

> *Can we design a unified framework that combines global descriptions of protocols and implementations?*

Clearly, a positive answer would retain the advantages of global descriptions for *both* the writing of protocols and that of implementations. Moreover, a natural following question is whether such a unified framework could offer more than just the sum of the parts: are there other advantages that can arise from the combination of global protocol descriptions with choreographies?

In order to answer the questions above, we build and analyse a model for a fully-global framework. In our model, developers design both protocols and implementations from a global viewpoint. Endpoint implementations can then be automatically generated:



The challenge of reaching our objective is twofold. First, since we aim at designing a model where choreographies can instantiate different protocols multiple times and interleave their execution, the model should ensure that these interleavings will not lead to bad behaviour. Second, it is not clear how common aspects of concurrent systems such as *asynchrony* (communications are asynchronous) and *parallelism* (parallel executions) should influence the interpretation of a choreography: choreographies describe communications as atomic actions, making concurrency less explicit.

2.1.1 Contributions

This chapter provides the following contributions:

Multiparty Choreographies. We introduce a choreography model with *multiparty* protocol instances (*sessions*) as first-class elements (§ 2.3) and provide an EPP that, under simple restrictions, correctly generates endpoint code from a choreography (§ 2.5).

Asynchrony and Parallelism. Our framework gives a novel and concise interpretation of asynchrony and parallelism, by inferring the implicit concurrent behaviour specified in a choreography (§ 2.3).

Typing and Type inference. We provide a type system (§ 2.4) for checking choreographies against protocol specifications given as multiparty session types [55]. For the first time, we formalise an operational semantics for multiparty session types that defines

the abstract execution of global protocol specifications (§ 2.4.1); we use our semantics for protocols to establish a tight operational correspondence between the execution of protocols and the choreographies they type (§ 2.4.4). Our type analysis plays a major role in ensuring the correctness of EPP code. Interestingly, due to the global nature of choreographies, our framework can generate correct endpoint code that is not allowed by current multiparty session typings (§ 2.5.6). We also give a type inference technique supporting the opposite methodology, i.e., extracting the protocols implemented in a choreography (§ 2.4.5).

Delegation. This is the first work to provide a choreography model supporting *session delegation*, a mobility mechanism for delegating the continuation of a protocol (§ 2.3). Due to asynchrony and parallelism, typing delegation (§ 2.4) is nontrivial since messages prior to and after delegation may be interleaved, making it difficult to check that channel ownerships are consistently respected.

Deadlock-freedom-by-design. Our framework seamlessly guarantees deadlock freedom (§ 2.5.5, Corollary 2.5.1.1), a notoriously hard problem in multiparty sessions types [21]. This feature follows from using a choreography as initial design tool.

Evaluation. We evaluate our programming model against examples of different nature, from multicore to distributed programming (§ 2.6).

2.2 Preview

In this section we give an informal description of our model, whose key elements are *protocols* and *choreographies*. A protocol is an abstract specification of the structure of some communications in a system, whereas a choreography describes a concrete system implementing one or more protocols. We represent protocols with *global types* [55], global descriptions where entities are abstracted as *roles* that communicate following a given conversation structure.

Example 2.2.1 (Two-buyer protocol)**.** In this protocol, two buyers `B1` and `B2` wish to share the purchase of a product from a seller `S`:

1. $\texttt{B1} \texttt{ -> } \texttt{S} : \langle \mathbf{string} \rangle ; \texttt{S} \texttt{ -> } \texttt{B1} : \langle \mathbf{int} \rangle ; \texttt{S} \texttt{ -> } \texttt{B2} : \langle \mathbf{int} \rangle ; \texttt{B1} \texttt{ -> } \texttt{B2} : \langle \mathbf{int} \rangle ;$
2. $\texttt{B2} \texttt{ -> } \texttt{S} : \{\ ok : \texttt{B2} \texttt{ -> } \texttt{S} : \langle \mathbf{string} \rangle ; \texttt{S} \texttt{ -> } \texttt{B2} : \langle \mathbf{date} \rangle, \quad quit : \mathsf{end}\ \}$

Above, `B1`, `B2` and `S` are called *roles*. Buyer `B1` sends to a seller `S` a purchase request of type $\mathbf{string}$. Then, `S` sends a quote to `B1` and another potential buyer `B2`. Thereafter, `B1` tells `B2` the amount she wishes to contribute with. Afterwards, `B2` notifies `S` of whether she has accepted (ok or $quit$). If so, `B2` sends to `S` a string (address) and, finally, `S` replies with a delivery date of type $\mathbf{date}$. □

In this Chapter, we introduce a choreography model for globally implementing protocols such as the one above. Its core elements are *processes* and *sessions*. A process represents a (logical) processing unit that executes a sequence of instructions. Each process has its own local variables, and can exchange messages with other processes by performing I/O actions. Processes can be programmed to be already active or dynamically created at runtime. A session is an instance of a protocol and implements communications between some processes. Sessions can be dynamically created by processes.

Example 2.2.2 (Two-buyer choreography)**.** We give a choreography implementing the two-buyer protocol in Example 2.2.1.

1. $\mathsf{b}_1[\mathtt{B1}], \mathsf{b}_2[\mathtt{B2}]$ **start** $\mathsf{s}[\mathtt{S}] : a(k)$;
2. $\mathsf{b}_1[\mathtt{B1}].book$ –> $\mathsf{s}[\mathtt{S}].x_1 : k$;
3. $\mathsf{s}[\mathtt{S}].\mathsf{quote}(x_1)$ –> $\mathsf{b}_1[\mathtt{B1}].y_1 : k$;
4. $\mathsf{s}[\mathtt{S}].\mathsf{quote}(x_1)$ –> $\mathsf{b}_2[\mathtt{B2}].z_1 : k$;
5. $\mathsf{b}_1[\mathtt{B1}].\mathsf{contrib}(y_1)$ –> $\mathsf{b}_2[\mathtt{B2}].z_2 : k$;
6. if $(z_1 - z_2 \leq 100)@\mathsf{b}_2$
7. then $\mathsf{b}_2[\mathtt{B2}]$ –> $\mathsf{s}[\mathtt{S}] : k[ok]$;
8. $\quad \mathsf{b}_2[\mathtt{B2}].addr$ –> $\mathsf{s}[\mathtt{S}].x_2 : k$;
9. $\quad \mathsf{s}[\mathtt{S}].ddate$ –> $\mathsf{b}_2[\mathtt{B2}].z_3 : k$
10. else $\mathsf{b}_2[\mathtt{B2}]$ –> $\mathsf{s}[\mathtt{S}] : k[quit]$

In Line 1, processes b_1, b_2 and freshly spawned process s start a session k through shared channel a playing roles $\mathtt{B1}$, $\mathtt{B2}$ and $\mathtt{S}$ respectively. In Lines 2–5, b_1 asks s for book "$book$" and gets back the quote "$\mathsf{quote}(book)$" which is also sent to b_2. Note that, e.g., b_1 uses its local variable y_1 to receive the evaluation of "$\mathsf{quote}(book)$". Then, b_1 tells b_2 the amount she wishes to contribute for the purchase, namely "$\mathsf{contrib}(\mathsf{quote}(book))$". In Line 6, b_2 evaluates the offer received by b_1 in the guard $(z_1 - z_2 \leq 100)@\mathsf{b}_2$. If positive, b_2 communicates her decision with the selection $\mathsf{b}_2[\mathtt{B2}]$ –> $\mathsf{s}[\mathtt{S}] : k[ok]$, sends her address $addr$ and receives the delivery date $ddate$ (Lines 7-9). Otherwise, b_2 aborts by selecting $quit$ in Line 10. □

Observe that the structure of session k in Example 2.2.2 is that of the protocol given in Example 2.2.1. The only differences are that data has become explicit and that we introduced the **start** primitive. The latter allows processes to synchronise on a shared name, e.g., a, and create new processes and sessions. In Line 1 of Example 2.2.2, b_1 and b_2 are already active processes while s is a service process, i.e., a dynamically spawned process. Active processes appear on the left-hand side of the **start** keyword, whereas (fresh) service processes appear on its right-hand side. Role annotations, e.g., $\mathsf{b}_1[\mathtt{B1}]$, relate each process to the role it plays in a session.

Example 2.2.3 (Two-buyer-helper choreography)**.** Choreographies can also describe multiple, interleaved instances of multiple protocols. Hereafter, we extend the two-buyer choreography from Example 2.2.2 with two other sessions, k' and k'', that b_1 and b_2 will

respectively use for getting help in the transaction.

...as Lines 1-5 in Example 2.2.2...

6. $\mathsf{b}_1[\mathtt{B}]$ **start** $\mathsf{h}_1[\mathtt{H}] : b(k')$;
7. $\mathsf{b}_1[\mathtt{B}].\mathsf{contrib}(y_1)/2$ -> $\mathsf{h}_1[\mathtt{H}].y : k'$;
8. $\mathsf{b}_1[\mathtt{B}]$ -> $\mathsf{h}_1[\mathtt{H}] : k'[done]$;

C_1

9. $\mathsf{b}_2[\mathtt{B}]$ **start** $\mathsf{h}_2[\mathtt{H}] : b(k'')$;
10. $\mathsf{b}_2[\mathtt{B}].((z_1 - z_2)/2)$ -> $\mathsf{h}_2[\mathtt{H}].z : k''$;
11. $\mathsf{b}_2[\mathtt{B}]$ -> $\mathsf{h}_2[\mathtt{H}] : k''[del]$;
12. $\mathsf{b}_2[\mathtt{B}].z_1$ -> $\mathsf{h}_2[\mathtt{H}].z' : k''$;
13. $\mathsf{b}_2[\mathtt{B}]$ -> $\mathsf{h}_2[\mathtt{H}] : k''\langle k[\mathtt{B2}]\rangle$;

C_2

14. **if** $((z/z') \leq 30\%)@\mathsf{h}_2$
15. **then** $\mathsf{h}_2[\mathtt{B2}]$ -> $\mathsf{s}[\mathtt{S}] : k[ok]$;
16. $\mathsf{h}_2[\mathtt{B2}].addr$ -> $\mathsf{s}[\mathtt{S}].x_2 : k$;
17. $\mathsf{s}[\mathtt{S}].ddate$ -> $\mathsf{h}_2[\mathtt{B2}].z'' : k$
18. **else** $\mathsf{h}_2[\mathtt{B2}]$ -> $\mathsf{s}[\mathtt{S}] : k[quit]$

C_3

The choreography starts with the first 5 lines of that in Example 2.2.2. In block C_1, process b_1 starts a new session with a helper process h_1, asks it to contribute for half of its part (Line 7), and informs it that it does not need to do more (Line 8). On the other hand, in block C_2, b_2 does the same with another process h_2 until Line 11. Differently now, b_2 asks h_2 to continue session k *by taking on its role* (Line 11). Then, it sends the total price received from s to h_2 (Line 12) and *delegates* the session reference (Line 13). Finally, in block C_3, h_2 completes the two-buyer protocol instead of b_2, checking that its own contribution is less than or equal to 30% of the total price. Note that h_1 and h_2 are started through the same shared channel b, which acts as a reusable channel in multiparty session types [55]. □

Our model has two features that interestingly influence the interpretation of a choreography in subtly different ways. Firstly, *parallelism*: process executions may concurrently proceed without any predetermined ordering unless causal constraints are introduced. Secondly, *asynchrony*: communications are asynchronous, so a process may send a message to another process and then immediately proceed before the message has actually been delivered by the network.

For instance, in Example 2.2.3, the processes whose behaviour is described in blocks C_1 and C_2 are different (b_1 and h_1 for C_1, b_2 and h_2 for C_2). Therefore, their executions may interleave due to parallelism; e.g., b_2 and h_2 may start session k'' *before* b_1 and h_1 start session k'. Even more, k'' may be completely executed before k' is started. Hence, the interpretation of $C_1; C_2$ should be equivalent to that of $C_2; C_1$, and, in general, to that of any interleaving of C_1 and C_2. Furthermore, in Lines 3 and 4 of Example 2.2.2, where s sends the quote to b_1 and then b_2, it may happen that b_2 (Line 4) receives the quote before b_1 due to asynchronous messaging. Parallelism and asynchrony are respectively handled by our *swapping* relation and our asynchronous semantics, both formalised in the next section (§ 2.3).

In general, we say that our relaxed sequential operator lifts the programmer from expressing the degree of concurrency (asynchrony and parallelism) of a system. Indeed, our

$$
\begin{array}{rcll}
C & ::= & \eta; C & \textit{(seq)} \\
 & | & \mathsf{if}\ e@\mathsf{p}\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2 & \textit{(cond)} \\
 & | & \mathbf{0} & \textit{(inact)} \\
 & | & \mathsf{def}\ X(\tilde{D}) = C'\ \mathsf{in}\ C & \textit{(def)} \\
 & | & X\langle \tilde{E} \rangle & \textit{(call)} \\
 & | & (\boldsymbol{\nu} r)\ C & \textit{(res)} \\
\hline
\eta & ::= & p.e \texttt{ -> } q.x : k & \textit{(com)} \\
 & | & p \texttt{ -> } q : k[l] & \textit{(sel)} \\
 & | & p \texttt{ -> } q : k\langle k'[\mathtt{C}] \rangle & \textit{(del)} \\
 & | & \tilde{p}\ \mathbf{start}\ \tilde{q} : a(k) & \textit{(start)} \\
\hline
p, q, \ldots & ::= & \mathsf{p}[\mathtt{A}] & \textit{(typed process)} \\
D & ::= & \mathsf{p}(\tilde{x}, \tilde{k}) & \textit{(def param)} \\
E & ::= & \mathsf{p}(\tilde{e}, \tilde{k}) & \textit{(call param)}
\end{array}
$$

Figure 2.1: Choreography Calculus, syntax.

framework will automatically infer the latter by looking at the process identifiers. We made this choice in favour of design minimality and simplicity. However, in applied languages and tools combining the sequential operator with explicit primitives for, e.g., parallelism, may be preferable for clarity purposes. We discuss this in § 2.8, Sequential and Parallel Operators.

2.3 Choreography Calculus

We introduce the *Choreography Calculus (CC)*, a choreography model with multiparty asynchronous sessions.

2.3.1 Syntax

The syntax of CC is reported in Figure 2.1. C is a choreography, p is a process identifier, $\mathtt{A}$ is a role, and k is a session identifier (or session channel). Interactions between processes are specified by the term $\eta; C$ which reads: *the system may execute the interaction η and continue as C*. We distinguish four different kinds of interaction: *(start)*, *(com)*, *(sel)*, and *(del)*. Term *(start)* denotes session initiation: processes $\tilde{\mathsf{p}}$ and $\tilde{\mathsf{q}}$ start a new multiparty session through shared channel a and tag it with a fresh identifier k, called session channel. Processes $\tilde{\mathsf{p}}$, dubbed the *active processes*, are already running, while processes $\tilde{\mathsf{q}}$, dubbed the *service processes*, are dynamically created and started. We assume that $|\tilde{p}| + |\tilde{q}| \geq 2$ (a session has at least two participating processes) and that $\tilde{p}$ is nonempty (a session is started by at least one running process). Each process is annotated with the role it plays

in the newly created session k. Term *(com)* denotes a communication where process $\mathtt{p}$ sends, over session k, the evaluation of a first-order expression e to process $\mathtt{q}$, which binds it to variable x. In *(sel)*, $\mathtt{p}$ communicates to $\mathtt{q}$ its selection of branch l. Through *(del)*, $\mathtt{p}$ delegates to $\mathtt{q}$ over k its role $\mathtt{C}$ in session k'.

CC also offers some standard programming language constructs. In *(cond)*, expression e is labelled with a process name, indicating where it is evaluated. Terms *(def)* defines a recursive procedure X, declaring parameters $\tilde{D}$ of the form $\mathtt{p}(\tilde{x}, \tilde{k})$, meaning that process $\mathtt{p}$ uses variables $\tilde{x}$ and sessions $\tilde{k}$ inside the procedure. A procedure can be called using term *(call)*, where we assume that each expression can only be either a variable or a value; we also assume that procedure calls are always guarded. Term *(res)* models name restriction, where a name r (for restricted name) can be a process or a session. We assume that name restriction is never used inside the body C' of a procedure. The term $\mathbf{0}$ denotes termination.

In a term $\eta; C$, η can bind session channels, processes and variables. When η is a *(start)*, $\tilde{\mathtt{p}}$ and a are free while k and $\tilde{\mathtt{q}}$ are bound (since they are freshly created). If η is a *(com)*, variable x is bound. As usual, r is bound in $(\boldsymbol{\nu} r)\, C$. We often omit $\mathbf{0}$, empty vectors, and irrelevant variables. In the remainder, $(\boldsymbol{\nu} r_1, \ldots, r_n)$ is a shortcut for $(\boldsymbol{\nu} r_1) \ldots (\boldsymbol{\nu} r_n)$.

Remark 2.3.1 (Role Annotations). For clarity, we annotate processes with roles in all interactions. Technically, this is necessary only for term *(start)* since roles can be inferred from session identifiers in all other terms. For example, consider the following (pseudo) choreography:

1. $\mathtt{p}[\mathtt{A}]$ **start** $\mathtt{q}[\mathtt{B}] : a(k)$;
2. $\mathtt{p}.e$ -> $\mathtt{q}.y : k$

Above, Line 2 defines a communication from process $\mathtt{p}$ to process $\mathtt{q}$ without specifying their roles in session k. However, it would be easy to infer that their roles are, respectively, $\mathtt{A}$ and $\mathtt{B}$ by looking at the preceding *(start)* term in Line 1. Avoiding role annotations may be preferable in a real tool, and indeed we do not require them in the language implementation that we present in Chapter 7. On the contrary, having explicit role annotations is more elegant when discussing our model, because in the semantics for choreographies that we present in the next section *(start)* terms can be executed and consumed; in such cases, explicit role annotations allow us to avoid using extra constructs, or term rewritings, to bookkeep the roles of each process in a session. □

2.3.2 Semantics

Above, we stated that the term $\eta; C$ specifies a system that *may* execute the interaction η and then continue as C. Processes, however, are assumed to run in parallel. As a consequence, some actions in C may be performed before η. For example, blocks C_1 and C_2 from Example 2.2.3 describe the behaviour of different processes. Therefore, as discussed in § 2.2, in an actual system run of these processes their executions may interleave due to parallelism. To deal with such cases, we define the *swapping* congruence relation $\simeq_{\mathcal{C}}$, which allows permutations of this kind of interaction sequences[1]. The relation $\simeq_{\mathcal{C}}$ is defined as the smallest congruence satisfying the rules in Figure 2.2. The rules exchange terms with disjoint sets of participating processes. Rule $\lfloor {}^{\text{CS}} |_{\text{ETA-ETA}} \rceil$ swaps two interactions

[1] Handling parallelism with a syntactic congruence simplifies our development, since swaps in a choreography do not influence the behaviour of its EPP. We will formalise this notion in § 2.5.

$$\frac{\mathsf{pn}(\eta) \asymp \mathsf{pn}(\eta')}{\eta;\eta' \simeq_{\mathcal{C}} \eta';\eta} \lfloor^{\text{CS}}|_{\text{ETA-ETA}}\rceil$$

$$\frac{\mathsf{p} \neq \mathsf{q}}{\begin{array}{c}\mathsf{if}\ e@\mathsf{p}\ \mathsf{then}\ (\ \mathsf{if}\ e'@\mathsf{q}\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2\ )\ \mathsf{else}\ (\ \mathsf{if}\ e'@\mathsf{q}\ \mathsf{then}\ C_1'\ \mathsf{else}\ C_2'\ ) \\ \simeq_{\mathcal{C}} \\ \mathsf{if}\ e'@\mathsf{q}\ \mathsf{then}\ (\mathsf{if}\ e@\mathsf{p}\ \mathsf{then}\ C_1\ \mathsf{else}\ C_1')\ \mathsf{else}\ (\mathsf{if}\ e@\mathsf{p}\ \mathsf{then}\ C_2\ \mathsf{else}\ C_2')\end{array}} \lfloor^{\text{CS}}|_{\text{COND-COND}}\rceil$$

$$\frac{\mathsf{p} \notin \mathsf{pn}(\eta)}{\mathsf{if}\ e@\mathsf{p}\ \mathsf{then}\ (\eta;C_1)\ \mathsf{else}\ (\eta;C_2) \simeq_{\mathcal{C}} \eta;(\mathsf{if}\ e@\mathsf{p}\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2)} \lfloor^{\text{CS}}|_{\text{ETA-COND}}\rceil$$

Figure 2.2: Choreography Calculus, swap relation $\simeq_{\mathcal{C}}$.

η and η' that do not share any process names. In the rule, the auxiliary function $\mathsf{pn}(\eta)$ returns the set of process names in η and $\asymp$ is a shortcut for asserting that two sets are disjoint (for two sets A and B, $A \asymp B$ if and only if $A \cap B = \emptyset$). Rule $\lfloor^{\text{CS}}|_{\text{COND-COND}}\rceil$ swaps two independent conditionals tested by different processes, rearranging the branches C_1' and C_2 to preserve the semantics of the choreography term. C_1' and C_2 are swapped to preserve the semantics of the term wrt the evaluations of the conditions. Finally, rule $\lfloor^{\text{CS}}|_{\text{ETA-COND}}\rceil$ rule swaps an interaction η out of a conditional if η prefixes both branches in the conditional and does not involve the process that evaluates the condition.

Asynchronous messaging can cause situations as the one discussed for Lines 3 and 4 of Example 2.2.2 in § 2.2, where s sends the quote to b_1, then to b_2, and b_2 may receive the quote before b_1. Unlike for parallelism, we address this issue directly in the operational semantics. This is because asynchrony is asymmetric: even though the receiving actions may interleave in a different order wrt that in the choreography, the sending actions instead will surely happen in the specified order, since the process performing the outputs is the same. This is different from parallelism, where the ordering of *both* receiving and sending actions may change. It would be unsafe to manipulate the syntax of the choreography for simulating asynchrony using a congruence relation such as swapping, since when we will generate the code for the sender process (cf. § 2.5) remembering the order of the outputs will be important[2].

Figure 2.3 contains the rules defining the labelled reduction semantics for CC, whose labels λ are defined as:

$$\lambda \ ::= \quad \eta \quad | \quad \tau@\mathsf{p} \quad | \quad (\boldsymbol{\nu} r)\ \lambda$$

A label is either an interaction η, an internal action $\tau@\mathsf{p}$ (used in the evaluation of conditionals), or another label with restricted name r (when new processes or session channels are created). Rule $\lfloor^{\text{C}}|_{\text{ACT}}\rceil$ models interactions that are not *(com)*. In the reductum, if η is a *(start)* then $\tilde{r}$ contains the freshly created service processes and the session channel. For all other cases, $\tilde{r}$ is empty. In $\lfloor^{\text{C}}|_{\text{COM}}\rceil$, we substitute variable x with value v (the evaluation of the expression e in a system that we leave unspecified) with the *localised substitution*

[2] We conjecture that an alternative syntactic notion to capture asynchrony can be defined as a precongruence relation. We leave the study of such alternative to future work.

$$\frac{\eta \in \{(sel),(del),(start)\} \quad \tilde{r} = \mathsf{bn}(\eta)}{\eta; C \xrightarrow{\eta} (\boldsymbol{\nu}\tilde{r})\, C} \lfloor^{\mathrm{C}}|_{\textsc{Act}}\rceil \qquad \frac{\eta = \mathsf{p}[\mathtt{A}].e \texttt{ -> } \mathsf{q}[\mathtt{B}].x : k \quad e \downarrow v}{\eta; C \xrightarrow{\eta[v/e]} C[v/x@\mathsf{q}]} \lfloor^{\mathrm{C}}|_{\textsc{Com}}\rceil$$

$$\frac{\begin{array}{lll} C \xrightarrow{\lambda} (\boldsymbol{\nu}\tilde{r})\, C' & \mathsf{snd}(\eta) \in \mathsf{fn}(\lambda) & \tilde{r} = \mathsf{bn}(\lambda) \\ \eta \neq (start) & \mathsf{rcv}(\eta) \notin \mathsf{fn}(\lambda) & \tilde{r} \notin \mathsf{fn}(\eta) \end{array}}{\eta; C \xrightarrow{\lambda} (\boldsymbol{\nu}\tilde{r})\, \eta; C'} \lfloor^{\mathrm{C}}|_{\textsc{Async}}\rceil$$

$$\frac{C_1 \xrightarrow{\lambda} C_1'}{\mathsf{def}\, X(\tilde{D}) = C_2 \,\mathsf{in}\, C_1 \xrightarrow{\lambda} \mathsf{def}\, X(\tilde{D}) = C_2 \,\mathsf{in}\, C_1'} \lfloor^{\mathrm{C}}|_{\textsc{Ctx}}\rceil$$

$$\frac{i = 1 \text{ if } e \downarrow \mathsf{true}\,,\ i = 2 \text{ otherwise}}{\mathsf{if}\, e@\mathsf{p}\, \mathsf{then}\, C_1\, \mathsf{else}\, C_2 \xrightarrow{\tau@\mathsf{p}} C_i} \lfloor^{\mathrm{C}}|_{\textsc{Cond}}\rceil \qquad \frac{C \xrightarrow{\lambda} C'}{(\boldsymbol{\nu}r)\, C \xrightarrow{(\boldsymbol{\nu}r)\,\lambda} (\boldsymbol{\nu}r)\, C'} \lfloor^{\mathrm{C}}|_{\textsc{Res}}\rceil$$

$$\frac{\mathcal{R} \in \{\simeq_{\mathcal{C}}, \equiv\} \quad C_1\, \mathcal{R}\, C_1' \quad C_1' \xrightarrow{\lambda} C_2' \quad C_2'\, \mathcal{R}\, C_2}{C_1 \xrightarrow{\lambda} C_2} \lfloor^{\mathrm{C}}|_{\textsc{Eq}}\rceil$$

Figure 2.3: Choreography Calculus, semantics.

$$(\boldsymbol{\nu}r)\, \mathbf{0} \equiv \mathbf{0} \quad (\boldsymbol{\nu}r)\, (\boldsymbol{\nu}r')\, C \equiv (\boldsymbol{\nu}r')\, (\boldsymbol{\nu}r)\, C \quad \mathsf{def}\, X(\tilde{D}) = C \,\mathsf{in}\, \mathbf{0} \;\equiv\; \mathbf{0}$$
$$\mathsf{def}\, X(\tilde{D}) = C' \,\mathsf{in}\, \big((\boldsymbol{\nu}r)\, C\big) \;\equiv\; (\boldsymbol{\nu}r)\, \big(\mathsf{def}\, X(\tilde{D}) = C' \,\mathsf{in}\, C\big) \;\text{ if }\; r \notin \mathsf{fn}(C')$$
$$\mathsf{def}\, X(\widetilde{\mathsf{p}(\tilde{x}, \tilde{k})}) = C' \,\mathsf{in}\, C[X\langle \widetilde{\mathsf{p}(\tilde{e}, \tilde{k})}\rangle] \;\equiv\; \mathsf{def}\, X(\widetilde{\mathsf{p}(\tilde{x}, \tilde{k})}) = C' \,\mathsf{in}\, C[C'[\widetilde{\tilde{e}_i/\tilde{x}_i@\mathsf{p}_i}]]$$

Figure 2.4: Choreography Calculus, structural congruence $\equiv$.

$C[v/x@\mathsf{q}]$, which substitutes x with v only under the free process name q in C, modelling local variables. Rule $\lfloor^{\mathrm{C}}|_{\textsc{Async}}\rceil$ captures the asynchronous behaviour of endpoint systems, allowing a process to send a message and then proceed freely before the intended receiver actually receives it. In the rule, the sender of η performs the action λ in the continuation C without waiting for the message in η to be delivered. We check that the receiver of η is not involved in λ since otherwise causality between η and λ would be violated. Finally, η is kept for the later observation of the message delivery. In rule $\lfloor^{\mathrm{C}}|_{\textsc{Eq}}\rceil$, the relation $\mathcal{R}$ can be either the swapping relation $\simeq_{\mathcal{C}}$ or the *structural congruence* $\equiv$. Structural congruence handles name restriction and recursion unfolding; it is formally defined as the smallest congruence supporting α-conversion and satisfying the rules reported in Figure 2.4. In the rule for recursion unfolding (last rule in Figure 2.4), we use the notation $C[X\langle \widetilde{\mathsf{p}(\tilde{e}, \tilde{k})}\rangle]$ for indicating that $X\langle \widetilde{\mathsf{p}(\tilde{e}, \tilde{k})}\rangle$ is a subterm of C that we replace with the term $C'[\widetilde{\tilde{e}_i/\tilde{x}_i@\mathsf{p}_i}]$ (the substitution of the procedure parameters in its body with the expressions used in the

$$
\begin{array}{rcll}
G & ::= & \texttt{A -> B} : \langle U \rangle ; G & \textit{(com)} \\
 & | & \texttt{A -> B} : \{l_i : G_i\}_{i \in I} & \textit{(branch)} \\
 & | & \mathsf{end} & \textit{(end)} \\
 & | & \mathsf{rec}\, \mathbf{t} ; G & \textit{(rec)} \\
 & | & \mathbf{t} & \textit{(call)} \\
U & ::= & S \mid G@\texttt{A} & \textit{(values)} \\
S & ::= & \mathbf{bool} \mid \mathbf{int} \mid \mathbf{string} \mid \ldots & \textit{(sort)}
\end{array}
$$

Figure 2.5: Global Types, syntax.

call) in the right-hand side.

Deadlock-freedom of Choreographies. Choreographies enjoy deadlock-freedom, provided that (i) they do not contain free variable names and (ii) the arguments in procedure calls match the parameters of their respective procedure definitions. We refer to property (ii) as well-sortedness in the remainder.

Theorem 2.3.2 (Deadlock-freedom)**.** *Let C be well-sorted and contain no free variable names; then, $C \not\equiv \mathbf{0}$ implies that there exist C', λ such that $C \xrightarrow{\lambda} C'$.*

Proof. By induction on the structure of C. The interesting cases are $C = \mathsf{def}\, X(\tilde{D}) = C' \,\mathsf{in}\, X\langle \tilde{E} \rangle$ and $C = p.e \texttt{ -> } q.x : k; C'$. For the first, we can apply structural congruence to expand the call because C is well-sorted and then the thesis follows by induction hypothesis. For the second case, we can apply rule $\lfloor {}^{\text{C}} | {}_{\text{COM}} \rceil$ because we can evaluate $e \downarrow v$ as C contains no free variable names. All other cases are trivial: they follow immediately by the semantics of the Choreography Calculus, since all other terms can be reduced by a corresponding rule. □

2.4 Typing Choreographies

We now present our typing system which allows to specify protocols in terms of global types [55, 21] and then check whether session behaviours in a choreography respect them.

2.4.1 Types

We report the syntax and semantics of global types, types for specifying multiparty protocols.

2.4.1.1 Syntax

The syntax of global types is reported in Figure 2.5. A type $\texttt{A -> B} : \langle U \rangle ; G$ abstracts an interaction from role `A` to role `B` with continuation G, where U, dubbed *carried type*, is the type of the exchanged message. A carried type U can either be a basic type S or a delegation type $G@\texttt{A}$. Communicating $G@\texttt{A}$ means that the sender role *delegates* to the receiver role her role `A` in protocol G. In the type $\texttt{A -> B} : \{l_i : G_i\}_{i \in I}$, role `A` can select

$$\frac{}{\mathtt{A} \to \mathtt{B} : \langle U \rangle; G \xrightarrow{\mathtt{A} \to \mathtt{B}:\langle U \rangle} G} \lfloor^{\mathrm{G}}|_{\textsc{Com}}\rceil$$

$$\frac{}{\mathtt{A} \to \mathtt{B} : \{l_i : G_i\}_{i \in I \cup \{j\}} \xrightarrow{\mathtt{A} \to \mathtt{B}:[l_j]} G_j} \lfloor^{\mathrm{G}}|_{\textsc{Branch}}\rceil$$

$$\frac{G[\mathsf{rec}\,\mathbf{t}; G/\mathbf{t}] \xrightarrow{\alpha} G'}{\mathsf{rec}\,\mathbf{t}; G \xrightarrow{\alpha} G'} \lfloor^{\mathrm{G}}|_{\textsc{Rec}}\rceil \qquad \frac{G_1 \simeq_{\mathcal{G}} G_1' \xrightarrow{\alpha} G_2' \simeq_{\mathcal{G}} G_2}{G_1 \xrightarrow{\alpha} G_2} \lfloor^{\mathrm{G}}|_{\textsc{Swap}}\rceil$$

$$\frac{G \xrightarrow{\alpha} G' \quad \mathtt{A} \in \mathsf{roles}(\alpha), \mathtt{B} \notin \mathsf{roles}(\alpha)}{\mathtt{A} \to \mathtt{B} : \langle U \rangle; G \xrightarrow{\alpha} \mathtt{A} \to \mathtt{B} : \langle U \rangle; G'} \lfloor^{\mathrm{G}}|_{\textsc{ACom}}\rceil$$

$$\frac{G_j \xrightarrow{\alpha} G_j' \quad \mathtt{A} \in \mathsf{roles}(\alpha), \mathtt{B} \notin \mathsf{roles}(\alpha)}{\mathtt{A} \to \mathtt{B} : \{l_i : G_i\}_{i \in I \cup \{j\}} \xrightarrow{\alpha} \mathtt{A} \to \mathtt{B} : \{l_j : G_j'\}} \lfloor^{\mathrm{G}}|_{\textsc{ABranch}}\rceil$$

Figure 2.6: Global Types, semantics.

one label l_i offered by role B and the protocol continues as G_i. Type end is the terminated protocol. The types $\mathsf{rec}\,\mathbf{t}; G$ and $\mathbf{t}$ handle recursion. We regard recursive types in the standard way [88], i.e., taking an equi-recursive view such that type variables only appear under prefixes.

2.4.1.2 Semantics

We give a semantics for global types, denoted by $\xrightarrow{\alpha}$, which expresses the (abstract) execution of protocols. Formally, $G \xrightarrow{\alpha} G'$ is the smallest relation on the recursion-unfolding of global types satisfying the rules given in Figure 2.6, where labels α are defined as:

$$\alpha \ ::= \quad \mathtt{A} \to \mathtt{B} : \langle U \rangle \quad | \quad \mathtt{A} \to \mathtt{B} : [l]$$

A label α shows which interaction is consumed. Since our discussion on asynchrony and parallelism applies also to protocols, we need to capture these aspects also in their semantics. Similarly to rule $\lfloor^{\mathrm{C}}|_{\textsc{Async}}\rceil$ for choreographies, rule $\lfloor^{\mathrm{G}}|_{\textsc{ACom}}\rceil$ models asynchrony in global types by allowing a sender role to proceed before the corresponding receiver has actually received the message. Rule $\lfloor^{\mathrm{G}}|_{\textsc{ABranch}}\rceil$ does the same for branching. Observe that since we are allowing an asynchronous action from inside branch G_j to take place, we restrict the branching to the choice $l_j : G_j$ in order to disable the other branches. In $\lfloor^{\mathrm{G}}|_{\textsc{Swap}}\rceil$, the relation $\simeq_{\mathcal{G}}$ for global types models parallelism and is defined similarly to $\simeq_{\mathcal{C}}$; formally, it is the smallest congruence satisfying the rules in Figure 2.7. The rules are similar to the ones for $\simeq_{\mathcal{C}}$, where conditional is now replaced by branching.

2.4.2 Type checking

We now introduce our multiparty session typing, which checks that sessions in a program (choreography without restrictions) follow the protocol specifications given as global types.

$$\frac{\{\mathtt{A},\mathtt{B}\} \asymp \{\mathtt{C},\mathtt{D}\}}{\mathtt{A} \to \mathtt{B} : \langle U\rangle;\ \mathtt{C} \to \mathtt{D} : \langle U'\rangle \simeq_{\mathcal{G}} \mathtt{C} \to \mathtt{D} : \langle U'\rangle;\ \mathtt{A} \to \mathtt{B} : \langle U\rangle} \lfloor^{\text{GS}}|_{\text{COM-COM}}\rceil$$

$$\frac{\{\mathtt{A},\mathtt{B}\} \asymp \{\mathtt{C},\mathtt{D}\}}{\mathtt{A} \to \mathtt{B} : \{l_i : \mathtt{C} \to \mathtt{D} : \langle U\rangle;\ G_i\}_{i\in I} \simeq_{\mathcal{G}} \mathtt{C} \to \mathtt{D} : \langle U\rangle;\ \mathtt{A} \to \mathtt{B} : \{l_i : G_i\}_{i\in I}} \lfloor^{\text{GS}}|_{\text{COM-BRA}}\rceil$$

$$\frac{\{\mathtt{A},\mathtt{B}\} \asymp \{\mathtt{C},\mathtt{D}\}}{\mathtt{A} \to \mathtt{B} : \{l_i : \mathtt{C} \to \mathtt{D} : \{l'_j : G_{ij}\}_{j\in J}\}_{i\in I} \simeq_{\mathcal{G}} \mathtt{C} \to \mathtt{D} : \{l'_j : \mathtt{A} \to \mathtt{B} : \{l_i : G_{ij}\}_{i\in I}\}_{j\in J}} \lfloor^{\text{GS}}|_{\text{BRA-BRA}}\rceil$$

Figure 2.7: Global Types, swap relation $\simeq_{\mathcal{G}}$.

(Unrestricted Env.)	Γ	$::=$	$\emptyset$	*(empty env.)*
		$\mid$	$\Gamma, a : G\langle\tilde{\mathtt{A}}\vert\tilde{\mathtt{B}}\rangle$	*(service)*
		$\mid$	$\Gamma, x@\mathsf{p} : S$	*(variable)*
		$\mid$	$\Gamma, X(\tilde{D}) : (\Gamma;\Delta)$	*(definition)*
		$\mid$	$\Gamma, \mathsf{p} : k[\mathtt{A}]$	*(ownership)*
(Session Env.)	Δ	$::=$	$\emptyset$	*(empty env.)*
		$\mid$	$k : G$	*(session)*

Figure 2.8: Choreography Calculus, typing environments.

We use two kinds of typing environments, the *unrestricted environments* Γ and the *session environments* Δ. Their syntax is reported in Figure 2.9.

An unrestricted environment Γ contains type information that can be reused in a type derivation. A typing $a : G\langle\tilde{\mathtt{A}}|\tilde{\mathtt{B}}\rangle$ carries the global type of a shared channel a, specifying how a session started through a has to be executed after initialisation; in the type, $\tilde{\mathtt{A}}$ are the roles of the active processes and $\tilde{\mathtt{B}}$ the roles of the service processes. In the remainder, whenever we write $a : G\langle\tilde{\mathtt{A}}|\tilde{\mathtt{B}}\rangle$ we assume that $\tilde{\mathtt{A}}$ and $\tilde{\mathtt{B}}$ are all the roles in G, formally $\{\tilde{\mathtt{A}}, \tilde{\mathtt{B}}\} = \mathsf{roles}(G)$ where roles returns the set of roles in a protocol G. We will use the knowledge of which roles are to be implemented as services to enforce that services remain available during execution. Γ also keeps the sort types of variables at processes, $x@\mathsf{p} : S$, and the environments needed to type recursive procedures, $X(\tilde{D}) : (\Gamma;\Delta)$. Finally, an *ownership typing* $\mathsf{p} : k[\mathtt{A}]$ asserts that p plays role $\mathtt{A}$ in session k. A session environment Δ records the global type of each running session k, $k : G$, tracking the protocols that sessions are expected to follow. Differently from unrestricted environments, session environments contain linear typings; specifically, each interaction inside a global type in a typing $k : G$ must be consumed exactly once. We adopt the standard assumption that typing environ-

ments are maps: a shared channel, a variable at a process, a recursive variable, or a session can appear at most once in the same environment. Typings $\mathtt{p}:k[\mathtt{A}]$ in a Γ are exceptions to this assumption: a process can appear more than once, but we can write $\Gamma, \mathtt{p}:k[\mathtt{A}]$ only if $\mathtt{p}$ is not associated to any other role in the same session k in Γ. Consequently, a process can participate in multiple sessions playing different roles, but it cannot participate in the same session with more than one role.

Typing judgements have the shape:

$$\Gamma \vdash C \triangleright \Delta$$

For the sake of presentation, we often highlight the term that we are typing as done above. Intuitively, a choreography C is well-typed provided that shared channels are used and processes play roles according to Γ, and session channels are used according to Δ. We can now present our typing rules. We discuss them below one by one.

Start. We type session starts as follows:

$$\frac{\Gamma \vdash a : G\langle \tilde{\mathtt{A}} | \tilde{\mathtt{B}} \rangle \quad \Gamma, \mathsf{init}(\widetilde{\mathtt{p}[\mathtt{A}]}, \widetilde{\mathtt{q}[\mathtt{B}]}, k) \vdash C \triangleright \Delta, k:G \quad \tilde{\mathtt{q}} \notin \Gamma}{\Gamma \vdash \widetilde{\mathtt{p}[\mathtt{A}]}\ \mathbf{start}\ \widetilde{\mathtt{q}[\mathtt{B}]} : a(k); C \triangleright \Delta} \lfloor^{\mathrm{T}}|_{\textsc{Start}}\rceil$$

Rule $\lfloor^{\mathrm{T}}|_{\textsc{Start}}\rceil$ types a *(start)* term by checking that, in the subterm C, session k is used according to the type G of the shared channel a used for creating the session. Each process in the *(start)* term is given ownership of the role declared for it in the created session k by using the function init, which returns a set of ownership assignments as formalised below:

$$\mathsf{init}(\widetilde{\mathtt{p}[\mathtt{A}]}, k) = \left\{ \mathtt{q}:k[\mathtt{B}] \quad | \quad \mathtt{q}[\mathtt{B}] \in \widetilde{\mathtt{p}[\mathtt{A}]} \right\}$$

We abuse the notation $\tilde{\mathtt{q}} \notin \Gamma$ for checking that no process in $\tilde{\mathtt{q}}$ appears in Γ, ensuring the freshness of the newly created service processes.

Communication. The typing rule for communications is:

$$\frac{\Gamma \vdash \mathtt{p}[\mathtt{A}] \texttt{->} \mathtt{q}[\mathtt{B}] : k \quad \Gamma \vdash e@\mathtt{p} : S \quad \Gamma, x@\mathtt{q} : S \vdash C \triangleright \Delta, k:G}{\Gamma \vdash \mathtt{p}[\mathtt{A}].e \texttt{->} \mathtt{q}[\mathtt{B}].x : k; C \triangleright \Delta, k:\mathtt{A} \texttt{->} \mathtt{B} : \langle S \rangle; G} \lfloor^{\mathrm{T}}|_{\textsc{Com}}\rceil$$

Above, we check that the communication is expected by the protocol for session k in the session environment. Furthermore, the processes performing the communication must own their respective roles in the session. This is obtained by the judgement $\Gamma \vdash \mathtt{p}[\mathtt{A}] \texttt{->} \mathtt{q}[\mathtt{B}] : k$, which reads "an interaction between process $\mathtt{p}$ playing role $\mathtt{A}$ and process $\mathtt{q}$ playing role $\mathtt{B}$ over session k respects the channel ownerships in Γ". We formalise this judgement with a system for *ownership typing*, whose only rule is the following one:

$$\frac{\Gamma \vdash \mathtt{p}:k[\mathtt{A}], \mathtt{q}:k[\mathtt{B}]}{\Gamma \vdash \mathtt{p}[\mathtt{A}] \texttt{->} \mathtt{q}[\mathtt{B}] : k} \lfloor^{\mathrm{O}}|_{\textsc{Com}}\rceil$$

Basically, a judgement $\Gamma \vdash \mathtt{p}[\mathtt{A}] \texttt{->} \mathtt{q}[\mathtt{B}] : k$ is valid if and only if the ownership typings $\mathtt{p}:k[\mathtt{A}]$ and $\mathtt{q}:k[\mathtt{B}]$ are in Γ[3]. Rule $\lfloor^{\mathrm{T}}|_{\textsc{Com}}\rceil$ also checks that the type of the expression sent

[3]For now, ownership typing is straightforward and its reasoning could be easily embedded directly in the typing rules for choreographies. However, defining it as a separate system will allow us to extend ownership reasoning elegantly in the rest of our presentation, for handling runtime terms in § 2.4.3 and § 3.4.

by the sender is the expected carried type S in the protocol of the session. Finally, the continuation C is typed by assigning the type S to the variable used by the receiver in the communication.

Selection. We type selections with the following rule:

$$\frac{\Gamma \vdash \mathtt{p[A]} \mathtt{->} \mathtt{q[B]} : k \quad \Gamma \vdash C \triangleright \Delta, k{:}G_j \quad j \in I}{\Gamma \vdash \mathtt{p[A]} \mathtt{->} \mathtt{q[B]} : k[l_j]; C \triangleright \Delta, k{:}\mathtt{A} \mathtt{->} \mathtt{B} : \{l_i : G_i\}_{i \in I}} \lfloor^{\mathrm{T}}|_{\textsc{Sel}}\rceil$$

A selection of a label l_j on session k is well-typed if the involved processes own their respective roles in the session and the label is in those allowed by the protocol of session k ($j \in I$). The continuation C must then implement the selected continuation G_j on session k.

Delegation. The typing rule for session delegations is:

$$\frac{\Gamma \vdash \mathtt{p[A]} \mathtt{->} \mathtt{q[B]} : k \quad \Gamma, \mathtt{q}{:}k'[\mathtt{C}] \vdash C \triangleright \Delta, k{:}G, k'{:}G'}{\Gamma, \mathtt{p}{:}k'[\mathtt{C}] \vdash \mathtt{p[A]} \mathtt{->} \mathtt{q[B]} : k\langle k'[\mathtt{C}]\rangle; C \triangleright \Delta, k{:}\mathtt{A} \mathtt{->} \mathtt{B} : \langle G'@\mathtt{C}\rangle; G, k'{:}G'} \lfloor^{\mathrm{T}}|_{\textsc{Del}}\rceil$$

Rule $\lfloor^{\mathrm{T}}|_{\textsc{Del}}\rceil$ addresses session delegation by transferring the ownership of role $\mathtt{C}$ in session k' from the sender process $\mathtt{p}$ to the receiver process $\mathtt{q}$. The global type G' of the delegated session must be the expected one in the protocol for session k; the latter also specifies which role $\mathtt{C}$ in session k' must be transferred.

Conditional. We type conditionals with the rule:

$$\frac{\Gamma \vdash e@\mathtt{p} : \mathbf{bool} \quad \Gamma \vdash C_1 \triangleright \Delta \quad \Gamma \vdash C_2 \triangleright \Delta}{\Gamma \vdash \mathsf{if}\ e@\mathtt{p}\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2 \triangleright \Delta} \lfloor^{\mathrm{T}}|_{\textsc{Cond}}\rceil$$

Rule $\lfloor^{\mathrm{T}}|_{\textsc{Cond}}\rceil$ is standard. First, we check that the expression used as condition can be evaluated as a boolean; then, we check that regardless of the branch chosen by process $\mathtt{p}$ all protocols are implemented as expected by the session environment Δ.

Restriction. The rule for typing a name restriction follows:

$$\frac{\Gamma \vdash C \triangleright \Delta}{\Gamma \setminus r \vdash (\boldsymbol{\nu} r)\, C \triangleright \Delta \setminus r} \lfloor^{\mathrm{T}}|_{\textsc{Res}}\rceil$$

Rule $\lfloor^{\mathrm{T}}|_{\textsc{Res}}\rceil$ is standard. Above, we abuse the notations $\Gamma \setminus r$ and $\Delta \setminus r$ for hiding all typings containing the name r from, respectively, environments Γ and Δ. Observe that if r is a process identifier, then only Γ is affected since Δ never refers to processes.

Termination. The typing rule for the terminated choreography is:

$$\frac{\mathsf{end}(\Delta)}{\Gamma \vdash \mathbf{0} \triangleright \Delta} \lfloor^{\mathrm{T}}|_{\textsc{End}}\rceil$$

Rule $\lfloor^{\mathrm{T}}|_{\textsc{End}}\rceil$ is carried over from the standard type system of global types [55]. A terminated choreography $\mathbf{0}$ is well-typed under any unrestricted environment Γ and session environment Δ, provided that each session k typed in Δ has type end, denoting that its protocol is terminated; the latter condition is checked in the premise of rule $\lfloor^{\mathrm{T}}|_{\textsc{End}}\rceil$ using the auxiliary predicate $\mathsf{end}(\Delta)$. Formally, the predicate $\mathsf{end}(\Delta)$ is true if and only if for all typings $k{:}G$ in Δ, we have that $G = \mathsf{end}$.

Recursion. We type definitions of recursive procedures and their calls with the following respective rules:

$$\frac{\Gamma, X(\tilde{D}):(\Gamma';\Delta') \vdash C \triangleright \Delta \quad \Gamma', X(\tilde{D}):(\Gamma';\Delta') \vdash C' \triangleright \Delta' \quad \Gamma'|_{\mathsf{sha}} \subseteq \Gamma}{\Gamma \vdash \mathsf{def}\, X(\tilde{D}) = C' \,\mathsf{in}\, C \triangleright \Delta} \lfloor^{\mathrm{T}}|_{\textsc{Def}}\rceil$$

$$\frac{\begin{array}{c}\mathsf{end}(\Delta) \quad \Gamma'' \vdash x_{ij}@\mathsf{p}_i : S_{ij} \quad \Gamma \vdash e_{ij}@\mathsf{p}_i : S_{ij} \quad \Gamma' \subseteq \Gamma \\ D = \mathsf{p}_1(\tilde{x_1}, \tilde{k_1}), \ldots, \mathsf{p}_n(\tilde{x_n}, \tilde{k_n}) \quad E = \mathsf{p}_1(\tilde{e_1}, \tilde{k_1}), \ldots, \mathsf{p}_n(\tilde{e_n}, \tilde{k_n})\end{array}}{\Gamma, X(\tilde{D}):(\Gamma', \Gamma'';\Delta') \vdash X\langle \tilde{E} \rangle \triangleright \Delta, \Delta'} \lfloor^{\mathrm{T}}|_{\textsc{Call}}\rceil$$

Rule $\lfloor^{\mathrm{T}}|_{\textsc{Def}}\rceil$ types a recursive procedure and the choreography in which it is used. We assume that we can write $X(\tilde{D}) : (\Gamma;\Delta)$, where $D = \mathsf{p}_1(\tilde{x_1}, \tilde{k_1}), \ldots, \mathsf{p}_n(\tilde{x_n}, \tilde{k_n})$, only when the following conditions hold:

(i) $\mathsf{q}:k[\mathtt{A}] \in \Gamma$ iff $\exists i \in [1,n].\, \mathsf{q} = \mathsf{p}_i \wedge k \in \tilde{k_i}$

(ii) $y@\mathsf{q}:S \in \Gamma$ iff $\exists i \in [1,n].\, \mathsf{q} = \mathsf{p}_i \wedge y \in \tilde{x_i}$

(iii) $\mathsf{dom}(\Delta) = \bigcup_{i\in[1,n]} \tilde{k_i}$

Above, we check that all the processes and sessions used for typing a definition body have been declared as parameters. The definition body can also use all the shared channels and other previously defined procedures that are in Γ ($\Gamma'|_{\mathsf{sha}} \subseteq \Gamma$, where $\Gamma'|_{\mathsf{sha}}$ denotes the subset of all shared channel and definition typings in Γ'). Rule $\lfloor^{\mathrm{T}}|_{\textsc{Call}}\rceil$ is standard, as it simply checks that all the expressions used for invoking the procedure have the same types as their respective parameter declarations in the procedure definition.

Example 2.4.1 (Choreography Typing)**.** The protocol in Example 2.2.1 types the shared channel a in Example 2.2.2.

2.4.3 Runtime typing

For showing that our typing discipline is sound (well-typed programs never go wrong), we need to extend our typing to handle the runtime behaviour of our choreography calculus. Specifically, we need to deal with two issues: asynchronous delegations and parallelism.

2.4.3.1 Typing Asynchronous Delegations

Asynchronous delegations, i.e., delegations executed using rule $\lfloor^{\mathrm{C}}|_{\textsc{Async}}\rceil$, introduce the need for a more refined typing of session ownerships that takes asynchrony into account when dealing with runtime terms. Formally, we check runtime choreographies with new judgements of the following form:

$$\Gamma;\Sigma \vdash C \triangleright \Delta$$

Above, the new environment Σ, dubbed *asynchrony environment*, contains information about sessions that have been delegated by rule $\lfloor^{\mathrm{C}}|_{\textsc{Async}}\rceil$. Asynchrony environments contain

$$\begin{array}{lrcll} \textit{(Asynchrony Env.)} & \Sigma & ::= & \emptyset & \textit{(empty env.)} \\ & & \mid & \Sigma, \mathtt{p} : k[\mathtt{A}] & \textit{(async. delegation)} \end{array}$$

Figure 2.9: Choreography Calculus, asynchrony typing environment Σ.

session ownership information, which we will use to record ownership transfers by means of asynchronous delegations:

We discuss this aspect with an example. Consider the following choreography:

$$C \;=\; \underbrace{\mathtt{p}[\mathtt{A}].e \texttt{ -> } \mathtt{q}[\mathtt{B}].x : k;}_{\eta_1} \;\; \underbrace{\mathtt{p}[\mathtt{C}] \texttt{ -> } \mathtt{r}[\mathtt{D}] : k'\langle k[\mathtt{A}]\rangle;}_{\eta_2} \;\; \underbrace{\mathtt{r}[\mathtt{A}].e' \texttt{ -> } \mathtt{s}[\mathtt{E}].y : k}_{\eta_3}$$

By rule $\lfloor {}^{\mathrm{C}} | {}_{\textsc{Async}} \rceil$, the choreography C may execute η_2 before η_1 and reduce to $\eta_1; \eta_3$. When typing $\eta_1; \eta_3$, we need to remember the delegation performed by η_2 using Σ, since we cannot construct an environment Γ containing ownership information for typing both η_1 and η_3 (the latter would require Γ to contain both $\mathtt{p} : k[\mathtt{A}]$ and $\mathtt{r} : k[\mathtt{A}]$, which are in contradiction). Therefore, the choreography $\eta_1; \eta_3$ would be well-typed with $\Sigma = \mathtt{p} : k[\mathtt{A}]$ but not with $\Sigma = \emptyset$. Parallelism can actually make things worse, e.g., $\eta_3; \eta_1$, a swap of $\eta_1; \eta_3$, is clearly still safe at endpoint, since the output in η_1 has already been executed and thus $\mathtt{p}$ will no longer use k.

We extend our type system to gracefully handle cases such as the one described above with two modifications. First, we change rule $\lfloor {}^{\mathrm{O}} | {}_{\textsc{Com}} \rceil$ to consider asynchronous delegations when typing session ownerships by using Σ:

$$\frac{(\Gamma \vdash \mathtt{p} : k[\mathtt{A}] \;\vee\; \Sigma \vdash \mathtt{p} : k[\mathtt{A}]) \quad \Gamma \vdash \mathtt{q} : k[\mathtt{B}] \quad \Sigma \not\vdash \mathtt{q} : k[\mathtt{B}]}{\Gamma \vdash \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{q}[\mathtt{B}] : k} \; \lfloor {}^{\mathrm{O}} | {}_{\textsc{Com}} \rceil$$

Above, we are now allowing the sender process $\mathtt{p}$ to play role $\mathtt{A}$ in a session k if it either owns the role ($\Gamma \vdash \mathtt{p} : k[\mathtt{A}]$) or it has asynchronously delegated the role to another process ($\Sigma \vdash \mathtt{p} : k[\mathtt{A}]$). For the receiver process $\mathtt{q}$, we require instead that it owns role $\mathtt{B}$ in session k ($\Gamma \vdash \mathtt{q} : k[\mathtt{B}]$) and that it has not asynchronously delegated it ($\Sigma \not\vdash \mathtt{q} : k[\mathtt{B}]$). The difference between the checks on Σ for, respectively, the sender and receiver processes reflects that asynchrony is a property of sending actions only. The second modification we need is for rule $\lfloor {}^{\mathrm{T}} | {}_{\textsc{Del}} \rceil$, which we change as follows:

$$\frac{\begin{array}{c} \Gamma; \Sigma \vdash \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{q}[\mathtt{B}] : k \quad \mathtt{p} : k'[\mathtt{C}] \notin \Sigma \quad G' \simeq_{\mathcal{G}} G'' \\ \Gamma, \mathtt{q} : k'[\mathtt{C}]; \Sigma \setminus \mathtt{q} : k'[\mathtt{C}] \;\vdash\; C \;\triangleright\; \Delta, k : G, k' : G' \end{array}}{\Gamma, \mathtt{p} : k'[\mathtt{C}]; \Sigma \;\vdash\; \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{q}[\mathtt{B}] : k\langle k'[\mathtt{C}]\rangle; C \;\triangleright\; \Delta, k : \mathtt{A} \texttt{ -> } \mathtt{B} : \langle G''@\mathtt{C}\rangle; G, k' : G'} \; \lfloor {}^{\mathrm{T}} | {}_{\textsc{Del}} \rceil$$

The new version of rule $\lfloor {}^{\mathrm{T}} | {}_{\textsc{Del}} \rceil$ is similar to the previous one: the only additions are the usage of environment Σ and the check $k'[\mathtt{C}] \notin \Sigma$. The latter ensures that $\mathtt{C}$ is played only by one process, in order to avoid races on the receiving of messages for $\mathtt{C}$ in session k.

2.4.3.2 Typing Parallelism

Consider the following protocol:

$$G \;=\; \mathtt{A} \texttt{ -> } \mathtt{B} : \langle \mathbf{int} \rangle; \quad \mathtt{C} \texttt{ -> } \mathtt{D} : \langle \mathbf{string} \rangle$$

which can correctly type public channel a in the choreography:

$$C \quad = \quad \begin{array}{l} \mathtt{p}[\mathtt{A}], \mathtt{q}[\mathtt{B}] \ \mathbf{start} \ \mathtt{r}[\mathtt{C}], \mathtt{s}[\mathtt{D}] : a(k); \\ \mathtt{p}[\mathtt{A}].e \texttt{ -> } \mathtt{q}[\mathtt{B}].x : k; \quad \mathtt{r}[\mathtt{C}].e' \texttt{ -> } \mathtt{s}[\mathtt{D}].y : k \end{array}$$

Since $\{\mathsf{p}, \mathsf{q}\} \cap \{\mathsf{r}, \mathsf{s}\} = \emptyset$, we can swap C with C' ($C \simeq_{\mathcal{C}} C'$) such that:

$$C' \quad = \quad \begin{array}{l} \mathtt{p}[\mathtt{A}], \mathtt{q}[\mathtt{B}] \ \mathbf{start} \ \mathtt{r}[\mathtt{C}], \mathtt{s}[\mathtt{D}] : a(k); \\ \mathtt{r}[\mathtt{C}].e' \texttt{ -> } \mathtt{s}[\mathtt{D}].y : k; \quad \mathtt{p}[\mathtt{A}].e \texttt{ -> } \mathtt{q}[\mathtt{B}].x : k \end{array}$$

Public channel a in C' does not have type G anymore, but C' is clearly still correct since we can easily follow its swap from C in type G using swapping for global types:

$$G \quad \simeq_{\mathcal{G}} \quad \mathtt{C} \texttt{ -> } \mathtt{D} : \langle \mathbf{string} \rangle; \quad \mathtt{A} \texttt{ -> } \mathtt{B} : \langle \mathbf{int} \rangle$$

Our type system, however, does not deal with swappings in global types, and would reject C'. We made this choice so that programmers do not need to think about the swap relations when writing programs, which could lead to confusing error messages in implementations. However, at runtime, we must consider swaps in order to preserve well-typedness wrt reductions (Subject Reduction). Therefore, our runtime type system augments rules $\lfloor^{\mathrm{T}}|_{\textsc{start}}\rceil$ and $\lfloor^{\mathrm{T}}|_{\textsc{del}}\rceil$ for typing up to $\simeq_{\mathcal{G}}$.

2.4.3.3 Runtime Typing Rules

We update all the rules for typing choreographies, by following the aforementioned observations on asynchrony and parallelism. The rules are reported in Figure 2.10. Each rule is extended to handle the asynchrony environment Σ. For recursive procedures, we also extend their typing to handle asynchronous delegations by adding the possibility to specify an asynchrony environment, formally $X(\tilde{D}) : (\Gamma; \Sigma; \Delta)$ where we assume that the following condition is respected, for $D = \mathsf{p}_1(\tilde{x_1}, \tilde{k_1}), \ldots, \mathsf{p}_n(\tilde{x_n}, \tilde{k_n})$:

$$\mathsf{q} : k[\mathtt{A}] \in \Sigma \quad \text{iff} \quad \exists i \in [1, n].\ \mathsf{q} = \mathsf{p}_i \ \wedge \ k \in \tilde{k_i}$$

2.4.4 Properties

Hereby, we present the main properties of our type system. In the remainder, we write $\Delta \xrightarrow{k:\alpha} \Delta'$ whenever (i) $k : G$ is in Δ, (ii) $G \xrightarrow{\alpha} G'$, and (iii) Δ' is the result of substituting $k : G$ in Δ with $k : G'$. Also, we write $\Delta \simeq_{\mathcal{G}} \Delta'$ iff $\mathsf{dom}(\Delta) = \mathsf{dom}(\Delta')$ and $\Delta(k) \simeq_{\mathcal{G}} \Delta'(k)$ for all $k \in \mathsf{dom}(\Delta)$.

Lemma 2.4.1 (Subject Swap)**.** *Assume* $\Gamma; \Sigma \vdash C \triangleright \Delta$*; then,* $C \simeq_{\mathcal{C}} C'$ *implies* $\Gamma; \Sigma \vdash C' \triangleright \Delta'$ *for some* Δ' *such that* $\Delta \simeq_{\mathcal{G}} \Delta'$.

Proof. By induction on the rules that define $\simeq_{\mathcal{C}}$ (Figure 2.2), showing that for each rule there is another corresponding rule in the definition of $\simeq_{\mathcal{G}}$ (Figure 2.7) for performing the appropriate swapping in Δ. □

Lemma 2.4.1 states that if a choreography C is well-typed, then each swapping $C \simeq_{\mathcal{C}} C'$ can be followed by the typing system by swapping the global types used for typing the sessions in C. We use this Lemma for proving the following result of subject reduction.

Next, we formalise the soundness of our typing system by establishing a relationship between the semantics of choreographies and that of global types.

$$\frac{\Gamma;\Sigma \vdash a : G\langle \tilde{\mathtt{A}}|\tilde{\mathtt{B}}\rangle \quad G \simeq_{\mathcal{G}} G' \quad \tilde{\mathtt{q}} \notin \Gamma;\Sigma \quad \Gamma, \mathsf{init}(\{\widetilde{\mathtt{p}[\mathtt{A}]}, \widetilde{\mathtt{q}[\mathtt{B}]}\}, k);\Sigma \vdash C \triangleright \Delta, k:G'}{\Gamma;\Sigma \vdash \widetilde{\mathtt{p}[\mathtt{A}]}\ \mathbf{start}\ \widetilde{\mathtt{q}[\mathtt{B}]} : a(k); C \triangleright \Delta} \lfloor^{\mathrm{T}}|_{\textsc{Start}}\rceil$$

$$\frac{\Gamma;\Sigma \vdash \mathtt{p}[\mathtt{A}] \texttt{->} \mathtt{q}[\mathtt{B}] : k \quad \Gamma \vdash e@\mathtt{p} : S \quad \Gamma, x@\mathtt{q}:S;\Sigma \vdash C \triangleright \Delta, k:G}{\Gamma;\Sigma \vdash \mathtt{p}[\mathtt{A}].e \texttt{->} \mathtt{q}[\mathtt{B}].x : k; C \triangleright \Delta, k:\mathtt{A} \texttt{->} \mathtt{B} : \langle S\rangle; G} \lfloor^{\mathrm{T}}|_{\textsc{Com}}\rceil$$

$$\frac{\Gamma;\Sigma \vdash \mathtt{p}[\mathtt{A}] \texttt{->} \mathtt{q}[\mathtt{B}] : k \quad \Gamma;\Sigma \vdash C \triangleright \Delta, k:G_j \quad j \in I}{\Gamma;\Sigma \vdash \mathtt{p}[\mathtt{A}] \texttt{->} \mathtt{q}[\mathtt{B}] : k[l_j]; C \triangleright \Delta, k:\mathtt{A} \texttt{->} \mathtt{B} : \{l_i : G_i\}_{i\in I}} \lfloor^{\mathrm{T}}|_{\textsc{Sel}}\rceil$$

$$\frac{\Gamma;\Sigma \vdash \mathtt{p}[\mathtt{A}] \texttt{->} \mathtt{q}[\mathtt{B}] : k \quad \mathtt{p}:k'[\mathtt{C}] \notin \Sigma \quad G' \simeq_{\mathcal{G}} G'' \quad \Gamma, \mathtt{q}:k'[\mathtt{C}];\Sigma \setminus \mathtt{q}:k'[\mathtt{C}] \vdash C \triangleright \Delta, k:G, k':G'}{\Gamma, \mathtt{p}:k'[\mathtt{C}];\Sigma \vdash \mathtt{p}[\mathtt{A}] \texttt{->} \mathtt{q}[\mathtt{B}] : k\langle k'[\mathtt{C}]\rangle; C \triangleright \Delta, k:\mathtt{A} \texttt{->} \mathtt{B} : \langle G''@\mathtt{C}\rangle; G, k':G'} \lfloor^{\mathrm{T}}|_{\textsc{Del}}\rceil$$

$$\frac{\Gamma \vdash e@\mathtt{p} : \mathbf{bool} \quad \Gamma;\Sigma \vdash C_1 \triangleright \Delta \quad \Gamma;\Sigma \vdash C_2 \triangleright \Delta}{\Gamma;\Sigma \vdash \mathsf{if}\ e@\mathtt{p}\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2 \triangleright \Delta} \lfloor^{\mathrm{T}}|_{\textsc{Cond}}\rceil$$

$$\frac{\Gamma;\Sigma \vdash C \triangleright \Delta}{\Gamma \setminus r;\Sigma \setminus r \vdash (\boldsymbol{\nu} r)\, C \triangleright \Delta \setminus r} \lfloor^{\mathrm{T}}|_{\textsc{Res}}\rceil \qquad \frac{\mathsf{end}(\Delta)}{\Gamma;\Sigma \vdash \mathbf{0} \triangleright \Delta} \lfloor^{\mathrm{T}}|_{\textsc{End}}\rceil$$

$$\frac{\Gamma, X(\tilde{D}):(\Gamma';\Sigma';\Delta');\Sigma \vdash C \triangleright \Delta \quad \Gamma', X(\tilde{D}):(\Gamma';\Sigma';\Delta');\Sigma' \vdash C' \triangleright \Delta' \quad \Gamma'|_{\mathsf{sha}} \subseteq \Gamma}{\Gamma;\Sigma \vdash \mathsf{def}\ X(\tilde{D}) = C'\ \mathsf{in}\ C \triangleright \Delta} \lfloor^{\mathrm{T}}|_{\textsc{Def}}\rceil$$

$$\frac{\begin{array}{c}\mathsf{end}(\Delta) \quad \Gamma'' \vdash x_{ij}@\mathsf{p}_i : S_{ij} \quad \Gamma \vdash e_{ij}@\mathsf{p}_i : S_{ij} \quad \Gamma' \subseteq \Gamma \quad \Sigma' \subseteq \Sigma \\ D = \mathsf{p}_1(\tilde{x_1}, \tilde{k_1}), \ldots, \mathsf{p}_n(\tilde{x_n}, \tilde{k_n}) \quad E = \mathsf{p}_1(\tilde{e_1}, \tilde{k_1}), \ldots, \mathsf{p}_n(\tilde{e_n}, \tilde{k_n})\end{array}}{\Gamma, X(\tilde{D}):(\Gamma', \Gamma'';\Sigma';\Delta') \vdash X\langle \tilde{E}\rangle \triangleright \Delta, \Delta'} \lfloor^{\mathrm{T}}|_{\textsc{Call}}\rceil$$

Figure 2.10: Choreography Calculus, runtime typing rules.

$$\frac{\Gamma;\Sigma \vdash \mathtt{p}[\mathtt{A}] \to \mathtt{q}[\mathtt{B}] : k \quad \Gamma \vdash v : S \quad \Gamma \vdash x@\mathtt{q} : S}{\Gamma;\Sigma \vdash \mathtt{p}[\mathtt{A}].v \to \mathtt{q}[\mathtt{B}].x : k \triangleright k : \mathtt{A} \to \mathtt{B} : \langle S \rangle} \lfloor^{\mathrm{L}}|_{\mathrm{COM}}\rceil$$

$$\frac{\Gamma;\Sigma \vdash \mathtt{p}[\mathtt{A}] \to \mathtt{q}[\mathtt{B}] : k}{\Gamma;\Sigma \vdash \mathtt{p}[\mathtt{A}] \to \mathtt{q}[\mathtt{B}] : k[l] \triangleright k : \mathtt{A} \to \mathtt{B} : [l]} \lfloor^{\mathrm{L}}|_{\mathrm{SEL}}\rceil$$

$$\frac{\Gamma;\Sigma \vdash \mathtt{p}[\mathtt{A}] \to \mathtt{q}[\mathtt{B}] : k \quad \Gamma \vdash \mathtt{p} : k'[\mathtt{C}]}{\Gamma;\Sigma \vdash \mathtt{p}[\mathtt{A}] \to \mathtt{q}[\mathtt{B}] : k\langle k'[\mathtt{C}]\rangle \triangleright k : \mathtt{A} \to \mathtt{B} : \langle G@\mathtt{C}\rangle} \lfloor^{\mathrm{L}}|_{\mathrm{DEL}}\rceil$$

$$\frac{\Gamma;\Sigma \vdash \lambda \triangleright k : \alpha}{\Gamma;\Sigma \vdash (\boldsymbol{\nu} r)\,\lambda \triangleright k : \alpha} \lfloor^{\mathrm{L}}|_{\mathrm{RES}}\rceil \qquad \text{(no rule for } \tau@\mathtt{p}\text{)}$$

Figure 2.11: Choreography Calculus, label typing.

Theorem 2.4.2 (Subject Reduction)**.** *Assume* $\Gamma;\Sigma \vdash C \triangleright \Delta$*; then,* $C \xrightarrow{\lambda} C'$ *implies that:*

- *if* $\lambda = (\boldsymbol{\nu}\tilde{r})\,\tau@\mathtt{p}$*, then there exists* Δ' *such that* $\Gamma;\Sigma \vdash C' \triangleright \Delta'$ *and* $\Delta \simeq_{\mathcal{G}} \Delta'$*;*
- *otherwise, there exist* Γ'*,* Σ' *and* Δ' *such that* $\Gamma';\Sigma' \vdash C' \triangleright \Delta'$*, where* $\Delta \xrightarrow{k:\alpha} \Delta'$ *and* $\Gamma;\Sigma \vdash \lambda \triangleright k : \alpha$*.*

Proof. See Appendix A.1. □

Theorem 2.4.2 states that if a choreography is well-typed and makes a reduction, then the reductum is still well-typed; furthermore, if the reduction was a communication on a session, then the corresponding global type of the session can also make a corresponding reduction. The correspondence between the labels of the choreography and global type reductions is formalised by the label typing judgement $\Gamma;\Sigma \vdash \lambda \triangleright k : \alpha$, whose rules are reported in Figure 2.11.

2.4.5 Type Inference

In our model, choreographies and global types share deep syntactic and semantic correspondences. We exploit this aspect to perform the *type inference* of public channels; therefore, we can automatically extract the protocols that a choreography implements with its sessions.

We define type inference following the standard methodology [88]. First, we define subtyping as set inclusion on branching labels, similarly to the covariant typing of rule $\lfloor^{\mathrm{T}}|_{\mathrm{SEL}}\rceil$. Then, we modify our rules to determine the principal type of a choreography. In the remainder of this section we shall consider only the typing of programs, therefore omitting the asynchrony environment Σ (which is used only for typing runtime terms). Moreover, we only refer to the shared names of Γ and ignore normal variables and recursion variables when talking about principal typing.

Subtyping is formally defined as follows.

$$\frac{I \subseteq J \quad \forall i \in I.\, G_i \ll G'_i}{\mathtt{A} \to \mathtt{B} : \{l_i : G_i\}_{i\in I} \ll \mathtt{A} \to \mathtt{B} : \{l_i : G'_i\}_{i\in J}} \lfloor^{\text{Sub}}|_{\text{Branch}}\rceil$$

$$\frac{G_1 \ll G'_1 \qquad G_2 \ll G'_2}{\mathtt{A} \to \mathtt{B} : \langle G_1 @ \mathtt{A}\rangle; G_2 \ll \mathtt{A} \to \mathtt{B} : \langle G'_1 @ \mathtt{A}\rangle; G'_2} \lfloor^{\text{Sub}}|_{\text{Del}}\rceil$$

$$\frac{(\, G \approx G' \vee G \simeq_{\mathcal{G}} G'\,) \quad G \ll G''}{G' \ll G''} \lfloor^{\text{Sub}}|_{\text{Eq}}\rceil \qquad \frac{\mathsf{end} \approx G}{\mathsf{end} \ll G} \lfloor^{\text{Sub}}|_{\text{End}}\rceil$$

Figure 2.12: Global Types, subtyping.

Definition 2.4.3 (Subtyping)**.** The subtyping $\ll$ is the smallest relation over closed and unfolded global types satisfying the rules reported in Figure 2.12. We extend $\ll$ to set inclusion and point-wise to the typing of shared names and sessions. Given two types G and G', we denote their least upper bound (lub) wrt $\ll$ with $G \triangledown G'$.

Remark 2.4.4 (Algorithmic Subtyping). Subtyping is algorithmically checkable. Given the simplicity of global types, checking the subtyping relation is decidable since (i) swapping is decidable (one-time unfolding of recursion is enough given that global types are regular trees) and (ii) global types are regular trees hence recursive types can be standardly dealt with as shown in [55] (cf. [88]). In particular, our swapping relation $\simeq_{\mathcal{G}}$ plays a similar rôle as the swap of outputs in [75].

Proposition 2.4.1.1 (Subsumption)**.** *Let* $\Gamma \ll \Gamma'$ *and* $\Delta \ll \Delta'$*. Then,* $\Gamma \vdash C \triangleright \Delta$ *implies* $\Gamma' \vdash C \triangleright \Delta'$*.*

Proof. Immediate by the definitions of subtyping and rule $\lfloor^{\text{T}}|_{\text{Sel}}\rceil$. □

We are now able to show that our type system has principal typing: this will follow by subsumption and minimal typing wrt subtyping.

Proposition 2.4.1.2 (Existence of Minimal Typing)**.** *Let* $\Gamma \vdash C \triangleright \emptyset$*; then, there exists* Γ_0 *such that* $\Gamma_0 \vdash C \triangleright \emptyset$ *and whenever* $\Gamma' \vdash C \triangleright \emptyset$ *for some* Γ'*, we have that* $\Gamma_0 \ll \Gamma'$*. The environment* Γ_0 *can be algorithmically calculated from* C *and is called the* minimal typing *of* C*.*

Proof. The proof is standard and consists in constructing the minimal typing system defining $\Gamma \vdash_{\mathsf{min}} C \triangleright \emptyset$, whose rules are reported in Figure 2.13. Note that we focus on the reconstruction of global types, and leave the reconstruction of variable types undefined since it is entirely standard (see [88]). To this end, the main modification is to change the rule for selections, $\lfloor^{\text{Min}}|_{\text{Sel}}\rceil$, to type a selection with a singleton branching global type and then use subtyping to determine the least upper bound of branch types in rules $\lfloor^{\text{Min}}|_{\text{Start2}}\rceil$ and $\lfloor^{\text{Min}}|_{\text{Def}}\rceil$ (for shared names) and $\lfloor^{\text{Min}}|_{\text{Cond}}\rceil$ for conditionals. In the rules $\lfloor^{\text{Min}}|_{\text{End}}\rceil$ and $\lfloor^{\text{Min}}|_{\text{Call}}\rceil$ we use some auxiliary information, which can be easily obtained through a standard preliminary top-down visit of the choreography syntax tree (cf. [31]). Specifically, vars and $\mathsf{ownerships}$ are respectively the variable and the ownership typings of the choreography whose type is being inferred. We denote with $\mathsf{vars}(X)$ and $\mathsf{ownerships}(X)$ the

same information, but obtained by inspecting the body of the inner-most recursive procedure X instead. The set sessions contains exactly all the session identifiers used in the choreography whose type we are inferring. Finally, $\mathsf{solve}(\Delta, \mathbf{t})$ solves the equation $\mathbf{t} = G$ for all global types contained in Δ. □

$$\frac{\Gamma, \mathsf{init}(\{\widetilde{\mathsf{p}[\mathtt{A}]}, \widetilde{\mathsf{q}[\mathtt{B}]}\}, k) \vdash_{\mathsf{min}} C \triangleright \Delta, k{:}G \quad \tilde{\mathsf{q}} \notin \Gamma \quad a \notin \Gamma}{\Gamma, a{:}G\langle \tilde{\mathtt{A}} | \tilde{\mathtt{B}} \rangle \vdash_{\mathsf{min}} \widetilde{\mathsf{p}[\mathtt{A}]}\ \mathbf{start}\ \widetilde{\mathsf{q}[\mathtt{B}]} : a(k); C \triangleright \Delta} \lfloor^{\text{MIN}}|_{\text{START1}}\rceil$$

$$\frac{\Gamma, \mathsf{init}(\{\widetilde{\mathsf{p}[\mathtt{A}]}, \widetilde{\mathsf{q}[\mathtt{B}]}\}, k), a{:}G\langle \tilde{\mathtt{A}} | \tilde{\mathtt{B}} \rangle \vdash_{\mathsf{min}} C \triangleright \Delta, k{:}G' \quad \tilde{\mathsf{q}} \notin \Gamma}{\Gamma, a{:}(G \triangledown G')\langle \tilde{\mathtt{A}} | \tilde{\mathtt{B}} \rangle \vdash_{\mathsf{min}} \widetilde{\mathsf{p}[\mathtt{A}]}\ \mathbf{start}\ \widetilde{\mathsf{q}[\mathtt{B}]} : a(k); C \triangleright \Delta} \lfloor^{\text{MIN}}|_{\text{START2}}\rceil$$

$$\frac{\Gamma \vdash \mathsf{p}[\mathtt{A}] \to \mathsf{q}[\mathtt{B}] : k \quad \Gamma \vdash e@\mathsf{p} : S \quad \Gamma, x@\mathsf{q} : S \vdash_{\mathsf{min}} C \triangleright \Delta, k{:}G}{\Gamma \vdash_{\mathsf{min}} \mathsf{p}[\mathtt{A}].e \to \mathsf{q}[\mathtt{B}].x : k; C \triangleright \Delta, k{:}\mathtt{A} \to \mathtt{B} : \langle S \rangle; G} \lfloor^{\text{MIN}}|_{\text{COM}}\rceil$$

$$\frac{\Gamma \vdash \mathsf{p}[\mathtt{A}] \to \mathsf{q}[\mathtt{B}] : k \quad \Gamma \vdash_{\mathsf{min}} C \triangleright \Delta, k{:}G}{\Gamma \vdash_{\mathsf{min}} \mathsf{p}[\mathtt{A}] \to \mathsf{q}[\mathtt{B}] : k[l]; C \triangleright \Delta, k{:}\mathtt{A} \to \mathtt{B} : \{l : G\}} \lfloor^{\text{MIN}}|_{\text{SEL}}\rceil$$

$$\frac{\Gamma \vdash \mathsf{p}[\mathtt{A}] \to \mathsf{q}[\mathtt{B}] : k \quad \Gamma, \mathsf{q} : k'[\mathtt{C}] \vdash_{\mathsf{min}} C \triangleright \Delta, k{:}G, k'{:}G'}{\Gamma, \mathsf{p} : k'[\mathtt{C}] \vdash_{\mathsf{min}} \mathsf{p}[\mathtt{A}] \to \mathsf{q}[\mathtt{B}] : k\langle k'[\mathtt{C}] \rangle; C \triangleright \Delta, k{:}\mathtt{A} \to \mathtt{B} : \langle G'@\mathtt{C} \rangle; G, k'{:}G'} \lfloor^{\text{MIN}}|_{\text{DEL}}\rceil$$

$$\frac{\Gamma \vdash e@\mathsf{p} : \mathbf{bool} \quad \Gamma_1 \vdash_{\mathsf{min}} C_1 \triangleright \Delta_1 \quad \Gamma_2 \vdash_{\mathsf{min}} C_2 \triangleright \Delta_2 \quad \Gamma = \Gamma_1 \triangledown \Gamma_2}{\Gamma \vdash_{\mathsf{min}} \mathsf{if}\ e@\mathsf{p}\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2 \triangleright \Delta_1 \triangledown \Delta_2} \lfloor^{\text{MIN}}|_{\text{COND}}\rceil$$

$$\frac{\Gamma \vdash_{\mathsf{min}} C \triangleright \Delta}{\Gamma \setminus r \vdash_{\mathsf{min}} (\boldsymbol{\nu} r)\, C \triangleright \Delta \setminus r} \lfloor^{\text{MIN}}|_{\text{RES}}\rceil$$

$$\frac{\Gamma = \mathsf{ownerships} \quad \Delta = \{k : \mathsf{end} \mid k \in \mathsf{sessions}\}}{\Gamma \vdash_{\mathsf{min}} \mathbf{0} \triangleright \Delta} \lfloor^{\text{MIN}}|_{\text{END}}\rceil$$

$$\frac{\begin{array}{c}\Gamma(X) = \Gamma'(X)\ \text{if}\ X \in \mathsf{dom}(\Gamma) \cap \mathsf{dom}(\Gamma') \\ \Gamma \vdash_{\mathsf{min}} C \triangleright \Delta \quad \Gamma' \vdash_{\mathsf{min}} C' \triangleright \Delta' \quad \Gamma'|_{\mathsf{sha}} \subseteq \Gamma\end{array}}{(\Gamma \triangledown \Gamma') \setminus X \vdash_{\mathsf{min}} \mathsf{def}\ X(\tilde{D}) = C'\ \mathsf{in}\ C \triangleright \mathsf{solve}(\Delta \triangledown \Delta', \mathbf{t}_X)} \lfloor^{\text{MIN}}|_{\text{DEF}}\rceil$$

$$\frac{\begin{array}{c}\Gamma = \mathsf{vars} \cup \mathsf{ownerships} \quad \tilde{k} \cup \tilde{k'} = \mathsf{sessions} \quad \tilde{k} = \bigcup_{i \in [1,n]} \tilde{k_i} \\ \Gamma' = \mathsf{vars}(X) \cup \mathsf{ownerships}(X) \quad \Gamma' \vdash x_{ij}@\mathsf{p}_i : S_{ij} \quad \Gamma \vdash e_{ij}@\mathsf{p}_i : S_{ij} \quad \Gamma' \subseteq \Gamma \\ D = \mathsf{p}_1(\tilde{x_1}, \tilde{k_1}), \ldots, \mathsf{p}_n(\tilde{x_n}, \tilde{k_n}) \quad E = \mathsf{p}_1(\tilde{e_1}, \tilde{k_1}), \ldots, \mathsf{p}_n(\tilde{e_n}, \tilde{k_n})\end{array}}{\Gamma, X(\tilde{D}) : (\Gamma'; \widetilde{k : \mathbf{t}_X}) \vdash_{\mathsf{min}} X\langle \tilde{E} \rangle \triangleright \widetilde{k : \mathbf{t}_X}, \widetilde{k' : \mathsf{end}}} \lfloor^{\text{MIN}}|_{\text{CALL}}\rceil$$

Figure 2.13: Choreography Calculus, minimal typing rules.

$$
\begin{array}{lcllll}
P, Q & ::= & \overline{a}[\tilde{\mathtt{A}}](k); P & \textit{(req)} & | & a[\mathtt{A}](k); P \quad \textit{(acc)} \\
 & | & !a[\mathtt{A}](k); P & \textit{(serv)} & | & P \mid Q \quad \textit{(par)} \\
 & | & k[\mathtt{A}]!\mathtt{B}\langle e\rangle; P & \textit{(com-s)} & | & k[\mathtt{B}]?\mathtt{A}(x); P \quad \textit{(com-r)} \\
 & | & k[\mathtt{A}]!\mathtt{B} \oplus l; P & \textit{(sel-s)} & | & k[\mathtt{B}]?\mathtt{A} \,\&\, \{l_i : P_i\}_{i \in I} \quad \textit{(branch)} \\
 & | & k[\mathtt{A}]!\mathtt{B}\langle k'[\mathtt{C}]\rangle; P & \textit{(del-s)} & | & k[\mathtt{B}]?\mathtt{A}(k'[\mathtt{C}]); P \quad \textit{(del-r)} \\
 & | & \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q & \textit{(cond)} & | & \mathbf{0} \quad \textit{(inact)} \\
 & | & \mathsf{def}\ X(\tilde{x}, \tilde{k}) = Q\ \mathsf{in}\ P & \textit{(def)} & | & X\langle \tilde{e}, \tilde{k}\rangle \quad \textit{(call)} \\
 & | & (\boldsymbol{\nu} k)\ P & \textit{(res)} & | & k : h \quad \textit{(s-queue)} \\
\\
h & ::= & m \cdot h & \textit{(queue)} & & \\
m & ::= & (\mathtt{A}, \mathtt{B}, \boldsymbol{w}) & \textit{(mesg)} & & \\
\boldsymbol{w} & ::= & v \mid l \mid k[\mathtt{A}] & \textit{(mesg-val)} & &
\end{array}
$$

Figure 2.14: Endpoint Calculus, syntax.

Corollary 2.4.1.1 (Principal Typing)**.** *Minimal typing is also principal typing.*

Example 2.4.5 (Two-buyer-helper type inference)**.** For example, from b in Example 2.2.3, we can infer the following global type:

$$
\mathtt{B} \to \mathtt{H} : \langle \mathbf{int} \rangle; \mathtt{B} \to \mathtt{H} : \left\{ \begin{array}{l} \mathit{done} : \ \mathsf{end}, \\ \quad \mathit{del} : \ \mathtt{B} \to \mathtt{H} : \langle \mathbf{int} \rangle; \\ \qquad\quad \mathtt{B} \to \mathtt{H} : \langle (\ldots \text{as Line 2 in Example 2.2.1} \ldots) @ \mathtt{B2} \rangle \end{array} \right\}
$$

2.5 Endpoint Projection and its Properties

We now address endpoint code generation. First, we recall an endpoint model that we shall use as a target language. Then, we show how to generate endpoint code for each process in a choreography and, finally, how to obtain the code for the entire system. Our code generation, the *EndPoint Projection (EPP)*, will satisfy the *EPP Theorem*, which gives a correspondence between the asynchronous semantics of choreographies and the one of endpoint terms.

2.5.1 Endpoint Model

We model endpoint code with the Endpoint Calculus, a variant of the calculus for multiparty sessions [55, 21].

2.5.1.1 Syntax

The syntax of terms in the endpoint calculus is reported in Figure 2.14. In the syntax, P, Q, ... are processes. Terms *(req)*, *(acc)* and *(serv)* support the creation of a session k (which is a bound name). Term *(req)* requests the creation of a session k with roles $\tilde{\mathtt{A}}$ through shared channel a; a requesting process is then responsible for implementing the first role in $\tilde{\mathtt{A}}$ in the new session k. A request for creating a session can be accepted on the same shared channel by an active process, term *(acc)*, or by a service process (which acts as replicated), term

(serv). Terms *(com-s)* and *(com-r)* model, respectively, the sending and the receiving of a value. Sending and receiving actions are tagged with roles, to express (i) the role played by the process in the session and (ii) which process we wish to send/receive a message to/from. Specifically, we read $k[\mathtt{A}]!\mathtt{B}\langle e\rangle$ as "from role $\mathtt{A}$ in session k, send the value of expression e to the process that plays role $\mathtt{B}$ in session k"; dually, we read $k[\mathtt{B}]?\mathtt{A}(x)$ as "receive a value for role $\mathtt{B}$ in session k from the process that plays role $\mathtt{A}$ and store it in variable x". The same reasoning applies to the other terms. Terms *(sel-s)* and *(branch)* model, respectively, the selection of a branch and the offering of some branches. Terms *(del-s)* and *(del-r)* handle session delegation, by respectively sending and receiving a session; both the session k' and the role $\mathtt{C}$ are bound in *(del-r)*, so the receiver of a delegation does not know what session and role it will receive. In term *(cond)*, a process evaluates a condition e to choose whether to continue as process P or Q. Term *(par)* models the parallel composition of processes, allowing their actions to interact. Term *(inact)* denotes a terminated process. Terms *(def)* and *(call)* model, respectively, the definition and call of a recursive procedure. The restriction term $(\boldsymbol{\nu} k)\ P$ is standard and restricts the scope of session k to process P. A session queue $k : h$ is a FIFO queue h for session k. Session queues are used to give an asynchronous semantics to communications. A message m in a queue is a triple $(\mathtt{A}, \mathtt{B}, \boldsymbol{w})$, where $\mathtt{A}$ is the role of the message sender, $\mathtt{B}$ is the intended receiver role for the message, and $\boldsymbol{w}$ is the content of the message (a value v, a label l, or a delegated channel $k[\mathtt{A}]$).

Example 2.5.1 (Two-Buyer Endpoint Implementation)**.** The process $P_{\mathsf{s}} \mid P_{\mathsf{b}_1} \mid P_{\mathsf{b}_2}$ is an endpoint implementation of Example 2.2.2. The code for each endpoint process is given below.

$$
\begin{aligned}
P_{\mathsf{s}} = \;& !a[\mathtt{S}](k);\ k[\mathtt{S}]?\mathtt{B1}(x_1);\ k[\mathtt{S}]!\mathtt{B1}\langle \mathsf{quote}(x_1)\rangle;\ k[\mathtt{S}]!\mathtt{B2}\langle \mathsf{quote}(x_1)\rangle;\\
& k[\mathtt{S}]?\mathtt{B2}\,\&\left\{\begin{array}{ll} ok: & k[\mathtt{S}]?\mathtt{B2}(x_2);\ k[\mathtt{S}]!\mathtt{B2}\langle ddate\rangle, \\ quit: & \mathbf{0} \end{array}\right\}\\
P_{\mathsf{b}_1} = \;& \overline{a}[\mathtt{B1}, \mathtt{B2}, \mathtt{S}](k);\ k[\mathtt{B1}]!\mathtt{S}\langle book\rangle;\ k[\mathtt{B1}]?\mathtt{S}(y_1);\ k[\mathtt{B1}]!\mathtt{B2}\langle \mathsf{contrib}(y_1)\rangle\\
P_{\mathsf{b}_2} = \;& a[\mathtt{B2}](k);\ k[\mathtt{B2}]?\mathtt{S}(z_1);\ k[\mathtt{B2}]?\mathtt{B1}(z_2);\\
& \mathsf{if}\ (z_1 - z_2 \leq 100)\\
& \qquad \mathsf{then}\ k[\mathtt{B2}]!\mathtt{S} \oplus ok;\ k[\mathtt{B2}]!\mathtt{S}\langle addr\rangle;\ k[\mathtt{B2}]?\mathtt{S}(z_3)\\
& \qquad \mathsf{else}\ k[\mathtt{B2}]!\mathtt{S} \oplus quit
\end{aligned}
$$

□

2.5.1.2 Endpoint Semantics

We give semantics to our endpoint model with a labelled reduction relation $\xrightarrow{\mu}$, defined as the smallest relation on processes P satisfying the rules reported in Figure 2.15. Labels, ranged over by μ, are defined as:

$$
\begin{array}{rcllll}
\mu & ::= & \tilde{\mathtt{A}}\ \mathbf{start}\ \tilde{\mathtt{B}} : a(k) & \textit{(start)} & \mid\ \tau & \textit{(internal)}\\
& \mid & !\mathtt{A} \texttt{->} \mathtt{B} : k\langle v\rangle & \textit{(com-s)} & \mid\ ?\mathtt{A} \texttt{->} \mathtt{B} : k\langle v\rangle & \textit{(com-r)}\\
& \mid & !\mathtt{A} \texttt{->} \mathtt{B} : k\langle k'[\mathtt{C}]\rangle & \textit{(del-s)} & \mid\ ?\mathtt{A} \texttt{->} \mathtt{B} : k\langle k'[\mathtt{C}]\rangle & \textit{(del-r)}\\
& \mid & !\mathtt{A} \texttt{->} \mathtt{B} : k[l] & \textit{(sel-s)} & \mid\ ?\mathtt{A} \texttt{->} \mathtt{B} : k[l] & \textit{(sel-r)}
\end{array}
$$

$$\frac{\tilde{\mathtt{C}} = \tilde{\mathtt{A}}, \tilde{\mathtt{B}} \quad \tilde{\mathtt{A}} = \mathtt{A}_1, \ldots, \mathtt{A}_n \quad \tilde{\mathtt{B}} = \mathtt{B}_1, \ldots, \mathtt{B}_m \quad R = \prod_{i \in [1,m]} !a[\mathtt{B}_i](k); Q_i}{\overline{a}[\tilde{\mathtt{C}}](k); P \mid \prod_{i \in [2,n]} a[\mathtt{A}_i](k); P_i \mid R \xrightarrow{\tilde{\mathtt{A}}\ \mathbf{start}\ \tilde{\mathtt{B}}:a(k)} (\boldsymbol{\nu} k)\left(P \mid \prod_{i \in [2,n]} P_i \mid \prod_{i \in [1,m]} Q_i \mid k : \emptyset\right) \mid R} \lfloor^{\mathrm{P}}\rceil_{\textsc{Start}}$$

$$\frac{e \downarrow v}{k[\mathtt{A}]!\mathtt{B}\langle e\rangle; P \mid k : h \xrightarrow{!\mathtt{A}\ \text{->}\ \mathtt{B}:k\langle v\rangle} P \mid k : h \cdot (\mathtt{A}, \mathtt{B}, v)} \lfloor^{\mathrm{P}}\rceil_{\textsc{Com-S}}$$

$$\frac{}{k[\mathtt{B}]?\mathtt{A}(x); P \mid k : (\mathtt{A}, \mathtt{B}, v) \cdot h \xrightarrow{?\mathtt{A}\ \text{->}\ \mathtt{B}:k\langle v\rangle} P[v/x] \mid k : h} \lfloor^{\mathrm{P}}\rceil_{\textsc{Com-R}}$$

$$\frac{}{k[\mathtt{A}]!\mathtt{B}\langle k'[\mathtt{C}]\rangle; P \mid k : h \xrightarrow{!\mathtt{A}\ \text{->}\ \mathtt{B}:k\langle k'[\mathtt{C}]\rangle} P \mid k : h \cdot (\mathtt{A}, \mathtt{B}, k'[\mathtt{C}])} \lfloor^{\mathrm{P}}\rceil_{\textsc{Del-S}}$$

$$\frac{}{k[\mathtt{B}]?\mathtt{A}(k'[\mathtt{C}]); P \mid k : (\mathtt{A}, \mathtt{B}, k'[\mathtt{C}]) \cdot h \xrightarrow{?\mathtt{A}\ \text{->}\ \mathtt{B}:k\langle k'[\mathtt{C}]\rangle} P \mid k : h} \lfloor^{\mathrm{P}}\rceil_{\textsc{Del-R}}$$

$$\frac{}{k[\mathtt{A}]!\mathtt{B} \oplus l; P \mid k : h \xrightarrow{!\mathtt{A}\ \text{->}\ \mathtt{B}:k[l]} P \mid k : h \cdot (\mathtt{A}, \mathtt{B}, l)} \lfloor^{\mathrm{P}}\rceil_{\textsc{Sel-S}}$$

$$\frac{j \in I}{k[\mathtt{B}]?\mathtt{A} \,\&\, \{l_i : P_i\}_{i \in I} \mid k : (\mathtt{A}, \mathtt{B}, l_j) \cdot h \xrightarrow{?\mathtt{A}\ \text{->}\ \mathtt{B}:k[l_j]} P_j \mid k : h} \lfloor^{\mathrm{P}}\rceil_{\textsc{Branch}}$$

$$\frac{P \xrightarrow{\mu} P'}{\mathsf{def}\ X(\tilde{x}, \tilde{k}) = Q = P\ \mathsf{in} \xrightarrow{\mu} \mathsf{def}\ X(\tilde{x}, \tilde{k}) = Q = P'\ \mathsf{in}} \lfloor^{\mathrm{P}}\rceil_{\textsc{Ctx}}$$

$$\frac{i = 1 \text{ if } e \downarrow \mathsf{true}\,,\ i = 2 \text{ otherwise}}{\mathsf{if}\ e\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 \xrightarrow{\tau} P_i} \lfloor^{\mathrm{P}}\rceil_{\textsc{Cond}} \qquad \frac{P \xrightarrow{\mu} P'}{(\boldsymbol{\nu} k)\, P \xrightarrow{(\boldsymbol{\nu} k)\,\mu} (\boldsymbol{\nu} k)\, P'} \lfloor^{\mathrm{P}}\rceil_{\textsc{Res}}$$

$$\frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q} \lfloor^{\mathrm{P}}\rceil_{\textsc{Par}} \qquad \frac{P \equiv P' \quad P' \xrightarrow{\mu} Q' \quad Q' \equiv Q}{P \xrightarrow{\mu} Q} \lfloor^{\mathrm{P}}\rceil_{\textsc{Struct}}$$

Figure 2.15: Endpoint Calculus, semantics.

$$(\boldsymbol{\nu} k)\ \mathbf{0} \equiv \mathbf{0} \qquad P \mid Q \equiv Q \mid P \qquad (P \mid Q) \mid R \equiv P \mid (Q \mid R)$$
$$(\boldsymbol{\nu} k)\ (\boldsymbol{\nu} k')\ P \equiv (\boldsymbol{\nu} k')\ (\boldsymbol{\nu} k)\ P \qquad ((\boldsymbol{\nu} k)\ P) \mid Q \equiv (\boldsymbol{\nu} k)\ (P \mid Q) \text{ if } k \notin \mathsf{fn}(Q)$$
$$k : h \cdot (\mathtt{A}, \mathtt{B}, \boldsymbol{w}) \cdot (\mathtt{C}, \mathtt{D}, \boldsymbol{w}') \cdot h' \equiv k : h \cdot (\mathtt{C}, \mathtt{D}, \boldsymbol{w}') \cdot (\mathtt{A}, \mathtt{B}, \boldsymbol{w}) \cdot h' \quad (\mathtt{A} \neq \mathtt{C} \text{ or } \mathtt{B} \neq \mathtt{D})$$
$$\mathsf{def}\ X(\tilde{x}, \tilde{k}) = P' \ \mathsf{in}\ (P \mid Q) \ \equiv\ (\mathsf{def}\ X(\tilde{x}, \tilde{k}) = P' \ \mathsf{in}\ P) \mid Q \qquad \text{if } X \notin \mathsf{fn}(Q)$$
$$\mathsf{def}\ X(\tilde{x}, \tilde{k}) = P' \ \mathsf{in}\ \mathbf{0} \ \equiv\ \mathbf{0}$$
$$\mathsf{def}\ X(\tilde{x}, \tilde{k}) = Q \ \mathsf{in}\ P[X\langle \tilde{e}, \tilde{k} \rangle] \ \equiv\ \mathsf{def}\ X(\tilde{x}, \tilde{k}) = Q \ \mathsf{in}\ P[Q[\tilde{e}/\tilde{x}]]$$

Figure 2.16: Endpoint Calculus, structural congruence $\equiv$.

Rule $\lfloor^{\mathrm{P}}|_{\textsc{Start}}\rceil$ initiates a multiparty session, by synchronising the processes willing to start it and thereafter by creating an empty session queue $k : \emptyset$. Rules $\lfloor^{\mathrm{P}}|_{\textsc{Com-S}}\rceil$, $\lfloor^{\mathrm{P}}|_{\textsc{Del-S}}\rceil$ and $\lfloor^{\mathrm{P}}|_{\textsc{Sel-S}}\rceil$ send, respectively, a value v, a session channel $k'[\mathtt{C}]$, and a label l by putting it in the queue for Symmetrically, rules $\lfloor^{\mathrm{P}}|_{\textsc{Com-R}}\rceil$ and $\lfloor^{\mathrm{P}}|_{\textsc{Del-R}}\rceil$ respectively extract a value and a session channel from a queue. Rule $\lfloor^{\mathrm{P}}|_{\textsc{Branch}}\rceil$ fetches a label l_j from the queue and then continues as process P_j. All other rules are standard. The structural congruence $\equiv$ used in rule $\lfloor^{\mathrm{P}}|_{\textsc{Struct}}\rceil$ is the smallest congruence supporting α-conversion and satisfying the rules reported in Figure 2.16.

Structural congruence allows us to permute messages with different pairs of roles in a queue. Technically, this is equivalent to having a full-duplex queue per each pair of roles [21]. In the rule for recursion unfolding (last rule in Figure 2.16), we use the notation $P[X\langle \tilde{e}, \tilde{k} \rangle]$ for indicating that $X\langle \tilde{e}, \tilde{k} \rangle$ is a subterm of P that we replace with the term $Q[\tilde{e}/\tilde{x}]$ (the substitution of the procedure parameters in its body with the expressions used in the call) in the right-hand side.

Example 2.5.2 (Two-Buyer Endpoint Semantics)**.** Applying the semantics of the Endpoint Calculus, we can prove the reduction

$$P_{\mathsf{b}_1} \mid P_{\mathsf{b}_2} \mid P_{\mathsf{s}} \xrightarrow{\mu} (\boldsymbol{\nu} k)\ (Q_{\mathsf{b}_1} \mid Q_{\mathsf{b}_2} \mid Q_{\mathsf{s}}) \mid P_{\mathsf{s}} \xrightarrow{\mu'} \ldots \xrightarrow{\mu''} P_{\mathsf{s}}$$

where the processes Q_{b_1}, Q_{b_2}, and Q_{s} are defined below.

$$\begin{array}{rcl}
Q_{\mathsf{b}_1} & = & k[\mathtt{B1}]!\mathtt{S}\langle book \rangle;\ k[\mathtt{B1}]?\mathtt{S}(y);\ k[\mathtt{B1}]!\mathtt{B2}\langle \mathsf{contrib}(y) \rangle \\
Q_{\mathsf{b}_2} & = & k[\mathtt{B2}]?\mathtt{S}(z_1);\ k[\mathtt{B2}]?\mathtt{B1}(z_2);\ \mathsf{if}\ (z_1 - z_2 \leq 100) \\
 & & \qquad \mathsf{then}\ k[\mathtt{B2}]!\mathtt{S} \oplus ok;\ k[\mathtt{B2}]!\mathtt{S}\langle addr \rangle;\ k[\mathtt{B2}]?\mathtt{S}(w) \\
 & & \qquad \mathsf{else}\ \ k[\mathtt{B2}]!\mathtt{S} \oplus quit \\
Q_{\mathsf{s}} & = & k[\mathtt{S}]?\mathtt{B1}(x);\ k[\mathtt{S}]!\mathtt{B1}\langle \mathsf{quote}(x) \rangle;\ k[\mathtt{S}]!\mathtt{B2}\langle \mathsf{quote}(x) \rangle; \\
 & & k[\mathtt{S}]?\mathtt{B2}\ \&\ \left\{ \begin{array}{ll} ok: & k[\mathtt{S}]?\mathtt{B2}(x');\ k[\mathtt{S}]!\mathtt{B2}\langle ddate \rangle \\ quit: & \mathbf{0} \end{array} \right\}
\end{array}$$

Above, we assume that $z_1 - z_2 > 100$, $\mu = \mathtt{B1}, \mathtt{B2}\ \mathbf{start}\ \mathtt{S} : a(k)$, $\mu' = !\mathtt{B1} \texttt{ -> } \mathtt{S} : k\langle book \rangle$, $\mu'' = ?\mathtt{B2} \texttt{ -> } \mathtt{S} : k[quit]$. Note that P_{s} will never reduce to $\mathbf{0}$ since it is a replicated service. Also, the process is confluent: it always reduces to P_{s}. □

2.5.2 Process Projection

We establish now how we can project the behaviour of a single process. For this purpose, we have to consider that the same process may appear in different branches of a choreography. For example, consider the following choreography:

$$\mathsf{if}\ e@\mathsf{p}\ \mathsf{then}\ \ \mathsf{p}[\mathtt{A}] \to \mathsf{q}[\mathtt{B}] : k[l_1]\ \ \mathsf{else}\ \ \mathsf{p}[\mathtt{A}] \to \mathsf{q}[\mathtt{B}] : k[l_2]$$

Above, p performs an internal choice and then, depending on the chosen branch, sends to q either label l_1 or label l_2. Observe that q does not know which branch has been chosen by p until it receives a label from the latter. Thus, the projection of q should be able to react to both labels. We address this issue with *merging* [31]. Whenever the code for a process p is scattered in different branches of a choreography, we will try to obtain an endpoint code that implements the behaviours of p in the different branches. Hence, in our example, the projection of q should be $k[\mathtt{B}]?\mathtt{A} \,\&\, \{l_1 : \mathbf{0},\ l_2 : \mathbf{0}\}$. We formally define the merging operator $\sqcup$ as follows:

Definition 2.5.3. $\sqcup$ is a partial commutative operator on processes such that:

1. $(k[\mathtt{A}]?\mathtt{B} \,\&\, \{l_i : P_i\}_{i\in I}) \ \sqcup\ (k[\mathtt{A}]?\mathtt{B} \,\&\, \{l_j : Q_j\}_{j\in J}) \ =$
   $k[\mathtt{A}]?\mathtt{B} \,\&\, (\ \{l_i : P_i\}_{i\in I\setminus J} \cup \{l_i : Q_i\}_{i\in J\setminus I} \cup \{l_i : (P_i \sqcup Q_i)\}_{i\in I\cap J}\ )$
2. otherwise, $P \sqcup Q$ is defined congruently up to $\equiv$.

We use merging to define the projection of the behaviour of a process in a choreography onto a term in our endpoint model.

Definition 2.5.4 (Process Projection)**.** $[\![C]\!]_{\mathsf{p}}$ is a partial operation from choreographies to endpoint processes, inductively defined on the structure of C by the rules in Figure 2.17.

Process projection is faithful to the originating choreography, i.e., it does not add (nor remove) any actions wrt what is specified by the choreography it projects. For a session start, we project the first process and the other active processes to a request and accepts respectively; the service processes are projected to endpoint services. In the case of a communication between two processes, the sender process is projected to an output and the receiver process to an input. The cases for selection and delegation are similar. The projection of a conditional $\mathsf{if}\ e@\mathsf{p}\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2$ is a local conditional for the process p; otherwise, it is the merging of the projections of the two branches C_1 and C_2, reflecting the fact that the other processes should behave accordingly to both branches.

Example 2.5.5 (Two-Buyer Projection)**.** The process projections of choreography given in Example 2.2.2 are the processes reported in Example 2.5.1. □

2.5.3 Linearity

The expressivity of the *(start)* primitive may introduce races on public channels. For example, the following choreography

$$\mathsf{p}[\mathtt{A}], \mathsf{q}[\mathtt{B}]\ \mathbf{start}\ : a(k); \quad \mathsf{r}[\mathtt{A}], \mathsf{s}[\mathtt{B}]\ \mathbf{start}\ : a(k') \tag{2.1}$$

$$
\begin{array}{rcl}
[\![\widetilde{\mathtt{p}[\mathtt{A}]} \ \mathbf{start}\ \widetilde{\mathtt{q}[\mathtt{B}]} : a(k); C]\!]_{\mathsf{r}} & = & \begin{cases} \overline{a}[\tilde{\mathtt{A}}, \tilde{\mathtt{B}}](k); [\![C]\!]_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{p}_1 \\ a[\mathtt{A}_i](k); [\![C]\!]_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{p}_i \text{ and } 2 \le i \le n \\ !a[\mathtt{B}_i](k); [\![C]\!]_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{q}_i \text{ and } 1 \le i \le m \\ [\![C]\!]_{\mathsf{r}} & \text{otherwise} \end{cases} \\
\left(\begin{array}{l} \widetilde{\mathtt{p}[\mathtt{A}]} = \mathsf{p}_1[\mathtt{A}_1], \ldots, \mathsf{p}_n[\mathtt{A}_n] \\ \widetilde{\mathtt{q}[\mathtt{B}]} = \mathsf{q}_1[\mathtt{B}_1], \ldots, \mathsf{q}_m[\mathtt{B}_m] \end{array} \right) & & \\
[\![\mathtt{p}[\mathtt{A}].e \text{ -> } \mathtt{q}[\mathtt{B}].x : k; C]\!]_{\mathsf{r}} & = & \begin{cases} k[\mathtt{A}]!\mathtt{B}\langle e \rangle; [\![C]\!]_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{p} \\ k[\mathtt{B}]?\mathtt{A}(x); [\![C]\!]_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{q} \\ [\![C]\!]_{\mathsf{r}} & \text{otherwise} \end{cases} \\
[\![\mathtt{p}[\mathtt{A}] \text{ -> } \mathtt{q}[\mathtt{B}] : k[l]; C]\!]_{\mathsf{r}} & = & \begin{cases} k[\mathtt{A}]!\mathtt{B} \oplus l; [\![C]\!]_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{p} \\ k[\mathtt{B}]?\mathtt{A} \,\&\, \{l : [\![C]\!]_{\mathsf{r}}\} & \text{if } \mathsf{r} = \mathsf{q} \\ [\![C]\!]_{\mathsf{r}} & \text{otherwise} \end{cases} \\
[\![\mathtt{p}[\mathtt{A}] \text{ -> } \mathtt{q}[\mathtt{B}] : k\langle k'[\mathtt{C}] \rangle; C]\!]_{\mathsf{r}} & = & \begin{cases} k[\mathtt{A}]!\mathtt{B}\langle k'[\mathtt{C}] \rangle; [\![C]\!]_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{p} \\ k[\mathtt{B}]?\mathtt{A}(k'[\mathtt{C}]); [\![C]\!]_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{q} \\ [\![C]\!]_{\mathsf{r}} & \text{otherwise} \end{cases} \\
[\![\mathsf{if}\ e@\mathsf{p}\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2]\!]_{\mathsf{r}} & = & \begin{cases} \mathsf{if}\ e\ \mathsf{then}\ [\![C_1]\!]_{\mathsf{r}} \mathsf{else}\ [\![C_2]\!]_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{p} \\ [\![C_1]\!]_{\mathsf{r}} \sqcup [\![C_2]\!]_{\mathsf{r}} & \text{otherwise} \end{cases} \\
[\![\mathsf{def}\ X(\tilde{D}) = C'\ \mathsf{in}\ C]\!]_{\mathsf{r}} & = & \begin{cases} \mathsf{def}\ X(\tilde{x_i}, \tilde{k_i}) = [\![C']\!]_{\mathsf{r}}\ \mathsf{in}\ [\![C]\!]_{\mathsf{r}} \\ \qquad \text{if } \mathsf{r} = \mathsf{p}_i \text{ and } 1 \le i \le n \\ [\![C]\!]_{\mathsf{r}} \quad \text{otherwise} \end{cases} \\
(\tilde{D} = \mathsf{p}_1(\tilde{x_1}, \tilde{k_1}), \ldots, \mathsf{p}_n(\tilde{x_n}, \tilde{k_n})) & & \\
[\![X\langle \tilde{E} \rangle]\!]_{\mathsf{r}} & = & \begin{cases} X\langle \tilde{e_i}, \tilde{k_i} \rangle & \text{if } \mathsf{r} = \mathsf{p}_i \text{ and } 1 \le i \le n \\ \mathbf{0} & \text{otherwise} \end{cases} \\
(\tilde{E} = \mathsf{p}_1(\tilde{e_1}, \tilde{k_1}), \ldots, \mathsf{p}_n(\tilde{e_n}, \tilde{k_n})) & & \\
[\![\mathbf{0}]\!]_{\mathsf{r}} & = & \mathbf{0} \\
(\text{no rule for } (\boldsymbol{\nu} r)\ C) & &
\end{array}
$$

Figure 2.17: Choreography Calculus, process projection.

features four processes starting two different sessions on the same public channel. If we run their projections in parallel, we have a race between p and r and another between q and s for synchronising on a. This may result in p starting a session with s and q starting a session with r, violating the choreography. Here, we introduce a condition, called linearity, for avoiding this kind of races.

In the sequel an *interaction node*, denoted by n, is an abstraction of a node in a choreography syntax tree. n can either be $\tilde{\mathsf{p}}\ \mathbf{start}\ \tilde{\mathsf{q}} : a$ (abstracting a *(start)* node, where $\tilde{\mathsf{p}}$ are free names) or $\mathsf{p}\ \texttt{->}\ \mathsf{q}$ (abstracting a session interaction, where both p and q are free names). We write $\mathsf{n}_1 \prec \mathsf{n}_2 \in C$ whenever n_1 precedes n_2 in the choreography C (as a consequence, n_1 and n_2 cannot appear in different branches of a *(cond)* term). We use interaction nodes for establishing dependencies between process actions, defined in the following.

Definition 2.5.6 (Dependency)**.** We write $\mathsf{n}_1 \prec_{\mathsf{p}} \mathsf{n}_2 \in C$ if $\mathsf{n}_1 \prec \mathsf{n}_2 \in C$ and either

1. $\mathsf{n}_1 = \tilde{\mathsf{p}}\ \mathbf{start}\ \tilde{\mathsf{q}} : a$ and $\mathsf{n}_2 = \mathsf{p}\ \texttt{->}\ \mathsf{q}$ where $\mathsf{p} \in \tilde{\mathsf{p}}, \tilde{\mathsf{q}}$; or,
2. $\mathsf{n}_1 = \tilde{\mathsf{p}}\ \mathbf{start}\ \tilde{\mathsf{q}} : a$ and $\mathsf{n}_2 = \tilde{\mathsf{r}}\ \mathbf{start}\ \tilde{\mathsf{s}} : b$ where $\mathsf{p} \in \tilde{\mathsf{p}}, \tilde{\mathsf{q}}$ and $\mathsf{p} \in \tilde{\mathsf{r}}$; or,
3. $\mathsf{n}_1 = \mathsf{q}\ \texttt{->}\ \mathsf{p}$ and $\mathsf{p} \in \mathsf{fn}(\mathsf{n}_2)$.

Intuitively, $\mathsf{n}_1 \prec_{\mathsf{p}} \mathsf{n}_2 \in C$ implies that the projection of process p for the originating node abstracted by n_2 will not be enabled before that for n_1. We can now define the *linearity* property:

Definition 2.5.7 (Linearity)**.** Let C be a choreography. We say that C is *linear* if for all nodes $\mathsf{n}_1 = \tilde{\mathsf{p}}\ \mathbf{start}\ \tilde{\mathsf{q}} : a$ and $\mathsf{n}_2 = \tilde{\mathsf{r}}\ \mathbf{start}\ \tilde{\mathsf{s}} : a$ such that $\mathsf{n}_1 \prec \mathsf{n}_2 \in C$ we have that $\forall \mathsf{r} \in \tilde{\mathsf{r}}.\ \exists \mathsf{p} \in \tilde{\mathsf{p}}, \tilde{\mathsf{q}}.\ \mathsf{n}_1 \prec_{\mathsf{p}} \ldots \prec_{\mathsf{r}} \mathsf{n}_2$.

Linearity checks that, for all start nodes $\mathsf{n}_1 \prec \mathsf{n}_2 \in C$ on the same a, each active process in n_2 depends on some process in n_1, avoiding races between active processes. Linearity is preserved by our semantics and is decidable, since a choreography is linear whenever its one-time unfolding of recursions is linear (cf. [55], which formulates a similar notion of linearity).

Example 2.5.8. In (2.1) we cannot build any dependencies unless, e.g., $\mathsf{p} = \mathsf{r}$ and $\mathsf{q} = \mathsf{s}$. Instead, the following choreography is linear with dependencies between p and r and between q and s.

$$\begin{aligned}&\mathsf{p}[\mathtt{A}], \mathsf{q}[\mathtt{B}]\ \mathbf{start}\ : a(k);\ \mathsf{p}[\mathtt{C}]\ \texttt{->}\ \mathsf{r}[\mathtt{D}] : k';\ \mathsf{q}[\mathtt{E}]\ \texttt{->}\ \mathsf{t}[\mathtt{F}] : k';\\ &\mathsf{t}[\mathtt{F}]\ \texttt{->}\ \mathsf{s}[\mathtt{G}] : k';\ \mathsf{r}[\mathtt{A}], \mathsf{s}[\mathtt{B}]\ \mathbf{start}\ : a(k)\end{aligned}$$

□

2.5.4 Endpoint Projection

Since different service processes may be started on the same public channel and play the same role, we use $\sqcup$ for merging their behaviours into a single replicated process. We identify such processes with $\lfloor C \rfloor^{a}_{\mathtt{A}}$, the *service grouping* operator, defined below.

Definition 2.5.9 (Service Grouping)**.** The operator $\lfloor C \rfloor^a_{\mathtt{A}}$ is inductively defined on C as follows:

$$\lfloor \widetilde{\mathsf{p}[\mathtt{A}]}\ \mathbf{start}\ \widetilde{\mathsf{q}[\mathtt{B}]} : a(k); C \rfloor^a_{\mathtt{A}} = \begin{cases} \{\mathsf{r}\} \cup \lfloor C \rfloor^a_{\mathtt{A}} & \text{if } \mathsf{r} \in \widetilde{\mathsf{q}[\mathtt{B}]} \\ \lfloor C \rfloor^a_{\mathtt{A}} & \text{otherwise} \end{cases}$$

$$\lfloor \mathsf{if}\ e@\mathsf{p}\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2 \rfloor^a_{\mathtt{A}} \;=\; \lfloor C_1 \rfloor^a_{\mathtt{A}} \cup \lfloor C_2 \rfloor^a_{\mathtt{A}} \qquad \lfloor X\langle \tilde{E} \rangle \rfloor^a_{\mathtt{p}} = \lfloor \mathbf{0} \rfloor^a_{\mathtt{p}} \;=\; \emptyset$$

$$\lfloor \mathsf{def}\ X(\tilde{D}) = C'\ \mathsf{in}\ C \rfloor^a_{\mathtt{A}} = \lfloor C' \rfloor^a_{\mathtt{A}} \cup \lfloor C \rfloor^a_{\mathtt{A}}$$

$$\lfloor \eta; C \rfloor^a_{\mathtt{A}} = \lfloor C \rfloor^a_{\mathtt{A}} \quad \text{if } \eta \text{ is not a } \textit{(start)}$$

A service grouping $\lfloor C \rfloor^a_{\mathtt{A}}$ visits the structure of a choreography for finding all the service processes started on the shared channel a that play role $\mathtt{A}$.

We can finally give the complete definition of EPP. In the following, we say that a choreography is restriction-free if it does not contain any subterm of the form $(\boldsymbol{\nu} r)\, C$.

Definition 2.5.10 (Endpoint Projection)**.** Let $C \equiv (\boldsymbol{\nu} \tilde{\mathsf{p}}, \tilde{k})\, C_f$, where C_f is restriction-free. The EPP of C, written $[\![C]\!]$, is:

$$[\![C]\!] \;=\; (\boldsymbol{\nu} \tilde{k}) \Bigg(\underbrace{\prod_{\mathsf{p} \in \mathsf{fn}(C_f)} [\![C_f]\!]_{\mathsf{p}}}_{\text{(i)}} \;\mid\; \underbrace{\prod_{k \in \mathsf{fn}(C_f)} k : \emptyset}_{\text{(ii)}} \Bigg) \mid \underbrace{\prod_{a,\mathtt{A}} \Big(\bigsqcup_{\mathsf{p} \in \lfloor C_f \rfloor^a_{\mathtt{A}}} [\![C_f]\!]_{\mathsf{p}} \Big)}_{\text{(iii)}}$$

The EPP of a choreography C is the parallel composition of (i) the projections of all active processes; (ii) the queues for all active sessions; and (iii) the replicated processes obtained by merging the projections of all service processes with same public channel and role.

Example 2.5.11. Let C be the two-buyer-helper choreography from Example 2.2.3. Since h_1 and h_2 are grouped under $\lfloor C \rfloor^b_{\mathtt{H}}$, the EPP of C will merge their behaviour into a single process, say, P_{h}; i.e., $[\![C]\!] = [\![C]\!]_{\mathsf{b}_1} \mid [\![C]\!]_{\mathsf{b}_2} \mid [\![C]\!]_{\mathsf{s}} \mid P_{\mathsf{h}}$ where $P_{\mathsf{h}} = [\![C]\!]_{\mathsf{h}_1} \sqcup [\![C]\!]_{\mathsf{h}_2}$ is given below:

$$!b[\mathtt{H}](k');\ k'[\mathtt{H}]?\mathtt{B}(z);\ k'[\mathtt{H}]?\mathtt{B}\ \&\ \left\{ \begin{array}{l} \mathit{done} : \mathbf{0}, \\ \quad \mathit{del} : k'[\mathtt{H}]?\mathtt{B}(z');\ k'[\mathtt{H}]?\mathtt{B}(k[\mathtt{B2}]); \\ \qquad \mathsf{if}\ (z - z' \leq 100) \\ \qquad\quad \mathsf{then}\ k[\mathtt{B2}]!\mathtt{S} \oplus \mathit{ok}; \\ \qquad\qquad\quad k[\mathtt{B2}]!\mathtt{S}\langle \mathit{addr} \rangle; \\ \qquad\qquad\quad k[\mathtt{B2}]?\mathtt{S}(z'') \\ \qquad\quad \mathsf{else}\ \ k[\mathtt{B2}]!\mathtt{S} \oplus \mathit{quit} \end{array} \right\}$$

□

Now that we have defined EPP, we can formalise the intuition that the swapping relation $\simeq_{\mathcal{C}}$ models the parallel execution of processes at the level of choreographies. Since at the endpoint level parallel execution is captured structurally by the parallel operator $\mid$, a swapping in a choreography does not influence the endpoint system generated by EPP.

Lemma 2.5.1 (EPP invariance under swapping)**.** *Let $C \simeq_{\mathcal{C}} C'$. Then, $[\![C]\!] = [\![C']\!]$.*

Proof (sketch). The main part of the proof is to show that process projection is invariant under the rules that define the swapping relation for choreographies (Figure 2.2). Rule $\lfloor^{\text{Sw}}|_{\text{Inter}}\rceil$ is a trivial case. For rule $\lfloor^{\text{Sw}}|_{\text{Cond-Inter}}\rceil$, we have to check that the projections of the processes in the swapped interaction η do not change. We simply need to observe that, in the definition of process projection for conditionals, for each p in η we have that $[\![\eta]\!]_{\mathsf{p}} = [\![\eta]\!]_{\mathsf{p}} \sqcup [\![\eta]\!]_{\mathsf{p}}$. The last case is rule $\lfloor^{\text{Sw}}|_{\text{Cond-Cond}}\rceil$. Here, by the definition of merging, we can observe that the projection of the choice and its swapped version is the same for all processes. □

2.5.5 EPP Theorem

Before showing the main theorem for our EPP we have to introduce some auxiliary concepts for establishing the relationship between a choreography, its projection, and their respective reduction labels.

Strict Reductions. In the following, we consider only *strict reductions* of terms, denoted by $\rightsquigarrow$. Strict reductions are reductions where $(\boldsymbol{\nu})$ -restricted names that are active, i.e. not under a prefix, are never renamed. Observe that we do not lose generality, since for every reduction there is always a corresponding strict reduction (cf. [31]). Referring only to strict reductions allows us to observe the actions performed by restricted names. Formally this is obtained by changing the rules for handling restriction in the semantics of the global and endpoint calculi. The updated rules are the ones concerning restriction:

Definition 2.5.12 (New Restriction Rules)**.**

$$\frac{C \overset{\lambda}{\rightsquigarrow} C'}{(\boldsymbol{\nu} r)\, C \overset{\lambda}{\rightsquigarrow} (\boldsymbol{\nu} r)\, C'} \lfloor^{\text{C}}|_{\text{Res}}\rceil \qquad \frac{P \overset{\mu}{\rightsquigarrow} P'}{(\boldsymbol{\nu} k)\, P \overset{\mu}{\rightsquigarrow} (\boldsymbol{\nu} k)\, P'} \lfloor^{\text{P}}|_{\text{Res}}\rceil$$

We denote a finite strict reduction chain with $C \overset{\tilde{\lambda}}{\rightsquigarrow}{}^{*} C'$, i.e. $\tilde{\lambda} = \lambda_1, \ldots, \lambda_n$ for $C \overset{\lambda_1}{\rightsquigarrow} \ldots \overset{\lambda_n}{\rightsquigarrow} C'$. We adopt the same notation for strict reduction chains in the endpoint calculus.

Pruning. In our endpoint model, replicated services remain always available. However, in a choreography services can be removed by reductions, e.g., by reducing a *(start)* term whose public channel does not appear in the reductum. To deal with this asymmetry we introduce *pruning*, a relation that allows us to ignore unused endpoint services and branches (this notion originally comes from [31]).

Definition 2.5.13 (Pruning)**.** Let $Q \equiv Q_0 \mid R$, where $R = \prod_{i \in I} !a_i[\mathtt{A}_i](k_i); R_i$. If furthermore we have that:

(i) $a_i \notin \mathsf{fn}(Q_0)$ for every $i \in I$;

(ii) $P \sqcup Q_0 = Q_0$ for some P;

(iii) $Q_0 \xrightarrow{\mu} Q_0'$ implies that there exists P' such that $P \xrightarrow{\mu} P'$ and $P' \prec Q_0'$.

$$\frac{\tilde{\lambda} \vdash \tilde{\mu}}{\widetilde{\mathtt{p}[\mathtt{A}]}\ \mathbf{start}\ \widetilde{\mathtt{q}[\mathtt{B}]} : a(k), \tilde{\lambda} \vdash \tilde{\mathtt{A}}\ \mathbf{start}\ \tilde{\mathtt{B}} : a(k), \tilde{\mu}} \lfloor^{\mathrm{TJ}}|_{\textsc{Start}}\rceil$$

$$\frac{?\mathtt{A} \text{ -> } \mathtt{B} : k \notin \tilde{\mu_1} \quad \tilde{\lambda} \vdash \tilde{\mu_1}, \tilde{\mu_2}}{\mathtt{p}[\mathtt{A}].v \text{ -> } \mathtt{q}[\mathtt{B}].x : k, \tilde{\lambda} \vdash\ !\mathtt{A} \text{ -> } \mathtt{B} : k\langle v\rangle, \tilde{\mu_1}, ?\mathtt{A} \text{ -> } \mathtt{B} : k\langle v\rangle, \tilde{\mu_2}} \lfloor^{\mathrm{TJ}}|_{\textsc{Com}}\rceil$$

$$\frac{?\mathtt{A} \text{ -> } \mathtt{B} : k \notin \tilde{\mu_1} \quad \tilde{\lambda} \vdash \tilde{\mu_1}, \tilde{\mu_2}}{\mathtt{p}[\mathtt{A}] \text{ -> } \mathtt{q}[\mathtt{B}] : k[l], \tilde{\lambda} \vdash\ !\mathtt{A} \text{ -> } \mathtt{B} : k[l], \tilde{\mu_1}, ?\mathtt{A} \text{ -> } \mathtt{B} : k[l], \tilde{\mu_2}} \lfloor^{\mathrm{TJ}}|_{\textsc{Sel}}\rceil$$

$$\frac{?\mathtt{A} \text{ -> } \mathtt{B} : k \notin \tilde{\mu_1} \qquad \tilde{\lambda} \vdash \tilde{\mu_1}, \tilde{\mu_2}}{\mathtt{p}[\mathtt{A}] \text{ -> } \mathtt{q}[\mathtt{B}] : k\langle k'[\mathtt{C}]\rangle, \tilde{\lambda} \vdash !\mathtt{A} \text{ -> } \mathtt{B} : k\langle k'[\mathtt{C}]\rangle, \tilde{\mu_1}, ?\mathtt{A} \text{ -> } \mathtt{B} : k\langle k'[\mathtt{C}]\rangle, \tilde{\mu_2}} \lfloor^{\mathrm{TJ}}|_{\textsc{Del}}\rceil$$

$$\frac{\tilde{\lambda} \vdash \tilde{\mu}}{\tau@\mathtt{p}, \tilde{\lambda} \vdash \tau, \tilde{\mu}} \lfloor^{\mathrm{TJ}}|_{\textsc{Cond}}\rceil \qquad \frac{}{\epsilon \vdash \epsilon} \lfloor^{\mathrm{TJ}}|_{\textsc{Empty}}\rceil$$

Figure 2.18: Choreography Calculus, trace judgements.

then we write $P \prec Q$ and say that P *prunes* Q.

Hereafter, we write $Q \succ P$ iff $P \prec Q$. Intuitively, $P \prec Q$ means that P is obtained from Q by:

- removing some unused services (i);
- removing some input branches (ii) that were not going to be used (iii).

Trace Judgements. We establish a formal relationship between the labels of the choreography calculus and those of the endpoint calculus, so that we can judge whether some actions performed in the endpoint calculus implement the communications described in a choreography. Intuitively, the entailment $\tilde{\lambda} \vdash \tilde{\mu}$ judges that the endpoint actions $\tilde{\mu}$ implement the global actions $\tilde{\lambda}$.

Definition 2.5.14 (EPP trace judgements)**.** The relation $\tilde{\lambda} \vdash \tilde{\mu}$ is the smallest relation satisfying the rules reported in Figure 2.18.

We briefly comment the rules for trace judgements. Rule $\lfloor^{\mathrm{TJ}}|_{\textsc{Start}}\rceil$ checks that a start performed in a choreography corresponds to an endpoint start with same roles, shared channel and session. Rule $\lfloor^{\mathrm{TJ}}|_{\textsc{Com}}\rceil$ checks that a communication in a choreography trace is performed at the enpoint level by checking that both the corresponding input and output actions are made. The rule allows for some interleaving of unrelated labels, which can happen due to the asynchrony of endpoint systems. Rules $\lfloor^{\mathrm{TJ}}|_{\textsc{Sel}}\rceil$ and $\lfloor^{\mathrm{TJ}}|_{\textsc{Del}}\rceil$ behave in the same way, respectively for selections and delegations. Rules $\lfloor^{\mathrm{TJ}}|_{\textsc{Cond}}\rceil$ and $\lfloor^{\mathrm{TJ}}|_{\textsc{Empty}}\rceil$ handle internal choices and empty traces, denoted by ϵ.

EPP Theorem. We can finally present the main result of this Chapter, the *EPP Theorem*, which formalises the relationship between the semantics of a well-typed, linear choreography and the semantics of its EPP.

Theorem 2.5.15 (EPP Theorem)**.** *Let $C \equiv (\boldsymbol{\nu} \tilde{\mathtt{p}}, \tilde{k})\ C_f$ be linear and well-typed, with C_f restriction-free; then,*

1. *(Completeness) $C \overset{\lambda}{\rightsquigarrow} C'$ implies that there exist P and C'' such that (i) $C' \overset{\tilde{\lambda}'}{\rightsquigarrow}{}^* C''$; (ii) $[\![C]\!] \overset{\tilde{\mu}}{\rightsquigarrow}{}^* P$ and $\lambda, \tilde{\lambda}' \vdash \tilde{\mu}$; and (iii) $[\![C'']\!] \prec P$.*
2. *(Soundness) $[\![C]\!] \overset{\tilde{\mu}}{\rightsquigarrow}{}^* P$ implies that there exist P' and C' such that (i) $P \overset{\tilde{\mu}'}{\rightsquigarrow}{}^* P'$; (ii) $C \overset{\tilde{\lambda}}{\rightsquigarrow}{}^* C'$ and $\tilde{\lambda} \vdash \tilde{\mu}, \tilde{\mu}'$; and (iii) $[\![C']\!] \prec P'$.*

Proof. See Appendix A.2. □

Above, point 1 states that an EPP can mimic (up to pruning) all the reductions of its originating choreography; on the other hand, point 2 says that an EPP always eventually reduces (up to pruning) to the projection of a (possibly reached after multiple reductions) reductum of its originating choreography. Both points ensure that the observables of a choreography are implemented by those of its EPP.

By Theorems 2.3.2 and 2.5.15, we can formalise our *deadlock-freedom-by-design* property. Below, $\xrightarrow{\tilde{\mu}}{}^*$ is the closure of $\xrightarrow{\mu}$.

Corollary 2.5.1.1 (Deadlock-freedom-by-design)**.** *Let C be linear and well-typed. Then, for any P such that $[\![C]\!] \xrightarrow{\tilde{\mu}}{}^* P$, we have that either $P \xrightarrow{\mu'} P'$ for some P', μ' or $\mathbf{0} \prec P$.*

Proof. By Theorem 2.3.2, C can always reduce until it becomes equivalent to $\mathbf{0}$ up to $\equiv$ (observe that the well-sortedness of C required by Theorem 2.3.2 is guaranteed by the type system); by Theorem 2.5.15, P must be able to follow the reductions of C and therefore P can only terminate when C does. □

2.5.6 Typing expressiveness

Previously proposed typing disciplines for session types ensure properties similar to ours, but performing type analysis directly on endpoint programs [31, 21, 55]. Our typing discipline subsumes a larger class of safe deadlock-free systems, by exploiting the extra information that we gain from defining implementations with choreographies. In particular, our typing system allows for two novel features wrt standard multiparty session typing: inter-protocol coherence and partial protocol implementation.

Inter-protocol coherence. Consider the protocol:

$$G \;=\; \mathtt{A} \to \mathtt{B} : \{ l_1 : \mathtt{C} \to \mathtt{A} : \langle \mathbf{int} \rangle,\ l_2 : \mathtt{C} \to \mathtt{B} : \langle \mathbf{int} \rangle \}$$

Above, A communicates to B a choice between labels l_1 and l_2. In the first case another role, C, is expected to communicate an integer to A; otherwise, C will communicate an integer to B. A potential use case for G could be that C possesses some good, and A decides where the

good should be sent to (A itself or B). Previous work on global types cannot type any system implementing G, since G cannot be projected onto a correct set of endpoint types [105]. Indeed, in the protocol, C is not informed of the choice made by A and thus cannot know whether it should communicate with A or B afterwards. We refer to this problem by saying that G is not *coherent* for previous type systems based on global types. In our framework, we do not consider protocol coherence because protocols such as G above can easily be implemented by interleaving them with other ones. For example, consider the following choreography:

$$
\begin{array}{lll}
1. & \mathsf{p}[\mathtt{A}] \ \mathbf{start} \ \mathsf{q}[\mathtt{B}], \mathsf{r}[\mathtt{C}] : a(k); \quad \mathsf{q}[\mathtt{D}], \mathsf{r}[\mathtt{E}] \ \mathbf{start} \ : b(k'); \\
2. & \mathsf{if} \ e@\mathsf{p} \ \mathsf{then} & \mathsf{p}[\mathtt{A}] \texttt{ -> } \mathsf{q}[\mathtt{B}] : k[l_1]; \ \mathsf{q}[\mathtt{D}] \texttt{ -> } \mathsf{r}[\mathtt{E}] : k'[l_1]; \\
3. & & \mathsf{r}[\mathtt{C}].\mathit{some\_int} \texttt{ -> } \mathsf{p}[\mathtt{A}].x : k \\
4. & \mathsf{else} & \mathsf{p}[\mathtt{A}] \texttt{ -> } \mathsf{q}[\mathtt{B}] : k[l_2]; \ \mathsf{q}[\mathtt{D}] \texttt{ -> } \mathsf{r}[\mathtt{E}] : k'[l_2]; \\
5. & & \mathsf{r}[\mathtt{C}].\mathit{some\_int} \texttt{ -> } \mathsf{q}[\mathtt{B}].y : k
\end{array}
$$

The choreography above can be typed correctly using G as type for a (we omit the typing for b). In order to notify r, playing role C in session k, of the choice performed by p, playing role A in session k, we make use of an additional session between q and r. We use this session, k', after q receives the choice from p. Observe that the choreography is typable and can be correctly projected by our EPP. The key aspect of this example is that our framework leaves the task of defining a coherent system to the implementation (the choreography). Hence, protocols can be designed at a higher level of abstraction. For example, in G we do not specify how C is notified of the choice. We call this aspect *inter-protocol coherence*, since it is the composition of protocols in a choreography that is checked for coherence (by checking whether its EPP is defined), and not each protocol in itself.

Partial protocol implementation. We now discuss partial protocol implementation with the following choreography:

$$
\begin{array}{lll}
1. & \mathsf{p}[\mathtt{A}], \mathsf{q}[\mathtt{B}] \ \mathbf{start} \ : a(k); & \mathsf{p}[\mathtt{A}] \texttt{ -> } \mathsf{q}[\mathtt{B}] : k[l_1]; \\
2. & \mathsf{p}[\mathtt{A}], \mathsf{q}[\mathtt{B}] \ \mathbf{start} \ : a(k'); & \mathsf{if} \ e@\mathsf{p} \ \mathsf{then} \ \mathsf{p}[\mathtt{A}] \texttt{ -> } \mathsf{q}[\mathtt{B}] : k'[l_2] \\
3. & & \mathsf{else} \ \mathsf{p}[\mathtt{A}] \texttt{ -> } \mathsf{q}[\mathtt{B}] : k'[l_3]
\end{array}
$$

The choreography above is typable in our system; a type for a is the following:

$$\mathtt{A} \texttt{ -> } \mathtt{B} : \{l_1 : \mathsf{end}, \ l_2 : \mathsf{end}, \ l_3 : \mathsf{end}\}$$

However, the endpoint projection for q would not be typable with standard contra-variant input typing, which requires that *at least* all the branches in the type are implemented [54, 55]. Again, this is a consequence of using choreographies: since in the choreography we know exactly which outputs will correspond to which inputs, we can ensure that the protocol branches that q does not implement will never be used.

2.6 Examples

Choreography-based programming can have several applications, ranging from multicore programming to distributed Web Services. Below, we present and discuss two possible example applications.

2.6.1 Streaming-AVP

In this example, we show how to combine two different protocols for implementing a streaming service for movie files. We start by giving the protocol for streaming:

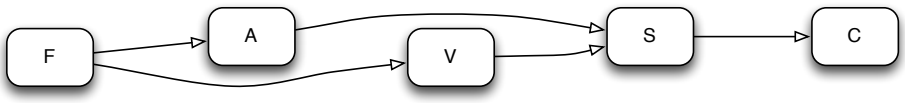
$$\mathsf{rec}\,\mathbf{t};\ \mathtt{S} \to \mathtt{C} : \langle \mathbf{bytes} \rangle;\ \mathtt{S} \to \mathtt{C} : \{\ again : \mathbf{t},\ \ end : \mathsf{end}\ \}$$

In the protocol above, `S` is a streaming server sending byte packets to a client `C`. After each packet, `S` communicates to `C` whether there are more bytes to be sent or the stream is over (choices $again$ and end). The other protocol (AVP, for Audio-Video Processing) is defined by the following global type:

$$\begin{array}{l}\mathsf{rec}\,\mathbf{t};\ \mathtt{F} \to \mathtt{A} : \langle \mathbf{bytes} \rangle;\ \mathtt{F} \to \mathtt{V} : \langle \mathbf{bytes} \rangle;\ \mathtt{A} \to \mathtt{S} : \langle \mathbf{bytes} \rangle;\ \mathtt{V} \to \mathtt{S} : \langle \mathbf{bytes} \rangle;\\ \mathtt{F} \to \mathtt{A} : \{\ again : \mathtt{A} \to \mathtt{V} : \langle \mathbf{bytes} \rangle;\ \mathtt{V} \to \mathtt{S} : \langle \mathbf{bytes} \rangle; \mathbf{t},\ \ end : \mathsf{end}\ \}\end{array}$$

Four roles participate in this protocol: a filesystem `F`, an audio decoder `A`, a video decoder `V`, and a sink `S`. The flow of information in this protocol consists of `F` sending the raw audio information to `A` and the raw video information to `V`, read from the movie file. Then both `A` and `V` send the processed decoded information to the sink `S`.

The goal of this example is to show how to interleave the two protocols so that the information produced by an implementation of AVP is forwarded to a client (e.g., a display) as pictured below:



We implement such a system as follows:

$$\begin{array}{l}\mathsf{c}[\mathtt{C}]\ \mathbf{start}\ \mathsf{s}[\mathtt{S}] : a(stream);\quad \mathsf{s}[\mathtt{S}]\ \mathbf{start}\ \mathsf{f}[\mathtt{F}], \mathsf{a}[\mathtt{A}], \mathsf{v}[\mathtt{V}] : b(avp);\\ \mathsf{def}\ \mathsf{AVPStreaming}(\mathsf{c}(stream), \mathsf{s}(stream, avp), \mathsf{f}(avp), \mathsf{a}(avp), \mathsf{v}(avp)) = C\ \mathsf{in}\\ \quad \mathsf{AVPStreaming}\langle \mathsf{c}(stream), \mathsf{s}(stream, avp), \mathsf{f}(avp), \mathsf{a}(avp), \mathsf{v}(avp)\rangle\end{array}$$

The choreography above starts two sessions (on a and b) corresponding to the two protocols specified above. The streamer in the first protocol, and the file system, the audio decoder and the video decoder in the second were chosen to be service processes, but other implementations may follow different directions. The core of the choreography is the term

C defined below.

1. $\mathsf{f}[\mathtt{F}].\mathsf{readAudioBytes}() \to \mathsf{a}[\mathtt{A}].audioByteChunk : avp;$
2. $\mathsf{f}[\mathtt{F}].\mathsf{readVideoBytes}() \to \mathsf{v}[\mathtt{V}].videoByteChunk : avp;$
3. $\mathsf{a}[\mathtt{A}].\mathsf{decodeA}(audioByteChunk) \to \mathsf{s}[\mathtt{S}].audioPacket : avp;$
4. $\mathsf{v}[\mathtt{V}].\mathsf{decodeV}(videoByteChunk) \to \mathsf{s}[\mathtt{S}].videoPacket : avp;$
5. $\mathsf{s}[\mathtt{S}].\mathsf{combine}(audioPacket, videoPacket) \to \mathsf{c}[\mathtt{C}].packet : stream;$
6. $\mathsf{if}\ (\mathsf{more}())@f$
7. $\quad \mathsf{f}[\mathtt{F}] \to \mathsf{a}[\mathtt{A}] : avp[again];\ \mathsf{a}[\mathtt{A}] \to \mathsf{v}[\mathtt{V}] : avp[again];$
8. $\quad \mathsf{v}[\mathtt{V}] \to \mathsf{s}[\mathtt{S}] : avp[again];\ \mathsf{s}[\mathtt{S}] \to \mathsf{c}[\mathtt{C}] : stream[again];$
9. $\quad \mathsf{AVPStreaming}\langle \mathsf{c}(stream), \mathsf{s}(stream, avp), \mathsf{f}(avp), \mathsf{a}(avp), \mathsf{v}(avp)\rangle$
10. else
11. $\quad \mathsf{f}[\mathtt{F}] \to \mathsf{a}[\mathtt{A}] : avp[end];\ \mathsf{a}[\mathtt{A}] \to \mathsf{v}[\mathtt{V}] : avp[end];$
12. $\quad \mathsf{v}[\mathtt{V}] \to \mathsf{s}[\mathtt{S}] : avp[end];\ \mathsf{s}[\mathtt{S}] \to \mathsf{c}[\mathtt{C}] : stream[end]$

Choreography C can be repeated several times. Lines 1–5 describe how the filesystem process f sends audio and video information to the processes implementing the audio and video decoders, respectively a and v. Then, the audio and video decoders send the processed information to the process s, which implements the sink. The same sink process s implements the streamer role in the other protocol and therefore sends to the client c the data obtained by combining the audio and video packets (as in multiplexing). Lines 7–9 correspond to the branch where process f will communicate to the other processes that there is more data to process. Similarly, termination is communicated to the other processes in the else-branch (Lines 11–12). The choreography above is well-typed wrt the two given global types.

The EPP of the choreography above is straightforward since it needs no merging of services. As an example, the following process is the EPP for process s:

$$
\begin{array}{l}
!a[\mathtt{S}](stream);\ \overline{b}[\mathtt{S}, \mathtt{F}, \mathtt{A}, \mathtt{V}](avp); \\
\mathsf{def}\ \mathsf{AVPStreaming}(stream, avp) = \\
\qquad \left(\begin{array}{l}
avp[\mathtt{S}]?\mathtt{A}(audioPacket);\ avp[\mathtt{S}]?\mathtt{V}(videoPacket); \\
stream[\mathtt{S}]!\mathtt{C}\langle \mathsf{combine}(audioPacket, videoPacket)\rangle; \\
avp[\mathtt{S}]?\mathtt{V}\ \& \left\{ \begin{array}{l} again : stream[\mathtt{S}]!\mathtt{C} \oplus again;\ \mathsf{AVPStreaming}\langle stream, avp\rangle \\ end : stream[\mathtt{S}]!\mathtt{C} \oplus end \end{array} \right\}
\end{array} \right) \\
\mathsf{in}\ \mathsf{AVPStreaming}\langle stream, avp\rangle
\end{array}
$$

2.6.2 OpenID and Logging

We give another example using a variant of OpenID [81], a protocol where a client (called user) authenticates to a server (called relying party) through a third-party identity provider. We define the protocol with the following global type:

$$
\begin{array}{l}
\mathtt{U} \to \mathtt{RP} : \langle \mathbf{string}\rangle;\ \mathtt{RP} \to \mathtt{IP} : \langle \mathbf{string}\rangle;\ \mathtt{U} \to \mathtt{IP} : \langle \mathbf{string}\rangle; \\
\mathtt{IP} \to \mathtt{RP} : \left\{ \begin{array}{rl} ok : & \mathtt{RP} \to \mathtt{U} : \{ok : \mathtt{RP} \to \mathtt{U} : \langle G@\mathtt{C}\rangle; \mathsf{end}\}, \\ fail : & \mathtt{RP} \to \mathtt{U} : \{fail : \mathsf{end}\} \end{array} \right\}
\end{array}
$$

Above, RP abstracts the relying party, IP the identity provider and U the user. First, U sends her username to RP, which forwards it to IP. Then, U sends her password to IP, which will notify RP of whether the username/password credentials are valid (ok or $fail$). Finally, RP forwards the notification to U. If successful, RP also delegates to U role C in a session of type G, where $G = \mathtt{S} \to \mathtt{C} : \langle \mathbf{string} \rangle$.

We interleave OpenID with another protocol where a client C asks a log server S for either a secret or a public log. Finally, S replies with the corresponding log content. Formally,

$$\mathtt{C} \to \mathtt{S} : \{ secret : \mathtt{S} \to \mathtt{C} : \langle \mathbf{string} \rangle,\ public : \mathtt{S} \to \mathtt{C} : \langle \mathbf{string} \rangle \}$$

Now, we can program our system as follows:

```
1.  rp[RP], u[U] start ip[IP] : publicOpenID(k);
2.  u[U].user -> rp.user : k;
3.  rp[RP].user -> ip[IP].username : k;
4.  u[U].pwd -> ip[IP].password : k;
5.  if (check(username, password))@ip
6.    ip[IP] -> rp[RP] : k[ok];
7.    rp[RP] -> u[U] : k[ok];
8.    if (high(username))@rp
9.      rp[C] start s[S] : log(k');
10.     rp[C] -> s[S] : k'[secret];
11.     rp[RP] -> u[U] : k⟨k'[C]⟩;
12.     s[S].secret_msg -> u[C].logContent : k'
13.   else
14.     rp[C] start s[S] : log(k');
15.     rp[C] -> s[S] : k'[public];
16.     rp[RP] -> u[U] : k⟨k'[C]⟩;
17.     s[S].public_msg -> u[C].logContent : k'
18. else
19.   ip[IP] -> rp[RP] : k[fail];
20.   ip[IP] -> u[U] : k[fail]
```

Above, rp, u, ip, and s are the endpoints of the system. Line 1 describes the initiation of a protocol instance between rp, u and ip, by means of the public name a. In Lines 2–4, u sends its credentials to rp and ip (only the username to rp). Then, ip checks the data received (Line 5) and communicates the outcome to rp (Lines 6 and 19). In both cases, the selection is forwarded to u (Lines 7 and 20). In the if-branch, rp checks the user access level. If high, it starts a new session (log) spawning s (Line 9) and asks for a secret log (Line 10). Consequently, rp will delegate its role in session k' to the user u through session k. Finally, u will get the requested log. The system works similarly in the public log case (Lines 14–17).

We conclude by showing the EPP for the service process s:

$$!log[\mathtt{S}](k');\ k'[\mathtt{S}]?\mathtt{C}\ \&\ \left\{ \begin{array}{l} secret : k'[\mathtt{S}]!\mathtt{C}\langle private\_msg \rangle, \\ public : k'[\mathtt{S}]!\mathtt{C}\langle public\_msg \rangle \end{array} \right\}$$

Note that the process projections of s for the different branches of the choreography (Lines 9–12 and 14–17) would yield different code which is then merged into the one above.

2.7 Related Work

Global methods for communicating systems occur in different forms, including MSC [59], security protocols [25, 29, 22] and automata theory [44]. However, these works are not intended as fully-fledged programming languages since they do not deal with, e.g., different layers of abstraction or value passing.

This is the first work proposing an asynchronous semantics for a choreography language based on sessions. To the best of our knowledge, the notion of *delayed input* [68] is the most similar result to the asynchrony modelled by our semantics.

The closest work to ours is [31], which proposes a *synchronous* choreography model without delegation based on *binary* session types, i.e., session types for protocols with two participants that describe communications from the point of view of one of the endpoints (binary session types are not global descriptions). Our framework shows that switching to multiparty asynchronous sessions with delegation introduces more complexity, but also that such complexity can be elegantly hidden from the programmer. Moreover, [31] has implicit process identifiers and deals with a stronger sequential operator, requiring two syntactic restrictions on choreographies, i.e., well-threadedness and connectedness. In contrast, our approach needs no such restrictions because of explicit process identifiers and a more relaxed sequential operator, given by our formalisations of asynchrony and parallelism in choreographies. Finally, [31] ensures EPP correctness based on a type preservation result, while we guarantee the same without the need for an endpoint typing.

Multiparty session types have been previously used for checking endpoint systems [55, 21, 36]. We have shown that they can be adopted for typing choreographies, defining a new class of correct well-typed endpoint systems (through EPP). Our global types as well as our endpoint model are taken from [21]. Other works have given an asynchronous semantics to global types: [55] defines a semantics in terms of that of the projection of global types while [37, 38] interprets global types as asynchronous communication automata. Our linearity notion is inspired by [55].

In [21], the authors guarantee *progress* for multiparty sessions by building additional restrictions on top of (endpoint) session typing. Processes satisfying progress do not get stuck provided that they can be run in parallel with other processes that would unlock stuck states. In our work progress, implied by deadlock-freedom, is an immediate consequence of our EPP Theorem, yielding a simpler analysis.

2.8 Discussion and Extensions

We discuss some aspects of choreography-driven programming and future extensions in relation to the work presented in this Chapter.

2.8.1 Approach

It may be unclear how the choreography-driven approach may deal with standard aspects of programming such as *team development* and *endpoint code reuse*.

Team development. Our framework supports team development, i.e., the development of a choreography by a team of many programmers, with (i) service merging and (ii) pro-

cedures.

(i) Our EPP merges service processes started on the same public channel and role into a single process. This allows two choreographies to be composed into a bigger system, whenever their respective service processes are mergeable. Mergeability can be assured by using a design pattern, i.e., enforcing service processes that need to be merged to start with distinct branches.

(ii) Procedures can be written and typed separately, so as to create libraries that can be used by other choreographies.

Both (i) and (ii) require choreographies to be composed before they are projected. In Chapter 3, we investigate how to extend code composition to choreographies developed and projected independently, so to support distributed choreographic programming and the integration of software projected by different vendors.

Endpoint code reuse. Endpoint code reuse may be necessary when parts of the system being designed are already implemented. For instance, we may want to reuse an existing identity provider service in the OpenID example in § 2.6.2. Our model does not currently offer a way of integrating existing endpoint code with the EPP of a choreography. We discuss some potential solutions. Using *bisimulation* techniques, we can verify some existing service code to be bisimilar to the code that would be generated by the EPP [26]. Alternatively, we could use a type system, such as multiparty session typing [55, 21], for guaranteeing that the existing code has a behaviour "compatible" with the choreography. We explore this second option in Chapter 3.

Both the techniques mentioned above can be adjusted to allow for refinement, i.e., the legacy code may do extra actions as long as they do not interfere with the good behaviour of the choreography. The resulting system would still guarantee communication safety and session fidelity (protocol compliance). Deadlock-freedom would also still be guaranteed, provided that the legacy code has been verified to be deadlock-free.

2.8.2 Extensions

Based on our conversations with our industry collaborators, we discuss some relevant aspects and extensions of the Choreography Calculus.

Sequential and Parallel operators. The relaxed semantics of our sequential operator ";" allows our framework to express parallelism with a minimal syntax. However, an explicit parallel operator may make choreographies more readable. We discuss here two possible extensions in this sense, based upon a hypothetical operator $C_1 \mid C_2$ equipped with the typical interleaving semantics of parallel composition. We use the following choreography as reference example:

$$C_{\mathsf{par}} \;=\; \mathsf{p}[\mathtt{A}].e \texttt{ -> } \mathsf{q}[\mathtt{B}].x : k \;\mid\; \mathsf{r}[\mathtt{C}].e' \texttt{ -> } \mathsf{s}[\mathtt{D}].y : k'$$

Whenever p, q, r and s are all different, C_{par} can be encoded using our sequential operator:

$$C_{\mathsf{seq}} \;=\; \mathsf{p}[\mathtt{A}].e \texttt{ -> } \mathsf{q}[\mathtt{B}].x : k;\; \mathsf{r}[\mathtt{C}].e' \texttt{ -> } \mathsf{s}[\mathtt{D}].y : k'$$

In fact, thanks to our swap relation $\simeq_{\mathcal{C}}$, C_{par} would behave exactly as C_{seq} . However, if the two interactions in C_{par} share a process name, e.g., $\mathsf{q} = \mathsf{s}$, the parallel operator $\mid$ would

not be syntax sugar anymore, since the projection of q would be a parallel composition of two input actions. This also means that q would no longer be a simple sequential process, raising the complexity of our framework and going out of the scope of this Chapter. In Chapter 3, we explore the formalisation of a parallel operator for choreographies.

Interactional exceptions. Our language does not offer specific features for error/exception recovery. A possible extension is to include exceptions in our choreography language in the spirit of what is informally suggested by [30] for binary sessions.

Dynamic join. We plan to extend our model to allow processes to dynamically join and leave an existing session, similarly to [28, 36]. We conjecture that asynchrony and parallelism will influence these extensions in a nontrivial way.

Multiple roles. In our model, a process can play only one role per session. However, there are cases in which multiple roles in a protocol may be implemented by a single process, which would be useful both for system simplification and resource saving. We plan to extend our typing system to allow for this occurrences. In [28, 17], the authors follow a similar direction, but using endpoint implementations.

Security. Global descriptions are particularly suitable for the study of security-related aspects in distributed systems [25, 29, 22]. Our choreographies give a global view of how sessions are interleaved and how their references are transmitted. It would be interesting to see how this aspect may influence the security analysis of a system.

2.9 Conclusions

In this Chapter, we presented a fully-fledged model for defining asynchronous system implementations as global programs. We developed a type system for checking choreographies against multiparty protocol specifications. Moreover, through type inference, developers can also use choreographies as implementation prototypes to infer new protocol standards. Our EPP generates correct endpoint code, ensuring nontrivial properties such as deadlock freedom and communication safety.

CHAPTER 3

Compositional Choreographies

3.1 Introduction

In Chapter 2 we have formalised a model, the Choreography Calculus (CC), for globally programming distributed systems by writing choreographies, and we have used global types as protocol specifications for the sessions implemented in a choreography. For example, a programmer may express a protocol for a choreography using the following global type:

$$\mathtt{B} \to \mathtt{C}\colon \langle \mathbf{string} \rangle; \mathtt{C} \to \mathtt{B}\colon \langle \mathbf{int} \rangle; \mathtt{B} \to \mathtt{T}\colon \left\{ \begin{array}{r} ok\colon \mathtt{B} \to \mathtt{T}\colon \langle \mathbf{string} \rangle; \mathtt{T} \to \mathtt{B}\colon \langle \mathbf{date} \rangle, \\ quit\colon \mathsf{end} \end{array} \right\}$$

Above, B, C and T are *roles* and abstractly represent endpoints in a system. In the protocol, a buyer B sends the name of a product to a catalogue C, which replies with the price for that product. Then, B notifies the transport role T of whether the price is accepted or not. In the first case (label ok), B sends also a delivery address to T and T replies with the expected delivery date. Otherwise (label $quit$), the protocol terminates immediately.

Our choreography calculus and, to the best of our knowledge, all previous choreography programming models (e.g., [61, 31]) require the programmer to implement the behaviour of all roles in a protocol where it is used; e.g., it would not be possible to write the choreography of a system that uses the protocol above but gives the implementations only of roles C and T, to make those reusable by other programs as software libraries through an API. This seriously hinders the applicability of choreographies in industrial settings, where the interoperability of different systems developed independently is the key. In particular, it is not currently possible to:

- use choreographies to develop software libraries that implement subsets of roles in protocols such that they can be reused from other systems;
- reuse an existing software library that implements subsets of roles in protocols from inside a choreography.

To tackle the issues above, we ask:

> *Can we design a choreography model in which the EPP of a choreography can be composed with other existing systems?*

The main problem is that existing choreography models rely on the complete knowledge of the implementation details of all endpoints to ensure that the systems generated by EPP will behave correctly. Such complete knowledge is not available when independently developed implementations of distributed protocols need to be composed.

In order to answer our question, we build a model for developing *partial choreographies*. Partial choreographies implement the behaviour of subsets of the roles in the protocols they use. Endpoint implementations are then automatically generated from partial choreographies and composed with other systems, with the guarantee that their overall execution will follow the intended protocols and the behaviour of the originating choreographies.

3.1.1 Contributions

In this Chapter we provide the following contributions.

Compositional Choreographies. We introduce a new language for choreographies in which the implementation of some roles in protocols can be omitted (§ 3.3). These *partial choreographies* can then be composed with others through message passing. Our language mixes choreography primitives, such as those of the Choreography Calculus from Chapter 2, with the typical sending/receiving actions of endpoint calculi, such as the π-calculus [69]. Building on our language, we show that it is possible to define a consistent semantics that homogeneously captures both systems with many participants or just a single endpoint (§ 3.3.2). We provide a notion of EPP that produces correct endpoint code from a choreography, and we show that the EPP of a choreography preserves its compositional properties (§ 3.5). Our model introduces *shared channel mobility* in choreographies, which gains a dynamism when two protocols are composed.

Typing. We provide a type system for checking choreographies against protocol specifications given as multiparty session types [55]. The type system ensures that the composition of different programs implements the intended protocols correctly (§ 3.4), and that our EPP produces code that follows the behaviour of the originating choreographies. Our framework guarantees that the EPP is still typable (§ 3.5); therefore, the EPP is reusable as a "black box" composable with other systems and the result of the composition can be checked for errors by referring only to types.

Deadlock-freedom and Progress among Composed Choreographies. In the presence of partial choreographies, we prove that we can (i) capture the existing methodologies for deadlock-freedom in complete choreographies as in [31] and Chapter 2 and (ii) extend the notion of progress for incomplete systems investigated in [55] to choreographies (§ 3.5). Our results demonstrate for the first time that choreographies can be effectively used also as a tool for progress in a compositional setting, offering a new viewpoint for investigating progress and giving a fresh look to the results in [31] and Chapter 2.

3.2 Motivation: a Use Case of Compositional Choreographies

We present motivations for this study by reporting a use case from our industry collaborators [6], and informally introducing our model. For clarity, we discuss only its most relevant parts. We report the extended version in Appendix B.1; a complete executable implementation in the Jolie language can be found at [72].

In our use case a buyer company needs to purchase a product from one of many available seller companies. The use case has two aspects that previous choreography models

cannot handle: (i) the system of the buyer company is developed independently from those of the seller companies, and use the latter as software modules without revealing internal implementation details; (ii) depending on the desired product, the buyer company selects a suitable seller company at runtime. We address these issues with *partial choreographies*. A partial choreography implements a subset of the roles in a protocol, leaving the implementation of the other roles to an external system. External systems can be discovered at runtime. In our case, the buyer company will select a seller and then run the protocol from the introduction by implementing only the buyer role B, and rely on the external seller system to implement the other two roles C and T.

Buyer Choreography. We now define a choreography for the buyer company, which we will refer to as C_B.

$C_B =$

1. u[U] **start** pd[PD] : $a(k)$; u[U].$prod$ -> pd[PD].x : k;
2. pd[PD] **start** r[R] : $b(k')$; pd[PD].x -> r[R].y : k'; r[R].find(y) -> pd[PD].z : k';
3. pd[B] **req** C, T : $z(k'')$; pd[B].x -> C : k''; C -> pd[B].$price$: k'';
4. if check($price$)@pd then
5. pd[B] -> T : $k'' \oplus ok$; pd[PD] -> u[U] : $k[del]$; pd[PD] -> u[U] : $k\langle k''[\mathtt{B}]\rangle$;
6. u[B].$addr$ -> T : k''; T -> u[B].$ddate$: k''
7. else
8. pd[B] -> T : $k'' \oplus quit$; pd[PD] -> u[U] : $k[quit]$

Above, a purchase in the buyer company is initiated by a user process u. In Line 1, process u and the freshly created process pd (for purchasing department) start a session k by synchronising on shared channel a. Each process is annotated with the role it plays in the protocol that the session implements. Then, still in Line 1, u sends the product $prod$ the user wishes to buy to pd. In Line 2 pd starts a new session k' with a fresh process r (a service registry) through shared channel b. Then, pd forwards the product name to r, which replies with the shared channel of the seller to contact for the purchase.

We refer to statements such as those in Lines 1–2 as complete, since they describe the behaviour of all participants, both sender and receiver(s). On the other hand, the continuation in Lines 3–8 is a partial choreography that relies on the selected external seller to implement the protocol shown in the introduction and perform the purchase.

The partial choreography in Lines 3–8 is depicted as a sequence chart in Figure 3.1.a, where dashed lines indicate interactions with external participants. In Line 3 pd *requests* a synchronisation on the shared channel stored in its local variable z to create the new session k'', declaring that it will play role B and that it expects the environment to implement roles C and T. Session k'' proceeds as specified by the protocol in the introduction. First, pd sends the product name stored in x through session k'' to the external process that is playing role C (the product catalogue executed by the seller company). Observe that here we do not specify the process name of the receiver, since that will be established by the external seller system. Then, pd waits to receive the price for the product from the external process playing role C in k''. In Line 4, pd checks whether the price is acceptable; if so, in Line 5 pd tells the external process playing role T (the transport process executed by the seller company) and user u (which remains internal to the buyer choreography) to proceed with the purchase (labels ok and del respectively). Still in Line 5, pd *delegates* to u the

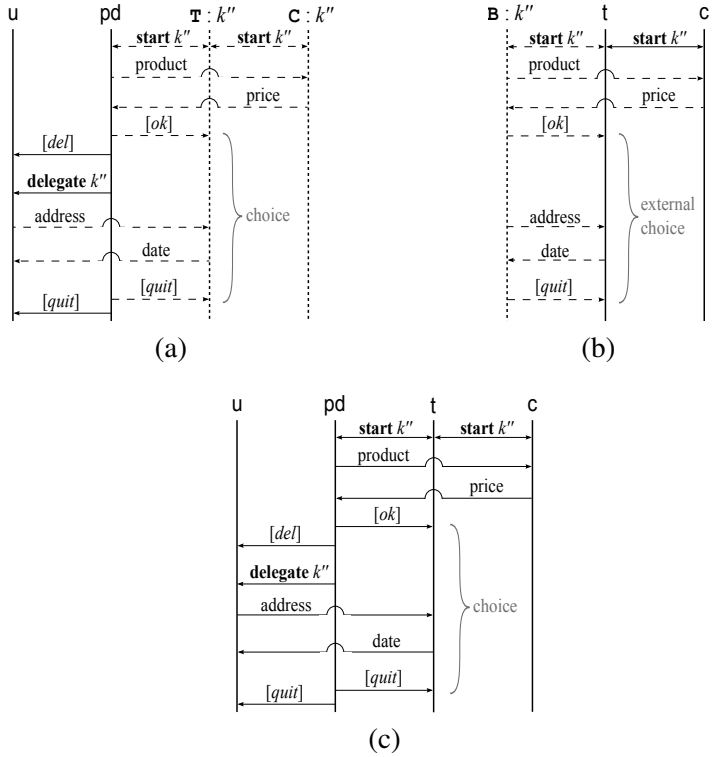


Figure 3.1: Sequence charts for buyer (a), seller (b), and their composition (c).

continuation of session k'' in its place, as role B. In Line 6, the user sends her address to T and receives a delivery date. If the price is not acceptable, Line 7, then in Line 8 pd informs the others to quit the purchase attempt.

Seller Choreography and Composition. We define now a choreography for a seller that can be contacted by C_B. Let the find function in C_B return shared channel c for electronic products, and c' for other products; we refer to the choreographies of the respective seller companies as C_S and C'_S. Below, we define C_S (C'_S, omitted, is similar).

$$C_S = \begin{array}{l} 1.\ \mathbf{acc}\ \mathtt{c[C]}, \mathtt{t[T]} : c(k'');\ \mathtt{B} \text{ -> } \mathtt{c[C]}.x_2 : k'';\ \mathtt{c[C]}.\mathsf{price}(x_2) \text{ -> } \mathtt{B} : k'';\\ 2.\ \mathtt{B} \text{ -> } \mathtt{t[T]} : k'' \& \left\{ \begin{array}{r} ok : \mathtt{B} \text{ -> } \mathtt{t[T]}.daddr : k'';\ \mathtt{t[T]}.\mathsf{time}(daddr) \text{ -> } \mathtt{B} : k'' \\ quit : \mathbf{0} \end{array} \right\} \end{array}$$

The choreography C_S, depicted as a sequence chart in Figure 3.1.b, starts by *accepting* the creation of session k'' through shared channel c, offering to spawn two fresh processes c and t. Choreographies starting with an acceptance act as replicated, modelling typical always-available modules. The acceptance in Line 1 would synchronise with the request made by C_B in the case $z = c$. Afterwards, c expects to receive the product name from the process playing B in session k'', and replies with the respective price. In Line 2, t (the process for the transport) waits for either label ok or $quit$. In the first case, t also waits for a delivery address and then sends back the expected time of arrival.

From the code of C_B and C_S and, graphically, from their respective sequence charts we can see that they are *compatible*: sending actions match receiving actions on the other

$$
\begin{array}{rcllll}
C & ::= & \eta; C & \textit{(seq)} & \mid & C_1 \mid C_2 \quad \textit{(par)} \\
 & \mid & \mathsf{if}\ e@\mathtt{p}\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2 & \textit{(cond)} & \mid & (\boldsymbol{\nu} r)\, C \quad \textit{(res)} \\
 & \mid & \mathsf{def}\ X(\tilde{D}) = C'\ \mathsf{in}\ C & \textit{(def)} & \mid & X\langle \tilde{E} \rangle \quad \textit{(call)} \\
 & \mid & \mathbf{0} & \textit{(inact)} & \mid & \mathtt{A} \texttt{->} q : k \& \{l_i : C_i\}_{i \in I} \quad \textit{(branch)}
\end{array}
$$

$$
\begin{array}{rcllll}
\eta & ::= & p\ \mathbf{start}\ \tilde{q} : a(k) & \textit{(start)} & \mid & p.e \texttt{->} q.x : k \quad \textit{(com)} \\
 & \mid & p \texttt{->} q : k[l] & \textit{(sel)} & \mid & p \texttt{->} q : k\langle k'[\mathtt{C}] \rangle \quad \textit{(del)} \\
 & \mid & p\ \mathbf{req}\ \tilde{\mathtt{B}} : u(k) & \textit{(req)} & \mid & \mathbf{acc}\ \tilde{q} : a(k) \quad \textit{(acc)} \\
 & \mid & p.e \texttt{->} \mathtt{B} : k & \textit{(com-s)} & \mid & \mathtt{A} \texttt{->} q.x : k \quad \textit{(com-r)} \\
 & \mid & p \texttt{->} \mathtt{B} : k\langle k'[\mathtt{C}] \rangle & \textit{(del-s)} & \mid & \mathtt{A} \texttt{->} q : k(k'[\mathtt{C}]) \quad \textit{(del-r)} \\
 & \mid & p \texttt{->} \mathtt{B} : k \oplus l & \textit{(sel-s)} & &
\end{array}
$$

$$
\begin{array}{rcl@{\qquad}rcl}
p, q & ::= & \mathtt{p}[\mathtt{A}] & u & ::= & x \mid a \\
D & ::= & \mathtt{p}(\tilde{x}, \tilde{k}) & E & ::= & \mathtt{p}(\tilde{e}, \tilde{k})
\end{array}
$$

Figure 3.2: Compositional Choreographies, syntax.

side and vice versa. Our model can recognise this by using roles in protocols as interfaces between partial choreographies (§ 3.4). The code for buyer and seller companies can be composed in a network with the parallel operator $\mid$ as: $C = C_B \mid C_S \mid C'_S$. Parallel composition allows partial terms in different choreographies to communicate. In § 3.3.2 we formalise a semantics for choreography composition. To give the intuition behind our semantics, let us consider the sequence charts in Figure 3.1.a and Figure 3.1.b; their composition will behave as the sequence chart in Figure 3.1.c.

3.3 A Calculus of Compositional Choreographies

This section introduces our model for compositional choreographies, a calculus where complete and partial actions can be freely interleaved.

3.3.1 Syntax

Figure 3.2 reports the syntax of our calculus. In the syntax, C is a choreography, η is a complete or partial action, p is a typed process identifier made by a process identifier $\mathtt{p}$ and a role annotation $\mathtt{A}$, k is a session identifier, and a is a shared channel. A term $\eta; C$ denotes a choreography that may execute action η and then proceed as C. In the productions for η, terms *(start)*, *(com)*, *(sel)* and *(del)* are complete actions, whereas all the others are partial. In the productions for C, term *(branch)* is also partial.

Complete Actions. Term *(start)* initiates a session: process $\mathtt{p}$ starts a new multiparty session through shared channel a and tags it with a fresh identifier k. $\mathtt{p}$ is already running and dubbed *active process*, while $\tilde{\mathtt{q}}$ (which we assume nonempty) is a set of bound *service processes* that are freshly created. $\mathtt{A}, \tilde{\mathtt{B}}$ represent the respective roles played by the processes in session k. Term *(com)* denotes a communication where process $\mathtt{p}$ sends, on

session k, the evaluation of a first-order expression e to process q, which binds it to its *local variable* x. Expressions may be shared channel names, capturing shared channel mobility. In *(sel)*, p communicates to q its selection of branch l. Term *(del)* models session mobility: process p delegates to q through session k its role $\mathtt{C}$ in session k'.

Partial Actions. In term *(req)*, process p is willing to start a new session k by synchronising through shared channel a with some other external processes. p is willing to play role $\mathtt{A}$ in the session and expects the other processes to play the other roles $\tilde{\mathtt{B}}$. *(req)* terms are supposed to synchronise with always-available *service processes*, modelled by term *(acc)*. In term *(acc)*, processes $\tilde{\mathsf{q}}$ are dynamically spawned whenever requested by a matching *(req)* term on the same shared channel a. Term *(com-s)* models the sending of a message from a process p to an external process playing role $\mathtt{B}$ in session k. Dually, in *(com-r)* process q receives a message intended for $\mathtt{B}$ in session k from the external process playing role $\mathtt{A}$. *(del-s)* and *(del-r)* model, respectively, the sending and receiving of a delegation of role $\mathtt{C}$ in session k'. *(sel-s)* models the sending of a selection of label l. *(sel-s)* can synchronise with a *(branch)* term, which offers a choice on multiple labels. Once a label l_i is selected, *(branch)* proceeds by executing its continuation C_i.

Other terms. In term *(cond)*, process p evaluates condition e to choose the continuation C_1 or C_2. Term *(res)* restricts the usage of a name r to a choreography C. r can be any name, i.e., a process identifier p, a session identifier k, or a shared channel a. Term *(par)* models the parallel composition of choreographies, allowing partial actions to interact through the network. The other terms are standard: terms *(def)*, *(call)* and *(inact)* model, respectively, a recursive procedure, a recursive call, and termination.

For clarity, we have annotated process identifiers with roles in all communications. Technically, this is necessary only for terms *(start)*, *(req)* and *(acc)* since roles can be inferred from session identifiers in all other terms (cf. Remark 2.3.1).

3.3.2 Semantics

We give semantics to choreographies with a labelled transition system (lts), whose rules are defined in Figure 3.3 and whose labels λ are defined as:

$$\begin{array}{rcl} \lambda & ::= & \eta \;\mid\; p \mathtt{ -> } q : k\langle v\rangle \;\mid\; p \mathtt{ -> } \mathtt{B} : k\langle v\rangle \;\mid\; \mathtt{A} \mathtt{ -> } q : k\langle v\rangle \\ & \mid & \mathtt{A} \mathtt{ -> } q : k\&l \;\mid\; \tau@\mathsf{p} \;\mid\; (\boldsymbol{\nu} r)\,\lambda \end{array}$$

We distinguish between labels representing complete or partial actions with the respective sets CAct and PAct. CAct is the smallest set containing (i) all η that are complete actions, (ii) complete communication labels of the form $p \mathtt{ -> } q : k\langle v\rangle$, and internal actions $\tau@\mathsf{p}$, closed under restrictions $(\boldsymbol{\nu} r)$. PAct is the smallest set containing all η that are partial actions, the partial communication labels $p \mathtt{ -> } \mathtt{B} : k\langle v\rangle$ and $\mathtt{A} \mathtt{ -> } q : k\langle v\rangle$, and the branching labels $\mathtt{A} \mathtt{ -> } q : k\&l$, similarly closed under name restrictions. We also use other auxiliary definitions. $\mathsf{fc}(C)$ returns the set of all session/role pairs $k[\mathtt{A}]$ such that k is free in C and there is a process performing an action as role $\mathtt{A}$ in session k in C. $\mathsf{rc}(\lambda)$ is defined only for partial labels that are not *(req)* or *(acc)*, and returns the session/role pair of the intended external sender or receiver of λ; e.g., $\mathsf{rc}(p.e \mathtt{ -> } \mathtt{B} : k) = k[\mathtt{B}]$. fn and bn denote the sets of free and bound names in a label or a term. $\mathsf{snd}(\eta)$ returns the name of the sender process in η, and is undefined if η has no sender process (e.g., when η is a *(com-r)*). $\mathsf{rcv}(\eta)$,

$$\frac{\eta \notin \{(com), (com\text{-}s), (com\text{-}r), (start), (acc)\}}{\eta; C \xrightarrow{\eta} C} \lfloor^{\mathrm{C}}|_{\textsc{Act}}\rceil$$

$$\frac{\eta = p\ \mathbf{start}\ \widetilde{\mathsf{q}[\mathtt{B}]} : a(k)}{\eta; C \xrightarrow{\eta} (\boldsymbol{\nu} k, \tilde{\mathsf{q}})\ C} \lfloor^{\mathrm{C}}|_{\textsc{Start}}\rceil \qquad \frac{\eta = p.e \texttt{ -> } \mathsf{q}[\mathtt{B}].x : k \quad e \downarrow v}{\eta; C \xrightarrow{p \texttt{ -> } \mathsf{q}[\mathtt{B}]:k\langle v\rangle} C[v/x@\mathsf{q}]} \lfloor^{\mathrm{C}}|_{\textsc{Com}}\rceil$$

$$\frac{\eta = p.e \texttt{ -> } \mathtt{B} : k \quad e \downarrow v}{\eta; C \xrightarrow{p \texttt{ -> } \mathtt{B}:k\langle v\rangle} C} \lfloor^{\mathrm{C}}|_{\textsc{Com-s}}\rceil \qquad \frac{\eta = \mathtt{A} \texttt{ -> } \mathsf{q}[\mathtt{B}].x : k}{\eta; C \xrightarrow{\mathtt{A} \texttt{ -> } \mathsf{q}[\mathtt{B}]:k\langle v\rangle} C[v/x@\mathsf{q}]} \lfloor^{\mathrm{C}}|_{\textsc{Com-r}}\rceil$$

$$\frac{j \in I}{\mathtt{A} \texttt{ -> } q : k\&\{l_i : C_i\}_{i\in I} \xrightarrow{\mathtt{A} \texttt{ -> } q:k\&l_j} C_j} \lfloor^{\mathrm{C}}|_{\textsc{Branch}}\rceil \qquad \frac{C_1 \xrightarrow{\lambda} C_1' \quad C_2 \xrightarrow{\lambda'} C_2'}{C_1 \mid C_2 \xrightarrow{\lambda \circ \lambda'} C_1' \mid C_2'} \lfloor^{\mathrm{C}}|_{\textsc{Sync}}\rceil$$

$$\frac{i = 1 \text{ if } e \downarrow \mathsf{true}, i = 2 \text{ otherwise}}{\mathsf{if}\ e@\mathsf{p}\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2 \xrightarrow{\tau@\mathsf{p}} C_i} \lfloor^{\mathrm{C}}|_{\textsc{Cond}}\rceil \qquad \frac{C \xrightarrow{\lambda} C'}{(\boldsymbol{\nu} r)\ C \xrightarrow{(\boldsymbol{\nu} r)\ \lambda} (\boldsymbol{\nu} r)\ C'} \lfloor^{\mathrm{C}}|_{\textsc{Res}}\rceil$$

$$\frac{C_1 \xrightarrow{\lambda} C_1'}{\mathsf{def}\ X(\tilde{D}) = C_2\ \mathsf{in}\ C_1 \xrightarrow{\lambda} \mathsf{def}\ X(\tilde{D}) = C_2\ \mathsf{in}\ C_1'} \lfloor^{\mathrm{C}}|_{\textsc{Ctx}}\rceil$$

$$\frac{C_1 \xrightarrow{\lambda} C_1' \quad \big(\lambda \in \mathsf{CAct} \ \vee\ \mathsf{rc}(\lambda) \notin \mathsf{fc}(C_2)\big)}{C_1 \mid C_2 \xrightarrow{\lambda} C_1' \mid C_2} \lfloor^{\mathrm{C}}|_{\textsc{Par}}\rceil$$

$$\frac{\mathcal{R} \in \{\equiv, \simeq_{\mathcal{C}}\} \quad C_1\ \mathcal{R}\ C_1' \quad C_1' \xrightarrow{\lambda} C_2' \quad C_2'\ \mathcal{R}\ C_2}{C_1 \xrightarrow{\lambda} C_2} \lfloor^{\mathrm{C}}|_{\textsc{Eq}}\rceil$$

$$\frac{\begin{array}{c} i \in [1,n] \quad \{\tilde{\mathsf{q}}\} = \{\tilde{\mathsf{q}}_1, \ldots, \tilde{\mathsf{q}}_n\} \quad \{\tilde{\mathtt{B}}\} = \{\tilde{\mathtt{B}}_1, \ldots, \tilde{\mathtt{B}}_n\} \quad C \xrightarrow{p\ \mathbf{req}\ \tilde{\mathtt{B}}:u(k)} C' \\ C_i = \mathbf{acc}\ \widetilde{\mathsf{q}[\mathtt{B}]}_i : a(k); C_i' \quad C'' = \prod_i C_i \quad \lambda = p\ \mathbf{start}\ \widetilde{\mathsf{q}[\mathtt{B}]}_1, \ldots, \widetilde{\mathsf{q}[\mathtt{B}]}_n : a(k) \end{array}}{C \mid C'' \xrightarrow{\lambda} (\boldsymbol{\nu} k, \tilde{\mathsf{q}})\ \Big(C' \mid \prod_i (C_i')\Big) \mid C''} \lfloor^{\mathrm{C}}|_{\textsc{P-Start}}\rceil$$

$$\frac{\begin{array}{c} C \xrightarrow{\lambda} (\boldsymbol{\nu} \tilde{r})\ C' \quad \mathsf{snd}(\eta) \in \mathsf{fn}(\lambda) \quad \mathsf{rcv}(\eta) \notin \mathsf{fc}(\lambda) \\ \tilde{r} = \mathsf{bn}(\lambda) \quad \tilde{r} \notin \mathsf{fn}(\eta) \quad \eta \notin \{(start), (acc)\} \quad \mathsf{sdc}(\eta, \lambda) \end{array}}{\eta; C \xrightarrow{\lambda} (\boldsymbol{\nu} \tilde{r})\ \eta; C'} \lfloor^{\mathrm{C}}|_{\textsc{Async}}\rceil$$

Figure 3.3: Compositional Choreographies, semantics.

instead, returns the session/role pair $k[\mathtt{A}]$ where k is the session used in η and $\mathtt{A}$ is the role of the receiver (similarly for $\mathsf{rcv}(\lambda)$). $\mathsf{fc}(\lambda)$ is as $\mathsf{fc}(C)$, but applied on labels. We comment

the rules. Rule $\lfloor^{\mathsf{C}}|_{\textsc{Act}}\rceil$ handles actions that can be simply consumed. Rule $\lfloor^{\mathsf{C}}|_{\textsc{Start}}\rceil$ starts a session with a global action, by restricting the names of the newly created session identifier k and processes $\tilde{\mathsf{q}}$. Rule $\lfloor^{\mathsf{C}}|_{\textsc{Com}}\rceil$ handles the communication of a value by substituting, in the continuation C, the binding occurrence x under process identifier q with value v (evaluated from expression e). Similarly, rules $\lfloor^{\mathsf{C}}|_{\textsc{Com-S}}\rceil$ and $\lfloor^{\mathsf{C}}|_{\textsc{Com-R}}\rceil$ implement the respective partial sending and receiving actions of a communication. In rule $\lfloor^{\mathsf{C}}|_{\textsc{Branch}}\rceil$, process q receives a selection on a branching label and proceeds accordingly. Rules $\lfloor^{\mathsf{C}}|_{\textsc{Cond}}\rceil$, $\lfloor^{\mathsf{C}}|_{\textsc{Res}}\rceil$, and $\lfloor^{\mathsf{C}}|_{\textsc{Ctx}}\rceil$ are standard. Rule $\lfloor^{\mathsf{C}}|_{\textsc{Par}}\rceil$ makes global actions observable and blocks partial actions if their counterpart is in the parallel branch C_2.

Rule $\lfloor^{\mathsf{C}}|_{\textsc{Sync}}\rceil$ is the main rule and enables two choreographies to perform compatible sending/receiving partial actions λ and λ' to interact and realise a global action, defined by $\lambda \circ \lambda'$. The function $\circ : \mathsf{PAct} \times \mathsf{PAct} \to \mathsf{CAct}$ is formally defined by the rules below:

$$\begin{array}{lcll}
\mathtt{p}[\mathtt{A}] \to \mathtt{B} : k\langle v\rangle & \circ & \mathtt{A} \to \mathtt{q}[\mathtt{B}] : k\langle v\rangle & = \quad \mathtt{p}[\mathtt{A}] \to \mathtt{q}[\mathtt{B}] : k\langle v\rangle \\
\mathtt{p}[\mathtt{A}] \to \mathtt{B} : k\langle k'[\mathtt{C}]\rangle & \circ & \mathtt{A} \to \mathtt{q}[\mathtt{B}] : k(k'[\mathtt{C}]) & = \quad \mathtt{p}[\mathtt{A}] \to \mathtt{q}[\mathtt{B}] : k\langle k'[\mathtt{C}]\rangle \\
\mathtt{p}[\mathtt{A}] \to \mathtt{B} : k \oplus l & \circ & \mathtt{A} \to \mathtt{q}[\mathtt{B}] : k\&l & = \quad \mathtt{p}[\mathtt{A}] \to \mathtt{q}[\mathtt{B}] : k[l]
\end{array}$$

Observe that if $\lambda \circ \lambda'$ is not defined (the actions are incompatible), then the rule cannot be applied. Similarly, $\lfloor^{\mathsf{C}}|_{\textsc{p-Start}}\rceil$ models a session start by synchronising a partial choreography that requests to start a session with other choreographies that can accept the request on the same shared channel. The choreographies accepting the request remain available afterwards, for reuse.

Rule $\lfloor^{\mathsf{C}}|_{\textsc{Async}}\rceil$ models asynchrony, allowing the sender process of an interaction η ($\mathsf{snd}(\eta)$) to send a message and then proceed freely before the intended receiver actually receives it. In the rule, we require asynchrony to preserve the message ordering in a session wrt receivers with a causality check: $\mathsf{rcv}(\eta) \notin \mathsf{fc}(\lambda)$ and $\mathsf{sdc}(\eta, \lambda)$ (the sender channel in η is not delegated in λ).

In rule $\lfloor^{\mathsf{C}}|_{\textsc{Eq}}\rceil$, the relation $\mathcal{R}$ can either be the swapping relation $\simeq_{\mathcal{C}}$, which swaps terms that describe the behaviour of different processes, or the structural congruence $\equiv$, which handles name restriction and recursion unfolding. The relations $\simeq_{\mathcal{C}}$ and $\equiv$ are defined as the smallest relations satisfying, respectively, the rules reported in Figure 3.4 and Figure 3.5. The rules follow the same reasoning and are an extension of those discussed in § 2.3.2, adding support for dealing with partial choreographies in $\simeq_{\mathcal{C}}$ and the parallel operator in $\equiv$.

Example 3.3.1 (Label composition)**.** The composition of labels performed by rules $\lfloor^{\mathsf{C}}|_{\textsc{Sync}}\rceil$ and $\lfloor^{\mathsf{C}}|_{\textsc{p-Start}}\rceil$ is a key aspect of our model. Consider the following two choreographies, C_C and C_P:

$$\begin{array}{rcl}
C_C & = & \mathtt{p}[\mathtt{A}].v \to \mathtt{q}[\mathtt{B}].x : k \\
C_P & = & \mathtt{p}[\mathtt{A}].v \to \mathtt{B} : k \mid \mathtt{A} \to \mathtt{q}[\mathtt{B}].x : k
\end{array}$$

The choreographies C_C and C_P describe the same system, in which a process p sends a value v to another process q over session k. However, C_C is a complete choreography and therefore describes the communication between p and q atomically, i.e., using a single complete statement. By contrast, C_P is the parallel composition of two partial choreographies, one defining the sending action of p towards q and the other defining the receiving action of q for a message from p.

$$\frac{\mathsf{pn}(\eta) \cap \mathsf{pn}(\eta') \;=\; \emptyset}{\eta;\eta' \;\simeq_{\mathcal{C}}\; \eta';\eta} \;\lfloor^{\text{CS}}|_{\text{ETA-ETA}}\rceil$$

$$\frac{\mathsf{p} \neq \mathsf{q}}{\begin{array}{c}\mathsf{if}\ e@\mathsf{p}\ \mathsf{then}\ (\ \mathsf{if}\ e'@\mathsf{q}\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2\ )\ \mathsf{else}\ (\ \mathsf{if}\ e'@\mathsf{q}\ \mathsf{then}\ C_1'\ \mathsf{else}\ C_2'\ ) \\ \simeq_{\mathcal{C}} \\ \mathsf{if}\ e'@\mathsf{q}\ \mathsf{then}\ (\mathsf{if}\ e@\mathsf{p}\ \mathsf{then}\ C_1\ \mathsf{else}\ C_1')\ \mathsf{else}\ (\mathsf{if}\ e@\mathsf{p}\ \mathsf{then}\ C_2\ \mathsf{else}\ C_2')\end{array}} \;\lfloor^{\text{CS}}|_{\text{COND-COND}}\rceil$$

$$\frac{\mathsf{p} \notin \mathsf{pn}(\eta)}{\mathsf{if}\ e@\mathsf{p}\ \mathsf{then}\ (\eta;C_1)\ \mathsf{else}\ (\eta;C_2)\ \simeq_{\mathcal{C}}\ \eta;(\mathsf{if}\ e@\mathsf{p}\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2)} \;\lfloor^{\text{CS}}|_{\text{ETA-COND}}\rceil$$

$$\frac{\mathsf{q} \notin \mathsf{pn}(\eta)}{\mathtt{A}\ \texttt{->}\ \mathsf{q}[\mathtt{B}] : k\&\{l_i : \eta; C_i\}_{i\in I}\ \simeq_{\mathcal{C}}\ \eta;\mathtt{A}\ \texttt{->}\ \mathsf{q}[\mathtt{B}] : k\&\{l_i : C_i\}_{i\in I}} \;\lfloor^{\text{CS}}|_{\text{ETA-BRA}}\rceil$$

$$\frac{\mathsf{p} \neq \mathsf{q}}{\begin{array}{c}\mathtt{A}\ \texttt{->}\ \mathsf{p}[\mathtt{B}] : k\&\{\ l_i : \mathtt{C}\ \texttt{->}\ \mathsf{q}[\mathtt{D}] : k'\&\{l'_{ij} : C_{ij}\}_{j\in J}\ \}_{i\in I} \\ \simeq_{\mathcal{C}} \\ \mathtt{C}\ \texttt{->}\ \mathsf{q}[\mathtt{D}] : k'\&\{\ l'_j : \mathtt{A}\ \texttt{->}\ \mathsf{p}[\mathtt{B}] : k\&\{l_{ij} : C_{ij}\}_{i\in I}\ \}_{j\in J}\end{array}} \;\lfloor^{\text{CS}}|_{\text{BRA-BRA}}\rceil$$

$$\frac{\mathsf{p} \neq \mathsf{q}}{\begin{array}{c}\mathtt{A}\ \texttt{->}\ \mathsf{p}[\mathtt{B}] : k\&\{\ l_i : \mathsf{if}\ e@\mathsf{q}\ \mathsf{then}\ C_{i1}\ \mathsf{else}\ C_{i2}\ \}_{i\in I} \\ \simeq_{\mathcal{C}} \\ \mathsf{if}\ e@\mathsf{q}\ \mathsf{then}\ (\ \mathtt{A}\ \texttt{->}\ \mathsf{p}[\mathtt{B}] : k\&\{l_i : C_{i1}\}_{i\in I}\ )\ \mathsf{else}\ (\ \mathtt{A}\ \texttt{->}\ \mathsf{p}[\mathtt{B}] : k\&\{l_i : C_{i2}\}_{i\in I}\ )\end{array}} \;\lfloor^{\text{CS}}|_{\text{BRA-COND}}\rceil$$

Figure 3.4: Compositional Choreographies, swap relation $\simeq_{\mathcal{C}}$.

The intuition behind the design of our semantics is that choreographies describing the same system should be indistinguishable by looking at their respective observable actions, regardless of whether they are complete or partial. We have used this observation as a consistency principle for mixing choreographies with typical sending/receiving actions. Formally, by our semantics we can easily see that both C_C and C_P have a unique transition, with the same label:

$$C_C \xrightarrow{\mathsf{p}[\mathtt{A}]\ \texttt{->}\ \mathsf{q}[\mathtt{B}]:k\langle v\rangle} \mathbf{0}$$

$$C_P \xrightarrow{\mathsf{p}[\mathtt{A}]\ \texttt{->}\ \mathsf{q}[\mathtt{B}]:k\langle v\rangle} \mathbf{0}$$

In § 3.5, we exploit this aspect of our semantics to define a tight correspondence between choreographies and their EPP. □

3.4 Typing Compositional Choreographies

We now present our typing discipline, which ensures that sessions in a choreography follow protocol specifications given as global types [55, 21]. The key advances from Chapter 2 are:

$$(\boldsymbol{\nu} r)\ \mathbf{0} \equiv \mathbf{0} \quad (\boldsymbol{\nu} r)\ (\boldsymbol{\nu} r')\ C \equiv (\boldsymbol{\nu} r')\ (\boldsymbol{\nu} r)\ C \quad \mathsf{def}\ X(\tilde{D}) = C\ \mathsf{in}\ \mathbf{0} \ \equiv\ \mathbf{0}$$

$$\mathsf{def}\ X(\tilde{D}) = C'\ \mathsf{in}\ \big((\boldsymbol{\nu} r)\ C\big) \ \equiv\ (\boldsymbol{\nu} r)\ \big(\mathsf{def}\ X(\tilde{D}) = C'\ \mathsf{in}\ C\big)\ \text{ if }\ r \notin \mathsf{fn}(C')$$

$$\mathsf{def}\ X(\widetilde{\mathsf{p}(\tilde{x}, \tilde{k})}) = C'\ \mathsf{in}\ C[X\langle \widetilde{\mathsf{p}(\tilde{e}, \tilde{k})} \rangle] \ \equiv\ \mathsf{def}\ X(\widetilde{\mathsf{p}(\tilde{x}, \tilde{k})}) = C'\ \mathsf{in}\ C[C'[\widetilde{\tilde{e}_i / \tilde{x}_i @ \mathsf{p}_i}]]$$

$$C \mid C' \ \equiv\ C' \mid C \qquad ((\boldsymbol{\nu} r)\ C) \mid C' \ \equiv\ (\boldsymbol{\nu} r)\ (C \mid C') \text{ if } r \notin \mathsf{fn}(C')$$

$$(C_1 \mid C_2) \mid C_3 \ \equiv\ C_1 \mid (C_2 \mid C_3)$$

Figure 3.5: Compositional Choreographies, structural congruence $\equiv$.

$$\begin{array}{rcl} G & ::= & \mathtt{A} \texttt{ -> } \mathtt{B} : \langle U \rangle ; G \mid \mathtt{A} \texttt{ -> } \mathtt{B} : \{l_i : G_i\}_{i \in I} \mid \mathsf{rec}\, \mathbf{t} ; G \mid \mathbf{t} \mid \mathsf{end} \\ T & ::= & !\mathtt{A}\langle U \rangle ; T \mid ?\mathtt{A}\langle U \rangle ; T \mid \ \oplus \mathtt{A}\{l_i : T_i\}_{i \in I} \mid \&\mathtt{A}\{l_i : T_i\}_{i \in I} \mid \mathsf{rec}\, \mathbf{t} ; T \mid \mathbf{t} \mid \mathsf{end} \\ S & ::= & G \mid \mathbf{int} \mid \mathbf{bool} \mid \ldots \qquad U \ ::= \ S \mid T \end{array}$$

Figure 3.6: Global and Local Types, syntax.

(i) introduction of the typing rules for partial choreographies and shared channel passing; and (ii) typing endpoints by local types, which offer transparent compositional properties for the behaviour of each process.

3.4.1 Types

We have two kinds of types: global types, used for defining protocols, and local types, used for defining the local behaviour of each endpoint in a protocol. Global types are related to local types with a notion of type projection, taken from [105].

3.4.1.1 Syntax

The syntax of global and local types is reported in Figure 3.6.

In the syntax, G is a global type and T is a local type. We first discuss global types. A global type $\mathtt{A}$ -> $\mathtt{B} : \langle U \rangle ; G$ abstracts a communication from role $\mathtt{A}$ to role $\mathtt{B}$ with continuation G, where U is the type of the exchanged message; U can either be a sort type S (used for typing values or shared channels), or a local type T (used for typing session delegation). In $\mathtt{A}$ -> $\mathtt{B} : \{l_i : G_i\}_{i \in I}$, role $\mathtt{A}$ selects one label l_i offered by role $\mathtt{B}$ and the global type proceeds as G_i. All other terms are standard.

We discuss now the syntax of local types. A local type $!\mathtt{A}\langle U \rangle ; T$ represents the sending of a message of type U to role $\mathtt{A}$, with continuation T. Dually, type $?\mathtt{A}\langle U \rangle ; T$ represents the receiving of a message of type U from role $\mathtt{A}$. Types $\oplus \mathtt{A}\{l_i : T_i\}_{i \in I}$ and $\&\mathtt{A}\{l_i : T_i\}_{i \in I}$ abstract the selection and the offering of some branches. The other terms are standard.

To relate a global type to the behaviour of an endpoint, we project a global type G onto a local type that represents the behaviour of a single role. We write $[\![G]\!]_{\mathtt{A}}$ to denote the projection of G onto the role $\mathtt{A}$. Formally, $[\![G]\!]_{\mathtt{A}}$ is inductively defined by the rules in

$$\llbracket \mathtt{A} \to \mathtt{B} : \langle U \rangle ; G \rrbracket_{\mathtt{C}} = \begin{cases} !\mathtt{B}\langle U \rangle ; \llbracket G \rrbracket_{\mathtt{C}} & \text{if } \mathtt{C} = \mathtt{A} \\ ?\mathtt{A}\langle U \rangle ; \llbracket G \rrbracket_{\mathtt{C}} & \text{if } \mathtt{C} = \mathtt{B} \\ \llbracket G \rrbracket_{\mathtt{C}} & \text{otherwise} \end{cases}$$

$$\llbracket \mathtt{A} \to \mathtt{B} : \{ l_i : G_i \}_{i \in I} \rrbracket_{\mathtt{C}} = \begin{cases} \oplus \mathtt{B} \{ l_i : \llbracket G_i \rrbracket_{\mathtt{C}} \}_{i \in I} & \text{if } \mathtt{C} = \mathtt{A} \\ \& \mathtt{A} \{ l_i : \llbracket G_i \rrbracket_{\mathtt{C}} \}_{i \in I} & \text{if } \mathtt{C} = \mathtt{B} \\ \bigsqcup_{i \in I} \llbracket G_i \rrbracket_{\mathtt{C}} & \text{otherwise} \end{cases}$$

$$\llbracket \mathsf{rec}\, \mathbf{t} ; G \rrbracket_{\mathtt{A}} = \mathsf{rec}\, \mathbf{t} ; \llbracket G \rrbracket_{\mathtt{A}} \quad (\text{if } \mathtt{A} \in \mathsf{roles}(G)) \qquad \llbracket \mathsf{rec}\, \mathbf{t} ; G \rrbracket_{\mathtt{A}} = \mathsf{end} \quad (\text{otherwise})$$

$$\llbracket \mathbf{t} \rrbracket_{\mathtt{A}} = \mathbf{t} \qquad \llbracket \mathsf{end} \rrbracket_{\mathtt{A}} = \mathsf{end}$$

Figure 3.7: Global Types, type projection.

Figure 3.7. The rules follow the same intuition of the EPP presented in § 2.5.2, yielding the local action of the role we are projecting for each given global type. In the rule for projecting a branching, we require the local behaviour of all roles univolved in the choice to be *merged* with the merging operator for local types $\sqcup$. Formally, $T \sqcup T'$ is isomorphic to T and T' up to branching, where all branches of T or T' with distinct labels are also included (this is just a reformulation for local types of the merging operator for choreographies described in § 2.5.2). Our definition of projection from global to local types is the same as in [105], where it is proven sound: global types and their projections have the same behaviour. We formalise the semantics of types in the following.

3.4.1.2 Semantics

Semantics of Global Types. We give a semantics to global and local types for expressing the (abstract) execution of protocols. $G \xrightarrow{\alpha} G'$ is the smallest relation on the recursion-unfolding of global types satisfying the rules reported in Figure 3.8. The semantics of global types is the same as that presented in § 2.4.1.2; we report the rules again here for the reader's convenience.

Semantics of Local Types. The semantics of local types is defined as a labelled transition system on typing environments Δ of the following form:

$$\Delta ::= k[\mathtt{A}] : T, \Delta \mid \emptyset$$

Formally, the semantics of local typings $\Delta \xrightarrow{\alpha} \Delta'$ is the smallest relation closed under the rules reported in Figure 3.9. The rules follow the intuition of the semantics for partial actions in choreographies. Rules $\lfloor^{\mathsf{L}}|_{\textsc{send}}\rceil$ and $\lfloor^{\mathsf{L}}|_{\textsc{recv}}\rceil$ model respectively the sending and receiving of values. $\lfloor^{\mathsf{L}}|_{\textsc{sel}}\rceil$ and $\lfloor^{\mathsf{L}}|_{\textsc{branch}}\rceil$ abstract selection. Rules $\lfloor^{\mathsf{L}}|_{\textsc{asend}}\rceil$ and $\lfloor^{\mathsf{L}}|_{\textsc{asel}}\rceil$ capture asynchrony. In rule $\lfloor^{\mathsf{L}}|_{\textsc{conc}}\rceil$, $\mathsf{rc}(\alpha)$ is similar to $\mathsf{rc}(\lambda)$. Finally, $\lfloor^{\mathsf{L}}|_{\textsc{sync}}\rceil$ synchronise (abstract) actions that are compatible according to the composition function for abstract labels $\circ$. Intuitively, $\alpha \circ \alpha'$ is defined for compatible sending/receiving actions and returns the corresponding global label α; it is formally defined below:

$$k[\mathtt{A}] : !\mathtt{B}\langle U \rangle \;\circ\; k[\mathtt{B}] : ?\mathtt{A}\langle U \rangle \;=\; k : \mathtt{A} \to \mathtt{B} : \langle U \rangle$$

$$k[\mathtt{A}] : \oplus \mathtt{B}\langle l \rangle \;\circ\; k[\mathtt{B}] : \& \mathtt{A}\langle l \rangle \;=\; k : \mathtt{A} \to \mathtt{B} : [l]$$

All other rules are standard.

$$\frac{}{\mathtt{A} \to \mathtt{B} : \langle U \rangle ; G \xrightarrow{\mathtt{A} \to \mathtt{B}:\langle U \rangle} G} \lfloor^{\mathrm{G}}|_{\textsc{Com}}\rceil$$

$$\frac{}{\mathtt{A} \to \mathtt{B} : \{l_i : G_i\}_{i \in I \cup \{j\}} \xrightarrow{\mathtt{A} \to \mathtt{B}:[l_j]} G_j} \lfloor^{\mathrm{G}}|_{\textsc{Branch}}\rceil$$

$$\frac{G[\mathsf{rec}\,\mathbf{t}; G/\mathbf{t}] \xrightarrow{\alpha} G'}{\mathsf{rec}\,\mathbf{t}; G \xrightarrow{\alpha} G'} \lfloor^{\mathrm{G}}|_{\textsc{Rec}}\rceil \qquad \frac{G_1 \simeq_{\mathcal{G}} G_1' \xrightarrow{\alpha} G_2' \simeq_{\mathcal{G}} G_2}{G_1 \xrightarrow{\alpha} G_2} \lfloor^{\mathrm{G}}|_{\textsc{Swap}}\rceil$$

$$\frac{G \xrightarrow{\alpha} G' \quad \mathtt{A} \in \mathsf{roles}(\alpha), \mathtt{B} \notin \mathsf{roles}(\alpha)}{\mathtt{A} \to \mathtt{B} : \langle U \rangle ; G \xrightarrow{\alpha} \mathtt{A} \to \mathtt{B} : \langle U \rangle ; G'} \lfloor^{\mathrm{G}}|_{\textsc{ACom}}\rceil$$

$$\frac{G_j \xrightarrow{\alpha} G_j' \quad \mathtt{A} \in \mathsf{roles}(\alpha), \mathtt{B} \notin \mathsf{roles}(\alpha)}{\mathtt{A} \to \mathtt{B} : \{l_i : G_i\}_{i \in I \cup \{j\}} \xrightarrow{\alpha} \mathtt{A} \to \mathtt{B} : \{l_j : G_j'\}} \lfloor^{\mathrm{G}}|_{\textsc{ABranch}}\rceil$$

Figure 3.8: Global Types, semantics.

3.4.2 Type checking

We now introduce our type checking discipline for checking compositional choreographies against global types. We use two kinds of typing environments, the *unrestricted environments* Γ and the *session environments* Δ. Their syntax is reported in Figure 3.10. The typing environments are nearly identical to those used for the Choreography Calculus in Chapter 2. The only differences are the service typings $a : G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{C}}\rangle$ in Γ and the local session typings $k[\mathtt{A}] : T$ in Δ. The syntax for Δ is taken from [21], where $k[\mathtt{A}] : T$ maps a local type T to a role $\mathtt{A}$ in a session k. In Γ, $x@\mathtt{p} : S$ types variable x of process $\mathtt{p}$ with type S. $X(\tilde{D}) : (\Gamma; \Delta)$ types recursive procedure X. $\mathtt{p} : k[\mathtt{A}]$ establishes that process $\mathtt{p}$ *owns* role $\mathtt{A}$ in session k. $a : G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{C}}\rangle$ types a shared channel a with global type G: $\mathtt{A}$ is the role of the active process that starts the session through a; $\tilde{\mathtt{B}}$ are the roles of the service processes; $\tilde{\mathtt{C}}$ are the roles, in $\tilde{\mathtt{B}}$, that a choreography implements for the shared channel a, enabling compositionality of services. Whenever we write $a : G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{C}}\rangle$ in Γ, we assume that $\tilde{\mathtt{C}} \subseteq \tilde{\mathtt{B}}$, $\mathtt{A} \notin \tilde{\mathtt{B}}$, and that $\mathtt{A}, \tilde{\mathtt{B}} = \mathsf{roles}(G)$. $\mathsf{roles}(G)$ returns the set of roles in a global type G.

We can write $\Gamma, \mathtt{p} : k[\mathtt{A}]$ only if $\mathtt{p}$ is not associated to any other role in session k in Γ (a process may only play one role per session). A process $\mathtt{p}$ may however appear more than once in a same Γ, allowing processes to run multiple sessions. Similarly, a session k may appear more than once in a same Δ, provided that it is always associated to different roles. As usual, we require all other kinds of occurrences in environments to have disjoint identifiers.

A typing judgement $\Gamma \vdash C \triangleright \Delta$ establishes that a choreography C is well-typed. Intuitively, C is well-typed if shared channels are used according to Γ and sessions are used according to Δ. Δ gives the session types of the free sessions in C. Following the design idea that services should always be available, shared by other models [31], in the remainder we assume that all *(acc)* terms in a choreography are not guarded by other actions. A selection of the rules defining our typing judgement is reported in Figure 3.11;

$$\frac{}{\mathtt{!A}\langle U\rangle;T \xrightarrow{\mathtt{!A}\langle U\rangle} T}\ \lfloor^{\mathrm{L}}|_{\textsc{Send}}\rceil \qquad \frac{}{?\mathtt{A}\langle U\rangle;T \xrightarrow{?\mathtt{A}\langle U\rangle} T}\ \lfloor^{\mathrm{L}}|_{\textsc{Recv}}\rceil$$

$$\frac{}{\oplus\mathtt{A}\{l_i : T_i\}_{i\in I\cup\{j\}} \xrightarrow{\oplus\mathtt{A}\langle l_j\rangle} T_j}\ \lfloor^{\mathrm{L}}|_{\textsc{Sel}}\rceil \qquad \frac{}{\&\mathtt{A}\{l_i : T_i\}_{i\in I\cup\{j\}} \xrightarrow{\&\mathtt{A}\langle l_j\rangle} T_j}\ \lfloor^{\mathrm{L}}|_{\textsc{Bra}}\rceil$$

$$\frac{T[\mathsf{rec}\,\mathbf{t};T/\mathbf{t}] \xrightarrow{\alpha} T'}{\mathsf{rec}\,\mathbf{t};T \xrightarrow{\alpha} T'}\ \lfloor^{\mathrm{L}}|_{\textsc{Rec}}\rceil \qquad \frac{T \xrightarrow{\alpha} T',\ \mathtt{A}\notin\mathsf{roles}(\alpha)}{\mathtt{!A}\langle U\rangle;T \xrightarrow{\alpha} \mathtt{!A}\langle U\rangle;T'}\ \lfloor^{\mathrm{L}}|_{\textsc{ASend}}\rceil$$

$$\frac{T_j \xrightarrow{\alpha} T_j',\ \mathtt{A}\notin\mathsf{roles}(\alpha)}{\oplus\mathtt{A}\{l_i : T_i\}_{i\in I\cup\{j\}} \xrightarrow{\alpha} \oplus\mathtt{A}\{l_j : T_J'\}}\ \lfloor^{\mathrm{L}}|_{\textsc{ASel}}\rceil \qquad \frac{T \xrightarrow{\alpha} T'}{k[\mathtt{A}]:T \xrightarrow{k[\mathtt{A}]:\alpha} k[\mathtt{A}]:T'}\ \lfloor^{\mathrm{L}}|_{\textsc{Lift}}\rceil$$

$$\frac{\Delta' \xrightarrow{\alpha} \Delta''\quad \big(\ \mathsf{rc}(\alpha)\notin\Delta\ \big)}{\Delta,\Delta' \xrightarrow{\alpha} \Delta,\Delta''}\ \lfloor^{\mathrm{L}}|_{\textsc{Conc}}\rceil \qquad \frac{\Delta_1 \xrightarrow{\alpha} \Delta_1'\quad \Delta_2 \xrightarrow{\alpha'} \Delta_2'}{\Delta_1,\Delta_2 \xrightarrow{\alpha\circ\alpha'} \Delta_1',\Delta_2'}\ \lfloor^{\mathrm{L}}|_{\textsc{Sync}}\rceil$$

Figure 3.9: Local Types, semantics.

(Unrestricted Env.)	Γ	$::=$	$\emptyset$	*(empty env.)*
		$\mid$	$\Gamma, a : G\langle \mathtt{A}\mid\tilde{\mathtt{B}}\mid\tilde{\mathtt{C}}\rangle$	*(service)*
		$\mid$	$\Gamma, x@\mathsf{p} : S$	*(variable)*
		$\mid$	$\Gamma, X(\tilde{D}) : (\Gamma;\Delta)$	*(definition)*
		$\mid$	$\Gamma, \mathsf{p} : k[\mathtt{A}]$	*(ownership)*
(Session Env.)	Δ	$::=$	$\emptyset$	*(empty env.)*
		$\mid$	$\Delta, k[\mathtt{A}] : T$	*(local session)*

Figure 3.10: Compositional Choreographies, typing environments.

the complete set of rules can be found in Appendix B.2.

We comment the typing rules. Rule $\lfloor^{\mathrm{T}}|_{\textsc{Start}}\rceil$ types a *(start)*; $a : G\langle \mathtt{A}\mid\tilde{\mathtt{B}}\mid\tilde{\mathtt{B}}\rangle$ checks that the choreography implements all roles in protocol G and that the processes $\tilde{\mathsf{q}}$ are fresh ($\tilde{\mathsf{q}} \notin \Gamma$); the continuation C is checked by adding Γ' and Δ' which contain, respectively, the process ownerships for k and the local types for the behaviour of each process ($(\Gamma', \Delta') = \mathsf{init}(\mathsf{p}[\mathtt{A}], \widetilde{\mathsf{q}[\mathtt{B}]}, k, G)$). Formally, the function init is defined as follows:

$$\mathsf{init}(\widetilde{\mathsf{p}[\mathtt{A}]}, k, G) = \big(\ \{\mathsf{q} : k[\mathtt{B}] \mid \mathsf{q}[\mathtt{B}] \in \widetilde{\mathsf{p}[\mathtt{A}]}\}, \{k[\mathtt{B}] : [\![G]\!]_{\mathtt{B}} \mid \mathsf{q}[\mathtt{B}] \in \widetilde{\mathsf{p}[\mathtt{A}]}\}\ \big)$$

Rule $\lfloor^{\mathrm{T}}|_{\textsc{Sel}}\rceil$ deals with selection, checking that the selected label l_j is specified in the local types and that the processes play the roles they are supposed to in session k according to Γ; the latter is ensured by the ownership judgement $\Gamma \vdash \mathsf{p}[\mathtt{A}]$ -> $\mathsf{q}[\mathtt{B}] : k$. Ownership judgements follow the same intuition as those presented for the Choreography Calculus (§ 2.4), but here we extend them to deal also with partial choreographies. The complete

$$\frac{\Gamma, a{:}G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{B}}\rangle, \Gamma' \vdash C \triangleright \Delta, \Delta' \quad (\Gamma', \Delta') = \mathsf{init}(\mathsf{p}[\mathtt{A}], \widetilde{\mathsf{q}[\mathtt{B}]}, k, G) \quad \tilde{\mathsf{q}} \notin \Gamma}{\Gamma, a{:}G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{B}}\rangle \vdash \mathsf{p}[\mathtt{A}]\ \mathbf{start}\ \widetilde{\mathsf{q}[\mathtt{B}]} : a(k); C \triangleright \Delta} \lfloor^{\mathrm{T}}|_{\textsc{Start}}\rceil$$

$$\frac{\Gamma \vdash \mathsf{p}[\mathtt{A}] \texttt{ -> } \mathsf{q}[\mathtt{B}] : k \quad \Gamma \vdash C \triangleright \Delta, k[\mathtt{A}]{:}T_j, k[\mathtt{B}]{:}T'_j \quad j \in I}{\Gamma \vdash \mathsf{p}[\mathtt{A}] \texttt{ -> } \mathsf{q}[\mathtt{B}] : k[l_j]; C \triangleright \Delta, k[\mathtt{A}]{:}\oplus\mathtt{B}\{l_i : T_i\}_{i\in I}, k[\mathtt{B}]{:}\,\&\mathtt{A}\{l_i : T'_i\}_{i\in I}} \lfloor^{\mathrm{T}}|_{\textsc{Sel}}\rceil$$

$$\frac{\Gamma \vdash x@\mathsf{p}{:}G\langle \mathtt{A}|\tilde{\mathtt{B}}|\emptyset\rangle \quad \Gamma, \mathsf{p}{:}k[\mathtt{A}] \vdash C \triangleright \Delta, k[\mathtt{A}]{:}[\![G]\!]_{\mathtt{A}}}{\Gamma \vdash \mathsf{p}[\mathtt{A}]\ \mathbf{req}\ \tilde{\mathtt{B}} : x(k); C \triangleright \Delta} \lfloor^{\mathrm{T}}|_{\textsc{Req}}\rceil$$

$$\frac{\Gamma, \Gamma_i \vdash C_i \triangleright \Delta_i \quad \mathsf{pco}(\Delta_1, \Delta_2)}{\Gamma, \Gamma_1 \circ \Gamma_2 \vdash C_1 \mid C_2 \triangleright \Delta_1, \Delta_2} \lfloor^{\mathrm{T}}|_{\textsc{Par}}\rceil \qquad \frac{\mathsf{cosha}(\Gamma) \quad \mathsf{end}(\Delta)}{\Gamma \vdash \mathbf{0} \triangleright \Delta} \lfloor^{\mathrm{T}}|_{\textsc{End}}\rceil$$

$$\frac{\Gamma, a{:}G\langle \mathtt{D}|\tilde{\mathtt{B}}|\emptyset\rangle, \Gamma' \vdash C \triangleright \Delta, \Delta' \quad (\Gamma', \Delta') = \mathsf{init}(\widetilde{\mathsf{q}[\mathtt{A}]}, k, G) \quad \tilde{\mathsf{q}} \notin \Gamma}{\Gamma, a{:}G\langle \mathtt{D}|\tilde{\mathtt{B}}|\tilde{\mathtt{A}}\rangle \vdash \mathbf{acc}\ \widetilde{\mathsf{q}[\mathtt{A}]} : a(k); C \triangleright \Delta} \lfloor^{\mathrm{T}}|_{\textsc{Acc}}\rceil$$

$$\frac{\Gamma \vdash e@\mathsf{p}{:}S \quad \Gamma \vdash \mathsf{p}[\mathtt{A}] \texttt{ -> } \mathtt{B} : k \quad \Gamma \vdash C \triangleright \Delta, k[\mathtt{A}]{:}T \quad \mathsf{q}{:}k[\mathtt{B}] \notin \Gamma}{\Gamma \vdash \mathsf{p}[\mathtt{A}].e \texttt{ -> } \mathtt{B} : k; C \triangleright \Delta, k[\mathtt{A}]{:}!\mathtt{B}\langle S\rangle; T} \lfloor^{\mathrm{T}}|_{\textsc{Com-s}}\rceil$$

$$\frac{\Gamma \vdash \mathtt{A} \texttt{ -> } \mathsf{q}[\mathtt{B}] : k \quad \Gamma \vdash C_i \triangleright \Delta, k[\mathtt{A}]{:}T_i \quad J \subseteq I \quad \mathsf{p}{:}k[\mathtt{A}] \notin \Gamma}{\Gamma \vdash \mathtt{A} \texttt{ -> } \mathsf{q}[\mathtt{B}] : k\&\{l_i{:}C_i\}_{i\in I} \triangleright \Delta, k[\mathtt{B}]{:}\,\&\mathtt{A}\{l_j : T_j\}_{j\in J}} \lfloor^{\mathrm{T}}|_{\textsc{Branch}}\rceil$$

Figure 3.11: Compositional Choreographies, selected typing rules.

rules for ownership typing are reported in Figure 3.12 (the rules are self-explanatory). In rule $\lfloor^{\mathrm{T}}|_{\textsc{Req}}\rceil$, we check that the choreography requesting the services is not responsible for implementing them, to avoid deadlocks due to the lack of services in parallel required by rule $\lfloor^{\mathrm{C}}|_{\textsc{p-Start}}\rceil$, and that the requesting process behaves as expected by its role in the protocol. We abuse the notation for unrestricted environments by assuming that we can write $\Gamma \vdash x@\mathsf{p} : G\langle \mathtt{A}|\tilde{\mathtt{B}}|\emptyset\rangle$ whenever $\Gamma \vdash x@\mathsf{p} : G$. Conversely, $\lfloor^{\mathrm{T}}|_{\textsc{Acc}}\rceil$ types an *(acc)* term by ensuring that all the roles for which the choreography is responsible are implemented (the other checks are similar to $\lfloor^{\mathrm{T}}|_{\textsc{Start}}\rceil$). This *distribution* of the responsibilities for implementing the different roles in a protocol is handled by rule $\lfloor^{\mathrm{T}}|_{\textsc{Par}}\rceil$, using the role distribution operation $\Gamma_1 \circ \Gamma_2$. Formally, $\Gamma_1 \circ \Gamma_2$ is defined as the union of Γ_1 and Γ_2 except for the typing of shared channels with the same name, which are merged with the following rule:

$$a{:}G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{C}}\rangle \;=\; a{:}G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{D}}\rangle \;\circ\; a{:}G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{E}}\rangle \qquad (\tilde{\mathtt{C}} = \tilde{\mathtt{D}} \uplus \tilde{\mathtt{E}})$$

Also in rule $\lfloor^{\mathrm{T}}|_{\textsc{Par}}\rceil$, predicate pco (for partial coherence, from [55]) checks that the local types for a session are the projection of a same global type; formally, $\mathsf{pco}(\Delta)$ holds if for all sessions k in Δ there exists a global type G such that the following condition holds:

$$\forall \mathtt{A}.\ k[\mathtt{A}]{:}T \in \Delta \;\Rightarrow\; T = [\![G]\!]_{\mathtt{A}}$$

$$\frac{\Gamma \vdash \mathtt{p} : k[\mathtt{A}], \mathtt{q} : k[\mathtt{B}]}{\Gamma \vdash \mathtt{p}[\mathtt{A}] \text{ -> } \mathtt{q}[\mathtt{B}] : k} \lfloor^{\mathrm{O}}|_{\textsc{Com}}\rceil \qquad \frac{\Gamma \vdash \mathtt{p} : k[\mathtt{A}]}{\Gamma \vdash \mathtt{p}[\mathtt{A}] \text{ -> } \mathtt{B} : k} \lfloor^{\mathrm{O}}|_{\textsc{Send}}\rceil \qquad \frac{\Gamma \vdash \mathtt{q} : k[\mathtt{B}]}{\Gamma \vdash \mathtt{A} \text{ -> } \mathtt{q}[\mathtt{B}] : k} \lfloor^{\mathrm{O}}|_{\textsc{Recv}}\rceil$$

Figure 3.12: Compositional Choreographies, ownership typing.

Note that pco allows for some role projections to be missing in Δ: we are just interested in checking that all the available participants agree on a same originating global type. In rule $\lfloor^{\mathrm{T}}|_{\textsc{End}}\rceil$ we check that all responsibilities have been implemented and that the sessions in Δ have been executed. Specifically, predicate $\mathsf{cosha}(\Gamma)$ checks that for every $a : G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{C}}\rangle$ in Γ either (i) $\tilde{\mathtt{C}} = \tilde{\mathtt{B}}$, meaning that a was used only internally with *(start)* terms; or (ii) $\tilde{\mathtt{C}} = \emptyset$, meaning that a is used compositionally in collaboration with other choreographies and all roles that the current choreography is responsible for ($\tilde{\mathtt{C}}$) have been implemented correctly with *(acc)* terms. Rules $\lfloor^{\mathrm{T}}|_{\textsc{Com-s}}\rceil$ and $\lfloor^{\mathrm{T}}|_{\textsc{Branch}}\rceil$ type respectively a sending action and a branching. They are very similar to their complete versions since local types allow us to look at the behaviour of processes independently. They also check that the counterpart for the partial action is not in the continuation, by ensuring that there is no process q such that q plays the other role for session k in Γ, which could obviously lead to a deadlock because process p would not have another process to communicate with in parallel as required by rule $\lfloor^{\mathrm{C}}|_{\textsc{Sync}}\rceil$.

3.4.3 Typing Expressiveness

Our typing system exploits the global information given by complete terms and seamlessly falls back to typical session typing when dealing with partial actions. In particular, $\lfloor^{\mathrm{T}}|_{\textsc{Sel}}\rceil$ judges that a choice in a protocol is implemented correctly even if only one of the branches is actually followed. This is sound because we are typing a complete term, and therefore we know that the other branches are not used. This aspect is the same as that of *partial protocol implementation* described in § 2.5.6. However, such the global knowledge needed for partial protocol implementation is not available in a partial choreography. For example, in rule $\lfloor^{\mathrm{T}}|_{\textsc{Branch}}\rceil$ we cannot know which branch will be selected by the sender and we must therefore require that the receiver process supports at least all the branches specified by the corresponding local type, as in standard session typing for endpoints [54, 55].

3.4.4 Properties

We conclude this section by presenting the expected main properties of our type system.

Theorem 3.4.1 (Typing Soundness)**.** *Let* $\Gamma \vdash C \triangleright \Delta$. *Then,*

- (Subject Swap) $C \simeq_{\mathcal{C}} C'$ *implies* $\Gamma \vdash C' \triangleright \Delta$.
- $C \xrightarrow{\lambda} C'$ *implies that there exists* Δ' *such that*
  - (Subject Reduction) $\Gamma' \vdash C' \triangleright \Delta'$ *for some* Γ'*;*
  - (Session Fidelity) *if* λ *is a communication on session* k*, then* $\Delta \xrightarrow{\alpha} \Delta'$ *with* $\Gamma \vdash \lambda \triangleright \alpha$*; otherwise,* $\Delta = \Delta'$.

$$
\begin{aligned}
[\![\mathtt{p}[\mathtt{A}]\ \mathbf{start}\ \widetilde{\mathtt{q}[\mathtt{B}]} : a(k); C]\!]_{\mathsf{r}} &= \begin{cases} \mathtt{p}[\mathtt{A}]\ \mathbf{req}\ \tilde{\mathtt{B}} : a(k); [\![C]\!]_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{p} \\ \mathbf{acc}\ \mathsf{r}[\mathtt{C}] : a(k); [\![C]\!]_{\mathsf{r}} & \text{if } \mathsf{r}[\mathtt{C}] \in \widetilde{\mathtt{q}[\mathtt{B}]} \\ [\![C]\!]_{\mathsf{r}} & \text{otherwise} \end{cases} \\
[\![\mathtt{p}[\mathtt{A}].e \texttt{ -> } \mathtt{q}[\mathtt{B}].x : k; C]\!]_{\mathsf{r}} &= \begin{cases} \mathtt{p}[\mathtt{A}].e \texttt{ -> } \mathtt{B} : k; [\![C]\!]_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{p} \\ \mathtt{A} \texttt{ -> } \mathtt{q}[\mathtt{B}].x : k; [\![C]\!]_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{q} \\ [\![C]\!]_{\mathsf{r}} & \text{otherwise} \end{cases} \\
[\![\mathtt{p}[\mathtt{A}].e \texttt{ -> } \mathtt{B} : k; C]\!]_{\mathsf{r}} &= \begin{cases} \mathtt{p}[\mathtt{A}].e \texttt{ -> } \mathtt{B} : k; [\![C]\!]_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{p} \\ [\![C]\!]_{\mathsf{r}} & \text{otherwise} \end{cases} \\
[\![\mathtt{A} \texttt{ -> } \mathtt{q}[\mathtt{B}].x : k; C]\!]_{\mathsf{r}} &= \begin{cases} \mathtt{A} \texttt{ -> } \mathtt{q}[\mathtt{B}].x : k; [\![C]\!]_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{p} \\ [\![C]\!]_{\mathsf{r}} & \text{otherwise} \end{cases} \\
[\![\mathsf{if}\ e@\mathsf{p}\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2]\!]_{\mathsf{r}} &= \begin{cases} \mathsf{if}\ e@\mathsf{p}\ \mathsf{then}\ [\![C_1]\!]_{\mathsf{r}}\ \mathsf{else}\ [\![C_2]\!]_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{p} \\ [\![C_1]\!]_{\mathsf{r}} \sqcup [\![C_2]\!]_{\mathsf{r}} & \text{otherwise} \end{cases} \\
[\![\mathtt{A} \texttt{ -> } \mathtt{q}[\mathtt{B}] : k\&\{l_i : C_i\}_{i \in I}]\!]_{\mathsf{r}} &= \begin{cases} \mathtt{A} \texttt{ -> } \mathtt{q}[\mathtt{B}] : k\&\{l_i : [\![C_i]\!]_{\mathsf{r}}\}_{i \in I} & \text{if } \mathsf{r} = \mathsf{q} \\ \bigsqcup_{i \in I} [\![C_i]\!]_{\mathsf{r}} & \text{otherwise} \end{cases} \\
[\![C_1 \mid C_2]\!]_{\mathsf{r}} &= [\![C_1]\!]_{\mathsf{r}} \mid [\![C_2]\!]_{\mathsf{r}}
\end{aligned}
$$

Figure 3.13: Compositional Choreographies, process projection (selected rules).

Proof. See Appendix B.2.2. □

Theorem 3.4.1 establishes that well-typedness is closed under transitions and that sessions in well-typed choreographies follow their protocols. The relationship between actions in protocols and choreographies is given by the judgement $\Gamma \vdash \lambda \triangleright \alpha$, which is formally defined in Appendix B.2.2, Figure B.3.

3.5 Properties of Compositional Choreographies

This section states the main properties of our framework wrt the execution of actual systems composed by endpoints.

3.5.1 Endpoint Projection

Endpoint Projection (EPP) generates correct endpoint code from a choreography. By endpoint code we mean choreographies that do not contain complete actions.

Process Projection. To define the complete EPP, we first define how the behaviour of a single process in a choreography can be projected. We denote this *process projection* of a process p in a choreography C with $[\![C]\!]_{\mathsf{p}}$. We discuss a selection of the rules defining process projection, reported in Figure 3.13; a complete definition is given in Appendix B.3.1. Process projection follows the structure of the originating choreography. In a *(start)*, we

project the active process p to a request and the service processes $\tilde{\mathsf{q}}$ to (always-available) accepts. In a *(com)*, the sender is projected to a partial sending action and the receiver to a partial receiving action. The projections of *(sel)* and *(del)*, omitted, follow the same principle. We also report the rule for projecting *(com-s)* and *(com-r)* to exemplify how we treat partial choreographies: these are simply projected as they are for their respective process, following the structure of the choreography. The projections of conditionals and partial branchings are the only special cases. In a conditional, we project it as it is for the process evaluating the condition, but for all other we merge their behaviours with the *merging* partial operator $\sqcup$ (as in § 2.5.2). $C \sqcup C'$ is defined only for partial choreographies that define the behaviour of a single process and returns a choreography isomorphic to C and C' up to branching, where all branches with distinct labels are also included. We use $\sqcup$ also in the projection of *(branch)* terms, where we require the behaviour of all processes not receiving the selection to be merged. As an example, the process projection for process u in the choreography C_B from our example in § 3.2 is (we omit the message type annotations for communications):

$$[\![C_B]\!]_{\mathsf{u}} = \begin{array}{l} \mathsf{u}[\mathtt{U}]\ \mathbf{req}\ \mathtt{PD} : a(k);\ \mathsf{u}[\mathtt{U}].prod \to \mathtt{PD} : k; \\ \mathtt{PD} \to \mathsf{u}[\mathtt{U}] : k\& \left\{ \begin{array}{ll} del : & \mathtt{PD} \to \mathsf{u}[\mathtt{U}] : k\langle k''[\mathtt{B}]\rangle;\ \mathsf{u}[\mathtt{B}].addr \to \mathtt{T} : k''; \\ & \mathtt{T} \to \mathsf{u}[\mathtt{U}].ddate : k'', \\ quit : & \mathbf{0} \end{array} \right\} \end{array}$$

Endpoint Projection. Using process projection, we can now define the EPP of a whole system. Since different service processes may be started through *(start)* terms on the same shared channel and play the same role, we use $\sqcup$ for merging their behaviours into a single service. We identify these processes with the service grouping operator $\lfloor C \rfloor^a_{\mathtt{A}}$ from § 2.5.4 (extended to partial choreographies), which computes the set of all service process names in a start or a request in C on shared channel a playing role $\mathtt{A}$. Formally, EPP is the endofunction $[\![C]\!]$ defined in the following.

Definition 3.5.1 (Endpoint Projection)**.** Let $C \equiv (\boldsymbol{\nu} \tilde{a}, \tilde{k}, \tilde{\mathsf{p}})\, C_f$, where C_f does not contain *(res)* terms. Then, the EPP of C is:

$$[\![C]\!] = (\boldsymbol{\nu}\tilde{a}) \left((\boldsymbol{\nu}\tilde{k}, \tilde{\mathsf{p}}) \Big(\underbrace{\prod_{\mathsf{p} \in \mathsf{fn}(C_f)} [\![C_f]\!]_{\mathsf{p}}}_{\text{(i)}} \Big) \mid \underbrace{\prod_{a,\mathtt{A}} \Big(\bigsqcup_{\mathsf{p} \in \lfloor C_f \rfloor^a_{\mathtt{A}}} [\![C_f]\!]_{\mathsf{p}} \Big)}_{\text{(ii)}} \right)$$

The EPP of a choreography C is the parallel composition of (i) the projections of all active processes and (ii) the merged projections of all service processes started under same shared channel and role. EPP respects the following Lemma, which shows that our model can adequately capture not only typical complete choreographies, but also scale down to describing the behaviour of a single endpoint.

Lemma 3.5.1 (Endpoint Choreographies)**.** *Let C be restriction-free, contain only partial terms, and be well-typed. If one of the following two conditions apply, then $C = [\![C]\!]$.*

1. *$C = \mathbf{acc}\ \mathsf{q}[\mathtt{B}] : a(k); C'$ and q is the only free process name in C';*
2. *otherwise, C has only one free process name.*

Proof. From the hypothesis and the definition of EPP, we see that $[\![C]\!] = [\![C]\!]_{\mathsf{q}}$ for some process q; then, the thesis follows by induction on the process projection $[\![C]\!]_{\mathsf{q}}$. □

We refer to choreographies that respect one of the two conditions above as *endpoint choreographies*. They implement either the behaviour of a single always-available service process (1), or that of a single free process (2). The EPP for these choreographies is the identity since they already model the behaviour of only one endpoint.

Service Distribution and EPP. Observe that the projection of services may lead to undesirable behaviour if service roles for shared channels are not distributed correctly. For example, if we put the choreography C_B from § 3.2 in parallel with a choreography with a conflicting service on shared channel b for role $\mathtt{R}$ (which is internally implemented in C_B) we obtain a race condition, *even if protocols are correctly implemented*. Consider the following choreography:

$$C_R = \ \mathbf{acc}\ \mathsf{h}[\mathtt{R}] : b(k');\ \mathtt{PD} \to \mathsf{h}[\mathtt{R}].x : k';\ \mathsf{h}[\mathtt{R}].c \to \mathtt{PD} : k'$$

If we put the projection of C_B in parallel with that of C_R, we get a race condition between the service processes r and h for role $\mathtt{R}$ on shared channel b. Hence, the projection of process pd may synchronise with the service offered by C_R for creating session k', instead of that by the projection of service process r in C_B. Consequently, C_B may not follow its intended behaviour. The distribution of service roles performed by our type system avoids this kind of situations. Observe that normal session typing cannot help us in detecting these problems, because the service process h correctly implements the same communication behaviour for session k' as service process r.

3.5.2 Main Properties

We can now present our main properties for compositional choreographies.

3.5.2.1 EPP Type Preservation

We build our results on the foundation that the EPP of a choreography is still typable. As in previous work [61, 31] and already discussed in § 2.5.5, we need to consider that in the projection of complete choreographies, due to merging, some projected processes may still offer branches that the original complete choreography has discarded with a conditional. Therefore, we state our type preservation result below under the *minimal typing* of choreographies $\vdash_{\mathsf{min}}$, in which the branches in rules $\lfloor^{\mathrm{T}}|_{\textsc{Sel}}\rceil$ and $\lfloor^{\mathrm{T}}|_{\textsc{Branch}}\rceil$ are typed using the respective minimal branch types (see Appendix B.3.2 for the full definitions). Below, $[\![\Gamma]\!]$ yields Γ where the typings of recursive procedures have been split into the typings of each procedure at each endpoint process ($[\![\Gamma]\!]$ is formally defined in B.3.2).

Theorem 3.5.2 (EPP Type Preservation)**.** *Let* $\Gamma \vdash_{\mathsf{min}} C \triangleright \Delta$. *Then,* $[\![\Gamma]\!] \vdash_{\mathsf{min}} [\![C]\!] \triangleright \Delta$.

Proof. See Appendix B.3.2. □

3.5.2.2 EPP Theorem

By Theorem 3.5.2, it follows that Theorem 3.4.1 applies also to the EPP of a choreography. We use this result to prove that EPP correctly implements the behaviour of the originating choreography, by establishing a formal relation between their respective semantics.

Theorem 3.5.3 (EPP Theorem)**.** *Let $C \equiv (\boldsymbol{\nu}\tilde{a}, \tilde{k}, \tilde{\mathsf{p}})\ C_f$, where C_f is restriction-free, be well-typed. Then,*

1. *(Completeness) $C \xrightarrow{\lambda} C'$ implies $[\![C]\!] \xrightarrow{\lambda}\succ [\![C']\!]$.*
2. *(Soundness) $[\![C]\!] \xrightarrow{\lambda} C'$ implies $C \xrightarrow{\lambda} C''$ and $[\![C'']\!] \prec C'$.*

Proof. See Appendix B.3.3. □

Above, the *pruning relation* $C \prec C'$ is a strong typed bisimilarity [31] such that C has some unused branches and always-available accepts. $\succ$ is a shortcut for $\prec$ interpreted in the opposite direction.

3.5.2.3 Deadlock-freedom

We introduce our results on deadlock-freedom and progress mentioned in the Introduction. First, we define deadlock-freedom:

Definition 3.5.4 (Deadlock-freedom)**.** We say that choreography C is deadlock-free if either (i) $C \equiv \mathbf{0}$ or (ii) there exist C' and λ such that $C \xrightarrow{\lambda} C'$ and C' is deadlock-free.

In our semantics (Figure 3.3) complete terms can always be executed; therefore, choreographies that do not contain partial terms, or *complete choreographies*, are deadlock-free. This recalls the result of deadlock-freedom for choreographies presented in § 2.3.2; below, we make the same assumption of well-sortedness, i.e., the arguments in procedure calls match the parameters of their respective procedure definitions.

Theorem 3.5.5 (Deadlock-freedom for Complete Choreographies)**.** *Let C be a well-sorted complete choreography and contain no free variable names. Then, C is deadlock-free.*

Proof. Since we are dealing with the complete fragment of compositional choreographies, the proof is the same as that for Theorem 2.3.2 in Chapter 2. □

By Theorems 3.5.3 and 3.5.5 we can obtain, as a corollary, that the EPP of well-typed complete choreographies never deadlock.

Corollary 3.5.1.1 (Deadlock-freedom for EPP)**.** *Let C be a complete choreography, contain no free variable names, and be well-typed. Then, $[\![C]\!]$ is deadlock-free.*

Proof. Since we are dealing with the complete fragment of compositional choreographies, the proof is the same as that for Corollary 2.5.1.1 in Chapter 2. □

3.5.2.4 Progress

Our model can also be used to talk of deadlock-freedom *compositionally*. In a compositional setting, a choreography may be unable to proceed in the global execution of a protocol because of partial actions that need to be executed in parallel composition with other choreographies. We say that a choreography can *progress* if it can be composed with another choreography such that (i) all free names can be restricted and the resulting system is still well-typed, ensuring that protocols are implemented correctly; and (ii) the composition is deadlock-free. Differently from deadlock-freedom for complete choreographies, progress for partial choreographies does not follow directly from the semantics. For example, the following choreography does not have the progress property:

$$\mathtt{A} \to \mathtt{q}[\mathtt{B}].x : k; \quad \mathtt{p}[\mathtt{A}].e \to \mathtt{B} : k$$

Above, $\mathtt{q}$ is waiting for a message on session k from $\mathtt{A}$, but that role is implemented by process $\mathtt{p}$ in the continuation. Thus, the two partial actions will never synchronise. As shown in § 3.4, our type system takes care of checking that roles in sessions or services are distributed correctly, avoiding cases such as this one and ensuring progress. In general, if a well-typed choreography does not contain inner *(par)* terms we know that it can progress, since role distribution ensures that there exists a compatible environment.

Theorem 3.5.6 (Progress for Partial Choreographies)**.** *Let C be a well-typed choreography containing no* (par) *terms. Then, there exists C' such that $(\boldsymbol{\nu}\tilde{r})\ (C \mid C')$, with $\tilde{r} = \mathsf{fn}(C \mid C')$, is well-typed and deadlock-free.*

Proof. See Appendix B.4. □

By Theorems 3.5.2 and 3.5.6, it follows as a corollary that also the EPP of a well-typed choreography can progress:

Corollary 3.5.1.2 (Progress for EPP)**.** *Let C contain no free variable names, be well-typed, and contain no* (par) *terms. Then, there exists C' such that $(\boldsymbol{\nu}\tilde{r})\ (\llbracket C \rrbracket \mid C')$ with $\tilde{r} = \mathsf{fn}(C \mid C')$, is well-typed and deadlock-free.*

Proof. Follows immediately by combining Theorems 3.5.2 and 3.5.6. □

3.5.2.5 Correctness of Choreography Composition

We end this section by presenting our key result: well-typed choreographies can be projected separately, and then their respective projections can be composed to obtain a correct endpoint implementation of a system.

First, we observe that the definition of EPP is compositional, i.e., separate choreographies can be projected before or after they are composed without influencing the final system. As an application example, this means that different vendors can develop choreographies

Lemma 3.5.2 (Compositional EPP)**.** *Let $C = C_1 \mid C_2$ be well-typed. Then, $\llbracket C \rrbracket \equiv \llbracket C_1 \rrbracket \mid \llbracket C_2 \rrbracket$.*

Proof. We recall the definition of EPP:

$$[\![C]\!] = (\boldsymbol{\nu}\tilde{a})\left((\boldsymbol{\nu}\tilde{k},\tilde{\mathsf{p}})\Big(\underbrace{\prod_{\mathsf{p}\in\mathsf{fn}(C_f)}[\![C_f]\!]_{\mathsf{p}}}_{\text{(i)}}\Big) \mid \underbrace{\prod_{a,\mathtt{A}}\Big(\bigsqcup_{\mathsf{p}\in\lfloor C_f\rfloor^a_{\mathtt{A}}}[\![C_f]\!]_{\mathsf{p}}\Big)}_{\text{(ii)}}\right)$$

We discuss the key points for ensuring that the parallel composition of the EPPs of $C_1 \mid C_2$ is equivalent to that of C, up to $\equiv$. For (i), i.e., the process projections of all free processes, we simply need to observe for any process p we have that $[\![C_1]\!]_{\mathsf{p}} \mid [\![C_2]\!]_{\mathsf{p}} = [\![C_1 \mid C_2]\!]_{\mathsf{p}}$, by definition of process projection. For (ii), i.e., the merged projections of all service processes, we observe that by the typing rule $\lfloor {}^{\mathrm{T}}|_{\text{PAR}}\rceil$ we have that C_1 and C_2 do not share any service processes that would be grouped together by the service grouping operator $\lfloor C_f \rfloor^a_{\mathtt{A}}$; hence, the service projections of the two respective choreographies are also simply put in parallel composition in the projection of C. □

By combining Lemma 3.5.2 with the Theorems shown so far we get the following corollary, which summarises the properties for well-typed compositions of choreographies obtained in this Chapter.

Corollary 3.5.2.1 (Compositional Choreographies)**.** *Let $C \mid C'$ be well-typed. Then,*

1. *(EPP Type Preservation) $[\![C]\!] \mid [\![C']\!]$ is well-typed.*
2. *(Completeness) $C \mid C' \xrightarrow{\lambda} C''$ implies $[\![C]\!] \mid [\![C']\!] \xrightarrow{\lambda} \succ [\![C'']\!]$.*
3. *(Soundness) $[\![C]\!] \mid [\![C']\!] \xrightarrow{\lambda} C''$ implies $C \mid C' \xrightarrow{\lambda} C'''$ and $[\![C''']\!] \prec C''$.*

Proof. Follows immediately by combining Theorems 3.5.2 and 3.5.3 with Lemma 3.5.2. □

Our corollary above formally addresses the issues mentioned in the Introduction. Choreographies (C and C' in the corollary) can be developed independently and then their respective projections can be composed.

3.6 Related Work

Previous works have tackled the problem of defining a formal model for choreographies and giving a correct EPP [61, 89, 31]. The main difference wrt our work is compositionality: previous models can only capture closed systems, and do not offer a methodology for composing choreographies. In [31], the authors ensure the typability of projections but do not handle neither asynchrony nor multiparty sessions; instead, they type choreographies with binary sessions. We have shown that choreographies can be made compositional by introducing partial terms to perform message passing with the environment, and that it is possible to ensure typability of EPP in a multiparty and asynchronous setting. This is the first work introducing a compositional multiparty session typing for choreographies, exploiting the projection of global types onto local types. Finally, [31] does not handle

shared channel passing and session delegation in a compositional setting, where sessions may be delegated to external or internal processes.

Compared to the Choreography Calculus proposed in Chapter 2, a major difficulty wrt composition is that the EPP of a choreography in the Choreography Calculus may be untypable with known type systems for session types. Typability of EPP is important to achieve composition, since a programmer may need to reuse a choreography *after* it has been projected, as a black box in a larger system. The Choreography Calculus in Chapter 2 is, to the best of our knowledge, the only previous work providing an asynchronous semantics for multiparty sessions in choreographies; however, asynchrony is modelled in two different ways in the choreography model and the endpoint model, raising complexity. As a consequence, the EPP Theorem in § 2.5 has a more complex formulation with weak transitions and confluence, whereas the EPP Theorem in this Chapter is formulated in a stronger form where EPP mimics its original choreography step by step. We observe, however, that this simplicity comes at the cost of a more restricting rule for asynchronous communications in the calculus of Compositional Choreographies, which prevents a process to delegate a channel that it is still using. We conjecture that this limitation may be removed by introducing explicit communication queues in compositional choreographies, which we leave as future work.

Multiparty session types have been previously used for typing endpoint programs [55, 21, 36]. In our setting, endpoint programs can be captured as special cases of partial choreographies. Our global types are taken from [21]. Differently from our framework, these works capture asynchronous communications with dedicated processes that model order-preserving message queues. An approach more similar to ours can be found in the notion of *delayed input* presented in [68]. [21] reports a type system for progress by building additional restrictions on top of standard multiparty session typing; our model yields a simpler analysis, since we can rely on the fact that complete terms in a choreography do not get stuck. Nevertheless, the model in [21] can capture sessions started by more than one active process. We leave an extension of our model in this direction as future work.

In [17] the authors use a concept similar to our partial choreographies for protocol specifications, to allow a single process to implement more than one role in a protocol. Differently from our approach, these are not fully-fledged system implementations but abstract behavioural types, which are then used to type check endpoint code. In our setting, the techniques in [17] can be seen as a more flexible way of handling the projection from global types to local types. An extension of our type system to allow for a process to play more than one role in a session as in [17, 36] is an interesting future work.

The relationship between choreographies and endpoints has been explored in, among others, [26, 89, 61, 55, 31]. Our work distinguishes itself by adopting the same calculus for describing choreographies and endpoints, simplifying the technical development.

3.7 Conclusions

We presented a new model where asynchronous communicating systems can be designed by composing choreographies that mix complete and partial actions. Choreographies can now be used as software libraries to design distributed software modularly. We developed a type system that checks choreographies against multiparty protocol specifications, ensuring

that the composition of compatible choreographies does not lead to bad behaviour. Relying on our typing discipline, our EPP generates correct endpoint code guaranteeing nontrivial properties such as deadlock-freedom and progress. The practical motivation and design of our model come from our experiences on working with a tool for Web Services, Jolie [60].

CHAPTER 4

Round-Trip Choreographic Programming

4.1 Introduction

In the previous Chapters we have explored formal models for Choreographic Programming that provide a notion of correct Endpoint Projection (EPP): systems can be globally designed by using choreographies and then a sound endpoint implementation can be obtained through EPP. However, in practice software developers using choreographies usually adopt Choreographic Programming together with the typical programming of each process [1, 79]. The idea is to use the choreographic view to check that a system follows the expected flow of interactions and to use the process view to program the internal actions of each process. This methodology, known as *round-trip development*, is supported by two operations: *endpoint projection* (EPP), which compiles a choreography to the code of the processes that realise it, and *choreography extraction*, which extracts from the code of some processes the choreography they realise. However, to the best of our knowledge, there is no programming model supporting choreography extraction; extraction is a harder problem than EPP since it is difficult to predict how a concurrent system will execute at runtime [18]. For this reason, finding solid theoretical foundations for round-trip development remains elusive. The aim of this Chapter is making the first step towards laying the foundations to tackle this issue. In particular, we ask:

> *Can we design a unified model that coherently offers both a choreographic view and a process view on a communicating system?*

The challenge in finding an answer lies in clarifying what are, exactly, the rules that underpin the reasoning behind choreographies and what is their relationship with the rules that underpin the reasoning behind processes.

A good starting point for answering our question is the recent line of work on a Curry-Howard correspondence between the π-calculus [69], a reference model for concurrent computation, and linear logic [27, 103], which elicits the logical reasoning behind processes. In particular, in their seminal paper, Caires and Pfenning present a logical reconstruction of session types [54] by showing that terms in the π-calculus can be used as proof terms for derivations in Intuitionistic Linear Logic (ILL) [27]. Such logical reconstruction establishes the following correspondences:

propositions	*as*	sessions
proofs	*as*	processes
cut elimination	*as*	communication

Caires and Pfenning provide thus a logical characterisation of the process view on a system as proofs in ILL. The last correspondence, in particular, proves that cut elimination describes the communications in a system. The key insight of this work is that choreographies and cut elimination play similar roles, since they both capture the message flow implemented by a system. We follow this intuition and formalise this relationship by presenting a logical reconstruction of choreographies in form of a new logic, called LCL. We show that proofs in LCL successfully capture a class of concurrent systems that supports round-trip development: it offers both a choreographic view and a process view convertible into one another, guaranteeing that the message flow described by the former and implemented by the latter is identical. Interestingly, the class of systems captured by LCL subsumes that captured by ILL, allowing us to apply round-trip global programming to all those systems programmable in the ILL fragment of [27].

4.1.1 Contributions

The main contributions of this Chapter are:

Linear Connection Logic. We introduce Linear Connection Logic (LCL), a logic for formally reasoning on the concurrent behaviour of multiple participants in the flavour of choreographies (§ 4.3). The key to our development is the introduction of "connections" between participants as first-class citizens in logical judgements (§ 4.2).

Round-trip Logical Reasoning. LCL is a conservative extension of Intuitionistic Linear Logic: on the one hand, it supports the typical reasoning found in linear logic, where resources are composed from the viewpoint of a single participant and then compatible participants are merged together; on the other hand, it also supports our new reasoning where resources are constructed collaboratively by multiple participants. We provide automatic transformations that allow us to switch from one methodology to the other (§ 4.4, Abstraction and Concretisation). We give an operational meaning to LCL proofs by defining how connections can be always eliminated (§ 4.4, β-reductions). We finally show that proofs in the two methodologies share a tight operational correspondence in terms of how they are normalised (§ 4.4.3, Correspondence Theorem).

Round-trip Choreographic Programming. We build a logical reconstruction of a fragment of Compositional Choreographies by using choreographies as proof terms for LCL (§ 4.6), as in a Curry-Howard correspondence. Our reconstruction yields a type theory for choreographies where propositions correspond to session types (§ 4.6, Typing). Based on our results on LCL, we define a model for round-trip choreographic programming and establish the following correspondences:

propositions	*as*	sessions
LCL proofs	*as*	compositional choreographies
proof concretisation	*as*	endpoint projection
proof abstraction	*as*	choreography extraction

Since it is always possible to transform proofs that correspond to processes to proofs that correspond to choreographies and vice versa, we are ultimately able to prove that it is always possible to switch between a choreographic view to a process view of a system: in our setting, choreographies and processes are in a direct correspondence.

4.2 Preview

In this section, we give an informal preview to the results in this Chapter. We will first revisit the Curry-Howard correspondence between π-calculus terms and ILL [27]. Building on ILL, we will informally introduce our logic LCL and show how it corresponds to an extension of the π-calculus with choreographies.

4.2.1 ILL and the π-calculus

Consider the following π-calculus example:

$$\underbrace{\overline{x}(tea);\ x(tr);\ \overline{tr}(p)}_{P_{\mathsf{client}}} \qquad \underbrace{x(tea);\ \overline{x}(tr);\ tr(p);\ \overline{b}(m)}_{P_{\mathsf{server}}} \qquad \underbrace{b(m)}_{P_{\mathsf{bank}}} \tag{4.1}$$

The three processes above, respectively P_{client}, P_{server}, and P_{bank}, if composed in parallel, will execute as follows: first, P_{client} sends to P_{server} a request for some tea on a channel x; then, P_{server} replies to P_{client} on the same channel x with a new channel tr (for transaction); P_{client} uses tr for sending to P_{server} the payment p for the tea; after receiving the payment, P_{server} finally deposits some money m in the bank by sending it over channel b to process P_{bank}.

The three processes above can be typed using the proof theory of ILL, by respectively associating their channels to formulae; each formula describes how a channel is used during execution, as in standard session types [54]. For example, channel x in process P_{client} has type $\mathbf{string} \otimes (\mathbf{string} \multimap \mathsf{end}) \multimap \mathsf{end}$. Intuitively, this means that x must be used as follows: first, we send a string; then, we receive a channel of type $\mathbf{string} \multimap \mathsf{end}$ and, finally, we stop using the channel (end). Concretely, in process P_{client}, the channel of type $\mathbf{string} \multimap \mathsf{end}$ received through x is channel tr. The type of tr means that the other process sending tr, i.e., P_{server}, will use it to receive a **string** and then terminate it; therefore, process P_{client} is now responsible for implementing the dual operation of that implemented by P_{server}, i.e., P_{client} has to execute an output (term $\overline{tr}(p)$). Similarly, channel b has type $\mathbf{int} \otimes \mathsf{end}$ in P_{server}, meaning that an integer will be sent over b.

In order to formally express the intuition above, we can write the following three judgements, where $A = \mathbf{string} \otimes (\mathbf{string} \multimap \mathsf{end}) \multimap \mathsf{end}$ and $B = \mathbf{int} \otimes \mathsf{end}$:

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

The right hand side of each judgement consists of a sequent of the form $\Delta \vdash x : A$, where Δ expresses what the process *needs* to use from the rest of the system in order to execute, while A is the behaviour that the process *provides* on channel x. For example, the judgement $P_{\mathsf{server}} \triangleright x : A \vdash b : B$ reads as "given a context that implements channel x with type A, process P_{server} implements channel b with type B".

Given the judgements above, we can compose the processes P_{client}, P_{server}, and P_{bank} as follows:

$$(\boldsymbol{\nu} x)\ (P_{\mathsf{client}} \mid (\boldsymbol{\nu} b)\ (P_{\mathsf{server}} \mid P_{\mathsf{bank}}))$$

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

Figure 4.1: A Cut Reduction in ILL.

Above, we have two compositions. The first is between P_{server} and P_{bank}, which communicate using channel b. The second is between such composition and P_{client}, using channel x. Logically, these compositions can be constructed by using the standard Cut rule of ILL as a typing rule, which we call TCut:

$$\dfrac{P \triangleright\ \Delta_1 \vdash x{:}A \quad Q \triangleright\ \Delta_2, x{:}A \vdash y{:}B}{(\boldsymbol{\nu} x)\,(P \mid Q) \triangleright\ \Delta_1, \Delta_2 \vdash y{:}B}\,\mathsf{TCut}$$

We read rule TCut as "If a π-calculus expression provides A, and another requires A to provide B, both can be executed in parallel to provide B".

Caires and Pfenning [27] show that proofs in ILL correspond to process terms in the π-calculus. In ILL, applications of rule Cut can always be reduced to smaller cuts until all cuts are eliminated, a procedure of proof normalisation known as cut elimination. As a corollary from the meta-theory of ILL, cut elimination provides a model of computation for reducing processes. The result of this interpretation is a tight operational correspondence between cut reductions and communications.

We illustrate a cut reduction, a step of cut elimination, in Figure 4.1. For readability, we omit the process terms and use the purely logical form of ILL judgements. The proof on the left-hand side applies a cut rule to two proofs, one providing $A \otimes B$, and the other providing D when provided with $A \otimes B$. On either side, $A \otimes B$ is called the *principal* formula created by the proofs above. Figure 4.1 shows how the same proof can be transformed into an equivalent proof, where the application of the Cut rule on formula $A \otimes B$ has been reduced to two applications of the Cut rule on the smaller formulae A and B.

Following Caires and Pfenning [27], the sample cut-reduction step in Figure 4.1 corresponds, in the π-calculus, to executing a communication step. The left hand side represents a system composed of two processes, one outputting on a channel of type $A \otimes B$, and another reading from the same channel. The result of executing the communication step yields a new system corresponding to the proof on the right-hand side. This is equivalent to the following π-calculus reduction:

$$(\boldsymbol{\nu} x)\,\big(\,\overline{x}(y); (P \mid Q) \mid x(y); R\,\big) \quad \rightarrow \quad (\boldsymbol{\nu} y)\,\big(\,P \mid (\boldsymbol{\nu} x)\,(Q \mid R)\,\big)$$

In the general case, Cut rules might have to be permuted to enable a cut reduction to apply. Cut elimination corresponds therefore to reducing a system composed of several processes, by transitively applying permutation and reduction steps. Cut-free proofs correspond then to a system that has successfully completed all its internal communications.

4.2.2 Towards a logic for choreographies

In our previous example, we have described a system using a standard *process view*, i.e., by separately defining the code for all processes and then by composing them. In general,

such an approach to programming distributed systems can be error-prone, in the sense that the programmer does not have a clear picture of how the system executes as a whole. By contrast, choreographies can be used for specifying the same system, by giving a syntactic global description of how messages are supposed to flow during execution. Below, we report a *choreographic view* of our example (4.1):

1. client –> server : $x(tea)$;
2. server –> client : $x(tr)$;
3. client –> server : $tr(p)$;
4. server –> bank : $b(m)$

This choreography specifies the sequence of communications that are supposed to happen when running the system it describes. For example, we read Line 1 as "process client sends tea to process server through channel x". In an implementation of the system described by the choreography above, we would have the client process performing an output of tea on channel x, while the server process would do the dual action, namely an input on x.

Now, our objective is to develop a relationship between the choreographic and the process views of the same system. Since communications correspond to cut reductions, it should be possible to use choreographies to describe cut reductions just as they can be used to describe communications. Therefore, we start our investigation by developing a model in which we can construct choreographies that describe the behaviour of the processes captured by Caires and Pfenning [27].

A key point for syntactically describing cut reductions is to be able to formally record, e.g., in a judgement, where we have applied a Cut rule in an ILL proof. Unfortunately, none of the judgements in Figure 4.1 gives any insight on the fact that processes were composed together with a Cut. If we consider the type systems for choreographies (as in [31] and our Chapters 2 and 3), we notice that they have different judgements than the typical ones for processes. The main difference is that a choreography typing judgement contains information about multiple processes and the interactions that they perform with each other. Following this idea, there are only two additions necessary to the work of Caires and Pfenning in order to capture interactions. First, we extend judgements in ILL to describe multiple processes. Our judgements are thus *hypersequents*, i.e., collections of multiple ILL sequents [16]. Second, we need a way to talk about the *connections* between sequents in a hypersequent, since two processes need to share a common connection for interacting. As an example, the following is a valid judgement in LCL:

$$\triangleright\, \Delta_1 \vdash x{:}\bullet A \mid \Delta_2, x{:}\bullet A \vdash B$$

Above, we have composed two ILL sequents with the operator $\mid$, which captures the parallel composition of processes (in this case, we have only two processes). The two sequents are connected by the resource A, denoted by the shared name x and the modality $\bullet$. Intuitively, $\bullet$ is a modality for hypotheses and conclusions of ILL sequents that tells whether a conclusion of a sequent, called producer, is *connected* to the hypothesis of another sequent, called consumer. Hypersequents and resource connections allow us to reason about interactions. Specifically, when reasoning about interactions, we are never allowed to reason only about one end of the connection in isolation, but we will need to reason about both ends in synchrony: when we build an interaction from a connection, we need to "update" the typing of both producer and consumer over their shared connection.

Thanks to our judgements, we can now express the cut elimination process as a proof. As an example, the following hypersequent:

$$\triangleright\, w_1 : C_1, w_2 : C_2 \vdash x : \bullet(A \otimes B) \mid x : \bullet(A \otimes B) \vdash z : D$$

models a system where a process can produce resource $A \otimes B$ by consuming resources C_1, C_2, and another process can produce resource D by consuming resource $A \otimes B$ produced by the first process. Importantly, the resource produced by the first sequent is no longer available since it is being used by the second sequent. Even more, we can also mimic reductions in cut elimination. Consider:

$$\triangleright\, w_1 : C_1 \vdash y : \bullet A \mid w_2 : C_2 \vdash x : \bullet B \mid y : \bullet A, x : \bullet B \vdash z : D$$

The new hypersequent describes a system that still needs to consume C_1 and C_2 in order to produce D. However, now we have three processes: one producing A from C_1, one producing B from C_2 and finally one using A and B separately for producing D. Additionally, both the first and the second sequent are connected to the third one. This corresponds to the situation seen in Figure 4.1.

Choreographically, we are now able to express communications. In fact, if the latter judgement above is the conclusion of a proof corresponding to some system P (the right-hand side of Figure 4.1), the former judgement says that we are in a system where there may be an interaction of type $A \otimes B$ between two processes and P is what we obtain after such interaction has been executed.

4.3 Linear Connection Logic

4.3.1 Hypersequents and Connections

We are now ready to present LCL. The syntax of hypersequents is reported in Figure 4.2. In the syntax, formulae are the same as in ILL. Hereby, we only present the multiplicative

(Propositions)	A, B	$::=$	$\mathbf{1} \mid A \otimes B \mid A \multimap B$
(Hypotheses and Conclusions)	T	$::=$	$x : A \mid x : \bullet A$
(Contexts)	Γ, Δ, Θ	$::=$	$\cdot \mid \Delta, T$
(Hypersequents)	Ψ	$::=$	$\Delta \vdash T \mid \Psi \mid \Psi$

Figure 4.2: LCL, syntax of hypersequents.

connectives $\otimes$ and $\multimap$; we will discuss the additive connectives of ILL later on in § 4.5. The proposition (or formula) $\mathbf{1}$ is the multiplicative unit, $A \otimes B$ stands for "resources A and B are available", and $A \multimap B$ says "if resource A is provided, then B is available". Contexts Δ and hypersequents Ψ are equivalent modulo associativity and commutativity. A hypersequent Ψ can be either a single sequent $\Delta \vdash T$, or the parallel composition of many sequents. Given a sequent $\Delta \vdash T$, we call Δ the hypotheses and T the conclusion of the sequent. The hypotheses and conclusion of a sequent are identified by variables,

denoted x, y, ..., and can be marked with the modality $\bullet$, representing a connection with another sequent.

We make the standard assumption that a variable x can appear at most once in any hypersequent, *unless* it is marked with $\bullet$. In LCL, bulleted variables appear exactly twice in a hypersequent, once as a hypothesis and once as a conclusion of two respective sequents which we say are then "connected". A provable hypersequent always has exactly one sequent with a non-bulleted conclusion, which we call the conclusion of the hypersequent. Similarly, we call non-bulleted hypotheses the hypotheses of the hypersequent. Intuitively, a provable hypersequent is a tree of sequents, whose root is the only sequent with a non-bulleted conclusion, and whose sequents have exactly one child for each of their bulleted hypotheses.

4.3.2 Proof Theory

We write $\triangleright\ \Psi$ for a judgement in LCL. We define the proof theory of judgements in two steps. First, we define the resource fragment of LCL in terms of left/right rules for resources A and their composition, and second we define the connection fragment and its rules for connections $\bullet A$.

4.3.2.1 Resource Fragment

The resource fragment of LCL is an embedding of the sequent formulation of ILL: the rules carry over directly. We introduce the fragment in several stages. Hereby we discuss only the multiplicative fragment of LCL; we will extend it to additive connectives in § 4.5 and discuss how it can be extended to exponentials in § 4.8.

Unit. The rules for unit are like the ones of standard linear logic. The right rule for $\mathbf{1}$ is the only axiom of LCL.

$$\frac{}{\triangleright \cdot \vdash x{:}\mathbf{1}}\ \mathsf{1R} \qquad\qquad \frac{\triangleright \Psi \mid \Delta \vdash T}{\triangleright \Psi \mid \Delta, x{:}\mathbf{1} \vdash T}\ \mathsf{1L}$$

Tensor. The left and right rules for tensor are almost standard, except that the non-principal components of the conclusion in $\otimes\mathsf{R}$ need to be split into Ψ_1 and Ψ_2 and distributed to the premises.

$$\frac{\triangleright \Psi_1 \mid \Delta_1 \vdash y{:}A \quad \triangleright \Psi_2 \mid \Delta_2 \vdash x{:}B}{\triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1, \Delta_2 \vdash x{:}A \otimes B}\ \otimes\mathsf{R} \qquad \frac{\triangleright \Psi \mid \Delta, y{:}A, x{:}B \vdash T}{\triangleright \Psi \mid \Delta, x{:}A \otimes B \vdash T}\ \otimes\mathsf{L}$$

Linear Implication. The rules for linear implication are also standard, and since it is a multiplicative connective, the non-principal components are split just as in the tensor case.

$$\frac{\triangleright \Psi \mid \Delta, y{:}A \vdash x{:}B}{\triangleright \Psi \mid \Delta \vdash x{:}A \multimap B}\ \multimap\mathsf{R} \qquad \frac{\triangleright \Psi_1 \mid \Delta_1 \vdash y{:}A \quad \triangleright \Psi_2 \mid \Delta_2, x{:}B \vdash T}{\triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1, \Delta_2, x{:}A \multimap B \vdash T}\ \multimap\mathsf{L}$$

Connection and Scoping. Any standard presentation of inference rules for linear logic would at this point introduce a Cut rule of the form

$$\frac{\triangleright\, \Psi_1 \mid \Delta_1 \vdash x{:}A \quad \triangleright\, \Psi_2 \mid \Delta_2, x{:}A \vdash T}{\triangleright\, \Psi_1 \mid \Psi_2 \mid \Delta, \Delta_2 \vdash T}\ \mathsf{Cut}$$

which we will not do here. Instead, we pull the Cut rule apart, and obtain two rules depending critically on hypersequents as an interim place to store information. The first rule, named Conn, forms a connection by combining the two premises of the Cut rule:

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

Rule Conn is only applicable for derivations of hypersequents ending in producing and consuming a resource A.

The second rule, called Scope, delimits the scope of a connection. After applying Scope, the scoped connection disappears:

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

It is easy to see that by composing the rules Conn and Scope, we obtain that the normal Cut rule is easily derivable in LCL.

4.3.2.2 Connection Fragment

Connections are first-class citizens in LCL, and they are therefore object of logical reasoning. Formally, the connection fragment of LCL is formed by the Conn and Scope rules and the rules reported below. We give three rules for reasoning on connections, one for each linear logic connective. The rules are called 1C, $\otimes$C, and $\multimap$ C. By such rules connections can be composed or decomposed, depending on whether the rules are read top-down or bottom-up.

Unit. A connection for the exchange of the unit resource $\mathbf{1}$ can always be introduced, since $\mathbf{1}$ can always be produced:

$$\frac{\triangleright\, \Psi \mid \Delta \vdash T}{\triangleright\, \Psi \mid \cdot \vdash x{:}\bullet\mathbf{1} \mid \Delta, x{:}\bullet\mathbf{1} \vdash T}\ \mathsf{1C}$$

Tensor. The connection rule for $\otimes$ combines two connections that share the same consumer. Technically, when two producer sequents $\Delta_1 \vdash y : \bullet A$ and $\Delta_2 \vdash x : \bullet B$ are connected to a consumer sequent $\Delta_3, y : \bullet A, x : \bullet B \vdash T$, we can merge the two connections into a single one, obtaining the sequents $\Delta_1, \Delta_2 \vdash x{:}\bullet(A \otimes B)$ and $\Delta_3, x{:}\bullet(A \otimes B) \vdash T$:

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

Linear Implication. For $\multimap$, its connection rule manipulates connections that have a causal dependency, i.e., we use this rule when a sequent needs to consume a connected resource in order to produce another connected resource. Technically, if $\Delta_1 \vdash y : \bullet A$ is connected to $\Delta_2, y : \bullet A \vdash x : \bullet B$, which is connected to $\Delta_3, x : \bullet B \vdash T$, then $\Delta_2 \vdash x : \bullet(A \multimap B)$ is connected to $\Delta_1, \Delta_3, x : \bullet(A \multimap B) \vdash T$.

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

From now on, we refer to the three rules above as C-rules. Observe that C-rules share a deep relationship with the cut reductions found in linear logic. For instance, the connections in the premise of rule $\otimes$C are a representation of the cuts in the right-hand side proof in Figure 4.1, and the conclusion is a representation of the cuts in the left-hand side proof. In general, all our C-rules are representations of cut reductions found in ILL. In the next section, we exploit this aspect to show that the resource fragment and the connection fragment share a deep relationship.

4.4 Proof Transformations

In LCL, a judgement containing connections can be derived by either (i) building resources separately, using the resource fragment, and then using the Conn rule, or (ii) using the connection fragment to compose more complex connections from simpler ones. For example, consider the following derivations, $\mathcal{D}$ and $\mathcal{E}$ respectively:

$$\dfrac{\dfrac{\dfrac{}{\triangleright \cdot \vdash x{:}\mathbf{1}}\ \mathsf{1R} \quad \dfrac{\dfrac{}{\triangleright \cdot \vdash y{:}\mathbf{1}}\ \mathsf{1R}}{\triangleright\, x{:}\mathbf{1} \vdash y{:}\mathbf{1}}\ \mathsf{1L}}{\triangleright \cdot \vdash x{:}\bullet\mathbf{1} \mid x{:}\bullet\mathbf{1} \vdash y{:}\mathbf{1}}\ \mathsf{Conn}^x}{\triangleright \cdot \vdash y{:}\mathbf{1}}\ \mathsf{Scope}^x \qquad \dfrac{\dfrac{\dfrac{}{\triangleright \cdot \vdash y{:}\mathbf{1}}\ \mathsf{1R}}{\triangleright \cdot \vdash x{:}\bullet\mathbf{1} \mid x{:}\bullet\mathbf{1} \vdash y{:}\mathbf{1}}\ \mathsf{1C}^x}{\triangleright \cdot \vdash y{:}\mathbf{1}}\ \mathsf{Scope}^x$$

Above, we have annotated some rule instances with the variable of the connection they apply to. We write such annotations only when they are relevant, and omit them otherwise. The two derivations $\mathcal{D}$ and $\mathcal{E}$ above reach the same conclusion by adopting, respectively, the methodologies (i) and (ii). This hints at that the connection fragment is redundant. However, it turns out that it is also, somehow, more efficient. As an example, consider a proof ending with the judgement:

$$\frac{\vdots}{\triangleright\, \Psi \mid \Delta_1 \vdash y{:}\bullet A \mid \Delta_2 \vdash x{:}\bullet B \mid \Delta_3, y{:}\bullet A, x{:}\bullet B \vdash T}$$

Now, assume that we wished to extend the proof to derive

$$\triangleright\, \Psi \mid \Delta_1, \Delta_2 \vdash x{:}\bullet A \otimes B \mid \Delta_3, x{:}\bullet A \otimes B \vdash T$$

When adopting methodology (ii), we can apply rule $\otimes$C and immediately complete the proof. However, if we follow (i), we need to "open up" the whole proof, split the connections on x and y to access their resources, use the latter to rebuild two proofs respectively providing and requiring $A \otimes B$, and finally complete the proof by applying Conn.

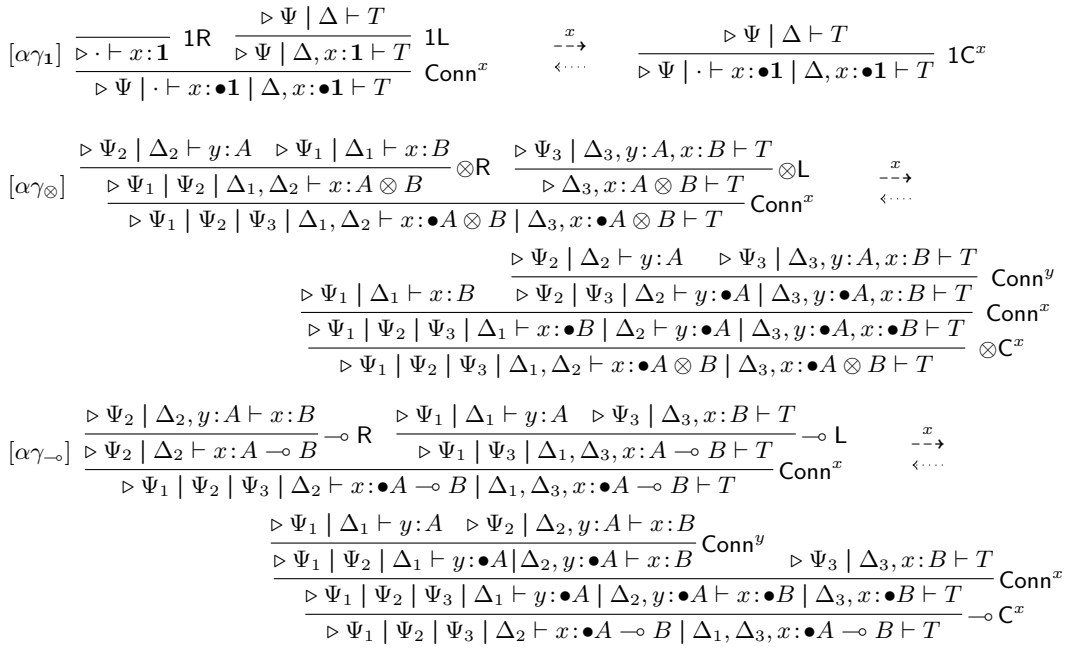


Figure 4.3: LCL, Abstraction and Concretisation transformations.

In this section, we show that the two methods are equivalent, meaning that any judgement proven using one method can also be proven with the other and vice versa. This means that applications of rule Conn and C-rules that apply to the same connection can be safely transformed into one another. We call the transformation from Conn to C-rules α-*abstraction*, written $\mathcal{D} \overset{x}{\dashrightarrow} \mathcal{E}$, since it abstracts away the formation of a connection x from two resources. Abstraction is an invertible operation: we call the opposite transformation γ-*concretisation*, written $\mathcal{E} \overset{x}{\cdots\!\rightarrow} \mathcal{D}$.

We further observe that both $\mathcal{D}$ and $\mathcal{E}$ above can be *reduced* to the simple application of the axiom 1R, yielding $\triangleright \cdot \vdash y : \mathbf{1}$. The process of erasing a connection x from a proof is called β-reduction, denoted by $\mathcal{D} \xrightarrow{x} \mathcal{D}'$. We will see that there are two kinds of β-reductions, depending on whether they apply to connections that have been constructed using rule Conn or C-rules. In this Chapter, we also show that these two kinds of reductions produce the same result. We call this result the "Correspondence Theorem", which is our main result on LCL. In § 4.6, we will use it to establish a tight correspondence between processes and choreographies.

4.4.1 Abstraction and Concretisation

The transformations α-abstraction and γ-concretisation are the inverse of one another. They are formally defined by the rules reported in Figure 4.3 closed under contexts, i.e., they can be applied to any subderivation in a proof. We explain the rules reading them from left to right, which corresponds to abstraction; for concretisation, we read them from right to left. Rule $[\alpha\gamma_{\mathbf{1}}]$ abstracts an application of the rule Conn with principal formula

$\mathbf{1}$ to an application of rule 1C. Rule $[\alpha\gamma_{\otimes}]$ transforms a connection of $A \otimes B$, obtained in the resource fragment, into the formation of a connection $A \otimes B$ within the connection fragment. Rule $[\alpha\gamma_{\multimap}]$ is similar.

In the previous example, we can easily see that $\mathcal{D} \overset{x}{\dashrightarrow} \mathcal{E}$ and $\mathcal{E} \overset{x}{\dashrightarrow} \mathcal{D}$. However, it is not always possible to apply abstraction or concretisation directly. For example, consider the following proof:

$$\dfrac{\dfrac{}{\triangleright \cdot \vdash x : \mathbf{1}}\ 1\mathsf{R} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\triangleright \cdot \vdash y : \mathbf{1}}\ 1\mathsf{R}}{\triangleright x : \mathbf{1} \vdash y : \mathbf{1}}\ 1\mathsf{L}}{\triangleright x : \mathbf{1}, z : \mathbf{1} \vdash y : \mathbf{1}}\ 1\mathsf{L}}{\triangleright x : \mathbf{1} \vdash y : \mathbf{1} \multimap \mathbf{1}}\ \multimap \mathsf{R}}{\triangleright \cdot \vdash x : \bullet\mathbf{1} \mid x : \bullet\mathbf{1} \vdash y : \mathbf{1} \multimap \mathbf{1}}\ \mathsf{Conn}$$

The application of rule Conn above cannot be abstracted, because it does not match the left-hand side of rule $(\alpha\gamma_{\mathbf{1}})$. However, we can find an equivalent proof for which $(\alpha\gamma_{\mathbf{1}})$ is applicable:

$$\dfrac{\dfrac{\dfrac{}{\triangleright \cdot \vdash x : \mathbf{1}}\ 1\mathsf{R} \quad \dfrac{\dfrac{}{\triangleright \cdot \vdash y : \mathbf{1}}\ 1\mathsf{R}}{\triangleright x : \mathbf{1} \vdash y : \mathbf{1}}\ 1\mathsf{L}}{\triangleright \cdot \vdash x : \bullet\mathbf{1} \mid x : \bullet\mathbf{1} \vdash y : \mathbf{1}}\ \mathsf{Conn}}{\triangleright \cdot \vdash x : \bullet\mathbf{1} \mid x : \bullet\mathbf{1}, z : \mathbf{1} \vdash y : \mathbf{1}}\ 1\mathsf{L}}{\triangleright \cdot \vdash x : \bullet\mathbf{1} \mid x : \bullet\mathbf{1} \vdash y : \mathbf{1} \multimap \mathbf{1}}\ \multimap \mathsf{R}$$

We observe that this proof can be obtained from the first by repeatedly permuting the application of Conn with the rule application in its right premise. Concretely, we can permute Conn with $\multimap$ R and then with 1L. In the general case, we can show that applications of rule Conn can always be permuted so that abstraction or concretisation can be applied. We call such permutations *commuting conversions*, denoted by the congruence relation $\equiv$. We give a selection of the rules defining $\equiv$ in Figure 4.4 (see Appendix C for the other rules).

We report a representative case for each class of permutations, i.e., permutations with rules in the resource fragment, with C-rules, with other Conn, and Scope. The rules are self-explanatory.

A Conn rule can always be permuted with rules that do not manipulate the principal formulae it connects. Formally, we can prove that it is always possible to push an application of rule Conn up until it reaches one of the patterns required for abstraction.

Lemma 4.4.1 (One-Step Abstraction)**.** *Let $\mathcal{D}$ be a proof containing an application of Conn^x. Then, there exists a proof $\mathcal{D}'$ such that $\mathcal{D} \equiv \mathcal{D}'$ and $\mathcal{D}' \overset{x}{\dashrightarrow} \mathcal{E}$ for some $\mathcal{E}$ (abbreviated as $\mathcal{D} \equiv \overset{x}{\dashrightarrow} \mathcal{E}$).*

By Lemma 4.4.1, Conn applications can always be abstracted in any order. Differently, concretisation is order-sensitive. Consider the following example:

$$\dfrac{\dfrac{\dfrac{\dfrac{\triangleright \Delta_1 \vdash x : A \quad \triangleright \Delta_3, x : A, y : B \vdash z : C}{\triangleright \Delta_1 \vdash x : \bullet A \mid \Delta_3, x : \bullet A, y : B \vdash z : C}\ \mathsf{Conn}^x \quad \triangleright \Delta_4, z : C \vdash T}{\triangleright \Delta_1 \vdash x : \bullet A \mid \Delta_3, x : \bullet A, y : B \vdash z : \bullet C \mid \Delta_4, z : \bullet C \vdash T}\ \mathsf{Conn}^z \quad \triangleright \Delta_2 \vdash y : B}{\triangleright \Delta_1 \vdash x : \bullet A \mid \Delta_2 \vdash y : \bullet B \mid \Delta_3, x : \bullet A, y : \bullet B \vdash z : \bullet C \mid \Delta_4, z : \bullet C \vdash T}\ \mathsf{Conn}^y}{\triangleright \Delta_1, \Delta_2 \vdash y : \bullet(A \otimes B) \mid \Delta_3, y : \bullet(A \otimes B) \vdash z : \bullet C \mid \Delta_4, z : \bullet C \vdash T}\ \otimes\mathsf{C}^y}{\triangleright \Delta_3 \vdash z : \bullet((A \otimes B) \multimap C) \mid \Delta_1, \Delta_2, \Delta_4, z : \bullet((A \otimes B) \multimap C) \vdash T}\ \multimap\mathsf{C}^z$$

[Conn/ $\multimap$ R/R/2]

$$\dfrac{\triangleright\,\Psi_1 \mid \Gamma \vdash x{:}C \quad \dfrac{\triangleright\,\Psi_2 \mid \Delta, x{:}C, z{:}A \vdash y{:}B}{\triangleright\,\Psi_2 \mid \Delta, x{:}C \vdash y{:}A \multimap B}\ \multimap \mathsf{R}}{\triangleright\,\Psi_1 \mid \Psi_2 \mid \Gamma \vdash x{:}\bullet C \mid \Delta, x{:}\bullet C \vdash y{:}A \multimap B}\ \mathsf{Conn}^x \quad\equiv\quad \dfrac{\dfrac{\triangleright\,\Psi_1 \mid \Gamma \vdash x{:}C \quad \triangleright\,\Psi_2 \mid \Delta, x{:}C, z{:}A \vdash y{:}B}{\triangleright\,\Psi_1 \mid \Psi_2 \mid \Gamma \vdash x{:}\bullet C \mid \Delta, x{:}\bullet C, z{:}A \vdash y{:}B}\ \mathsf{Conn}^x}{\triangleright\,\Psi_1 \mid \Psi_2 \mid \Gamma \vdash x{:}\bullet C \mid \Delta, x{:}\bullet C \vdash y{:}A \multimap B}\ \multimap \mathsf{R}$$

[Conn/ $\otimes$ C/L/2]

$$\dfrac{\dfrac{\triangleright\,\Psi_1 \mid \Delta_1 \vdash z{:}\bullet A \mid \Delta_2 \vdash y{:}\bullet B \mid \Delta_3, z{:}\bullet A, y{:}\bullet B \vdash x{:}C}{\triangleright\,\Psi_1 \mid \Delta_1, \Delta_2 \vdash y{:}\bullet A \otimes B \mid \Delta_3, y{:}\bullet A \otimes B \vdash x{:}C}\ \otimes \mathsf{C}^y \quad \triangleright\,\Psi_2 \mid \Delta_4, x{:}C \vdash T}{\triangleright\,\Psi_1 \mid \Psi_2 \mid \Delta_1, \Delta_2 \vdash y{:}\bullet A \otimes B \mid \Delta_3, y{:}\bullet A \otimes B \vdash x{:}\bullet C \mid \Delta_4, x{:}\bullet C \vdash T}\ \mathsf{Conn}^x \quad\equiv$$

$$\dfrac{\dfrac{\triangleright\,\Psi_1 \mid \Delta_1 \vdash z{:}\bullet A \mid \Delta_2 \vdash y{:}\bullet B \mid \Delta_3, z{:}\bullet A, y{:}\bullet B \vdash x{:}C \quad \triangleright\,\Psi_2 \mid \Delta_4, x{:}C \vdash T}{\triangleright\,\Psi_1 \mid \Psi_2 \mid \Delta_1 \vdash z{:}\bullet A \mid \Delta_2 \vdash y{:}\bullet B \mid \Delta_3, z{:}\bullet A, y{:}\bullet B \vdash x{:}\bullet C \mid \Delta_4, x{:}\bullet C \vdash T}\ \mathsf{Conn}^x}{\triangleright\,\Psi_1 \mid \Psi_2 \mid \Delta_1, \Delta_2 \vdash y{:}\bullet A \otimes B \mid \Delta_3, y{:}\bullet A \otimes B \vdash x{:}\bullet C \mid \Delta_4, x{:}\bullet C \vdash T}\ \otimes \mathsf{C}^y$$

[Conn/Conn/2]

$$\dfrac{\triangleright\,\Psi_1 \mid \Delta_1 \vdash x{:}B \quad \dfrac{\triangleright\,\Psi_2 \mid \Delta_2, x{:}B \vdash y{:}A \quad \triangleright\,\Psi_3 \mid \Delta_3, y{:}A \vdash T}{\triangleright\,\Psi_2 \mid \Psi_3 \mid \Delta_2, x{:}B \vdash y{:}\bullet A \mid \Delta_3, y{:}\bullet A \vdash T}\ \mathsf{Conn}^y}{\triangleright\,\Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash x{:}\bullet B \mid \Delta_2, x{:}\bullet B \vdash y{:}\bullet A \mid \Delta_3, y{:}\bullet A \vdash T}\ \mathsf{Conn}^x \quad\equiv$$

$$\dfrac{\dfrac{\triangleright\,\Psi_1 \mid \Delta_1 \vdash x{:}B \quad \triangleright\,\Psi_2 \mid \Delta_2, x{:}B \vdash y{:}A}{\triangleright\,\Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash x{:}\bullet B \mid \Delta_2, x{:}\bullet B \vdash y{:}A}\ \mathsf{Conn}^x \quad \triangleright\,\Psi_3 \mid \Delta_3, y{:}A \vdash T}{\triangleright\,\Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash x{:}\bullet B \mid \Delta_2, x{:}\bullet B \vdash y{:}\bullet A \mid \Delta_3, y{:}\bullet A \vdash T}\ \mathsf{Conn}^y$$

[Conn/Scope/L/2]

$$\dfrac{\dfrac{\triangleright\,\Psi_1 \mid \Delta \vdash y{:}\bullet B \mid \Gamma_1, y{:}\bullet B \vdash x{:}A}{\triangleright\,\Psi_1 \mid \Delta, \Gamma_1 \vdash x{:}A}\ \mathsf{Scope}^y \quad \triangleright\,\Psi_2 \mid \Gamma_2, x{:}A \vdash T}{\triangleright\,\Psi_1 \mid \Psi_2 \mid \Delta, \Gamma_1 \vdash x{:}\bullet A \mid \Gamma_2, x{:}\bullet A \vdash T}\ \mathsf{Conn}^x \quad\equiv$$

$$\dfrac{\dfrac{\triangleright\,\Psi_1 \mid \Delta \vdash y{:}\bullet B \mid \Gamma_1, y{:}\bullet B \vdash x{:}A \quad \triangleright\,\Psi_2 \mid \Gamma_2, x{:}A \vdash T}{\triangleright\,\Psi_1 \mid \Psi_2 \mid \Delta \vdash y{:}\bullet B \mid \Gamma_1, y{:}\bullet B \vdash x{:}\bullet A \mid \Gamma_2, x{:}\bullet A \vdash T}\ \mathsf{Conn}^x}{\triangleright\,\Psi_1 \mid \Psi_2 \mid \Delta, \Gamma_1 \vdash x{:}\bullet A \mid \Gamma_2, x{:}\bullet A \vdash T}\ \mathsf{Scope}^y$$

Figure 4.4: LCL, commuting conversions for rule Conn (selection).

In the example above, we can neither concretise $\otimes\mathsf{C}^y$, because the Conn rules above do not match the required right-hand pattern from Figure 4.3, nor $\multimap \mathsf{C}^z$, because it is preceded by a $\otimes\mathsf{C}$. Even by using commuting conversions, we observe that in this case we cannot concretise $\multimap \mathsf{C}^z$ because it is not possible to permute Conn^z and Conn^y down to $\multimap \mathsf{C}^z$ as required for its concretisation. Intuitively, this reflects the fact that in this example $\multimap \mathsf{C}^z$ has a dependency on $\otimes\mathsf{C}^y$: $\multimap \mathsf{C}^z$ forms the connection $\bullet((A \otimes B) \multimap C)$ by using the connection $\bullet(A \otimes B)$ formed by $\otimes\mathsf{C}^y$. We further observe that it is instead possible to use the commuting conversion $[\mathsf{Conn}/\mathsf{Conn}/2]$ to permute Conn^x with Conn^z and reach the case for the concretisation rule $[\alpha\gamma_\otimes]$. By doing that and applying such concretisation, we reach the proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\triangleright \Delta_1 \vdash x{:}A} \quad \dfrac{}{\triangleright \Delta_2 \vdash y{:}B}}{\triangleright \Delta_1, \Delta_2 \vdash y{:}A \otimes B}\otimes\mathsf{R}}{\triangleright \Delta_1, \Delta_2 \vdash y{:}\bullet(A \otimes B) \mid \Delta_3, y{:}\bullet(A \otimes B) \vdash z{:}\bullet C \mid \Delta_4, z{:}\bullet C \vdash T}\,\mathsf{Conn}^y}{\triangleright \Delta_3 \vdash z{:}\bullet((A \otimes B) \multimap C) \mid \Delta_1, \Delta_2, \Delta_4, z{:}\bullet((A \otimes B) \multimap C) \vdash T}\multimap\mathsf{C}^z}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\triangleright \Delta_3, x{:}A, y{:}B \vdash z{:}C} \quad \dfrac{}{\triangleright \Delta_4, z{:}C \vdash T}}{\triangleright \Delta_3, x{:}A, y{:}B \vdash z{:}\bullet C \mid \Delta_4, z{:}\bullet C \vdash T}\,\mathsf{Conn}^z}{\triangleright \Delta_3, y{:}A \otimes B \vdash z{:}\bullet C \mid \Delta_4, z{:}\bullet C \vdash T}\otimes\mathsf{L}$$

In general, we say that a C-rule depends on a preceding C-rule if the former composes a connection formed by the latter. In the sequel, we write C^x for either $\otimes\mathsf{C}^x$ or $\multimap \mathsf{C}^x$. Moreover, $\mathcal{D} :: \triangleright \Psi$ denotes a proof $\mathcal{D}$ ending with the judgment $\triangleright \Psi$.

Lemma 4.4.2 (One-Step Concretisation)**.** *Let $\mathcal{D}$ be a proof containing a subderivation of the form:*

$$\dfrac{\mathcal{E} :: \triangleright \Psi'}{\triangleright \Psi}\ \mathsf{C}^x$$

such that C^x does not depend on any C*-rule in $\mathcal{E}$. Then, there exists a proof $\mathcal{D}'$ such that $\mathcal{D} \equiv \mathcal{D}'$ and $\mathcal{D}' \overset{x}{\cdots\!\rightarrow} \mathcal{F}$ for some $\mathcal{F}$ (abbreviated $\mathcal{D} \equiv \overset{x}{\cdots\!\rightarrow} \mathcal{F}$).*

It is easy to see from the definitions of commuting conversions, abstraction, and concretisation that judgements are always preserved:

Lemma 4.4.3 ($\alpha\gamma$ Preservation)**.**

- *If $\mathcal{D} :: \triangleright \Psi$ and $\mathcal{D} \equiv \mathcal{D}'$ then $\mathcal{D}' :: \triangleright \Psi$.*
- *If $\mathcal{D} :: \triangleright \Psi$ and $\mathcal{D} \overset{x}{\dashrightarrow} \mathcal{D}'$ then $\mathcal{D}' :: \triangleright \Psi$.*
- *If $\mathcal{D} :: \triangleright \Psi$ and $\mathcal{D} \overset{x}{\cdots\!\rightarrow} \mathcal{D}'$ then $\mathcal{D}' :: \triangleright \Psi$.*

We can now generalise our one-step results for abstraction and concretisation to characterise two normalisation procedures that respectively produce Conn-free or C-free proofs. In the following, we call the former α-normal and the latter γ-normal. Further, we denote the transitive closure of $\overset{x}{\dashrightarrow}$ up to commuting conversions $\equiv$ with $\overset{\tilde{x}}{\dashrightarrow\!\!\rightarrow}$ (resp. $\overset{\tilde{x}}{\cdots\!\!\rightarrow\!\!\!\!\rightarrow}$ for $\overset{x}{\cdots\!\rightarrow}$). As an example, we abbreviate $\mathcal{D} \equiv \overset{x}{\dashrightarrow} \equiv \overset{x'}{\dashrightarrow} \mathcal{E}$ with $\mathcal{D} \overset{x,x'}{\dashrightarrow\!\!\rightarrow} \mathcal{E}$.

Theorem 4.4.1 ($\alpha\gamma$ Normalisation)**.** *Let $\mathcal{D} :: \triangleright \Psi$ be a proof. Then:*

- *(α-normalisation) $\mathcal{D} \overset{\tilde{x}}{\dashrightarrow\!\!\rightarrow} \mathcal{E}$ for some $\tilde{x}$ and $\mathcal{E} :: \triangleright \Psi$ α-normal;*

- *(γ-normalisation)* $\mathcal{D} \overset{\tilde{x}}{\cdots\!\!\gg} \mathcal{F}$ *for some* $\tilde{x}$ *and* $\mathcal{F} :: \triangleright\ \Psi$ *γ-normal.*

Proof. By repeatedly applying Lemmas 4.4.1 and 4.4.3 for the first result, and Lemmas 4.4.2 and 4.4.3 for the second. □

4.4.2 β-Reductions

The Scope rule merges two sequents in a hypersequent by eliminating their connection, similarly to the Cut rule in ILL. Just like in ILL, where applications of the Cut rule can always be eliminated, we prove that Scope can be eliminated from any proof in LCL.

As an example, the following proof contains two applications of the Scope rule:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\triangleright \cdot \vdash x : \mathbf{1}}\,\mathsf{1R}}{\triangleright \cdot \vdash y : \bullet\mathbf{1} \mid y : \bullet\mathbf{1} \vdash x : \mathbf{1}}\,\mathsf{1C}^{y}}{\triangleright \cdot \vdash x : \mathbf{1}}\,\mathsf{Scope}^{y} \qquad \dfrac{\dfrac{}{\triangleright \cdot \vdash z : \mathbf{1}}\,\mathsf{1R}}{\triangleright x : \mathbf{1} \vdash z : \mathbf{1}}\,\mathsf{1L}}{\triangleright \cdot \vdash x : \bullet\mathbf{1} \mid x : \bullet\mathbf{1} \vdash z : \mathbf{1}}\,\mathsf{Conn}^{x}}{\triangleright \cdot \vdash z : \mathbf{1}}\,\mathsf{Scope}^{x}$$

Above, Scope^{x} delimits the scope of a connection x that is introduced by a Conn^{x} rule and Scope^{y} delimits the scope of connection y introduced by $\mathsf{1C}^{y}$. Applications of rule Scope can be eliminated by using β-reductions, denoted by $\mathcal{D} \xrightarrow{t} \mathcal{D}'$, where t can be x or $\bullet x$; formally, we define $\mathcal{D} \xrightarrow{t} \mathcal{D}'$ as the context closure of the rules that can be found in Figure 4.5.

There are six β-rules. The first three rules define reductions within the resource fragment and the other three within the connection fragment. β-reductions preserve provability. We illustrate the reduction of Scope^{y} in the example above, using two β-rules labelled $\bullet y$ and x.

$$\ldots \xrightarrow{\bullet y} \dfrac{\dfrac{\dfrac{}{\triangleright \cdot \vdash x : \mathbf{1}}\,\mathsf{1R} \qquad \dfrac{\dfrac{}{\triangleright \cdot \vdash z : \mathbf{1}}\,\mathsf{1R}}{\triangleright x : \mathbf{1} \vdash z : \mathbf{1}}\,\mathsf{1L}}{\triangleright \cdot \vdash x : \bullet\mathbf{1} \mid x : \bullet\mathbf{1} \vdash z : \mathbf{1}}\,\mathsf{Conn}^{x}}{\triangleright \cdot \vdash z : \mathbf{1}}\,\mathsf{Scope}^{x} \xrightarrow{x} \dfrac{}{\triangleright \cdot \vdash z : \mathbf{1}}\,\mathsf{1R}$$

Similarly to α- and γ-reductions, β-reduction might not be directly applicable to a proof, because a Scope-rule needs to be permuted to an appropriate place in the derivation. In Figure 4.4, we have already encountered selected commuting conversions for Conn, and in Figure 4.6 we complement them by selected commuting conversions rules for Scope.

A Scope rule can always be permuted with other rules that do not manipulate its principal formulae. Formally, we can prove that it is always possible to push an application of rule Scope up, until we match one of the patterns required for β-reduction.

Lemma 4.4.4 (One-Step Reduction)**.** *Let $\mathcal{D}$ be a proof containing an application of* Scope^{x}*. Then, there exists a proof $\mathcal{D}'$ such that $\mathcal{D} \equiv \mathcal{D}'$ and $\mathcal{D}' \xrightarrow{t} \mathcal{E}$, where t is either x or $\bullet x$ and for some $\mathcal{E}$ (abbreviated as $\mathcal{D} \equiv \xrightarrow{x} \mathcal{E}$).*

Proposition 4.4.4.1 (β Preservation)**.** *If $\mathcal{D} :: \triangleright\ \Psi$ and $\mathcal{D} \xrightarrow{t} \mathcal{D}'$ then $\mathcal{D}' :: \triangleright\ \Psi$.*

$$[\beta_{\mathbf{1}}] \quad \dfrac{\dfrac{\dfrac{}{\triangleright \cdot \vdash x{:}\mathbf{1}}\,\mathsf{1R} \quad \dfrac{\triangleright \Psi \mid \Delta \vdash T}{\triangleright \Psi \mid \Delta, x{:}\mathbf{1} \vdash T}\,\mathsf{1L}}{\triangleright \Psi \mid \cdot \vdash x{:}\bullet\mathbf{1} \mid \Delta, x{:}\bullet\mathbf{1} \vdash T}\,\mathsf{Conn}^x}{\triangleright \Psi \mid \Delta \vdash T}\,\mathsf{Scope}^x \qquad \xrightarrow{x} \qquad \triangleright \Psi \mid \Delta \vdash T$$

$$[\beta_{\otimes}] \quad \dfrac{\dfrac{\dfrac{\triangleright \Psi_2 \mid \Delta_2 \vdash y{:}A \quad \triangleright \Psi_1 \mid \Delta_1 \vdash x{:}B}{\triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1, \Delta_2 \vdash x{:}A \otimes B}\,\otimes\mathsf{R} \quad \dfrac{\triangleright \Psi_3 \mid \Delta_3, y{:}A, x{:}B \vdash T}{\triangleright \Psi_3 \mid \Delta_3, x{:}A \otimes B \vdash T}\,\otimes\mathsf{L}}{\triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1, \Delta_2 \vdash x{:}\bullet A \otimes B \mid \Delta_3, x{:}\bullet A \otimes B \vdash T}\,\mathsf{Conn}^x}{\triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1, \Delta_2, \Delta_3 \vdash T}\,\mathsf{Scope}^x \quad \xrightarrow{x}$$

$$\dfrac{\dfrac{\dfrac{\triangleright \Psi_1 \mid \Delta_1 \vdash x{:}B \quad \dfrac{\triangleright \Psi_2 \mid \Delta_2 \vdash y{:}A \quad \triangleright \Psi_3 \mid \Delta_3, y{:}A, x{:}B \vdash T}{\triangleright \Psi_2 \mid \Psi_3 \mid \Delta_2 \vdash y{:}\bullet A \mid \Delta_3, y{:}\bullet A, x{:}B \vdash T}\,\mathsf{Conn}^y}{\triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash x{:}\bullet B \mid \Delta_2 \vdash y{:}\bullet A \mid \Delta_3, y{:}\bullet A, x{:}\bullet B \vdash T}\,\mathsf{Conn}^x}{\triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash x{:}\bullet B \mid \Delta_2, \Delta_3, x{:}\bullet B \vdash T}\,\mathsf{Scope}^y}{\triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1, \Delta_2, \Delta_3 \vdash T}\,\mathsf{Scope}^x$$

$$[\beta_{\multimap}] \quad \dfrac{\dfrac{\dfrac{\triangleright \Psi_2 \mid \Delta_2, y{:}A \vdash x{:}B}{\triangleright \Psi_2 \mid \Delta_2 \vdash x{:}A \multimap B}\,\multimap\mathsf{R} \quad \dfrac{\triangleright \Psi_1 \mid \Delta_1 \vdash y{:}A \quad \triangleright \Psi_3 \mid \Delta_3, x{:}B \vdash T}{\triangleright \Psi_1 \mid \Psi_3 \mid \Delta_1, \Delta_3, x{:}A \multimap B \vdash T}\,\multimap\mathsf{L}}{\triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1, \Delta_2, x{:}\bullet A \multimap B \vdash T \mid \Delta_3 \vdash x{:}\bullet A \multimap B}\,\mathsf{Conn}^x}{\triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1, \Delta_2, \Delta_3 \vdash T}\,\mathsf{Scope}^x \qquad \xrightarrow{x}$$

$$\dfrac{\dfrac{\dfrac{\triangleright \Psi_1 \mid \Delta_1 \vdash y{:}A \quad \triangleright \Psi_2 \mid \Delta_2, y{:}A \vdash x{:}B}{\triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1 \vdash y{:}\bullet A \mid \Delta_2, y{:}\bullet A \vdash x{:}B}\,\mathsf{Conn}^y \quad \triangleright \Psi_3 \mid \Delta_3, x{:}B \vdash T}{\triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash y{:}\bullet A \mid \Delta_2, y{:}\bullet A \vdash x{:}\bullet B \mid \Delta_3, x{:}\bullet B \vdash T}\,\mathsf{Conn}^x}{\triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1, \Delta_2 \vdash x{:}\bullet B \mid \Delta_3, x{:}\bullet B \vdash T}\,\mathsf{Scope}^y}{\triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1, \Delta_2, \Delta_3 \vdash T}\,\mathsf{Scope}^x$$

$$[\beta_{\mathbf{1}\mathsf{C}}] \quad \dfrac{\dfrac{\triangleright \Psi \mid \Delta \vdash T}{\triangleright \Psi \mid \cdot \vdash x{:}\bullet\mathbf{1} \mid \Delta, x{:}\bullet\mathbf{1} \vdash T}\,\mathsf{1C}^x}{\triangleright \Psi \mid \Delta \vdash T}\,\mathsf{Scope}^x \quad \xrightarrow{\bullet x} \quad \triangleright \Psi \mid \Delta \vdash T$$

$$[\beta_{\otimes\mathsf{C}}] \quad \dfrac{\dfrac{\dfrac{\triangleright \Psi \mid \Delta_1 \vdash x{:}\bullet B \mid \Delta_2 \vdash y{:}\bullet A \mid \Delta_3, y{:}\bullet A, x{:}\bullet B \vdash T}{\triangleright \Psi \mid \Delta_1, \Delta_2 \vdash x{:}\bullet A \otimes B \mid \Delta_3, x{:}\bullet A \otimes B \vdash T}\,\otimes\mathsf{C}^x}{\triangleright \Psi \mid \Delta_1, \Delta_2, \Delta_3 \vdash T}\,\mathsf{Scope}^x \quad \xrightarrow{\bullet x}$$

$$\dfrac{\dfrac{\triangleright \Psi \mid \Delta_1 \vdash x{:}\bullet B \mid \Delta_2 \vdash y{:}\bullet A \mid \Delta_3, y{:}\bullet A, x{:}\bullet B \vdash T}{\triangleright \Psi \mid \Delta_1 \vdash x{:}\bullet B \mid \Delta_2, \Delta_3, x{:}\bullet B \vdash T}\,\mathsf{Scope}^y}{\triangleright \Psi \mid \Delta_1, \Delta_2, \Delta_3 \vdash T}\,\mathsf{Scope}^x$$

$$[\beta_{\multimap\mathsf{C}}] \quad \dfrac{\dfrac{\dfrac{\triangleright \Psi \mid \Delta_1 \vdash y{:}\bullet A \mid \Delta_2, x{:}\bullet B \vdash T \mid \Delta_3, y{:}\bullet A \vdash x{:}\bullet B}{\triangleright \Psi \mid \Delta_1, \Delta_2, x{:}\bullet A \multimap B \vdash T \mid \Delta_3 \vdash x{:}\bullet A \multimap B}\,\multimap\mathsf{C}^x}{\triangleright \Psi \mid \Delta_1, \Delta_2, \Delta_3 \vdash T}\,\mathsf{Scope}^x \quad \xrightarrow{\bullet x}$$

$$\dfrac{\dfrac{\dfrac{\triangleright \Psi \mid \Delta_1 \vdash y{:}\bullet A \mid \Delta_2, x{:}\bullet B \vdash T \mid \Delta_3, y{:}\bullet A \vdash x{:}\bullet B}{\triangleright \Psi \mid \Delta_1, \Delta_3 \vdash x{:}\bullet B \mid \Delta_2, x{:}\bullet B \vdash T}\,\mathsf{Scope}^y}{\triangleright \Psi \mid \Delta_1, \Delta_2, \Delta_3 \vdash T}\,\mathsf{Scope}^x$$

Figure 4.5: LCL, β-reductions.

$$[\mathsf{Scope}/\otimes\mathsf{R}/\mathsf{R}]\quad \dfrac{\dfrac{\triangleright\,\Psi_1 \mid \Gamma_1 \vdash y{:}A \qquad \triangleright\,\Psi_2 \mid \Delta \vdash z{:}\bullet C \mid \Gamma_2, z{:}\bullet C \vdash x{:}B}{\triangleright\,\Psi_1 \mid \Psi_2 \mid \Delta \vdash z{:}\bullet C \mid \Gamma_1, \Gamma_2, z{:}\bullet C \vdash x{:}A \otimes B}\otimes\mathsf{R}}{\triangleright\,\Psi_1 \mid \Psi_2 \mid \Delta, \Gamma_1, \Gamma_2 \vdash x{:}A \otimes B}\,\mathsf{Scope} \quad\equiv\quad \dfrac{\triangleright\,\Psi_1 \mid \Gamma_1 \vdash y{:}A \qquad \dfrac{\triangleright\,\Psi_2 \mid \Delta \vdash z{:}\bullet C \mid \Gamma_2, z{:}\bullet C \vdash x{:}B}{\triangleright\,\Psi_2 \mid \Delta, \Gamma_2 \vdash x{:}B}\,\mathsf{Scope}}{\triangleright\,\Psi_1 \mid \Psi_2 \mid \Delta, \Gamma_1, \Gamma_2 \vdash x{:}A \otimes B}\otimes\mathsf{R}$$

$$[\mathsf{Scope}/\otimes\mathsf{C}/2]\quad \dfrac{\dfrac{\triangleright\,\Psi \mid \Delta_1 \vdash z{:}\bullet C \mid \Delta_2, z{:}\bullet C \vdash y{:}\bullet A \mid \Delta_3 \vdash x{:}\bullet B \mid \Delta_4, y{:}\bullet A, x{:}\bullet B \vdash T}{\triangleright\,\Psi \mid \Delta_1 \vdash z{:}\bullet C \mid \Delta_2, \Delta_3, z{:}\bullet C \vdash x{:}\bullet A \otimes B \mid \Delta_4, x{:}\bullet A \otimes B \vdash T}\otimes\mathsf{C}}{\triangleright\,\Psi \mid \Delta_1, \Delta_2, \Delta_3 \vdash x{:}\bullet A \otimes B \mid \Delta_3, x{:}\bullet A \otimes B \vdash T}\,\mathsf{Scope} \quad\equiv\quad \dfrac{\triangleright\,\Psi \mid \Delta_1 \vdash z{:}\bullet C \mid \Delta_2, z{:}\bullet C \vdash y{:}\bullet A \mid \Delta_3 \vdash x{:}\bullet B \mid \Delta_4, y{:}\bullet A, x{:}\bullet B \vdash T}{\dfrac{\triangleright\,\Psi \mid \Delta_1, \Delta_2 \vdash y{:}\bullet A \mid \Delta_3 \vdash x{:}\bullet B \mid \Delta_4, y{:}\bullet A, x{:}\bullet B \vdash T}{\triangleright\,\Psi \mid \Delta_1, \Delta_2, \Delta_3 \vdash x{:}\bullet A \otimes B \mid \Delta_3, x{:}\bullet A \otimes B \vdash T}\otimes\mathsf{C}}\,\mathsf{Scope}$$

$$[\mathsf{Scope}/\mathsf{Scope}/2]\quad \dfrac{\dfrac{\triangleright\,\Psi \mid \Delta_1 \vdash y{:}\bullet A \mid \Delta_2 \vdash x{:}\bullet B \mid \Delta_3, y{:}\bullet A, x{:}\bullet B \vdash T}{\triangleright\,\Psi \mid \Delta_1 \vdash y{:}\bullet A \mid \Delta_2, \Delta_3, y{:}\bullet A \vdash T}\,\mathsf{Scope}}{\triangleright\,\Psi \mid \Delta_1, \Delta_2, \Delta_3 \vdash T}\,\mathsf{Scope} \quad\equiv\quad \dfrac{\triangleright\,\Psi \mid \Delta_1 \vdash y{:}\bullet A \mid \Delta_2 \vdash x{:}\bullet B \mid \Delta_3, yA, x{:}\bullet B \vdash T}{\dfrac{\triangleright\,\Psi \mid \Delta_2 \vdash x{:}\bullet B \mid \Delta_1, \Delta_3, x{:}\bullet B \vdash T}{\triangleright\,\Psi \mid \Delta_1, \Delta_2, \Delta_3 \vdash T}\,\mathsf{Scope}}\,\mathsf{Scope}$$

$$[\mathsf{Scope}/\mathsf{Conn}/\mathsf{L}/2]\quad \dfrac{\dfrac{\dfrac{\triangleright\,\Psi_1 \mid \Delta \vdash x{:}\bullet B \mid \Gamma_1, x{:}\bullet B \vdash y{:}A \qquad \triangleright\,\Psi_2 \mid \Gamma_2, y{:}A \vdash T}{\triangleright\,\Psi_1 \mid \Psi_2 \mid \Delta \vdash x{:}\bullet B \mid \Gamma_1, x{:}\bullet B \vdash y{:}\bullet A \mid \Gamma_2, y{:}\bullet A \vdash T}\,\mathsf{Conn}}{\triangleright\,\Psi_1 \mid \Psi_2 \mid \Delta, \Gamma_1 \vdash y{:}\bullet A \mid \Gamma_2, y{:}\bullet A \vdash T}\,\mathsf{Scope} \quad\equiv\quad \dfrac{\dfrac{\triangleright\,\Psi_1 \mid \Delta \vdash x{:}\bullet B \mid \Gamma_1, x{:}\bullet B \vdash y{:}A}{\triangleright\,\Psi_1 \mid \Delta, \Gamma_1 \vdash y{:}A}\,\mathsf{Scope} \qquad \triangleright\,\Psi_2 \mid \Gamma_2, y{:}A \vdash T}{\triangleright\,\Psi_1 \mid \Psi_2 \mid \Delta, \Gamma_1 \vdash y{:}\bullet A \mid \Gamma_2, y{:}\bullet A \vdash T}\,\mathsf{Conn}$$

Figure 4.6: LCL, commuting conversions for rule Scope (selection).

We can now generalize our one-step reduction result and characterize a multi-step normalization procedure that produces Scope-free proofs. In the following we call such a proof β-normal. Furthermore, we denote the transitive closure of $\rightarrow$ up to commuting conversions $\equiv$ with $\longrightarrow\!\!\!\!\!\rightarrow$.

Theorem 4.4.2 (β Normalisation)**.** *Let $\mathcal{D} :: \triangleright\ \Psi$ be a proof then$\mathcal{D} \overset{\tilde{t}}{\longrightarrow\!\!\!\!\!\rightarrow} \mathcal{D}'$ for some $\tilde{x}$ and $\mathcal{E} :: \triangleright\ \Psi$ β-normal.*

4.4.3 Correspondence Theorem

We can now present the main result of our development of LCL, namely the Correspondence Theorem below:

Theorem 4.4.3 (Correspondence Theorem)**.** *Let $\tilde{x} = x_1, \ldots, x_n$ and $\mathsf{rev}(\tilde{x}) = x_n, \ldots, x_1$. Then,*

- *(α-correspondence) $\mathcal{D} \overset{\tilde{x}}{\longrightarrow\!\!\!\!\!\rightarrow} \mathcal{F}$ for some $\mathcal{F}$ implies that $\mathcal{D} \overset{\tilde{x}}{\dashrightarrow\!\!\!\!\!\rightarrow} \mathcal{E}$ for some $\mathcal{E}$ such that $\mathcal{E} \overset{\bullet\tilde{x}}{\longrightarrow\!\!\!\!\!\rightarrow} \mathcal{F}$.*
- *(γ-correspondence) $\mathcal{D} \overset{\bullet\tilde{x}}{\longrightarrow\!\!\!\!\!\rightarrow} \mathcal{F}$ for some $\mathcal{F}$ implies that $\mathcal{D} \overset{\mathsf{rev}(\tilde{x})}{\cdots\!\!\!\!\!\rightarrow} \mathcal{E}$ for some $\mathcal{E}$ such that $\mathcal{E} \overset{\tilde{x}}{\longrightarrow\!\!\!\!\!\rightarrow} \mathcal{F}$.*

Proof. We show the proof for α-correspondence; the proof for γ-correspondence is similar: the only difference is that it needs to be proven separately for each connective.

We can depict α-correspondence with the following diagram:

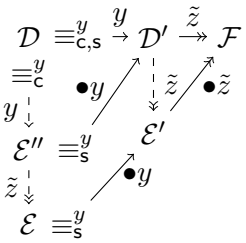
$$\begin{array}{ccc} \mathcal{D} & \overset{\tilde{x}}{\twoheadrightarrow} & \mathcal{F} \\ \tilde{x} \downarrow & \nearrow \bullet\tilde{x} & \\ \mathcal{E} & & \end{array}$$

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

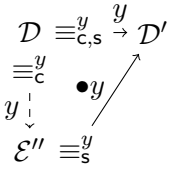
- **Case** $\tilde{x}$ is empty. The thesis holds for $\mathcal{D} = \mathcal{E} = \mathcal{F}$.
- **Case** $\tilde{x} = y, \tilde{z}$. In this case we know that $\mathcal{D} \equiv^{y}_{\mathsf{c,s}} \overset{y}{\rightarrow} \mathcal{D}'$. Our induction hypothesis is then that there exists $\mathcal{E}'$ α-normal such that:

$$\begin{array}{ccc} \mathcal{D}' & \overset{\tilde{z}}{\twoheadrightarrow} & \mathcal{F} \\ \tilde{z} \downarrow & \nearrow \bullet\tilde{z} & \\ \mathcal{E}' & & \end{array}$$

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



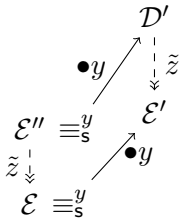
We can split the diagram in two parts. The first part is the upper-left triangle that instantiates every arrow to its one-time application:



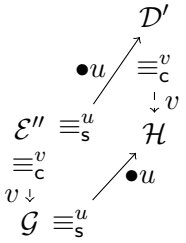
This diagram commutes because, from the definitions of the proof transformations in Figures 4.3–4.6, we can derive:

$$\equiv^{y}_{\mathsf{c}} \equiv^{y}_{\mathsf{s}} \xrightarrow{y} \quad = \quad \equiv^{y}_{\mathsf{c}} \dashrightarrow^{y} \equiv^{y}_{\mathsf{s}} \xrightarrow{\bullet y}$$

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



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



This diagram commutes because we can derive the equality:

$$\equiv^{u}_{\mathsf{s}} \xrightarrow{\bullet u} \equiv^{v}_{\mathsf{c}} \dashrightarrow^{v} \quad = \quad \equiv^{v}_{\mathsf{c}} \dashrightarrow^{v} \equiv^{u}_{\mathsf{s}} \xrightarrow{\bullet u}$$

□

4.5 Additive Fragment

The logic LCL introduced in § 4.3 and the proof transformations introduced in § 4.4 cover only the multiplicative fragment of linear logic, but it does not (yet) allow us to express choice or selection. In linear logic these are respectively captured by the connectives of additive conjunction $A\&B$ and additive disjunction $A \oplus B$. $A\&B$ means that A or B can be made available but not both, whereas $A \oplus B$ means that either A is available or B is available but we do not know which one. We extend LCL accordingly:

$$\textit{(Propositions)} \quad A, B \quad ::= \quad \ldots \mid A \oplus B \mid A\&B$$

Resource Fragment. The left and right rules for additive conjunction $\&$ for the resource fragment are a straightforward adoption of the standard rules from ILL. Observe that the context is not split in the rules $\&\mathsf{R}$ and $\oplus\mathsf{L}$, a hallmark characteristic of the additives.

$$\frac{\triangleright\, \Psi \mid \Delta \vdash x{:}A \quad \triangleright\, \Psi \mid \Delta \vdash x{:}B}{\triangleright\, \Psi \mid \Delta \vdash x{:}A\&B}\ \&\mathsf{R}$$

$$\frac{\triangleright\, \Psi \mid \Delta, x{:}A \vdash T}{\triangleright\, \Psi \mid \Delta, x{:}A\&B \vdash T}\ \&\mathsf{L}_1 \qquad \frac{\triangleright\, \Psi \mid \Delta, x{:}B \vdash T}{\triangleright\, \Psi \mid \Delta, x{:}A\&B \vdash T}\ \&\mathsf{L}_2$$

The left and right rules for the $\oplus$ are also a straightforward adoption of the standard rules for linear logic.

$$\frac{\triangleright\, \Psi \mid \Delta \vdash x{:}A}{\triangleright\, \Psi \mid \Delta \vdash x{:}A \oplus B}\ \oplus\mathsf{R}_1 \qquad \frac{\triangleright\, \Psi \mid \Delta \vdash x{:}B}{\triangleright\, \Psi \mid \Delta \vdash x{:}A \oplus B}\ \oplus\mathsf{R}_2$$

$$\frac{\triangleright\, \Psi \mid \Delta, x{:}A \vdash T \quad \triangleright\, \Psi \mid \Delta, x{:}B \vdash T}{\triangleright\, \Psi \mid \Delta, x{:}A \oplus B \vdash T}\ \oplus\mathsf{L}$$

Connection Fragment. We turn now to the connection fragment, where we add four connection rules for $A\&B$ and $A\oplus B$. We define two connections rule for $A\&B$ expressing external choice, and two rules for $A \oplus B$, expressing external selection.

$$\frac{\triangleright\, \Psi \mid \Psi' \mid \Delta_1 \vdash x{:}\bullet A \mid \Delta_2, x{:}\bullet A \vdash T \quad \triangleright\, \Psi' \mid \Delta_1 \vdash x{:}B}{\triangleright\, \Psi \mid \Psi' \mid \Delta_1 \vdash x{:}\bullet A\&B \mid \Delta_2, x{:}\bullet A\&B \vdash T}\ \&\mathsf{C}_1$$

$$\frac{\triangleright\, \Psi \mid \Delta_1 \vdash x{:}A \quad \triangleright\, \Psi \mid \Psi' \mid \Delta_1 \vdash x{:}\bullet B \mid \Delta_2, x{:}\bullet B \vdash T}{\triangleright\, \Psi \mid \Psi' \mid \Delta_1 \vdash x{:}\bullet A\&B \mid \Delta_2, x{:}\bullet A\&B \vdash T}\ \&\mathsf{C}_2$$

$$\frac{\triangleright\, \Psi \mid \Psi' \mid \Delta_1 \vdash x{:}\bullet A \mid \Delta_2, x{:}\bullet A \vdash T \quad \triangleright\, \Psi' \mid \Delta_2, x{:}B \vdash T}{\triangleright\, \Psi \mid \Psi' \mid \Delta_1 \vdash x{:}\bullet A \oplus B \mid \Delta_2, x{:}\bullet A \oplus B \vdash T}\ \oplus\mathsf{C}_1$$

$$\frac{\triangleright\, \Psi \mid \Delta_2, x{:}A \vdash T \quad \triangleright\, \Psi \mid \Psi' \mid \Delta_1 \vdash x{:}\bullet B \mid \Delta_2, x{:}\bullet B \vdash T}{\triangleright\, \Psi \mid \Psi' \mid \Delta_1 \vdash x{:}\bullet A \oplus B \mid \Delta_2, x{:}\bullet A \oplus B \vdash T}\ \oplus\mathsf{C}_2$$

Properties. We can extend α, γ, and β-reduction to the additive fragment in a straightforward way. We omit the rules here (see Appendix C).

The seemingly irrelevant premises in some of the rules for the additives (the leftmost premise of $\oplus\mathsf{C}_1$ and $\&\mathsf{C}_1$ and the rightmost premise of $\oplus\mathsf{C}_2$ and $\&\mathsf{C}_2$) are there for bookkeeping purposes; they ensure that the resources are appropriately consumed in the branches that were not selected, and permit us to prove all our results obtained so far also for this extended version of LCL.

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

Figure 4.7: Internal Compositional Choreographies, syntax.

4.6 Typing Choreographies with LCL

In this Section we present the calculus of Internal Compositional Choreographies (ICC), a fragment of that presented in Chapter 3 with only internal mobility (from the internal π-calculus [93]). We then develop a typing discipline for internal compositional choreographies by using them as proof terms for proofs in LCL, thus establishing a Curry-Howard correspondence between our calculus and LCL and revisiting the results from § 4.4.

4.6.1 Syntax

Let x, y, z range over a set of names $\mathcal{N}$. Then, the syntax of compositional choreographies, denoted by P, is reported in Figure 4.7. Compositional choreographies P can be either processes performing process actions, which form the *process fragment* of ICC, or choreographies of system communications, which form the *choreographic fragment* of ICC.

Processes. An *(output)* $\overline{x}(y);(P \mid Q)$ sends a *fresh* name y over channel x and then proceeds with the parallel composition $P \mid Q$, whereas an *(input)* $x(y);P$ receives y over x and proceeds as P. In a *(left sel)* $x.\mathsf{inl};P$, we send over channel x our choice of the left branch offered by the receiver. The term *(right sel)* $x.\mathsf{inr};P$ selects the right branch instead. Selections communicate with the term *(case)* $x.\mathsf{case}(P,Q)$, which offers a left branch P and a right branch Q. The term *(par)* $P \mid_x P$ models parallel composition; here, differently from the output case, the two composed processes are not independent, but share the communication channel x. The term *(res)* is the standard restriction. Terms *(close)* and *(wait)* model, respectively, the request and acceptance for closing a channel, following the typical closing handshake in real-world communication protocols such as TCP.

Choreographies. Term *(res)* for name restriction is the same as for processes. A *(global com)* $\overrightarrow{x(y)};P$ describes a system where a fresh name y is communicated over a channel x, and then continues as P, where y is bound in P. The terms *(global left sel)* and *(global right sel)* model systems where, respectively, a left branch or a right branch is selected on channel x. Unused branches in global selections, e.g., Q in $\overrightarrow{x.\mathsf{l}}\,(P,Q)$, are unnecessary in our setting since they are never executed; however, they will be convenient

$$\frac{P \triangleright \Psi \mid \Delta \vdash T}{\mathsf{wait}[x]; P \triangleright \Psi \mid \Delta, x{:}\mathbf{1} \vdash T}\ \mathsf{T1L} \qquad \frac{}{\mathsf{close}[x] \triangleright \cdot \vdash x{:}\mathbf{1}}\ \mathsf{T1R}$$

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

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

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

$$\frac{P \triangleright \Psi \mid \Delta \vdash x{:}A}{x.\mathsf{inl}; P \triangleright \Psi \mid \Delta \vdash x{:}A \oplus B}\ \mathsf{T} \oplus \mathsf{R}_1 \qquad \frac{Q \triangleright \Psi \mid \Delta \vdash x{:}B}{x.\mathsf{inr}; Q \triangleright \Psi \mid \Delta \vdash x{:}A \oplus B}\ \mathsf{T} \oplus \mathsf{R}_2$$

$$\frac{P \triangleright \Psi \mid \Delta, x{:}A \vdash T}{x.\mathsf{inl}; P \triangleright \Psi \mid \Delta, x{:}A\&B \vdash T}\ \mathsf{T}\&\mathsf{L}_1 \qquad \frac{Q \triangleright \Psi \mid \Delta, x{:}B \vdash T}{x.\mathsf{inr}; Q \triangleright \Psi \mid \Delta, x{:}A\&B \vdash T}\ \mathsf{T}\&\mathsf{L}_2$$

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

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

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

Figure 4.8: Internal Compositional Choreographies, typing rules for the process fragment.

for our development of endpoint projection, which will follow the concretisation transformation for LCL proofs. Finally, term *(global close)* models the closure of a channel.

4.6.2 Typing

We can now show how compositional choreographies can be typed using the proof theory of LCL. A typing judgement has the following form:

$$P \triangleright \Psi$$

Above, P is highlighted for readability. Now, in terms of channels, we say that a sequent $x_1 : T_1, \ldots, x_n : T_n \vdash x : T$ provides usage T on channel x and requires the usages $T_1, \ldots, T_n$ on their respective channels $x_1, \ldots, x_n$.

The typing rules defining when a term P is well-typed are given in Figure 4.8 and Figure 4.9. The rules are exactly the same as those for LCL; the only difference is that they are now put in correspondence with the ICC terms that they type. Observe that the rules for typing the process fragment are those of the resource fragment of LCL, whereas the rules for typing the choreographic fragment of ICC are those of the connection fragment of LCL, eliciting the logical difference between processes and choreographies. The rules in

$$\frac{P \triangleright \Psi \mid \Delta \vdash T}{\overrightarrow{\mathsf{close}[x]}; P \triangleright \Psi \mid \cdot \vdash x{:}\bullet\mathbf{1} \mid \Delta, x{:}\bullet\mathbf{1} \vdash T}\ \mathsf{T1C}$$

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

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

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

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

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

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

Figure 4.9: Internal Compositional Choreographies, typing rules for the choreographic fragment.

Figure 4.8, corresponding to the process fragment, type terms of the internal π-calculus and follows the idea of sessions types as proposed in [27]. The two rules for $\mathsf{T1L}$ and $\mathsf{T1R}$ type, respectively, the acceptance and request for closing a channel. The rules for $\otimes$ correspond to input and output. More precisely, $\mathsf{T} \otimes \mathsf{L}$ types a process performing an input of a name y over a channel x. As in standard session typing [54], the continuation of the process P will be able to interact over y and x with type A and B respectively. Dually, $\mathsf{T} \otimes \mathsf{R}$ types a process performing an output over channel x of some fresh channel y. Once such output has been performed, two processes, P and Q will handle, separately, channel y and channel x respectively. Note that, as observed in [27, 103], forcing that the two channels are operated by different processes in parallel avoids any possible deadlock caused by the interleaving of y and x. The cases for $\multimap$, $\oplus$, and $\&$ are treated similarly. Rule Conn types the parallel composition of two processes that can be connected; the latter is registered by the annotation x on the parallel constructor. Rule Scope types name restriction.

We now comment the typing rules for the choreographic fragment in Figure 4.9. Rule $\mathsf{1C}$ is a weakening rule for connections on the unit type. As in the case of $\mathsf{1L}$, the process term does not change. In rule $\otimes\mathsf{C}$, we type a communication $\overrightarrow{x(y)}; P$ with $A \otimes B$ if the continuation P performs independent communications on x and y. Differently, rule $\multimap\ \mathsf{C}$ types a communication $\overrightarrow{x(y)}; P$ when the communications on x in P are causally dependent on the communications on y. In rule $\oplus\mathsf{C}_1$ and $\oplus\mathsf{C}_2$, we type the global selection of a left and a right branch respectively. Rules $\&\mathsf{C}_1$ and $\&\mathsf{C}_2$ also type global selection. The difference between the two operators is that branching is either on the required or the

provided behaviour of x.

Remark 4.6.1 (Process identifiers). In typical choreography calculi, such as those presented in Chapters 2 and 3, choreography terms identify the processes involved in a communication explicitly using an Alice-Bob notation [61, 31]. For instance, a communication of a bound name y over a channel x would be written as $\mathsf{p} \texttt{ -> } \mathsf{q} : x(y)$, where p and q are process identifiers representing respectively the sender and the receiver for the communication. In ICC, process identifiers are implicit: for each communication $\overrightarrow{x(y)}$ there are a sender and a receiver that can be automatically inferred by applying our concretisation transformation γ. Omitting process identifiers is just a matter of presentational convenience, as our typing with hypersequents makes them redundant; a way of retaining them would be to annotate each sequent in a hypersequent with a process name, e.g., for rule $\mathsf{T} \otimes \mathsf{C}$:

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

□

Example 4.6.2. We formalise and extend our example from § 4.2 as follows:

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

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

Process $P_{\mathsf{client}'}$, the new version of the client, implements a client that selects the right choice of a branching on channel x and then asks for some tea; then, it proceeds as P_{client} from § 4.2. Note that we have enhanced the processes with all expected closing of channels. The server $P_{\mathsf{server}'}$, instead, now offers to the client a choice between buying a tea (as in § 4.2) and getting a free glass of water. Since the water is free, the payment to the bank is not performed in this case. In either case, the bank is notified of whether a payment will occur or not, respectively right and left branch in P_{bank}. The processes are composed as a system in P.

Term C is the equivalent choreographic representation of P (where $\mathsf{close}[\mathit{tea}, \overrightarrow{p}, tr, m, x, b]$ is a shortcut for $\overrightarrow{\mathsf{close}}[\mathit{tea}]; \ldots; \overrightarrow{\mathsf{close}}[b]$). We can type channel x as $(\mathbf{string} \otimes \mathsf{end}) \oplus (\mathbf{string} \otimes (\mathbf{string} \multimap \mathsf{end}) \multimap \mathsf{end})$ in both the process and the choreographic versions. The type of channel b is: $\mathsf{end} \oplus (\mathbf{string} \otimes \mathsf{end})$. For clarity, we have used concrete data types instead of the abstract basic type $\mathbf{1}$. □

[Scope/Conn/L]	$(\boldsymbol{\nu} y)\ (P \mid_x Q) \ \equiv\ (\boldsymbol{\nu} y)\ P \mid_x Q$	$\left(y \notin \mathsf{fn}(Q)\right)$
[Scope/Conn/R]	$(\boldsymbol{\nu} y)\ (P \mid_x Q) \ \equiv\ P \mid_x (\boldsymbol{\nu} y)\ Q$	$\left(y \notin \mathsf{fn}(P)\right)$
[Scope/Scope]	$(\boldsymbol{\nu} y)\ (\boldsymbol{\nu} x)\ P \ \equiv\ (\boldsymbol{\nu} x)\ (\boldsymbol{\nu} y)\ P$	
[Scope/1L]	$(\boldsymbol{\nu} x)\ \mathsf{wait}[y]; P \ \equiv\ \mathsf{wait}[y]; (\boldsymbol{\nu} x)\ P$	
[Scope/ $\otimes$ R/L], [Scope/ $\multimap$ L/L]	$(\boldsymbol{\nu} w)\ \overline{x}(y); (P \mid Q) \ \equiv\ \overline{x}(y); ((\boldsymbol{\nu} w)\ P \mid Q)$	$\left(w \notin \mathsf{fn}(Q)\right)$
[Scope/ $\otimes$ R/R], [Scope/ $\multimap$ L/R]	$(\boldsymbol{\nu} w)\ \overline{x}(y); (P \mid Q) \ \equiv\ \overline{x}(y); (P \mid (\boldsymbol{\nu} w)\ Q)$	$\left(w \notin \mathsf{fn}(P)\right)$
[Scope/ $\otimes$ L], [Scope/ $\multimap$ R]	$(\boldsymbol{\nu} w)\ x(y); P \ \equiv\ x(y); (\boldsymbol{\nu} w)\ P$	
[Scope/ $\oplus$ R_1], [Scope/&L_1]	$(\boldsymbol{\nu} w)\ x.\mathsf{inl}; P \ \equiv\ x.\mathsf{inl}; (\boldsymbol{\nu} w)\ P$	
[Scope/ $\oplus$ R_2], [Scope/&L_2]	$(\boldsymbol{\nu} w)\ x.\mathsf{inr}; P \ \equiv\ x.\mathsf{inr}; (\boldsymbol{\nu} w)\ P$	
[Scope/ $\oplus$ L], [Scope/&R]	$(\boldsymbol{\nu} w)\ x.\mathsf{case}(P, Q) \ \equiv\ x.\mathsf{case}((\boldsymbol{\nu} w)\ P\ ,\ (\boldsymbol{\nu} w)\ Q)$	
[Scope/1C]	$(\boldsymbol{\nu} w)\ \overrightarrow{\mathsf{close}}[x]; P \ \equiv\ \overrightarrow{\mathsf{close}}[x]; (\boldsymbol{\nu} w)\ P$	
[Scope/ $\otimes$ C], [Scope/ $\multimap$ C]	$(\boldsymbol{\nu} w)\ \overrightarrow{x(y)}; P \ \equiv\ \overrightarrow{x(y)}; (\boldsymbol{\nu} w)\ P$	
[Scope/ $\oplus$ C_1/L], [Scope/&C_1/L]	$(\boldsymbol{\nu} w)\ \overrightarrow{x.\mathsf{l}}\ (P, Q) \ \equiv\ \overrightarrow{x.\mathsf{l}}\ ((\boldsymbol{\nu} w)\ P, Q)$	$\left(w \notin \mathsf{fn}(Q)\right)$
[Scope/ $\oplus$ C_1/L/R], [Scope/&C_1/L/R]	$(\boldsymbol{\nu} w)\ \overrightarrow{x.\mathsf{l}}\ (P, Q) \ \equiv\ \overrightarrow{x.\mathsf{l}}\ ((\boldsymbol{\nu} w)\ P, (\boldsymbol{\nu} w)\ Q)$	$\left(w \in \mathsf{fn}(Q)\right)$
[Scope/ $\oplus$ C_2/R], [Scope/&C_2/R]	$(\boldsymbol{\nu} w)\ \overrightarrow{x.\mathsf{r}}\ (P, Q) \ \equiv\ \overrightarrow{x.\mathsf{r}}\ (P, (\boldsymbol{\nu} w)\ Q)$	$\left(w \notin \mathsf{fn}(P)\right)$
[Scope/ $\oplus$ C_2/L/R], [Scope/&C_2/L/R]	$(\boldsymbol{\nu} w)\ \overrightarrow{x.\mathsf{r}}\ (P, Q) \ \equiv\ \overrightarrow{x.\mathsf{r}}\ ((\boldsymbol{\nu} w)\ P, (\boldsymbol{\nu} w)\ Q)$	$\left(w \in \mathsf{fn}(P)\right)$

Figure 4.10: Internal Compositional Choreographies, structural equivalence ($\equiv$) rules for restriction.

4.6.3 Structural Equivalence

Since well-typed compositional choreographies are proof terms for proofs in LCL, we can refer to the commuting conversions for proofs denoted by $\equiv$ from § 4.4 to establish a notion of structural equivalence in our calculus. For example, if we annotate the proofs in the commuting conversion [Scope/Scope] in Figure 4.6 with proof terms, we get the following equivalence:

$$[\mathsf{Scope/Scope}] \qquad (\boldsymbol{\nu} y)\ (\boldsymbol{\nu} x)\ P \ \equiv\ (\boldsymbol{\nu} x)\ (\boldsymbol{\nu} y)\ P$$

Based on all the commuting conversions in LCL, we define the structural congruence $P \equiv P'$ as the smallest congruence that relates P and P' when their corresponding proofs can be transformed in one another applying a commuting conversion. We report the rules defining $\equiv$ in Figures 4.10 and 4.11, respectively corresponding to the commuting conversions in LCL for the rules Scope and Conn.

We comment some rules. In all of them, we assume all names to be different. Rule [Scope/ $\otimes$ C] allows us to swap the restriction of a name w after a communication of name y on channel x (we recall that we assume w to be different from x and y). In rule [Scope/ $\otimes$ R/R] we swap a restriction inside one of the two parallel processes after a sending action, provided that the restricted channel is not a free name in the process it is pushed into. Rule [Scope/Scope] states that two restrictions can always be swapped with one another. All the other rules follow similar reasonings.

Since structural congruence is defined by our commuting conversions in LCL, which preserve the concluding judgement of a proof, we immediately derive the following result:

[Conn/Conn]	$(P \mid_y Q) \mid_x R \equiv P \mid_y (Q \mid_x R)$	
[Conn/1L/L]	$\mathsf{wait}[x]; P \mid_y Q \equiv \mathsf{wait}[x]; (P \mid_y Q)$	
[Conn/1L/R]	$P \mid_y \mathsf{wait}[x]; Q \equiv \mathsf{wait}[x]; (P \mid_y Q)$	
[Conn/ $\otimes$R/R/L], [Conn/$\multimap$L/R/L]	$P \mid_w \overline{x}(y); (Q \mid R) \equiv \overline{x}(y); ((P \mid_w Q) \mid R)$	
[Conn/ $\otimes$R/R/R], [Conn/$\multimap$L/R/R]	$P \mid_w \overline{x}(y); (Q \mid R) \equiv \overline{x}(y); (Q \mid (P \mid_w R))$	
[Conn/ $\otimes$ L/L]	$x(y); P \mid_w Q \equiv x(y); (P \mid_w Q)$	
[Conn/ $\otimes$ L/R], [Conn/ $\multimap$ R/R]	$P \mid_w x(y); Q \equiv x(y); (P \mid_w Q)$	
[Conn/ $\multimap$ L/L/R]	$\overline{x}(y); (P \mid Q) \mid_w R \equiv \overline{x}(y); (P \mid (Q \mid_w R))$	
[Conn/ $\oplus$ R_1/R], [Conn/&L_1/R]	$P \mid_w x.\mathsf{inl}; Q \equiv x.\mathsf{inl}; (P \mid_w Q)$	
[Conn/ $\oplus$ R_2/R], [Conn/&L_2/R]	$P \mid_w x.\mathsf{inr}; Q \equiv x.\mathsf{inr}; (P \mid_w Q)$	
[Conn/ $\oplus$ L/L]	$x.\mathsf{case}(P,Q)\mid_w R \equiv x.\mathsf{case}((P\mid_w R),(Q\mid_w R))$	
[Conn/ $\oplus$ L/R], [Conn/&R/R]	$P\mid_w x.\mathsf{case}(Q,R) \equiv x.\mathsf{case}((P\mid_w Q),(P\mid_w R))$	
[Conn/&L_1/L]	$x.\mathsf{inl}; P \mid_w Q \equiv x.\mathsf{inl}; (P \mid_w Q)$	
[Conn/&L_2/L]	$x.\mathsf{inr}; P \mid_w Q \equiv x.\mathsf{inr}; (P \mid_w Q)$	
[Conn/1C/L]	$\overrightarrow{\mathsf{close}}[x]; P \mid_w Q \equiv \overrightarrow{\mathsf{close}}[x]; (P \mid_w Q)$	
[Conn/1C/R]	$P \mid_w \overrightarrow{\mathsf{close}}[x]; Q \equiv \overrightarrow{\mathsf{close}}[x]; (P \mid_w Q)$	
[Conn/$\otimes$ C/L], [Conn/ $\multimap$C/L]	$\overrightarrow{x(y)}; P \mid_w Q \equiv \overrightarrow{x(y)}; (P \mid_w Q)$	$\left(y \notin \mathsf{fn}(Q)\right)$
[Conn/$\otimes$C/R], [Conn/ $\multimap$ C/R]	$P \mid_w \overrightarrow{x(y)}; Q \equiv \overrightarrow{x(y)}; (P \mid_w Q)$	$\left(y \notin \mathsf{fn}(P)\right)$
[Conn/$\oplus$ C_1/L]	$\overrightarrow{x.\mathsf{l}}\ (P,Q)\mid_w R \equiv \overrightarrow{x.\mathsf{l}}\ ((P\mid_w R),(Q\mid_w R))$	$\left(w \in \mathsf{fn}(P) \cap \mathsf{fn}(Q)\right)$
[Conn/$\oplus$ C_1/R], [Conn/&C_1/R]	$P\mid_w \overrightarrow{x.\mathsf{l}}\ (Q,R) \equiv \overrightarrow{x.\mathsf{l}}\ ((P\mid_w Q),(P\mid_w R))$	$\left(w \in \mathsf{fn}(Q) \cap \mathsf{fn}(R)\right)$
[Conn/$\oplus$$\mathsf{C}_1$/R/L],[Conn/&$\mathsf{C}_1$/R/L]	$P\mid_w \overrightarrow{x.\mathsf{l}}\ (Q,R) \equiv \overrightarrow{x.\mathsf{l}}\ ((P \mid_w Q)\ , R)$	$\left(w \in \mathsf{fn}(Q), w \notin \mathsf{fn}(R)\right)$
[Conn/$\oplus$ C_2/L]	$\overrightarrow{x.\mathsf{r}}\ (P,Q)\mid_w R \equiv \overrightarrow{x.\mathsf{r}}\ ((P\mid_w R),(Q\mid_w R))$	$\left(w \in \mathsf{fn}(P) \cap \mathsf{fn}(Q)\right)$
[Conn/$\oplus$ C_2/R], [Conn/&C_2/R]	$P\mid_w \overrightarrow{x.\mathsf{r}}\ (Q,R) \equiv \overrightarrow{x.\mathsf{r}}\ ((P\mid_w Q),(P\mid_w R))$	$\left(w \in \mathsf{fn}(Q) \cap \mathsf{fn}(R)\right)$
[Conn/$\oplus$$\mathsf{C}_2$/R/R],[Conn/&$\mathsf{C}_2$/R/L]	$P\mid_w \overrightarrow{x.\mathsf{r}}\ (Q,R) \equiv \overrightarrow{x.\mathsf{r}}\ (Q\ , (P \mid_w R))$	$\left(w \notin \mathsf{fn}(Q), w \in \mathsf{fn}(R)\right)$
[Conn/&C_1/L]	$\overrightarrow{x.\mathsf{l}}\ (P,Q) \mid_w R \equiv \overrightarrow{x.\mathsf{l}}\ ((P \mid_w R),\ Q)$	$\left(w \in \mathsf{fn}(P), w \notin \mathsf{fn}(Q)\right)$
[Conn/&C_2/L]	$\overrightarrow{x.\mathsf{r}}\ (P,Q) \mid_w R \equiv \overrightarrow{x.\mathsf{r}}\ (P\ , (Q \mid_w R))$	$\left(w \notin \mathsf{fn}(P), w \in \mathsf{fn}(Q)\right)$

Figure 4.11: Internal Compositional Choreographies, structural equivalence ($\equiv$) rules for parallel composition.

Theorem 4.6.3 (Subject Congruence)**.** *$P \triangleright \Psi$ and $P \equiv Q$ implies that $Q \triangleright \Psi$.*

4.6.4 Reduction Semantics

We get the semantics of well-typed ICC terms directly by Curry-Howard correspondence with β-reductions in LCL. Figure 4.12 represents the β-reductions on proofs from Figure 4.5 as ICC term reductions; the reduction relation on ICC terms $P \xrightarrow{t} P'$ is closed under term contexts. In the process fragment, when the redex channel is x and it has type $\mathbf{1}$, then we just erase the restriction on x together with the parallel composition on x of the inactive process with some other process P. In the choreographic fragment, this corresponds to just removing the restriction on x.

When x has type $A \otimes B$, then a new session of type A is created. In the process fragment, we obtain $(\boldsymbol{\nu} x)\,(\boldsymbol{\nu} y)\,\big(Q \mid_x (P \mid_y R)\big)$, showing that the continuation of the input R is now linked to the process P through the freshly created channel y and to Q through x.

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

Figure 4.12: Internal Compositional Choreographies, reductions.

In the choreographic fragment, we abstract from the way processes are linked, only seeing that the new restriction $(\boldsymbol{\nu} y)$ is introduced. The cases for $\multimap$ and branching are similar.

Directly from the results in § 4.4, we obtain:

Theorem 4.6.4 (Subject Reduction)**.** *If $P \triangleright \Psi$ and $P \xrightarrow{t} Q$ then $Q \triangleright \Psi$.*

Additionally, well-typed terms with restrictions never get stuck:

Theorem 4.6.5 (Progress)**.** *If $P \triangleright \Psi$ and P contains restrictions then there exists Q such that $P \xrightarrow{t} Q$.*

4.6.5 Endpoint Projection and Choreography Extraction

By using abstraction and concretisation from § 4.4, we obtain Choreography Extraction and Endpoint Projection respectively. Figure 4.13 shows the corresponding rules, redefining the relations for abstraction $\dashrightarrow$ and for concretisation $\cdots\!\!\rightarrow$ directly on processes. Note how a choreography P corresponds to the parallel composition of two processes connected through a channel x.

Example 4.6.6. Using the rules in Figure 4.13 and the structural congruence $\equiv$, we can now transform $P \mid_x Q$ to R in Example 4.6.2 and viceversa.

In the sequel, as done for proofs, we denote with $\twoheadrightarrow$, $\dashrightarrow\!\!\!\!\twoheadrightarrow$, and $\cdots\!\!\rightarrow\!\!\!\!\!\rightarrow$ the transitive closure, up-to $\equiv$, of $\rightarrow$, $\dashrightarrow$, and $\cdots\!\!\rightarrow$ respectively. Finally, we can state the main theorem of this section:

Theorem 4.6.7 (Correspondence)**.** *Let P be well typed. Then,*

$$
\begin{array}{lll}
[\alpha\gamma_{\mathbf{1}}] & \mathsf{close}[x] \mid_x \mathsf{wait}[x]; P & \overset{x}{\underset{\leftarrow}{\dashrightarrow}} \overrightarrow{\mathsf{close}[x]}; P \\
[\alpha\gamma_{\otimes}] & \overline{x}(y); (P \mid Q) \mid_x x(y); R & \overset{x}{\underset{\leftarrow}{\dashrightarrow}} \overrightarrow{x(y)}; \big(P \mid_y (Q \mid_x R)\big) \\
[\alpha\gamma_{\multimap}] & x(y); P \mid_x \overline{x}(y); (Q \mid R) & \overset{x}{\underset{\leftarrow}{\dashrightarrow}} \overrightarrow{x(y)}; \big((Q \mid_y P) \mid_x R\big) \\
[\alpha\gamma_{\&_1}] & x.\mathsf{case}(P,Q) \mid_x x.\mathsf{inl}; R & \overset{x}{\underset{\leftarrow}{\dashrightarrow}} \overrightarrow{x.\mathsf{l}}\ ((P \mid_x R), Q) \\
[\alpha\gamma_{\&_2}] & x.\mathsf{case}(P,Q) \mid_x x.\mathsf{inr}; R & \overset{x}{\underset{\leftarrow}{\dashrightarrow}} \overrightarrow{x.\mathsf{r}}\ (P\ ,\ Q \mid_x R) \\
[\alpha\gamma_{\oplus_1}] & x.\mathsf{inl}; P \mid_x x.\mathsf{case}(Q,R) & \overset{x}{\underset{\leftarrow}{\dashrightarrow}} \overrightarrow{x.\mathsf{l}}\ ((P \mid_x Q)\ ,\ R) \\
[\alpha\gamma_{\oplus_2}] & x.\mathsf{inr}; P \mid_x x.\mathsf{case}(Q,R) & \overset{x}{\underset{\leftarrow}{\dashrightarrow}} \overrightarrow{x.\mathsf{r}}\ (Q\ ,\ (P \mid_x R))
\end{array}
$$

Figure 4.13: Internal Compositional Choreographies, endpoint projection and choreography extraction.

- *(Choreography Extraction)* $P \overset{\tilde{x}}{\longrightarrow\!\!\!\!\rightarrow} P'$*, with* P' *restriction-free, implies* $P \overset{\tilde{x}}{\dashrightarrow\!\!\!\!\rightarrow} Q$ *such that* $Q \overset{\bullet\tilde{x}}{\longrightarrow\!\!\!\!\rightarrow} P'$.
- *(Endpoint Projection)* $P \overset{\bullet\tilde{x}}{\longrightarrow\!\!\!\!\rightarrow} P'$*, with* P' *restriction-free, implies* $P \overset{\tilde{x}}{\cdots\!\!\!\!\rightarrow} Q$ *such that* $Q \overset{\tilde{x}}{\longrightarrow\!\!\!\!\rightarrow} P'$.

Proof. Follows immediately by Theorem 4.4.3, since choreographies are proof terms. □

4.7 Related Work

The developments in this Chapter start from the work [27], where Caires and Pfenning introduce a Curry-Howard correspondence between the π-calculus and Intuitionistic Linear Logic (ILL). The typable process fragment of our compositional choreographies corresponds to the typable processes in [27]. The only key difference is that, since we split the Cut rule into Conn and Scope, we are now able to separate the parallel operator from restriction, yielding a bigger number of processes. However, the extra processes, if reducible, are always convertible to those where a Conn is immediately followed by a Scope, hence equivalent to those in [27].

Based on the developments in [27], Wadler introduces a Curry-Howard correspondence between the π-calculus and Classical Linear Logic (CLL) [103]. We conjecture that the concepts that we used to develop LCL from ILL can also be adopted in the classical setting; our proofs should be reformulated in the new setting.

Previous works have tackled the problem of defining a formal model for choreographies and giving a correct transformation from choreographies to processes using process calculi [31]. The main contributions introduced by our work in this regard are that our transformations are *invertible* and *complete*, i.e., for all well-typed terms it is always possible to transform choreographies into processes and vice versa. Our commuting conversions can be seen as a logical characterisation of the *swapping* relation in Chapter 3, which permutes independent communications in a choreography.

In general, transforming processes in the choreographies they implement is a known hard problem [18], and our work is the first giving an elegant solution. In this work we have not considered many interesting features supported by [31] and in our Chapters 2 and 3, such as multiparty sessions, asynchrony, and replicated services. We plan to introduce them as future work. Probably, the work closest to ours in providing choreographic synthesis from endpoint processes is the one in [63]. However, the authors synthesis choreography-like types from endpoint session types, whereas we synthesis programs. Furthermore, they do not provide a unified model in that their approach does not capture compositional choreographies.

Our mixing of choreography terms with process terms is similar to that found in [17] for global protocols. The main difference wrt [17] is that we support the abstraction and concretisation of program terms, whereas [17] handles the simpler setting of abstraction and concretisation of protocols; handling program terms is more complicated, since we have to deal with name passing and session interleaving, both nontrivial problems in the context of session types [21].

With respect to Chapter 3, the main difference is in the typing system. Specifically, the type system in Chapter 3 does not employ hypersequents and therefore loses information on where the endpoints of connections are actually located. This information makes Choreography Extraction possible in our setting.

4.8 Discussion

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

Exponentials. Our work focuses on the multiplicative and additive fragments of linear logic, but we have designed LCL to extend it with exponentials in the future. In particular, we suspect that it is possible to split the known cut-rule for exponentials

$$\frac{\Gamma;\cdot \vdash A \quad \Gamma, A; \Delta \vdash C}{\Gamma; \Delta \vdash C}\ \mathsf{Cut!}$$

into a connection rule and a scope rule such as:

$$\frac{\triangleright\, \Psi_1 \mid \Gamma;\cdot \vdash x{:}A \quad \triangleright\, \Psi_2 \mid \Gamma, x{:}A; \Delta \vdash T}{\triangleright\, \Psi_1 \mid \Psi_2 \mid \Gamma;\cdot \vdash x{:}\bullet A \mid \Gamma, x{:}\bullet A; \Delta \vdash T}\ \mathsf{Conn!}$$

$$\frac{\triangleright\, \Psi \mid \Gamma;\cdot \vdash x{:}\bullet A \mid \Gamma, x{:}\bullet A; \Delta \vdash T}{\triangleright\, \Psi \mid \Gamma; \Delta \vdash T}\ \mathsf{Scope!}$$

We believe that the development presented in this Chapter generalises to exponentials as well. However, variable names, or an equivalent notation, would become mandatory in the proof theory because otherwise connections between unrestricted and linear resources may be confused, breaking our Correspondence Theorem. We leave a detailed investigation of this fragment to future work.

Variables. The variable assignments in our typing discipline for choreographies are inspired by [27] and is somehow impure wrt linearity. For example, in the typing rule $\otimes$R the variable x associated to $A \otimes B$ is also associated to B in the premise. Another

possibility would be to follow Bellin-Scott's Curry-Howard [19], associating to $A \otimes B$ a fresh variable that is different from those assigned to A and B in the premises. While this choice would influence our proof terms in § 4.6, it would not alter our results in any way. We have chosen to follow the assignments in [27] since they give a more natural correspondence with session types for process calculi, where the type of a channel tells how the channel will be used later on during execution, implying that the same channel name should be reused for the continuation of a type just like we do in our rules.

ILL. Any proof in ILL can be easily transformed into a proof in our logic. In fact, we can simply carry over any rule application in our resource fragment directly, and represent every instance of the Cut rule with consecutive applications of rules Conn and Scope. As a consequence of this transformation, it follows that (i) proofs in linear logic are special cases of γ-normal proofs in LCL, (ii) our proof reductions are a conservative extension of the cut reductions of linear logic, and (iii) our abstraction result applies to linear logic. In particular, (iii) means that it is always possible to reconstruct the choreography implemented by some processes typed using linear logic, simply by carrying over the proof to LCL.

Semantics. Due to the commuting conversions supported by LCL, our language supports term equivalences that are not included in the π-calculus, e.g., $[\mathsf{Conn}/ \multimap \mathsf{R}/\mathsf{R}]$ in Figure 4.11. This aspect has already been noted in [103]; it is not a problem in our setting because our process fragment is exactly the same as presented in [85], where the authors prove that in well-typed processes the extra equivalences do not produce new reductions. However, another possibility would be to state our results using only the commuting conversions supported by the normal π-calculus, as done in [27]. This however would add to the complexity of our theory and we chose not to do so for presentational reasons.

Multiparty Sessions. Another interesting extension to our framework would be to generalise our binary sessions in ICC to multiparty sessions in the spirit of Chapters 2 and 3. Being able to model multiparty session types corresponds to being able to identify multiparty sessions, where each role in the session is identified by some local type [55]. We conjecture that this can be achieved by modifying rule Scope into a MultiScope rule, so that it can execute more than one connection at the same time. In this way, each application of rule MultiScope would identify a multiparty session. We leave this as interesting future work.

4.9 Conclusions

We introduced LCL, a conservative extension of intuitionistic linear logic to hypersequents that express connections between different sequents. LCL includes a new set of rules that can manipulate connections; we showed that any judgement provable with the set of rules corresponding to linear logic in LCL can also be proven with our new rules and vice versa. Moreover, the new system is sound wrt the cut elimination process, represented by our β-reductions. Finally, we showed that LCL is isomorphic to an extension of the π-calculus, called internal compositional choreographies, and, in particular, choreographic behaviour corresponds to the new set of rules for manipulating connections. We can now use internal

compositional choreographies to safely apply the methodology of round-trip development, where systems can be designed by using both a choreographic view and a process view.

Part II

Implementation

CHAPTER 5

Programming Sessions with Correlation Sets

5.1 Introduction

In Part I, we have focused on the formalisation of models for Choreographic Programming, which are all based upon the fundamental concept of *session*. In this Chapter and the next, we investigate a methodology for implementing sessions that we will finally use in the development of a framework for Choreographic Programming in Chapter 7.

A session is an abstract mechanism for matching sending and receiving actions performed by processes that wish to exchange a message. In the real world, such processes are typically executed inside some network nodes; therefore, whenever a process wishes to send a message to another process over a network, the sender must provide enough information in the message to (i) identify the destination network node and (ii) identify the receiver process running inside the destination network node.

Usually, routing information in messages is handled by a middleware layer. For example, two processes can establish a TCP/IP socket connection to reliably exchange messages with each other [24]; a socket API will allow each process to perform sending and receiving actions on the socket, dealing internally with the details of adding the proper message headers for routing each message to its destination (e.g., a TCP port). This pattern appears repeatedly in different implementations of sessions, e.g., in middleware for Inter-Process Communications (IPC) or web sessions [2, 86, 82]. The main differences among these implementations come from using different portions of data in messages as information for their routing; e.g., a web-oriented middleware may require matching the value of an HTTP cookie [43] with the value of a variable of a process running in a web server, whereas an IPC middleware may use unique object identifiers. In other words, there is a common conceptual ground between different implementations of sessions. Inspired by this observation, we ask:

> *Can we design a general programming framework for sessions, in which the data used for routing messages can be specified by the programmer?*

More specifically, we are interested in obtaining a framework that can be adapted to different application domains, ranging, e.g., from binary sessions between a web browser and a web server to more complex multiparty sessions supporting distributed transactions in other settings.

We start our investigation from the mechanism *of correlation sets*, introduced by WS-BPEL [79] (BPEL for short), the reference orchestration language for Web Services [12]. Correlation sets are program parameters used to define routing policies for delivering incoming messages to the correct running process within a server (also called service). A

message is relayed to an internal process whenever a part of its data content matches a part of the process variable state. These parts are defined programmatically by the correlation sets. Correlation sets are widely used in Service-Oriented Computing (SOC); their role resembles that of unique keys in relational databases: they uniquely identify a process from a portion of their data.

Correlation sets can be seen as a parameterised mechanism for implementing sessions, where the parameters instruct which portions of data to use in messages for supporting sessions; therefore, they may be seen as a solid candidate for answering our question. However, the programming of correlation sets is error-prone in currently available tools. For example, a programmer may declare his intention to use a single session identifier variable *sid* to identify a session (by defining a correlation set), but then forget to actually instantiate such variable in the program; this error would immediately lead to a deadlock, since the session would then be unable to receive messages.

In this Chapter, we develop a language for programming correlation-based sessions. Our main contribution is the formalisation of a semantics for correlation-based message routing, and the development of a typing discipline for avoiding bad correlation programming.

5.1.1 Contributions

This Chapter provides the following main contributions.

Correlation-based Programming. We formalise a language model for programming sessions by using correlation sets (§ 5.3), where processes run inside services that can be composed in networks and communicate and each correlation set defines a session through which a process can be reached. The main aspect of our model is the usage of operation names and message paths (i.e., paths pointing to content in messages) in the definitions of correlation sets. Our model supports (i) a semantics for *asynchronous* communications between processes that execute inside services (which represent network nodes), and (ii) a notion of correlation *aliasing* which allows programmers to decouple the routing data in messages from that used to distinguish processes. Both features are based on our structure for correlation set definitions.

Typing. We define a typing discipline that prevents the occurrence of some runtime errors due to bad correlation programming (§ 5.4). We guarantee that, in a well-typed network, processes can always be uniquely identified by using correlation data. Our results show how to discipline message routing programming based solely on data for obtaining a determinism similar to that of π-calculus channels [69].

Implementation and Evaluation. We provide an implementation of our model by extending the interpreter of the Jolie language [74, 60] (§ 5.5). We use our implementation to evaluate our approach, by programming a nontrivial real-world example showing a fully-functional distributed user authentication system based on multiparty sessions, inspired by the OpenID Authentication specifications [81] (§ 5.6).

5.2 Preview

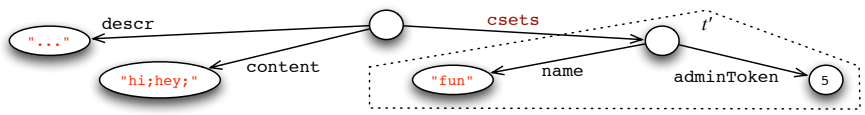
In this section, we outline with an example the main ideas of our language. Formal syntax and semantics will be given in § 5.3.

Our example is a common distributed scenario with a chat service supporting the management of chat rooms. Chat rooms are identified by name, as in IRC servers [5]. The service allows users to: create new chat rooms, publish a message in a chat room, retrieve published messages from existing chat rooms, and close chat rooms. When a client requests the creation of a chat room, the service checks that no other room with the same name exists. It then sends an *administration token* back to the invoker. Each chat room has two sessions. The first session is identified by the chat room name and can be used to publish messages in the room or retrieve the history of messages published so far. The second session is identified by the administrator token and is intended to be used by the initial creator whenever she wishes to close the room.

In our example, we implement each chat room with a separate process running inside our chat service.

5.2.1 Data Structures

Each process implementing a chat room has a data structure representing its local state where its name, description, published messages, and administration token are stored. In our language, we represent data as trees where nodes are values of basic data types such as strings and integers. For instance, the state of a chat room is represented by the tree:



(1)

The root has three children pointed to by labels `descr`, `content` and `csets`. Subnode `csets` has two other children, `name` and `adminToken`. Data trees are accessed in programs by means of *paths*. Paths are sequences of edge names separated by dots, and can be used for traversing a tree starting from its root. Paths can be used in assignments and expressions. For example, the tree above could be initialised in our language with the following assignments:

```
descr = "..."; content = "hi;hey;";
csets.name = "fun"; csets.adminToken = 5
```

For brevity, we refer to a path as a variable, and the node it points to as its value. So in this case variable `content` would have value `"hi;hey;"`.

5.2.2 Communication Behaviour

In our language, data is exchanged between processes that run inside services by means of message passing. As in Web Services, messages are labelled by *operations*. Given operations `create`, `publish`, `read` and `close`, we could program the chat service behaviour as:

```
create(name)(csets.adminToken) { csets.adminToken = new };
run = 1; while( run ) {
    [publish(msg)] { content = content + msg.content + ";" }
    [read(req)(content) { 0 }] { 0 }
    [close(req)] { run = 0 }
}
```

The first instruction is an input on operation `create`. The content of the received message (a data tree) will be stored as a subtree of `name` which is a path in the local state. We call this input instruction a process start since its execution will start a new chat romm. Moreover, it is also a Request-Response (as in WSDL [13]): the client will wait for the server to reply with the content of `csets.adminToken` that is sent back once the local code in curly brackets `{ csets.adminToken = new }` is executed. **new** is a primitive that returns a locally-fresh token. After invocation, the process enters a loop containing a choice of three inputs with operations `publish` (for publishing in the chat room), `read` (for reading already published messages), and `close` (for closing the chat room). The inputs with operations `publish` and `close` are standard inputs called One-Way while the one with operation `read` is a Request-Response.

Dually to the server, we can give an example program for a client:

```
roomName = "MyRoom"; create@Chat(roomName)(adminToken);
msg1.roomName = roomName; msg1.content = "hi";
msg2.roomName = roomName; msg2.content = "hey";
{ publish@Chat(msg1) | publish@Chat(msg2) };
read@Chat(roomName)(chatContent); close@Chat(adminToken)
```

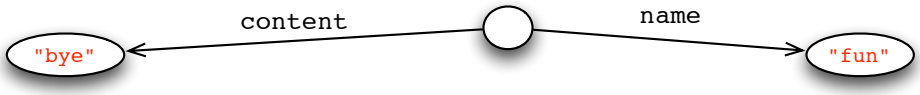
This client example performs a Solicit-Response output (dual of Request-Response) on operation `create`. The message is sent at location `Chat`, the location of the chat server. Locations (cf. URIs) define where services are deployed, modeling locality. The instruction is completed when the response from the server is received and assigned to `adminToken`. Thereafter, the client sends messages to the chat with two Notification outputs (dual to One-Way) executed in parallel by means of the **|** operator. Finally, the client reads the content of the chat room through operation `read` and closes it by means of operation `close`.

In our language, messages are delivered asynchronously to processes. After a message is sent, it is guaranteed that the receiving process has buffered it, but not that it has consumed it. This can lead to bad behaviour. For this reason, the semantics of our language in § 5.3 preserves ordering of buffered messages.

5.2.3 Correlation Sets and Aliasing

The chat service may have many running processes executing in parallel, each one representing a chat room. Moreover, each process needs a way for safely identifying messages coming from the room creator, since the room creator is the only one authorised to close the chat room. Correlation sets address this kind of issues. In our language, a correlation set represents a session and is a set of paths, called correlation variables, that define which nodes of a session state identify the session. A correlation set is defined by means of the keyword **cset**. Our chat service has two correlation sets: **cset** `{name}` and

cset {adminToken}. The first implements a session for reading and publishing messages, and the second implements a session with the room creator. For example, if the chat server receives a message carrying the tree:



the first correlation set will then associate the message to the process running with the state shown in (1), since both message and session share the same value for correlation variable name, and route the message to it. We call this association *correlation*, and we say that the message *correlates* with the process. The value for correlation variable name is stored in the subtree **csets** in the session state. More generally, in our language every correlation value must be put in that subtree. This makes modifications to data that influences correlation explicit. We exploit this aspect in the definition of our type system, in § 5.4.

Correlation sets are specified by the receiver: the client does not need to be aware of the correlation sets of the invoked service but needs only to send messages with the expected data structures, enabling loose coupling. Correlation sets are also suitable for the integration with different technologies. For example, a web server session can be identified by the correlation set $c = \{\texttt{sid}\}$, the session identifier usually stored in a browser cookie[1].

Above, we associated a message to a process by matching the value of the same path name in the message tree and the process state. Such a mechanism is limiting, because the fact that the two paths must be the same means that there is tight coupling between the service implementation and its interface. This could be even completely unfeasible. Consider, for instance, the case in which a programmer must write a service that interacts with a legacy application. The interface of the service will have to be in accordance to what the legacy application expects. Let us assume now that the legacy application will send two different kinds of message to our new chat service, on different operations. The first contains the room name under path roomName and the other in the root of the message data tree; this is the behaviour of the client that we showed before. How can we relate both values to the same correlation variable inside the process implementing the desired chat room? We address this issue with a notion of aliasing: a correlation variable is defined together with a list of aliases that tell where to retrieve, in a message, the value to be compared with that of the process state, depending on the type of the incoming message (aliasing can be looked at as a type itself). Hence, the correlation set definitions for the chat service become:

```
cset { name: Create Publish.roomName Read }
cset { adminToken: Close }
```

where, for brevity, we assume the input message type of each operation has the same name with an uppercase initial. Data types will be presented in § 5.5.

[1] We will develop further this idea in Chapter 6.

5.3 Model

In this section we introduce our language model, the Correlation Calculus, and formalise its data, syntax, and semantics.

5.3.1 Data Trees and Correlation

Let t range over a set of *data trees* $\mathcal{T}$, with edges denoted by $\mathtt{x}$,$\mathtt{y}$,$\mathtt{z}$,... and nodes denoted by v. v is a value, which can be a string, an integer, a *location* or the undefined value $v_\perp$. Values is the set of all values. In programs, data trees are accessed by *paths*. A path $\mathtt{p}$ is a sequence of tree edges $\mathtt{x}_1.\ \ldots\ .\mathtt{x}_n$ denoting an endofunction on data trees defined as:

$$\mathtt{p}(t) = \begin{cases} t & \text{if } \mathtt{p} = \epsilon \\ \mathtt{p}'(t') & \text{if } \mathtt{p} = \mathtt{x}.\mathtt{p}' \text{ and } \mathtt{x} \text{ is an edge from the root of } t \text{ to } t\text{'s subtree } t' \\ t_\perp & \text{if } \mathtt{p} = \mathtt{x}.\mathtt{p}' \text{ and there is no edge } \mathtt{x} \text{ from } t \text{ to a subtree } t' \end{cases}$$

where ϵ denotes the empty sequence and $t_\perp$ a tree with a single node with value $v_\perp$. We denote the set of possible paths with Paths. Furthermore, we require paths written in programs to be nonempty. We extract the value of the root of a tree by using the function $\langle\!\langle\rangle\!\rangle : \mathcal{T} \to \mathsf{Values}$.

Definition 5.3.1 (Correlation Set)**.** A correlation set, denoted by c, is a set of paths corresponding to those values that identify a running session of a service: $c \subseteq \mathsf{Paths}$. A service may define more than one correlation set: we denote with C a set of correlation sets, $C \subseteq \mathcal{P}(\mathsf{Paths})$.

We model correlation aliasing by means of an *aliasing function*, α_C, which establishes where to retrieve correlation values in a message received for an operation. Let $\mathcal{O}$ be the set of possible operations, ranged over by $\mathtt{o}$. An aliasing α_C is a function that given an operation $\mathtt{o}$ returns a correlation set $c \in C$ and a function from paths contained in c to paths in the incoming message:

$$\alpha_C : \mathcal{O} \rightharpoonup C \times (\mathsf{Paths} \rightharpoonup \mathsf{Paths})$$

The aliasing function α_C bases aliases on operations, and not on message types like in § 5.2. This is just a matter of technical convenience; in our language implementation (§ 5.5), aliases are defined on message types and are internally converted to an aliasing function as described in this section.

We now present our definition of correlation in terms of the relation $\vdash_{\alpha_C}$:

Definition 5.3.2 (Correlation $\vdash$)**.** A data tree t' received for operation $\mathtt{o}$ *correlates* with a data tree t with respect to an aliasing α_C, written $t', \mathtt{o} \vdash_{\alpha_C} t$, whenever

$$\exists c, f.\ c \neq \emptyset\ \wedge\ \alpha_C(\mathtt{o}) = (c, f)\ \wedge\ \forall \mathtt{p} \in c.\ \langle\!\langle (f(\mathtt{p}))(t') \rangle\!\rangle = \langle\!\langle \mathtt{csets}.\mathtt{p}(t) \rangle\!\rangle \neq v_\perp$$

5.3.2 Syntax

The syntax and semantics of our model are structured in three layers. The *behavioural layer* models the actions performed by processes, the *service layer* handles the definition of correlation sets, running processes, and session instantiation, and the *network layer* deals with the deployment of services and their communications. This layering, which originally comes from [51], is reflected in our language implementation presented in § 5.5.

$$
\begin{array}{lcll}
B & ::= & \sum_i \texttt{[}\eta_i\texttt{]\{}B_i\texttt{\}} & \textit{(input choice)} \\
 & | & \eta & \textit{(input)} \\
 & | & \overline{\eta} & \textit{(output)} \\
 & | & \texttt{if(}e\texttt{)\{}B_1\texttt{\}else\{}B_2\texttt{\}} & \textit{(cond)} \\
 & | & \texttt{while(}e\texttt{)\{}B\texttt{\}} & \textit{(loop)} \\
 & | & \texttt{p=}e & \textit{(assign)} \\
 & | & B_1; B_2 & \textit{(seq)} \\
 & | & B_1 \mid B_2 & \textit{(par)} \\
 & | & \mathbf{0} & \textit{(inact)} \\
\hline
\eta & ::= & \texttt{o(p)} & \textit{(one-way)} \\
 & | & \texttt{o(p)(p')\{}B\texttt{\}} & \textit{(request-response)} \\
\overline{\eta} & ::= & \texttt{o@p(p')} & \textit{(notification)} \\
 & | & \texttt{o@p(p')(p'')} & \textit{(solicit-response)} \\
e & ::= & \texttt{new} & \textit{(new)} \\
 & | & l & \textit{(location)} \\
 & | & \texttt{p} & \textit{(path)} \\
 & | & \ldots & \textit{(first-order expr)}
\end{array}
$$

Figure 5.1: Correlation Calculus, syntax of behaviours.

5.3.2.1 Process layer

Behavioural terms, ranged over by B, and defined by the grammar reported in Figure 5.1 where r denotes a *channel name*, $l, l', \ldots$ *locations* and $e, e', \ldots$ unspecified first-order expressions that include locations, paths, and an operator `new` for generating locally-fresh values. Input-guarded branching is available through (choice). Communications can be unidirectional (one-way) or bidirectional (request-response). `o(p)` reads an incoming message for operation `o` and places the received tree in the local state tree under path `p`. Dually, `o@p(p')` sends a message for operation `o` to the location stored in the state node `p` points to, carrying the data in the local state pointed by `p'`. Alternatively, (request-response) and (solicit-response) allow for Request-Response communications. In particular, in `o(p)(p'){`B`}`, the value at `p'` will be returned after B is executed. All other constructs are standard.

5.3.2.2 Service layer

The syntax of services, denoted by S, is reported in Figure 5.2. A service consists of a behaviour definition that and an aliasing α_C, defining its correlation sets. Term *(service)* models a typical replicated service offering the always available operations in η_i for starting processes that will execute behaviour B_i, respectively. Normally, services become active only after they are invoked. For this reason, a system needs at least one service to spon-

$$S \quad ::= \quad \sum_i \, [\,\eta_i\,]\,\{\,B_i\,\} \triangleright_{\alpha_C} \mathbf{0} \qquad \textit{(service)} \qquad | \qquad \mathbf{0} \triangleright_{\alpha_C} B \cdot t_\perp \cdot \epsilon \qquad \textit{(starter)}$$

Figure 5.2: Correlation Calculus, syntax of services.

$$N, M ::= \quad [S]_l \quad | \quad (\boldsymbol{\nu} r)\, N \quad | \quad N \mid N \quad | \quad \mathbf{0} \qquad \textit{(network)}$$

Figure 5.3: Correlation Calculus, syntax of networks.

taneously start invoking other services. Term *(starter)* captures such services. A starter specifies a single process executing behaviour B, which will start its execution without the need to be triggered.

5.3.2.3 Network layer

Services are *deployed* on locations and composed in parallel to form networks, denoted by $N, M, \ldots$, using the syntax in Figure 5.3. We assume that, for any network, it is never the case that two services are deployed with the same location.

5.3.3 Semantics

We extend the language syntax with runtime terms to model execution. The terms are given in Figure 5.4. Services are extended to support multiple locally running processes (denoted by P). Each process consists of a runtime behaviour, a state t, and a FIFO message queue $\tilde{m}$, with ϵ representing the empty queue. m is a message of the form $(r, \mathtt{o}, t)$ where r is a channel, $\mathtt{o}$ an operation and t the message content. The terms $\mathsf{Wait}(r, \mathtt{p})$ and $\mathsf{Exec}(r, \mathtt{p}, B)$ model runtime Request-Response communications, where channel r is used to communicate a response.

We equip our model with a structural congruence $\equiv$, defined as the smallest congruence relation on B, P, S, and N such that $(\ |\ , \mathbf{0})$ is a commutative monoid, it supports alpha-conversion, $\mathbf{0}; B \equiv B$, $B \equiv B'$ and $P \equiv P'$ imply $[B \triangleright_{\alpha_C} P]_l \equiv [B' \triangleright_{\alpha_C} P']_l$, $(\boldsymbol{\nu} r)(\boldsymbol{\nu} r)' N \equiv (\boldsymbol{\nu} r)'(\boldsymbol{\nu} r) N$ and such that $((\boldsymbol{\nu} r) N) \mid N' \equiv (\boldsymbol{\nu} r)(N \mid N')$ if $r \notin \mathsf{cn}(N')$, where cn is a function that returns the set of channel names in a term.

We give semantics to terms with a labelled transition system (lts), in which labels are ranged over by μ. We describe the semantics of each layer separately.

$$\begin{array}{rcll} S & ::= & B \triangleright_{\alpha_C} P & \textit{(running service)} \\ P, Q & ::= & B \cdot t \cdot \tilde{m} \quad | \quad P \mid Q & \textit{(running processes)} \\ B & ::= & \ldots \mid \mathsf{Wait}(r, \mathtt{p}) \mid \mathsf{Exec}(r, \mathtt{p}, B) & \textit{(running behaviours)} \end{array}$$

Figure 5.4: Correlation Calculus, syntax of runtime terms.

$$\frac{}{\texttt{o@p(p')(p'')} \xrightarrow{(\nu r)\,\texttt{o@p(p')}} \mathsf{Wait}(r, \texttt{p}'')} \lfloor^{\text{B}}|_{\text{SOLICIT}}\rceil$$

$$\frac{}{\texttt{o@p(p')} \xrightarrow{(\nu r)\,\texttt{o@p(p')}} \mathbf{0}} \lfloor^{\text{B}}|_{\text{NOTIFY}}\rceil \qquad \frac{j \in J \quad \eta_j \xrightarrow{\mu} B'_j}{\sum_{i\in J}\,[\eta_i]\,\{B_i\} \xrightarrow{\mu} B'_j; B_j} \lfloor^{\text{B}}|_{\text{CHOICE}}\rceil$$

$$\frac{}{\texttt{o(p)} \xrightarrow{r:\texttt{o(p)}} \mathbf{0}} \lfloor^{\text{B}}|_{\text{ONEWAY}}\rceil \qquad \frac{}{\texttt{o(p)(p')}\,\{B\} \xrightarrow{r:\texttt{o(p)}} \mathsf{Exec}(r, \texttt{p}', B)} \lfloor^{\text{B}}|_{\text{REQUEST}}\rceil$$

$$\frac{}{\mathsf{Exec}(r, \texttt{p}, \mathbf{0}) \xrightarrow{\overline{r}\,\texttt{p}} \mathbf{0}} \lfloor^{\text{B}}|_{\text{EXECEND}}\rceil \qquad \frac{}{\mathsf{Wait}(r, \texttt{p}) \xrightarrow{r\,\texttt{p}} \mathbf{0}} \lfloor^{\text{B}}|_{\text{WAIT}}\rceil$$

$$\frac{B \xrightarrow{\mu} B'}{\mathsf{Exec}(r, \texttt{p}, B) \xrightarrow{\mu} \mathsf{Exec}(r, \texttt{p}, B')} \lfloor^{\text{B}}|_{\text{EXEC}}\rceil \qquad \frac{e(t) = \text{false}}{\texttt{while(}e\texttt{)}\,\{B\} \xrightarrow{\mathsf{read}\ t} \mathbf{0}} \lfloor^{\text{B}}|_{\text{WHILEEND}}\rceil$$

$$\frac{B \xrightarrow{\mu} B''}{B; B' \xrightarrow{\mu} B''; B} \lfloor^{\text{B}}|_{\text{SEQ}}\rceil \qquad \frac{B \xrightarrow{\mu} B''}{B \mid B' \xrightarrow{\mu} B'' \mid B} \lfloor^{\text{B}}|_{\text{PAR}}\rceil \qquad \frac{}{\texttt{p} = e \xrightarrow{\texttt{p}=e} \mathbf{0}} \lfloor^{\text{B}}|_{\text{ASSIGN}}\rceil$$

$$\frac{i = 1 \text{ if } e(t) = \text{true}, i = 2 \text{ otherwise}}{\texttt{if(}e\texttt{)}\,\{B_1\}\texttt{else}\{B_2\} \xrightarrow{\mathsf{read}\ t} B_i} \lfloor^{\text{B}}|_{\text{COND}}\rceil$$

$$\frac{e(t) = \text{true}}{\texttt{while(}e\texttt{)}\,\{B\} \xrightarrow{\mathsf{read}\ t} B; \texttt{while(}e\texttt{)}\,\{B\}} \lfloor^{\text{B}}|_{\text{WHILE}}\rceil$$

Figure 5.5: Correlation Calculus, semantics of behaviours.

5.3.3.1 Behavioural layer

The rules defining the semantics of behavioural terms are reported in Figure 5.5. Rules $\lfloor^{\text{B}}|_{\text{ONEWAY}}\rceil$ and $\lfloor^{\text{B}}|_{\text{NOTIFY}}\rceil$ allow, respectively, for the receiving and sending of asynchronous one-way messages. Rules $\lfloor^{\text{B}}|_{\text{REQUEST}}\rceil$ and $\lfloor^{\text{B}}|_{\text{SOLICIT}}\rceil$ do similarly for Request-Response patterns, handling also the subsequent response computation and sending. The computation of the response is handled by rule $\lfloor^{\text{B}}|_{\text{EXEC}}\rceil$; when the response computation terminates, the caller and the callee communicate again by means of the private and previously established channel r. The modeling of Request-Response replies through private channels supports classic client-server communications, where the client could be unable to expose inputs of its own due to external restrictions, e.g. firewalls.

5.3.3.2 Service layer

The service layer supports the execution of behaviours inside processes. The rules are reported in Figure 5.6. Rule $\lfloor^{\text{S}}|_{\text{START}}\rceil$ implements the spawning of a new local process by receiving a message that does not correlate with any running process on any correlation

$$\frac{B \xrightarrow{r:\mathtt{o}(\mathtt{p})} B'}{B \cdot t \cdot (r, \mathtt{o}, t') :: \tilde{m} \xrightarrow{\tau} B' \cdot t \leftarrow_{\mathtt{p}} t' \cdot \tilde{m}} \lfloor \text{S} | \textsc{Get} \rceil$$

$$\frac{B \xrightarrow{(\boldsymbol{\nu} r)\ \mathtt{o}@\mathtt{p}(\mathtt{p}')} B'}{B \cdot t \cdot \tilde{m} \xrightarrow{(\boldsymbol{\nu} r)\ \mathtt{o}@\langle\!\langle \mathtt{p}(t) \rangle\!\rangle (\mathtt{p}'(t))} B' \cdot t \cdot \tilde{m}} \lfloor \text{S} | \textsc{Send} \rceil$$

$$\frac{B \xrightarrow{\overline{r}\,\mathtt{p}} B'}{B \cdot t \cdot \tilde{m} \xrightarrow{\overline{r}\,\mathtt{p}(t)} B' \cdot t \cdot \tilde{m}} \lfloor \text{S} | \textsc{sr} \rceil \qquad \frac{B \xrightarrow{r\,\mathtt{p}} B'}{B \cdot t \cdot \tilde{m} \xrightarrow{r\,t'} B' \cdot t \leftarrow_{\mathtt{p}} t' \cdot \tilde{m}} \lfloor \text{S} | \textsc{rr} \rceil$$

$$\frac{B \xrightarrow{\mathtt{p}=e} B'}{B \cdot t \cdot \tilde{m} \xrightarrow{\tau} B' \cdot t \twoheadleftarrow_{\mathtt{p}} e(t) \cdot \tilde{m}} \lfloor \text{S} | \textsc{Assign} \rceil \qquad \frac{B \xrightarrow{\mathsf{read}\ t} B'}{B \cdot t \cdot \tilde{m} \xrightarrow{\tau} B' \cdot t \cdot \tilde{m}} \lfloor \text{S} | \textsc{Read} \rceil$$

$$\frac{Q \xrightarrow{\mu} Q'}{B \triangleright_{\alpha_C} P \mid Q \xrightarrow{\mu} B \triangleright_{\alpha_C} P \mid Q'} \lfloor \text{S} | \textsc{Lift} \rceil$$

$$\frac{t', \mathtt{o} \vdash_{\alpha_C} t}{B \triangleright_{\alpha_C} P \mid B' \cdot t \cdot \tilde{m} \xrightarrow{(\boldsymbol{\nu} r)\ \mathtt{o}(t')} B \triangleright_{\alpha_C} P \mid B' \cdot t \cdot \tilde{m} :: (r, \mathtt{o}, t')} \lfloor \text{S} | \textsc{Corr} \rceil$$

$$\frac{t, \mathtt{o} \not\vdash_{\alpha_C} P \qquad B \xrightarrow{r:\mathtt{o}(\mathtt{p})} B' \qquad t' = \mathsf{init}(t, \mathtt{o}, \alpha_C)}{B \triangleright_{\alpha_C} P \xrightarrow{(\boldsymbol{\nu} r)\ \mathtt{o}(t)} B \triangleright_{\alpha_C} P \mid B' \cdot t_\perp \leftarrow_{\mathtt{p}} t \leftarrow_{\mathtt{csets}} t' \cdot \epsilon} \lfloor \text{S} | \textsc{Start} \rceil$$

$$\mathsf{init}(t, \mathtt{o}, \alpha_C) = \begin{cases} t_\perp \twoheadleftarrow_{\mathtt{p}_1} f(\mathtt{p}_1)(t) \ldots \twoheadleftarrow_{\mathtt{p}_n} f(\mathtt{p}_n)(t) & \text{if } \alpha_C(\mathtt{o}) = (\{\mathtt{p}_1, \ldots, \mathtt{p}_n\}, f) \\ t_\perp & \text{if } \mathtt{o} \notin \mathsf{dom}(\alpha_C) \\ \text{undefined} & \text{otherwise} \end{cases}$$

Figure 5.6: Correlation Calculus, semantics of services.

$$\frac{S_1 \xrightarrow{(\nu r)\ \mathsf{o}@l_2(t)} S_1' \quad S_2 \xrightarrow{(\nu r)\ \mathsf{o}(t)} S_2' \quad r \notin \mathsf{cn}(S_1) \cup \mathsf{cn}(S_2)}{[S_1]_{l_1} \mid [S_2]_{l_2} \xrightarrow{\tau} (\boldsymbol{\nu} r)\ (\ [S_1']_{l_1} \mid [S_2']_{l_2}\ )}\ \lfloor^{\mathrm{N}}|_{\textsc{Com}}\rceil$$

$$\frac{S_1 \xrightarrow{r\,t} S_1' \quad S_2 \xrightarrow{\overline{r}\,t} S_2'}{(\boldsymbol{\nu} r)\ (\ [S_1]_{l_1} \mid [S_2]_{l_2}\ ) \xrightarrow{\tau} [S_1']_{l_1} \mid [S_2']_{l_2}}\ \lfloor^{\mathrm{N}}|_{\textsc{Response}}\rceil$$

$$\frac{N_1 \xrightarrow{\tau} N_1'}{N_1 \mid N_2 \xrightarrow{\tau} N_1' \mid N_2}\ \lfloor^{\mathrm{N}}|_{\textsc{Par}}\rceil \qquad \frac{N_1 \equiv N_1' \quad N_1' \xrightarrow{\mu} N_2' \quad N_2' \equiv N_2}{N_1 \xrightarrow{\mu} N_2}\ \lfloor^{\mathrm{N}}|_{\textsc{Eq}}\rceil$$

Figure 5.7: Correlation Calculus, semantics of networks.

set (thus giving precedence to existing processes), initialising its `csets` subtree if there is an aliasing definition for operation o. Note that the initialisation function $\mathsf{init}(t, \mathsf{o}, \alpha_C)$ is partial and undefined if the message does not contain all the correlation data specified in α_C for o; in this case, rule $\lfloor^{\mathrm{S}}|_{\textsc{Start}}\rceil$ can not be applied. The predicate $t', \mathsf{o} \not\vdash_{\alpha_C} P$ is defined whenever there is no state t in P such that $t', \mathsf{o} \vdash_{\alpha_C} t$. Moreover, $t \leftarrow_{\mathtt{p}} t'$ is a function that returns a new tree obtained from t by replacing the subtree pointed by $\mathtt{p}$ with t'; the function automatically creates the missing nodes for traversing t with $\mathtt{p}$, initializing them with $v_\perp$. Function $t \twoheadleftarrow_{\mathtt{p}} e(t')$ does the same but replaces only the root in $\mathtt{p}(t)$ with the value that results from the evaluation of e on t', $e(t')$. Rule $\lfloor^{\mathrm{S}}|_{\textsc{Get}}\rceil$ allows a running process to fetch the first element from its message queue. Rule $\lfloor^{\mathrm{S}}|_{\textsc{Send}}\rceil$ propagates the label for a sending, which will be used by the network layer for performing the actual message transmission; the rule substitutes the paths `p` and `p'` in the original label with, respectively, the location pointed by `p` and the data tree pointed by `p'` stored in the process state. Rules $\lfloor^{\mathrm{S}}|_{\textsc{Sr}}\rceil$ and $\lfloor^{\mathrm{S}}|_{\textsc{Rr}}\rceil$ close a Request-Response communication by exchanging the final reply. Rule $\lfloor^{\mathrm{S}}|_{\textsc{Assign}}\rceil$ models variable assignment. Rule $\lfloor^{\mathrm{S}}|_{\textsc{Corr}}\rceil$ allows a running session to receive a correlating message and store it in its local queue (we omit the condition for handling the special case of an empty queue $\tilde{m} = \epsilon$). All other rules are standard [51].

5.3.3.3 Network layer

The outer layer of our semantics, the network layer, deals with inter-service interactions. The rules are standard and reported in Figure 5.7.

5.4 Typing and Properties

In this section we discuss some desirable properties of services that can be captured with our language. Some of them are based on conditions that we will guarantee through the use of a typing system.

5.4.1 Properties

Our properties focus on the integrity of sessions and communications.

Property 5.4.0.1 (Message delivery atomicity). *Let* $N \equiv (\boldsymbol{\nu}\tilde{r})\ ([S_1]_{l_1} \mid M)$ *such that* $S_1 \xrightarrow{(\boldsymbol{\nu}r')\ \mathtt{o}@l_2(t')} S_1'$ *and* $N \xrightarrow{\tau} (\boldsymbol{\nu}\tilde{r})\ (\boldsymbol{\nu}r')\ ([S_1']_{l_1} \mid M')$. *Then,* $M \equiv [S_2]_{l_2} \mid M''$, $M' \equiv [S_2']_{l_2} \mid M''$ *and either:*

(i) $S_2 \equiv B \triangleright_{\alpha_C} P \mid B' \cdot t \cdot \tilde{m}$ *and* $S_2' \equiv B \triangleright_{\alpha_C} P \mid B' \cdot t \cdot \tilde{m} :: (r, \mathtt{o}, t')$; *or,*

(ii) $S_2 \equiv B \triangleright_{\alpha_C} P$ *and* $S_2' \equiv B \triangleright_{\alpha_C} P \mid B' \cdot t \leftarrow_{\mathtt{p}} t' \cdot \epsilon$ *for some* $t, \mathtt{p}$.

The property above states that if a service successfully executes a message sending then there is another service in the network that either (i) put the message in the queue of a correlating process or (ii) started a new process with a state containing the message data. This is guaranteed by our semantics since a message sending is completed only by synchronising with the receiver by means of rule $\lfloor^{\mathrm{S}}|_{\textsc{Start}}\rceil$ or rule $\lfloor^{\mathrm{S}}|_{\textsc{Corr}}\rceil$.

Property 5.4.0.2 (No session ambiguity). *For all* t', $\mathtt{o}$ *and service* $B \triangleright_{\alpha_C} P$, *there is at most one running session* $B' \cdot t \cdot \tilde{m}$ *in* P *such that* $t', \mathtt{o} \vdash_{\alpha_C} t$.

Our second property states that a service can never have more than one running process that correlates with the same message on any session (i.e., by any correlation set definition in the service). Such a situation would lead to non-deterministic assignments of incoming messages, which goes against the principle that a session should be uniquely identifiable.

Property 5.4.0.3 (Possible inputs). *Let* $S \equiv B \triangleright_{\alpha_C} P \mid B' \cdot t \cdot \tilde{m}$. *If* $B' \xrightarrow{r:\mathtt{o}\ (\mathtt{p})} B''$ *then at least one of the following holds:*

(i) $\tilde{m} = \tilde{m}' :: (r, \mathtt{o}, t') :: \tilde{m}''$;

(ii) $S \xrightarrow{(\boldsymbol{\nu}r)\ \mathtt{o}\ (t')} B \triangleright_{\alpha_C} P \mid P' \cdot t \cdot \tilde{m} :: (r, \mathtt{o}, t')$.

Property 3 says that if a process needs to perform an input action, then a message for such input action is in its queue and/or the enclosing service is able to receive a message for the process by correlation. In other words, whenever a process tries to perform an input action its state has the related correlation set fully instantiated.

Properties 5.4.0.2 and 5.4.0.3 depend on the states of the processes running in a service. Bad programming can lead to executions for which the properties do not hold. For example, for $\alpha_C = [\mathtt{join} \mapsto (\{\mathtt{x}\}, [\mathtt{x} \mapsto \epsilon])]$, if a service with behaviour

```
start(a);csets.x = 5;join(b)
```

gets invoked twice on operation `start`, it will spawn two processes which will both execute **`csets`**`.x = 5`. After that, by α_C, both processes can correlate with a message for operation `join` with value `5` as root node. This situation breaks Property 5.4.0.2, leading to non-deterministic message routing. Also, if $\alpha_C = [\mathtt{join} \mapsto (\{\mathtt{x}, \mathtt{y}\}, [\mathtt{x} \mapsto \epsilon, \mathtt{y} \mapsto \mathtt{y}])]$, we break Property 5.4.0.3: the two processes would be stuck forever waiting for a message for `join`, because rule $\lfloor^{\mathrm{S}}|_{\textsc{Corr}}\rceil$ could never be applied due to the lack of a value for `y` in the process states.

We address bad correlation programming as exemplified above with a typing discipline, which we formalise in the following.

5.4.2 Type system

We present a type system that focuses on the manipulation of correlation data. Our typing performs an initialisation analysis for correlation variables. Although this is a well-established technique, our setting requires particular attention to the concurrent execution of multiple processes, and the interplay between process behaviour and the aliasing function α_C.

Typing judgements have the form

$$\Gamma \vdash B : \Delta_N | \Delta_P$$

where $\Delta_N \subseteq \mathsf{Paths}$ and $\Delta_P \subseteq \mathsf{Paths} \times \{\circ, \bullet\}$. We also make use of the judgements $\vdash S$ and $\vdash N$, for asserting that services and networks are well-typed. In a judgement $\Gamma \vdash B : \Delta_N | \Delta_P$, Δ_N says which correlation paths B *needs* to be initialised its execution, and Δ_P contains the correlation paths *provided*, i.e., initialised, by B. In Δ_P each correlation path is flagged with either $\circ$, telling that the path carries a fresh value, or $\bullet$, telling that the path does not carry a fresh value. An environment $\Gamma : \mathcal{O} \rightharpoonup C$ maps each operation to its correlation set. The typing rules are reported in Figure 5.8.

We comment the typing rules. Rule $\lfloor^{\mathrm{T}}|_{\textsc{Nil}}\rceil$ types the terminated behaviour, which does not need nor provide any correlation path. Rule $\lfloor^{\mathrm{T}}|_{\textsc{Assign}}\rceil$ types assignments to paths that are not used for correlation ($\mathtt{p} \neq \mathtt{csets.p'}$). Rules $\lfloor^{\mathrm{T}}|_{\textsc{Cset-New}}\rceil$ and $\lfloor^{\mathrm{T}}|_{\textsc{Cset-Expr}}\rceil$ type, respectively, the assignment of a fresh or a non-fresh value to a correlation path. In $\lfloor^{\mathrm{T}}|_{\textsc{Cset-Expr}}\rceil$ we require e to be defined, i.e., that its evaluation will not yield $v_\perp$. This is a simple (but omitted) definite assignments analysis. Rule $\lfloor^{\mathrm{T}}|_{\textsc{Cond}}\rceil$ types a conditional by requiring the correlation paths needed by both branches and checking that the different branches provide the same correlation paths. Rule $\lfloor^{\mathrm{T}}|_{\textsc{Choice}}\rceil$ follows similar reasoning to rule $\lfloor^{\mathrm{T}}|_{\textsc{Cond}}\rceil$. In rule $\lfloor^{\mathrm{T}}|_{\textsc{Par}}\rceil$, we type the parallel composition of two behaviours by requiring the paths of both and checking that they provide different paths (to avoid data races on correlation paths). Rule $\lfloor^{\mathrm{T}}|_{\textsc{Seq}}\rceil$ types a sequential composition, allowing the second behaviour to use the paths provided by the first. Rule $\lfloor^{\mathrm{T}}|_{\textsc{While}}\rceil$ is standard, and cannot provide any paths. In rules $\lfloor^{\mathrm{T}}|_{\textsc{Noti}}\rceil$ and $\lfloor^{\mathrm{T}}|_{\textsc{Sr}}\rceil$ we do not require any correlation paths, since those are used only for receiving. Rule $\lfloor^{\mathrm{T}}|_{\textsc{Ow}}\rceil$ types a one-way input, requiring the correlation paths necessary for the operation ($\Gamma(\circ)$) and checking that the correlation set for the operation is not empty (which would make it impossible to correlate a message for it). Differently, in rule $\lfloor^{\mathrm{T}}|_{\textsc{Ow-Starter}}\rceil$ we judge that the correlation set for an input operation used at the top level of a service definition (identified by the annotation s in $\vdash_s$) will be provided, since our rule $\lfloor^{\mathrm{S}}|_{\textsc{Start}}\rceil$ will initialise it in the state of the newly created process. In the rule, $\Gamma(\!(\circ)\!)$ returns the correlation set for $\circ$ in Γ if it is defined, or the empty set otherwise; formally:

$$\Gamma(\!(\circ)\!) = \begin{cases} \Gamma(\circ) & \text{if } \circ \in \mathsf{dom}(\Gamma) \\ \emptyset & \text{otherwise} \end{cases}$$

Rules $\lfloor^{\mathrm{T}}|_{\textsc{Rr}}\rceil$ and $\lfloor^{\mathrm{T}}|_{\textsc{Rr-Starter}}\rceil$ follow similar reasonings. In rule $\lfloor^{\mathrm{T}}|_{\textsc{Starter}}\rceil$, a starter is well-typed if its behaviour is well-typed. The premise $\Gamma = \pi_1(\alpha_C)$ ensures that Γ agrees with the correlation aliasing function of the service; formally:

$$\pi_1(\alpha_C) = \{[\circ \mapsto c] \mid \alpha_C(\circ) = [c \mapsto f]\}$$

$$\frac{}{\Gamma \vdash \mathbf{0} : \emptyset|\emptyset} \lfloor^{\mathrm{T}}|_{\textsc{Nil}}\rceil \qquad \frac{\mathtt{p} \neq \mathtt{csets.p'}}{\Gamma \vdash \mathtt{p} = e : \emptyset|\emptyset} \lfloor^{\mathrm{T}}|_{\textsc{Assign}}\rceil$$

$$\frac{}{\Gamma \vdash \mathtt{csets.p} = \mathtt{new} : \emptyset|\{\mathtt{p}^{\circ}\}} \lfloor^{\mathrm{T}}|_{\textsc{Cset-New}}\rceil \qquad \frac{e \neq \mathtt{new} \quad e \text{ defined}}{\Gamma \vdash \mathtt{csets.p} = e : \emptyset|\{\mathtt{p}^{\bullet}\}} \lfloor^{\mathrm{T}}|_{\textsc{Cset-Expr}}\rceil$$

$$\frac{\Gamma \vdash B_1 : \Delta_{N_1}|\Delta_P \quad \Gamma \vdash B_2 : \Delta_{N_2}|\Delta_P}{\Gamma \vdash \mathtt{if}(e)\,\{B_1\}\,\mathtt{else}\,\{B_2\} : \Delta_{N_1} \cup \Delta_{N_2}|\Delta_P} \lfloor^{\mathrm{T}}|_{\textsc{Cond}}\rceil$$

$$\frac{\Gamma \vdash \eta_i; B_i : \Delta_{N_i}|\Delta_P \quad \Delta_N = \bigcup_{i\in I} \Delta_{N_i}}{\Gamma \vdash \sum_{i\in I} [\eta_i]\,\{B_i\} : \Delta_N|\Delta_P} \lfloor^{\mathrm{T}}|_{\textsc{Choice}}\rceil$$

$$\frac{\Gamma \vdash B_1 : \Delta_{N_1}|\Delta_{P_1} \quad \Gamma \vdash B_2 : \Delta_{N_2}|\Delta_{P_2}}{\Gamma \vdash B_1 \mid B_2 : \Delta_{N_1} \cup \Delta_{N_2}|\Delta_{P_1} \uplus \Delta_{P_2}} \lfloor^{\mathrm{T}}|_{\textsc{Par}}\rceil$$

$$\frac{\Gamma \vdash B_1 : \Delta_{N_1}|\Delta_{P_1} \quad \Gamma \vdash B_2 : \Delta_{N_2}|\Delta_{P_2} \quad \Delta' = (\Delta_{N_2} \setminus \Delta_{P_1}) \cup \Delta_{N_1}}{\Gamma \vdash B_1; B_2 : \Delta'|\Delta_{P_1} \uplus \Delta_{P_2}} \lfloor^{\mathrm{T}}|_{\textsc{Seq}}\rceil$$

$$\frac{\Gamma \vdash B : \Delta_N|\emptyset}{\Gamma \vdash \mathtt{while}(e)\,\{B\} : \Delta_N|\emptyset} \lfloor^{\mathrm{T}}|_{\textsc{While}}\rceil$$

$$\frac{}{\Gamma \vdash \mathtt{o@p(p')} : \emptyset|\emptyset} \lfloor^{\mathrm{T}}|_{\textsc{Noti}}\rceil \qquad \frac{\mathtt{p''} \neq \mathtt{csets.p'''}}{\Gamma \vdash \mathtt{o@p(p')(p'')} : \emptyset|\emptyset} \lfloor^{\mathrm{T}}|_{\textsc{Sr}}\rceil$$

$$\frac{\Gamma(\mathtt{o}) = c \neq \emptyset \quad \mathtt{p} \neq \mathtt{csets.p'}}{\Gamma \vdash \mathtt{o(p)} : c|\emptyset} \lfloor^{\mathrm{T}}|_{\textsc{Ow}}\rceil \qquad \frac{\mathtt{p} \neq \mathtt{csets.p'}}{\Gamma \vdash_s \mathtt{o(p)} : \emptyset|\Gamma((\mathtt{o}))} \lfloor^{\mathrm{T}}|_{\textsc{Ow-Start}}\rceil$$

$$\frac{\Gamma(\mathtt{o}) = c \neq \emptyset \quad \Gamma \vdash B : \Delta_N|\Delta_P}{\Gamma \vdash \mathtt{o(p)(p')}\,\{B\} : c \cup \Delta_N|\Delta_P} \lfloor^{\mathrm{T}}|_{\textsc{Rr}}\rceil$$

$$\frac{\Gamma \vdash B : \Delta_N|\Delta_P}{\Gamma \vdash_s \mathtt{o(p)(p')}\,\{B\} : \Delta_N|\Delta_P \uplus \Gamma((\mathtt{o}))} \lfloor^{\mathrm{T}}|_{\textsc{Rr-Start}}\rceil$$

$$\frac{\Gamma = \pi_1(\alpha_C) \quad \Gamma \vdash B : \emptyset|\Delta_P \quad \Delta_P \ltimes C}{\vdash \mathbf{0} \triangleright_{\alpha_C} B \cdot t_\perp \cdot \epsilon} \lfloor^{\mathrm{T}}|_{\textsc{Starter}}\rceil$$

$$\frac{\begin{array}{c} i \in I \quad \Gamma = \pi_1(\alpha_C) \quad \Gamma \vdash_s \eta_i : c|\Delta_{P_i} \quad \Gamma \vdash_s B_i : \Delta'_{N_i}|\Delta'_{P_i} \\ \Delta'_{N_i} \subseteq c \cup \Delta_{P_i} \quad \Delta_{P_i} \cap \Delta'_{P_i} = \emptyset \quad \Delta_{P_i} \uplus \Delta'_{P_i} \ltimes C \end{array}}{\vdash \sum_{i\in I} [\eta_i]\,\{B_i\} \triangleright_{\alpha_C} \mathbf{0}} \lfloor^{\mathrm{T}}|_{\textsc{Service}}\rceil$$

Figure 5.8: Correlation Calculus, typing rules.

$$\frac{t, \circ \vdash_{\alpha_C} t_1 \quad t, \circ \vdash_{\alpha_C} t_2 \quad S = B \triangleright_{\alpha_C} P \mid B_1 \cdot t_1 \cdot \tilde{m}_1 \mid B_2 \cdot t_2 \cdot \tilde{m}_2}{[S]_l \xrightarrow{\texttt{wrong}\, l} [S]_l} \lfloor^{\mathrm{N}}|_{\textsc{Wrong-Corr}}\rceil$$

$$\frac{B' \xrightarrow{r:\circ\,(\mathrm{p})} B'' \quad (r, \mathtt{o}, t') \notin \tilde{m} \qquad B \triangleright_{\alpha_C} P \mid B' \cdot t \cdot \tilde{m} \overset{(\nu r)\ \circ\ (t')}{\nrightarrow} B \triangleright_{\alpha_C} P \mid B' \cdot t \cdot \tilde{m} :: (r, \mathtt{o}, t')}{[B \triangleright_{\alpha_C} P \mid B' \cdot t \cdot \tilde{m}]_l \xrightarrow{\texttt{wrong}\, l} [B \triangleright_{\alpha_C} P \mid B' \cdot t \cdot \tilde{m}]_l} \lfloor^{\mathrm{N}}|_{\textsc{Wrong-Input}}\rceil$$

$$\frac{N \xrightarrow{\texttt{wrong}\, l} N'}{N \mid N'' \xrightarrow{\texttt{wrong}\, l} N' \mid N''} \lfloor^{\mathrm{N}}|_{\textsc{Wrong}}\rceil$$

Figure 5.9: Correlation Calculus, semantics of runtime errors.

Furthermore, the relation $\ltimes$ captures that, for the sake of being uniquely identifiable by correlation, a process needs at least one correlation variable to be fresh for every correlation set that is completely initialised:

Definition 5.4.1 (Correlation set freshness relation $\ltimes$)**.**

$$\Delta_P \ltimes C \quad \text{iff} \quad \forall c \in C \,.\, (\nexists \mathtt{p} \in c \,.\, \mathtt{p} \notin \Delta_P) \;\Rightarrow\; (\exists \mathtt{p} \in c \,.\, \mathtt{p}^{\circ} \in \Delta_P)$$

5.4.3 Typing soundness

We prove our typing sound by introducing a notion of error in the semantics of the Correlation Calculus, with the rules reported in Figure 5.9. Specifically, we add two error rules and another rule for lifting their observable behaviour. Both error rules use a label $\texttt{wrong}\, l$ that carries the location l of the originating service. Rule $\lfloor^{\mathrm{N}}|_{\textsc{Wrong-Corr}}\rceil$ requires that a service has two running sessions that correlate with the same message t for the same operation $\circ$, the negation of Property 5.4.0.2. Rule $\lfloor^{\mathrm{N}}|_{\textsc{Wrong-Input}}\rceil$, instead, is active when a running process wishes to input on an operation for which there is no message in its message queue and the correlation mechanism can route no new message to such a session, the negation of Property 5.4.0.3.

Next, we augment our typing system to deal with runtime terms and networks, with the rules reported in Figure 5.10. Rule $\lfloor^{\mathrm{T}}|_{\textsc{Wait}}\rceil$ types a response wait, which does not require nor provide any correlation path. Rule $\lfloor^{\mathrm{T}}|_{\textsc{Exec}}\rceil$ types the runtime computation of a response for a request with the environments of its executing body. Rule $\lfloor^{\mathrm{T}}|_{\textsc{Process}}\rceil$ types a process, using the process state to satisfy the required correlation paths of the process behaviour. This is formalised by the predicate $\mathsf{co}(t, \Delta_N|\Delta_P)$, which we assume holds only when the following conditions hold:

$$\forall \mathtt{p} \in \Delta_N.\mathtt{csets.p}(t) \text{ defined} \quad \text{and} \quad \forall \mathtt{p} \in \Delta_P.\mathtt{csets.p}(t) \text{ undefined}$$

In rule $\lfloor^{\mathrm{T}}|_{\textsc{RTService}}\rceil$, we add to the same premises of rule $\lfloor^{\mathrm{T}}|_{\textsc{Service}}\rceil$ the extra condition $\vdash B'_j \cdot t_j \cdot \tilde{m}_j$ (the runtime processes in the service must be well-typed) and

$$\nexists k, k' \in J, c \in C.\, \forall \mathtt{p} \in c.\, \mathtt{csets.p}(t_k) = \mathtt{csets.p}(t_{k'}) \neq v_{\perp}$$

$$\frac{}{\Gamma \vdash \mathsf{Wait}(r, \mathtt{p}) : \emptyset|\emptyset} \lfloor^{\mathrm{T}}|_{\textsc{Wait}}\rceil \qquad \frac{\Gamma \vdash B : \Delta_N|\Delta_P}{\Gamma \vdash \mathsf{Exec}(r, \mathtt{p}, B) : \Delta_N|\Delta_P} \lfloor^{\mathrm{T}}|_{\textsc{Exec}}\rceil$$

$$\frac{\Gamma \vdash B : \Delta_N|\Delta_P \quad \mathsf{co}(t, \Delta_N|\Delta_P)}{\vdash B \cdot t \cdot \tilde{m}} \lfloor^{\mathrm{T}}|_{\textsc{Process}}\rceil$$

$$\frac{\begin{array}{c} i \in I \quad \Gamma = \pi_1(\alpha_C) \quad \Gamma \vdash_s \eta_i : c|\Delta_{P_i} \quad \Gamma \vdash_s B_i : \Delta'_{N_i}|\Delta'_{P_i} \\ \Delta'_{N_i} \subseteq c \cup \Delta_{P_i} \quad \Delta_{P_i} \cap \Delta'_{P_i} = \emptyset \quad \Delta_{P_i} \uplus \Delta'_{P_i} \ltimes C \\ \vdash B'_j \cdot t_j \cdot \tilde{m}_j \quad \nexists k, k' \in J, c \in C.\, \forall \mathtt{p} \in c.\, \mathtt{csets.p}(t_k) = \mathtt{csets.p}(t_{k'}) \neq v_\perp \end{array}}{\vdash \sum_{i \in I} [\eta_i]\{B_i\} \triangleright_{\alpha_C} \prod_{j \in J} B'_j \cdot t_j \cdot \tilde{m}_j} \lfloor^{\mathrm{T}}|_{\textsc{RTService}}\rceil$$

$$\frac{\vdash S}{\vdash [S]_l} \lfloor^{\mathrm{T}}|_{\textsc{Location}}\rceil \qquad \frac{\vdash N \quad \vdash N'}{\vdash N \mid N'} \lfloor^{\mathrm{T}}|_{\textsc{Network}}\rceil$$

Figure 5.10: Correlation Calculus, runtime typing rules.

which ensures that each process is distinguishable from the others by at least one correlation value.

We can finally show the main results of our type system:

Theorem 5.4.2 (Subject Reduction)**.** $\vdash N$ *and* $N \xrightarrow{\mu} N'$ *implies that* $\vdash N'$.

Proof. By induction on the derivation of $N \xrightarrow{\mu} N'$. □

Theorem 5.4.3 (Typing soundness)**.** *Let* l *be a location in* N*. Then,*

(i) $\vdash N$ *implies that* $N \overset{\mathtt{wrong}\,l}{\not\rightarrow} N'$*; and,*

(ii) $\vdash N$ *implies that* $N \mid N' \overset{\mathtt{wrong}\,l}{\not\rightarrow} N''$

Proof. Immediate by the definition of the runtime typing rules and the semantics of the Correlation Calculus. □

Note that point (ii) of Theorem 5.4.3 states that safety is independent of its context, i.e., a well-typed network respects our properties regardless of its context.

5.5 Implementation

We report here a brief description of the Jolie interpreter architecture and the changes that we introduced to support our proposal.

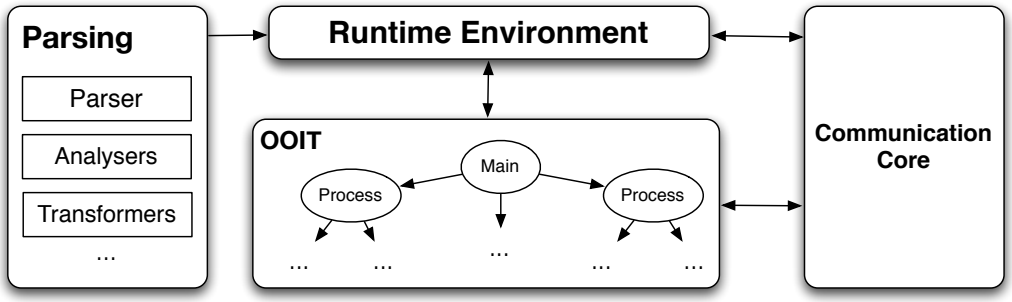


Figure 5.11: Jolie, interpreter architecture.

Jolie interpreter architecture. The Jolie interpreter is developed in the Java language; its structure is depicted in Figure 5.11. The architecture of the Jolie interpreter resembles that of our Correlation Calculus, with the OOIT, the *Runtime Environment* and the *Communication Core* representing, respectively, the behavioural, service and network layers [71]. The three components are loosely coupled, operating through their respective APIs just like our three layers interact by means of the labels. The interpreter architecture has the structure: The *Parsing* module encompasses the parsing of programs, extracting the related ASTs (Abstract Syntax Trees) and analysing them. The AST is used to produce the OOIT (Object-Oriented Interpretation Tree), which is composed by generic `Process` objects, each one responsible for executing the semantics of a single rule in the behavioural semantic layer. OOIT nodes do not deal directly with process state and communication semantics, leaving this task to the *Runtime Environment* and the *Communication Core* modules. The former manages the execution states of running sessions, whereas the latter supports the other components in performing communications and listens for messages coming from external services.

Changes. We updated the Parsing module for handling the syntax for defining correlation sets, adding also an analyser that implements our type system and the generator that creates the aliasing function α_C for the Runtime Environment. The Java class used by the Runtime Environment for controlling the execution of a process has been augmented with a queue that stores received messages. When a node from the OOIT asks for a message input, the Runtime Environment checks the message queue of its session as specified by rule $\lfloor {}^{\text{S}}|_{\text{GET}} \rceil$. The queue is filled by the Communication Core looking for correlation on incoming messages as defined in rule $\lfloor {}^{\text{S}}|_{\text{CORR}} \rceil$. If no correlating process is found it asks the Runtime Environment to start a new process with the message, following rule $\lfloor {}^{\text{S}}|_{\text{START}} \rceil$. If the process can not be started, a `CorrelationError` fault is sent to the invoker and the message is discarded (cf. [73, 49] for the semantics of faults in Jolie).

Request-Response interactions are supported by abstract channel objects. The OOIT nodes involved in Request-Response communications – i.e. the two nodes implementing rules $\lfloor {}^{\text{B}}|_{\text{REQUEST}} \rceil$ and $\lfloor {}^{\text{B}}|_{\text{SOLICIT}} \rceil$ and their continuation – are given the channel of interest as a parameter and can use it for sending or receiving responses as specified by rules $\lfloor {}^{\text{B}}|_{\text{EXECEND}} \rceil$ and $\lfloor {}^{\text{B}}|_{\text{WAIT}} \rceil$. These channels are managed by the Communication Core and hide the underlying complexity for managing the receiving of the reply, which is dependent on underlying transport details, e.g., one could use separate TCP/IP sockets or additional tags in message

headers.

5.6 Example

We present now an example inspired by the OpenID Authentication specifications [81]. OpenID is a widely adopted decentralised authentication protocol that allows a service, called *relying party*, to authenticate a user, the *client*, by relying on another external service that is responsible for handling identities, the *identity provider*. When the client requests access to the relying party, the latter opens an authentication session in the identity provider. The client can then send its authentication credentials to the session in the identity provider, which will inform the relying party on the result of the authentication attempt.

We implemented the protocol in our updated version of Jolie. The example can be downloaded at [70], where we support web browser clients by means of the Jolie integration with HTTP that will be described in Chapter 6.

The code below is a sketch of the relying party service:

```
cset { clientToken: ... }
cset { secureToken: AuthMessage.secureToken }
interface RelyingPartyInterface {
OneWay: authSucceeded(AuthMessage), authFailed(AuthMessage)
RequestResponse: login(LoginRequest)(Redirection) }
main {
  login( loginRequest )( redirection ) {
    clientToken = new; secureToken = new;
    openRequest.relyingPartyIdentifier = MY_IDENTIFIER;
    openRequest.clientToken = csets.clientToken;
    openRequest.secureToken = csets.secureToken;
    openAuth@IdentityProvider( openRequest );
    /* ... build redirection message for client ... */
  };  [ authSucceeded( message ) ] { /* ... */ }
      [ authFailed( message ) ]    { /* ... */ }
}
```

First, the service receives a request on the Request-Response operation `login` from the client for initiating the protocol. The body of `login` generates two fresh tokens: `clientToken`, referred by the first correlation set[2], and `secureToken`, referred by the second one. We will use `clientToken` for receiving messages from the client and `secureToken` for receiving messages from the identity provider. The client is not informed about `secureToken`, preventing it to maliciously act as the identity provider. The body of `login` performs a call to the identity provider, opening an authentication session and communicating `secureToken`. We can now safely reply to the client that invoked operation `login`: Property 5.4.0.1, from § 5.4, guarantees that the process in the identity provider has been started at this point and that the client will therefore find it ready. The reply will redirect the client to the identity provider. The relying party will then wait for a notification about the result of the

[2] We omit the aliasings for `clientToken` in the relying party implementation sketch, since it will only be used after establishing whether the user can log in.

authentication attempt, hence the input choice on the operations `authSucceeded` and `authFailed`, which correlate through `secureToken`.

We now show the identity provider behavioural code sketch omitting the interface definitions: we assume input types to be named with their respective operation names with an initial uppercase letter.

```
cset { relyingPartyIdentifier:
          OpenAuthentication.relyingPartyIdentifier
          Authenticate.relyingPartyIdentifier,
       token: OpenAuthentication.token Authenticate.token }
main {
  openAuth( openRequest ); authenticate( authRequest );
  /* ... verify authentication ... */
  message.secureToken = openRequest.secureToken;
  if ( verified ) { authSucceeded@RelyingParty( message ) }
  else            { authFailed@RelyingParty( message ) }
}
```

The service can start a process with an input on `openAuth` (to be called by the relying party). The operation receives the values for initialising the correlation set, which is composed by two variables: `relyingPartyIdentifier` and `token`. We need both variables because there may be multiple active processes for handling requests from different relying parties: two relying parties may generate a same value for `token`. We solve this issue by adding the identifier, e.g., a URL, of the relying party to the correlation set. After the process has been started, we wait for the user credentials on operation `authenticate`. The credentials are verified and the result sent to the relying party.

5.7 Related Work

Previous versions of Jolie (including its initial formal model SOCK [51]) feature correlation sets where correlation data is manipulated within processes. However, they support no correlation aliasing and no static analysis for identifying bad correlation programming. Moreover, they do not feature multiple correlation sets. All correlation variables are, instead, put in one single correlation set, which does not act as unique session identifier: sessions may be ambiguous under correlation and hence message routing can be non-deterministic.

Our approach takes inspiration from BPEL [79], which supports multiple correlation sets for identifying different processes and the sessions they participate in. In BPEL, correlation programming is mixed with that of behaviours: correlation sets are scoped in specific code blocks, and different input actions inside the same behaviour can use different correlation sets for receiving even if they use the same operation name. This makes BPEL programming more error-prone than in our approach, where correlation sets are based on the service interface (its operations) and are defined independently from behaviours. Our language expressiveness is still high, due to our support for correlation data manipulation inside processes. BPEL does not support correlation programming with a typing discipline, but relies on runtime faults for signaling undesired situations that the programmer specifies

manually. Finally, BPEL does not come with formal specifications, making it impossible to apply formal reasoning as we did in § 5.4 for guaranteeing safety properties.

Blite [65] is a model for service orchestration in which programs can be compiled to BPEL processes [32]. The model is formally specified, but the final compilation to BPEL makes the approach suffer from the unpredictable behaviour of the execution engine, due to the lack of formality of BPEL specifications. Similarly, the calculus for web services COWS [64] allows to correlate sessions based on channel usage. COWS features several tools for static analyses and an interpreter, however it lacks a fully-fledged language implementation like Jolie.

In [57], the authors present an implementation of channel-based sessions relying on session types [54]. In their setting, message routing does not rely on data transmission like in our model.

5.8 Conclusions

We have presented a language for programming services with correlation sets. Our approach features a direct manipulation of correlation data in programs and a notion of correlation aliasing. We have shown how both aspects can be disciplined by means of a type system. The applicability of our work has been demonstrated by exposing implementations of real-world scenarios where correlation sets can be successfully employed. Our solution has replaced the previous correlation mechanism in the Jolie language [74]. The features guaranteed by our Properties 5.4.0.2 and 5.4.0.3 are similar to those provided by private channels in the π-calculus [69]. In our approach different sessions use different instances of correlation sets, much like in the π-calculus replications of a same process use different private channels.

Our semantics for message queues can lead to deadlocks, because a process must consume messages in the same order in which they are received. In Chapter 7, we extend the implementation of Jolie to handle a separate message queue for each correlation set, to support more concurrency between sessions (each represented by a correlation set). Another issue in our model is that it does not handle the garbage collection of processes, i.e., terminated processes are not removed from their executing service. Handling this aspect is nontrivial, because a terminated process may have some messages left in its queue which must be dealt with. We leave the investigation of this issues to future work.

More refined forms of static analysis may be developed for correlation. An interesting aspect would be to analyzse the behaviour of service networks by introducing behavioural types for participants such as session types [54, 55]. Another topic to be explored is that of security. Programs may be checked to establish that correlation values are not *compromised*.

CHAPTER 6

Process-aware Web Programming

6.1 Introduction

In this Chapter, we extend our language implementation from Chapter 5 to develop a programming framework for the development of process-aware web information systems. We then use such extension to evaluate how our methodology of programming sessions with correlation sets integrates with existing web technologies (e.g., HTTP and HTML) and practices (e.g., web servers and multiparty sessions over HTTP).

A Process-Aware Information System (PAIS) is an information system based upon the execution of business processes. PAIS's are largely adopted in many application scenarios [39], from inter-process communication to automated business integration. Since processes can assume many different structures (see [14] for a systematic account), many formal methods [98, 48, 64, 31], tools [100, 74, 53, 56, 45], and standards [79, 102, 1] have been developed to provide languages for their definition, verification, and execution.

In the last two decades, web applications have joined the trend of process-awareness. Web processes are usually implemented server-side on top of *sessions*, which track incoming messages related to the same conversation. Sessions are supported with a local memory state, which lives through different client invocations until the session is terminated.

The major frameworks for developing web applications (e.g., PHP, Ruby on Rails, and Java EE) do not support the explicit programming of structured processes. As a workaround, programmers usually simulate the latter by exploiting the session-local memory state. For example, consider a process where a user has to authenticate through a `login` operation before accessing another operation, say `createNews` (for posting a news on a website). This would be implemented by defining the `login` and the `createNews` operations separately. The code for `login` would update a bookkeeping variable in the session state and the implementation for `createNews` would check that variable when it is invoked by the user. Although widely used, this approach is error-prone: since processes can assume quite complex structures, simulating them through bookkeeping variables soon becomes cumbersome. Consequently, the produced code may be poorly readable and hard to maintain.

The limitations described above can be avoided by adopting a multi-tier architecture. For example, we may stratify an application by employing a web server technology (e.g., Apache Tomcat) for serving content to web browsers; a web scripting framework (e.g., PHP) for programmable request processing; a process-oriented language (e.g., WS-BPEL [79]) for modelling the application processes; and, finally, mediation technologies such as proxies and ESB [33] for integrating the web application within larger systems. Such an architecture would offer a good separation of concerns. However, the resulting system would be highly heterogeneous, requiring a specific know-how for handling each

part. Thus, it would be hard to maintain and potentially prone to breakage in case of modifications.

The aim of this Chapter is to simplify the programming of process-aware web information systems. We present a programming framework that successfully captures the different components of such systems (web servers, processes, ...) and their integration using a homogeneous set of concepts. We build our results on top of Jolie [74, 60] (§ 6.2), a general-purpose service-oriented programming language that can handle both the modelling of processes (without bookkeeping code) and their integration within larger distributed systems [60].

6.1.1 Contributions

We report our major contributions.

Web processes. We integrate the Jolie language with the HTTP protocol, enabling processes written in Jolie to send and receive HTTP messages (§ 6.3). The integration is *seamless*, meaning that the processes defined in Jolie remain abstract from the underlying HTTP mechanisms and data formats: Jolie's data structures are transparently transformed to HTTP messages and vice versa (§ 6.3.1). Transformations can be configured through separate parameters (§ 6.3.2).

Web servers as processes. We enable Jolie processes to model the programming of web servers, for serving content to clients (§ 6.4.1). Hence, in our framework web servers are not ad-hoc third-party programs anymore, but are instead modelled using the same language that we use for defining processes.

Multiparty sessions. A session in a web server is typically dedicated to a single web client. The latter can refer to it through a session identifier. In process-oriented languages, instead, sessions can be between more than two participants. Different participants are identified by using different session identifiers, or multiple identifiers in the case of correlation sets [79]. We combine correlation sets with HTTP to introduce multiparty sessions in web applications (§ 6.4.2).

Multi-tiering. Aggregation [71] is a primitive of Jolie that allows for the composition of separate services in an information system, as in ESB [33]. Our HTTP implementation supports aggregation transparently. We show how to use this combination to obtain multi-tiered architectures.

6.2 An overview of Jolie

Jolie [74] is a general-purpose service-oriented programming language, released as an open-source project [60] and formally specified as a process calculus [71, 48] (see Chapter 5). In this section we describe some of its features relevant for our discussion.

A Jolie program defines a service and is a composition of two parts: *behaviour* and *deployment*. A behaviour defines the implementation of the operations offered by a service; it consists of communication and computation instructions, composed into a structured process (a workflow) using constructs such as sequences, parallels, and internal/external choices. Behaviours rely on *communication ports* to perform communications, which are

B	::=	η	*(input)*
	|	$\overline{\eta}$	*(output)*
	|	`[` η_1 `]` `{` B_1 `}` ... `[` η_n `]` `{` B_n `}`	*(input choice)*
	|	**`if`** `(` e `)` B_1 **`else`** B_2	*(cond)*
	|	**`while`** `(` e `)` B	*(while)*
	|	B **`;`** B'	*(seq)*
	|	B **`|`** B'	*(par)*
	|	**`throw`**`(f)`	*(throw)*
	|	`x =` e	*(assign)*
	|	`x -> y`	*(alias)*
	|	**`nullProcess`**	*(inact)*

η	::=	`o(x)`	*(one-way)*
	|	`o(x)` `(`e`)` `{` B `}`	*(request-response)*
$\overline{\eta}$	::=	`o@OP(`e`)`	*(notification)*
	|	`o@OP(`e`)(y)`	*(solicit-response)*

Figure 6.1: Jolie, behavioural syntax (selection).

to be correctly defined in the deployment part. The latter can also make use of architectural primitives for handling the structure of an information system. Formally, a Jolie program is structured as:

$$D \quad \textbf{main} \ \{ \ B \ \}$$

where D represents the deployment part and B the behavioral part.

Behaviours. Figure 6.1 reports the (selected) syntax for service behaviours, which offers primitives for performing communications, computation, and their composition in processes. We briefly comment the syntax. Terms *(input)*, *(output)*, and *(input choice)* implement communications. An input η can either be a one-way or a request-response, following the WSDL standard [13]. Statement *(one-way)* receives a message for operation `o` and stores its content in variable `x`. Term *(request-response)* receives a message for operation `o` in variable `x`, executes behaviour B (called the *body* of the request-response input), and then sends the value of the evaluation of expression e to the invoker. *(notification)* and *(solicit-response)* dually implement the outputs towards the input primitives. *(notification)* sends a message containing the value of the evaluation of expression e. *(solicit-response)* sends a message with the evaluation of e and then waits for a response from the invoked service, storing it afterwards in variable `y`. In the output statements, `OP` is a reference to an *output port*, to be defined in the deployment part. *(input choice)* is similar to the `pick` construct in WS-BPEL: when a message for an input η_i can be received, all other branches are deactivated and η_i and, afterwards, its respective branch behaviour B_i are executed.

Terms *(cond)* and *(while)* implement respectively the standard conditional and iteration constructs. *(seq)* models sequential execution and reads as: execute B, wait for its termination, and then run B'. *(par)*, instead, runs B and B' in parallel. *(throw)* throws a fault

IP ::= **inputPort** P $\quad$ OP ::= **outputPort** P
$Port$ ::= id {
Location: Loc
Protocol: $Proto$
Interfaces: $\mathsf{iface}_1, \ldots, \mathsf{iface}_n$
}

Figure 6.2: Jolie, syntax of ports (selection).

signal `f`, interrupting execution (we omit the syntax for handling faults). If a fault signal is thrown from inside a request-response body, the invoker is automatically notified of the fault [73].

Term *(assign)* stores the result of the evaluation of expression e in variable `x`. *(alias)* makes variable `x` an alias for variable `y`, i.e., accessing `x` will be equivalent to accessing `y`. Term **`nullProcess`** denotes the empty behaviour.

Jolie natively supports structured data manipulation. In Jolie's memory model the program state is a tree (with array nodes, see [71]), and every variable, say `x`, can be a *path* to a node of the memory tree. Paths are constructed through the dot operator; e.g., the following sequence of assignments

```
person.name = "John"; person.age = 42
```

would lead to a state containing a tree with root label `person`. For clarity, a corresponding XML representation would be:

```
<person> <name>John</name>
         <age>42</age>     </person>
```

Deployments. We introduce now the syntax for deployments (see [71] for a more complete presentation). The basic deployment primitives are *input ports*, denoted by IP, and *output ports*, denoted by OP, which respectively support input and output communications with other services. Input and output ports are dual concepts and their syntaxes are quite similar. Ports are based upon the three basic concepts of *location*, *protocol* and *interface*. Their (selected) syntax is reported in Figure 6.2. In the syntax of ports, Loc is a URI (Uniform Resource Identifier), defining the location of the port; $Proto$ is an identifier referring to the data protocol to use in the port, specifying how input/output messages through the port should be decoded/encoded; the iface_i's are references to the interfaces accessible through the port.

Jolie supports different locations and protocols. For instance, a valid Loc for accepting TCP/IP connections on TCP port 8000 would be `"socket://localhost:8000"`. Other supported locations are Unix sockets, Bluetooth communication channels, and local memory. Some supported instances of $Proto$ are `sodep` [60], `soap` [10], and `xmlrpc` [15].

The interfaces declared in a communication port define the operations accessible through it. Each interface defines a set of operations, pairing each with (i) the operation type (one-way or request-response) and (ii) the types of its carried messages. For example, the following interface

```
interface SumIface { RequestResponse: sum(SumT)(int) }
```

defines an interface `SumIface` with a request-response operation `sum`, which expects input messages of type `SumT` and returns messages of type **`int`** (integers). Data types for messages follow a tree-like structure; e.g., we could define `SumT` as follows:

```
type SumT:void { .x:int .y:int }
```

We can read the code above as: a message of type `SumT` is a tree with an empty root node (**`void`**) and two subnodes, `x` and `y`, that have both type **`int`**.

A Jolie example. We give an example of how to combine behaviour and deployment definitions, by showing a simple service defined in Jolie. The code follows:

```
type SumT:void { .x:int .y:int }
interface SumIface
  { RequestResponse: sum(SumT)(int) }

inputPort MyInput {
Location: "socket://localhost:8000"
Protocol: soap
Interfaces: SumIface
}

main {
  sum( req )( resp ) {
    resp = req.x + req.y
  }
}
```

Above, input port `MyInput` deploys the interface `SumIface` (and thus the `sum` operation) on TCP port 8000, waiting for TCP/IP socket connections by invokers using the `soap` protocol. The behavioural code in **`main`** defines a request-response input on operation `sum`. In this Chapter, we implicitly assume that all services are deployed with the **`concurrent`** execution modality for supporting multiple session executions, from [71] (which yields the semantics of concurrent process execution that we have formalised before in § 5.3). This means that whenever the first input of the behavioural definition of a service can receive a message, Jolie will spawn a dedicated process to execute the rest of the behaviour. This process will be equipped with a local variable state and will proceed in parallel to all the others. Therefore, in our example, whenever our service receives a request for operation `sum` it will spawn a new parallel process instance. The latter will enter into the body of `sum`, assign to variable `resp` the result of adding the subnodes `x` and `y` of the request message, and finally send back this result to the original invoker.

6.3 Extending Jolie to HTTP

We extend Jolie to support web applications by introducing a new protocol for communication ports, named `http`. The latter follows the HTTP protocol specifications and integrates

the Jolie message semantics to that of HTTP and its different content encodings.

6.3.1 Message transformation

The basic issue to address for integrating Jolie with the HTTP protocol is establishing how to transform HTTP messages in messages for the input and output primitives of Jolie and vice versa. Hereby we discuss primarily how our implementation manages request messages; response messages are similarly handled. The (abstract) structure of a *request message* in HTTP is:

$$Method\ Resource\ \texttt{HTTP}/Version\ Headers\ Body$$

Above, $Method$ specifies the action the client intends to perform and can be picked by a static set of keywords, such as `GET`, `PUT`, `POST`, etc. $Resource$ is a URI telling which resource the client is requesting. $Version$ is the HTTP protocol version of the message. $Headers$ include descriptive information (such as the encoding of the message body) or even configuration parameters that are supposed to be respected by the receiver (e.g., the wish to close the connection immediately after the response is sent, which our implementation handles automatically). Finally, $Body$ contains the content of the HTTP message.

A Jolie message is composed by an operation name and a (structured) value. Hence, we need to establish where to retrieve (or write) them in an HTTP message. For operations, we interpret the path part of the $Resource$ URI as the operation name. We have chosen not to use $Method$ for operations since it cannot assume user-defined names, as operations require. $Method$ can still be read and written by Jolie programs through a configuration parameter of our extension, described in § 6.3.2. The message value, instead, is obtained from the $Body$ part and the rest of the $Resource$ URI. We need the latter to access REST interfaces and to be able to decode *querystring* parameters as Jolie values.

An HTTP message content may be encoded in different formats. Our `http` extension handles querystrings, form encodings (simple and multipart), XML, JSON [7], and GWT-RPC[1] [4]. Programmers can use the `format` parameter (§ 6.3.2) to control the data format for encoding and decoding messages. For incoming request messages, if the `Content-Type` HTTP header is present then it is used to auto-detect the data format of $Body$. If a response is sent back from Jolie and the request format was JSON and GWT-RPC, then `http` defaults to the same format for encoding the response content. As an example of message translation, the HTTP message:

```
GET /sum?x=2&y=3 HTTP/1.1
```

would be interpreted as a Jolie message for operation `sum`. The querystring `x=2&y=3` would be translated to a structured value with subnodes `x` and `y`, containing respectively the strings "`2`" and "`3`".

[1] We have also developed a companion GWT-RPC client library, called `jolie-gwt`, for a more convenient access to web services written in Jolie by integrating with the standard GWT development tools. In our library, the operation name is encoded in the HTTP message content instead of the $Resource$ field, following the GWT specifications.

Automatic type casting. Querystrings and other common message formats used in web applications, such as HTML form encodings, do not carry type information. Instead, they simply carry string representations of values that could have been typed on the invoker's side. However, type information is necessary for supporting services such as the sum service in § 6.2, which specifically requires its input values to be integers. To cope with such cases, we introduce the notion of *automatic type casting*. Automatic type casting reads incoming messages that do not carry type information (such as querystrings or HTML forms) and tries to cast their content values to the types expected by the service interface for the message operation. As an example, consider the querystring `x=2&y=3` above. Since its HTTP message is a request for operation `sum`, the automatic type casting mechanism would retrieve the typing for the operation and see that nodes `x` and `y` should have type **`int`**. Therefore, it would try to re-interpret the strings "`2`" and "`3`" as integers before giving the message to the Jolie interpreter. Of course, type casting may fail; e.g., in `x=hello` the string `hello` cannot be cast to an integer for `x`. In such cases, our `http` protocol will send back a `TypeMismatch` fault to the invoker. The latter may catch the fault in its web user interface code.

6.3.2 Configuration Parameters

We augment the deployment syntax of Jolie to support *configuration parameters* for our `http` protocol. Specifically, these can be accessed through *(assign)* and *(alias)* instructions put aside the protocol declaration of a port. For instance, the following input port definition

```
inputPort MyInput {
  /* ... */
  Protocol: http {
    .default = "d"; .debug = true;
    .method -> m
  }
}
```

would set the **`default`** parameter to `"d"`, set the `debug` parameter to **`true`**, and bind the `method` parameter to the value of variable `m` in the current Jolie process instance.

We briefly describe some notable configuration parameters. All of them can be modified at runtime (using the standard Jolie constructs for dynamic port binding, from [71], which we omit here). Parameter **`default`** allows to mark an operation as a special fallback operation that will receive messages that cannot be handled by any operation defined in the interface of its service. Parameter `cookies` allows to store and retrieve data from browser cookies, by mapping cookie values in HTTP messages to subnodes in Jolie messages. Parameter `method` allows to read/write the $Method$ field for the latest received/next to send HTTP message through the port. Parameter `format` can be used to force the data format of output HTTP messages, such as `json` (for JSON), or `xml` (for XML). The parameter `alias` allows to map values inside a Jolie message to resource paths in the HTTP message, to support interactions with REST services. Parameter `redirect` gives access to the **`Location`** field in HTTP, allowing to redirect clients to other locations. The parameter `cacheControl` allows to send directives to the client on how the responses sent to it should be cached. Finally, parameter `debug` allows to print debug

messages on screen whenever an HTTP message is sent or received.

6.4 Web Programming with Jolie

In this section we discuss how `http` extension can be used to cover some useful web application patterns.

6.4.1 Modelling Web Servers

We first address how to program a web server for providing static content (e.g., the resources for a web user interface) to web clients.

The main challenge in dealing with modelling a web server is that, in Jolie and other service-oriented technologies such as WS-BPEL [79], a service interface is a statically defined set of operations. Differently, web servers make a dynamic set of resources available to clients: the code for implementing a web server does not change if its set of exposed resources (from, e.g., a part of a filesystem) is modified.

To deal with dynamic resource names, i.e., resource names that we do not know at design time, we need a means to bridge the static definition of interfaces to them. We address this issue by using our **`default`** operation parameter. The default operation is a special operation marked as a fallback in case a client sends a request message with an operation that is not statically defined by the service. Instead, the message is wrapped in the following data structure (we omit some subnodes not relevant for this discussion):

```
type DefaultOperationHttpRequest:void {
        .operation:string
        .data:undefined
}
```

where `operation` is the name of the operation that has been requested by the client and `data` is the data content of the message.

Parameter **`default`** allows us to model a simple web server easily: whenever we receive a request for the default operation, we try to find a file in the local filesystem that has the same name as the operation originally requested by the client. We have used this mechanism to implement Leonardo [9], a web server implementation written in pure Jolie. For clarity, here we report a simplified version [2]:

Listing 6.1: Leonardo Web Server (excerpt)

```
/* ... */

interface MyInterface {
RequestResponse:
  d( DefaultOperationHttpRequest )
   ( undefined )
}
```

[2] The entire implementation of Leonardo is made of about 80 LOCs and can be downloaded at [9]

```
inputPort HTTPInput {
Location: "socket://localhost:80/"
Protocol: http
  { .default = "d" /* ... */ }
Interfaces: MyInterface
}

main {
  d( req )( resp ) {
    /* ... */
    readFile@File( req.operation )( resp )
  }
}
```

Above, we have set the **default** parameter for the `http` protocol in input port `HTTPInput` to operation `d`. Therefore, when a message for an unhandled operation is received through input port `HTTPInput`, it will be managed by the implementation of operation `d`. The body of the latter invokes operation `readFile` of the `File` service from the Jolie standard library, which reads the file with the same name as the originally request operation (`req.operation`). Finally, the data read from the file (`resp`) is returned back to the client.

6.4.2 Multiparty Sessions

We present an implementation sketch of an extended version of the process-aware scenario mentioned in the Introduction, where a user can access a `createNews` operation for posting a news on a website after she has successfully logged in through a `login` operation. We remind the reader that our `http` protocol accepts invocations with different formats, so we will leave the code for the user interface unspecified; e.g., we could use AJAX calls with the JSON format for calling operation `login`, or the following HTML form:

```
<form action="login" method="POST">
  <input type="text" name="user"/>
  <input type="password" name="pwd"/>
  <input type="submit"/>
</form>
```

Our scenario will execute as follows. First, the user will download the web interface from our service implementation. Afterwards, she will call the `login` operation for authenticating. We will use an external `Authenticator` service for checking the user's credentials. If the authentication is not successful, we will interrupt the process by throwing a fault (which will also automatically notify the user of the error). Otherwise, if the authentication is successful we will wait for the user to invoke the `createNews` operation. When the latter is invoked, we will notify in parallel two external services of the news creation request: `Logger` and `Moderator`. The former is used to log the user's operation. The latter, instead, is an external service handling another web application for moderating news creation requests. Specifically, service `Moderator` (omitted here) is

responsible for showing the news creation request on a moderation list. Moderators can access the list through a web interface and get redirected to our service for approving or rejecting the news creation request.

A critical aspect of our implementation is the modelling of *sessions*, or conversations. Assume that, e.g., two users are logged in the service at the same time and, therefore, are supported by two separate process instances in our Jolie service. When a message for operation `createNews` arrives, how can we know if it is from the first user or the second? We address this issue by using the correlation mechanism described in Chapter 5. A correlation set specifies special variables that identify an internal service process from the others. In this example, we combine our correlation set mechanism with HTTP cookies, which are usually employed for storing session identifiers in web browsers. Our example will have two correlation sets consisting of one variable each, respectively `userSid` and `modSid`. We will use the first to identify calls from the user, and the second to identify messages from the moderator. Having two separate identifiers for our process instance is a fundamental aspect of *multiparty sessions*, such as the one in this example, for reasons of security; e.g., since we will not make `modSid` known to the user, she will be unable to (maliciously) impersonate the moderator.

We can finally show the code for our service:

Listing 6.2: A moderated news service

```
/* Types, Interfaces, Output ports, ... */

inputPort MyInput {
Location: "socket://localhost:8000/"
Protocol: http {
  .cookies.userSid = "userSid";
  .cookies.modSid = "modSid";
  .default = "d"
}
Interfaces: MyIface
}

cset { userSid: createNews.userSid }
cset { modSid:
  approve.modSid reject.modSid }

main {
  [ d( req )( resp ) ] { /* ... */ }

  [ login( cred )( r ) {
    check@Authenticator( cred )( ok );
    if ( ok ) {
      csets.userSid = new;
      r.userSid = csets.userSid
    } else { throw( AuthFailed ) }
  } ] {
```

```
    createNews( news );
    csets.modSid = new;
    { log@Logger( cred.username )
      | notify@Moderator( csets.modSid ) };
    [ approve() ] { /* ... */ }
    [ reject() ]  { /* ... */ }
  }
}
```

Above, we have reused the web server pattern from § 6.4.1 to provide the resources for the (omitted) web user interface to web browser clients. We combine that pattern with a process that starts with a request-response input on `login`, using an *(input choice)*. When `login` is invoked, a new process is started which immediately checks if the user's credentials are valid through an external `Authenticator` service. If they are valid (condition `ok`), then we instantiate the correlation variable **`csets`**`.userSid` (**`csets`** is a special keyword for accessing correlation variables in the behaviour) to a *fresh* value, given by primitive **`new`**; otherwise, we throw a fault `AuthFailed`, therefore notifying the client and interrupting the execution of the process. We return **`csets`**`.userSid` to the user through the response message for `login`. Observe that we configured `http` with `.cookies.userSid = "userSid"`. Hence, `r.userSid` will be encoded as a cookie in our HTTP response to the client. When the user's client will call our service on operation `createNews`, our cookie configuration for `userSid` will convert the cookie in the HTTP request to a subnode `userSid` inside the request message, which we will use for correlating with the correct process instance. After `createNews` is invoked, we instantiate our second correlation variable: `modSid`. Then, we use the parallel compositor `|` to notify the `Logger` and `Moderator` services in parallel. The latter is informed of the value for `modSid`, which we will expect as a cookie (per our `http` configuration) in incoming messages from the moderator's user interface. The cookie will be used to correlate with our process, which is finally waiting for a decision between the `approve` operation and the `reject` operation.

Observe that our two correlation set definitions (the **`cset`** blocks before **`main`**) specify that operation `createNews` can be invoked only through correlation variable `userSid`, whereas `accept` and `reject` can be accessed only through `modSid` [3], modelling our aforementioned security aspect.

A remarkable aspect of the combination between our `http` extension and Jolie is that we abstract from where correlation data is encoded in the HTTP message. Instead of using a cookie, the web user interface may also send the value for a correlation variable through a querystring (enabling *process-aware hyperlinks*), or inside the HTTP message content. Our extension transparently support these different methods without requiring specific configuration.

6.4.3 Multi-tiering

In § 6.4.2, we have used a single service to handle both the content serving and the process execution. However, usually it is more desirable to separate the service responsible

[3] We assume that operations are declared with request types with the same respective names.

for serving content (the web server) from the service responsible for process execution. Ideally, this separation of concerns should allow to perform changes in the web server (e.g., implementing sitemaps or cache optimisations) abstracting from the internal code of the process executors and vice versa. More in general, we want to stratify our service in different *tiers*, as in classical multi-tiered web architectures. We reach this objective by exploiting the *aggregation* mechanism [71]. Aggregation is a composition primitive where an input port exposes operations that are not implemented in its service behaviour, but are instead delegated to another external service. Using aggregation we can split the web server code and the process code from our news service in two separate services.

We show first the code for the service responsible for handling the news moderation process:

```
/* Types, Interfaces, Output ports, ... */

inputPort MyInput {
Location: "socket://localhost:8001/"
Protocol: soap
Interfaces: MyIface
}

cset { userSid: createNews.userSid }
cset { modSid:
  approve.modSid reject.modSid }

main {
  login( cred )( r ) { /* ... */ }
}
```

The code above is taken from Listing 6.2. We have changed input port `MyInput` to be deployed on a different location using the `soap` protocol and we have removed the code for handling content serving. The rest of the service code is unmodified (the body of input `login` is the same). Content service is moved to the following separate service:

```
/* Types, Interfaces, Output ports, ... */

outputPort News { /* ... */ }

inputPort WebInput {
Location: "socket://localhost:8000/"
Protocol: http {
  .cookies.userSid = "userSid";
  .cookies.modSid = "modSid";
  .default = "d"
}
Interfaces: ContentIface
Aggregates: News
}
```

```
main {
  d( req )( resp ) { /* ... */ }
}
```

The service above implements the web content server for our web application. `ContentIface` is an interface defining only operation `d`. Input port `WebInput` takes care of receiving HTTP messages from web clients and aggregates the news service through output port `News` (which points to input port `MyInput` of the news service). When a message is received, Jolie will check whether its operation is defined in the interface of `News`. If so, then the message will be transparently forwarded to the news service and the subsequent response from the latter will be given back to the client. Otherwise, it will be interpreted as an invocation to be handled through the default operation `d`.

Our `http` extension can be combined with aggregation also for handling Multi-Service Architectures, i.e. web architectures where a single web application interacts with multiple services. For example, we may build a web server that supports both the user interface for users and for news moderators. Then, some web clients running the user interfaces would need to access the processes inside service `News` while others would need to access service `Moderator`. We can allow web clients to access both through the same web server by adding `Moderator` to the list of aggregated output ports inside the web server:

```
Aggregates: News, Moderator
```

Remarkably, since all the invocations from the web client to the aggregated services pass through the web server, this programming methodology respects the Same Origin Policy by design.

6.5 Related Work

The frameworks most similar to ours are those for modelling business processes, such as WS-BPEL [79], WS-CDL [102], and YAWL [100]. Differently from our approach, these tools are integrated with web applications through third-party tools, increasing the overall complexity of the system. Some of the ideas presented in this Chapter (e.g., the **`default`** parameter for implementing web servers) may be easily applied to WS-BPEL, making our work a useful reference.

Other works offer tools for supporting the development of process-aware web applications. [92] presents a process-based approach to deal with user actions through web interfaces using EPML; like Jolie, EPML is formally specified and comes with an execution engine. JOpera comes with an integration layer for offering REST-based interfaces to business processes [84]. These solutions are formed by integrating separate modules for process modelling, computation, and system integration. In contrast, our framework addresses all these aspects using the same language. EPML can integrate with other languages to integrate user interfaces with process execution; we are currently investigating in a similar direction (see § 6.6.1, *Scaffolding of User Interfaces*).

Hop [96, 23] and GWT [4] are programming frameworks that deal with the programming of both the user interface and the server-side application logic using a single codebase, which gets then compiled in the code for the client interface and the services. Dif-

ferently, in this Chapter we do not deal with the generation of client code. Instead, we support the seamless integration of existing technologies (HTML, AJAX calls, JSON, ...) with our services. The client code compiled from GWT projects can be reused with our `http` extension, which is able to parse GWT requests. Hop and GWT do not support the process-aware modelling primitives offered by our framework, nor its ability to compose many services on a same HTTP communication port (through aggregation).

Our **`default`** configuration parameter for `http` allows a service implementation to catch and reply to invocations for operations that were not known at design time. The same aspect has been previously modelled through mobility mechanisms for names in process calculi, e.g. in [69, 94, 50]. However, our approach is less powerful because the cited approaches elevate the received operation names at the language level; e.g., a service may receive an operation name through a variable and then use the latter in *(input)* and *(output)* primitives as operations. This is not possible in our behavioural language, since operations in input and output statements are statically defined. We purposefully chose not to support this kind of mobility, since it would have made the definitions of Jolie interfaces change at runtime. This would break the basic assumption of statically defined operations, which is used in our typing system in Chapter 5, in the WSDL standard for Web Services [13], and in formal theories models for the verification of concurrent programming languages (e.g., session types [54], where selection labels can be thought of as operations).

6.6 Conclusions

We have presented a framework for the programming of process-aware web applications. Through examples, we have shown how our solution subsumes useful web design patterns and how it captures complex scenarios involving, e.g., multiparty sessions and the Same Origin Policy. Our `http` extension is open source and is included in the standard distribution of Jolie [8]. Remarkably, our integration is *seamless*, meaning that existing Jolie code can easily be ported to HTTP by changing only the **`Protocol`** part of its communication ports to `http` and its configuration language. An important consequence is that the programmer does not need to deal with the differences between the data formats employed in HTTP messages (e.g., form encodings, querystrings, JSON, ...), since they will all be translated to Jolie data structures. This also means that all the techniques developed for the verification and execution of Jolie programs (as the typing system in Chapter 5) can be transparently applied to the process-aware web application logic written in our framework.

Our framework has also been evaluated in the development of industrial products and is now used in production at italianaSoftware [6], a software development company that uses Jolie as reference programming language. For instance, the company's website [6] and Web Catalogue, a proprietary E-Commerce platform with a codebase of more than 400 services, use the framework and the programming patterns presented in this Chapter.

6.6.1 Future Work

Hereby, we discuss possible future extensions of our work.

Reversibility. A common problem in handling the interaction between a web user interface and a business process is that the user may decide to take a step back in the execution

flow (e.g., by pressing the "Back" button). This possibility must be manually taken into account in the design of the process, increasing its complexity. We plan to extend Jolie with *reversibility techniques* [62], which allow distributed processes to be reversed to previous states by transparently dealing with the required communications to the involved parties.

Scaffolding of User Interfaces. [4] We will develop a scaffolding tool for user interfaces, starting from the process structure of a service. Specifically, given a behaviour B in Jolie, it would be possible to automatically generate a user interface that follows the communication structure of the behaviour. This would be in line with the notions of *duality* formalised in [54, 52].

Behavioural analyses. Since our framework makes the process logic of a web application explicit, it would be possible to develop a tool for checking that the invocations performed by a web user interface written in, e.g., Javascript, match the structure of their corresponding Jolie service. We plan to investigate this possibility using solutions similar to the ones presented in [56, 45].

Declarative data validation. Our framework exploits the message data types declared in the interfaces of a Jolie service to *validate* the content of incoming messages from web user interfaces. We plan to extend this declarative support to data validation by introducing an assertion language for message types that can check more complex properties (e.g., integer ranges, regular expressions, ...).

Extensions to other web protocols. We will extend our work by adding support to new emerging protocols for web application communications, such as WebSocket [58] and SPDY [3].

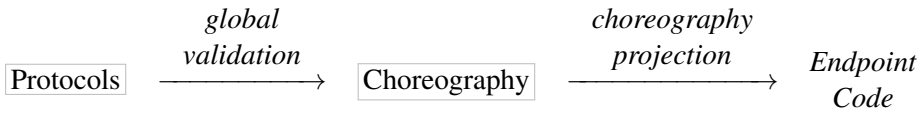
[4] The initial idea for this point comes originally from Claudio Guidi, during a private conversation.

CHAPTER 7

Chor: a Framework for Choreographic Programming

7.1 Introduction

In Chapter 2 we have presented the Choreography Calculus, a calculus in which systems can be developed by writing a choreography, verifying it against protocol specifications given as global types [55], and then projecting a correct endpoint implementation. This methodology is depicted below:



Building on the Choreography Calculus, in this Chapter we present Chor, a prototype framework for Choreographic Programming. Chor offers a programming language, based on choreographies, and an Integrated Development Environment (IDE) developed as an Eclipse plugin [40] for the writing of programs. All code is open source and can be consulted at the Chor website [34], along with an introductory video tutorial. Chor is available also on the Eclipse Marketplace, the official distribution channel for Eclipse-based software.

In the development methodology suggested with Chor, depicted in Figure 7.1, developers can first use our IDE to write protocol specifications and choreographies. The programmer is supported by on-the-fly verification which takes care of checking (i) the syntactic correctness of program terms and (ii) the type compliance of the choreography wrt the protocol specifications, using our typing discipline from § 2.4. Program errors are reported using syntax highlighting, allowing for an interactive programming experience.

Once the global program is completed, developers can automatically project it to an endpoint implementation. Endpoint implementations are given in the Jolie programming language [74]. Nevertheless, Chor is designed to be extended to multiple endpoint languages: potentially, each process in a choreography could be implemented with a different

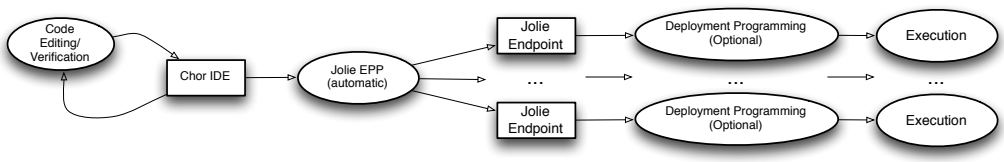


Figure 7.1: Chor, development methodology.

endpoint technology. We plan future extensions to support projecting endpoints to, e.g., Java, C#, or WS-BPEL [79].

Each Jolie endpoint program comes with its own deployment information, given as a term separated from the code implementing the behaviour of the projected process. This part can be optionally customised by the programmer, which can be useful for adapting the endpoint programs to some specific network or communication technology. For example, programs generated by Chor can be integrated with web applications by using the `http` extension presented in Chapter 6.

Finally, the Jolie endpoint programs can be executed with our extension of the Jolie interpreter presented in Chapter 5. As expected, they will implement the originating choreography.

7.1.1 Contributions

This Chapter provides the following contributions.

A programming language for choreographies. We present a language implementation for programming concurrent systems with choreographies (§ 7.2). Our language natively supports the development of multiparty sessions and their type checking against protocol specifications, following the model in Chapter 2. Programs in Chor are written in an IDE that supports on-the-fly type checking.

Endpoint Projection. A choreography program written in Chor can be automatically compiled to a set of Jolie programs that will implement the processes in the originating choreography (§ 7.3). We refer to this compilation procedure, as in the other Chapters, as Endpoint Projection (EPP). The key to our implementation of projection is using correlation sets (from Chapter 5) for supporting the execution of multiparty sessions.

Evaluation. We evaluate Chor with a series of real-world examples, e.g., authentication protocols and a use case from the Ocean Observatories Initiative (OOI) [80] (§ 7.4).

7.2 Language

In this section, we discuss the syntax of the Chor language.

7.2.1 Program structure

The syntax of programs is reported in Figure 7.2. A Chor program starts with the statement **`program`** `id`, which declares a name for the program with the identifier `id` (giving names to programs is reserved for future use in project management features). A program is then composed by two parts: a $Preamble$ and a $Body$.

The program preamble defines the protocols (* indicates that a nonterminal may be repeated) and public names that will be used in the choreography of the program. Each protocol definition is identified in the program by a name `g`, which is associated to a global type G. Public names, which are the shared names from our Choreography Calculus (see § 2.3), are used to start sessions in a choreography. Each public name, identified by a name `a`, is associated to a protocol `g` that the session started through `a` will be expected to follow.

Program	::=	**program** id; *Preamble* *Body*	*(program)*
Preamble	::=	*Protocol** *Public**	*(preamble)*
Protocol	::=	**protocol** g { *G* }	*(protocol)*
Public	::=	**public** a : g	*(public channel)*
Body	::=	*Def** **main** { *C* }	*(body)*

Figure 7.2: Chor, syntax of programs.

G	::=	A -> B : op(*U*) ; *G*	*(com)*
	|	A -> B : {op$_i$ (U_i) : G_i}$_{i \in I}$	*(branch)*
	|	g	*(call)*
	|	**end**	*(end)*
U	::=	**void** | **bool** | **string** | **int**	*(data type)*
	|	g@C	*(delegation type)*

Figure 7.3: Chor, syntax of protocols.

The program body gives a list of procedure definitions Def^* and a main procedure **main** containing a choreography C, which is the program entry-point for execution.

7.2.2 Protocols

Protocols are expressed in terms of global types, whose syntax is reported in Figure 7.3. A *(com)* global type A -> B : op(*U*) ; *G* specifies a communication from role A to role B through operation op, where the content of the message is required to have type U. Then, the global type proceeds as G. A *(branch)* global type is similar, but A can now choose among a set of different operations, each one with a corresponding carried type U_i and continuation G_i. In term *(call)*, we indicate that the global type proceeds as the global type associated to protocol g. Type **end** indicates termination of a protocol; it is automatically inserted by our implementation and it thus omitted in programs.

Remark 7.2.1 (Global types in Chor). Global types in Chor are a variant of those presented for the Choreography Calculus in § 2.4.1. Operations are simply the labels l used in § 2.4.1 for handling branchings; here, we chose to call labels operations to be nearer to the terminology of implemented languages for service-oriented computing. The only major difference wrt § 2.4.1 is that here we require the specification of an operation also for normal communications with no branching. This does not change the expressivity of global types in any way since we could, e.g., retain typical global types by adopting a standard operation name for non-branching communications. Our syntax allows us to give global types that will relate directly to our generated Jolie code, in which communications always specify an operation. □

Def	::=	**define** d($\widetilde{\mathtt{p}}$) ($\tilde{D}$) { C }	*(def)*
D	::=	k:g[p[A]]	*(def param)*
C	::=	η ; C	*(interaction)*
	|	τ ; C	*(local action)*
	|	**if** (e) @p C_1 **else** C_2	*(cond)*
	|	d($\widetilde{\mathtt{p}}$) ($\widetilde{\mathtt{k}}$)	*(call)*
	|	{ C }	*(block)*
	|	**end**	*(end)*
η	::=	p[A] **start** q[B] : a(k); C	*(start)*
	|	p.e -> q.x : op(k) ; C	*(com)*
	|	p -> q : op(k(k')) ; C	*(del)*
τ	::=	**local@**p(x = e)	*(assign)*
	|	**ask@**p(e, x)	*(ask)*
	|	**show@**p(e)	*(show)*

Figure 7.4: Chor, syntax of choreographies.

7.2.3 Choreographies

Procedure definitions and choreographies have similar syntax to that presented in the Choreography Calculus from Chapter 2; it is reported in Figure 7.4.

Procedures. Term *(def)* defines a procedure d with a body C, declaring the processes $\widetilde{\mathtt{p}}$ and the sessions $\widetilde{\mathtt{k}}$ that are used inside C. Each session is typed with a protocol g and the processes that have to implement it playing their respective roles in C.

Interactions. A choreography C defines the behaviour of a system of processes. In term *(start)*, the active processes $\widetilde{\mathtt{p}}$ start the (fresh) service processes $\widetilde{\mathtt{q}}$ by synchronising on public channel a to start a new session k; each process is annotated with the role A it plays in the session. In term *(com)*, the sender process p sends the evaluation of its local expression e to the receiver process q, which stores the received value in its local variable x, through operation op on session k; e and x can be omitted when unnecessary. Term *(del)* implements a session delegation: process p delegates to process q, through session k, its role in session k'.

Remark 7.2.2 (Process role annotations). Differently than in the Choreography Calculus in 2, Chor requires processes to be annotated with the role they play in a session only in session starts and the declaration of procedure parameters. Process roles are automatically inferred in all other terms. □

Local actions. We extend the Choreography Calculus from 2 with actions that can be performed locally by a process. Term *(assign)* implements a local variable assignment, where process p assigns the evaluation of its local expression e to its local variable x.

Terms *(ask)* and *(show)* are used for interacting with a user. In term *(ask)*, process p asks the user to input a value for variable x by displaying message e with a message dialog. In term *(show)*, process p shows the user a message e.

Other terms. In term *(cond)*, process p evaluates an internal condition e and decides whether the system should proceed as choreography C_1 or C_2. Term *(call)* implements a procedure call, passing the processes $\widetilde{p}$ and sessions $\widetilde{k}$ as parameters. Term *(block)* is the standard block construct, used for explicitly grouping statements. Term *(end)* denotes termination; as in global types, it is automatically used by our implementation and is omitted in programs.

Example 7.2.3 (Chor program example)**.** We exemplify the syntax of Chor by giving a simple program:

```
1  program simple;
2
3  protocol SimpleProtocol { C -> S: hi( string ) }
4
5  public a: SimpleProtocol
6
7  main
8  {
9    client[C] start server[S] : a( k );
10   ask@client( "[client] Message?", msg );
11   client.msg -> server.x : hi( k );
12   show@server( "[server] " + x )
13 }
```

Program simple above starts by declaring a protocol SimpleProtocol, in which role C (for client) sends a string to a role S (for server) through operation hi. In the choreography of the program, process code and a fresh service process server start a session k by synchronising on the public channel a. Process client then asks the user for an input message and stores it in its local variable msg, which is then sent to process server through operation hi on session k, implementing protocol SimpleProtocol. Finally, process server displays the received message on screen. □

7.3 Implementation

In this section we comment the major aspects of the implementation of Chor.

7.3.1 Chor IDE

Chor comes with an IDE (Integrated Development Environment) developed as an Eclipse [40] plugin based on the Xtext framework [104]. The IDE offers three main components: a code editor for Chor programs; an on-the-fly type checker; and an automated endpoint projection (EPP) implementation for obtaining executable code from a Chor program.

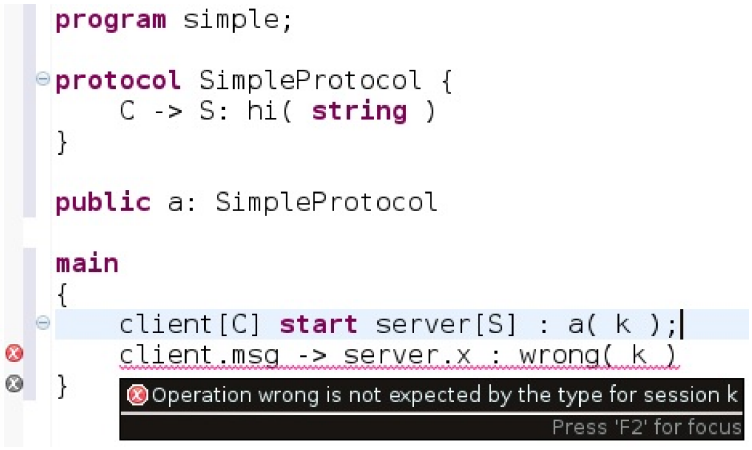


Figure 7.5: Chor, example of error reporting.

The code editor takes care of checking that a program follows our syntax and offers basic refactoring capabilities, e.g., the name of a protocol can be modified and then the change is automatically reflected in all references to the protocol in the rest of the program.

Our type checker is an implementation of the typing discipline in § 2.4. Since type checking is efficient (it has polynomial computational complexity) we run it on the fly, i.e., whenever the program is modified. The programmer is then interactively notified of mistakes, such as not implementing a protocol correctly in a session. This is visually represented by the typical red underlining of errors in terms. We give an example of error reporting in Figure 7.5, in which session `k` uses operation `wrong` instead of the correct operation `hi` specified by the protocol. In the figure, we also show the entire error message that the user can access through the standard Eclipse user interface.

The Endpoint Projection (EPP) procedure in Chor targets our extension of the Jolie language from Chapter 5. The choice of Jolie has several reasons: (i) Jolie offers constructs similar to those of our endpoint calculus, e.g., replicated services and input choices, making part of our EPP straightforward; (ii) the support for programming multiparty sessions based on correlation sets, which we used for implementing the sessions programmed in Chor; and (iii) Jolie supports a wide range of compatibility with other technologies.

7.3.2 Deployment

By default, our endpoint programs will operate on top of TCP/IP sockets. However, since Jolie also supports other communication technologies – e.g. local memory IPC and Bluetooth – and data formats – e.g. HTTP (cf. Chapter 6) and SOAP [10] – programmers may customise deployment information of each endpoint. Hence, some endpoints may communicate over, e.g., HTTP, while others, e.g., using fast binary data formats. Additionally, different endpoints may be deployed in different machines and/or networks, giving developers freedom on choosing the degree of distribution of the system, from a single machine (e.g., multi-core systems) to a completely distributed setting (one endpoint program per machine).

Notably, customising the deployment of an endpoint program does not necessarily require updating the code of the others. Supporting this flexibility has required a careful

implementation of session starts (rule $\lfloor {}^{\mathrm{P}} | {}_{\textsc{Start}} \rceil$ from our endpoint model in § 2.5.1), which are coordinated by special "start services". The (endpoint projections of the) active processes willing to start a session contact the appropriate start service. Then, the start service spawns the (projections of the) service processes by calling the external services that implement them. In such message exchanges, which follow a variant of the standard Two-Phase Commit protocol [47], each endpoint informs the start service of the *binding information* (e.g., IP address and data format) on which the endpoint can be reached. Finally, the start service informs all participants about all necessary bindings, so that each party can dynamically update its references to the others (e.g., socket, bluetooth, or local inter-process connections).

Another key feature is that our implementation of message queues for sessions is based on the correlation mechanism developed in Chapter 5. Specifically, for each session/role pair that a process will play during execution we generate a corresponding correlation set. We have extended the implementation of Jolie to handle a separate message queue for each different correlation set, so to handle messages in different sessions concurrently as required by our endpoint model in § 2.5.1. Afterwards, the programmer can customise correlation for each deployment artifact. For instance, some processes may identify sessions using HTTP cookies (using our configuration parameters for HTTP in Jolie from Chapter 6), while others may use SOAP headers (as in the WS-Addressing specifications [11]).

Example 7.3.1 (Endpoint Projection in Chor)**.** We give an example of EPP by reporting a snippet of the code generated for process `server` from Example 7.2.3:

```
1  main
2  {
3    _start();
4    csets.tid = new;
5    _myRef.binding << global.inputPorts.MyInputPort;
6    _myRef.tid = csets.tid;
7    _start_S@a(_myRef)(_sessionDescriptor.k);
8    k_C << _sessionDescriptor.k.C.binding;
9    hi(x);
10   showMessageDialog@SwingUI("[server] " + x)()
11 }
```

The Jolie code above for process `server` waits to be started by receiving an input on operation `_start` (which acts as replicated). This starts the commit protocol for creating session `k`. In Lines 4–6, the process initialises its correlation value for the session (`csets.tid`), and then stores its binding information (e.g., its IP address) and correlation value for the session in variable `_myRef`. In Line 7, the process completes the session start protocol by sending its binding and correlation information to the service responsible for synchronising the processes involved in the creation of the session (service `a`). Still in Line 7, we receive in response a session descriptor for session `k`, which contains the binding information for reaching the other processes in the session. In Lines 9–10, we receive the message on operation `hi` from the client and display it on screen as indicated by the choreography. □

7.3.3 Delegation

Session delegation is a nontrivial mechanism at the level of endpoint implementation. The main concern lies in updating channel references (bindings). For instance, assume that a session k has some process participants, say p and $\widetilde{q}$. Suppose now that p delegates its role in k to another process r through a different session. In such a situation, all the processes in $\widetilde{q}$ need their external references to be updated for reaching r instead of p when communicating with the session/role pair delegated by p. In our formal endpoint model (see § 2.5.1) this necessity is completely abstracted away by the synchronisations on the centralised message queues (one per session). However, in our implementation of Chor, message queues are completely distributed: each process owns a message queue which must be reached explicitly by other process, following our model for correlation-based sessions from Chapter 5. In [57] the authors present a survey of possible solutions to this problem in asynchronous scenarios[1]. The main challenge is that the processes in $\widetilde{q}$ may send messages to process p before getting notified of the delegation; in [57], this issue is solved by making process p resending these messages to process r, adding extra communications. In our EPP implementation, instead, each process in $\widetilde{q}$ knows when the delegation will happen since we have that information from the originating choreography. Hence, we program the processes in $\widetilde{q}$ to wait for receiving the updated binding information for reaching process r before proceeding in the session, guaranteeing that we are always in the optimal case where no messages are sent to the wrong recipient.

7.4 Examples

In this section, we discuss some examples that we have implemented using the Chor language. Specifically, our examples below show how to deal with five typical aspects of distributed systems: multiparty sessions, session interleaving, service selection, delegation, and recursive protocols. More examples, including the implementations of the examples in § 2.6, are available at the Chor website [34].

7.4.1 Multiparty Sessions

We show how to address multiparty sessions by implementing a protocol inspired by the OpenID specifications for distributed authentication [81]:

```
1 program openid;
2
3 protocol OpenID {
4   U -> RP: username( string );
5   RP -> IP: username( string );
6   U -> IP: password( string );
7   IP -> RP: {
8   ok( void );
9     RP -> U: ok( void ),
```

[1]The work in [57] actually deals with binary sessions, not multiparty as in here, but the main arguments presented therein remain valid also in the multiparty case.

```
10    fail(string);
11      RP -> U: fail( string )
12    }
13  }
14
15  public publicOpenID: OpenID
16
17  main
18  {
19    rp[RP], u[U] start ip[IP]: publicOpenID( k );
20
21    ask@u( "[u] Insert Username", user );
22
23    u.user -> rp.user: username( k );
24    rp.user -> ip.username: username( k );
25
26    ask@u( "[u] Insert Password", pwd );
27
28    u.pwd -> ip.password: password( k );
29
30    ask@ip(
31      "[ip] Accept username '" + username +
32      "' and password '" + password + "' ?",
33      accept
34    );
35    if (accept == "yes")@ip {
36      ip -> rp: ok( k );
37      rp -> u: ok( k );
38      show@u( "[u] Authentication successful!" )
39    } else {
40      ip."Invalid credentials" -> rp.error: fail( k );
41      rp.error -> u.error: fail( k );
42      show@u( "[u] " + error )
43    }
44  }
```

In the code above, we start by declaring a protocol `OpenID` where a user `U` wants to authenticate to a relying party `RP` using an external identity provider `IP`. The user starts by sending her username to `RP`, which forwards it to `IP`; then, the user sends her password to `IP`. In Lines 7–12, the identity provider `IP` selects either branch `ok` or `fail` on the relying party, which terminates the protocol by forwarding the choice to the user.

We implement the protocol in our choreography with the session `k` and the three processes `rp`, `u`, and `ip`. Lines 21–28 implement Lines 4–6 in the protocol, by asking information to the user and then communicating it. In Line 30, the `ip` asks whether the presented username/password credentials are valid; in Lines 35–43, the choice is then forwarded to processes `rp` and `u`, completing the protocol.

7.4.2 Session Interleaving

In the example below, we give an implementation of a system where two sessions are interleaved to reach a common goal.

```
1  program buyerseller;
2
3  protocol BuyerSeller {
4    Buyer -> Seller: buy( string );
5    Seller -> Buyer: price( int );
6    Buyer -> Seller: {
7      ok(void),
8      abort(void)
9    }
10 }
11
12 protocol AskUser {
13   Application -> User: question( string );
14   User -> Application: {
15     yes( void ),
16     no( void ),
17     maybe( void )
18   }
19 }
20
21 public a : BuyerSeller
22 public b : AskUser
23
24 main
25 {
26   buyer[Buyer] start seller[Seller] : a( k );
27   buyer."Coffee" -> seller.product : buy( k );
28   seller.100 -> buyer.price : price( k );
29
30   buyer[Application] start user[User]: b( k2 );
31   buyer.( "Can I pay " + price + "?" )
32     -> user.question: question( k2 );
33
34   local@user( s = question );
35   ask@user( s, answer );
36
37   if (answer == "yes")@user {
38     user -> buyer: yes( k2 );
39     buyer -> seller: ok( k );
40     show@seller( "[seller] Product sent!" )
41   } else {
42     user -> buyer: no( k2 );
```

```
43      buyer -> seller: abort( k );
44      show@seller( "[seller] Transaction aborted!" )
45    }
46  }
```

Above, we specify two protocols: `BuyerSeller` and `AskUser`. In protocol `BuyerSeller`, a `Buyer` asks a `Seller` for the price of a product using operation `buy`; the `Seller` replies with operation `price`, and then the `Buyer` finally notifies the `Seller` of whether she wishes to proceed with the purchase (operation `ok`) or not (operation `abort`). Protocol `AskUser` is very simple: an `Application` asks a `User` a question and then waits for an answer from the `User`, which can be `yes`, `no`, or `maybe`.

The choreography starts by instantiating protocol `BuyerSeller` as session `k`, on which the buyer asks for some coffee and the seller replies with the price 100. Now, in Line 30, the buyer starts another session `k2` with another process `user` by playing role `Application`. The buyer uses session `k2` to asks whether the user is willing to pay the price required by the seller. Depending on the user's choice, buyer completes session `k` accordingly (Lines 37–45).

Observe that in session `k2` we are not implementing option `maybe`, since we do not need it. Our type checker will still accept the choreography as safe, following our typing discipline from § 2.4 (see § 2.5.6 for details).

7.4.3 Service Selection and Delegation

We report an example from the Ocean Observatories Initiative (OOI) [80], in which a user connects to an instrument for reading environmental data through a service responsible for authorising user commands.

```
1  program instrument;
2
3  protocol AuthCommand {
4    U -> A: username( string );
5    U -> A: password( string );
6    A -> U: {
7    valid(Connect@U),
8    fail(void)
9    }
10 }
11
12 protocol Connect {
13   U -> R: connect( string );
14   R -> U: {
15     ok( ExecCommand@C ),
16     unavailable( void )
17   }
18 }
19
20 protocol ExecCommand {
```

```
21    C -> I: {
22    readTemperature(void);
23      I -> C: result(string),
24    readPressure(void);
25      I -> C: result(string)
26    }
27  }
28
29  public a : AuthCommand
30  public b : Connect
31  public instrument1 : ExecCommand
32  public instrument2 : ExecCommand
33
34  define findAndExec( u, r )( connect[Connect: u[U], r[R]] )
35  {
36    ask@u( "[u] What instrument
37      do you want to connect to? (inst1/inst2)",
38      name );
39    u.name -> r.name: connect( connect );
40    if ( name == "inst1" )@r {
41      r[C] start i[I]: instrument1( exec );
42      r -> u: ok( connect( exec ) );
43      u -> i: readTemperature( exec );
44      i."28 C" -> u.temp: result( exec )
45    } else if ( name == "inst2" )@r {
46      r[C] start i[I]: instrument2( exec );
47      r -> u: ok( connect( exec ) );
48      u -> i: readTemperature( exec );
49      i."2 C" -> u.temp: result( exec )
50    } else {
51      r -> u: unavailable( connect )
52    }
53  }
54
55  main
56  {
57    u[U] start a[A]: a( auth );
58    ask@u( "[u] Insert username", username );
59    ask@u( "[u] Insert password", pwd );
60    u.username -> a.username: username( auth );
61    u.pwd -> a.pwd: password( auth );
62    ask@a( "[a] Confirm credentials? (yes/no) : "
63      + username + " : " + password,
64      confirm );
65    if ( confirm == "yes" )@a {
66      a[U] start r[R]: b( connect );
```

```
67      a -> u: valid( auth( connect ) );
68      findAndExec( u, r )( connect )
69    } else {
70      a -> u: fail( auth )
71    }
72  }
```

We have three protocols. In protocol `AuthCommand`, a user `U` sends her credentials to an authoriser `A` which decides whether the credentials are valid or not. If the credentials are valid, then the authoriser delegates to the user access to a session for connecting to an instrument. The latter will follow protocol `Connect`, in which the user asks a registry `R` for a connection to a specific instrument. If the instrument is available, the registry delegates to the user a session for communicating with the instrument she requested, which follows protocol `ExecCommand`. Finally, in protocol `ExecCommand`, a client `C` (which will be the user in our choreography) asks an instrument `I` for a temperature or a pressure reading.

In the choreography, Lines 57–72 implement protocol `AuthCommand`. If successful, the authoriser starts a session `connect` with the registry and delegates it to the user; the choreography then proceeds by calling procedure `findAndExec`. In the procedure, the user selects an instrument and asks the registry for being connected to it. If the instrument is known by the registry, the latter creates a session with the instrument and then delegates it to the user. Finally, the user uses the instrument for getting a reading.

7.4.4 Recursive Protocols

Our last example is about recursive protocols. Specifically, we define a protocol for streaming some packets from a server to a client.

```
1  program stream;
2
3  protocol StreamingProtocol {
4    S -> C: packet( string );
5    S -> C: {
6      again( void ); StreamingProtocol,
7      stop( void )
8    }
9  }
10
11 public a : StreamingProtocol
12
13 define doStreaming
14      ( c, s )
15      ( stream[StreamingProtocol: c[C], s[S]] )
16 {
17   ask@s( "[s] Input data to send for packet number " + i,
18     data );
19
```

```
20    s.data -> c.data: packet( stream );
21    local@c( result = result + data );
22    local@s( i = i + 1 );
23    if ( i < nPackets )@s {
24      s -> c: again( stream );
25      doStreaming( c, s )( stream )
26    } else {
27      s -> c: stop( stream );
28      show@c( "[c] Received data: " + result )
29    }
30  }
31
32  main
33  {
34    c[C] start s[S]: a( k );
35    ask@s( "[s] How many packets do you want to send?",
36      nPackets );
37    local@s( i = 0 );
38
39    doStreaming( c, s )( k )
40  }
```

Above, we have only one protocol `StreamingProtocol`. In the protocol, the server `S` sends a packet to the client `C` and then informs the client on whether there are more packets or not. In the first case (operation `again`), the protocol continues; otherwise (operation `stop`), it terminates immediately.

In the choreography, a client process `c` starts a server process `s` to implement the protocol on session `k`. The server then establishes how many packets to send and the choreography proceeds by calling procedure `doStreaming`. In the procedure, the server gets some data to send to the client and sends it. The client aggregates the received data in its local variable `result`. Thereafter, the server checks whether there are more packets to send. If so, the server informs the client of continuing and the procedure recurs by invoking itself; otherwise, the server informs the client that there are no more packets and the choreography terminates.

7.5 Related Work

From a language perspective, Chor is essentially an implementation of the Choreography Calculus presented in Chapter 2, so we refer the reader to § 2.7 for a discussion on related work about the language design.

On the side of tools for choreographic programming, the works nearest to Chor are WS-CDL [102] and BPMN [1]. The main difference is that Chor is strongly typed and supports the type checking of sessions against formally-defined protocols, whereas WS-CDL and BPMN do not come with this kind of typing discipline and are thus more error-prone. Moreover, the semantics of both languages is informally specified, making their interpretation potentially unclear, whereas Chor follows the formal semantics given in Chapter 2.

Consequently, it is not possible to formally reason about safety properties on WS-CDL and BPMN, whereas the design of Chor is based on the foundations investigated in Chapter 2. However, regarding EPP, Chor targets Jolie code instead of the endpoint model given for the Choreography Calculus in § 2.5. An interesting future work is therefore to formally prove that adopting the Jolie model given in Chapter 5 for endpoint programs does not alter the safety properties guaranteed in § 2.5.

7.6 Conclusions

We presented Chor, a prototype framework for Choreographic Programming. Through examples, we evaluated Chor against a series of use cases and shown that it is already expressive enough to handle, e.g., the interleaving of protocols, service selection, and session delegation.

7.6.1 Future Work

We discuss some directions for future work on Chor.

Language Extensions. Chor is an implementation of the model in Chapter 2, the Choreography Calculus, which does not support all the features presented in this dissertation in the later Chapters. We plan to extend Chor to handle the composition of choreographies and round-trip development as formalised, respectively, in Chapters 3 and 4.

Global Deployment. Chor projects Jolie programs with a default deployment configuration that can be edited afterwards for each endpoint. This may be inconvenient for the programmer, since a choreography may describe many participants. Therefore, we plan to introduce a global deployment language for choreographies, from which the deployment configuration of each endpoint could be automatically generated. We envision that this extension could be relevant for facilitating checks on the correctness of a deployment configuration of a system, e.g., consistent I/O connections.

Scaffolding. Since in our model a protocol and a session behaviour in a choreography have similar structures, we could implement a scaffolding tool in our IDE that, given a protocol, would generate a prototype choreography with "dummy" data that implements it. Then, programmers would refine and interleave different prototypes to obtain the desired behaviour.

Local Code. The local actions in Chor are still very limited and are only meant to exemplify the idea of introducing internal computation at endpoints in choreographies. We plan to investigate how to embed an entire language for internal actions, such as Jolie or Java, so to reach a more general result. This would also be helpful for reusing already existing libraries from other frameworks in choreographies.

Part III

Appendices

APPENDIX A

Global Programming: Additional Material

A.1 Proof of Theorem 2.4.2

We first give some auxiliary results.

Below, we report the Substitution Lemma. Note that we do not require substitution for session channels, processes and recursion variables [101].

Lemma A.1.1 (Substitution)**.** *Assume* $\Gamma; \Sigma \vdash C \triangleright \Delta$*; then,* $\Gamma \vdash x@\mathtt{p} : S$ *and* $\Gamma \vdash v : S$ *implies* $\Gamma; \Sigma \vdash C[v/x@\mathtt{p}] \triangleright \Delta$.

Proof. Immediate by induction on the typing rules reported in Figure 2.10. □

Typing is preserved by structural congruence.

Lemma A.1.2 (Subject Congruence)**.** *Assume* $\Gamma; \Sigma \vdash C \triangleright \Delta$*; then,* $C \equiv C'$ *implies* $\Gamma; \Sigma \vdash C' \triangleright \Delta$ *(up to* α*-renaming).*

Proof. The proof is standard, by induction on the rules defining the structural congruence relation $\equiv$ reported in Figure 2.4. □

The following Lemma allows us to move session ownership typings to asynchrony environments.

Lemma A.1.3 (Asynchronous delegation)**.** $\Gamma, \mathtt{p} : k[\mathtt{A}]; \Sigma \setminus \mathtt{p} : k[\mathtt{A}] \vdash C \triangleright \Delta$ *and* $\mathtt{q} : k[\mathtt{A}] \notin \Sigma$ *implies* $\Gamma, \mathtt{p} : k[\mathtt{A}]; \Sigma, \mathtt{q} : k[\mathtt{A}] \vdash C \triangleright \Delta$.

Proof. Immediate from the typing rules reported in Figure 2.10. □

We can now prove Subject Reduction, by establishing the stronger result below. In the following, $\Gamma[\mathtt{p} \mapsto k[\mathtt{A}]]$ is Γ where all ownerships $k[\mathtt{A}]$ have been erased and then ownership $\mathtt{p} : k[\mathtt{A}]$ is added.

Theorem A.1.1 (Subject Reduction)**.** *Let* $\Gamma; \Sigma \vdash C \triangleright \Delta$*; then,* $C \xrightarrow{\lambda} C'$ *implies* $\Gamma'; \Sigma' \vdash C' \triangleright \Delta'$ *for some* Γ'*,* Σ' *and* Δ' *such that*

(i) if $\lambda \in \{\tau@\mathtt{p}, (start)\}$ *then* $\Gamma = \Gamma'$*,* $\Sigma = \Sigma'$ *and* $\Delta \simeq_{\mathcal{G}} \Delta'$*;*

(ii) if $\lambda = \mathtt{p}[\mathtt{A}]\ \texttt{->}\ \mathtt{q}[\mathtt{B}] : k\langle k'[\mathtt{C}]\rangle$ *and* $C \xrightarrow{\lambda} C'$ *is by* $\lceil {}^{\mathrm{C}}|_{\mathrm{ACT}} \rceil$ *then* $\Gamma' = \Gamma[\mathtt{q} \mapsto k'[\mathtt{C}]]$*,* $\Sigma' = \Sigma$ *and* $\Delta \xrightarrow{k:\alpha} \Delta'$ *such that* $\Gamma; \Sigma \vdash \lambda \triangleright k : \alpha$*;*

(iii) if $\lambda = \mathtt{r}[\mathtt{D}] \texttt{ -> } \mathtt{s}[\mathtt{E}] : k''\langle k'''[\mathtt{F}]\rangle$ *and* $C \xrightarrow{\lambda} C'$ *is by* $\lfloor^{\mathrm{C}}|_{\textsc{Async}}\rceil$ *then* $\Gamma' = \Gamma[\mathtt{s} \mapsto k'''[\mathtt{F}]]$, $\Sigma' = \Sigma, \mathtt{r} : k'''[\mathtt{F}]$ *and* $\Delta \xrightarrow{k:\alpha} \Delta'$ *such that* $\Gamma; \Sigma \vdash \lambda \triangleright k : \alpha$*;*

(iv) otherwise, $\Gamma = \Gamma'$, $\Sigma = \Sigma'$ *and* $\Delta \xrightarrow{k:\alpha} \Delta'$ *such that* $\Gamma; \Sigma \vdash \lambda \triangleright k : \alpha$.

Proof. The proof is by induction on the derivation of $C \xrightarrow{\lambda} C'$.

- **Case** $\lfloor^{\mathrm{C}}|_{\textsc{Act}}\rceil$. The case is:

$$\frac{\eta \in \{(sel), (del), (start)\} \quad \tilde{r} = \mathsf{bn}(\eta)}{C \;=\; \eta; C_1 \xrightarrow{\eta} (\boldsymbol{\nu}\tilde{r})\, C_1 \;=\; C'} \lfloor^{\mathrm{C}}|_{\textsc{Act}}\rceil$$

  We have three subcases, depending on the form of η.

  - **Case** $\eta = \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{q}[\mathtt{B}] : k[l_j]$. Since C is well-typed, we know that:

$$\frac{\Gamma; \Sigma \vdash \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{q}[\mathtt{B}] : k \quad \Gamma; \Sigma \vdash C' \triangleright \Delta'', k : G_j \quad j \in I}{\Gamma; \Sigma \vdash \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{q}[\mathtt{B}] : k[l_j]; C' \triangleright \Delta'', k : \mathtt{A} \texttt{ -> } \mathtt{B} : \{l_i : G_i\}_{i \in I}} \lfloor^{\mathrm{T}}|_{\textsc{Sel}}\rceil$$

    Hence C' is also well-typed by the premise of $\lfloor^{\mathrm{T}}|_{\textsc{Sel}}\rceil$. Moreover, we can see that:

$$\Delta = \Delta'', k : \mathtt{A} \texttt{ -> } \mathtt{B} : \{l_i : G_i\}_{i \in I} \xrightarrow{k:\mathtt{A} \texttt{ -> } \mathtt{B}:[l_j]} \Delta'', k : G_j = \Delta'$$

    The thesis follows now by rule $\lfloor^{\mathrm{L}}|_{\textsc{Sel}}\rceil$:

$$\frac{\Gamma; \Sigma \vdash \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{q}[\mathtt{B}] : k \quad j \in I}{\Gamma; \Sigma \vdash \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{q}[\mathtt{B}] : k[l_j] \triangleright k : \mathtt{A} \texttt{ -> } \mathtt{B} : [l_j]} \lfloor^{\mathrm{L}}|_{\textsc{Sel}}\rceil$$

  - **Case** $\eta = \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{q}[\mathtt{B}] : k\langle k'[\mathtt{C}]\rangle$. Since C is well-typed, we know that:

$$\frac{\begin{array}{c}\Gamma''; \Sigma \vdash \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{q}[\mathtt{B}] : k \quad \mathtt{p} : k'[\mathtt{C}] \notin \Sigma \quad G' \simeq_{\mathcal{G}} G'' \\ \Gamma'', \mathtt{q} : k'[\mathtt{C}]; \Sigma \setminus \mathtt{q} : k'[\mathtt{C}] \vdash C' \triangleright \Delta'', k : G, k' : G'\end{array}}{\Gamma'', \mathtt{p} : k'[\mathtt{C}]; \Sigma \vdash \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{q}[\mathtt{B}] : k\langle k'[\mathtt{C}]\rangle; C' \triangleright \Delta'', k : \mathtt{A} \texttt{ -> } \mathtt{B} : \langle G''@\mathtt{C}\rangle; G, k' : G'} \lfloor^{\mathrm{T}}|_{\textsc{Del}}\rceil$$

    where $\Gamma = \Gamma'', \mathtt{p} : k'[\mathtt{C}]$. Hence, C' is also well-typed. Moreover, we can derive:

$$\Delta = \Delta'', k : \mathtt{A} \texttt{ -> } \mathtt{B} : \langle G''@\mathtt{C}\rangle; G, k' : G' \xrightarrow{k:\mathtt{A} \texttt{ -> } \mathtt{B}:\langle G''@\mathtt{C}\rangle} \Delta'', k : G, k' : G' = \Delta'$$

    The thesis follows by rule $\lfloor^{\mathrm{L}}|_{\textsc{Del}}\rceil$:

$$\frac{\Gamma; \Sigma \vdash \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{q}[\mathtt{B}] : k \quad \Gamma \vdash \mathtt{p} : k'[\mathtt{C}]}{\Gamma; \Sigma \vdash \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{q}[\mathtt{B}] : k\langle k'[\mathtt{C}]\rangle \triangleright k : \mathtt{A} \texttt{ -> } \mathtt{B} : \langle G''@\mathtt{C}\rangle} \lfloor^{\mathrm{L}}|_{\textsc{Del}}\rceil$$

  - **Case** $\eta = \widetilde{\mathtt{p}[\mathtt{A}]}\ \mathbf{start}\ \widetilde{\mathtt{q}[\mathtt{B}]} : a(k)$. We know that $\tilde{r} = k, \tilde{\mathtt{q}}$. Also, since C is well-typed we know that:

$$\frac{\begin{array}{c}\Gamma; \Sigma \vdash a : G\langle \tilde{\mathtt{A}} | \tilde{\mathtt{B}}\rangle \quad G \simeq_{\mathcal{G}} G' \quad \tilde{\mathtt{q}} \notin \Gamma; \Sigma \\ \Gamma, \mathsf{init}(\{\widetilde{\mathtt{p}[\mathtt{A}]}, \widetilde{\mathtt{q}[\mathtt{B}]}\}, k); \Sigma \vdash C_1 \triangleright \Delta, k : G'\end{array}}{\Gamma; \Sigma \vdash \widetilde{\mathtt{p}[\mathtt{A}]}\ \mathbf{start}\ \widetilde{\mathtt{q}[\mathtt{B}]} : a(k); C_1 \triangleright \Delta} \lfloor^{\mathrm{T}}|_{\textsc{Start}}\rceil$$

Let $\tilde{\mathsf{q}} = \mathsf{q}_1, \ldots, \mathsf{q}_n$. Since $\tilde{\mathsf{q}} \notin \Gamma; \Sigma$, we can easily obtain the thesis by applying $\lfloor^{\mathrm{T}}|_{\mathrm{RES}}\rceil$ n times. Formally:

$$\frac{\begin{array}{c}\Gamma, \mathsf{init}(\{\widetilde{\mathsf{p}[\mathtt{A}]}, \widetilde{\mathsf{q}[\mathtt{B}]}\}, k); \Sigma \vdash C_1 \triangleright \Delta \\ \vdots \ \lfloor^{\mathrm{T}}|_{\mathrm{RES}}\rceil \ n \text{ times} \\ \Gamma \setminus \tilde{\mathsf{q}}; \Sigma \vdash (\boldsymbol{\nu}\tilde{\mathsf{q}})\, C_1 \triangleright \Delta\end{array}}{\Gamma \setminus \tilde{r}; \Sigma \vdash (\boldsymbol{\nu}\tilde{r})\, C_1 \triangleright \Delta \setminus k} \lfloor^{\mathrm{T}}|_{\mathrm{RES}}\rceil$$

- **Case** $\lfloor^{\mathrm{C}}|_{\mathrm{COM}}\rceil$. The case is:

$$\frac{\eta = \mathsf{p}[\mathtt{A}].e \texttt{ -> } \mathsf{q}[\mathtt{B}].x : k \quad e \downarrow v}{C = \ \eta; C_1 \xrightarrow{\eta[v/e]} C_1[v/x@\mathsf{q}] \ = C'} \lfloor^{\mathrm{C}}|_{\mathrm{COM}}\rceil$$

Since C is well-typed, we know that:

$$\frac{\Gamma; \Sigma \vdash \mathsf{p}[\mathtt{A}] \texttt{ -> } \mathsf{q}[\mathtt{B}] : k \quad \Gamma \vdash e@\mathsf{p} : S \quad \Gamma, x@\mathsf{q} : S; \Sigma \vdash C_1 \triangleright \Delta'', k : G}{\Gamma; \Sigma \vdash \mathsf{p}[\mathtt{A}].e \texttt{ -> } \mathsf{q}[\mathtt{B}].x : k; C_1 \triangleright \Delta'', k : \mathtt{A} \texttt{ -> } \mathtt{B} : \langle S \rangle; G} \lfloor^{\mathrm{T}}|_{\mathrm{COM}}\rceil$$

Hence, by the Substitution Lemma A.1.1 we know that C' is also well-typed. Moreover, we can derive:

$$\Delta = \Delta'', k : \mathtt{A} \texttt{ -> } \mathtt{B} : \langle S \rangle; G \xrightarrow{k:\mathtt{A} \texttt{ -> } \mathtt{B}:\langle S \rangle} \Delta'', k : G = \Delta'$$

The thesis follows by rule $\lfloor^{\mathrm{L}}|_{\mathrm{COM}}\rceil$:

$$\frac{\Gamma; \Sigma \vdash \mathsf{p}[\mathtt{A}] \texttt{ -> } \mathsf{q}[\mathtt{B}] : k \quad \Gamma \vdash v : S \quad \Gamma \vdash x@\mathsf{q} : S}{\Gamma; \Sigma \vdash \mathsf{p}[\mathtt{A}].v \texttt{ -> } \mathsf{q}[\mathtt{B}].x : k \triangleright k : \mathtt{A} \texttt{ -> } \mathtt{B} : \langle S \rangle} \lfloor^{\mathrm{L}}|_{\mathrm{COM}}\rceil$$

- **Case** $\lfloor^{\mathrm{C}}|_{\mathrm{ASYNC}}\rceil$. The case is:

$$\frac{\begin{array}{lll} C_1 \xrightarrow{\lambda} (\boldsymbol{\nu}\tilde{r})\, C_1' & \mathsf{snd}(\eta) \in \mathsf{fn}(\lambda) & \tilde{r} = \mathsf{bn}(\lambda) \\ \eta \neq \textit{(start)} & \mathsf{rcv}(\eta) \notin \mathsf{fn}(\lambda) & \tilde{r} \notin \mathsf{fn}(\eta) \end{array}}{C = \ \eta; C_1 \xrightarrow{\lambda} (\boldsymbol{\nu}\tilde{r})\, \eta; C_1' \ = C'} \lfloor^{\mathrm{C}}|_{\mathrm{ASYNC}}\rceil$$

We proceed by case analysis on η.

  - **Case** $\eta = \mathsf{p}[\mathtt{A}].e \texttt{ -> } \mathsf{q}[\mathtt{B}].x : k$. Since C is well-typed, we know that:

$$\frac{\Gamma; \Sigma \vdash \mathsf{p}[\mathtt{A}] \texttt{ -> } \mathsf{q}[\mathtt{B}] : k \quad \Gamma \vdash e@\mathsf{p} : S \quad \Gamma, x@\mathsf{q} : S; \Sigma \vdash C_1 \triangleright \Delta'', k : G}{\Gamma; \Sigma \vdash \mathsf{p}[\mathtt{A}].e \texttt{ -> } \mathsf{q}[\mathtt{B}].x : k; C_1 \triangleright \Delta'', k : \mathtt{A} \texttt{ -> } \mathtt{B} : \langle S \rangle; G} \lfloor^{\mathrm{T}}|_{\mathrm{COM}}\rceil$$

Now, we apply the induction hypothesis to $C_1 \xrightarrow{\lambda} (\boldsymbol{\nu}\tilde{r})\, C_1'$. According to the Theorem statement that we are proving, we have four possibilities:

(i) if $\lambda \in \{\tau@\mathsf{p}, \textit{(start)}\}$ then we know that:

$$\Gamma, x@\mathsf{q} : S; \Sigma \vdash (\boldsymbol{\nu}\tilde{r})\, C_1' \triangleright \Delta''', k : G'$$

where $\Delta'', k{:}G \simeq_{\mathcal{G}} \Delta''', k{:}G'$. Assume $\tilde{r} = r_1, \ldots, r_n$; then:

$$\begin{array}{c} \Gamma'', \Gamma, x@\mathsf{q}{:}S; \Sigma'', \Sigma \vdash C_1' \triangleright \Delta_1, k{:}G' \\ \vdots \ \lfloor^{\mathrm{T}}|_{\textsc{Res}}\rceil \ n \text{ times} \\ \Gamma, x@\mathsf{q}{:}S; \Sigma \vdash (\boldsymbol{\nu}\tilde{r})\, C_1' \triangleright \Delta''', k{:}G' \end{array}$$

Let $\Gamma''' = \Gamma'', \Gamma, x@\mathsf{q} : S$; since $\tilde{r} \notin \mathsf{fn}(\eta)$, we can prove the thesis as follows:

$$\dfrac{\begin{array}{c}\Gamma'''; \Sigma'', \Sigma \vdash \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{q}[\mathtt{B}] : k \quad \Gamma''' \vdash e@\mathtt{p}{:}S \quad \Gamma'''; \Sigma'', \Sigma \vdash C_1' \triangleright \Delta_1, k{:}G' \\ \Gamma'', \Gamma; \Sigma'', \Sigma \vdash \eta; C_1' \triangleright \Delta_1, k{:}\mathtt{A} \texttt{ -> } \mathtt{B} : \langle S\rangle; G' \\ \vdots \ \lfloor^{\mathrm{T}}|_{\textsc{Res}}\rceil \ n \text{ times} \\ \Gamma; \Sigma \vdash (\boldsymbol{\nu}\tilde{r})\, \eta; C_1' \triangleright \Delta''', k{:}\mathtt{A} \texttt{ -> } \mathtt{B} : \langle S\rangle; G'\end{array}}{\Gamma'', \Gamma; \Sigma'', \Sigma \vdash \eta; C_1' \triangleright \Delta_1, k{:}\mathtt{A} \texttt{ -> } \mathtt{B} : \langle S\rangle; G'}\ \lfloor^{\mathrm{T}}|_{\textsc{Com}}\rceil$$

(ii) if $\lambda = \mathtt{r}[\mathtt{D}] \texttt{ -> } \mathtt{s}[\mathtt{E}] : k''\langle k'''[\mathtt{F}]\rangle$ and we have applied $\lfloor^{\mathrm{C}}|_{\textsc{Act}}\rceil$, then $\tilde{r}$ is empty and we know:

$$\dfrac{\dfrac{\lambda \in \{\mathit{(sel)}, \mathit{(del)}, \mathit{(start)}\} \quad \tilde{r} = \mathsf{bn}(\lambda)}{C_1 = \lambda; C_2 \xrightarrow{\lambda} C_2 = C_1'}\ \lfloor^{\mathrm{C}}|_{\textsc{Act}}\rceil}{C = \eta; \lambda; C_2 \xrightarrow{\lambda} \eta; C_2 = C'}\ \lfloor^{\mathrm{C}}|_{\textsc{Async}}\rceil$$

$$\left(\begin{array}{lll} \mathsf{snd}(\eta) \in \mathsf{fn}(\lambda) & \tilde{r} = \mathsf{bn}(\lambda) & \\ \eta \neq \mathit{(start)} & \mathsf{rcv}(\eta) \notin \mathsf{fn}(\lambda) & \tilde{r} \notin \mathsf{fn}(\eta) \end{array}\right)$$

From the conditions of rule $\lfloor^{\mathrm{C}}|_{\textsc{Async}}\rceil$, we know that $\mathsf{p} \in \{\mathsf{r}, \mathsf{s}\}$ and that $\mathsf{q} \notin \{\mathsf{r}, \mathsf{s}\}$. Now, we distinguish subcases depending on k, k'', and k'''.

- ∗ **Case** $k'' = k'''$. This case is not allowed, since C_1 is well-typed and rule $\lfloor^{\mathrm{T}}|_{\textsc{Del}}\rceil$ forbids delegating a session over itself.
- ∗ **Case** $k \neq k''$ and $k = k'''$. Let $\Gamma = \Gamma'', \mathsf{p} : k[\mathtt{A}]$. Since C_1 is well-typed, we know that $\mathsf{p} = \mathsf{r}$, $\mathtt{A} = \mathtt{F}$, and:

$$\dfrac{[\ldots] \quad \dfrac{\begin{array}{c}\Gamma''; \Sigma \vdash \mathtt{p}[\mathtt{D}] \texttt{ -> } \mathtt{s}[\mathtt{E}] : k'' \quad \mathtt{p}{:}k[\mathtt{A}] \notin \Sigma \quad G_2 \simeq_{\mathcal{G}} G_2' \\ \Gamma'', \mathtt{s}{:}k[\mathtt{A}]; \Sigma \setminus \mathtt{s}{:}k[\mathtt{A}] \vdash C_2 \triangleright \Delta'', k{:}G_2, k''{:}G'\end{array}}{\Gamma'', \mathtt{p}{:}k[\mathtt{A}]; \Sigma \vdash \lambda; C_2 \triangleright \Delta'', k{:}G_2, k''{:}\mathtt{D} \texttt{ -> } \mathtt{E} : \langle G_2'@\mathtt{A}\rangle; G'}\ \lfloor^{\mathrm{T}}|_{\textsc{Del}}\rceil}{\Gamma'', \mathtt{p}{:}k[\mathtt{A}]; \Sigma \vdash \eta; \lambda; C_2 \triangleright \Delta'', k{:}\mathtt{A} \texttt{ -> } \mathtt{B} : \langle S\rangle; G_2, k''{:}\mathtt{D} \texttt{ -> } \mathtt{E} : \langle G_2'@\mathtt{A}\rangle; G'}\ \lfloor^{\mathrm{T}}|_{\textsc{Com}}\rceil$$

Now the thesis follows by rule $\lfloor^{\mathrm{T}}|_{\textsc{Com}}\rceil$, Lemma A.1.3, and the semantics of global types:

$$\dfrac{[\ldots] \quad \Gamma'', \mathtt{s}{:}k[\mathtt{A}]; \Sigma, \mathtt{p}{:}k[\mathtt{A}] \vdash C_2 \triangleright \Delta'', k{:}G_2, k''{:}G'}{\Gamma'', \mathtt{s}{:}k[\mathtt{A}]; \Sigma, \mathtt{p}{:}k[\mathtt{A}] \vdash \eta; C_2 \triangleright \Delta'', k{:}\mathtt{A} \texttt{ -> } \mathtt{B} : \langle S\rangle; G_2, k''{:}G'}\ \lfloor^{\mathrm{T}}|_{\textsc{Com}}\rceil$$

- ∗ **Case** $k = k''$ and $k \neq k'''$. Let $\Gamma = \Gamma'', \mathsf{p} : k'[\mathtt{C}]$. Since C_1 is well-typed we know that $\mathsf{p} = \mathsf{r}$, $\mathtt{A} = \mathtt{D}$, and:

$$\dfrac{[\ldots] \quad \dfrac{\begin{array}{c}\Gamma''; \Sigma \vdash \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{s}[\mathtt{E}] : k \quad \mathtt{p}{:}k'''[\mathtt{F}] \notin \Sigma \quad G' \simeq_{\mathcal{G}} G''' \\ \Gamma'', \mathtt{s}{:}k'''[\mathtt{F}]; \Sigma \setminus \mathtt{s}{:}k'''[\mathtt{F}] \vdash C_2 \triangleright \Delta'', k{:}G_2, k'''{:}G'\end{array}}{\Gamma'', \mathtt{p}{:}k'''[\mathtt{F}]; \Sigma \vdash \lambda; C_2 \triangleright \Delta'', k{:}\mathtt{A} \texttt{ -> } \mathtt{E} : \langle G'''@\mathtt{F}\rangle; G_2, k'''{:}G'}\ \lfloor^{\mathrm{T}}|_{\textsc{Del}}\rceil}{\Gamma'', \mathtt{p}{:}k'''[\mathtt{F}]; \Sigma \vdash \eta; \lambda; C_2 \triangleright \Delta'', k{:}\mathtt{A} \texttt{ -> } \mathtt{B} : \langle S\rangle; \mathtt{A} \texttt{ -> } \mathtt{E} : \langle G'''@\mathtt{F}\rangle; G_2, k'''{:}G'}\ \lfloor^{\mathrm{T}}|_{\textsc{Com}}\rceil$$

Now the thesis follows by rule $\lfloor^{\mathrm{T}}|_{\textsc{Com}}\rceil$, Lemma A.1.3:

$$\frac{[\ldots] \quad \Gamma'', \mathsf{s}:k'''[\mathtt{F}]; \Sigma, \mathsf{p}:k'''[\mathtt{F}] \ \vdash \ C_2 \ \triangleright \ \Delta'', k:G_2, k''':G'}{\Gamma'', \mathsf{s}:k'''[\mathtt{F}]; \Sigma, \mathsf{p}:k'''[\mathtt{F}] \ \vdash \ \eta; C_2 \ \triangleright \ \Delta'', k:\mathtt{A} \texttt{ -> } \mathtt{B} : \langle S \rangle; G_2, k''':G'} \ \lfloor^{\mathrm{T}}|_{\textsc{Com}}\rceil$$

An interesting difference wrt the previous case is that we need to prove the following reduction for the session typing:

$$\mathtt{A} \texttt{ -> } \mathtt{B} : \langle S \rangle; \mathtt{A} \texttt{ -> } \mathtt{E} : \langle G'''@\mathtt{F} \rangle; G_2 \xrightarrow{\mathtt{A} \texttt{ -> } \mathtt{E}:\langle G'''@\mathtt{F}\rangle} \mathtt{A} \texttt{ -> } \mathtt{B} : \langle S \rangle; G_2$$

which can be proven using rule $\lfloor^{\mathrm{G}}|_{\textsc{ACom}}\rceil$.

* **Case** $k \neq k''$ and $k \neq k'''$ and $k'' \neq k'''$. This case follows the previous proofs, although it is easier since there is no overlapping of sessions.

(iii) if $\lambda = \mathsf{r}[\mathtt{D}] \texttt{ -> } \mathsf{s}[\mathtt{E}] : k''\langle k'''[\mathtt{F}] \rangle$ and we have applied $\lfloor^{\mathrm{C}}|_{\textsc{Async}}\rceil$, then $\tilde{r}$ is empty and we know:

$$\frac{\dfrac{C_2 \xrightarrow{\lambda} C_2'}{C_1 = \ \eta'; C_2 \xrightarrow{\lambda} \eta'; C_2 \ = C_1'} \ \lfloor^{\mathrm{C}}|_{\textsc{Async}}\rceil}{C = \ \eta; \eta'; C_2 \xrightarrow{\lambda} \eta; \eta'; C_2' \ = C'} \ \lfloor^{\mathrm{C}}|_{\textsc{Async}}\rceil$$

$$\begin{pmatrix} \mathsf{snd}(\eta) \in \mathsf{fn}(\lambda) & \tilde{r} = \mathsf{bn}(\lambda) & \\ \eta \neq \textit{(start)} & \mathsf{rcv}(\eta) \notin \mathsf{fn}(\lambda) & \tilde{r} \notin \mathsf{fn}(\eta) \\ \mathsf{snd}(\eta') \in \mathsf{fn}(\lambda) & \mathsf{rcv}(\eta') \notin \mathsf{fn}(\lambda) & \end{pmatrix}$$

The induction hypothesis is:

$$\Gamma[\mathsf{s} \mapsto k'''[\mathtt{F}]], x@\mathsf{q}:S; \Sigma, \mathsf{r}:k'''[\mathtt{F}] \ \vdash \ C_1' \ \triangleright \ \Delta_1'$$

where

$$\Delta_1 \ = \Delta'', k:G \xrightarrow{k''':\mathtt{D} \texttt{ -> } \mathtt{E}:\langle G'''@\mathtt{F}\rangle} = \Delta_1'$$

Now, we distinguish subcases depending on k, k'', and k'''.

* **Case** $k'' = k'''$. This case is not allowed, since C_1 is well-typed and rule $\lfloor^{\mathrm{T}}|_{\textsc{Del}}\rceil$ forbids delegating a session over itself.
* **Case** $k \neq k''$ and $k = k'''$. Since C_1 is well-typed we know that $\mathsf{p} = \mathsf{r}$ and $\mathtt{A} = \mathtt{F}$. Assume $\Gamma' = \Gamma[\mathsf{s} \mapsto k'''[\mathtt{F}]]$ and $\Sigma' = \Sigma, \mathsf{r}:k'''[\mathtt{F}]$. By rule $\lfloor^{\mathrm{T}}|_{\textsc{Com}}\rceil$ we obtain:

$$\frac{\Gamma'; \Sigma' \vdash \mathsf{p}[\mathtt{A}] \texttt{ -> } \mathsf{q}[\mathtt{B}] : k \quad \Gamma' \vdash e@\mathsf{p}:S \quad \Gamma', x@\mathsf{q}:S; \Sigma' \ \vdash \ C_1' \ \triangleright \ \Delta'', k:G'}{\Gamma'; \Sigma' \ \vdash \ \mathsf{p}[\mathtt{A}].e \texttt{ -> } \mathsf{q}[\mathtt{B}].x : k; C_1' \ \triangleright \ \Delta'', k:\mathtt{A} \texttt{ -> } \mathtt{B} : \langle S \rangle; G} \ \lfloor^{\mathrm{T}}|_{\textsc{Com}}\rceil$$

The thesis now follows by induction hypothesis.

* **Case** $k = k''$ and $k \neq k'''$. Since C_1 is well-typed we know that $\mathsf{p} = \mathsf{r}$ and $\mathtt{A} = \mathtt{D}$. The reasoning is now similar to the case above. The only interesting difference is to show that:

$$\mathtt{A} \texttt{ -> } \mathtt{B} : \langle S \rangle; \mathtt{A} \texttt{ -> } \mathtt{E} : \langle G'''@\mathtt{F} \rangle; G \xrightarrow{\mathtt{A} \texttt{ -> } \mathtt{E}:\langle G'''@\mathtt{F}\rangle} \mathtt{A} \texttt{ -> } \mathtt{B} : \langle S \rangle; G$$

which follows from rule $\lfloor^{\mathrm{G}}|_{\textsc{ACom}}\rceil$.

∗ **Case** $k \neq k''$ and $k \neq k'''$ and $k'' \neq k'''$. This case follows the previous proofs, although it is easier since there is no overlapping of sessions.

(iv) This case follows directly from the induction hypothesis.

– **Case** $\eta \in \{(sel), (del)\}$. These cases are similar to the one for $\eta = \mathtt{p}[\mathtt{A}].e \texttt{ -> } \mathtt{q}[\mathtt{B}].x : k$.

– **Case** $\eta = \tilde{p}\ \mathbf{start}\ \tilde{q} : a(k)$. This case is not allowed by rule $\lfloor^{\mathrm{C}}|_{\textsc{Async}}\rceil$.

- **Case** $\lfloor^{\mathrm{C}}|_{\textsc{Ctx}}\rceil$. Follows easily by induction hypothesis.

- **Case** $\lfloor^{\mathrm{C}}|_{\textsc{Cond}}\rceil$. In this case we have that:

$$\frac{i = 1 \text{ if } e \downarrow \text{true}\,,\ i = 2 \text{ otherwise}}{\mathsf{if}\ e@\mathsf{p}\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2 \xrightarrow{\tau@\mathsf{p}} C_i}\ \lfloor^{\mathrm{C}}|_{\textsc{Cond}}\rceil$$

  Since C is well-typed, we know that:

$$\frac{\Gamma \vdash e@\mathsf{p} : \mathbf{bool} \quad \Gamma;\Sigma \vdash C_1 \triangleright \Delta \quad \Gamma;\Sigma \vdash C_2 \triangleright \Delta}{\Gamma;\Sigma \vdash \mathsf{if}\ e@\mathsf{p}\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2 \triangleright \Delta}\ \lfloor^{\mathrm{T}}|_{\textsc{Cond}}\rceil$$

  The thesis follows immediately from the premises of the typing above for both cases, $C' = C_1$ and $C' = C_2$.

- **Case** $\lfloor^{\mathrm{C}}|_{\textsc{Res}}\rceil$. This case follows easily by induction hypothesis.

- **Case** $\lfloor^{\mathrm{C}}|_{\textsc{Eq}}\rceil$. In this case we have that:

$$\frac{\mathcal{R} \in \{\simeq_{\mathcal{C}}, \equiv\} \quad C_1\,\mathcal{R}\,C_1' \quad C_1' \xrightarrow{\lambda} C_2' \quad C_2'\,\mathcal{R}\,C_2}{C_1 \xrightarrow{\lambda} C_2}\ \lfloor^{\mathrm{C}}|_{\textsc{Eq}}\rceil$$

  We have two subcases. For $\mathcal{R} = \simeq_{\mathcal{C}}$, we conclude by Lemma 2.4.1. Otherwise, for $\mathcal{R} = \equiv$, we conclude by Lemma A.1.2.

□

A.2 Proof of Theorem 2.5.15

Again, we start by presenting some auxiliary results before giving the main proof.

We begin by showing properties of EPP wrt substitutions:

Lemma A.2.1 (EPP Substitution)**.**

$$[\![C[v/x@\mathsf{p}]]\!]_{\mathsf{p}} \;=\; [\![C]\!]_{\mathsf{p}}[v/x]$$

Proof. Immediate from the definition of process projection. □

Lemma A.2.2 (EPP Substitution Locality)**.** *Let* q *be a free process name in* C'*, i.e.* $\mathsf{q} \in \mathsf{fn}(C')$*, and*

$$[\![C]\!] = (\boldsymbol{\nu}\tilde{k})\left(\prod_{\mathsf{p}\,\in\,\mathsf{fn}(C')} [\![C']\!]_{\mathsf{p}} \mid \prod_{k\,\in\,\mathsf{fn}(C')} k : \emptyset\right) \mid \prod_{a,\mathtt{A}}\left(\bigsqcup_{\mathsf{p}\in\lfloor C'\rfloor^{a}_{\mathtt{A}}} [\![C']\!]_{\mathsf{p}}\right)$$

Then

$$[\![C[v/x@\mathsf{q}]]\!] = (\boldsymbol{\nu}\tilde{k})\left([\![C[v/x@\mathsf{q}]]\!]_{\mathsf{q}} \mid \prod_{\mathsf{p}\,\in\,\mathsf{fn}(C)\setminus\mathsf{q}} [\![C]\!]_{\mathsf{p}} \mid \prod_{k\,\in\,\mathsf{fn}(C)} k : \emptyset\right) \mid \prod_{a,\mathtt{A}}\left(\bigsqcup_{\mathsf{p}\in\lfloor C\rfloor^{a}_{\mathtt{A}}} [\![C]\!]_{\mathsf{p}}\right)$$

Proof. Immediate from the definition of EPP and Lemma A.2.1. □

Lemma A.2.2 says that a substitution under a free process q in a choreography C affects its projection $[\![C]\!]$ only in the process projection of q.

Lemma A.2.3 (Linearity)**.** *Let* C *be linear, and* $[\![C]\!] \overset{\tilde{\mu}}{\rightsquigarrow}{}^{*}\ P$ *for some* P*. If* P *has a redex at* a*, then* $a[\mathtt{A}]$ *is enabled at most once in* P *for any* $\mathtt{A}$*.*

Proof. By induction on the length of the reduction chain from $[\![C]\!]$ to P. □

Lemma A.2.4 (In-session Linearity)**.** *Let* C *be well-typed and*

$$[\![C]\!] \;\equiv\; (\boldsymbol{\nu}\tilde{k})\,(P \mid Q)$$

where $Q = k[\mathtt{A}]!\mathtt{B}; Q'$ *(*$k[\mathtt{A}]!\mathtt{B}$ *is any output from* $\mathtt{A}$ *to* $\mathtt{B}$*). Then, there are no actions with free subject* $k[\mathtt{A}]$ *in* P*.*

Proof (Sketch). From the well-typedness of C, we can derive that P does not have any output (or input) on the free channel k with role $\mathtt{A}$ (this check is performed by the ownership typing in Γ). □

Now we observe that pruning is transitive, and show that it is preserved by all the process projections that are not involved in the reduction of a choreography.

Proposition A.2.4.1 (Pruning preservation)**.** *Let* $P \prec Q$*. Then,* $P \xrightarrow{\mu} P'$ *implies that there exists* Q' *such that* $Q \xrightarrow{\mu} Q'$ *and* $P' \prec Q'$*.*

Lemma A.2.5 (Passive processes pruning invariance)**.** *Let* C *be restriction-free. Then* $C \overset{\lambda}{\rightsquigarrow} C'$ *implies that for every* $\mathsf{p} \in \mathsf{fn}(C) \setminus \mathsf{fn}(\lambda)$

$$[\![C']\!]_{\mathsf{p}} \quad \prec \quad [\![C]\!]_{\mathsf{p}}$$

Proof. By case analysis on the rules defining the semantics of the Choreography Calculus. The only interesting case is $\lfloor {}^{\mathsf{C}}|_{\textsc{Cond}}\rceil$. In this case, the projections of the processes receiving selections are merged. The thesis follows immediately by definition of pruning. Observe that we can safely ignore potential swaps in C performed in its reduction, thanks to Lemma 2.5.1. □

Lemma A.2.6 (Pruning Lemma)**.** *Let C be well-typed and $[\![C]\!] = ((\boldsymbol{\nu}\tilde{k})\ P) \mid R$ where*

$$R = \prod_{a,\mathtt{A}} \Big(\bigsqcup_{\mathsf{p} \in \lfloor C \rfloor^a_{\mathtt{A}}} [\![C]\!]_{\mathsf{p}} \Big)$$

Let now R' be the process obtained from R by (i) adding some new services on new public channels and (ii) merging the services in R with some other services:

$$R' = \prod_{a,\mathtt{A}} \Big(\big(\bigsqcup_{\mathsf{p} \in \lfloor C \rfloor^a_{\mathtt{A}}} [\![C]\!]_{\mathsf{p}} \big) \sqcup R^a_{\mathtt{A}} \Big) \mid \prod_{i \in I} !a_i[\mathtt{A}_i](k_i); P_i$$

where for all $i \in I$, $a_i \notin \mathsf{fn}(C)$. Then, R' defined implies: $[\![C]\!] \prec ((\boldsymbol{\nu}\tilde{k})\ P) \mid R'$

Proof (Sketch). From the well-typedness of C, we can derive that P does not use any of the new public channels a_i. The same reasoning can be used for proving that all the new branches introduced in R' are never going to be used. □

Lemma A.2.7 (EPP under $\equiv$)**.** *Let $C \equiv C'$. Then, $[\![C]\!] \equiv [\![C']\!]$.*

We can finally prove the EPP Theorem.

Theorem A.2.1 (EPP Theorem)**.** *Let $C \equiv (\boldsymbol{\nu}\tilde{\mathsf{p}}, \tilde{k})\ C_f$ be linear and well-typed, with C_f restriction-free; then,*

1. *(Completeness) $C \overset{\lambda}{\rightsquigarrow} C'$ implies that there exist P and C'' such that (i) $C' \overset{\tilde{\lambda}'}{\rightsquigarrow}{}^* C''$; (ii) $[\![C]\!] \overset{\tilde{\mu}}{\rightsquigarrow}{}^* P$ and $\lambda, \tilde{\lambda}' \vdash \tilde{\mu}$; and (iii) $[\![C'']\!] \prec P$.*
2. *(Soundness) $[\![C]\!] \overset{\tilde{\mu}}{\rightsquigarrow}{}^* P$ implies that there exist P' and C' such that (i) $P \overset{\tilde{\mu}'}{\rightsquigarrow}{}^* P'$; (ii) $C \overset{\tilde{\lambda}}{\rightsquigarrow}{}^* C'$ and $\tilde{\lambda} \vdash \tilde{\mu}, \tilde{\mu}'$; and (iii) $[\![C']\!] \prec P'$.*

For clarity, we split the proof in two parts. We first prove the completeness part and then the soundness part of the Theorem.

Proof (Completeness). The proof is by induction on the derivation of $C \overset{\lambda}{\rightsquigarrow} C'$.

- **Case** $\lfloor {}^{\mathrm{C}} |_{\mathrm{COM}} \rceil$. We know that

$$C = \mathsf{p}[\mathtt{A}].e \texttt{ -> } \mathsf{q}[\mathtt{B}].x : k; C_1 \quad \overset{\mathsf{p}[\mathtt{A}].v \texttt{ -> } \mathsf{q}[\mathtt{B}].x:k}{\rightsquigarrow} \quad C_1[v/x@\mathsf{q}] = C' \quad (e \downarrow v) \quad \text{(A.1)}$$

  We choose $C' = C''$. From the definition of EPP we have:

$$[\![C]\!] \equiv (\boldsymbol{\nu}k, \tilde{k}) \Big(k[\mathtt{A}]!\mathtt{B}\langle e\rangle; [\![C''_f]\!]_{\mathsf{p}} \quad \mid \quad k[\mathtt{B}]?\mathtt{A}(x); [\![C''_f]\!]_{\mathsf{q}} \quad \mid \quad k : \emptyset$$

$$\mid \prod_{\mathsf{p} \in \mathsf{fn}(C_f) \setminus \mathsf{p},\mathsf{q}} [\![C_f]\!]_{\mathsf{p}} \mid \prod_{k' \in \tilde{k}} k' : \emptyset \Big) \mid \prod_{a,\mathtt{A}} \big(\bigsqcup_{\mathsf{p} \in \lfloor C_f \rfloor^a_{\mathtt{A}}} [\![C_f]\!]_{\mathsf{p}} \big)$$

where $(\boldsymbol{\nu}\tilde{r})\ C''_f \equiv C_1$, with C''_f restriction-free, and $\tilde{k}$ are the free session channel names in C_f excluding k. Using $\lfloor^{\mathsf{P}}|_{\text{COM-S}}\rceil$ we can derive:

$$[\![C]\!] \overset{!\mathtt{A}\ \text{->}\ \mathtt{B}:k\langle v\rangle}{\rightsquigarrow} (\boldsymbol{\nu}k,\tilde{k})\left([\![C''_f]\!]_{\mathsf{p}} \mid k[\mathtt{B}]?\mathtt{A}(x);[\![C''_f]\!]_{\mathsf{q}} \mid k:(\mathtt{A},\mathtt{B},v)\right.$$
$$\left. \mid \prod_{\mathsf{p}\in \mathsf{fn}(C_f)\setminus \mathsf{p},\mathsf{q}} [\![C_f]\!]_{\mathsf{p}} \mid \prod_{k'\in\tilde{k}} k':\emptyset\right) \mid \prod_{a,\mathtt{A}}\Big(\bigsqcup_{\mathsf{p}\in\lfloor C_f\rfloor^a_{\mathtt{A}}} [\![C_f]\!]_{\mathsf{p}}\Big) \;=\; Q$$

By $\lfloor^{\mathsf{P}}|_{\text{COM-R}}\rceil$ we obtain:

$$Q \overset{?\mathtt{A}\ \text{->}\ \mathtt{B}:k\langle v\rangle}{\rightsquigarrow} (\boldsymbol{\nu}k,\tilde{k})\left([\![C''_f]\!]_{\mathsf{p}} \mid [\![C''_f]\!]_{\mathsf{q}}[v/x] \mid k:\emptyset\right.$$
$$\left. \mid \prod_{\mathsf{p}\in \mathsf{fn}(C_f)\setminus \mathsf{p},\mathsf{q}} [\![C_f]\!]_{\mathsf{p}} \mid \prod_{k'\in\tilde{k}} k':\emptyset\right) \mid \prod_{a,\mathtt{A}}\Big(\bigsqcup_{\mathsf{p}\in\lfloor C_f\rfloor^a_{\mathtt{A}}} [\![C_f]\!]_{\mathsf{p}}\Big) \;=\; P$$

which proves (ii).

Let us see now the projection of C''. From (A.1) and Lemma A.2.2 we have:

$$[\![C'']\!] \;\equiv (\boldsymbol{\nu}\tilde{k'})\left(([\![C''_f]\!]_{\mathsf{p}}) \mid ([\![C''_f[v/x@\mathsf{q}]]\!]_{\mathsf{q}}) \mid \prod_{k'\in\tilde{k'}} k':\emptyset\right.$$
$$\left. \mid \prod_{\mathsf{p}\in \mathsf{fn}(C''_f)\setminus \mathsf{p},\mathsf{q}} [\![C''_f]\!]_{\mathsf{p}}\right) \mid \prod_{a,\mathtt{A}}\Big(\bigsqcup_{\mathsf{p}\in\lfloor C''_f\rfloor^a_{\mathtt{A}}} [\![C''_f]\!]_{\mathsf{p}}\Big)$$

where $\tilde{k'}$ are the free session channels in C''_f. From Lemma A.2.1 we obtain:

$$[\![C'']\!] \;\equiv (\boldsymbol{\nu}\tilde{k'})\left([\![C''_f]\!]_{\mathsf{p}} \mid [\![C''_f]\!]_{\mathsf{q}}[v/x] \mid \prod_{k'\in\tilde{k'}} k':\emptyset\right.$$
$$\left. \mid \prod_{\mathsf{p}\in \mathsf{fn}(C''_f)\setminus \mathsf{p},\mathsf{q}} [\![C''_f]\!]_{\mathsf{p}}\right) \mid \prod_{a,\mathtt{A}}\Big(\bigsqcup_{\mathsf{p}\in\lfloor C''_f\rfloor^a_{\mathtt{A}}} [\![C''_f]\!]_{\mathsf{p}}\Big)$$

We observe now that $\mathsf{fn}(C'') \subseteq \mathsf{fn}(C)$, because the reduction from C to C'' has not added any free name. Hence, by Lemma A.2.5 we get:

$$[\![C'']\!] \;\prec (\boldsymbol{\nu}\tilde{k'})\left([\![C''_f]\!]_{\mathsf{p}} \mid [\![C''_f]\!]_{\mathsf{q}}[v/x] \mid \prod_{k'\in\tilde{k'}} k':\emptyset\right.$$
$$\left. \mid \prod_{\mathsf{p}\in \mathsf{fn}(C_f)\setminus \mathsf{p},\mathsf{q}} [\![C_f]\!]_{\mathsf{p}}\right) \mid \prod_{a,\mathtt{A}}\Big(\bigsqcup_{\mathsf{p}\in\lfloor C''_f\rfloor^a_{\mathtt{A}}} [\![C''_f]\!]_{\mathsf{p}}\Big)$$

Using the same observation again ($\mathsf{fn}(C'') \subseteq \mathsf{fn}(C)$), it follows from the definition of EPP (and merging) that:

$$\begin{aligned}[\![C'']\!] \;\; &\prec (\boldsymbol{\nu}\tilde{k'}) \Bigg([\![C''_f]\!]_{\mathsf{p}} \mid [\![C''_f]\!]_{\mathsf{q}}[v/x] \mid \prod_{k' \in \tilde{k'}} k' : \emptyset \\ &\qquad \mid \prod_{\mathsf{p} \in \mathsf{fn}(C_f) \setminus \mathsf{p},\mathsf{q}} [\![C_f]\!]_{\mathsf{p}} \Bigg) \mid \prod_{a,\mathtt{A}} \Big(\bigsqcup_{\mathsf{p} \in \lfloor C_f \rfloor^a_{\mathtt{A}}} [\![C_f]\!]_{\mathsf{p}} \Big) \\ &\prec P\end{aligned}$$

which proves (iii).

- **Case** $\lfloor {}^{\mathrm{C}} |_{\textsc{Act}} \rceil$. In this case we have:

$$\frac{\eta \in \{(sel), (del), (start)\} \quad \tilde{r} = \mathsf{bn}(\eta)}{C = \; \eta; C_1 \xrightarrow{\eta} (\boldsymbol{\nu}\tilde{r})\, C_1 \; = C'} \; \lfloor {}^{\mathrm{C}} |_{\textsc{Act}} \rceil$$

We choose $C' = C''$ and proceed by case analysis on η.

  - **Case** $\eta = \mathsf{p}[\mathtt{A}] \texttt{ -> } \mathsf{q}[\mathtt{B}] : k[l]$. We know that

$$C = \mathsf{p}[\mathtt{A}] \texttt{ -> } \mathsf{q}[\mathtt{B}] : k[l]; C' \xrightarrow{\mathsf{p}[\mathtt{A}] \texttt{ -> } \mathsf{q}[\mathtt{B}]:k[l]} C' \tag{A.2}$$

From the definition of EPP we have:

$$\begin{aligned}[\![C]\!] \equiv (\boldsymbol{\nu}k, \tilde{k}) \Bigg(&k[\mathtt{A}]!\mathtt{B} \oplus l; [\![C''_f]\!]_{\mathsf{p}} \quad \mid \quad k[\mathtt{B}]?\mathtt{A} \,\&\, \{l : [\![C''_f]\!]_{\mathsf{q}}\} \quad \mid \quad k : \emptyset \\ &\mid \prod_{\mathsf{p} \in \mathsf{fn}(C_f) \setminus \mathsf{p},\mathsf{q}} [\![C_f]\!]_{\mathsf{p}} \mid \prod_{k' \in \tilde{k}} k' : \emptyset \Bigg) \mid \prod_{a,\mathtt{A}} \Big(\bigsqcup_{\mathsf{p} \in \lfloor C_f \rfloor^a_{\mathtt{A}}} [\![C_f]\!]_{\mathsf{p}} \Big)\end{aligned}$$

where $(\boldsymbol{\nu}\tilde{r})\; C''_f \equiv C'$, with C''_f restriction-free, and $\tilde{k}$ are the free session channel names in C_f excluding k. Using $\lfloor {}^{\mathrm{P}} |_{\textsc{sel-s}} \rceil$ we can derive:

$$\begin{aligned}[\![C]\!] \;\; \overset{!\mathtt{A} \texttt{ -> } \mathtt{B}:k[l]}{\rightsquigarrow} \;\; (\boldsymbol{\nu}k, \tilde{k}) \Bigg(&[\![C''_f]\!]_{\mathsf{p}} \mid k[\mathtt{B}]?\mathtt{A} \,\&\, \{l : [\![C''_f]\!]_{\mathsf{q}}\} \mid k : (\mathtt{A}, \mathtt{B}, l) \\ &\mid \prod_{\mathsf{p} \in \mathsf{fn}(C_f) \setminus \mathsf{p},\mathsf{q}} [\![C_f]\!]_{\mathsf{p}} \mid \prod_{k' \in \tilde{k}} k' : \emptyset \Bigg) \mid \prod_{a,\mathtt{A}} \Big(\bigsqcup_{\mathsf{p} \in \lfloor C_f \rfloor^a_{\mathtt{A}}} [\![C_f]\!]_{\mathsf{p}} \Big) \; = \; Q\end{aligned}$$

By $\lfloor {}^{\mathrm{P}} |_{\textsc{Branch}} \rceil$ we obtain:

$$\begin{aligned}Q \;\; \overset{?\mathtt{A} \texttt{ -> } \mathtt{B}:k[l]}{\rightsquigarrow} \;\; (\boldsymbol{\nu}k, \tilde{k}) \Bigg(&[\![C''_f]\!]_{\mathsf{p}} \mid [\![C''_f]\!]_{\mathsf{q}} \mid k : \emptyset \\ &\mid \prod_{\mathsf{p} \in \mathsf{fn}(C_f) \setminus \mathsf{p},\mathsf{q}} [\![C_f]\!]_{\mathsf{p}} \mid \prod_{k' \in \tilde{k}} k' : \emptyset \Bigg) \mid \prod_{a,\mathtt{A}} \Big(\bigsqcup_{\mathsf{p} \in \lfloor C_f \rfloor^a_{\mathtt{A}}} [\![C_f]\!]_{\mathsf{p}} \Big) \; = \; P\end{aligned}$$

which proves (ii).

Let us see now the projection of C''. From (A.2) we have:

$$\begin{aligned}\llbracket C'' \rrbracket \;\equiv\; (\boldsymbol{\nu}\tilde{k'}) \Bigg((\llbracket C''_f \rrbracket_{\mathsf{p}}) \mid (\llbracket C''_f \rrbracket_{\mathsf{q}}) \mid \prod_{k' \in \tilde{k'}} k' : \emptyset \\ \mid \prod_{\mathsf{p} \in \mathsf{fn}(C''_f) \setminus \mathsf{p},\mathsf{q}} \llbracket C''_f \rrbracket_{\mathsf{p}} \Bigg) \mid \prod_{a,\mathtt{A}} \Big(\bigsqcup_{\mathsf{p} \in \lfloor C''_f \rfloor^a_{\mathtt{A}}} \llbracket C''_f \rrbracket_{\mathsf{p}} \Big)\end{aligned}$$

where $\tilde{k'}$ are the free session channels in C''_f. We observe now that $\mathsf{fn}(C'') \subseteq \mathsf{fn}(C)$, because the reduction from C to C'' has not added any free name. Hence, by Lemma A.2.5 we get:

$$\begin{aligned}\llbracket C'' \rrbracket \;\prec\; (\boldsymbol{\nu}\tilde{k'}) \Bigg(\llbracket C''_f \rrbracket_{\mathsf{p}} \mid \llbracket C''_f \rrbracket_{\mathsf{q}} \mid \prod_{k' \in \tilde{k'}} k' : \emptyset \\ \mid \prod_{\mathsf{p} \in \mathsf{fn}(C_f) \setminus \mathsf{p},\mathsf{q}} \llbracket C_f \rrbracket_{\mathsf{p}} \Bigg) \mid \prod_{a,\mathtt{A}} \Big(\bigsqcup_{\mathsf{p} \in \lfloor C''_f \rfloor^a_{\mathtt{A}}} \llbracket C''_f \rrbracket_{\mathsf{p}} \Big)\end{aligned}$$

Using the same observation again ($\mathsf{fn}(C'') \subseteq \mathsf{fn}(C)$), it follows from the definition of EPP (and merging) that:

$$\begin{aligned}\llbracket C'' \rrbracket \;\prec\; (\boldsymbol{\nu}\tilde{k'}) \Bigg(\llbracket C''_f \rrbracket_{\mathsf{p}} \mid \llbracket C''_f \rrbracket_{\mathsf{q}} \mid \prod_{k' \in \tilde{k'}} k' : \emptyset \\ \mid \prod_{\mathsf{p} \in \mathsf{fn}(C_f) \setminus \mathsf{p},\mathsf{q}} \llbracket C_f \rrbracket_{\mathsf{p}} \Bigg) \mid \prod_{a,\mathtt{A}} \Big(\bigsqcup_{\mathsf{p} \in \lfloor C_f \rfloor^a_{\mathtt{A}}} \llbracket C_f \rrbracket_{\mathsf{p}} \Big) \\ \prec P\end{aligned}$$

which proves (iii).

– **Case** $\eta = \mathsf{p}[\mathtt{A}] \to \mathsf{q}[\mathtt{B}] : k\langle k'[\mathtt{C}] \rangle$. We know that

$$C = \mathsf{p}[\mathtt{A}] \to \mathsf{q}[\mathtt{B}] : k\langle k'[\mathtt{C}] \rangle; C' \xrightarrow{\mathsf{p}[\mathtt{A}] \to \mathsf{q}[\mathtt{B}] : k\langle k'[\mathtt{C}] \rangle} C' \tag{A.3}$$

From the definition of EPP we have:

$$\begin{aligned}\llbracket C \rrbracket \equiv (\boldsymbol{\nu} k, \tilde{k}) \Bigg(k[\mathtt{A}]!\mathtt{B}\langle k'[\mathtt{C}] \rangle; \llbracket C''_f \rrbracket_{\mathsf{p}} \quad \mid \quad k[\mathtt{B}]?\mathtt{A}(k'[\mathtt{C}]); \llbracket C''_f \rrbracket_{\mathsf{q}} \quad \mid \quad k : \emptyset \\ \mid \prod_{\mathsf{p} \in \mathsf{fn}(C_f) \setminus \mathsf{p},\mathsf{q}} \llbracket C_f \rrbracket_{\mathsf{p}} \mid \prod_{k' \in \tilde{k}} k' : \emptyset \Bigg) \mid \prod_{a,\mathtt{A}} \Big(\bigsqcup_{\mathsf{p} \in \lfloor C_f \rfloor^a_{\mathtt{A}}} \llbracket C_f \rrbracket_{\mathsf{p}} \Big)\end{aligned}$$

where $(\boldsymbol{\nu}\tilde{r})\ C''_f \equiv C'$, with C''_f restriction-free, and $\tilde{k}$ are the free session

channel names in C_f excluding k. Using $\lfloor {}^{\mathrm{P}} | {}_{\text{DEL-S}} \rceil$ we can derive:

$$[\![C]\!] \overset{\texttt{!A -> B:}k\langle k'[\texttt{C}]\rangle}{\rightsquigarrow} (\boldsymbol{\nu} k, \tilde{k}) \left([\![C''_f]\!]_{\mathsf{p}} \mid k[\texttt{B}]?\texttt{A}(k'[\texttt{C}]); [\![C''_f]\!]_{\mathsf{q}} \mid k : (\texttt{A}, \texttt{B}, k'[\texttt{C}]) \right.$$
$$\left. \mid \prod_{\mathsf{p} \in \mathsf{fn}(C_f) \setminus \mathsf{p},\mathsf{q}} [\![C_f]\!]_{\mathsf{p}} \mid \prod_{k' \in \tilde{k}} k' : \emptyset \right) \mid \prod_{a,\texttt{A}} \Big(\bigsqcup_{\mathsf{p} \in \lfloor C_f \rfloor^{a}_{\texttt{A}}} [\![C_f]\!]_{\mathsf{p}} \Big) \;=\; Q$$

By $\lfloor {}^{\mathrm{P}} | {}_{\text{DEL-R}} \rceil$ we obtain:

$$Q \overset{\texttt{?A -> B:}k\langle k'[\texttt{C}]\rangle}{\rightsquigarrow} (\boldsymbol{\nu} k, \tilde{k}) \left([\![C''_f]\!]_{\mathsf{p}} \mid [\![C''_f]\!]_{\mathsf{q}} \mid k : \emptyset \right.$$
$$\left. \mid \prod_{\mathsf{p} \in \mathsf{fn}(C_f) \setminus \mathsf{p},\mathsf{q}} [\![C_f]\!]_{\mathsf{p}} \mid \prod_{k' \in \tilde{k}} k' : \emptyset \right) \mid \prod_{a,\texttt{A}} \Big(\bigsqcup_{\mathsf{p} \in \lfloor C_f \rfloor^{a}_{\texttt{A}}} [\![C_f]\!]_{\mathsf{p}} \Big) \;=\; P$$

which proves (ii). (iii) follows by same reasoning as the previous case.

– **Case** $\eta = \widetilde{\mathsf{p}[\texttt{A}]}\ \mathbf{start}\ \widetilde{\mathsf{q}[\texttt{B}]} : a(k)$. Let $\tilde{\mathsf{p}} = \mathsf{p}_1, \ldots, \mathsf{p}_n$ and $\tilde{\mathsf{q}} = \mathsf{q}_1, \ldots, \mathsf{q}_m$. We know that:

$$C = \widetilde{\mathsf{p}[\texttt{A}]}\ \mathbf{start}\ \widetilde{\mathsf{q}[\texttt{B}]} : a(k); C_1 \overset{\widetilde{\mathsf{p}[\texttt{A}]}\ \mathbf{start}\ \widetilde{\mathsf{q}[\texttt{B}]}:a(k)}{\rightsquigarrow} (\boldsymbol{\nu} k, \mathsf{q}_1, \ldots, \mathsf{q}_m)\, C_1 = C'$$

From the definition of EPP we have that:

$$[\![C]\!] \;\equiv (\boldsymbol{\nu} \tilde{k'}) \left(\overline{a}[\tilde{\texttt{A}}, \tilde{\texttt{B}}](k); [\![C''_f]\!]_{\mathsf{p}_1} \mid \prod_{i \in [2,n]} a[\texttt{A}_i](k); [\![C''_f]\!]_{\mathsf{p}_i} \mid \prod_{k' \in \tilde{k'}} k' : \emptyset \right.$$
$$\left. \mid \prod_{\mathsf{p} \in \mathsf{fn}(C_f) \setminus \tilde{\mathsf{p}}} [\![C_f]\!]_{\mathsf{p}} \right) \mid \prod_{i \in [1,m]} !a[\texttt{B}_i](k); \Big(\bigsqcup_{\mathsf{q} \in \lfloor C_f \rfloor^{a}_{\texttt{B}_i}} [\![C''_f]\!]_{\mathsf{q}} \Big)$$
$$\mid \prod_{a' \neq a,\texttt{A}} \Big(\bigsqcup_{\mathsf{p} \in \lfloor C_f \rfloor^{a'}_{\texttt{A}}} [\![C_f]\!]_{\mathsf{p}} \Big) = Q$$

where $\tilde{k}$ are the free session channel names in C_f. We base our partitioning of the services in the EPP on the fact that, thanks to the well-typedness of C, we are sure that the roles $\tilde{\texttt{B}}$ are the only ones that are played by the service processes in C on shared channel a.

Let $\mu = \tilde{\texttt{A}}\ \mathbf{start}\ \tilde{\texttt{B}} : a(k)$. Applying $\lfloor {}^{\mathrm{P}} | {}_{\text{START}} \rceil$ we can derive

$$Q \overset{\mu}{\rightsquigarrow} (\boldsymbol{\nu} \tilde{k'}, k) \left(\prod_{i \in [1,n]} [\![C''_f]\!]_{\mathsf{p}_i} \mid \prod_{k' \in \tilde{k'}, k} k' : \emptyset \mid \prod_{\mathsf{p} \in \mathsf{fn}(C_f) \setminus [\tau]} [\![C_f]\!]_{\mathsf{p}} \right)$$
$$\mid \prod_{i \in [1,m]} !a[\texttt{B}_i](k); \Big(\bigsqcup_{\mathsf{q} \in \lfloor C_f \rfloor^{a}_{\texttt{B}_i}} [\![C''_f]\!]_{\mathsf{q}} \Big) \mid \prod_{a' \neq a,\texttt{A}} \Big(\bigsqcup_{\mathsf{p} \in \lfloor C_f \rfloor^{a'}_{\texttt{A}}} [\![C_f]\!]_{\mathsf{p}} \Big) \;=\; P$$

which proves (ii). (iii) follows from similar reasoning as in the case for $\lfloor {}^{\mathrm{C}} | {}_{\text{COM}} \rceil$.

- **Case** $\lfloor {}^{\mathrm{C}} | {}_{\textsc{Async}} \rceil$. In this case we have that:

$$\frac{\begin{array}{lll} C_1 \overset{\lambda}{\rightsquigarrow} (\boldsymbol{\nu}\tilde{r})\, C_1' & \mathsf{snd}(\eta) \in \mathsf{fn}(\lambda) & \tilde{r} = \mathsf{bn}(\lambda) \\ \eta \neq \textit{(start)} & \mathsf{rcv}(\eta) \notin \mathsf{fn}(\lambda) & \tilde{r} \notin \mathsf{fn}(\eta) \end{array}}{C = \eta; C_1 \overset{\lambda}{\rightsquigarrow} (\boldsymbol{\nu}\tilde{r})\, \eta; C_1' = C'} \lfloor {}^{\mathrm{C}} | {}_{\textsc{Async}} \rceil$$

Since $\mathsf{snd}(\eta) \in \mathsf{fn}(\lambda)$, we know that $\lambda \in \{\tau@\mathsf{p}, \textit{(com)}, \textit{(sel)}, \textit{(del)}, \textit{(start)}\}$ We proceed by mutual case analysis on η and λ. We report one of the most interesting cases (due to overlapping of session names and roles): $\eta = \mathsf{p}[\mathtt{A}].e \texttt{ -> } \mathsf{q}[\mathtt{B}].x : k$ and $\lambda = \mathsf{p}[\mathtt{A}].v' \texttt{ -> } \mathsf{r}[\mathtt{C}].y : k$; all the other cases follow by similar reasoning to that below.

By rules $\lfloor {}^{\mathrm{C}} | {}_{\textsc{Com}} \rceil$ and $\lfloor {}^{\mathrm{C}} | {}_{\textsc{Res}} \rceil$ we have that, for $\lambda' = \mathsf{p}[\mathtt{A}].e \texttt{ -> } \mathsf{q}[\mathtt{B}].x : k$:

$$C' \overset{\lambda'}{\rightsquigarrow} (\boldsymbol{\nu}\tilde{r})\, C_1'[v/x@\mathsf{q}] = C'' \quad (e \downarrow v) \tag{A.4}$$

From the definition of EPP we have:

$$[\![C]\!] \equiv (\boldsymbol{\nu}k, \tilde{k}) \Bigg(k[\mathtt{A}]!\mathtt{B}\langle e\rangle; [\![C_f'']\!]_{\mathsf{p}} \mid k[\mathtt{B}]?\mathtt{A}(x); [\![C_f'']\!]_{\mathsf{q}} \mid [\![C_f'']\!]_{\mathsf{r}} \mid k : \emptyset$$
$$\mid \prod_{\mathsf{p} \in \mathsf{fn}(C_f) \setminus \mathsf{p},\mathsf{q},\mathsf{r}} [\![C_f]\!]_{\mathsf{p}} \mid \prod_{k' \in \tilde{k}} k' : \emptyset \Bigg) \mid \prod_{a,\mathtt{A}} \Big(\bigsqcup_{\mathsf{p} \in \lfloor C_f \rfloor^a_{\mathtt{A}}} [\![C_f]\!]_{\mathsf{p}} \Big)$$

where $(\boldsymbol{\nu}\tilde{r'})\, C_f'' \equiv C_1$, with C_f'' restriction-free, and $\tilde{k}$ are the free session channel names in C_f excluding k. Using $\lfloor {}^{\mathrm{P}} | {}_{\textsc{Com-S}} \rceil$ we can derive:

$$[\![C]\!] \overset{\mathtt{!A\ ->\ B}:k\langle v\rangle}{\rightsquigarrow} (\boldsymbol{\nu}k, \tilde{k}) \Bigg([\![C_f'']\!]_{\mathsf{p}} \mid k[\mathtt{B}]?\mathtt{A}(x); [\![C_f'']\!]_{\mathsf{q}} \mid [\![C_f'']\!]_{\mathsf{r}} \mid k : (\mathtt{A}, \mathtt{B}, v)$$
$$\mid \prod_{\mathsf{p} \in \mathsf{fn}(C_f) \setminus \mathsf{p},\mathsf{q},\mathsf{r}} [\![C_f]\!]_{\mathsf{p}} \mid \prod_{k' \in \tilde{k}} k' : \emptyset \Bigg) \mid \prod_{a,\mathtt{A}} \Big(\bigsqcup_{\mathsf{p} \in \lfloor C_f \rfloor^a_{\mathtt{A}}} [\![C_f]\!]_{\mathsf{p}} \Big) = Q$$

Observe that the only difference between Q and $[\![C_1]\!]$ is the projection of process q, which in Q is $k[\mathtt{B}]?\mathtt{A}(x); [\![C_f'']\!]_{\mathsf{q}}$ and in $[\![C_1]\!]$ is simply $[\![C_f'']\!]_{\mathsf{q}}$. Since $\mathsf{rcv}(\eta) = \mathsf{q} \notin \mathsf{fn}(\lambda)$, we know that q is not involved in the reduction $C_1 \overset{\lambda}{\rightsquigarrow} (\boldsymbol{\nu}\tilde{r})\, C_1'$. Hence, we can apply the induction hypothesis, add the prefix $k[\mathtt{B}]?\mathtt{A}(x)$ to the projection of q in $[\![C_1']\!]$ and by rule $\lfloor {}^{\mathrm{P}} | {}_{\textsc{Com-R}} \rceil$ obtain:

$$Q \overset{\tilde{\mu'}}{\rightsquigarrow}{}^{*} \overset{\mathtt{?A\ ->\ B}:k\langle v\rangle}{\rightsquigarrow} (\boldsymbol{\nu}k, \tilde{k}) \Bigg([\![C_f'']\!]_{\mathsf{p}} \mid [\![C_f'']\!]_{\mathsf{q}}[v/x] \mid k : \emptyset$$
$$\mid \prod_{\mathsf{p} \in \mathsf{fn}(C_f) \setminus \mathsf{p},\mathsf{q}} [\![C_f]\!]_{\mathsf{p}} \mid \prod_{k' \in \tilde{k}} k' : \emptyset \Bigg) \mid \prod_{a,\mathtt{A}} \Big(\bigsqcup_{\mathsf{p} \in \lfloor C_f \rfloor^a_{\mathtt{A}}} [\![C_f]\!]_{\mathsf{p}} \Big) = P$$

The thesis follows now by the definition of pruning and Lemmas A.2.2 and A.2.5. Observe that the message $(\mathtt{A}, \mathtt{B}, v)$ in the queue for k does not interfere with the reductions labelled $\tilde{\mu'}$, since $\mathtt{B}$ cannot be delegated by q in $\tilde{\mu'}$ and therefore we can

always swap such messages with others inserted in the queue for k using the structural congruence rule for session queues reported in Figure 2.16.

- **Case** $\lfloor^{\mathrm{C}}|_{\textsc{Ctx}}\rceil$. Follows immediately from the definition of EPP and the induction hypothesis.

- **Case** $\lfloor^{\mathrm{C}}|_{\textsc{Cond}}\rceil$. In this case we have:

$$\frac{i = 1 \text{ if } e \downarrow \text{true}\,,\ i = 2 \text{ otherwise}}{\mathsf{if}\ e@\mathsf{p}\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2 \xrightarrow{\tau@\mathsf{p}} C_i = C'}\ \lfloor^{\mathrm{C}}|_{\textsc{Cond}}\rceil$$

  We report the case for $i = 1$ (the other case, for $i = 2$, is similar). We choose $C' = C''$. From the definition of EPP we have:

$$[\![C]\!] \equiv (\boldsymbol{\nu}\tilde{k})\left(\mathsf{if}\ e\ \mathsf{then}\ [\![C'_f]\!]_{\mathsf{p}}\ \mathsf{else}\ [\![C''_f]\!]_{\mathsf{p}} \mid \prod_{\mathsf{p}\in\mathsf{fn}(C_f)\setminus\mathsf{p}} [\![C_f]\!]_{\mathsf{p}} \mid \prod_{k'\in\tilde{k}} k' : \emptyset\right)$$
$$\mid \prod_{a,\mathtt{A}}\Big(\bigsqcup_{\mathsf{p}\in\lfloor C_f\rfloor^a_{\mathtt{A}}} [\![C_f]\!]_{\mathsf{p}}\Big) \quad = Q$$

  where $(\boldsymbol{\nu}\tilde{r'})\ C'_f \equiv C_1$ and $(\boldsymbol{\nu}\tilde{r''})\ C''_f \equiv C_2$, with C'_f and C''_f restriction-free, and $\tilde{k}$ are the free session channel names in C_f. By rule $\lfloor^{\mathrm{P}}|_{\textsc{Cond}}\rceil$ we get:

$$Q \overset{\tau}{\rightsquigarrow} (\boldsymbol{\nu}\tilde{k})\left([\![C'_f]\!]_{\mathsf{p}} \mid \prod_{\mathsf{p}\in\mathsf{fn}(C_f)\setminus\mathsf{p}} [\![C_f]\!]_{\mathsf{p}} \mid \prod_{k'\in\tilde{k}} k' : \emptyset\right)$$
$$\mid \prod_{a,\mathtt{A}}\Big(\bigsqcup_{\mathsf{p}\in\lfloor C_f\rfloor^a_{\mathtt{A}}} [\![C_f]\!]_{\mathsf{p}}\Big) \quad = P$$

  The thesis follows now by Lemma A.2.5 and the definition of process projection.

- **Case** $\lfloor^{\mathrm{C}}|_{\textsc{Res}}\rceil$. Follows immediately from the definition of EPP and the induction hypothesis.

- **Case** $\lfloor^{\mathrm{C}}|_{\textsc{Eq}}\rceil$. In this case we have:

$$\frac{\mathcal{R} \in \{\simeq_{\mathcal{C}}, \equiv\} \quad C\,\mathcal{R}\,C'_1 \quad C'_1 \overset{\lambda}{\rightsquigarrow} C'_2 \quad C'_2\,\mathcal{R}\,C'}{C \overset{\lambda}{\rightsquigarrow} C'}\ \lfloor^{\mathrm{C}}|_{\textsc{Eq}}\rceil$$

  For $\mathcal{R} = \simeq_{\mathcal{C}}$, the thesis follows from the induction hypothesis and Lemma 2.5.1. Otherwise, for $\mathcal{R} = \equiv$, the thesis follows from the induction hypothesis and Lemma A.2.7.

□

Proof (Soundness). We proceed by induction on the structure of C_f.

- **Case** $C_f = \mathtt{p}[\mathtt{A}].e \texttt{ -> } \mathtt{q}[\mathtt{B}].x : k; C_b$. The induction hypothesis is:

$$\begin{aligned}\text{for all } \tilde{\mu_1} \quad [\![(\boldsymbol{\nu}\tilde{\mathtt{p}k})\ C_b]\!] \overset{\tilde{\mu_1}}{\rightsquigarrow}{}^* P_1 \ \Rightarrow P_1 &\overset{\tilde{\mu_1'}}{\rightsquigarrow}{}^* P_1' \\ (\boldsymbol{\nu}\tilde{\mathtt{p}k})\ C_b &\overset{\tilde{\lambda'}}{\rightsquigarrow}{}^* C_1' \\ \tilde{\lambda'} &\vdash \tilde{\mu_1}, \tilde{\mu_1'} \\ [\![C_1']\!] &\prec P_1'\end{aligned}$$

From the definition of EPP we have

$$\begin{aligned}[\![C]\!] \ \equiv (\boldsymbol{\nu}\tilde{k}) \Bigg(\prod_{\mathtt{p} \in \mathsf{fn}(C_b)\setminus \mathtt{p},\mathtt{q}} [\![C_b]\!]_{\mathtt{p}} \mid k[\mathtt{A}]!\mathtt{B}\langle e\rangle; [\![C_b]\!]_{\mathtt{p}} \mid k[\mathtt{B}]?\mathtt{A}(x); [\![C_b]\!]_{\mathtt{q}} \\ \mid \prod_{k' \in \mathsf{fn}(C_b)\setminus k} k' : \emptyset \mid k : \emptyset \Bigg) \mid \prod_{a,\mathtt{A}} \Bigg(\bigsqcup_{\mathtt{p} \in \lfloor C_b \rfloor^a_{\mathtt{A}}} [\![C_b]\!]_{\mathtt{p}} \Bigg)\end{aligned}$$

The projection of $\mathtt{q}$ is stuck on an input and the projection of $\mathtt{p}$ can perform an output. Lemma 2.5.1 allows us not to consider potential swaps in C when looking at its EPP, since it remains unchanged. From the induction hypothesis and the semantics for the choreography calculus we get that

$$C \overset{\tilde{\lambda}}{\rightsquigarrow}{}^* C_1'[v/x@\mathtt{q}] \ = \ C' \qquad (e \downarrow v)$$

where $\tilde{\lambda} = \tilde{\lambda_1'}, \mathtt{p}[\mathtt{A}].v \texttt{ -> } \mathtt{q}[\mathtt{B}].x : k, \tilde{\lambda_2'}$ for some λ_1' and λ_2' such that $\tilde{\lambda'} = \tilde{\lambda_1'}, \tilde{\lambda_2'}$. Now we have two cases, depending on the actions performed in the reduction chain $[\![C]\!] \overset{\tilde{\mu}}{\rightsquigarrow}{}^* P$. If the projection of $\mathtt{p}$ does not perform its output in these reductions, then we apply the induction hypothesis choosing $\tilde{\mu} = \tilde{\mu_1}$ and we obtain

$$\begin{aligned}[\![C]\!] \overset{\tilde{\mu_1}}{\rightsquigarrow}{}^* (\boldsymbol{\nu}\tilde{k''}) \Bigg(P_b \mid k[\mathtt{A}]!\mathtt{B}\langle e\rangle; [\![C_b]\!]_{\mathtt{p}} \mid k[\mathtt{B}]?\mathtt{A}(x); [\![C_b]\!]_{\mathtt{q}} \mid k : h \Bigg) \\ \mid \prod_{a,\mathtt{A}} \Bigg(\bigsqcup_{\mathtt{p} \in \lfloor C_b \rfloor^a_{\mathtt{A}}} [\![C_b]\!]_{\mathtt{p}} \Bigg) \ = \ P\end{aligned}$$

where P_b may contain some new processes spawned by starting some services and some session queues. From Lemma A.2.4 we also know that $(\mathtt{A}, \mathtt{B}, \boldsymbol{w}) \notin h$ for any $\boldsymbol{w}$, since all the other processes cannot play the same roles of $\mathtt{p}$ and $\mathtt{q}$ in k. Now we need to reduce P to P'. Applying rules $\lfloor^{\mathtt{P}}|_{\text{COM-S}}\rceil$ and $\lfloor^{\mathtt{P}}|_{\text{COM-R}}\rceil$ we obtain

$$\begin{aligned}P \overset{\mathtt{!A} \texttt{ -> } \mathtt{B}:k\langle v\rangle}{\rightsquigarrow} \overset{\mathtt{?A} \texttt{ -> } \mathtt{B}:k\langle v\rangle}{\rightsquigarrow} (\boldsymbol{\nu}\tilde{k''}) \Bigg(P_b \mid [\![C_b]\!]_{\mathtt{p}} \mid [\![C_b]\!]_{\mathtt{q}}[v/x] \mid k : h \Bigg) \\ \mid \prod_{a,\mathtt{A}} \Bigg(\bigsqcup_{\mathtt{p} \in \lfloor C_b \rfloor^a_{\mathtt{A}}} [\![C_b]\!]_{\mathtt{p}} \Bigg) \ = \ Q\end{aligned}$$

where $e \downarrow v$. From well-typedness we know that $[\![C_b]\!]_{\mathtt{q}}$ is the only process using name x. Therefore $Q = P_1[v/x]$. The thesis follows now from the induction hypothesis, choosing $\tilde{\mu'} = \mathtt{!A\ ->\ B} : k\langle v\rangle, \mathtt{?A\ ->\ B} : k\langle v\rangle, \tilde{\mu'_1}$.

Let us now consider the case in which the projection of $\mathtt{p}$ performs its output in the first reduction chain of $[\![C]\!]$. By applying similar reasoning to the previous case, we can split the reduction chain $\tilde{\mu}$ as follows. First we have:

$$[\![C]\!] \overset{\tilde{\mu_1}}{\rightsquigarrow}{}^{*} (\boldsymbol{\nu}\tilde{k''}) \left(P_b \mid k[\mathtt{A}]!\mathtt{B}\langle e\rangle; [\![C_b]\!]_{\mathtt{p}} \mid k[\mathtt{B}]?\mathtt{A}(x); [\![C_b]\!]_{\mathtt{q}} \mid k : h \right)$$
$$\mid \prod_{a,\mathtt{A}} \left(\bigsqcup_{\mathtt{p} \in \lfloor C_b \rfloor^a_{\mathtt{A}}} [\![C_b]\!]_{\mathtt{p}} \right) = P_{1_1}$$

where $(\mathtt{A}, \mathtt{B}, \boldsymbol{w}) \notin h$ for any $\boldsymbol{w}$. When $\mathtt{p}$ performs the output we obtain:

$$P_{1_1} \overset{\mathtt{!A\ ->\ q}:k\langle v\rangle}{\rightsquigarrow} (\boldsymbol{\nu}\tilde{k''}) \left(P_b \mid [\![C_b]\!]_{\mathtt{p}} \mid k[\mathtt{B}]?\mathtt{A}(x); [\![C_b]\!]_{\mathtt{q}} \mid k : h \cdot (\mathtt{A}, \mathtt{B}, v) \right)$$
$$\mid \prod_{a,\mathtt{A}} \left(\bigsqcup_{\mathtt{p} \in \lfloor C_b \rfloor^a_{\mathtt{A}}} [\![C_b]\!]_{\mathtt{p}} \right) = P_{1_2}$$

where $e \downarrow v$. Now there are two subcases: the projection of $\mathtt{q}$ may perform its input in the reduction chain from P_{1_2} to P or not. Let us see the first subcase. We have:

$$P_{1_2} \overset{\tilde{\mu_{1_2}}}{\rightsquigarrow}{}^{*} \overset{\mathtt{?A\ ->\ q}:k\langle v\rangle}{\rightsquigarrow} (\boldsymbol{\nu}\tilde{k''}) \left(P_b \mid [\![C_b]\!]_{\mathtt{p}} \mid ([\![C_b]\!]_{\mathtt{q}}[v/x]) \mid k : h \right)$$
$$\mid \prod_{a,\mathtt{A}} \left(\bigsqcup_{\mathtt{p} \in \lfloor C_b \rfloor^a_{\mathtt{A}}} [\![C_b]\!]_{\mathtt{p}} \right) = P_{1_3}$$

Therefore we can split the reduction chain from $[\![C]\!]$ to P as:

$$[\![C]\!] \overset{\tilde{\mu_1}}{\rightsquigarrow}{}^{*} P_{1_1} \overset{\mathtt{!A\ ->\ B}:k\langle v\rangle}{\rightsquigarrow} P_{1_2} \overset{\tilde{\mu_{1_2}}}{\rightsquigarrow}{}^{*} \overset{\mathtt{?A\ ->\ B}:k\langle v\rangle}{\rightsquigarrow} P_{1_3} \overset{\tilde{\mu_{1_3}}}{\rightsquigarrow}{}^{*} P = P_1[v/x]$$

The thesis now follows by induction hypothesis.

Let us see the other subcase, in which $\mathtt{q}$ does not perform the input in the reduction chain from $[\![C]\!]$ to P. By similar reasoning we can split the reduction chain as:

$$[\![C]\!] \overset{\tilde{\mu_1}}{\rightsquigarrow}{}^{*} P_{1_1} \overset{\mathtt{!A\ ->\ B}:k\langle v\rangle}{\rightsquigarrow} P_{1_2} \overset{\tilde{\mu_{1_2}}}{\rightsquigarrow}{}^{*} P$$

Now we can choose to perform the input in the reduction chain from P to P':

$$P \overset{\mathtt{?A\ ->\ B}:k\langle v\rangle}{\rightsquigarrow} P'' \overset{\tilde{\mu'_1}}{\rightsquigarrow}{}^{*} P'$$

The thesis now follows by similar reasoning to the previous subcase.

- **Case** $C_f = \mathtt{p[A]} \texttt{ -> } \mathtt{q[B]} : k[l]; C_b$. Similar to the case $C_f = \mathtt{p[A]}.e \texttt{ -> } \mathtt{q[B]}.x : k; C_b$.

- **Case** $C_f = \mathtt{p[A]} \texttt{ -> } \mathtt{q[B]} : k\langle k'[\mathtt{C}]\rangle; C_b$. Similar to the case $C_f = \mathtt{p[A]}.e \texttt{ -> } \mathtt{q[B]}.x : k; C_b$.

- **Case** $C_f = \widetilde{\mathtt{p[A]}}\ \mathbf{start}\ \widetilde{\mathtt{q[B]}} : a(k); C_b$. The induction hypothesis is:

$$\begin{aligned}\text{for all } \tilde{\mu_1} \quad [\![(\boldsymbol{\nu}\tilde{\tau}\tilde{k})\ C_b]\!] \overset{\tilde{\mu_1}}{\rightsquigarrow}{}^* P_1 \ \Rightarrow P_1 &\overset{\tilde{\mu_1'}}{\rightsquigarrow}{}^* P_1' \\ (\boldsymbol{\nu}\tilde{\mathtt{p}}\tilde{k})\ C_b &\overset{\tilde{\lambda_1}}{\rightsquigarrow}{}^* C_1' \\ \tilde{\lambda_1} \ &\vdash \ \tilde{\mu_1}, \tilde{\mu_1'} \\ [\![C_1']\!] &\prec P_1'\end{aligned}$$

  The thesis can now be proven with similar reasoning as for the case $C_f = \mathtt{p[A]}.e \texttt{ -> } \mathtt{q[B]}.x : k; C_b$. The only important difference is that we have to check that the projection of

$$\widetilde{\mathtt{p[A]}}\ \mathbf{start}\ \widetilde{\mathtt{q[B]}} : a(k)$$

  should not introduce any race on a. This is guaranteed by Lemma A.2.3.

- **Case** $C_f = \mathsf{if}\ e@\mathtt{p}\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2$. From the definition of EPP we have:

$$\begin{aligned}[\![C]\!] \ \equiv (\boldsymbol{\nu}\tilde{k}) \Bigg(\prod_{\mathtt{q} \in \mathsf{fn}(C_f)\setminus \mathtt{p}} [\![C_1]\!]_\mathtt{q} \sqcup [\![C_2]\!]_\mathtt{q} \ \mid\ \mathsf{if}\ e\ \mathsf{then}\ [\![C_1]\!]_\mathtt{p}\ \mathsf{else}\ [\![C_2]\!]_\mathtt{p} \\ \mid \prod_{k' \in \mathsf{fn}(C_f)} k' : \emptyset \Bigg) \mid \prod_{a,\mathtt{A}} \Bigg(\bigsqcup_{\mathtt{q} \in \lfloor C_f \rfloor^a_\mathtt{A}} [\![C_1]\!]_\mathtt{q} \sqcup [\![C_2]\!]_\mathtt{q} \Bigg)\end{aligned}$$

  Let $e \downarrow$ true (the other case is similar). The induction hypothesis is then: The induction hypothesis is:

$$\begin{aligned}\text{for all } \tilde{\mu_1} \quad [\![(\boldsymbol{\nu}\tilde{\mathtt{p}}\tilde{k})\ C_1]\!] \overset{\tilde{\mu_1}}{\rightsquigarrow}{}^* P_1 \ \Rightarrow P_1 &\overset{\tilde{\mu_1'}}{\rightsquigarrow}{}^* P_1' \\ (\boldsymbol{\nu}\tilde{\mathtt{p}}\tilde{k})\ C_1 &\overset{\tilde{\lambda'}}{\rightsquigarrow}{}^* C_1' \\ \tilde{\lambda'} \ &\vdash \ \tilde{\mu_1}, \tilde{\mu_1'} \\ [\![C_1']\!] &\prec P_1'\end{aligned}$$

  From the induction hypothesis and the semantics for the choreography calculus we get that:

$$C \overset{\tilde{\lambda}}{\rightsquigarrow}{}^* C_1' \ = \ C'$$

  where $\tilde{\lambda} = \tilde{\lambda_1'}, \tau@\mathtt{p}, \tilde{\lambda_2'}$ for some λ_1' and λ_2' such that $\tilde{\lambda'} = \tilde{\lambda_1'}, \tilde{\lambda_2'}$.

Now we have two cases, depending on the actions performed in the reduction chain $[\![C]\!] \overset{\tilde{\mu}}{\rightsquigarrow}{}^* P$. If the projection of p does not execute its conditional in these reductions, then we apply the induction hypothesis choosing $\tilde{\mu} = \tilde{\mu_1}$ and we obtain:

$$[\![C]\!] \overset{\tilde{\mu_1}}{\rightsquigarrow}{}^* (\boldsymbol{\nu}\tilde{k''}) \left(P_b \mid \mathsf{if}\, e\, \mathsf{then}\, [\![C_1]\!]_{\mathsf{p}}\, \mathsf{else}\, [\![C_2]\!]_{\mathsf{p}} \right) \mid \prod_{a,\mathtt{A}} \left(\bigsqcup_{\mathsf{q} \in \lfloor C_f \rfloor^a_{\mathtt{A}}} [\![C_1]\!]_{\mathsf{q}} \sqcup [\![C_2]\!]_{\mathsf{q}} \right) \;=\; P$$

where P_b may contain some new processes spawned by starting some services and some session queues. Now we need to reduce P to P'. Applying rule $\lfloor {}^{\mathrm{P}} | {}_{\mathrm{COND}} \rceil$ we obtain:

$$P \overset{\tau}{\rightsquigarrow} (\boldsymbol{\nu}\tilde{k''}) \left(P_b \mid [\![C_1]\!]_{\mathsf{p}} \right) \mid \prod_{a,\mathtt{A}} \left(\bigsqcup_{\mathsf{q} \in \lfloor C_f \rfloor^a_{\mathtt{A}}} [\![C_1]\!]_{\mathsf{q}} \sqcup [\![C_2]\!]_{\mathsf{q}} \right) \;=\; P'$$

The thesis follows now from the induction hypothesis and the definition of pruning, choosing $\tilde{\mu'} = \tau, \tilde{\mu'_1}$.

Let us now consider the case in which the projection of p executes its conditional in the reduction chain $[\![C]\!] \overset{\tilde{\mu}}{\rightsquigarrow}{}^* P$. By applying similar reasoning to the previous case, we can split the reduction chain $\tilde{\mu}$ as follows. First we have:

$$[\![C]\!] \overset{\tilde{\mu_1}}{\rightsquigarrow}{}^* (\boldsymbol{\nu}\tilde{k''}) \left(P_b \mid \mathsf{if}\, e\, \mathsf{then}\, [\![C_1]\!]_{\mathsf{p}}\, \mathsf{else}\, [\![C_2]\!]_{\mathsf{p}} \right) \mid \prod_{a,\mathtt{A}} \left(\bigsqcup_{\mathsf{q} \in \lfloor C_f \rfloor^a_{\mathtt{A}}} [\![C_1]\!]_{\mathsf{q}} \sqcup [\![C_2]\!]_{\mathsf{q}} \right) \;=\; P_{1_1}$$

When p executes the conditional we obtain:

$$P_{1_1} \overset{\tau}{\rightsquigarrow} (\boldsymbol{\nu}\tilde{k''}) \left(P_b \mid [\![C_1]\!]_{\mathsf{p}} \right) \mid \prod_{a,\mathtt{A}} \left(\bigsqcup_{\mathsf{q} \in \lfloor C_f \rfloor^a_{\mathtt{A}}} [\![C_1]\!]_{\mathsf{q}} \sqcup [\![C_2]\!]_{\mathsf{q}} \right) \overset{\tilde{\mu_2}}{\rightsquigarrow}{}^* \succ P_1$$

The thesis now follows by induction hypothesis.

- **Case** $C_f = \mathbf{0}$. Trivial.
- **Case** $C_f = \mathsf{def}\, X(\tilde{D}) = C_1 \,\mathsf{in}\, C_2$. Follows by Lemma A.2.7 and the induction hypothesis.
- **Case** $C_f = X\langle \tilde{E} \rangle$. This case is not allowed by the hypothesis that C is well-typed.
- **Case** $C_f = (\boldsymbol{\nu} r)\, C_b$. Follows immediately by induction hypothesis.

□

APPENDIX B

Compositional Choreographies: Additional Material

B.1 Complete Use Case

We report an extended version of our case study, in which the buyer choreography also performs an internal authentication of its user before proceeding with the purchase.

Buyer Choreography. We define the choreography for the buyer, C_B. For clarity, we have extracted two parts of it in separate terms, C'_B and C''_B. C_B starts with a choreography for its internal network, where it perform some checks for authenticating a user. Then, it proceeds by dynamically selecting a seller in C'_B, which is finally contacted with a partial choreography in C''_B. The code follows.

$C_B =$

1. $\mathtt{u}[\mathtt{U}]$ **start** $\mathtt{a}[\mathtt{A}], \mathtt{pd}[\mathtt{PD}] : a(k)$; $\quad \mathtt{u}[\mathtt{U}].cred \to \mathtt{a}[\mathtt{A}].x : k$;
2. if $\mathsf{auth}(x)@\mathtt{a}$ then
3. $\quad \mathtt{a}[\mathtt{A}] \to \mathtt{pd}[\mathtt{PD}] : k[ok]$; $\quad \mathtt{pd}[\mathtt{PD}] \to \mathtt{u}[\mathtt{u}] : k[ok]$;
4. $\quad \mathtt{u}[\mathtt{U}].prod \to \mathtt{pd}[\mathtt{PD}].y : k$; C'_B
5. else
6. $\quad \mathtt{a}[\mathtt{A}] \to \mathtt{pd}[\mathtt{PD}] : k[quit]$; $\quad \mathtt{pd}[\mathtt{PD}] \to \mathtt{u}[\mathtt{u}] : k[quit]$

$C'_B =$

7. $\mathtt{pd}[\mathtt{PD}]$ **start** $\mathtt{r}[\mathtt{R}] : b(k')$; $\quad \mathtt{pd}[\mathtt{PD}].y \to \mathtt{r}[\mathtt{R}].z : k'$;
8. $\mathtt{r}[\mathtt{R}].\mathsf{find}(z) \to \mathtt{pd}[\mathtt{PD}].w : k'$; $\quad C''_B$

$C''_B =$

9. $\mathtt{pd}[\mathtt{B}]$ **req** $\mathtt{C}, \mathtt{T} : w(k'')$; $\quad \mathtt{pd}[\mathtt{B}].y \to \mathtt{C} : k''$;
10. $\mathtt{C} \to \mathtt{pd}[\mathtt{B}].x_2 : k''$;
11. if $\mathsf{check}(x_2)@\mathtt{pd}$ then
12. $\quad \mathtt{pd}[\mathtt{B}] \to \mathtt{T} : k'' \oplus ok$; $\quad \mathtt{pd}[\mathtt{PD}] \to \mathtt{u}[\mathtt{U}] : k[del]$; $\quad \mathtt{pd}[\mathtt{PD}] \to \mathtt{u}[\mathtt{U}] : k\langle k''[\mathtt{B}]\rangle$;
13. $\quad \mathtt{u}[\mathtt{B}].addr \to \mathtt{T} : k''$; $\quad \mathtt{T} \to \mathtt{u}[\mathtt{B}].ddate : k''$
14. else
15. $\quad \mathtt{pd}[\mathtt{B}] \to \mathtt{T} : k'' \oplus quit$; $\quad \mathtt{pd}[\mathtt{PD}] \to \mathtt{u}[\mathtt{U}] : k[quit]$

We comment the code above. Inside the buyer, a purchase is initiated by a user process u. In Line 1, process u and the freshly created processes a and pd start a session k by synchronising on shared channel a. Each process is annotated with the role it plays in the protocol that the session implements (we omit the protocol for session k). Then, u sends her credentials "$cred$" to a. In Line 2, a tries to authenticate the credentials. If successful, in Line 3 the choice ok is communicated to the others and in Line 4 u sends the name of the product she wishes to purchase to pd; the choreography proceeds then as C'_B. Otherwise, the choice $quit$ is communicated from a to the others.

C'_B defines how the buyer actually purchases the product. In our use case, the buyer has a registry of suppliers that reports which seller should be used for purchasing each kind of product. This aspect is captured by *mobility of shared channels*. In Line 7 pd starts a new session k' with a fresh process r through shared channel b. Then, pd sends the name

of the product to be purchased to r. In Line 8, r sends to pd the name of the shared channel to contact the selected seller, computed with the internal function find(z), to pd. Finally, the choreography proceeds as C''_B.

C''_B is a partial choreography that relies on an external seller to implement the protocol shown in the introduction and perform the purchase. In Line 9, pd *requests* a synchronisation on the shared channel stored in its local variable w to create the new session k''. pd declares that it will play role B, and that it expects the environment to implement roles C and T for session k''. Session k'' proceeds now as specified by the protocol in the introduction. First, right after the request in Line 9, pd sends the product name y through session k'' to the external process that is playing role C (the product catalogue executed by the seller). Observe that here we do not specify the actual process name of the receiver, since that will be established by the environment. In Line 10, pd waits to receive the price for the product from the external process playing role C in k''. In Line 11, pd checks whether the price is acceptable. If so, in Line 12 it will tell the external process playing role T (the transport process executed by the seller) and user u (which remains internal to the buyer choreography) to proceed with the purchase (labels ok and del respectively). Still in Line 13, pd will also *delegate* to u the continuation of session k'' in its place, as role B. In Line 14, the user can now send her delivery address to T and receive the expected delivery date. If the price is not acceptable, then in Lines 15-16 pd informs the others to quit the purchase attempt.

Seller Choreography and Composition. We define now a choreography for a seller that can be contacted by C_B (through C''_B). The exact seller system contacted by C''_B depends on the result of find(z) in C'_B. Let us assume that find returns shared channel c for computer-related products, and c' for every other kind of products; we refer to the choreography implementations of the respective seller companies as C_S and C'_S. Below, we define the implementation for the first (the second is similar).

$$
C_S = \begin{array}{ll}
1. & \mathbf{acc}\ \mathtt{c}[\mathtt{C}], \mathtt{t}[\mathtt{T}] : c(k'');\quad \mathtt{B} \text{ -> } \mathtt{c}[\mathtt{C}].y_2 : k'';\\
2. & \mathtt{c}[\mathtt{C}].\mathsf{price}(y_2) \text{ -> } \mathtt{B} : k'';\\
3. & \mathtt{B} \text{ -> } \mathtt{t}[\mathtt{T}] : k'' \& \left\{ \begin{array}{ll} ok: & \mathtt{B} \text{ -> } \mathtt{t}[\mathtt{T}].daddr : k'';\quad \mathtt{t}[\mathtt{T}].\mathsf{time}(daddr) \text{ -> } \mathtt{B} : k'' \\ quit: & \mathbf{0} \end{array} \right\}
\end{array}
$$

Above, the seller choreography C_S starts by *accepting* the creation of session k'' through shared channel c, offering to spawn two fresh processes c and t. Choreographies starting with an acceptance act as replicated processes, modeling typical always-available modules. The acceptance in Line 1 would synchronise with the request made by C'' in the case for $w = c$. Right afterwards, still in Line 1, c[C] expects to receive the product name from the process playing B in session k''. In Line 2, c sends back the price for the product. Finally, in Line 3, t (the process for the transport) waits to receive either label ok or $quit$. In the first case, t will also wait to receive a delivery address and send back the expected time of arrival.

Now that we have the code for both buyer and seller companies (we omit the code for (C'_S), we can compose their choreographies in a network with the parallel operator | as: $C = C_B \mid C_S \mid C'_S$.

$$\frac{\Gamma, a{:}G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{B}}\rangle, \Gamma' \vdash C \triangleright \Delta, \Delta' \quad (\Gamma', \Delta') = \mathsf{init}(\mathtt{p}[\mathtt{A}], \widetilde{\mathtt{q}[\mathtt{B}]}, k, G) \quad \tilde{\mathtt{q}} \notin \Gamma}{\Gamma, a{:}G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{B}}\rangle \vdash \mathtt{p}[\mathtt{A}]\ \mathbf{start}\ \widetilde{\mathtt{q}[\mathtt{B}]} : a(k); C \triangleright \Delta} \lfloor^{\mathrm{T}}|_{\textsc{Start}}\rceil$$

$$\frac{\Gamma \vdash \mathtt{p}[\mathtt{A}] \texttt{->} \mathtt{q}[\mathtt{B}] : k \quad \Gamma \vdash e@\mathtt{p} : S \quad \Gamma, x@\mathtt{q} : S \vdash C \triangleright \Delta, k[\mathtt{A}] : T, k[\mathtt{B}] : T'}{\Gamma \vdash \mathtt{p}[\mathtt{A}].e \texttt{->} \mathtt{q}[\mathtt{B}].x : k; C \triangleright \Delta, k[\mathtt{A}] : !\mathtt{B}\langle S\rangle; T, k[\mathtt{B}] : ?\mathtt{A}\langle S\rangle; T'} \lfloor^{\mathrm{T}}|_{\textsc{Com}}\rceil$$

$$\frac{\Gamma \vdash \mathtt{p}[\mathtt{A}] \texttt{->} \mathtt{q}[\mathtt{B}] : k \quad \Gamma, \mathtt{q} : k'[\mathtt{C}] \vdash C \triangleright \Delta, k[\mathtt{A}] : T, k[\mathtt{B}] : T', k'[\mathtt{C}] : T''}{\Gamma, \mathtt{p} : k'[\mathtt{C}] \vdash \mathtt{p}[\mathtt{A}] \texttt{->} \mathtt{q}[\mathtt{B}] : k\langle k'[\mathtt{C}]\rangle; C \triangleright \Delta, k[\mathtt{A}] : !\mathtt{B}\langle T''\rangle; T, k[\mathtt{B}] : ?\mathtt{A}\langle T''\rangle; T', k'[\mathtt{C}] : T''} \lfloor^{\mathrm{T}}|_{\textsc{Del}}\rceil$$

$$\frac{\Gamma \vdash \mathtt{p}[\mathtt{A}] \texttt{->} \mathtt{q}[\mathtt{B}] : k \quad \Gamma \vdash C \triangleright \Delta, k[\mathtt{A}] : T_j, k[\mathtt{B}] : T'_j \quad j \in I}{\Gamma \vdash \mathtt{p}[\mathtt{A}] \texttt{->} \mathtt{q}[\mathtt{B}] : k[l_j]; C \triangleright \Delta, k[\mathtt{A}] : \oplus\mathtt{B}\{l_i : T_i\}_{i\in I}, k[\mathtt{B}] : \&\mathtt{A}\{l_i : T'_i\}_{i\in I}} \lfloor^{\mathrm{T}}|_{\textsc{Sel}}\rceil$$

$$\frac{\Gamma \vdash e@\mathtt{p} : \mathbf{bool} \quad \Gamma \vdash C_i \triangleright \Delta}{\Gamma \vdash \mathsf{if}\ e@\mathtt{p}\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2 \triangleright \Delta} \lfloor^{\mathrm{T}}|_{\textsc{Cond}}\rceil \qquad \frac{\mathsf{cosha}(\Gamma) \quad \Delta\ \mathsf{end\ only}}{\Gamma \vdash \mathbf{0} \triangleright \Delta} \lfloor^{\mathrm{T}}|_{\textsc{End}}\rceil$$

$$\frac{\Gamma, a{:}G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{B}}\rangle \vdash C \triangleright \Delta}{\Gamma \vdash (\boldsymbol{\nu} a)\, C \triangleright \Delta} \lfloor^{\mathrm{T}}|_{\textsc{ResSha}}\rceil$$

$$\frac{\begin{array}{c}\mathsf{end}(\Delta) \quad \Gamma'' \vdash x_{ij}@\mathtt{p}_i : S_{ij} \quad \Gamma \vdash e_{ij}@\mathtt{p}_i : S_{ij} \quad \Gamma' \subseteq \Gamma \\ D = \mathtt{p}_1(\tilde{x_1}, \tilde{k_1}), \ldots, \mathtt{p}_n(\tilde{x_n}, \tilde{k_n}) \quad E = \mathtt{p}_1(\tilde{e_1}, \tilde{k_1}), \ldots, \mathtt{p}_n(\tilde{e_n}, \tilde{k_n})\end{array}}{\Gamma, X(\tilde{D}) : (\Gamma', \Gamma''; \Delta') \vdash X\langle \tilde{E}\rangle \triangleright \Delta, \Delta'} \lfloor^{\mathrm{T}}|_{\textsc{Call}}\rceil$$

$$\frac{\Gamma, X(\tilde{D}) : (\Gamma'; \Delta') \vdash C \triangleright \Delta \quad \Gamma', X(\tilde{D}) : (\Gamma'; \Delta') \vdash C' \triangleright \Delta' \quad \Gamma'|_{\mathsf{sha}} \subseteq \Gamma}{\Gamma \vdash \mathsf{def}\ X(\tilde{D}) = C'\ \mathsf{in}\ C \triangleright \Delta} \lfloor^{\mathrm{T}}|_{\textsc{Def}}\rceil$$

$$\frac{\Gamma \vdash C \triangleright \Delta \quad \mathsf{co}(\Delta, k)}{\Gamma \setminus k \vdash (\boldsymbol{\nu} k)\, C \triangleright \Delta \setminus k} \lfloor^{\mathrm{T}}|_{\textsc{ResLin}}\rceil \qquad \frac{\Gamma \vdash C \triangleright \Delta}{\Gamma \setminus \mathtt{p} \vdash (\boldsymbol{\nu}\mathtt{p})\, C \triangleright \Delta} \lfloor^{\mathrm{T}}|_{\textsc{ResProc}}\rceil$$

Figure B.1: Compositional Choreographies, typing rules for complete terms.

B.2 Typing

We report all the rules of our typing system and the proof of their soundness.

B.2.1 Typing rules

Figure B.1 reports the rules for typing complete terms, while Figure B.2 the rules for typing partial terms. In rule $\lfloor^{\mathrm{T}}|_{\textsc{ResLin}}\rceil$, the predicate $\mathsf{co}(\Delta, k)$ (from [21]) holds if there exists G such that:

$$\Delta|_k = \{k[\mathtt{A}] : [\![G]\!]_{\mathtt{A}} \mid \mathtt{A} \in \mathsf{roles}(G)\}$$

In other words, the portion of Δ typing session k must be the projection of some global type G. All the other typing rules follow the reasoning reported in § 3.4.

$$\frac{\Gamma \vdash a : G\langle \mathtt{A}|\tilde{\mathtt{B}}|\emptyset\rangle \qquad \Gamma, \mathsf{p} : k[\mathtt{A}] \vdash C \triangleright \Delta, k[\mathtt{A}] : [\![G]\!]_{\mathtt{A}}}{\Gamma \vdash \mathsf{p}[\mathtt{A}]\ \mathbf{req}\ \tilde{\mathtt{B}} : a(k); C \triangleright \Delta} \lfloor^{\mathrm{T}}|_{\text{REQ1}}\rceil$$

$$\frac{\Gamma \vdash x@\mathsf{p} : G\langle \mathtt{A}|\tilde{\mathtt{B}}|\emptyset\rangle \qquad \Gamma, \mathsf{p} : k[\mathtt{A}] \vdash C \triangleright \Delta, k[\mathtt{A}] : [\![G]\!]_{\mathtt{A}}}{\Gamma \vdash \mathsf{p}[\mathtt{A}]\ \mathbf{req}\ \tilde{\mathtt{B}} : x(k); C \triangleright \Delta} \lfloor^{\mathrm{T}}|_{\text{REQ2}}\rceil$$

$$\frac{\Gamma, a : G\langle \mathtt{D}|\tilde{\mathtt{B}}|\emptyset\rangle, \Gamma' \vdash C \triangleright \Delta, \Delta' \quad (\Gamma', \Delta') = \mathsf{init}(\widetilde{\mathsf{q}[\mathtt{A}]}, k, G) \quad \tilde{\mathsf{q}} \notin \Gamma}{\Gamma, a : G\langle \mathtt{D}|\tilde{\mathtt{B}}|\tilde{\mathtt{A}}\rangle \vdash \mathbf{acc}\ \widetilde{\mathsf{q}[\mathtt{A}]} : a(k); C \triangleright \Delta} \lfloor^{\mathrm{T}}|_{\text{ACC}}\rceil$$

$$\frac{\Gamma \vdash e@\mathsf{p} : S \quad \Gamma \vdash \mathsf{p}[\mathtt{A}] \texttt{->} \mathtt{B} : k \quad \Gamma \vdash C \triangleright \Delta, k[\mathtt{A}] : T \quad \mathsf{q} : k[\mathtt{B}] \notin \Gamma}{\Gamma \vdash \mathsf{p}[\mathtt{A}].e \texttt{->} \mathtt{B} : k; C \triangleright \Delta, k[\mathtt{A}] : !\mathtt{B}\langle S\rangle; T} \lfloor^{\mathrm{T}}|_{\text{COM-S}}\rceil$$

$$\frac{\Gamma \vdash \mathtt{A} \texttt{->} \mathsf{q}[\mathtt{B}] : k \quad \Gamma, x@\mathsf{p} : S \vdash C \triangleright \Delta, k[\mathtt{B}] : T \quad \mathsf{p} : k[\mathtt{A}] \notin \Gamma}{\Gamma \vdash \mathtt{A} \texttt{->} \mathsf{q}[\mathtt{B}].x : k; C \triangleright \Delta, k[\mathtt{A}] : ?\mathtt{B}\langle S\rangle; T} \lfloor^{\mathrm{T}}|_{\text{COM-R}}\rceil$$

$$\frac{\Gamma \vdash \mathsf{p}[\mathtt{A}] \texttt{->} \mathtt{B} : k \quad \Gamma \vdash C \triangleright \Delta, k[\mathtt{A}] : T \quad \mathsf{q} : k[\mathtt{B}] \notin \Gamma}{\Gamma, \mathsf{p} : k'[\mathtt{C}] \vdash \mathsf{p}[\mathtt{A}] \texttt{->} \mathtt{B} : k\langle k'[\mathtt{C}]\rangle; C \triangleright \Delta, k[\mathtt{A}] : !\mathtt{B}\langle T''\rangle; T, k'[\mathtt{C}] : T''} \lfloor^{\mathrm{T}}|_{\text{DEL-S}}\rceil$$

$$\frac{\Gamma \vdash \mathtt{A} \texttt{->} \mathsf{q}[\mathtt{B}] : k \quad \Gamma, \mathsf{q} : k'[\mathtt{C}] \vdash C \triangleright \Delta, k[\mathtt{B}] : T', k'[\mathtt{C}] : T'' \quad \mathsf{p} : k[\mathtt{A}] \notin \Gamma}{\Gamma \vdash \mathtt{A} \texttt{->} \mathsf{q}[\mathtt{B}] : k(k'[\mathtt{C}]); C \triangleright \Delta, k[\mathtt{B}] : ?\mathtt{A}\langle T''\rangle; T'} \lfloor^{\mathrm{T}}|_{\text{DEL-R}}\rceil$$

$$\frac{\Gamma \vdash \mathsf{p}[\mathtt{A}] \texttt{->} \mathtt{B} : k \quad j \in I \quad \Gamma \vdash C \triangleright \Delta, k[\mathtt{A}] : T_j \quad \mathsf{q} : k[\mathtt{B}] \notin \Gamma}{\Gamma \vdash \mathsf{p}[\mathtt{A}] \texttt{->} \mathtt{B} : k \oplus l_j; C \triangleright \Delta, k[\mathtt{A}] : \oplus\mathtt{B}\{l_i : T_i\}_{i\in I}} \lfloor^{\mathrm{T}}|_{\text{SEL-S}}\rceil$$

$$\frac{\Gamma \vdash \mathtt{A} \texttt{->} \mathsf{q}[\mathtt{B}] : k \quad \Gamma \vdash C_i \triangleright \Delta, k[\mathtt{A}] : T_i \quad J \subseteq I \quad \mathsf{p} : k[\mathtt{A}] \notin \Gamma}{\Gamma \vdash \mathtt{A} \texttt{->} \mathsf{q}[\mathtt{B}] : k\&\{l_i : C_i\}_{i\in I} \triangleright \Delta, k[\mathtt{B}] : \&\mathtt{A}\{l_j : T_j\}_{j\in J}} \lfloor^{\mathrm{T}}|_{\text{BRANCH}}\rceil$$

$$\frac{\Gamma_i \vdash C_i \triangleright \Delta_i \quad \mathsf{pco}(\Delta_1, \Delta_2)}{\Gamma_1 \circ \Gamma_2 \vdash C_1 \mid C_2 \triangleright \Delta_1, \Delta_2} \lfloor^{\mathrm{T}}|_{\text{PAR}}\rceil$$

Figure B.2: Compositional Choreographies, typing rules for partial terms.

$$\frac{\Gamma \vdash \mathtt{p}[\mathtt{A}] \to \mathtt{q}[\mathtt{B}] : k \quad \vdash v : S}{\Gamma \vdash \mathtt{p}[\mathtt{A}] \to \mathtt{q}[\mathtt{B}] : k\langle v\rangle \triangleright k : \mathtt{A} \to \mathtt{B} : \langle S\rangle} \lfloor^{\mathrm{LT}}|_{\textsc{Com}}\rceil$$

$$\frac{\Gamma \vdash \mathtt{p}[\mathtt{A}] \to \mathtt{B} : k \quad \vdash v : S}{\Gamma \vdash \mathtt{p}[\mathtt{A}] \to \mathtt{B} : k\langle v\rangle \triangleright k[\mathtt{A}] : !\mathtt{B}\langle S\rangle} \lfloor^{\mathrm{LT}}|_{\textsc{Com-S}}\rceil \qquad \frac{\Gamma \vdash \mathtt{A} \to \mathtt{q}[\mathtt{B}] : k}{\Gamma \vdash \mathtt{A} \to \mathtt{q}[\mathtt{B}] : k\langle v\rangle \triangleright k[\mathtt{B}] : ?\mathtt{A}\langle S\rangle} \lfloor^{\mathrm{LT}}|_{\textsc{Com-R}}\rceil$$

$$\frac{\Gamma \vdash \mathtt{p}[\mathtt{A}] \to \mathtt{q}[\mathtt{B}] : k}{\Gamma \vdash \mathtt{p}[\mathtt{A}] \to \mathtt{q}[\mathtt{B}] : k[l] \triangleright k : \mathtt{A} \to \mathtt{B} : [l]} \lfloor^{\mathrm{LT}}|_{\textsc{Sel}}\rceil$$

$$\frac{\Gamma \vdash \mathtt{p}[\mathtt{A}] \to \mathtt{B} : k}{\Gamma \vdash \mathtt{p}[\mathtt{A}] \to \mathtt{B} : k \oplus l \triangleright k[\mathtt{A}] : \oplus\mathtt{B}\langle l\rangle} \lfloor^{\mathrm{LT}}|_{\textsc{Sel-S}}\rceil \qquad \frac{\Gamma \vdash \mathtt{A} \to \mathtt{q}[\mathtt{B}] : k}{\Gamma \vdash \mathtt{A} \to \mathtt{q}[\mathtt{B}] : k\&l \triangleright k[\mathtt{B}] : \&\mathtt{A}\langle l\rangle} \lfloor^{\mathrm{LT}}|_{\textsc{Bra}}\rceil$$

$$\frac{\Gamma \vdash \mathtt{p}[\mathtt{A}] \to \mathtt{q}[\mathtt{B}] : k \quad \Gamma \vdash \mathtt{p} : k'[\mathtt{C}]}{\Gamma \vdash \mathtt{p}[\mathtt{A}] \to \mathtt{q}[\mathtt{B}] : k\langle k'[\mathtt{C}]\rangle \triangleright k : \mathtt{A} \to \mathtt{B} : \langle T\rangle} \lfloor^{\mathrm{LT}}|_{\textsc{Del}}\rceil$$

$$\frac{\Gamma \vdash \mathtt{p}[\mathtt{A}] \to \mathtt{B} : k}{\Gamma \vdash \mathtt{p}[\mathtt{A}] \to \mathtt{B} : k\langle k'[\mathtt{C}]\rangle \triangleright k[\mathtt{A}] : !\mathtt{B}\langle T\rangle} \lfloor^{\mathrm{LT}}|_{\textsc{Del-S}}\rceil \qquad \frac{\Gamma \vdash \mathtt{A} \to \mathtt{q}[\mathtt{B}] : k}{\Gamma \vdash \mathtt{A} \to \mathtt{q}[\mathtt{B}] : k(k'[\mathtt{C}]) \triangleright k[\mathtt{B}] : ?\mathtt{A}\langle T\rangle} \lfloor^{\mathrm{LT}}|_{\textsc{Del-R}}\rceil$$

$$\frac{\Gamma \vdash \lambda \triangleright k : \alpha}{\Gamma \vdash (\boldsymbol{\nu} r)\,\lambda \triangleright k : \alpha} \lfloor^{\mathrm{LT}}|_{\textsc{Res}}\rceil \qquad \text{(no rule for } \tau@\mathtt{p}\text{)}$$

Figure B.3: Compositional Choreographies, label typing.

B.2.2 Proof of Theorem 3.4.1

We report the full definition of the judgement $\Gamma \vdash \lambda \triangleright k : \alpha$ in Figure B.3.

We now show some auxiliary results that we will use in our proof of Theorem 3.4.1.

Lemma B.1 (Type Projection is invariant under $\simeq_{\mathcal{G}}$)**.** *Let G and G' be global types. Then, $G \simeq_{\mathcal{G}} G'$ implies $[\![G]\!]_{\mathtt{A}} = [\![G']\!]_{\mathtt{A}}$ for every role $\mathtt{A}$.*

Proof. Immediate, from the definitions of $\simeq_{\mathcal{G}}$ and type projection, since $\simeq_{\mathcal{G}}$ allows to swap only terms that have different roles. □

Lemma B.2 (Substitution)**.** *Assume $\Gamma \vdash C \triangleright \Delta$. Then, $\Gamma \vdash x@\mathtt{p} : S, v : S$ implies $\Gamma \vdash C[v/x@\mathtt{p}] \triangleright \Delta$.*

Proof. By induction on the typing rules. □

Lemma B.3 (Subject Congruence)**.** *$\Gamma \vdash C \triangleright \Delta$ and $C \equiv C'$ imply $\Gamma \vdash C' \triangleright \Delta$ (up to α-renaming).*

Proof. By induction on the rules that define $\equiv$. □

Lemma B.4 (Subject Swap)**.** *Let $\Gamma \vdash C \triangleright \Delta$; then, $C \simeq_{\mathcal{C}} C'$ implies $\Gamma \vdash C' \triangleright \Delta$.*

Proof. Easy induction on the derivation of $C \simeq_{\mathcal{C}} C'$. □

We can now prove typing soundness, by establishing the stronger results below. In the following, $\mathsf{upd}(\Gamma, \lambda) = \Gamma'$ is defined only if $\lambda \in \{\textit{(del)}, \textit{(del-s)}, \textit{(del-r)}\}$ and is defined as:

$$\mathsf{upd}(\Gamma, \lambda) = \begin{cases} \Gamma[\mathtt{q} \mapsto k'[\mathtt{C}]] & \text{if } \lambda = \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{q}[\mathtt{B}] : k\langle k'[\mathtt{C}]\rangle \\ \Gamma \setminus \mathtt{p} : k'[\mathtt{C}] & \text{if } \lambda = \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{B} : k\langle k'[\mathtt{C}]\rangle \\ \Gamma, \mathtt{q} : k'[\mathtt{C}] & \text{if } \lambda = \mathtt{A} \texttt{ -> } \mathtt{q}[\mathtt{B}] : k(k'[\mathtt{C}]) \end{cases}$$

Theorem B.1 (Typing Soundness)**.** *Let* $\Gamma \vdash C \triangleright \Delta$*; then,* $C \xrightarrow{\lambda} C'$ *implies* $\Gamma' \vdash C' \triangleright \Delta'$ *for some* Γ' *and* Δ' *such that:*

(i) if $\lambda \in \{\tau@\mathtt{p}, \textit{(start)}, \textit{(req)}, \textit{(acc)}\}$*, then* $\Gamma' = \Gamma$ *and* $\Delta' = \Delta$*;*

(ii) if $\lambda \in \{\textit{(del)}, \textit{(del-s)}, \textit{(del-r)}\}$*, then* $\Gamma' = \mathsf{upd}(\Gamma, \lambda)$ *and* $\Delta \xrightarrow{k:\alpha} \Delta'$ *such that* $\Gamma \vdash \lambda \triangleright k : \alpha$*;*

(iii) otherwise, $\Gamma' = \Gamma$ *and* $\Delta \xrightarrow{k:\alpha} \Delta'$ *such that* $\Gamma \vdash \lambda \triangleright k : \alpha$*.*

Proof. The proof is by induction on the derivation of $C \xrightarrow{\lambda} C'$. It is almost the same as the proof of Theorem 2.4.2 given in Appendix A.1. The interesting differences are: (i) the interaction of partial terms, e.g., by rule $\lfloor^{\mathrm{C}}|_{\textsc{Sync}}\rceil$; and (ii) the usage of local types instead of global types for typing interactions.

- **Case** $\lfloor^{\mathrm{C}}|_{\textsc{Act}}\rceil$. The case is:

$$\frac{\eta \notin \{\textit{(com)}, \textit{(com-s)}, \textit{(com-r)}, \textit{(start)}, \textit{(acc)}\}}{C = \eta; C' \xrightarrow{\eta} C'} \lfloor^{\mathrm{C}}|_{\textsc{Act}}\rceil$$

  We proceed by case analysis on η.

  - **Case** $\eta = \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{q}[\mathtt{B}] : k[l_j]$. Since C is well-typed, we know that:

$$\frac{\Gamma \vdash \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{q}[\mathtt{B}] : k \quad \Gamma \vdash C' \triangleright \Delta, k[\mathtt{A}] : T_j, k[\mathtt{B}] : T'_j \quad j \in I}{\Gamma \vdash \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{q}[\mathtt{B}] : k[l_j]; C' \triangleright \Delta, k[\mathtt{A}] : \oplus\mathtt{B}\{l_i : T_i\}_{i\in I}, k[\mathtt{B}] : \&\mathtt{A}\{l_i : T'_i\}_{i\in I}} \lfloor^{\mathrm{T}}|_{\textsc{Sel}}\rceil$$

    We can easily prove the thesis with the following two derivations:

$$\dfrac{\dfrac{\dfrac{j \in I}{\oplus\mathtt{B}\{l_i : T_i\}_{i\in I} \xrightarrow{\alpha_1} T_j} \lfloor^{\mathrm{L}}|_{\textsc{Sel}}\rceil}{k[\mathtt{A}] : \oplus\mathtt{B}\{l_i : T_i\}_{i\in I} \xrightarrow{k:\alpha_1} k[\mathtt{A}] : T_j} \lfloor^{\mathrm{L}}|_{\textsc{Lift}}\rceil \quad \dfrac{\dfrac{j \in I}{\&\mathtt{A}\{l_i : T'_i\}_{i\in I} \xrightarrow{\alpha_2} T'_j} \lfloor^{\mathrm{L}}|_{\textsc{Bra}}\rceil}{k[\mathtt{B}] : \&\mathtt{A}\{l_i : T'_i\}_{i\in I} \xrightarrow{k:\alpha_2} k[\mathtt{B}] : T'_j} \lfloor^{\mathrm{L}}|_{\textsc{Lift}}\rceil}{k[\mathtt{A}] : \oplus\mathtt{B}\{l_i : T_i\}_{i\in I}, k[\mathtt{B}] : \&\mathtt{A}\{l_i : T'_i\}_{i\in I} \xrightarrow{\alpha} k[\mathtt{A}] : T_j, k[\mathtt{B}] : T'_j} \lfloor^{\mathrm{L}}|_{\textsc{Sync}}\rceil}{\Delta, k[\mathtt{A}] : \oplus\mathtt{B}\{l_i : T_i\}_{i\in I}, k[\mathtt{B}] : \&\mathtt{A}\{l_i : T'_i\}_{i\in I} \xrightarrow{\alpha} \Delta, k[\mathtt{A}] : T_j, k[\mathtt{B}] : T'_j} \lfloor^{\mathrm{L}}|_{\textsc{Conc}}\rceil$$

    where $\alpha_1 = \oplus\mathtt{A}\langle l_j\rangle$, $\alpha_2 = \&\mathtt{A}\langle l_j\rangle$ and $\alpha = \alpha_1 \circ \alpha_2 = k : \mathtt{A} \texttt{ -> } \mathtt{B} : k[l_j]$; and, finally:

$$\frac{\Gamma \vdash \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{q}[\mathtt{B}] : k}{\Gamma \vdash \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{q}[\mathtt{B}] : k[l_j] \triangleright k : \mathtt{A} \texttt{ -> } \mathtt{B} : [l_j]} \lfloor^{\mathrm{LT}}|_{\textsc{Sel}}\rceil$$

– **Case** $\eta = \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{B} : k \oplus l_j$. Since C is well-typed, we know that:

$$\frac{\Gamma \vdash \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{B} : k \quad j \in I \quad \Gamma \vdash \boxed{C'} \triangleright \Delta, k[\mathtt{A}]{:}T_j \quad \mathtt{q}{:}k[\mathtt{B}] \notin \Gamma}{\Gamma \vdash \boxed{\mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{B} : k \oplus l_j; C'} \triangleright \Delta, k[\mathtt{A}]{:} \oplus \mathtt{B}\{l_i : T_i\}_{i \in I}} \lfloor^{\mathrm{T}}|_{\textsc{Sel-S}}\rceil$$

We can easily prove the thesis with the following two derivations:

$$\frac{\dfrac{j \in I}{\oplus \mathtt{B}\{l_i : T_i\}_{i \in I} \xrightarrow{\alpha_1} T_j}\ \lfloor^{\mathrm{L}}|_{\textsc{Sel}}\rceil}{k[\mathtt{A}]{:} \oplus \mathtt{B}\{l_i : T_i\}_{i \in I} \xrightarrow{k:\alpha_1} k[\mathtt{A}]{:}T_j}\ \lfloor^{\mathrm{L}}|_{\textsc{Lift}}\rceil$$

where $\alpha_1 = \oplus \mathtt{A}\langle l_j \rangle$; and, finally:

$$\frac{\Gamma \vdash \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{q}[\mathtt{B}] : k}{\Gamma \vdash \boxed{\mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{B} : k \oplus l_j} \triangleright k[\mathtt{A}]{:} \oplus \mathtt{B}\langle l_j \rangle}\ \lfloor^{\mathrm{LT}}|_{\textsc{Sel-S}}\rceil$$

– **Case** $\eta \in \{\textit{(req)}, \textit{(del)}, \textit{(del-s)}, \textit{(del-r)}\}$. These cases are similar to the two above.

- **Case** $\lfloor^{\mathrm{C}}|_{\textsc{Start}}\rceil$. The case is:

$$\frac{\eta = \mathtt{p}[\mathtt{A}]\ \mathbf{start}\ \widetilde{\mathtt{q}[\mathtt{B}]} : a(k)}{C = \ \eta; C_1 \xrightarrow{\eta} (\boldsymbol{\nu} k, \tilde{\mathtt{q}})\ C_1 \ = C'}\ \lfloor^{\mathrm{C}}|_{\textsc{Start}}\rceil$$

Since C is well-typed we know that, for $\Gamma = \Gamma'', a{:}G\langle \mathtt{A} | \tilde{\mathtt{B}} | \tilde{\mathtt{B}} \rangle$:

$$\frac{\Gamma'', a{:}G\langle \mathtt{A} | \tilde{\mathtt{B}} | \tilde{\mathtt{B}} \rangle, \Gamma' \vdash \boxed{C_1} \triangleright \Delta, \Delta' \quad (\Gamma', \Delta') = \mathsf{init}(\mathtt{p}[\mathtt{A}], \widetilde{\mathtt{q}[\mathtt{B}]}, k, G) \quad \tilde{\mathtt{q}} \notin \Gamma''}{\Gamma'', a{:}G\langle \mathtt{A} | \tilde{\mathtt{B}} | \tilde{\mathtt{B}} \rangle \vdash \boxed{\mathtt{p}[\mathtt{A}]\ \mathbf{start}\ \widetilde{\mathtt{q}[\mathtt{B}]} : a(k); C_1} \triangleright \Delta}\ \lfloor^{\mathrm{T}}|_{\textsc{Start}}\rceil$$

Let $\tilde{\mathtt{q}} = \mathtt{q}_1, \ldots, \mathtt{q}_n$. Since $\tilde{\mathtt{q}} \notin \Gamma''$, we can easily obtain the thesis by applying $\lfloor^{\mathrm{T}}|_{\textsc{ResProc}}\rceil$ n times. Formally:

$$\frac{\begin{array}{c}\Gamma, \Gamma' \vdash \boxed{C_1} \triangleright \Delta, \Delta' \\ \vdots\ \lfloor^{\mathrm{T}}|_{\textsc{ResProc}}\rceil\ n \text{ times} \\ \Gamma \vdash \boxed{(\boldsymbol{\nu} \tilde{\mathtt{q}})\ C_1} \triangleright \Delta, \Delta'\end{array}}{\Gamma \vdash \boxed{(\boldsymbol{\nu} k, \tilde{\mathtt{q}})\ C_1} \triangleright \Delta}\ \lfloor^{\mathrm{T}}|_{\textsc{ResLin}}\rceil$$

- **Cases** $\lfloor^{\mathrm{C}}|_{\textsc{Com}}\rceil$, $\lfloor^{\mathrm{C}}|_{\textsc{Com-S}}\rceil$, $\lfloor^{\mathrm{C}}|_{\textsc{Com-R}}\rceil$, $\lfloor^{\mathrm{C}}|_{\textsc{Branch}}\rceil$. These cases are similar to that of $\lfloor^{\mathrm{C}}|_{\textsc{Act}}\rceil$.

- **Case** $\lfloor^{\mathrm{C}}|_{\textsc{Sync}}\rceil$. The case is:

$$\frac{C_1 \xrightarrow{\lambda_1} C_1' \quad C_2 \xrightarrow{\lambda_2} C_2'}{C = \ C_1 \mid C_2 \ \xrightarrow{\lambda_1 \circ \lambda_2}\ C_1' \mid C_2' \ = C'}\ \lfloor^{\mathrm{C}}|_{\textsc{Sync}}\rceil$$

Since C is well-typed we know that:

$$\frac{\Gamma_i \vdash \boxed{C_i} \triangleright \Delta_i \quad \mathsf{pco}(\Delta_1, \Delta_2)}{\Gamma_1 \circ \Gamma_2 \vdash \boxed{C_1 \mid C_2} \triangleright \Delta_1, \Delta_2}\ \lfloor^{\mathrm{T}}|_{\textsc{Par}}\rceil$$

where $\Gamma = \Gamma_1 \circ \Gamma_2$ and $\Delta = \Delta_1 \circ \Delta_2$. Hence, we can apply the induction hypothesis and obtain:

$$\Delta_1 \xrightarrow{\alpha_1} \Delta_1' \quad \text{such that} \quad \Gamma_1 \vdash \lambda_1 \triangleright \alpha_1$$

$$\Delta_2 \xrightarrow{\alpha_2} \Delta_2' \quad \text{such that} \quad \Gamma_2 \vdash \lambda_2 \triangleright \alpha_2$$

The proof proceeds now by cases on λ_1 and λ_2. Since we know that $\lambda_1 \circ \lambda_2$ is defined (the two labels are one the co-action of the other), the cases are all similar. We report the case for $\lambda_1 = \mathtt{p}[\mathtt{A}]\ \mathtt{->}\ \mathtt{B} : k\langle v \rangle$ and $\lambda_2 = \mathtt{A}\ \mathtt{->}\ \mathtt{q}[\mathtt{B}] : k\langle v \rangle$; we know that the roles specified in the labels are compatible because $\lambda = \lambda_1 \circ \lambda_2$ is defined. Since $\Gamma_1 \vdash \lambda_1 \triangleright \alpha_1$ we know that:

$$\frac{\Gamma \vdash \mathtt{p}[\mathtt{A}]\ \mathtt{->}\ \mathtt{B} : k \quad \vdash v : S_1}{\Gamma \vdash \lambda_1 \triangleright k[\mathtt{A}] : !\mathtt{B}\langle S_1 \rangle}\ \lfloor^{\text{LT}}|_{\text{COM-S}}\rceil$$

Analogously, from $\Gamma_2 \vdash \lambda_2 \triangleright \alpha_2$ we know that:

$$\frac{\Gamma \vdash \mathtt{p}[\mathtt{A}]\ \mathtt{->}\ \mathtt{B} : k \quad \vdash v : S_2}{\Gamma \vdash \lambda_2 \triangleright k[\mathtt{B}] : ?\mathtt{A}\langle S_2 \rangle}\ \lfloor^{\text{LT}}|_{\text{COM-R}}\rceil$$

From $\mathsf{pco}(\Delta_1, \Delta_2)$, we know that $S_1 = S_2 = S$. Therefore, we can choose $\alpha = \alpha_1 \circ \alpha_2$, $\Delta = \Delta_1, \Delta_2$, and prove the thesis with the following derivation:

$$\frac{\Delta_1 \xrightarrow{\alpha_1} \Delta_1' \quad \Delta_2 \xrightarrow{\alpha_2} \Delta_2'}{\Delta_1, \Delta_2 \xrightarrow{\alpha_1 \circ \alpha_2} \Delta_1', \Delta_2'}\ \lfloor^{\text{L}}|_{\text{SYNC}}\rceil$$

- **Case** $\lfloor^{\text{C}}|_{\text{COND}}\rceil$. The case is:

$$\frac{i = 1 \text{ if } e \downarrow \text{true}, i = 2 \text{ otherwise}}{C = \mathsf{if}\ e@\mathsf{p}\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2 \xrightarrow{\tau@\mathsf{p}} C_i}\ \lfloor^{\text{C}}|_{\text{COND}}\rceil$$

  Since C is well-typed, we know that:

$$\frac{\Gamma \vdash e@\mathsf{p} : \mathbf{bool} \quad \Gamma \vdash C_i \triangleright \Delta}{\Gamma \vdash \mathsf{if}\ e@\mathsf{p}\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2 \triangleright \Delta}\ \lfloor^{\text{T}}|_{\text{COND}}\rceil$$

  The thesis follows immediately from the premises of the typing derivation above, for both the cases $C' = C_1$ and $C' = C_2$.

- **Cases** $\lfloor^{\text{C}}|_{\text{RES}}\rceil$, $\lfloor^{\text{C}}|_{\text{CTX}}\rceil$, $\lfloor^{\text{C}}|_{\text{PAR}}\rceil$. These cases follow easily by induction hypothesis.

- **Case** $\lfloor^{\text{C}}|_{\text{EQ}}\rceil$.

$$\frac{\mathcal{R} \in \{\simeq_{\mathcal{C}}, \equiv\} \quad C_1\,\mathcal{R}\,C_1' \quad C_1' \xrightarrow{\lambda} C_2' \quad C_2'\,\mathcal{R}\,C_2}{C_1 \xrightarrow{\lambda} C_2}\ \lfloor^{\text{C}}|_{\text{EQ}}\rceil$$

  We have two subcases. For $\mathcal{R} = \simeq_{\mathcal{C}}$, we conclude by Lemma B.4 and the induction hypothesis. Otherwise, for $\mathcal{R} = \equiv$, we conclude by Lemma B.3 and the induction hypothesis.

- **Case** $\lfloor^{\text{C}}|_{\text{P-START}}\rceil$. This case is similar to case $\lfloor^{\text{C}}|_{\text{SYNC}}\rceil$.

- **Case** $\lfloor^{\text{C}}|_{\text{ASYNC}}\rceil$. This case is similar to case $\lfloor^{\text{C}}|_{\text{ASYNC}}\rceil$ in the proof of Theorem A.1.1.

□

B.3 Endpoint Projection

B.3.1 Process Projection

We report the complete definition of process projection in Figure B.4. It follows the same intuition described in § 3.5.1. Partial terms are projected as they are for their respective processes. Observe that procedures get renamed by suffixing them with the name of the process that we are projecting: we will make use of this aspect later for proving that process projections are still well-typed.

B.3.2 Proof of Theorem 3.5.2

Minimal Typing. We start by defining the minimal typing system $\vdash_{\mathsf{min}}$, by proceeding as in § 2.4.5 for the Choreography Calculus.

First, we define subtyping relations for local and global types by set inclusion of branching labels. As in [31], we need to distinguish between the subtyping of branching types when we are typing complete or partial terms (in [31], this corresponds to typing choreographies or endpoint terms). To this end, we introduce the annotation $\bullet$ for branching local types, i.e., we write $\&_{\bullet}\mathtt{A}\{\widetilde{l : T}\}$ when typing a branching implemented by complete terms.

Definition B.3.1 (Subtyping)**.** The subtyping $T \ll T'$ is the smallest relation over closed and unfolded local types satisfying the rules reported in Figure B.5. The subtyping $G \ll G'$ is the smallest relation over closed and unfolded local types satisfying the rules reported in Figure B.6. We extend both subtyping relations to set inclusion and point-wise to the typing of shared names and sessions, respectively. Given two types G and G', we denote their least upper bound (lub) wrt $\ll$ with $G \triangledown G'$ (the same for local types).

Our definition of subtyping for local types follows [31], whereas that for global types is similar to that presented in § 2.4.5.

Building on subtyping, we define the minimal typing system $\vdash_{\mathsf{min}}$. The rules for typing complete terms are reported in Figure B.7, and the rules for typing partial terms are given in Figure B.8.

Remark B.3.2 (Comparison with minimal typing for the Choreography Calculus). Here, we abuse the term "minimal typing" to refer to using the minimal global and local types for typing sessions and shared names such that the projection of a choreography is still typable. In 2.4.5 we had a stronger objective, i.e., building the minimal typing environments for typing a choreography. Since we do not need such property for our presentation here, the rules do not require so many modifications and remain more similar to the runtime typing rules presented before for Compositional Choreographies. We refer the reader to [75] for a discussion on the type inference of global types from local types (which is not our aim here). □

Typing Projection. Since EPP projects recursive definitions at each different process from its own point of view, such projected procedures will have different types. To deal with this aspect, we define the projection of unrestricted environments.

$$
\begin{array}{rcl}
[\![\mathsf{p}[\mathtt{A}] \ \mathbf{start}\ \widetilde{\mathsf{q}[\mathtt{B}]} : a(k); C]\!]_{\mathsf{r}} & = & \begin{cases} \mathsf{p}[\mathtt{A}] \ \mathbf{req}\ \tilde{\mathtt{B}} : a(k); [\![C]\!]_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{p} \\ \mathbf{acc}\ \mathsf{r}[\mathtt{C}] : a(k); [\![C]\!]_{\mathsf{r}} & \text{if } \mathsf{r}[\mathtt{C}] \in \widetilde{\mathsf{q}[\mathtt{B}]} \\ [\![C]\!]_{\mathsf{r}} & \text{otherwise} \end{cases} \\
[\![\mathbf{acc}\ \widetilde{\mathsf{q}[\mathtt{A}]} : a(k); C]\!]_{\mathsf{r}} & = & \begin{cases} \mathbf{acc}\ \mathsf{r}[\mathtt{C}] : a(k); [\![C]\!]_{\mathsf{r}} & \text{if } \mathsf{r}[\mathtt{C}] \in \widetilde{\mathsf{q}[\mathtt{A}]} \\ [\![C]\!]_{\mathsf{r}} & \text{otherwise} \end{cases} \\
[\![\mathsf{p}[\mathtt{A}].e \texttt{ -> } \mathsf{q}[\mathtt{B}].x : k; C]\!]_{\mathsf{r}} & = & \begin{cases} \mathsf{p}[\mathtt{A}].e \texttt{ -> } \mathtt{B} : k; [\![C]\!]_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{p} \\ \mathtt{A} \texttt{ -> } \mathsf{q}[\mathtt{B}].x : k; [\![C]\!]_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{q} \\ [\![C]\!]_{\mathsf{r}} & \text{otherwise} \end{cases} \\
[\![\mathsf{p}[\mathtt{A}] \texttt{ -> } \mathsf{q}[\mathtt{B}] : k[l]; C]\!]_{\mathsf{r}} & = & \begin{cases} \mathsf{p}[\mathtt{A}] \texttt{ -> } \mathtt{B} : k \oplus l; [\![C]\!]_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{p} \\ \mathtt{A} \texttt{ -> } \mathsf{q}[\mathtt{B}] : k\&\{l : [\![C]\!]_{\mathsf{r}}\} & \text{if } \mathsf{r} = \mathsf{q} \\ [\![C]\!]_{\mathsf{r}} & \text{otherwise} \end{cases} \\
[\![\mathsf{p}[\mathtt{A}] \texttt{ -> } \mathsf{q}[\mathtt{B}] : k\langle k'[\mathtt{C}]\rangle; C]\!]_{\mathsf{r}} & = & \begin{cases} \mathsf{p}[\mathtt{A}] \texttt{ -> } \mathtt{B} : k\langle k'[\mathtt{C}]\rangle; [\![C]\!]_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{p} \\ \mathtt{A} \texttt{ -> } \mathsf{q}[\mathtt{B}] : k(k'[\mathtt{C}]); [\![C]\!]_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{q} \\ [\![C]\!]_{\mathsf{r}} & \text{otherwise} \end{cases} \\
[\![\eta; C]\!]_{\mathsf{r}} & = & \begin{cases} \eta; [\![C]\!]_{\mathsf{r}} & \text{if } \eta \in \left\{ \begin{array}{l} \textit{(req)}, \textit{(com-s)}, \textit{(com-r)}, \\ \textit{(del-s)}, \textit{(del-r)}, \textit{(sel-s)} \end{array} \right\} \\ & \text{and } \{\mathsf{r}\} = \mathsf{pn}(\eta) \\ [\![C]\!]_{\mathsf{r}} & \text{otherwise} \end{cases} \\
[\![\mathsf{if}\ e@\mathsf{p}\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2]\!]_{\mathsf{r}} & = & \begin{cases} \mathsf{if}\ e@\mathsf{p}\ \mathsf{then}\ [\![C_1]\!]_{\mathsf{r}}\ \mathsf{else}\ [\![C_2]\!]_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{p} \\ [\![C_1]\!]_{\mathsf{r}} \sqcup [\![C_2]\!]_{\mathsf{r}} & \text{otherwise} \end{cases} \\
[\![\mathtt{A} \texttt{ -> } \mathsf{q}[\mathtt{B}] : k\&\{l_i : C_i\}_{i \in I}]\!]_{\mathsf{r}} & = & \begin{cases} \mathtt{A} \texttt{ -> } \mathsf{q}[\mathtt{B}] : k\&\{l_i : [\![C_i]\!]_{\mathsf{r}}\}_{i \in I} & \text{if } \mathsf{r} = \mathsf{q} \\ \bigsqcup_{i \in I} [\![C_i]\!]_{\mathsf{r}} & \text{otherwise} \end{cases} \\
[\![\mathsf{def}\ X(\tilde{D}) = C'\ \mathsf{in}\ C]\!]_{\mathsf{r}} & = & \begin{cases} \mathsf{def}\ X_{\mathsf{r}}(D_i) = [\![C']\!]_{\mathsf{r}}\ \mathsf{in}\ [\![C]\!]_{\mathsf{r}} \\ \qquad \text{if } \mathsf{r} = \mathsf{p}_i \text{ and } 1 \leq i \leq n \\ [\![C]\!]_{\mathsf{r}} \quad \text{otherwise} \end{cases} \\
(\tilde{D} = \mathsf{p}_1(\tilde{x_1}, \tilde{k_1}), \ldots, \mathsf{p}_n(\tilde{x_n}, \tilde{k_n})) & & \\
[\![X\langle \tilde{E}\rangle]\!]_{\mathsf{r}} & = & \begin{cases} X_{\mathsf{r}}\langle E_i\rangle & \text{if } \mathsf{r} = \mathsf{p}_i \text{ and } 1 \leq i \leq n \\ \mathbf{0} & \text{otherwise} \end{cases} \\
(\tilde{E} = \mathsf{p}_1(\tilde{e_1}, \tilde{k_1}), \ldots, \mathsf{p}_n(\tilde{e_n}, \tilde{k_n})) & & \\
[\![C_1 \mid C_2]\!]_{\mathsf{r}} & = & [\![C_1]\!]_{\mathsf{r}} \mid [\![C_2]\!]_{\mathsf{r}} \\
[\![\mathbf{0}]\!]_{\mathsf{r}} & = & \mathbf{0} \\
(\text{no rule for } (\boldsymbol{\nu} r)\ C) & &
\end{array}
$$

Figure B.4: Compositional Choreographies, process projection.

$$\frac{}{S \ll S} \lfloor \textsc{SubL} | \textsc{Val} \rceil$$

$$\frac{T \ll T' \quad U \ll U'}{!\mathtt{A}\langle U\rangle;T \ll !\mathtt{A}\langle U'\rangle;T'} \lfloor \textsc{SubL} | \textsc{Send} \rceil \qquad \frac{T \ll T' \quad U' \ll U}{?\mathtt{A}\langle U\rangle;T \ll ?\mathtt{A}\langle U'\rangle;T'} \lfloor \textsc{SubL} | \textsc{Recv} \rceil$$

$$\frac{I \subseteq J \quad \forall i \in I.\, T_i \ll T'_i}{\&\mathtt{A}\{l_i : T_i\}_{i\in I} \ll \&\mathtt{A}\{l_i : T'_i\}_{i\in J}} \lfloor \textsc{SubL} | \textsc{Branch} \rceil$$

$$\frac{J \subseteq I \quad \forall i \in J.\, T_i \ll T'_i}{\&_\bullet\mathtt{A}\{l_i : T_i\}_{i\in I} \ll \&_\bullet\mathtt{A}\{l_i : T'_i\}_{i\in J}} \lfloor \textsc{SubL} | \textsc{GBranch} \rceil$$

$$\frac{J \subseteq I \quad \forall i \in J.\, T_i \ll T'_i}{\oplus\mathtt{A}\{l_i : T_i\}_{i\in I} \ll \oplus\mathtt{A}\{l_i : T'_i\}_{i\in J}} \lfloor \textsc{SubL} | \textsc{Sel} \rceil$$

$$\frac{(T \approx T' \vee T \simeq_{\mathcal{G}} T') \quad T \ll T''}{T' \ll T''} \lfloor \textsc{SubL} | \textsc{Eq} \rceil \qquad \frac{\mathsf{end} \approx T}{\mathsf{end} \ll T} \lfloor \textsc{SubL} | \textsc{End} \rceil$$

Figure B.5: Local Types, subtyping.

$$\frac{I \subseteq J \quad \forall i \in I.\, G_i \ll G'_i}{\mathtt{A} \to \mathtt{B} : \{l_i : G_i\}_{i\in I} \ll \mathtt{A} \to \mathtt{B} : \{l_i : G'_i\}_{i\in J}} \lfloor \textsc{SubG} | \textsc{Branch} \rceil$$

$$\frac{U \ll U' \qquad G_2 \ll G'_2}{\mathtt{A} \to \mathtt{B} : \langle U\rangle; G_2 \ll \mathtt{A} \to \mathtt{B} : \langle U'\rangle; G'_2} \lfloor \textsc{SubG} | \textsc{Com} \rceil$$

$$\frac{(G \approx G' \vee G \simeq_{\mathcal{G}} G') \quad G \ll G''}{G' \ll G''} \lfloor \textsc{SubG} | \textsc{Eq} \rceil \qquad \frac{\mathsf{end} \approx G}{\mathsf{end} \ll G} \lfloor \textsc{SubG} | \textsc{End} \rceil$$

Figure B.6: Global Types, subtyping.

$$\frac{\Gamma, \Gamma' \vdash_{\mathsf{min}} C \triangleright \Delta, \Delta' \quad (\Gamma', \Delta') = \mathsf{init}(\mathtt{p}[\mathtt{A}], \widetilde{\mathtt{q}[\mathtt{B}]}, k, G) \quad \tilde{\mathtt{q}} \notin \Gamma}{\Gamma, a : G\langle \mathtt{A} | \tilde{\mathtt{B}} | \tilde{\mathtt{B}} \rangle \vdash_{\mathsf{min}} \mathtt{p}[\mathtt{A}] \ \mathbf{start} \ \widetilde{\mathtt{q}[\mathtt{B}]} : a(k); C \triangleright \Delta} \ \lfloor^{\text{MIN}}|_{\text{START1}}\rceil$$

$$\frac{\Gamma, a : G\langle \mathtt{A} | \tilde{\mathtt{B}} | \tilde{\mathtt{B}} \rangle, \Gamma' \vdash_{\mathsf{min}} C \triangleright \Delta, \Delta' \quad (\Gamma', \Delta') = \mathsf{init}(\mathtt{p}[\mathtt{A}], \widetilde{\mathtt{q}[\mathtt{B}]}, k, G') \quad \tilde{\mathtt{q}} \notin \Gamma}{\Gamma, a : (G \triangledown G')\langle \mathtt{A} | \tilde{\mathtt{B}} | \tilde{\mathtt{B}} \rangle \vdash_{\mathsf{min}} \mathtt{p}[\mathtt{A}] \ \mathbf{start} \ \widetilde{\mathtt{q}[\mathtt{B}]} : a(k); C \triangleright \Delta} \ \lfloor^{\text{MIN}}|_{\text{START2}}\rceil$$

$$\frac{\Gamma \vdash \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{q}[\mathtt{B}] : k \quad \Gamma \vdash e@\mathtt{p} : S \quad \Gamma, x@\mathtt{q} : S \vdash_{\mathsf{min}} C \triangleright \Delta, k[\mathtt{A}] : T, k[\mathtt{B}] : T'}{\Gamma \vdash_{\mathsf{min}} \mathtt{p}[\mathtt{A}].e \texttt{ -> } \mathtt{q}[\mathtt{B}].x : k; C \triangleright \Delta, k[\mathtt{A}] : !\mathtt{B}\langle S \rangle; T, k[\mathtt{B}] : ?\mathtt{A}\langle S \rangle; T'} \ \lfloor^{\text{MIN}}|_{\text{COM}}\rceil$$

$$\frac{\Gamma \vdash \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{q}[\mathtt{B}] : k \qquad \Gamma, \mathtt{q} : k'[\mathtt{C}] \vdash_{\mathsf{min}} C \triangleright \Delta, k[\mathtt{A}] : T, k[\mathtt{B}] : T', k'[\mathtt{C}] : T''}{\Gamma, \mathtt{p} : k'[\mathtt{C}] \vdash_{\mathsf{min}} \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{q}[\mathtt{B}] : k\langle k'[\mathtt{C}] \rangle; C \triangleright \Delta, k[\mathtt{A}] : !\mathtt{B}\langle T'' \rangle; T, k[\mathtt{B}] : ?\mathtt{A}\langle T'' \rangle; T', k'[\mathtt{C}] : T''} \ \lfloor^{\text{MIN}}|_{\text{DEL}}\rceil$$

$$\frac{\Gamma \vdash \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{q}[\mathtt{B}] : k \quad \Gamma \vdash_{\mathsf{min}} C \triangleright \Delta, k[\mathtt{A}] : T, k[\mathtt{B}] : T'}{\Gamma \vdash_{\mathsf{min}} \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{q}[\mathtt{B}] : k[l]; C \triangleright \Delta, k[\mathtt{A}] : \oplus \mathtt{B}\{l : T\}, k[\mathtt{B}] : \&_\bullet \mathtt{A}\{l : T'\}} \ \lfloor^{\text{MIN}}|_{\text{SEL}}\rceil$$

$$\frac{\Gamma = \Gamma_1 \triangledown \Gamma_2 \quad \Gamma \vdash e@\mathtt{p} : \mathbf{bool} \quad \Gamma_i \vdash_{\mathsf{min}} C_i \triangleright \Delta}{\Gamma \vdash_{\mathsf{min}} \mathsf{if}\ e@\mathtt{p}\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2 \triangleright \Delta_1 \triangledown \Delta_2} \ \lfloor^{\text{MIN}}|_{\text{COND}}\rceil \qquad \frac{\mathsf{cosha}(\Gamma) \quad \Delta \ \mathsf{end\ only}}{\Gamma \vdash_{\mathsf{min}} \mathbf{0} \triangleright \Delta} \ \lfloor^{\text{MIN}}|_{\text{END}}\rceil$$

$$\frac{\Gamma, a : G\langle \mathtt{A} | \tilde{\mathtt{B}} | \tilde{\mathtt{B}} \rangle \vdash_{\mathsf{min}} C \triangleright \Delta}{\Gamma \vdash_{\mathsf{min}} (\boldsymbol{\nu} a)\, C \triangleright \Delta} \ \lfloor^{\text{MIN}}|_{\text{RESSHA}}\rceil$$

$$\frac{\begin{array}{c}\mathsf{end}(\Delta) \quad \Gamma'' \vdash x_{ij}@\mathtt{p}_i : S_{ij} \quad \Gamma \vdash e_{ij}@\mathtt{p}_i : S_{ij} \quad \Gamma' \subseteq \Gamma \\ D = \mathtt{p}_1(\tilde{x_1}, \tilde{k_1}), \ldots, \mathtt{p}_n(\tilde{x_n}, \tilde{k_n}) \quad E = \mathtt{p}_1(\tilde{e_1}, \tilde{k_1}), \ldots, \mathtt{p}_n(\tilde{e_n}, \tilde{k_n})\end{array}}{\Gamma, X(\tilde{D}) : (\Gamma', \Gamma''; \Delta') \vdash_{\mathsf{min}} X\langle \tilde{E} \rangle \triangleright \Delta, \Delta'} \ \lfloor^{\text{MIN}}|_{\text{CALL}}\rceil$$

$$\frac{\begin{array}{c}\Gamma, X(\tilde{D}) : (\Gamma'; \Delta') \vdash_{\mathsf{min}} C \triangleright \Delta \\ \Gamma', X(\tilde{D}) : (\Gamma'; \Delta') \vdash_{\mathsf{min}} C' \triangleright \Delta' \quad \Gamma'|_{\mathsf{sha}} \subseteq \Gamma\end{array}}{\Gamma \triangledown \Gamma' \vdash_{\mathsf{min}} \mathsf{def}\ X(\tilde{D}) = C' \ \mathsf{in}\ C \triangleright \Delta} \ \lfloor^{\text{MIN}}|_{\text{DEF}}\rceil$$

$$\frac{\Gamma \vdash_{\mathsf{min}} C \triangleright \Delta \quad \mathsf{co}(\Delta, k)}{\Gamma \setminus k \vdash_{\mathsf{min}} (\boldsymbol{\nu} k)\, C \triangleright \Delta \setminus k} \ \lfloor^{\text{MIN}}|_{\text{RESLIN}}\rceil \qquad \frac{\Gamma \vdash_{\mathsf{min}} C \triangleright \Delta}{\Gamma \setminus \mathtt{p} \vdash_{\mathsf{min}} (\boldsymbol{\nu} \mathtt{p})\, C \triangleright \Delta} \ \lfloor^{\text{MIN}}|_{\text{RESPROC}}\rceil$$

Figure B.7: Compositional Choreographies, minimal typing rules for complete terms.

$$\frac{\Gamma \vdash a : G\langle \mathtt{A}|\tilde{\mathtt{B}}|\emptyset\rangle \qquad \Gamma, \mathsf{p} : k[\mathtt{A}] \ \vdash_{\mathsf{min}} \ C \ \triangleright \ \Delta, k[\mathtt{A}] : [\![G]\!]_{\mathtt{A}}}{\Gamma \ \vdash_{\mathsf{min}} \ \mathsf{p}[\mathtt{A}] \ \mathbf{req} \ \tilde{\mathtt{B}} : a(k); C \ \triangleright \ \Delta} \ \lfloor^{\text{MIN}}|_{\text{REQ1}}\rceil$$

$$\frac{\Gamma \vdash x@\mathsf{p} : G\langle \mathtt{A}|\tilde{\mathtt{B}}|\emptyset\rangle \qquad \Gamma, \mathsf{p} : k[\mathtt{A}] \ \vdash_{\mathsf{min}} \ C \ \triangleright \ \Delta, k[\mathtt{A}] : [\![G]\!]_{\mathtt{A}}}{\Gamma \ \vdash_{\mathsf{min}} \ \mathsf{p}[\mathtt{A}] \ \mathbf{req} \ \tilde{\mathtt{B}} : x(k); C \ \triangleright \ \Delta} \ \lfloor^{\text{MIN}}|_{\text{REQ2}}\rceil$$

$$\frac{\Gamma, a : G\langle \mathtt{D}|\tilde{\mathtt{B}}|\emptyset\rangle, \Gamma' \ \vdash_{\mathsf{min}} \ C \ \triangleright \ \Delta, \Delta'' \quad (\Gamma', \Delta') = \mathsf{init}(\widetilde{\mathsf{q}[\mathtt{A}]}, k, G) \quad \Delta'' \ll \Delta' \quad \tilde{\mathsf{q}} \notin \Gamma}{\Gamma, a : G\langle \mathtt{D}|\tilde{\mathtt{B}}|\tilde{\mathtt{A}}\rangle \ \vdash_{\mathsf{min}} \ \mathbf{acc} \ \widetilde{\mathsf{q}[\mathtt{A}]} : a(k); C \ \triangleright \ \Delta} \ \lfloor^{\text{MIN}}|_{\text{ACC}}\rceil$$

$$\frac{\Gamma \ \vdash \ e@\mathsf{p} : S \quad \Gamma \vdash \mathsf{p}[\mathtt{A}] \ \texttt{->} \ \mathtt{B} : k \quad \Gamma \ \vdash_{\mathsf{min}} \ C \ \triangleright \ \Delta, k[\mathtt{A}] : T \quad \mathsf{q} : k[\mathtt{B}] \notin \Gamma}{\Gamma \ \vdash_{\mathsf{min}} \ \mathsf{p}[\mathtt{A}].e \ \texttt{->} \ \mathtt{B} : k; C \ \triangleright \ \Delta, k[\mathtt{A}] : !\mathtt{B}\langle S\rangle; T} \ \lfloor^{\text{MIN}}|_{\text{COM-S}}\rceil$$

$$\frac{\Gamma \vdash \mathtt{A} \ \texttt{->} \ \mathsf{q}[\mathtt{B}] : k \quad \Gamma, x@\mathsf{p} : S \ \vdash_{\mathsf{min}} \ C \ \triangleright \ \Delta, k[\mathtt{B}] : T \quad \mathsf{p} : k[\mathtt{A}] \notin \Gamma}{\Gamma \ \vdash_{\mathsf{min}} \ \mathtt{A} \ \texttt{->} \ \mathsf{q}[\mathtt{B}].x : k; C \ \triangleright \ \Delta, k[\mathtt{A}] : ?\mathtt{B}\langle S\rangle; T} \ \lfloor^{\text{MIN}}|_{\text{COM-R}}\rceil$$

$$\frac{\Gamma \vdash \mathsf{p}[\mathtt{A}] \ \texttt{->} \ \mathtt{B} : k \quad \Gamma \ \vdash_{\mathsf{min}} \ C \ \triangleright \ \Delta, k[\mathtt{A}] : T \quad \mathsf{q} : k[\mathtt{B}] \notin \Gamma}{\Gamma, \mathsf{p} : k'[\mathtt{C}] \ \vdash_{\mathsf{min}} \ \mathsf{p}[\mathtt{A}] \ \texttt{->} \ \mathtt{B} : k\langle k'[\mathtt{C}]\rangle; C \ \triangleright \ \Delta, k[\mathtt{A}] : !\mathtt{B}\langle T''\rangle; T, k'[\mathtt{C}] : T''} \ \lfloor^{\text{MIN}}|_{\text{DEL-S}}\rceil$$

$$\frac{\begin{array}{c}\Gamma \vdash \mathtt{A} \ \texttt{->} \ \mathsf{q}[\mathtt{B}] : k \quad \mathsf{p} : k[\mathtt{A}] \notin \Gamma \\ \Gamma, \mathsf{q} : k'[\mathtt{C}] \ \vdash_{\mathsf{min}} \ C \ \triangleright \ \Delta, k[\mathtt{B}] : T', k'[\mathtt{C}] : T''\end{array}}{\Gamma \ \vdash_{\mathsf{min}} \ \mathtt{A} \ \texttt{->} \ \mathsf{q}[\mathtt{B}] : k(k'[\mathtt{C}]); C \ \triangleright \ \Delta, k[\mathtt{B}] : ?\mathtt{A}\langle T''\rangle; T'} \ \lfloor^{\text{MIN}}|_{\text{DEL-R}}\rceil$$

$$\frac{\Gamma \vdash \mathsf{p}[\mathtt{A}] \ \texttt{->} \ \mathtt{B} : k \quad j \in I \quad \Gamma \ \vdash_{\mathsf{min}} \ C \ \triangleright \ \Delta, k[\mathtt{A}] : T_j \quad \mathsf{q} : k[\mathtt{B}] \notin \Gamma}{\Gamma \ \vdash_{\mathsf{min}} \ \mathsf{p}[\mathtt{A}] \ \texttt{->} \ \mathtt{B} : k \oplus l_j; C \ \triangleright \ \Delta, k[\mathtt{A}] : \oplus\mathtt{B}\{l_i : T_i\}_{i\in I}} \ \lfloor^{\text{MIN}}|_{\text{SEL-S}}\rceil$$

$$\frac{\Gamma \vdash \mathtt{A} \ \texttt{->} \ \mathsf{q}[\mathtt{B}] : k \quad \Gamma \ \vdash_{\mathsf{min}} \ C_i \ \triangleright \ \Delta, k[\mathtt{A}] : T_i \quad \mathsf{p} : k[\mathtt{A}] \notin \Gamma}{\Gamma \ \vdash_{\mathsf{min}} \ \mathtt{A} \ \texttt{->} \ \mathsf{q}[\mathtt{B}] : k\&\{l_i : C_i\}_{i\in I} \ \triangleright \ \Delta, k[\mathtt{B}] : \&\mathtt{A}\{l_i : T_i\}_{i\in I}} \ \lfloor^{\text{MIN}}|_{\text{BRANCH}}\rceil$$

$$\frac{\Gamma_i \ \vdash_{\mathsf{min}} \ C_i \ \triangleright \ \Delta_i \quad \mathsf{pco}(\Delta_1, \Delta_2)}{\Gamma_1 \circ \Gamma_2 \ \vdash_{\mathsf{min}} \ C_1 \mid C_2 \ \triangleright \ \Delta_1, \Delta_2} \ \lfloor^{\text{MIN}}|_{\text{PAR}}\rceil$$

Figure B.8: Compositional Choreographies, minimal typing rules for partial terms.

Definition B.3.3 (Typing Projection)**.** Let $\Gamma = \Gamma'', \Gamma_{\mathsf{def}}$, where Γ'' does not contain definition typings $(X(\tilde{D}) : (\Gamma; \Delta))$ and Γ_{def} contains only definition typings; then, the projection of Γ, written $[\![\Gamma]\!]$ is defined as:

$$[\![\Gamma]\!] \;=\; \Gamma'', \{[\![\Gamma_{\mathsf{def}}]\!]_{\mathsf{p}} \mid \mathsf{p} \text{ s.t. } X(\tilde{D}) \in \Gamma_{\mathsf{def}} \text{ and } \mathsf{p} \in \tilde{D}\}$$

where

$$[\![\Gamma_{\mathsf{def}}]\!]_{\mathsf{p}} \;=\; \left\{ \begin{array}{l} X_{\mathsf{p}}(\mathsf{p}(\tilde{x}, \tilde{k})) : ([\![\Gamma']\!]_{\mathsf{p}}; \Delta'|_{\widetilde{k[\mathtt{A}]}}) \mid \\ \qquad\qquad X(\tilde{D}) : (\Gamma'; \Delta') \in \Gamma_{\mathsf{def}} \text{ and } \mathsf{p}(\tilde{x}, \tilde{k}) \in \tilde{D} \\ \qquad\qquad \text{and } \widetilde{k[\mathtt{A}]} = \{k[\mathtt{A}] | k \in \tilde{k} \text{ and } \mathsf{p} : k[\mathtt{A}] \in \Gamma'\} \end{array} \right\}$$

We introduce now some auxiliary lemmas, which will be useful in the proofs of our theorems.

Lemma C.1 (EPP Free Names)**.** *Let C be a choreography. Then,* $\mathsf{fn}(C) = \mathsf{fn}([\![C]\!])$.

Proof. Immediate, from the definition of EPP. □

Lemma C.2 (EPP Substitution Lemma)**.** $[\![C[v/x@\mathsf{p}]]\!]_{\mathsf{p}} = [\![C]\!]_{\mathsf{p}}[v/x@\mathsf{p}]$.

Proof. Immediate from the definition of process projection. □

Lemma C.3 (EPP Substitution Locality)**.** *Let $C \equiv (\boldsymbol{\nu} \tilde{a}, \tilde{k}, \tilde{\mathsf{p}})\, C_f$, where C_f is restriction-free, and $\mathsf{p} \in \mathsf{fn}(C)$. Then,*

$$[\![C[v/x@\mathsf{p}]]\!] = (\boldsymbol{\nu} \tilde{a}) \left((\boldsymbol{\nu} \tilde{k}, \tilde{\mathsf{p}}) \left([\![C_f[v/x@\mathsf{p}]]\!]_{\mathsf{p}} \mid \prod_{\mathsf{p} \in \mathsf{fn}(C_f)} [\![C_f]\!]_{\mathsf{p}} \right) \mid \prod_{a, \mathtt{A}} \Big(\bigsqcup_{\mathsf{p} \in \lfloor C_f \rfloor^a_{\mathtt{A}}} [\![C_f]\!]_{\mathsf{p}} \Big) \right)$$

Proof. Immediate from the definition of EPP and Lemma C.2. □

Lemma C.4 (EPP Swap Invariance)**.** *Let $C \simeq_{\mathcal{C}} C'$. Then,* $[\![C]\!] = [\![C']\!]$.

Sketch. The main part of the proof is to show that process projection is invariant under the rules for the swapping relation $\simeq_{\mathcal{C}}$, reported in Figure 3.4. $\lfloor^{\text{SW}}|_{\text{ETA-ETA}}\rceil$ is a trivial case. For $\lfloor^{\text{SW}}|_{\text{ETA-COND}}\rceil$, we have to check that the projections of the processes in the swapped interaction η do not change. This follows immediately from the definition of EPP for *(cond)* terms, since merging of the same η is the identity. The other cases follow by similar reasoning on the merging operator. □

Lemma C.5 (Weakening)**.** *Let* $\Gamma \vdash_{\mathsf{min}} C \triangleright \Delta$ *and a be a shared name such that $a \notin \Gamma$; then,* $\Gamma, a : G\langle \mathtt{A} | \tilde{\mathtt{B}} | \emptyset \rangle \vdash_{\mathsf{min}} C \triangleright \Delta$ *for any G, $\mathtt{A}$, $\tilde{\mathtt{B}}$.*

Lemma C.6 (Composability of typing projections)**.** *Let $\Gamma \circ \Gamma' = \Gamma''$; then,* $[\![\Gamma]\!] \circ [\![\Gamma']\!] = [\![\Gamma'']\!]$.

We can finally prove Theorem 3.5.2.

Theorem C.4 (EPP Type Preservation)**.** *Let* $\Gamma \vdash_{\mathsf{min}} C \triangleright \Delta$. *Then,* $[\![\Gamma]\!] \vdash_{\mathsf{min}} [\![C]\!] \triangleright \Delta$.

Proof. Without any loss of generality, we assume that C is restriction-free (the case in which C contains restriction follows easily from this proof). We proceed by induction on the typing derivation $\Gamma \vdash_{\mathsf{min}} C \triangleright \Delta$. The proof is fundamentally similar to that presented in [31]. However, differently from [31], here we also have to deal with partial terms that may appear inside C. We report the most interesting cases.

- **Case** $\lfloor^{\text{MIN}}|_{\text{START1}}\rceil$. In this case we have that:

$$\frac{\Gamma'', \Gamma' \vdash_{\mathsf{min}} C' \triangleright \Delta, \Delta' \quad (\Gamma', \Delta') = \mathsf{init}(\mathsf{p}[\mathtt{A}], \widetilde{\mathsf{q}[\mathtt{B}]}, k, G) \quad \tilde{\mathsf{q}} \notin \Gamma''}{\Gamma'', a{:}G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{B}}\rangle \vdash_{\mathsf{min}} \mathsf{p}[\mathtt{A}]\ \mathbf{start}\ \widetilde{\mathsf{q}[\mathtt{B}]} : a(k); C' \triangleright \Delta}\ \lfloor^{\text{MIN}}|_{\text{START1}}\rceil$$

From the definition of EPP we have that:

$$\begin{aligned}
[\![C]\!] &\equiv \quad C_{\mathsf{act}} \mid C_{\mathsf{env}} \\
C_{\mathsf{act}} &= \quad \mathsf{p}[\mathtt{A}]\ \mathbf{req}\ \tilde{\mathtt{B}} : a(k); [\![C']\!]_{\mathsf{p}} \mid \textstyle\prod_{\mathsf{r}[\mathtt{C}] \in \widetilde{\mathsf{q}[\mathtt{B}]}} \mathbf{acc}\ \mathsf{r}[\mathtt{C}] : a(k); [\![C']\!]_{\mathsf{r}} \\
C_{\mathsf{env}} &= \quad \textstyle\prod_{\mathsf{r} \in \mathsf{fn}(C') \setminus \mathsf{p}} [\![C']\!]_{\mathsf{r}} \mid \prod_{a' \neq a, \mathtt{C}} \left(\bigsqcup_{\mathsf{p} \in \lfloor C' \rfloor^{a'}_{\mathtt{C}}} [\![C']\!]_{\mathsf{p}} \right)
\end{aligned}$$

From the induction hypothesis, we know $[\![\Gamma'', \Gamma']\!] \vdash_{\mathsf{min}} [\![C']\!] \triangleright \Delta, \Delta'$. From the definition of EPP we have that:

$$\begin{aligned}
[\![C']\!] &\equiv \quad C'_{\mathsf{act}} \mid C_{\mathsf{env}} \\
C'_{\mathsf{act}} &= \quad [\![C']\!]_{\mathsf{p}} \mid \textstyle\prod_{\mathsf{r}[\mathtt{C}] \in \widetilde{\mathsf{q}[\mathtt{B}]}} [\![C']\!]_{\mathsf{r}}
\end{aligned}$$

Let $[\![\Gamma'', \Gamma']\!] = \Gamma_{\mathsf{p}} \circ \Gamma_{\tilde{\mathsf{q}}} \circ \Gamma_{\mathsf{env}}$, $\widetilde{\mathsf{q}[\mathtt{B}]} = \mathsf{q}_1[\mathtt{B}_1], \ldots, \mathsf{q}_n[\mathtt{B}_n]$, and $\tilde{\mathsf{q}}' = \tilde{\mathsf{q}} \setminus \mathsf{q}_1$. Since $[\![C']\!]$ is well-typed, and applications of rule $\lfloor^{\text{MIN}}|_{\text{PAR}}\rceil$ are commutable, we know that:

$$\dfrac{\Gamma_{\mathsf{p}} \vdash_{\mathsf{min}} [\![C']\!]_{\mathsf{p}} \triangleright \Delta_{\mathsf{p}} \qquad \dfrac{\vdots \qquad\qquad \vdots}{\dfrac{\Gamma_{\mathsf{q}_1} \vdash_{\mathsf{min}} [\![C']\!]_{\mathsf{q}_1} \triangleright \Delta_{\mathsf{q}_1} \quad \Gamma_{\tilde{\mathsf{q}}'} \vdash_{\mathsf{min}} \prod_{\mathsf{r} \in \tilde{\mathsf{q}}'} [\![C']\!]_{\mathsf{r}} \triangleright \Delta_{\tilde{\mathsf{q}}'}}{\Gamma_{\mathsf{q}_1} \circ \Gamma_{\tilde{\mathsf{q}}'} \vdash_{\mathsf{min}} \prod_{\mathsf{r}[\mathtt{C}] \in \widetilde{\mathsf{q}[\mathtt{B}]}} [\![C']\!]_{\mathsf{r}} \triangleright \Delta_{\mathsf{q}_1}, \Delta_{\tilde{\mathsf{q}}'}}\ \lfloor^{\text{MIN}}|_{\text{PAR}}\rceil}}{\Gamma_{\mathsf{p}} \circ \Gamma_{\tilde{\mathsf{q}}} \vdash_{\mathsf{min}} C'_{\mathsf{act}} \triangleright \Delta_{\mathsf{p}}, \Delta_{\tilde{\mathsf{q}}}}\ \lfloor^{\text{MIN}}|_{\text{PAR}}\rceil$$

Now we distribute $a{:}G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{B}}\rangle$ as:

$$a{:}G\langle \mathtt{A}|\tilde{\mathtt{B}}|\tilde{\mathtt{B}}\rangle \ = \ a{:}G\langle \mathtt{A}|\tilde{\mathtt{B}}|\emptyset\rangle \circ a{:}G\langle \mathtt{A}|\tilde{\mathtt{B}}|\mathtt{B}_1\rangle \circ \ldots \circ a{:}G\langle \mathtt{A}|\tilde{\mathtt{B}}|\mathtt{B}_n\rangle$$

and use the previous typing derivation to obtain the following derivations, for p:

$$\frac{\Gamma_{\mathsf{p}} \vdash_{\mathsf{min}} [\![C']\!]_{\mathsf{p}} \triangleright \Delta_{\mathsf{p}}}{\Gamma_{\mathsf{p}} \setminus k, a{:}G\langle \mathtt{A}|\tilde{\mathtt{B}}|\emptyset\rangle \vdash_{\mathsf{min}} \mathsf{p}[\mathtt{A}]\ \mathbf{req}\ \tilde{\mathtt{B}} : a(k); [\![C']\!]_{\mathsf{p}} \triangleright \Delta_{\mathsf{p}} \setminus k}\ \lfloor^{\text{MIN}}|_{\text{REQ1}}\rceil$$

and for all $\mathsf{q}_i \in \tilde{\mathsf{q}}$:

$$\frac{\Gamma_{\mathsf{q}_i} \vdash_{\mathsf{min}} [\![C']\!]_{\mathsf{q}_i} \triangleright \Delta_{\mathsf{q}_i}}{\Gamma_{\mathsf{q}_i} \setminus (k, \mathsf{q}_i), a{:}G\langle \mathtt{A}|\tilde{\mathtt{B}}|\mathtt{B}_i\rangle \vdash_{\mathsf{min}} \mathbf{acc}\ \mathsf{q}_i[\mathtt{B}_i] : a(k); [\![C']\!]_{\mathsf{p}} \triangleright \Delta_{\mathsf{q}_i} \setminus k}\ \lfloor^{\text{MIN}}|_{\text{ACC}}\rceil$$

The thesis follows now by applying rule $\lfloor^{\text{MIN}}|_{\text{PAR}}\rceil$ to type $[\![C]\!]$ from the derivations above.

- **Case** $\lfloor^{\text{MIN}}|_{\text{COM}}\rceil$. We know that

$$\frac{\Gamma \vdash \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{q}[\mathtt{B}] : k \quad \Gamma \vdash e@\mathtt{p} : S \quad \Gamma, x@\mathtt{q} : S \vdash_{\mathsf{min}} C' \triangleright \Delta', k[\mathtt{A}] : T, k[\mathtt{B}] : T'}{\Gamma \vdash_{\mathsf{min}} \mathtt{p}[\mathtt{A}].e \texttt{ -> } \mathtt{q}[\mathtt{B}].x : k; C' \triangleright \Delta', k[\mathtt{A}] : !\mathtt{B}\langle S\rangle; T, k[\mathtt{B}] : ?\mathtt{A}\langle S\rangle; T'} \lfloor^{\text{MIN}}|_{\text{COM}}\rceil$$

  where $C = \mathtt{p}[\mathtt{A}].e \texttt{ -> } \mathtt{q}[\mathtt{B}].x : k; C'$. From the definition of EPP we have that:

$$\begin{aligned}[\![C]\!] \equiv \quad & C_{\mathsf{act}} \mid \textstyle\prod_{a,\mathtt{A}} \left(\bigsqcup_{\mathtt{p} \in \lfloor C' \rfloor^a_{\mathtt{A}}} [\![C']\!]_{\mathtt{p}} \right) \\ C_{\mathsf{act}} = \quad & \mathtt{p}[\mathtt{A}].e \texttt{ -> } \mathtt{B} : k; [\![C']\!]_{\mathtt{p}} \mid \mathtt{A} \texttt{ -> } \mathtt{q}[\mathtt{B}].x : k; [\![C']\!]_{\mathtt{q}} \\ & \mid \textstyle\prod_{\mathtt{r} \in \mathsf{fn}(C') \setminus \mathtt{p},\mathtt{q}} [\![C']\!]_{\mathtt{r}}\end{aligned}$$

  We can type $\mathtt{p}[\mathtt{A}].e \texttt{ -> } \mathtt{B} : k; [\![C']\!]_{\mathtt{p}}$ and $\mathtt{A} \texttt{ -> } \mathtt{q}[\mathtt{B}].x : k; [\![C']\!]_{\mathtt{q}}$ by using $\lfloor^{\text{MIN}}|_{\text{COM-S}}\rceil$ and $\lfloor^{\text{MIN}}|_{\text{COM-R}}\rceil$ respectively, since their premises are a subset of those given in our hypothesis to $\lfloor^{\text{MIN}}|_{\text{COM}}\rceil$. Then, the thesis follows by induction hypothesis.

- **Case** $\lfloor^{\text{MIN}}|_{\text{COM-S}}\rceil$. We know that

$$\frac{\Gamma \vdash e@\mathtt{p} : S \quad \Gamma \vdash \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{B} : k \quad \Gamma \vdash_{\mathsf{min}} C' \triangleright \Delta', k[\mathtt{A}] : T \quad \mathtt{q} : k[\mathtt{B}] \notin \Gamma}{\Gamma \vdash_{\mathsf{min}} \mathtt{p}[\mathtt{A}].e \texttt{ -> } \mathtt{B} : k; C' \triangleright \Delta', k[\mathtt{A}] : !\mathtt{B}\langle S\rangle; T} \lfloor^{\text{MIN}}|_{\text{COM-S}}\rceil$$

  where $C = \mathtt{p}[\mathtt{A}].e \texttt{ -> } \mathtt{B} : k; C'$. From the definition of EPP we have that:

$$\begin{aligned}[\![C]\!] \equiv \quad & (\boldsymbol{\nu} \tilde{a}) \left((\boldsymbol{\nu} \tilde{k}, \tilde{\mathtt{p}})\, C_{\mathsf{act}} \mid \textstyle\prod_{a,\mathtt{A}} \left(\bigsqcup_{\mathtt{p} \in \lfloor C' \rfloor^a_{\mathtt{A}}} [\![C']\!]_{\mathtt{p}} \right) \right) \\ C_{\mathsf{act}} = \quad & \mathtt{p}[\mathtt{A}].e \texttt{ -> } \mathtt{B} : k; [\![C']\!]_{\mathtt{p}} \mid \textstyle\prod_{\mathtt{r} \in \mathsf{fn}(C') \setminus \{\mathtt{p}\}} [\![C']\!]_{\mathtt{r}}\end{aligned}$$

  The thesis now follows by applying $\lfloor^{\text{MIN}}|_{\text{COM-S}}\rceil$ for typing $\mathtt{p}[\mathtt{A}].e \texttt{ -> } \mathtt{B} : k; [\![C']\!]_{\mathtt{p}}$ and then by applying the induction hypothesis in the reconstruction of the typing of C.

- **Case** $\lfloor^{\text{MIN}}|_{\text{COM-R}}\rceil$. We know that

$$\frac{\Gamma \vdash \mathtt{A} \texttt{ -> } \mathtt{q}[\mathtt{B}] : k \quad \Gamma, x@\mathtt{p} : S \vdash_{\mathsf{min}} C' \triangleright \Delta', k[\mathtt{B}] : T \quad \mathtt{p} : k[\mathtt{A}] \notin \Gamma}{\Gamma \vdash_{\mathsf{min}} \mathtt{A} \texttt{ -> } \mathtt{q}[\mathtt{B}].x : k; C' \triangleright \Delta', k[\mathtt{A}] : ?\mathtt{B}\langle S\rangle; T} \lfloor^{\text{MIN}}|_{\text{COM-R}}\rceil$$

  where $C = \mathtt{A} \texttt{ -> } \mathtt{q}[\mathtt{B}].x : k; C'$. From the definition of EPP we have that:

$$\begin{aligned}[\![C]\!] \equiv \quad & C_{\mathsf{act}} \mid \textstyle\prod_{a,\mathtt{A}} \left(\bigsqcup_{\mathtt{p} \in \lfloor C' \rfloor^a_{\mathtt{A}}} [\![C']\!]_{\mathtt{p}} \right) \\ C_{\mathsf{act}} = \quad & \mathtt{A} \texttt{ -> } \mathtt{q}[\mathtt{B}].x : k; [\![C']\!]_{\mathtt{q}} \mid \textstyle\prod_{\mathtt{r} \in \mathsf{fn}(C') \setminus \{\mathtt{q}\}} [\![C']\!]_{\mathtt{r}}\end{aligned}$$

  The thesis now follows by applying $\lfloor^{\text{MIN}}|_{\text{COM-R}}\rceil$ for typing $\mathtt{A} \texttt{ -> } \mathtt{q}[\mathtt{B}].x : k; [\![C']\!]_{\mathtt{q}}$ and then by applying the induction hypothesis.

- **Case** $\lfloor^{\text{MIN}}|_{\text{PAR}}\rceil$. We know that:

$$\frac{\Gamma_i \vdash_{\mathsf{min}} C_i \triangleright \Delta_i \quad \mathsf{pco}(\Delta_1, \Delta_2)}{\Gamma_1 \circ \Gamma_2 \vdash_{\mathsf{min}} C_1 \mid C_2 \triangleright \Delta_1, \Delta_2} \lfloor^{\text{MIN}}|_{\text{PAR}}\rceil$$

  where $C = C_1 \mid C_2$. The thesis follows by the induction hypothesis and Lemmas 3.5.2 and C.6.

□

B.3.3 Proof of Theorem 3.5.3

In order to prove that our EPP procedure is correct, we need to relate the behaviour of choreographies with that of their respective projections. Similarly to § 2.5.5, we will make use of the notion of strict transitions, defined in the following.

Definition B.3.5 (Strict Transition)**.** A strict transition is a transition where $\boldsymbol{\nu}$-restricted names that are active, i.e., not under a prefix, are not renamed.

Strict transitions are the base of our main assumption in the following proofs:

Assumption B.3.6 (Transitions and Restriction)**.** We assume that all transitions $C \xrightarrow{\lambda} C'$ are *strict transitions*. Furthermore, we assume that rule $\lfloor^{\mathrm{C}}|_{\mathrm{RES}}\rceil$ is changed to the following form:

$$\frac{C \xrightarrow{\lambda} C'}{(\boldsymbol{\nu} r)\, C \xrightarrow{\lambda} (\boldsymbol{\nu} r)\, C'} \lfloor^{\mathrm{C}}|_{\mathrm{RES}}\rceil$$

The assumption above allows us to observe actions of restricted names. Observe that our assumption does not make our proofs any less general, since for every transition there is always a corresponding strict transition (cf. [31]).

Lemma C.7 (Passive processes pruning invariance)**.** *Let C be restriction-free. Then $C \xrightarrow{\lambda} C'$ implies that for every $\mathsf{p} \in \mathsf{fn}(C) \setminus \mathsf{fn}(\lambda)$*

$$[\![C']\!]_{\mathsf{p}} \quad \prec \quad [\![C]\!]_{\mathsf{p}}$$

Proof. By case analysis on the rules defining the semantics of the Choreography Calculus. The only interesting case is $\lfloor^{\mathrm{C}}|_{\mathrm{COND}}\rceil$. In this case, the projections of the processes receiving selections are merged. The thesis follows immediately by definition of pruning. Observe that we can safely ignore potential swaps in C performed in its reduction, thanks to Lemma C.4. □

Lemma C.8 (Pruning Lemma)**.** *Let C be well-typed and $[\![C]\!] = ((\boldsymbol{\nu}\tilde{k})\, C') \mid C''$ where*

$$C'' = \prod_{a,\mathtt{A}} \Big(\bigsqcup_{\mathsf{p} \in \lfloor C \rfloor^{a}_{\mathtt{A}}} [\![C]\!]_{\mathsf{p}} \Big)$$

Let now C''' be the process obtained from C'' by (i) adding some new services on new public channels and (ii) merging the services in C'' with some other services:

$$C''' = \prod_{a,\mathtt{A}} \Big(\big(\bigsqcup_{\mathsf{p} \in \lfloor C \rfloor^{a}_{\mathtt{A}}} [\![C]\!]_{\mathsf{p}} \big) \sqcup R^{a}_{\mathtt{A}} \Big) \mid \prod_{i \in I} !a_i[\mathtt{A}_i](k_i); P_i$$

where for all $i \in I$, $a_i \notin \mathsf{fn}(C)$. Then, C''' defined implies: $[\![C]\!] \prec ((\boldsymbol{\nu}\tilde{k})\, C') \mid C'''$

Proof (Sketch). From the well-typedness of C, we can derive that C' does not use any of the new public channels a_i. The same reasoning can be used for proving that all the new branches introduced in C''' are never going to be used. □

Lemma C.9 (EPP under $\equiv$)**.** *Let $C \equiv C'$. Then, $[\![C]\!] \equiv [\![C']\!]$.*

Theorem C.7 (EPP Theorem)**.** *Let $C \equiv (\boldsymbol{\nu} \tilde{a}, \tilde{k}, \tilde{\mathsf{p}})\ C_f$, where C_f is restriction-free, be well-typed. Then,*

1. *(Completeness) $C \xrightarrow{\lambda} C'$ implies $[\![C]\!] \xrightarrow{\lambda} \succ [\![C']\!]$.*
2. *(Soundness) $[\![C]\!] \xrightarrow{\lambda} C'$ implies $C \xrightarrow{\lambda} C''$ and $[\![C'']\!] \prec C'$.*

We report the proofs for the two results of completeness and soundness separately. They are essentially simpler instances of those seen in Appendix A.2, since differently than for the Choreography Calculus from Chapter 2, with Compositional Choreographies we can now relate the behaviour of a choreography and its EPP step by step. We also have to deal with partial actions, but these are projected as they are in most cases by our definition of process projection, so they add no conceptual complexity.

Proof (Completeness). We proceed by induction on the derivation of $C \xrightarrow{\lambda} C'$.

- **Case** $\lfloor^{\mathrm{C}}|_{\textsc{Com}}\rceil$. We know that:

$$\mathsf{p}[\mathtt{A}].e \texttt{ -> } \mathsf{q}[\mathtt{B}].x : k; C'' \xrightarrow{\lambda} C''[v/x@\mathsf{q}] = C' \qquad (e \downarrow v)$$

  where $\lambda = \mathsf{p}[\mathtt{A}] \texttt{ -> } \mathsf{q}[\mathtt{B}] : k\langle v \rangle$. From the definition of EPP we have:

$$\begin{aligned} [\![C]\!] \equiv \quad & (\boldsymbol{\nu}\tilde{a}) \left((\boldsymbol{\nu}\tilde{k}, \tilde{\mathsf{p}})\ C_{\mathsf{act}} \mid \textstyle\prod_{a,\mathtt{A}} \left(\bigsqcup_{\mathsf{p} \in \lfloor C_f \rfloor^a_{\mathtt{A}}} [\![C_f]\!]_{\mathsf{p}} \right) \right) \\ C_{\mathsf{act}} = \quad & \mathsf{p}[\mathtt{A}].e \texttt{ -> } \mathtt{B} : k; [\![C''_f]\!]_{\mathsf{p}} \mid \mathtt{A} \texttt{ -> } \mathsf{q}[\mathtt{B}].x : k; [\![C''_f]\!]_{\mathsf{q}} \\ & \mid \textstyle\prod_{\mathsf{r} \in \mathsf{fn}(C_f) \setminus \mathsf{p},\mathsf{q}} [\![C_f]\!]_{\mathsf{r}} \end{aligned}$$

  We can now apply rule $\lfloor^{\mathrm{C}}|_{\textsc{Com-S}}\rceil$ for the sending action, rule $\lfloor^{\mathrm{C}}|_{\textsc{Com-R}}\rceil$ for the receiving action, and then rule $\lfloor^{\mathrm{C}}|_{\textsc{Sync}}\rceil$, proving that:

$$C_{\mathsf{act}} \xrightarrow{\lambda} [\![C''_f]\!]_{\mathsf{p}} \mid [\![C''_f]\!]_{\mathsf{q}}[v/x@\mathsf{q}] \mid \prod_{\mathsf{r} \in \mathsf{fn}(C_f) \setminus \mathsf{p},\mathsf{q}}$$

  By the transition above and rules $\lfloor^{\mathrm{C}}|_{\textsc{Par}}\rceil$, $\lfloor^{\mathrm{C}}|_{\textsc{Res}}\rceil$ and $\lfloor^{\mathrm{C}}|_{\textsc{Eq}}\rceil$ we can finally prove the thesis by Lemmas C.3 and C.7.

- **Case** $\lfloor^{\mathrm{C}}|_{\textsc{Act}}\rceil$. Similar to the case above.
- **Case** $\lfloor^{\mathrm{C}}|_{\textsc{Start}}\rceil$. Similar to the case above.
- **Case** $\lfloor^{\mathrm{C}}|_{\textsc{Com-S}}\rceil$. We know that:

$$\mathsf{p}[\mathtt{A}].e \texttt{ -> } \mathtt{B} : k; C' \xrightarrow{\lambda} C'$$

where $\lambda = \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{B} : k\langle v \rangle$ and $e \downarrow v$. From the definition of EPP we have:

$$\begin{aligned}
[\![C]\!] \equiv \quad & (\boldsymbol{\nu}\tilde{a}) \left((\boldsymbol{\nu}\tilde{k}, \tilde{\mathtt{p}})\, C_{\mathsf{act}} \mid \prod_{a,\mathtt{A}} \left(\bigsqcup_{\mathtt{p} \in \lfloor C_f \rfloor^a_{\mathtt{A}}} [\![C_f]\!]_{\mathtt{p}} \right) \right) \\
C_{\mathsf{act}} = \quad & \mathtt{p}[\mathtt{A}].e \texttt{ -> } \mathtt{B} : k; [\![C'_f]\!]_{\mathtt{p}} \\
& \mid \textstyle\prod_{\mathtt{r} \in \mathsf{fn}(C_f) \setminus \mathtt{p}} [\![C_f]\!]_{\mathtt{r}}
\end{aligned}$$

Since C is well-typed, we know that:

$$\frac{\Gamma \vdash e@\mathtt{p} : S \quad \Gamma \vdash \mathtt{p}[\mathtt{A}] \texttt{ -> } \mathtt{B} : k \quad \Gamma \vdash C' \triangleright \Delta, k[\mathtt{A}] : T \quad \mathtt{q} : k[\mathtt{B}] \notin \Gamma}{\Gamma \vdash \mathtt{p}[\mathtt{A}].e \texttt{ -> } \mathtt{B} : k; C' \triangleright \Delta, k[\mathtt{A}] : !\mathtt{B}\langle S \rangle; T} \lfloor^{\mathrm{T}}|_{\text{COM-S}}\rceil$$

Hence, we know that the receiver for the message sent by process $\mathtt{p}$ is not in C and therefore, by Theorem 3.5.2, not in $[\![C]\!]$. Consequently, we can apply rule $\lfloor^{\mathrm{C}}|_{\text{COM-S}}\rceil$ for the sending action and rule $\lfloor^{\mathrm{C}}|_{\text{PAR}}\rceil$ for obtaining:

$$C_{\mathsf{act}} \xrightarrow{\lambda} [\![C'_f]\!]_{\mathtt{p}} \mid \prod_{\mathtt{r} \in \mathsf{fn}(C_f) \setminus \mathtt{p}}$$

By the transition above and rules $\lfloor^{\mathrm{C}}|_{\text{PAR}}\rceil$, $\lfloor^{\mathrm{C}}|_{\text{RES}}\rceil$ and $\lfloor^{\mathrm{C}}|_{\text{EQ}}\rceil$ we can finally prove the thesis by applying Lemma C.7.

- **Case** $\lfloor^{\mathrm{C}}|_{\text{COM-R}}\rceil$. We know that:

$$\mathtt{A} \texttt{ -> } \mathtt{q}[\mathtt{B}].x : k; C'' \xrightarrow{\lambda} C''[v/x@\mathtt{q}] = C'$$

where $\lambda = \mathtt{A} \texttt{ -> } \mathtt{q}[\mathtt{B}] : k\langle v \rangle$. From the definition of EPP we have:

$$\begin{aligned}
[\![C]\!] \equiv \quad & (\boldsymbol{\nu}\tilde{a}) \left((\boldsymbol{\nu}\tilde{k}, \tilde{\mathtt{p}})\, C_{\mathsf{act}} \mid \prod_{a,\mathtt{A}} \left(\bigsqcup_{\mathtt{p} \in \lfloor C_f \rfloor^a_{\mathtt{A}}} [\![C_f]\!]_{\mathtt{p}} \right) \right) \\
C_{\mathsf{act}} = \quad & \mathtt{A} \texttt{ -> } \mathtt{q}[\mathtt{B}].x : k; [\![C''_f]\!]_{\mathtt{q}} \\
& \mid \textstyle\prod_{\mathtt{r} \in \mathsf{fn}(C_f) \setminus \mathtt{q}} [\![C_f]\!]_{\mathtt{r}}
\end{aligned}$$

Since C is well-typed, we know that:

$$\frac{\Gamma \vdash \mathtt{A} \texttt{ -> } \mathtt{q}[\mathtt{B}] : k \quad \Gamma, x@\mathtt{p} : S \vdash C'' \triangleright \Delta, k[\mathtt{B}] : T \quad \mathtt{p} : k[\mathtt{A}] \notin \Gamma}{\Gamma \vdash \mathtt{A} \texttt{ -> } \mathtt{q}[\mathtt{B}].x : k; C'' \triangleright \Delta, k[\mathtt{A}] : ?\mathtt{B}\langle S \rangle; T} \lfloor^{\mathrm{T}}|_{\text{COM-R}}\rceil$$

Hence, we know that the receiver for the message sent by process $\mathtt{p}$ is not in C and therefore, by Theorem 3.5.2, not in $[\![C]\!]$. Consequently, we can apply rule $\lfloor^{\mathrm{C}}|_{\text{COM-R}}\rceil$ for the sending action and rule $\lfloor^{\mathrm{C}}|_{\text{PAR}}\rceil$ for obtaining:

$$C_{\mathsf{act}} \xrightarrow{\lambda} [\![C'_f]\!]_{\mathtt{q}} \mid \prod_{\mathtt{r} \in \mathsf{fn}(C_f) \setminus \mathtt{q}}$$

By the transition above and rules $\lfloor^{\mathrm{C}}|_{\text{PAR}}\rceil$, $\lfloor^{\mathrm{C}}|_{\text{RES}}\rceil$ and $\lfloor^{\mathrm{C}}|_{\text{EQ}}\rceil$ we can finally prove the thesis by applying Lemma C.7.

- **Case** $\lfloor^{\mathrm{C}}|_{\textsc{Branch}}\rceil$. Similar to case $\lfloor^{\mathrm{C}}|_{\textsc{Com-R}}\rceil$.
- **Case** $\lfloor^{\mathrm{C}}|_{\textsc{Sync}}\rceil$. In this case we have:

$$\frac{C_1 \xrightarrow{\lambda_1} C_1' \quad C_2 \xrightarrow{\lambda_2} C_2'}{C = C_1 \mid C_2 \xrightarrow{\lambda_1 \circ \lambda_2} C_1' \mid C_2' = C'} \lfloor^{\mathrm{C}}|_{\textsc{Sync}}\rceil$$

  where $\lambda = \lambda_1 \circ \lambda_2$. Using the induction hypothesis we can prove the following:

$$\frac{[\![C_1]\!] \xrightarrow{\lambda_1} \succ [\![C_1']\!] \quad [\![C_2]\!] \xrightarrow{\lambda_2} \succ [\![C_2']\!]}{[\![C_1]\!] \mid [\![C_2]\!] \xrightarrow{\lambda_1 \circ \lambda_2} \succ [\![C_1']\!] \mid [\![C_2']\!]} \lfloor^{\mathrm{C}}|_{\textsc{Sync}}\rceil$$

  The thesis now follows by Lemma 3.5.2.
- **Case** $\lfloor^{\mathrm{C}}|_{\textsc{Cond}}\rceil$. Follows by similar reasoning of that for case $\lfloor^{\mathrm{C}}|_{\textsc{Com-S}}\rceil$ and by the definition of merging.
- **Case** $\lfloor^{\mathrm{C}}|_{\textsc{Res}}\rceil$. In this case we have:

$$\frac{C'' \xrightarrow{\lambda} C'''}{C = (\boldsymbol{\nu} r)\, C'' \xrightarrow{(\boldsymbol{\nu} r)\,\lambda} (\boldsymbol{\nu} r)\, C''' = C'} \lfloor^{\mathrm{C}}|_{\textsc{Res}}\rceil$$

  Using the induction hypothesis we can prove:

$$\frac{[\![C'']\!] \xrightarrow{\lambda} \succ [\![C''']\!]}{(\boldsymbol{\nu} r)\, [\![C'']\!] \xrightarrow{(\boldsymbol{\nu} r)\,\lambda} \succ (\boldsymbol{\nu} r)\, [\![C''']\!]} \lfloor^{\mathrm{C}}|_{\textsc{Res}}\rceil$$

  The thesis follows by definition of pruning.
- **Case** $\lfloor^{\mathrm{C}}|_{\textsc{Async}}\rceil$. We know that:

$$C_{\mathsf{a}} \xrightarrow{\lambda} (\boldsymbol{\nu}\tilde{r})\, C_{\mathsf{a}}' \;\Rightarrow\; \eta; C_{\mathsf{a}} \xrightarrow{\lambda} (\boldsymbol{\nu}\tilde{r})\, \eta; C_{\mathsf{a}}' \begin{pmatrix} \mathsf{snd}(\eta) \in \mathsf{fn}(\lambda) & \tilde{r} = \mathsf{bn}(\lambda) \\ \mathsf{rcv}(\eta) \notin \mathsf{fc}(\lambda) & \tilde{r} \notin \mathsf{fn}(\eta) \\ \eta \notin \{(start), (acc)\} & \mathsf{sdc}(\eta, \lambda) \end{pmatrix} \tag{B.1}$$

  where $C = \eta; C_{\mathsf{a}}$ and $C' = (\boldsymbol{\nu}\tilde{r})\, \eta; C_{\mathsf{a}}'$. Let us analyse the case in which $\eta = \mathtt{p}[\mathtt{A}].e \texttt{ -> } \mathtt{q}[\mathtt{B}].x : k$ (the other cases are similar). By the definition of EPP we have that:

$$\begin{aligned} [\![C]\!] \equiv \quad & (\boldsymbol{\nu}\tilde{a}) \left((\boldsymbol{\nu}\tilde{k}, \tilde{\mathsf{p}})\, C_{\mathsf{act}} \mid \prod_{a,\mathtt{A}} \left(\bigsqcup_{\mathsf{p} \in \lfloor C_f \rfloor_{\mathtt{A}}^{a}} [\![C_f]\!]_{\mathsf{p}} \right) \right) \\ C_{\mathsf{act}} = \quad & \mathtt{p}[\mathtt{A}].e \texttt{ -> } \mathtt{B} : k; [\![C_f'']\!]_{\mathtt{p}} \mid \mathtt{A} \texttt{ -> } \mathtt{q}[\mathtt{B}].x : k; [\![C_f'']\!]_{\mathtt{q}} \\ & \mid \textstyle\prod_{\mathsf{r} \in \mathsf{fn}(C_f) \setminus \mathtt{p},\mathtt{q}} [\![C_f]\!]_{\mathsf{r}} \end{aligned}$$

  From the side conditions in B.1, we know that we can apply $\lfloor^{\mathrm{C}}|_{\textsc{Async}}\rceil$ to $\mathtt{p}[\mathtt{A}].e \texttt{ -> } \mathtt{B} : k; [\![C_f'']\!]_{\mathtt{p}}$. Moreover, we know that $\mathsf{rcv}(\eta) = k[\mathtt{B}] \notin \mathsf{fc}(\lambda)$. Therefore, we have two possibilities for $[\![C]\!]$ to mimic λ: either λ is an internal action $\tau@\mathtt{p}$ that can be performed also by $[\![C_f'']\!]_{\mathtt{p}}$, or $[\![C_f'']\!]_{\mathtt{p}}$ needs to interact with another process in parallel. In both cases, the thesis follows from the induction hypothesis by case analysis on λ.

- **Case** $\lfloor^{\mathrm{C}}|_{\textsc{Eq}}\rceil$. For $\mathcal{R} = \equiv$, the thesis follows by induction hypothesis and rule $\lfloor^{\mathrm{C}}|_{\textsc{Eq}}\rceil$. Otherwise, for $\mathcal{R} = \simeq_{\mathcal{C}}$, the thesis follows from the induction hypothesis and Lemma C.4.

- **Cases** $\lfloor^{\mathrm{C}}|_{\textsc{Ctx}}\rceil$, $\lfloor^{\mathrm{C}}|_{\textsc{Par}}\rceil$, $\lfloor^{\mathrm{C}}|_{\textsc{p-Start}}\rceil$. These cases are similar to the ones discussed above.

□

Proof (Soundness). The proof proceeds by induction on the structure of C_f. We report the most interesting cases.

- **Case** $C_f = \mathsf{p}[\mathtt{A}].e \texttt{ -> } \mathsf{q}[\mathtt{B}].x : k; C_c$. From the definition of EPP we have that:

$$\begin{aligned}
[\![C]\!] \equiv\ & (\boldsymbol{\nu}\tilde{a})\left((\boldsymbol{\nu}\tilde{k},\tilde{\mathsf{p}})\ C_r \mid \textstyle\prod_{a,\mathtt{A}}\left(\bigsqcup_{\mathsf{p}\in\lfloor C_c\rfloor^a_{\mathtt{A}}}[\![C_c]\!]_{\mathsf{p}}\right)\right)\\
C_r = \quad & \mathsf{p}[\mathtt{A}].e \texttt{ -> } \mathtt{B} : k; [\![C_c]\!]_{\mathsf{p}} \mid \mathtt{A} \texttt{ -> } \mathsf{q}[\mathtt{B}].x : k; [\![C_c]\!]_{\mathsf{q}}\\
& \mid \textstyle\prod_{\mathsf{r}\in\mathsf{fn}(C_c)\setminus\mathsf{p},\mathsf{q}}[\![C_c]\!]_{\mathsf{r}}
\end{aligned}$$

  We proceed now by case analysis on the reduction $[\![C]\!] \xrightarrow{\lambda} C'$.

  - p and q communicate through $\lfloor^{\mathrm{C}}|_{\textsc{Sync}}\rceil$, by executing their respective prefixes $\mathsf{p}[\mathtt{A}].e \texttt{ -> } \mathtt{B} : k$ and $\mathtt{A} \texttt{ -> } \mathsf{q}[\mathtt{B}].x : k$. In this case, we obtain that:

$$\begin{aligned}
C' \equiv\ & (\boldsymbol{\nu}\tilde{a})\left((\boldsymbol{\nu}\tilde{k},\tilde{\mathsf{p}})\ C_r \mid \textstyle\prod_{a,\mathtt{A}}\left(\bigsqcup_{\mathsf{p}\in\lfloor C_c\rfloor^a_{\mathtt{A}}}[\![C_c]\!]_{\mathsf{p}}\right)\right)\\
C_r = \quad & [\![C_c]\!]_{\mathsf{p}} \mid [\![C_c]\!]_{\mathsf{q}}[v/x@\mathsf{q}] \mid \textstyle\prod_{\mathsf{r}\in\mathsf{fn}(C_c)\setminus\mathsf{p},\mathsf{q}}[\![C_c]\!]_{\mathsf{r}}
\end{aligned}$$

    Also, we know that $\lambda = \mathsf{p}[\mathtt{A}] \texttt{ -> } \mathsf{q}[\mathtt{B}] : k\langle v\rangle$. Clearly, C can mimic the transition of $[\![C]\!]$ by executing its prefix $\mathsf{p}[\mathtt{A}].e \texttt{ -> } \mathsf{q}[\mathtt{B}].x : k$ through rule $\lfloor^{\mathrm{C}}|_{\textsc{Com}}\rceil$. We obtain that $C \xrightarrow{\lambda} C''[v/x@\mathsf{q}]$. The thesis follows by Lemma C.3.

  - p may communicate with another process r in $\prod_{\mathsf{r}\in\mathsf{fn}(C_c)\setminus\mathsf{p},\mathsf{q}}$. In this case, it must be that p is executing a sending action and that the derivation for its transition ended with an application of rule $\lfloor^{\mathrm{C}}|_{\textsc{Async}}\rceil$. Therefore, it must be case (from the premises of $\lfloor^{\mathrm{C}}|_{\textsc{Async}}\rceil$) that we can apply $\lfloor^{\mathrm{C}}|_{\textsc{Async}}\rceil$ also for C and mimic the transition correctly.

  - p may start a new session with a service in $\prod_{a,\mathtt{A}}\left(\bigsqcup_{\mathsf{p}\in\lfloor C_c\rfloor^a_{\mathtt{A}}}[\![C_c]\!]_{\mathsf{p}}\right)$. The reasoning for this case is similar to the one above.

  - Other two processes, say r and s in $\mathsf{fn}(C_r)\setminus\mathsf{p},\mathsf{q}$, may interact inside $\prod_{\mathsf{r}\in\mathsf{fn}(C_c)\setminus\mathsf{p},\mathsf{q}}$. In this case, since we know that r and s are different than p and q, C can mimic the transition by using the swapping relation $\lfloor^{\mathrm{C}}|_{\textsc{Swap}}\rceil$.

  - A process r in $\prod_{\mathsf{r}\in\mathsf{fn}(C_c)\setminus\mathsf{p},\mathsf{q}}$ may start a new session with a service in $\prod_{a,\mathtt{A}}\left(\bigsqcup_{\mathsf{p}\in\lfloor C_c\rfloor^a_{\mathtt{A}}}[\![C_c]\!]_{\mathsf{p}}\right)$. The reasoning for this case is similar to the one above.

- **Case** $C_f = \mathsf{p}[\mathtt{A}].e \texttt{ -> } \mathtt{B} : k; C_c$. From the definition of EPP we have that:

$$\begin{aligned} [\![C]\!] \equiv \quad & (\boldsymbol{\nu}\tilde{a}) \left((\boldsymbol{\nu}\tilde{k}, \tilde{\mathsf{p}})\ C_r \mid \prod_{a,\mathtt{A}} \left(\bigsqcup_{\mathsf{p} \in \lfloor C_c \rfloor^a_{\mathtt{A}}} [\![C_c]\!]_{\mathsf{p}} \right) \right) \\ C_r = \quad & \mathsf{p}[\mathtt{A}].e \texttt{ -> } \mathtt{B} : k; [\![C_c]\!]_{\mathsf{p}} \mid \prod_{\mathsf{r} \in \mathsf{fn}(C_c) \setminus \mathsf{p}} [\![C_c]\!]_{\mathsf{r}} \end{aligned}$$

  We proceed now by case analysis on the reduction $[\![C]\!] \xrightarrow{\lambda} C'$.

  - p may execute its sending action $p.e \texttt{ -> } \mathtt{B} : k$ by $\lfloor^{\mathrm{C}}|_{\textsc{Com-S}}\rceil$. In this case, by rules $\lfloor^{\mathrm{C}}|_{\textsc{Par}}\rceil$ and $\lfloor^{\mathrm{C}}|_{\textsc{Res}}\rceil$ we would have that $\lambda = \mathsf{p}[\mathtt{A}] \texttt{ -> } \mathtt{B} : k\langle v \rangle$, where $e \downarrow v$, and that:

$$\begin{aligned} C' \equiv \quad & (\boldsymbol{\nu}\tilde{a}) \left((\boldsymbol{\nu}\tilde{k}, \tilde{\mathsf{p}})\ C_r \mid \prod_{a,\mathtt{A}} \left(\bigsqcup_{\mathsf{p} \in \lfloor C_c \rfloor^a_{\mathtt{A}}} [\![C_c]\!]_{\mathsf{p}} \right) \right) \\ C_r = \quad & [\![C_c]\!]_{\mathsf{p}} \mid \prod_{\mathsf{r} \in \mathsf{fn}(C_c) \setminus \mathsf{p}} [\![C_c]\!]_{\mathsf{r}} \end{aligned}$$

    The thesis follows from the definition of pruning.

  - p may communicate with another process r in $\prod_{\mathsf{r} \in \mathsf{fn}(C_c) \setminus \mathsf{p}}$. In this case, it must be that p is executing a sending action and that the derivation for its transition ended with an application of rule $\lfloor^{\mathrm{C}}|_{\textsc{Async}}\rceil$. Therefore, it must be case (from the premises of $\lfloor^{\mathrm{C}}|_{\textsc{Async}}\rceil$) that we can apply $\lfloor^{\mathrm{C}}|_{\textsc{Async}}\rceil$ also for C and mimic the transition correctly.

  - p may start a new session with a service in $\prod_{a,\mathtt{A}} \left(\bigsqcup_{\mathsf{p} \in \lfloor C_c \rfloor^a_{\mathtt{A}}} [\![C_c]\!]_{\mathsf{p}} \right)$. The reasoning for this case is similar to the one above.

  - Other two processes, say r and s in $\mathsf{fn}(C_r) \setminus \mathsf{p}$, may interact inside $\prod_{\mathsf{r} \in \mathsf{fn}(C_c) \setminus \mathsf{p}}$. In this case, since we know that r and s are different than p, C can mimic the transition by using the swapping relation $\lfloor^{\mathrm{C}}|_{\textsc{Swap}}\rceil$.

  - A process r in $\prod_{\mathsf{r} \in \mathsf{fn}(C_c) \setminus \mathsf{p}}$ may start a new session with a service in $\prod_{a,\mathtt{A}} \left(\bigsqcup_{\mathsf{p} \in \lfloor C_c \rfloor^a_{\mathtt{A}}} [\![C_c]\!]_{\mathsf{p}} \right)$. The reasoning for this case is similar to the one above.

- **Case** $C_f = C_1 \mid C_2$. From the definition of EPP we have that:

$$\begin{aligned} [\![C]\!] \equiv \quad & (\boldsymbol{\nu}\tilde{a}) \left((\boldsymbol{\nu}\tilde{k}, \tilde{\mathsf{p}})\ C_r \mid \prod_{a,\mathtt{A}} \left(\bigsqcup_{\mathsf{p} \in \lfloor C_f \rfloor^a_{\mathtt{A}}} [\![C_1 \mid C_2]\!]_{\mathsf{p}} \right) \right) \\ C_r = \quad & \prod_{\mathsf{r} \in \mathsf{fn}(C_f)} [\![C_1 \mid C_2]\!]_{\mathsf{r}} \end{aligned}$$

By Lemma 3.5.2, we can rewrite the above as:

$$
\begin{aligned}
[\![C]\!] \equiv \quad & [\![C_1]\!] \mid [\![C_2]\!] \\
[\![C_1]\!] \equiv \quad & (\boldsymbol{\nu} \tilde{a}_1) \left((\boldsymbol{\nu} \tilde{k}_1, \tilde{\mathsf{p}}_1)\, C_{r1} \mid \prod_{a,\mathtt{A}} \left(\bigsqcup_{\mathsf{p} \in \lfloor C_{r1} \rfloor^a_{\mathtt{A}}} [\![C_{r1}]\!]_{\mathsf{p}} \right) \right) \\
C_{r1} = \quad & \prod_{\mathsf{r} \in \mathsf{fn}(C_{r1})} [\![C_{r1}]\!]_{\mathsf{r}} \\
[\![C_2]\!] \equiv \quad & (\boldsymbol{\nu} \tilde{a}_2) \left((\boldsymbol{\nu} \tilde{k}_2, \tilde{\mathsf{p}}_2)\, C_{r2} \mid \prod_{a,\mathtt{A}} \left(\bigsqcup_{\mathsf{p} \in \lfloor C_{r2} \rfloor^a_{\mathtt{A}}} [\![C_{r2}]\!]_{\mathsf{p}} \right) \right) \\
C_{r2} = \quad & \prod_{\mathsf{r} \in \mathsf{fn}(C_{r2})} [\![C_{r2}]\!]_{\mathsf{r}}
\end{aligned}
$$

where $\tilde{a}_i$, $\tilde{k}_i$, and $\tilde{\mathsf{p}}_i$ are respectively the free shared channel, session, and process names in C_i where $i \in \{1, 2\}$. We proceed now by case analysis on the last applied rule for the transition $[\![C]\!] \xrightarrow{\lambda} C'$.

– **Case** $\lfloor {}^{\mathrm{C}}|_{\textsc{Par}} \rceil$. In this case, we know that:

$$
\frac{[\![C_1]\!] \xrightarrow{\lambda} C_1' \quad (\, \lambda \in \mathsf{CAct} \vee \mathsf{rc}(\lambda) \notin \mathsf{fc}([\![C_2]\!]))}{[\![C_1]\!] \mid [\![C_2]\!] \xrightarrow{\lambda} C_1' \mid [\![C_2]\!]} \lfloor {}^{\mathrm{C}}|_{\textsc{Par}} \rceil
$$

Now we have two subcases, depending on the condition respected by λ.

* If $\lambda \in \mathsf{CAct}$, then the thesis follows by induction hypothesis.
* Otherwise, if $\lambda \notin \mathsf{CAct}$ and $\mathsf{rc}(\lambda) \notin \mathsf{fc}([\![C_2]\!])$, then from the typing rules we know that $\mathsf{rc}(\lambda)$ does not appear in $[\![C_1]\!]$ either. The thesis follows then from the induction hypothesis and Lemma C.1.

– **Case** $\lfloor {}^{\mathrm{C}}|_{\textsc{Sync}} \rceil$. In this case we know that:

$$
[\![C_1]\!] \xrightarrow{\lambda_1} C_1' \quad [\![C_2]\!] \xrightarrow{\lambda_2} C_2' \quad \Rightarrow \quad [\![C_1]\!] \mid [\![C_2]\!] \xrightarrow{\lambda} C'
$$

where $\lambda = \lambda_1 \circ \lambda_2$. From the typing rules we know that C_1 and C_2 have disjoint session/role pairs. Therefore, the terms emitting the labels λ_1 and λ_2 cannot be the projection of a complete action in C; rather, the two terms must have already been separate partial actions in C. We can conclude by applying the induction hypothesis to C_1 and C_2.

– **Case** $\lfloor {}^{\mathrm{C}}|_{\textsc{p-Start}} \rceil$. This case is similar to the one above.

The other cases follow similar reasonings to the ones above. □

B.4 Proof of Theorem 3.5.6

We first discuss the intuition behind our proof method. Our objective is to prove that given a (possibly partial) well-typed choreography C that contains no *(par)* terms, there exists another choreography C' such that the parallel composition of C and C' is deadlock-free.

The idea is to transform C into a new *complete* choreography $(\!|C|\!)$, obtained by "filling" all partial terms in C with their missing participants. By well-typedness, we know that the projection of $(\!|C|\!)$, $[\![(\!|C|\!)]\!]$, will contain that of $[\![C]\!]$ since we will be simply adding new processes. We will then derive the result we wish for (progress for partial choreographies) by applying Theorems 3.5.5 and 3.5.3.

We start by defining $(\!|C|\!)$, i.e., the transformation of a generic choreography C into a complete choreography. Formally, $(\!|C|\!)$ is defined as $(\!|C|\!)_\emptyset$, which is inductively defined by the rules reported in Figure B.9. Completion adds the missing participants to each partial term in a choreography[1]. We use the parameter Γ to remember which process has been assigned to each role in each session, to guarantee that the completion of a well-typed choreography is a well-typed and projectable complete choreography:

Lemma D.1 (Completion Lemma)**.** *Let C be well-typed and $[\![C]\!]$ be defined; then $(\!|C|\!)$ is a well-typed complete choreography and its projection, $[\![(\!|C|\!)]\!]$ is defined.*

Proof. By induction on the rules reported in Figure B.9. □

We can finally prove Theorem 3.5.6.

Theorem D.1 (Progress for Partial Choreographies)**.** *Let C be a choreography, be well-typed, and contain no* (par) *terms. Then, there exists C' such that $(\boldsymbol{\nu}\tilde{r})\ (C \mid C')$ with $\tilde{r} = \mathsf{fn}(C \mid C')$, is well-typed and deadlock-free.*

Proof. By Lemma D.1, we know that $(\!|C|\!)$ is well-typed and that $[\![(\!|C|\!)]\!]$ is defined. From the typing rules, specifically the distribution of service roles, we know that the projection of the completion of C simply adds the partial choreographies of the missing processes in C to $[\![C]\!]$. Let C' be the parallel composition of such partial choreographies; then we have:

$$[\![(\!|C|\!)]\!] \equiv [\![C]\!] \mid C' \tag{B.2}$$

By Theorem 3.5.2 we know that $[\![(\!|C|\!)]\!]$ is well-typed. Hence, by (B.2) and rule $\lfloor^{\mathrm{T}}|_{\mathrm{PAR}}\rceil$ we know that also C' is well-typed. From this and rule $\lfloor^{\mathrm{T}}|_{\mathrm{PAR}}\rceil$ again we can conclude that $(\boldsymbol{\nu}\tilde{r})\ (C \mid C')$ is well-typed.

Now, by Theorem 3.5.5 we know that $(\!|C|\!)$ is deadlock-free since it is a complete choreography. By Corollary 3.5.1.1, $[\![(\!|C|\!)]\!]$ is also deadlock-free. Finally, since $[\![(\!|C|\!)]\!] \equiv [\![C]\!] \mid C'$ then by Theorem 3.5.3 we know that also $(\boldsymbol{\nu}\tilde{r})\ C \mid C'$ is deadlock-free since C behaves as $[\![C]\!]$. □

[1] Completion is actually defined only for choreography programs, i.e., choreographies in which all sessions are created at runtime. We made this choice for readability, but we could easily extend completion to runtime choreographies by adding more typing information to $(\!|\,|\!)$.

$$
\begin{array}{rcll}
(\!|\eta; C|\!)_\Gamma & = & \eta; (\!|C|\!)_\Gamma & (\ \eta \text{ is a complete term}\ ) \\[1ex]
(\!|\mathsf{p}[\mathtt{A}].e \texttt{ -> } \mathtt{B} : k; C|\!)_\Gamma & = & \mathsf{p}[\mathtt{A}].e \texttt{ -> } \mathsf{q}[\mathtt{B}].x : k; \\
 & & (\!|C|\!)_{\Gamma, x@\mathsf{q}:S} & (\ \Gamma \vdash \mathsf{q} : k[\mathtt{B}]\ x \text{ fresh}\ ) \\[1ex]
(\!|\mathtt{A} \texttt{ -> } \mathsf{q}[\mathtt{B}].x : k; C|\!)_\Gamma & = & \mathsf{p}[\mathtt{A}].v \texttt{ -> } \mathsf{q}[\mathtt{B}].x : k; \\
 & & (\!|C|\!)_\Gamma & (\ \Gamma \vdash \mathsf{p} : k[\mathtt{A}], x@\mathsf{q} : S\ ) \\[1ex]
(\!|\mathsf{p}[\mathtt{A}] \texttt{ -> } \mathtt{B} : k \oplus l; C|\!)_\Gamma & = & \mathsf{p}[\mathtt{A}] \texttt{ -> } \mathsf{q}[\mathtt{B}] : k[l]; \\
 & & (\!|C|\!)_\Gamma & (\ \Gamma \vdash \mathsf{q} : k[\mathtt{B}]\ ) \\[1ex]
(\!|\mathsf{p}[\mathtt{A}] \texttt{ -> } \mathtt{B} : k\langle k'[\mathtt{C}]\rangle; C|\!)_\Gamma & = & \mathsf{p}[\mathtt{A}] \texttt{ -> } \mathsf{q}[\mathtt{B}] : k\langle k'[\mathtt{C}]\rangle; \\
 & & (\!|C|\!)_{\Gamma, \mathsf{q}:k'[\mathtt{C}]} & (\ \Gamma \vdash \mathsf{q} : k[\mathtt{B}]\ ) \\[1ex]
(\!|\mathtt{A} \texttt{ -> } \mathsf{q}[\mathtt{B}] : k(k'[\mathtt{C}]); C|\!)_\Gamma & = & \mathsf{p}[\mathtt{A}] \texttt{ -> } \mathsf{q}[\mathtt{B}] : k\langle k'[\mathtt{C}]\rangle; \\
 & & (\!|C|\!)_{\Gamma \setminus \mathsf{p}:k'[\mathtt{C}]} & (\ \Gamma \vdash \mathsf{p} : k[\mathtt{A}], \mathsf{p} : k'[\mathtt{C}]\ ) \\[1ex]
(\!|\mathsf{p}[\mathtt{A}]\ \mathbf{req}\ \tilde{\mathtt{B}} : a(k); C|\!)_\Gamma & = & \mathsf{p}[\mathtt{A}]\ \mathbf{start}\ \widetilde{\mathsf{q}[\mathtt{B}]} : a(k); \\
 & & (\!|C|\!)_{\Gamma, \mathsf{q}_i : \widetilde{k[\mathtt{B}_i]}} & (\ \tilde{\mathsf{q}} \text{ fresh}\ ) \\[1ex]
(\!|\mathbf{acc}\ \widetilde{\mathsf{r}[\mathtt{C}]} : a(k); C|\!)_\Gamma & = & \mathsf{p}[\mathtt{A}]\ \mathbf{start}\ \widetilde{\mathsf{q}[\mathtt{B}]} : a(k); \\
 & & (\!|C|\!)_{\Gamma, \mathsf{s}_i : \widetilde{k[\mathtt{B}_i]}} & \left(\begin{array}{l} \widetilde{\mathsf{r}[\mathtt{C}]} \subseteq \widetilde{\mathsf{q}[\mathtt{B}]} \\ \Gamma \vdash a : G\langle \mathtt{A} | \tilde{\mathtt{B}} | \tilde{\mathtt{C}} \rangle \\ \tilde{\mathsf{q}} \setminus \tilde{\mathsf{r}} = \tilde{\mathsf{s}} \text{ fresh} \end{array} \right) \\[1ex]
(\!|\mathtt{A} \texttt{ -> } \mathsf{q}[\mathtt{B}] : k\&\{l : C\}|\!)_\Gamma & = & \mathsf{p}[\mathtt{A}] \texttt{ -> } \mathsf{q}[\mathtt{B}] : k[l]; \\
 & & (\!|C|\!)_\Gamma & (\ \Gamma \vdash \mathsf{p} : k[\mathtt{A}]\ ) \\[1ex]
(\!|\mathtt{A} \texttt{ -> } \mathsf{q}[\mathtt{B}] : k\&\{l_i : C_i\}_{i\in[1,n]}|\!)_\Gamma & = & & \\
 & & \multicolumn{2}{l}{\mathsf{if}\ \mathsf{true}@\mathsf{p}\ \mathsf{then}\ (\!|\mathtt{A} \texttt{ -> } \mathsf{q}[\mathtt{B}] : k\&\{l_1 : C_1\}|\!)} \\
 & & \multicolumn{2}{l}{\mathsf{else}\ (\!|\mathtt{A} \texttt{ -> } \mathsf{q}[\mathtt{B}] : k\&\{l_i : C_i\}_{i\in[2,n]}|\!)_\Gamma \quad (\ \Gamma \vdash \mathsf{p} : k[\mathtt{A}]\ n \geq 2\ )} \\[1ex]
(\!|\mathsf{if}\ e@\mathsf{p}\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2|\!)_\Gamma & = & \mathsf{if}\ e@\mathsf{p}\ \mathsf{then}\ (\!|C_1|\!) & \\
 & & \quad \mathsf{else}\ (\!|C_2|\!) & \\[1ex]
(\!|\mathsf{def}\ X(\tilde{D}) = C'\ \mathsf{in}\ C|\!)_\Gamma & = & \mathsf{def}\ X(\tilde{D}, \tilde{D}') & \\
 & & \quad = (\!|C'|\!)_{\Gamma, X(\tilde{D}')} & \\
 & & \mathsf{in}\ (\!|C|\!)_{\Gamma, X(\tilde{D}')} & (\ \tilde{D}' = \mathsf{args}(\Gamma)|_{\mathsf{fn}(C')}\ ) \\[1ex]
(\!|X\langle \tilde{E} \rangle|\!)_{\Gamma, X(\tilde{D}')} & = & X\langle \tilde{E}, \tilde{E}' \rangle & (\ \tilde{E}' = \mathsf{vals}(\tilde{D}')\ ) \\[1ex]
(\!|\mathbf{0}|\!)_\Gamma & = & \mathbf{0} &
\end{array}
$$

Figure B.9: Compositional Choreographies, completion.

APPENDIX C

Round-Trip Choreographic Programming: Additional Material

In this Appendix we report all our proof transformations that justify our commuting conversions, β-reductions and $\alpha\gamma$-rules. For presentational convenience, we report LCL proofs with their respective proof terms (which are highlighted).

The appendix is organised as follows:

C.1 Commuting Conversions

Commuting Scope with Conn

$$[\mathsf{Scope/Conn/L}] \quad (\boldsymbol{\nu} y)\,(P \mid_x Q) \equiv (\boldsymbol{\nu} y)\,P \mid_x Q \qquad \Big(y \notin \mathsf{fn}(Q)\Big)$$

Case 1:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash y : \bullet D \mid \Delta_2, y : \bullet D \vdash T' \mid \Delta \vdash x : A \quad Q \triangleright \Psi' \mid \Delta', x : A \vdash T}{P \mid_x Q \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash y : \bullet D \mid \Delta_2, y : \bullet D \vdash T' \mid \Delta \vdash x : \bullet A \mid \Delta', x : \bullet A \vdash T}\ \mathsf{Conn}}{(\boldsymbol{\nu} y)\,(P \mid_x Q) \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta_2 \vdash T' \mid \Delta \vdash x : \bullet A \mid \Delta', x : \bullet A \vdash T}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash y : \bullet D \mid \Delta_2, y : \bullet D \vdash T' \mid \Delta \vdash x : A}{\dfrac{(\boldsymbol{\nu} y)\,P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash T' \mid \Delta \vdash x : A \qquad Q \triangleright \Psi' \mid \Delta', x : A \vdash T}{(\boldsymbol{\nu} y)\,P \mid_x Q \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta_2 \vdash T' \mid \Delta \vdash x : \bullet A \mid \Delta', x : \bullet A \vdash T}\ \mathsf{Conn}}\ \mathsf{Scope}$$

Case 2:

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

is equivalent to ($\equiv$)

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

$$[\mathsf{Scope/Conn/R}] \quad (\boldsymbol{\nu} y)\,(P \mid_x Q) \equiv P \mid_x (\boldsymbol{\nu} y)\,Q \qquad \Big(y \notin \mathsf{fn}(P)\Big)$$

Case 1:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta \vdash x : A \quad Q \triangleright \Psi' \mid \Delta_1 \vdash y : \bullet D \mid \Delta_2, y : \bullet D \vdash T' \mid \Delta', x : A \vdash T}{P \mid_x Q \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash y : \bullet D \mid \Delta_2, y : \bullet D \vdash T' \mid \Delta \vdash x : \bullet A \mid \Delta', x : \bullet A \vdash T}\ \mathsf{Conn}}{(\boldsymbol{\nu} y)\,(P \mid_x Q) \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta_2 \vdash T' \mid \Delta \vdash x : \bullet A \mid \Delta', x : \bullet A \vdash T}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{P \triangleright \Psi \mid \Delta \vdash x : A \qquad \dfrac{Q \triangleright \Psi' \mid \Delta_1 \vdash y : \bullet D \mid \Delta_2, y : \bullet D \vdash T' \mid \Delta', x : A \vdash T}{(\boldsymbol{\nu} y)\,Q \triangleright \Psi' \mid \Delta_1, \Delta_2 \vdash T' \mid \Delta', x : A \vdash T}\ \mathsf{Scope}}{P \mid_x (\boldsymbol{\nu} y)\,Q \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta_2 \vdash T' \mid \Delta \vdash x : \bullet A \mid \Delta', x : \bullet A \vdash T}\ \mathsf{Conn}$$

Case 2:

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

is equivalent to ($\equiv$)

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

Case 3:

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

is equivalent to ($\equiv$)

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

Commuting Scope **with** Scope

$$\boxed{[\textsf{Scope/Scope}] \quad (\boldsymbol{\nu} y)\,(\boldsymbol{\nu} x)\,P \equiv (\boldsymbol{\nu} x)\,(\boldsymbol{\nu} y)\,P}$$

Case 1:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, y : \bullet A \vdash T'' \mid \Delta_3 \vdash x : \bullet B \mid \Delta_4, x : \bullet B \vdash T'}{(\boldsymbol{\nu} x)\,P \triangleright \Psi \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, y : \bullet A \vdash T'' \mid \Delta_3, \Delta_4 \vdash T'}\,\textsf{Scope}}{(\boldsymbol{\nu} y)\,(\boldsymbol{\nu} x)\,P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash T'' \mid \Delta_3, \Delta_4 \vdash T'}\,\textsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, y : \bullet A \vdash T'' \mid \Delta_3 \vdash x : \bullet B \mid \Delta_4, x : \bullet B \vdash T'}{(\boldsymbol{\nu} y)\,P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash T'' \mid \Delta_3 \vdash x : \bullet B \mid \Delta_4, x : \bullet B \vdash T'}\,\textsf{Scope}}{(\boldsymbol{\nu} x)\,(\boldsymbol{\nu} y)\,P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash T'' \mid \Delta_3, \Delta_4 \vdash T'}\,\textsf{Scope}$$

Case 2:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \quad \Delta_1 \vdash y : \bullet A \quad \mid \quad \Delta_3, y : \bullet A \vdash x : \bullet B \quad \mid \quad \Delta_4, x : \bullet B \vdash T'}{(\boldsymbol{\nu} x)\,P \triangleright \Psi \mid \quad \Delta_1 \vdash y : \bullet A \quad \mid \quad \Delta_3, \Delta_4, y : \bullet A \vdash T'}\,\textsf{Scope}}{(\boldsymbol{\nu} y)\,(\boldsymbol{\nu} x)\,P \triangleright \Psi \mid \quad \Delta_1, \Delta_3, \Delta_4 \vdash T'}\,\textsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \quad \Delta_1 \vdash y : \bullet A \quad \mid \quad \Delta_3, y : \bullet A \vdash x : \bullet B \quad \mid \quad \Delta_4, x : \bullet B \vdash T'}{(\boldsymbol{\nu} y)\,P \triangleright \Psi \mid \quad \Delta_1, \Delta_3 \vdash x : \bullet B \quad \mid \quad \Delta_4, x : \bullet B \vdash T'}\,\textsf{Scope}}{(\boldsymbol{\nu} x)\,(\boldsymbol{\nu} y)\,P \triangleright \Psi \mid \quad \Delta_1, \Delta_3, \Delta_4 \vdash T'}\,\textsf{Scope}$$

Case 3:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash y : \bullet A \mid \Delta_3 \vdash x : \bullet B \mid \Delta_4, y : \bullet A, x : \bullet B \vdash T'}{(\boldsymbol{\nu} x)\ P \triangleright \Psi \mid \Delta_1 \vdash y : \bullet A \mid \Delta_3, \Delta_4, y : \bullet A \vdash T'}\ \mathsf{Scope}}{(\boldsymbol{\nu} y)\ (\boldsymbol{\nu} x)\ P \triangleright \Psi \mid \Delta_1, \Delta_3, \Delta_4 \vdash T'}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash y : \bullet A \mid \Delta_3 \vdash x : \bullet B \mid \Delta_4, y : \bullet A, x : \bullet B \vdash T'}{(\boldsymbol{\nu} y)\ P \triangleright \Psi \mid \Delta_3 \vdash x : \bullet B \mid \Delta_1, \Delta_4, x : \bullet B \vdash T'}\ \mathsf{Scope}}{(\boldsymbol{\nu} x)\ (\boldsymbol{\nu} y)\ P \triangleright \Psi \mid \Delta_1, \Delta_3, \Delta_4 \vdash T'}\ \mathsf{Scope}$$

Commuting Scope with 1L (no case for 1R)

$$\boxed{[\mathsf{Scope}/\mathsf{1L}] \quad (\boldsymbol{\nu} x)\ \mathsf{wait}[y]; P \equiv \mathsf{wait}[y]; (\boldsymbol{\nu} x)\ P}$$

Case 1:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash x : \bullet C \mid \Delta_2, x : \bullet C \vdash T \mid \Delta \vdash T'}{\mathsf{wait}[y];\ P \triangleright \Psi \mid \Delta_1 \vdash x : \bullet C \mid \Delta_2, x : \bullet C \vdash T \mid \Delta, y : \mathbf{1} \vdash T'}\ \mathsf{1L}}{(\boldsymbol{\nu} x)\ \mathsf{wait}[y];\ P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash T \mid \Delta, y : \mathbf{1} \vdash T'}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash x : \bullet C \mid \Delta_2, x : \bullet C \vdash T \mid \Delta \vdash T'}{(\boldsymbol{\nu} x)\ P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash T \mid \Delta \vdash T'}\ \mathsf{Scope}}{\mathsf{wait}[y];\ (\boldsymbol{\nu} x)\ P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash T \mid \Delta, y : \mathbf{1} \vdash T'}\ \mathsf{1L}$$

Case 2:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash x : \bullet C \mid \Delta, x : \bullet C \vdash T'}{\mathsf{wait}[y];\ P \triangleright \Psi \mid \Delta_1 \vdash x : \bullet C \mid \Delta, x : \bullet C, y : \mathbf{1} \vdash T'}\ \mathsf{1L}}{(\boldsymbol{\nu} x)\ \mathsf{wait}[y];\ P \triangleright \Psi \mid \Delta_1, \Delta, y : \mathbf{1} \vdash T'}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash x : \bullet C \mid \Delta, x : \bullet C \vdash T'}{(\boldsymbol{\nu} x)\ P \triangleright \Psi \mid \Delta_1, \Delta \vdash T'}\ \mathsf{Scope}}{\mathsf{wait}[y];\ (\boldsymbol{\nu} x)\ P \triangleright \Psi \mid \Delta_1, \Delta, y : \mathbf{1} \vdash T'}\ \mathsf{1L}$$

Case 3:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta \vdash x : \bullet C \mid \Delta_2, x : \bullet C \vdash T}{\mathsf{wait}[y];\ P \triangleright \Psi \mid \Delta, y : \mathbf{1} \vdash x : \bullet C \mid \Delta_2, x : \bullet C \vdash T}\ \mathsf{1L}}{(\boldsymbol{\nu} x)\ \mathsf{wait}[y];\ P \triangleright \Psi \mid \Delta_2, \Delta, y : \mathbf{1} \vdash T}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta \vdash x : \bullet C \mid \Delta_2, x : \bullet C \vdash T}{(\boldsymbol{\nu} x)\ P \triangleright \Psi \mid \Delta_2, \Delta \vdash T}\ \mathsf{Scope}}{\mathsf{wait}[y];\ (\boldsymbol{\nu} x)\ P \triangleright \Psi \mid \Delta_2, \Delta, y : \mathbf{1} \vdash T}\ \mathsf{1L}$$

Commuting Scope with ⊗R and ⊗L

$$\boxed{[\mathsf{Scope}/\otimes\mathsf{R}/\mathsf{L}] \quad (\boldsymbol{\nu} w)\,\overline{x}(y);(P \mid Q) \equiv \overline{x}(y);((\boldsymbol{\nu} w)\,P \mid Q) \qquad \Big(w \notin \mathsf{fn}(Q)\Big)}$$

Case 1:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T \mid \Delta \vdash y : A \qquad Q \triangleright \Psi' \mid \Delta' \vdash x : B}{\overline{x}(y);(P \mid Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T \mid \Delta, \Delta' \vdash x : A \otimes B}\otimes\mathsf{R}}{(\boldsymbol{\nu} w)\,\overline{x}(y);(P \mid Q) \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta_2 \vdash T \mid \Delta, \Delta' \vdash x : A \otimes B}\mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T \mid \Delta \vdash y : A}{(\boldsymbol{\nu} w)\,P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash T \mid \Delta \vdash y : A}\mathsf{Scope} \qquad Q \triangleright \Psi' \mid \Delta' \vdash x : B}{\overline{x}(y);((\boldsymbol{\nu} w)\,P \mid Q) \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta_2 \vdash T \mid \Delta, \Delta' \vdash x : A \otimes B}\otimes\mathsf{R}$$

Case 2:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta, w : \bullet C \vdash y : A \qquad Q \triangleright \Psi' \mid \Delta' \vdash x : B}{\overline{x}(y);(P \mid Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash w : \bullet C \mid \Delta, \Delta', w : \bullet C \vdash x : A \otimes B}\otimes\mathsf{R}}{(\boldsymbol{\nu} w)\,\overline{x}(y);(P \mid Q) \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta, \Delta' \vdash x : A \otimes B}\mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta, w : \bullet C \vdash y : A}{(\boldsymbol{\nu} w)\,P \triangleright \Psi \mid \Delta_1, \Delta \vdash y : A}\mathsf{Scope} \qquad Q \triangleright \Psi' \mid \Delta' \vdash x : B}{\overline{x}(y);((\boldsymbol{\nu} w)\,P \mid Q) \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta, \Delta' \vdash x : A \otimes B}\otimes\mathsf{R}$$

$$\boxed{[\mathsf{Scope}/\otimes\mathsf{R}/\mathsf{R}] \quad (\boldsymbol{\nu} w)\,\overline{x}(y);(P \mid Q) \equiv \overline{x}(y);(P \mid (\boldsymbol{\nu} w)\,Q) \qquad \Big(w \notin \mathsf{fn}(P)\Big)}$$

Case 1:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta \vdash y : A \qquad Q \triangleright \Psi' \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T \mid \Delta' \vdash x : B}{\overline{x}(y);(P \mid Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T \mid \Delta, \Delta' \vdash x : A \otimes B}\otimes\mathsf{R}}{(\boldsymbol{\nu} w)\,\overline{x}(y);(P \mid Q) \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta_2 \vdash T \mid \Delta, \Delta' \vdash x : A \otimes B}\mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{P \triangleright \Psi \mid \Delta \vdash y : A \qquad \dfrac{Q \triangleright \Psi' \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T \mid \Delta' \vdash x : B}{(\boldsymbol{\nu} w)\,Q \triangleright \Psi' \mid \Delta_1, \Delta_2 \vdash T \mid \Delta' \vdash x : B}\mathsf{Scope}}{\overline{x}(y);(P \mid (\boldsymbol{\nu} w)\,Q) \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta_2 \vdash A \mid \Delta, \Delta' \vdash x : A \otimes B}\otimes\mathsf{R}$$

Case 2:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta \vdash y : A \qquad Q \triangleright \Psi' \mid \Delta_1 \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash x : B}{\overline{x}(y); (P \mid Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash w : \bullet C \mid \Delta, \Delta', w : \bullet C \vdash x : A \otimes B}\otimes\mathsf{R}}{(\boldsymbol{\nu} w)\, \overline{x}(y); (P \mid Q) \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta, \Delta' \vdash x : A \otimes B}\,\mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{P \triangleright \Psi \mid \Delta \vdash y : A \qquad \dfrac{Q \triangleright \Psi' \mid \Delta_1 \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash x : B}{(\boldsymbol{\nu} w)\, Q \triangleright \Psi' \mid \Delta_1, \Delta' \vdash x : B}\,\mathsf{Scope}}{\overline{x}(y); (P \mid (\boldsymbol{\nu} w)\, Q) \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta, \Delta' \vdash x : A \otimes B}\otimes\mathsf{R}$$

$$\boxed{[\mathsf{Scope}/\otimes\mathsf{L}] \quad (\boldsymbol{\nu} w)\, x(y); P \equiv x(y); (\boldsymbol{\nu} w)\, P}$$

Case 1:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet D \mid \Delta_2, w : \bullet D \vdash T \mid \Delta, y : A,\ x : B \vdash T''}{x(y); P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet D \mid \Delta_2, w : \bullet D \vdash T \mid \Delta, x : A \otimes B \vdash T''}\otimes L}{(\boldsymbol{\nu} w)\, x(y); P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash T \mid \Delta, x : A \otimes B \vdash T''}\,\mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet D \mid \Delta_2, w : \bullet D \vdash T \mid \Delta, y : A,\ x : B \vdash T''}{(\boldsymbol{\nu} w)\, P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash T \mid \Delta, y : A,\ x : B \vdash T''}\,\mathsf{Scope}}{x(y); (\boldsymbol{\nu} w)\, P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash T \mid \Delta, x : A \otimes B \vdash T''}\otimes L$$

Case 2:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet D \mid \Delta, w : \bullet D, y : A,\ x : B \vdash T''}{x(y); P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet D \mid \Delta, w : \bullet D, x : A \otimes B \vdash T''}\otimes L}{(\boldsymbol{\nu} w)\, x(y); P \triangleright \Psi \mid \Delta_1, \Delta, x : A \otimes B \vdash T''}\,\mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet D \mid \Delta, w : \bullet D, y : A, x : B \vdash T''}{(\boldsymbol{\nu} w)\, P \triangleright \Psi \mid \Delta_1, \Delta, y : A,\ x : B \vdash T''}\,\mathsf{Scope}}{x(y); (\boldsymbol{\nu} w)\, P \triangleright \Psi \mid \Delta_1, \Delta, x : A \otimes B \vdash T''}\otimes L$$

Commuting Scope with $\multimap$ R and $\multimap$ L

$$\boxed{[\mathsf{Scope}/\multimap\mathsf{R}] \quad (\boldsymbol{\nu} w)\, x(y); P \equiv x(y); (\boldsymbol{\nu} w)\, P}$$

Case 1:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T \mid \Delta, y : A \vdash x : B}{x(y); P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T \mid \Delta \vdash x : A \multimap B}\ \multimap R}{(\boldsymbol{\nu} w)\, x(y); P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash T \mid \Delta \vdash x : A \multimap B}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T \mid \Delta, y : A \vdash x : B}{(\boldsymbol{\nu} w)\, P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash T \mid \Delta, y : A \vdash x : B}\ \mathsf{Scope}}{x(y); (\boldsymbol{\nu} w)\, P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash T \mid \Delta \vdash x : A \multimap B}\ \multimap R$$

Case 2:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta, w : \bullet C, y : A \vdash x : B}{x(y); P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta, w : \bullet C \vdash x : A \multimap B}\ \multimap R}{(\boldsymbol{\nu} w)\, x(y); P \triangleright \Psi \mid \Delta_1, \Delta \vdash x : A \multimap B}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta, w : \bullet C, y : A \vdash x : B}{(\boldsymbol{\nu} w)\, P \triangleright \Psi \mid \Delta_1, \Delta, y : A \vdash x : B}\ \mathsf{Scope}}{x(y)(\boldsymbol{\nu} w)\, P \triangleright \Psi \mid \Delta_1, \Delta \vdash x : A \multimap B}\ \multimap R$$

$$[\mathsf{Scope}/\multimap \mathsf{L}/\mathsf{L}] \quad (\boldsymbol{\nu} w)\, \overline{x}(y); (P \mid Q) \equiv \overline{x}(y); ((\boldsymbol{\nu} w)\, P \mid Q) \qquad \Big(w \notin \mathsf{fn}(Q)\Big)$$

Case 1:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet D \mid \Delta_2, w : \bullet D \vdash T \mid \Delta \vdash y : A \qquad Q \triangleright \Psi' \mid \Delta', x : B \vdash T''}{\overline{x}(y); (P \mid Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash w : \bullet D \mid \Delta_2, w : \bullet D \vdash T \mid \Delta, \Delta', x : A \multimap B \vdash T''}\ \mathsf{Scope}}{(\boldsymbol{\nu} w)\, \overline{x}(y); (P \mid Q) \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta_2 \vdash T \mid \Delta, \Delta', x : A \multimap B \vdash T''}\ \multimap L$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet D \mid \Delta_2, w : \bullet D \vdash T \mid \Delta \vdash y : A}{(\boldsymbol{\nu} w)\, P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash T \mid \Delta \vdash y : A}\ \mathsf{Scope} \qquad Q \triangleright \Psi' \mid \Delta', x : B \vdash T''}{\overline{x}(y); ((\boldsymbol{\nu} w)\, P \mid Q) \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta_2 \vdash T \mid \Delta, \Delta', x : A \multimap B \vdash T''}\ \multimap L$$

Case 2:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet D \mid \Delta, w : \bullet D \vdash y : A \qquad Q \triangleright \Psi' \mid \Delta', x : B \vdash T''}{\overline{x}(y); (P \mid Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash w : \bullet D \mid \Delta, \Delta', w : \bullet D, x : A \multimap B \vdash T''}\ \multimap L}{(\boldsymbol{\nu} w)\, \overline{x}(y); (P \mid Q) \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta, \Delta', x : A \multimap B \vdash T''}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet D \mid \Delta, w : \bullet D \vdash y : A}{(\boldsymbol{\nu} w)\, P \triangleright \Psi \mid \Delta_1, \Delta \vdash y : A}\ \mathsf{Scope} \qquad Q \triangleright \Psi' \mid \Delta', x : B \vdash T''}{\overline{x}(y); ((\boldsymbol{\nu} w)\, P \mid Q) \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta, \Delta', x : A \multimap B \vdash T''}\ \multimap L$$

$$\boxed{[\mathsf{Scope}/ \multimap \mathsf{L/R}] \quad (\boldsymbol{\nu} w)\, \overline{x}(y); (P \mid Q) \equiv \overline{x}(y); (P \mid (\boldsymbol{\nu} w)\, Q) \qquad \Big(w \notin \mathsf{fn}(P)\Big)}$$

Case 1:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta \vdash y : A \qquad Q \triangleright \Psi' \mid \Delta_1 \vdash w : \bullet D \mid \Delta_2, w : \bullet D \vdash T \mid \Delta', x : B \vdash T''}{\overline{x}(y); (P \mid Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash w : \bullet D \mid \Delta_2, w : \bullet D \vdash T \mid \Delta, \Delta', x : A \multimap B \vdash T''}\ \multimap L}{(\boldsymbol{\nu} w)\, \overline{x}(y); (P \mid Q) \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta_2 \vdash T \mid \Delta, \Delta', x : A \multimap B \vdash T''}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{P \triangleright \Psi \mid \Delta \vdash y : A \qquad \dfrac{Q \triangleright \Psi' \mid \Delta_1 \vdash w : \bullet D \mid \Delta_2, w : \bullet D \vdash T \mid \Delta', x : B \vdash T''}{(\boldsymbol{\nu} w)\, Q \triangleright \Psi' \mid \Delta_1, \Delta_2 \vdash T \mid \Delta', x : B \vdash T''}\ \mathsf{Scope}}{\overline{x}(y); (P \mid (\boldsymbol{\nu} w)\, Q) \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta_2 \vdash T \mid \Delta, \Delta', x : A \multimap B \vdash T''}\ \multimap L$$

Case 2:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta \vdash y : A \qquad Q \triangleright \Psi' \mid \Delta_1 \vdash w : \bullet D \mid \Delta', w : \bullet D, x : B \vdash T''}{\overline{x}(y); (P \mid Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash w : \bullet D \mid \Delta, \Delta', w : \bullet D, x : A \multimap B \vdash T''}\ \multimap L}{(\boldsymbol{\nu} w)\, \overline{x}(y); (P \mid Q) \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta, \Delta', x : A \multimap B \vdash T''}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{P \triangleright \Psi \mid \Delta \vdash y : A \qquad \dfrac{Q \triangleright \Psi' \mid \Delta_1 \vdash w : \bullet D \mid \Delta', w : \bullet D, x : B \vdash T''}{(\boldsymbol{\nu} w)\, Q \triangleright \Psi' \mid \Delta_1, \Delta', x : B \vdash T''}\ \mathsf{Scope}}{\overline{x}(y); (P \mid (\boldsymbol{\nu} w)\, Q) \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta, \Delta', x : A \multimap B \vdash T''}\ \multimap L$$

Case 3:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta \vdash y : A \qquad Q \triangleright \Psi' \mid \Delta_1, x : B \vdash w : \bullet D \mid \Delta', w : \bullet D \vdash T''}{\overline{x}(y); (P \mid Q) \triangleright \Psi \mid \Psi' \mid \Delta_1, x : A \multimap B \vdash w : \bullet D \mid \Delta, \Delta', w : \bullet D \vdash T''}\ \multimap L}{(\boldsymbol{\nu} w)\, \overline{x}(y); (P \mid Q) \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta, \Delta', x : A \multimap B \vdash T''}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{P \triangleright \Psi \mid \Delta \vdash y : A \qquad \dfrac{Q \triangleright \Psi' \mid \Delta_1, x : B \vdash w : \bullet D \mid \Delta', w : \bullet D \vdash T''}{(\boldsymbol{\nu} w)\, Q \triangleright \Psi' \mid \Delta_1, \Delta', x : B \vdash T''}\ \mathsf{Scope}}{\overline{x}(y); (P \mid (\boldsymbol{\nu} w)\, Q) \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta, \Delta', x : A \multimap B \vdash T''}\ \multimap L$$

Commuting Scope with $\oplus$R and $\oplus$L

$$\boxed{[\mathsf{Scope}/\oplus \mathsf{R}_1] \quad (\boldsymbol{\nu} w)\ x.\mathsf{inl}; P \equiv x.\mathsf{inl}; (\boldsymbol{\nu} w)\ P}$$

Case 1:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T \mid \Delta \vdash x : A}{x.\mathsf{inl}; P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T \mid \Delta \vdash x : A \oplus B}\ \oplus\mathsf{R}_1}{(\boldsymbol{\nu} w)\ x.\mathsf{inl}; P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash T \mid \Delta \vdash x : A \oplus B}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T \mid \Delta \vdash x : A}{(\boldsymbol{\nu} w)\ P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash T \mid \Delta \vdash x : A}\ \mathsf{Scope}}{x.\mathsf{inl}; (\boldsymbol{\nu} w)\ P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash T \mid \Delta \vdash x : A \oplus B}\ \oplus\mathsf{R}_1$$

Case 2:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash x : A}{x.\mathsf{inl}; P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash x : A \oplus B}\ \oplus\mathsf{R}_1}{(\boldsymbol{\nu} w)\ x.\mathsf{inl}; P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash x : A \oplus B}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash x : A}{(\boldsymbol{\nu} w)\ P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash x : A}\ \mathsf{Scope}}{x.\mathsf{inl}; (\boldsymbol{\nu} w)\ P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash x : A \oplus B}\ \oplus\mathsf{R}_1$$

$$\boxed{[\mathsf{Scope}/\oplus \mathsf{R}_2] \quad (\boldsymbol{\nu} w)\ x.\mathsf{inr}; P \equiv x.\mathsf{inr}; (\boldsymbol{\nu} w)\ P}$$

Case 1:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T \mid \Delta \vdash x : B}{x.\mathsf{inr}; P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T \mid \Delta \vdash x : A \oplus B}\ \oplus\mathsf{R}_2}{(\boldsymbol{\nu} w)\ x.\mathsf{inr}; P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash T \mid \Delta \vdash x : A \oplus B}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T \mid \Delta \vdash x : B}{(\boldsymbol{\nu} w)\ P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash T \mid \Delta \vdash x : B}\ \mathsf{Scope}}{x.\mathsf{inr}; (\boldsymbol{\nu} w)\ P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash T \mid \Delta \vdash x : A \oplus B}\ \oplus\mathsf{R}_2$$

Case 2:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash x : B}{x.\mathsf{inr}; P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash x : A \oplus B}\ \oplus\mathsf{R}_2}{(\boldsymbol{\nu} w)\ x.\mathsf{inr}; P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash x : A \oplus B}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash x : B}{(\boldsymbol{\nu} w)\ P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash x : B}\ \mathsf{Scope}}{x.\mathsf{inr}; (\boldsymbol{\nu} w)\ P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash x : A \oplus B}\ \oplus\mathsf{R}_2$$

$$\boxed{[\mathsf{Scope}/\oplus\mathsf{L}] \quad (\boldsymbol{\nu} w)\ x.\mathsf{case}(P,Q) \equiv x.\mathsf{case}((\boldsymbol{\nu} w)\ P\ ,\ (\boldsymbol{\nu} w)\ Q)}$$

Case 1:

$$\dfrac{\dfrac{P \triangleright\ \Psi \mid \Delta_1 \vdash w : \bullet D \mid \Delta_2, w : \bullet D \vdash T \mid \Delta, x : A \vdash T' \qquad Q \triangleright\ \Psi \mid \Delta_1 \vdash w : \bullet D \mid \Delta_2, w : \bullet D \vdash T \mid \Delta, x : B \vdash T'}{x.\mathsf{case}(P,Q) \triangleright\ \Psi \mid \Delta_1 \vdash w : \bullet D \mid \Delta_2, w : \bullet D \vdash T \mid \Delta, x : A \oplus B \vdash T'}\ \oplus\mathsf{L}}{(\boldsymbol{\nu} w)\ x.\mathsf{case}(P,Q) \triangleright\ \Psi \mid \Delta_1, \Delta_2 \vdash T \mid \Delta, x : A \oplus B \vdash T'}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright\ \Psi \mid \Delta_1 \vdash w : \bullet D \mid \Delta_2, w : \bullet D \vdash T \mid \Delta, x : A \vdash T'}{(\boldsymbol{\nu} w)\ P \triangleright\ \Psi \mid \Delta_1, \Delta_2 \vdash T \mid \Delta, x : A \vdash T'}\ \mathsf{Scope} \qquad \dfrac{Q \triangleright\ \Psi \mid \Delta_1 \vdash w : \bullet D \mid \Delta_2, w : \bullet D \vdash T \mid \Delta, x : B \vdash T'}{(\boldsymbol{\nu} w)\ Q \triangleright\ \Psi \mid \Delta_1, \Delta_2 \vdash T \mid \Delta, x : B \vdash T'}\ \mathsf{Scope}}{x.\mathsf{case}((\boldsymbol{\nu} w)\ P\ ,\ (\boldsymbol{\nu} w)\ Q) \triangleright\ \Psi \mid \Psi' \mid \Delta_1, \Delta_2 \vdash T \mid \Delta, \Delta', x : A \oplus B \vdash T''}\ \oplus\mathsf{L}$$

Case 2:

$$\dfrac{\dfrac{P \triangleright\ \Psi \mid \Delta_1, x : A \vdash w : \bullet D \mid \Delta_2, w : \bullet D \vdash T \qquad Q \triangleright\ \Psi \mid \Delta_1, x : B \vdash w : \bullet D \mid \Delta_2, w : \bullet D \vdash T}{x.\mathsf{case}(P,Q) \triangleright\ \Psi \mid \Delta_1, x : A \oplus B \vdash w : \bullet D \mid \Delta_2, w : \bullet D \vdash T}\ \oplus\mathsf{L}}{(\boldsymbol{\nu} w)\ x.\mathsf{case}(P,Q) \triangleright\ \Psi \mid \Delta_1, \Delta_2, x : A \oplus B \vdash T}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright\ \Psi \mid \Delta_1, x : A \vdash w : \bullet D \mid \Delta_2, w : \bullet D \vdash T}{(\boldsymbol{\nu} w)\ P \triangleright\ \Psi \mid \Delta_1, \Delta_2, x : A \vdash T}\ \mathsf{Scope} \qquad \dfrac{Q \triangleright\ \Psi \mid \Delta_1, x : B \vdash w : \bullet D \mid \Delta_2, w : \bullet D \vdash T}{(\boldsymbol{\nu} w)\ Q \triangleright\ \Psi \mid \Delta_1, \Delta_2, x : B \vdash T}\ \mathsf{Scope}}{x.\mathsf{case}((\boldsymbol{\nu} w)\ P\ ,\ (\boldsymbol{\nu} w)\ Q) \triangleright\ \Psi \mid \Psi' \mid \Delta_1, \Delta_2, x : A \oplus B \vdash T}\ \oplus\mathsf{L}$$

Case 3:

$$\dfrac{\dfrac{P \triangleright\ \Psi \mid \Delta_1 \vdash w : \bullet D \mid \Delta_2, x : A, w : \bullet D \vdash T \qquad Q \triangleright\ \Psi \mid \Delta_1 \vdash w : \bullet D \mid \Delta_2, x : B, w : \bullet D \vdash T}{x.\mathsf{case}(P,Q) \triangleright\ \Psi \mid \Delta_1 \vdash w : \bullet D \mid \Delta_2, x : A \oplus B, w : \bullet D \vdash T}\ \oplus\mathsf{L}}{(\boldsymbol{\nu} w)\ x.\mathsf{case}(P,Q) \triangleright\ \Psi \mid \Delta_1, \Delta_2, x : A \oplus B \vdash T}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright\ \Psi \mid \Delta_1 \vdash w : \bullet D \mid \Delta_2, x : A, w : \bullet D \vdash T}{(\boldsymbol{\nu} w)\ P \triangleright\ \Psi \mid \Delta_1, \Delta_2, x : A \vdash T}\ \mathsf{Scope} \qquad \dfrac{Q \triangleright\ \Psi \mid \Delta_1 \vdash w : \bullet D \mid \Delta_2, x : B, w : \bullet D \vdash T}{(\boldsymbol{\nu} w)\ Q \triangleright\ \Psi \mid \Delta_1, \Delta_2, x : B \vdash T}\ \mathsf{Scope}}{x.\mathsf{case}((\boldsymbol{\nu} w)\ P\ ,\ (\boldsymbol{\nu} w)\ Q) \triangleright\ \Psi \mid \Psi' \mid \Delta_1, \Delta_2, x : A \oplus B \vdash T}\ \oplus\mathsf{L}$$

Commuting Scope with &R and &L

$$\boxed{[\mathsf{Scope}/\&\mathsf{R}] \quad (\boldsymbol{\nu} w)\ x.\mathsf{case}(P,Q) \equiv x.\mathsf{case}((\boldsymbol{\nu} w)\ P, (\boldsymbol{\nu} w)\ Q)}$$

Case 1:

$$\dfrac{\dfrac{P \triangleright\ \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T \mid \Delta \vdash x : A \quad Q \triangleright\ \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T \mid \Delta \vdash x : B}{x.\mathsf{case}(P,Q) \triangleright\ \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T \mid \Delta \vdash x : A\&B}\ \&\mathsf{R}}{(\boldsymbol{\nu} w)\ x.\mathsf{case}(P,Q) \triangleright\ \Psi \mid \Delta_1, \Delta_2 \vdash T \mid \Delta \vdash x : A\&B}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright\ \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T \mid \Delta \vdash x : A}{(\boldsymbol{\nu} w)\ P \triangleright\ \Psi \mid \Delta_1, \Delta_2 \vdash T \mid \Delta \vdash x : A}\ \mathsf{Scope} \quad \dfrac{Q \triangleright\ \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T \mid \Delta \vdash x : B}{(\boldsymbol{\nu} w)\ Q \triangleright\ \Psi \mid \Delta_1, \Delta_2 \vdash T \mid \Delta \vdash x : B}\ \mathsf{Scope}}{x.\mathsf{case}((\boldsymbol{\nu} w)\ P, (\boldsymbol{\nu} w)\ Q) \triangleright\ \Psi \mid \Delta_1, \Delta_2 \vdash T \mid \Delta \vdash x : A\&B}\ \&\mathsf{R}$$

Case 2:

$$\dfrac{\dfrac{P \triangleright\ \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash x : A \quad Q \triangleright\ \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash x : B}{x.\mathsf{case}(P,Q) \triangleright\ \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash x : A\&B}\ \&\mathsf{R}}{(\boldsymbol{\nu} w)\ x.\mathsf{case}(P,Q) \triangleright\ \Psi \mid \Delta_1, \Delta_2 \vdash x : A\&B}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright\ \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash x : A}{(\boldsymbol{\nu} w)\ P \triangleright\ \Psi \mid \Delta_1, \Delta_2 \vdash x : A}\ \mathsf{Scope} \quad \dfrac{Q \triangleright\ \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash x : B}{(\boldsymbol{\nu} w)\ Q \triangleright\ \Psi \mid \Delta_1, \Delta_2 \vdash x : B}\ \mathsf{Scope}}{x.\mathsf{case}((\boldsymbol{\nu} w)\ P, (\boldsymbol{\nu} w)\ Q) \triangleright\ \Psi \mid \Delta_1, \Delta_2 \vdash x : A\&B}\ \&\mathsf{R}$$

$$\boxed{[\mathsf{Scope}/\&\mathsf{L}_1] \quad (\boldsymbol{\nu} w)\ x.\mathsf{inl}; P \equiv x.\mathsf{inl}; (\boldsymbol{\nu} w)\ P}$$

Case 1:

$$\dfrac{\dfrac{P \triangleright\ \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T \mid \Delta, x : A \vdash T'}{x.\mathsf{inl}; P \triangleright\ \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T \mid \Delta, x : A\&B \vdash T'}\ \&\mathsf{L}_1}{(\boldsymbol{\nu} w)\ x.\mathsf{inl}; P \triangleright\ \Psi \mid \Delta_1, \Delta_2 \vdash T \mid \Delta, x : A\&B \vdash T'}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright\ \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T \mid \Delta, x : A \vdash T'}{(\boldsymbol{\nu} w)\ P \triangleright\ \Psi \mid \Delta_1, \Delta_2 \vdash T \mid \Delta, x : A \vdash T'}\ \mathsf{Scope}}{x.\mathsf{inl}; (\boldsymbol{\nu} w)\ P \triangleright\ \Psi \mid \Delta_1, \Delta_2 \vdash T \mid \Delta, x : A\&B \vdash T'}\ \&\mathsf{L}_1$$

Case 2:

$$\dfrac{\dfrac{P \triangleright\ \Psi \mid \Delta_1, x : A \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T}{x.\mathsf{inl}; P \triangleright\ \Psi \mid \Delta_1, x : A\&B \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T}\ \&\mathsf{L}_1}{(\boldsymbol{\nu} w)\ x.\mathsf{inl}; P \triangleright\ \Psi \mid \Delta_1, \Delta_2, x : A\&B \vdash T}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1, x : A \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T}{(\boldsymbol{\nu} w)\, P \triangleright \Psi \mid \Delta_1, \Delta_2, x : A \vdash T}\ \mathsf{Scope}}{x.\mathsf{inl};\, (\boldsymbol{\nu} w)\, P \triangleright \Psi \mid \Delta_1, \Delta_2, x : A \& B \vdash T}\ \&\mathsf{L}_1$$

Case 3:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, x : A, w : \bullet C \vdash T}{x.\mathsf{inl};\, P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, x : A \& B, w : \bullet C \vdash T}\ \&\mathsf{L}_1}{(\boldsymbol{\nu} w)\, x.\mathsf{inl};\, P \triangleright \Psi \mid \Delta_1, \Delta_2, x : A \& B \vdash T}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, x : A, w : \bullet C \vdash T}{(\boldsymbol{\nu} w)\, P \triangleright \Psi \mid \Delta_1, \Delta_2, x : A \vdash T}\ \mathsf{Scope}}{x.\mathsf{inl};\, (\boldsymbol{\nu} w)\, P \triangleright \Psi \mid \Delta_1, \Delta_2, x : A \& B \vdash T}\ \&\mathsf{L}_1$$

$$\boxed{[\mathsf{Scope}/\&\mathsf{L}_2] \quad (\boldsymbol{\nu} w)\, x.\mathsf{inr};\, P \equiv x.\mathsf{inr};\, (\boldsymbol{\nu} w)\, P}$$

Case 1:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T \mid \Delta, x : B \vdash T'}{x.\mathsf{inr};\, P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T \mid \Delta, x : A \& B \vdash T'}\ \&\mathsf{L}_2}{(\boldsymbol{\nu} w)\, x.\mathsf{inr};\, P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash T \mid \Delta, x : A \& B \vdash T'}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T \mid \Delta, x : B \vdash T'}{(\boldsymbol{\nu} w)\, P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash T \mid \Delta, x : B \vdash T'}\ \mathsf{Scope}}{x.\mathsf{inr};\, (\boldsymbol{\nu} w)\, P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash T \mid \Delta, x : A \& B \vdash T'}\ \&\mathsf{L}_2$$

Case 2:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1, x : B \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T}{x.\mathsf{inr};\, P \triangleright \Psi \mid \Delta_1, x : A \& B \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T}\ \&\mathsf{L}_2}{(\boldsymbol{\nu} w)\, x.\mathsf{inr};\, P \triangleright \Psi \mid \Delta_1, \Delta_2, x : A \& B \vdash T}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1, x : B \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T}{(\boldsymbol{\nu} w)\, P \triangleright \Psi \mid \Delta_1, \Delta_2, x : B \vdash T}\ \mathsf{Scope}}{x.\mathsf{inr};\, (\boldsymbol{\nu} w)\, P \triangleright \Psi \mid \Delta_1, \Delta_2, x : A \& B \vdash T}\ \&\mathsf{L}_2$$

Case 3:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, x : B, w : \bullet C \vdash T}{x.\mathsf{inr};\, P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, x : A \& B, w : \bullet C \vdash T}\ \&\mathsf{L}_2}{(\boldsymbol{\nu} w)\, x.\mathsf{inr};\, P \triangleright \Psi \mid \Delta_1, \Delta_2, x : A \& B \vdash T}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, x : B, w : \bullet C \vdash T}{(\boldsymbol{\nu} w)\, P \triangleright \Psi \mid \Delta_1, \Delta_2, x : B \vdash T}\ \mathsf{Scope}}{x.\mathsf{inr};\, (\boldsymbol{\nu} w)\, P \triangleright \Psi \mid \Delta_1, \Delta_2, x : A \& B \vdash T}\ \&\mathsf{L}_2$$

Commuting Scope with 1C

$$[\mathsf{Scope}/1\mathsf{C}] \quad (\boldsymbol{\nu} w)\ \overrightarrow{\mathsf{close}}[x]; P \equiv \overrightarrow{\mathsf{close}}[x]; (\boldsymbol{\nu} w)\ P$$

Case 1:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T \mid \Delta \vdash T'}{\overrightarrow{\mathsf{close}}[x]; P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T \mid \cdot \vdash x : \bullet \mathbf{1} \mid \Delta, x : \bullet \mathbf{1} \vdash T'}\,1\mathsf{C}}{(\boldsymbol{\nu} w)\ \overrightarrow{\mathsf{close}}[x]; P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash T \mid \cdot \vdash x : \bullet \mathbf{1} \mid \Delta, x : \bullet \mathbf{1} \vdash T'}\,\mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T \mid \Delta \vdash T'}{(\boldsymbol{\nu} w)\ P \triangleright \Psi \mid \Delta_1 \Delta_2 \vdash T \mid \Delta \vdash T'}\,\mathsf{Scope}}{\overrightarrow{\mathsf{close}}[x]; (\boldsymbol{\nu} w)\ P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash T \mid \cdot \vdash x : \bullet \mathbf{1} \mid \Delta, x : \bullet \mathbf{1} \vdash T'}\,1\mathsf{C}$$

Case 2:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T}{\overrightarrow{\mathsf{close}}[x]; P \triangleright \Psi \mid \cdot \vdash x : \bullet \mathbf{1} \mid \Delta_1, x : \bullet \mathbf{1} \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T \mid}\,1\mathsf{C}}{(\boldsymbol{\nu} w)\ \overrightarrow{\mathsf{close}}[x]; P \triangleright \Psi \mid \cdot \vdash x : \bullet \mathbf{1} \mid \Delta_1, \Delta_2, x : \bullet \mathbf{1} \vdash T}\,\mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T}{(\boldsymbol{\nu} w)\ P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash T}\,\mathsf{Scope}}{\overrightarrow{\mathsf{close}}[x]; (\boldsymbol{\nu} w)\ P \triangleright \Psi \mid \cdot \vdash x : \bullet \mathbf{1} \mid \Delta_1, \Delta_2, x : \bullet \mathbf{1} \vdash T}\,1\mathsf{C}$$

Case 3:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T}{\overrightarrow{\mathsf{close}}[x]; P \triangleright \Psi \mid \cdot \vdash x : \bullet \mathbf{1} \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, x : \bullet \mathbf{1}, w : \bullet C \vdash T \mid}\,1\mathsf{C}}{(\boldsymbol{\nu} w)\ \overrightarrow{\mathsf{close}}[x]; P \triangleright \Psi \mid \cdot \vdash x : \bullet \mathbf{1} \mid \Delta_1, \Delta_2, x : \bullet \mathbf{1} \vdash T}\,\mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T}{(\boldsymbol{\nu} w)\ P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash T}\,\mathsf{Scope}}{\overrightarrow{\mathsf{close}}[x]; (\boldsymbol{\nu} w)\ P \triangleright \Psi \mid \cdot \vdash x : \bullet \mathbf{1} \mid \Delta_1, \Delta_2, x : \bullet \mathbf{1} \vdash T}\,1\mathsf{C}$$

Commuting Scope with $\otimes$C

$$[\mathsf{Scope}/\otimes\mathsf{C}] \quad (\boldsymbol{\nu} w)\ \overrightarrow{x(y)}; P \equiv \overrightarrow{x(y)}; (\boldsymbol{\nu} w)\ P$$

Case 1:

$$\dfrac{\dfrac{P \triangleright\ \Psi \mid \Delta_4 \vdash w : \bullet D \mid \Delta_5, w : \bullet D \vdash T \mid \Delta_1 \vdash x : \bullet B \mid \Delta_2 \vdash y : \bullet A \mid \Delta_3, y : \bullet A, x : \bullet B \vdash T''}{\overrightarrow{x(y)}; P \triangleright\ \Psi \mid \Delta_4 \vdash w : \bullet D \mid \Delta_5, w : \bullet D \vdash T \mid \Delta_1, \Delta_2 \vdash x : \bullet A \otimes B \mid \Delta_3, x : \bullet A \otimes B \vdash T''}\ \otimes\mathsf{C}}{(\boldsymbol{\nu} w)\ \overrightarrow{x(y)}; P \triangleright\ \Psi \mid \Delta_4, \Delta_5 \vdash T \mid \Delta_1, \Delta_2 \vdash \bullet A \otimes B^x \mid \Delta_3, \bullet A \otimes B^x \vdash T''}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright\ \Psi \mid \Delta_4 \vdash w : \bullet D \mid \Delta_5, w : \bullet D \vdash T \mid \Delta_1 \vdash x : \bullet B \mid \Delta_2 \vdash y : \bullet A \mid \Delta_3, y : \bullet A, x : \bullet B \vdash T''}{(\boldsymbol{\nu} w)\ P \triangleright\ \Psi \mid \Delta_4, \Delta_5 \vdash T \mid \Delta_1 \vdash x : \bullet B \mid \Delta_2 \vdash y : \bullet A \mid \Delta_3, y : \bullet A, x : \bullet B \vdash T''}\ \mathsf{Scope}}{\overrightarrow{x(y)}; (\boldsymbol{\nu} w)\ P \triangleright\ \Psi \mid \Delta_4, \Delta_5 \vdash T \mid \Delta_1, \Delta_2 \vdash x : \bullet A \otimes B \mid \Delta_3, x : \bullet A \otimes B \vdash T''}\ \otimes\mathsf{C}$$

Case 2:

$$\dfrac{\dfrac{P \triangleright\ \Psi \mid \Delta_4 \vdash w : \bullet D \mid \Delta_1, w : \bullet D \vdash x : \bullet B \mid \Delta_2 \vdash y : \bullet A \mid \Delta_3, y : \bullet A, x : \bullet B \vdash T''}{\overrightarrow{x(y)}; P \triangleright\ \Psi \mid \Delta_4 \vdash w : \bullet D \mid \Delta_1, \Delta_2, w : \bullet D \vdash x : \bullet A \otimes B \mid \Delta_3, x : \bullet A \otimes B \vdash T''}\ \otimes\mathsf{C}}{(\boldsymbol{\nu} w)\ \overrightarrow{x(y)}; P \triangleright\ \Psi \mid \Delta_1, \Delta_2, \Delta_4 \vdash x : \bullet A \otimes B \mid \Delta_3, x : \bullet A \otimes B \vdash T''}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright\ \Psi \mid \Delta_4 \vdash w : \bullet D \mid \Delta_1, w : \bullet D \vdash x : \bullet B^x \mid \Delta_2 \vdash y : \bullet A \mid \Delta_3, y : \bullet A, x : \bullet B \vdash T''}{(\boldsymbol{\nu} w)\ P \triangleright\ \Psi \mid \Delta_1, \Delta_4 \vdash x : \bullet B \mid \Delta_2 \vdash y : \bullet A \mid \Delta_3, y : \bullet A, x : \bullet B \vdash T''}\ \mathsf{Scope}}{\overrightarrow{x(y)}; (\boldsymbol{\nu} w)\ P \triangleright\ \Psi \mid \Delta_1, \Delta_2, \Delta_4 \vdash x : \bullet A \otimes B \mid \Delta_3, x : \bullet A \otimes B \vdash T''}\ \otimes\mathsf{C}$$

Case 3:

$$\dfrac{\dfrac{P \triangleright\ \Psi \mid \Delta_4 \vdash w : \bullet D \mid \Delta_1 \vdash x : \bullet B \mid \Delta_2, w : \bullet D \vdash y : \bullet A \mid \Delta_3, y : \bullet A, x : \bullet B \vdash T''}{\overrightarrow{x(y)}; P \triangleright\ \Psi \mid \Delta_4 \vdash w : \bullet D \mid \Delta_1, \Delta_2, w : \bullet D \vdash x : \bullet A \otimes B \mid \Delta_3, x : \bullet A \otimes B \vdash T''}\ \otimes\mathsf{C}}{(\boldsymbol{\nu} w)\ \overrightarrow{x(y)}; P \triangleright\ \Psi \mid \Delta_1, \Delta_2, \Delta_4 \vdash x : \bullet A \otimes B \mid \Delta_3, x : \bullet A \otimes B \vdash T''}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright\ \Psi \mid \Delta_4 \vdash w : \bullet D \mid \Delta_1 \vdash x : \bullet B \mid \Delta_2, w : \bullet D \vdash y : \bullet A \mid \Delta_3, y : \bullet A, x : \bullet B \vdash T''}{(\boldsymbol{\nu} w)\ P \triangleright\ \Psi \mid \Delta_1 \vdash x : \bullet B \mid \Delta_2, \Delta_4 \vdash y : \bullet A \mid \Delta_3, y : \bullet A, x : \bullet B \vdash T''}\ \mathsf{Scope}}{\overrightarrow{x(y)}; (\boldsymbol{\nu} w)\ P \triangleright\ \Psi \mid \Delta_1, \Delta_2, \Delta_4 \vdash x : \bullet A \otimes B \mid \Delta_3, x : \bullet A \otimes B \vdash T''}\ \otimes\mathsf{C}$$

Case 4:

$$\dfrac{\dfrac{P \triangleright\ \Psi \mid \Delta_4 \vdash w : \bullet D \mid \Delta_1 \vdash x : \bullet B \mid \Delta_2 \vdash y : \bullet A \mid \Delta_3, w : \bullet D, y : \bullet A, x : \bullet B \vdash T''}{\overrightarrow{x(y)}; P \triangleright\ \Psi \mid \Delta_4 \vdash w : \bullet D \mid \Delta_1, \Delta_2 \vdash x : \bullet A \otimes B \mid \Delta_3, \bullet D, x : \bullet A \otimes B \vdash T''}\ \otimes\mathsf{C}}{(\boldsymbol{\nu} w)\ \overrightarrow{x(y)}; P \triangleright\ \Psi \mid \Delta_1, \Delta_2 \vdash x : \bullet A \otimes B \mid \Delta_3, \Delta_4, x : \bullet A \otimes B \vdash T''}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_4 \vdash w : \bullet D \mid \Delta_1 \vdash x : \bullet B \mid \Delta_2 \vdash y : \bullet A \mid \Delta_3, w : \bullet D, y : \bullet A, x : \bullet B \vdash T''}{(\boldsymbol{\nu} w)\ P \triangleright \Psi \mid \Delta_1 \vdash x : \bullet B \mid \Delta_2 \vdash y : \bullet A \mid \Delta_3, \Delta_4, y : \bullet A, x : \bullet B \vdash T''}\ \mathsf{Scope}}{\overrightarrow{x(y)}; (\boldsymbol{\nu} w)\ P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash x : \bullet A \otimes B \mid \Delta_3, \Delta_4, x : \bullet A \otimes B \vdash T''}\ \otimes \mathsf{C}$$

Case 5:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash x : \bullet B \mid \Delta_2 \vdash y : \bullet A \mid \Delta_3, y : \bullet A, x : \bullet B \vdash w : \bullet D \mid \Delta_4, w : \bullet D \vdash T''}{\overrightarrow{x(y)}; P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash x : \bullet A \otimes B \mid \Delta_3, x : \bullet A \otimes B \vdash w : \bullet D \mid \Delta_4, w : \bullet D \vdash T''}\ \otimes \mathsf{C}}{(\boldsymbol{\nu} w)\ \overrightarrow{x(y)}; P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash x : \bullet A \otimes B \mid \Delta_3, \Delta_4, x : \bullet A \otimes B \vdash T''}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash x : \bullet B \mid \Delta_2 \vdash y : \bullet A \mid \Delta_3, y : \bullet A, x : \bullet B \vdash w : \bullet D \mid \Delta_4, w : \bullet D \vdash T''}{(\boldsymbol{\nu} w)\ P \triangleright \Psi \mid \Delta_1 \vdash x : \bullet B \mid \Delta_2 \vdash y : \bullet A \mid \Delta_3, \Delta_4, y : \bullet A, x : \bullet B \vdash T''}\ \mathsf{Scope}}{\overrightarrow{x(y)}; (\boldsymbol{\nu} w)\ P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash x : \bullet A \otimes B \mid \Delta_3, \Delta_4, x : \bullet A \otimes B \vdash T''}\ \otimes \mathsf{C}$$

Commuting Scope with $\multimap$ C

$$[\mathsf{Scope}/ \multimap \mathsf{C}] \quad (\boldsymbol{\nu} w)\ \overrightarrow{x(y)}; P \equiv \overrightarrow{x(y)}; (\boldsymbol{\nu} w)\ P$$

Case 1:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_4 \vdash w : \bullet D \mid \Delta_5, w : \bullet D \vdash T' \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, x : \bullet B \vdash T'' \mid \Delta_3, y : \bullet A \vdash x : \bullet B}{\overrightarrow{x(y)}; P \triangleright \Psi \mid \Delta_4 \vdash w : \bullet D \mid \Delta_5, w : \bullet D \vdash T' \mid \Delta_1, \Delta_2, x : \bullet A \multimap B \vdash T'' \mid \Delta_3 \vdash x : \bullet A \multimap B}\ \multimap \mathsf{C}}{(\boldsymbol{\nu} w)\ \overrightarrow{x(y)}; P \triangleright \Psi \mid \Delta_4, \Delta_5 \vdash T' \mid \Delta_1, \Delta_2, x : \bullet A \multimap B^x \vdash T'' \mid \Delta_3 \vdash x : \bullet A \multimap B}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_4 \vdash w : \bullet D \mid \Delta_5, w : \bullet D \vdash T' \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, x : \bullet B \vdash T'' \mid \Delta_3, y : \bullet A \vdash x : \bullet B}{(\boldsymbol{\nu} w)\ P \triangleright \Psi \mid \Psi \mid \Delta_4, \Delta_5 \vdash T' \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, x : \bullet B \vdash T'' \mid \Delta_3, y : \bullet A \vdash x : \bullet B}\ \mathsf{Scope}}{\overrightarrow{x(y)}; (\boldsymbol{\nu} w)\ P \triangleright \Psi \mid \Delta_4, \Delta_5 \vdash T' \mid \Delta_1, \Delta_2, x : \bullet A \multimap B \vdash T'' \mid \Delta_3 \vdash x : \bullet A \multimap B}\ \multimap \mathsf{C}$$

Case 2:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_4 \vdash w : \bullet D \mid \Delta_1, w : \bullet D \vdash y : \bullet A \mid \Delta_2, x : \bullet B \vdash T'' \mid \Delta_3, y : \bullet A \vdash x : \bullet B}{\overrightarrow{x(y)}; P \triangleright \Psi \mid \Delta_4 \vdash w : \bullet D \mid \Delta_1, \Delta_2, w : \bullet D, x : \bullet A \multimap B \vdash T'' \mid \Delta_3 \vdash x : \bullet A \multimap B}\ \multimap \mathsf{C}}{(\boldsymbol{\nu} w)\ \overrightarrow{x(y)}; P \triangleright \Psi \mid \Delta_1, \Delta_2, \Delta_4, x : \bullet A \multimap B \vdash T'' \mid \Delta_3 \vdash x : \bullet A \multimap B}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_4 \vdash w : \bullet D \mid \Delta_1, w : \bullet D \vdash y : \bullet A \mid \Delta_2, x : \bullet B \vdash T'' \mid \Delta_3, y : \bullet A \vdash x : \bullet B}{(\boldsymbol{\nu} w)\ P \triangleright \Psi \mid \Psi \mid \Delta_1, \Delta_4 \vdash y : \bullet A \mid \Delta_2, x : \bullet B \vdash T'' \mid \Delta_3, y : \bullet A \vdash x : \bullet B}\ \mathsf{Scope}}{\overrightarrow{x(y)}; (\boldsymbol{\nu} w)\ P \triangleright \Psi \mid \Delta_1, \Delta_2, \Delta_4, x : \bullet A \multimap B \vdash T'' \mid \Delta_3 \vdash x : \bullet A \multimap B}\ \multimap \mathsf{C}$$

Case 3:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_4 \vdash w : \bullet D \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, w : \bullet D, x : \bullet B \vdash T'' \mid \Delta_3, y : \bullet A \vdash x : \bullet B}{\overrightarrow{x(y)}; P \triangleright \Psi \mid \Delta_4 \vdash w : \bullet D \mid \Delta_1, \Delta_2, w : \bullet D, x : \bullet A \multimap B \vdash T'' \mid \Delta_3 \vdash x : \bullet A \multimap B}\ \multimap \mathsf{C}}{(\boldsymbol{\nu} w)\ \overrightarrow{x(y)}; P \triangleright \Psi \mid \Delta_1, \Delta_2, \Delta_4, x : \bullet A \multimap B \vdash T'' \mid \Delta_3 \vdash x : \bullet A \multimap B}\ \mathsf{Scope}$$

is equivalent to

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_4 \vdash w : \bullet D \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, w : \bullet D, x : \bullet B \vdash T'' \mid \Delta_3, y : \bullet A \vdash x : \bullet B}{(\boldsymbol{\nu} w)\ P \triangleright \Psi \mid \Psi \mid \Delta_4, \Delta_5 \vdash T' \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, \Delta_4, x : \bullet B \vdash T'' \mid \Delta_3, y : \bullet A \vdash x : \bullet B}\ \mathsf{Scope}}{\overrightarrow{x(y)}; (\boldsymbol{\nu} w)\ P \triangleright \Psi \mid \Delta_1, \Delta_2, \Delta_4, x : \bullet A \multimap B \vdash T'' \mid \Delta_3 \vdash x : \bullet A \multimap B}\ \multimap \mathsf{C}$$

Case 4:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_4 \vdash w : \bullet D \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, x : \bullet B \vdash T'' \mid \Delta_3, w : \bullet D, y : \bullet A \vdash x : \bullet B}{\overrightarrow{x(y)}; P \triangleright \Psi \mid \Delta_4 \vdash w : \bullet D \mid \Delta_1, \Delta_2, x : \bullet A \multimap B \vdash T'' \mid \Delta_3, w : \bullet D \vdash x : \bullet A \multimap B}\ \multimap \mathsf{C}}{(\boldsymbol{\nu} w)\ \overrightarrow{x(y)}; P \triangleright \Psi \mid \Delta_1, \Delta_2, x : \bullet A \multimap B \vdash T'' \mid \Delta_3, \Delta_4 \vdash x : \bullet A \multimap B}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_4 \vdash w : \bullet D \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, x : \bullet B \vdash T'' \mid \Delta_3, w : \bullet D, y : \bullet A \vdash x : \bullet B}{(\boldsymbol{\nu} w)\ P \triangleright \Psi \mid \Psi \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, x : \bullet B \vdash T'' \mid \Delta_3, \Delta_4, y : \bullet A \vdash x : \bullet B}\ \mathsf{Scope}}{\overrightarrow{x(y)}; (\boldsymbol{\nu} w)\ P \triangleright \Psi \mid \Delta_1, \Delta_2, x : \bullet A \multimap B \vdash T'' \mid \Delta_3, \Delta_4 \vdash x : \bullet A \multimap B}\ \multimap \mathsf{C}$$

Case 5:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_4, w : \bullet D \vdash T' \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, x : \bullet B \vdash w : \bullet D \mid \Delta_3, y : \bullet A \vdash x : \bullet B}{\overrightarrow{x(y)}; P \triangleright \Psi \mid \Delta_4, w : \bullet D \vdash T' \mid \Delta_1, \Delta_2, x : \bullet A \multimap B \vdash w : \bullet D \mid \Delta_3 \vdash x : \bullet A \multimap B}\ \multimap \mathsf{C}}{(\boldsymbol{\nu} w)\ \overrightarrow{x(y)}; P \triangleright \Psi \mid \Delta_1, \Delta_2, \Delta_4, x : \bullet A \multimap B \vdash T' \mid \Delta_3 \vdash x : \bullet A \multimap B}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_4, w : \bullet D \vdash T' \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, x : \bullet B^x \vdash w : \bullet D \mid \Delta_3, y : \bullet A \vdash x : \bullet B}{(\boldsymbol{\nu} w)\ P \triangleright \Psi \mid \Psi \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, \Delta_4, x : \bullet B \vdash T' \mid \Delta_3, y : \bullet A \vdash x : \bullet B}\ \mathsf{Scope}}{\overrightarrow{x(y)}; (\boldsymbol{\nu} w)\ P \triangleright \Psi \mid \Delta_1, \Delta_2, \Delta_4, x : \bullet A \multimap B \vdash T' \mid \Delta_3 \vdash x : \bullet A \multimap B}\ \multimap \mathsf{C}$$

Commuting Scope with $\oplus\mathsf{C}_1$ and $\oplus\mathsf{C}_2$

$$[\mathsf{Scope}/\oplus\mathsf{C}_1/\mathsf{L}] \quad (\boldsymbol{\nu} w)\ \overrightarrow{x.\mathsf{I}}\ (P,Q) \equiv \overrightarrow{x.\mathsf{I}}\ ((\boldsymbol{\nu} w)\ P, Q) \qquad \Big(w \notin \mathsf{fn}(Q)\Big)$$

Case 1:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T' \quad Q \triangleright \Psi' \mid \Delta_2, x : B \vdash T}{\overrightarrow{x.\mathsf{I}}\ (P,Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \oplus B \mid \Delta_2, x : \bullet A \oplus B \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\ \oplus\mathsf{C}_1}{(\boldsymbol{\nu} w)\ \overrightarrow{x.\mathsf{I}}\ (P,Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \oplus B \mid \Delta_2, x : \bullet A \oplus B \vdash T \mid \Delta, \Delta' \vdash T'}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}{(\boldsymbol{\nu} w)\ P \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash T \mid \Delta, \Delta' \vdash T'}\ \mathsf{Scope} \quad Q \triangleright \Psi' \mid \Delta_2, x : B \vdash T}{\overrightarrow{x.\mathsf{I}}\ ((\boldsymbol{\nu} w)\ P, Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \oplus B \mid \Delta_2, x : \bullet A \oplus B \vdash T \mid \Delta, \Delta' \vdash T'}\ \oplus\mathsf{C}_1$$

Case 2:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Psi' \mid \Delta_1, w : \bullet C \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash T \mid \Delta \vdash w : \bullet C \quad Q \triangleright \Psi' \mid \Delta_2, x : B \vdash T}{\overrightarrow{x.\mathsf{I}}\ (P,Q) \triangleright \Psi \mid \Psi' \mid \Delta_1, w : \bullet C \vdash x : \bullet A \oplus B \mid \Delta_2, x : \bullet A \oplus B \vdash T \mid \Delta \vdash w : \bullet C}\ \mathsf{Scope}}{(\boldsymbol{\nu} w)\ \overrightarrow{x.\mathsf{I}}\ (P,Q) \triangleright \Psi \mid \Psi' \mid \Delta, \Delta_1 \vdash x : \bullet A \oplus B \mid \Delta_2, x : \bullet A \oplus B \vdash T}\ \oplus\mathsf{C}_1$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Psi' \mid \Delta_1, w : \bullet C \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash T \mid \Delta \vdash w : \bullet C}{(\boldsymbol{\nu} w)\ P \triangleright \Psi \mid \Psi' \mid \Delta, \Delta_1 \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash T}\ \mathsf{Scope} \quad Q \triangleright \Psi' \mid \Delta_2, x : B \vdash T}{\overrightarrow{x.\mathsf{I}}\ ((\boldsymbol{\nu} w)\ P, Q) \triangleright \Psi \mid \Psi' \mid \Delta, \Delta_1 \vdash x : \bullet A \oplus B \mid \Delta_2, x : \bullet A \oplus B \vdash T}\ \oplus\mathsf{C}_1$$

$$[\mathsf{Scope}/\oplus\mathsf{C}_1/\mathsf{L}/\mathsf{R}] \quad (\boldsymbol{\nu} w)\ \overrightarrow{x.\mathsf{I}}\ (P,Q) \equiv \overrightarrow{x.\mathsf{I}}\ ((\boldsymbol{\nu} w)\ P, (\boldsymbol{\nu} w)\ Q) \qquad \Big(w \in \mathsf{fn}(Q)\Big)$$

Case 1:

$$\dfrac{\dfrac{P \triangleright \begin{array}{l}\Psi \mid \Psi' \\ \mid\ \Delta_1 \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash T \\ \mid\ \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'\end{array} \quad Q \triangleright \begin{array}{l}\Psi' \mid \Delta_2, x : B \vdash T \\ \mid\ \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'\end{array}}{\overrightarrow{x.\mathsf{I}}\ (P,Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \oplus B \mid \Delta_2, x : \bullet A \oplus B \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\ \oplus\mathsf{C}_1}{(\boldsymbol{\nu} w)\ \overrightarrow{x.\mathsf{I}}\ (P,Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \oplus B \mid \Delta_2, x : \bullet A \oplus B \vdash T \mid \Delta, \Delta' \vdash T'}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \begin{array}{l}\Psi \mid \Psi' \\ \mid\ \Delta_1 \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash T \\ \mid\ \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'\end{array}}{(\boldsymbol{\nu} w)\ P \triangleright \begin{array}{l}\Psi \mid \Psi' \\ \mid\ \Delta_1 \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash T \\ \mid\ \Delta, \Delta' \vdash T'\end{array}}\ \mathsf{Scope} \quad Q \triangleright \begin{array}{l}\Psi' \mid \Delta_2, x : B \vdash T \\ \mid\ \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'\end{array}}{(\boldsymbol{\nu} w)\ Q \triangleright \Psi' \mid \Delta_2, x : B \vdash T \mid \Delta, \Delta' \vdash T'}\ \mathsf{Scope}}{\overrightarrow{x.\mathsf{I}}\ ((\boldsymbol{\nu} w)\ P, (\boldsymbol{\nu} w)\ Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \oplus B \mid \Delta_2, x : \bullet A \oplus B \vdash T \mid \Delta, \Delta' \vdash T'}\ \oplus\mathsf{C}_1$$

Case 2:

$$\dfrac{\dfrac{P \triangleright \begin{array}{l}\Psi \mid \Psi' \\ \mid\ \Delta_1 \vdash x : \bullet A \\ \mid\ \Delta \vdash w : \bullet C\end{array} \quad \mid\ \Delta_2, w : \bullet C, x : \bullet A \vdash T \qquad Q \triangleright \Psi' \mid \Delta_2, w : \bullet C, x : B \vdash T \quad \mid\ \Delta \vdash w : \bullet C}{\overrightarrow{x.\mathsf{l}}\ (P, Q) \triangleright \Psi \mid \Psi' \mid\ \Delta_1 \vdash x : \bullet A \oplus B \quad \mid\ \Delta_2, w : \bullet C, x : \bullet A \oplus B \vdash T \quad \mid\ \Delta \vdash w : \bullet C}\ \oplus\mathsf{C}_1}{(\boldsymbol{\nu} w)\ \overrightarrow{x.\mathsf{l}}\ (P, Q) \triangleright \Psi \mid \Psi' \mid\ \Delta_1 \vdash x : \bullet A \oplus B \quad \mid\ \Delta, \Delta_2, x : \bullet A \oplus B \vdash T}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \begin{array}{l}\Psi \mid \Psi' \\ \mid\ \Delta_1 \vdash x : \bullet A \\ \mid\ \Delta \vdash w : \bullet C\end{array} \quad \mid\ \Delta_2, w : \bullet C, x : \bullet A \vdash T}{(\boldsymbol{\nu} w)\ P \triangleright \begin{array}{l}\Psi \mid \Psi' \\ \mid\ \Delta_1 \vdash x : \bullet A \quad \mid\ \Delta, \Delta_2, x : \bullet A \vdash T\end{array}}\ \mathsf{Scope} \qquad \dfrac{Q \triangleright \begin{array}{l}\Psi' \mid \Delta_2, w : \bullet C, x : B \vdash T \\ \mid\ \Delta \vdash w : \bullet C\end{array}}{(\boldsymbol{\nu} w)\ Q \triangleright \Psi' \mid \Delta, \Delta_2, x : B \vdash T}\ \mathsf{Scope}}{\overrightarrow{x.\mathsf{l}}\ ((\boldsymbol{\nu} w)\ P, (\boldsymbol{\nu} w)\ Q) \triangleright \Psi \mid \Psi' \mid\ \Delta_1 \vdash x : \bullet A \oplus B \quad \mid\ \Delta, \Delta_2, x : \bullet A \oplus B \vdash T}\ \oplus\mathsf{C}_1$$

Case 3:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T' \quad Q \triangleright \Psi' \mid \Delta_2, x : B \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}{\overrightarrow{x.\mathsf{l}}\ (P, Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \oplus B \mid \Delta_2, x : \bullet A \oplus B \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\ \mathsf{Scope}}{(\boldsymbol{\nu} w)\ \overrightarrow{x.\mathsf{l}}\ (P, Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \oplus B \mid \Delta', \Delta_2, x : \bullet A \oplus B \vdash T'}\ \oplus\mathsf{C}_1$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \begin{array}{l}\Psi \mid \Psi' \\ \mid\ \Delta_1 \vdash x : \bullet A \quad \mid\ \Delta_2, x : \bullet A \vdash w : \bullet C \\ \mid\ \Delta', w : \bullet C \vdash T'\end{array}}{(\boldsymbol{\nu} w)\ P \triangleright \begin{array}{l}\Psi \mid \Psi' \\ \mid\ \Delta_1 \vdash x : \bullet A \mid \Delta_2, \Delta', x : \bullet A \vdash T'\end{array}}\ \mathsf{Scope} \qquad \dfrac{Q \triangleright \Psi' \mid \Delta_2, x : B \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}{(\boldsymbol{\nu} w)\ Q \triangleright \Psi' \mid \Delta_2, \Delta', x : B \vdash T'}\ \mathsf{Scope}}{\overrightarrow{x.\mathsf{l}}\ ((\boldsymbol{\nu} w)\ P, (\boldsymbol{\nu} w)\ Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \oplus B \mid \Delta', \Delta_2, x : \bullet A \oplus B \vdash T'}\ \oplus\mathsf{C}_1$$

$$\boxed{[\mathsf{Scope}/\oplus\mathsf{C}_2/\mathsf{R}] \quad (\boldsymbol{\nu} w)\ \overrightarrow{x.\mathsf{r}}\ (P, Q) \equiv \overrightarrow{x.\mathsf{r}}\ (P, (\boldsymbol{\nu} w)\ Q) \qquad \Big(w \notin \mathsf{fn}(P),\ w \neq x\Big)}$$

Case 1:

$$\dfrac{\dfrac{P \triangleright \Psi' \mid \Delta_2, x : A \vdash T \quad Q \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet B \mid \Delta_2, x : \bullet B \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}{\overrightarrow{x.\mathsf{r}}\ (P, Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \oplus B \mid \Delta_2, x : \bullet A \oplus B \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\ \mathsf{Scope}}{(\boldsymbol{\nu} w)\ \overrightarrow{x.\mathsf{r}}\ (P, Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \oplus B \mid \Delta_2, x : \bullet A \oplus B \vdash T \mid \Delta, \Delta' \vdash T'}\ \oplus\mathsf{C}_2$$

is equivalent to ($\equiv$)

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

Case 2:

$$\dfrac{\dfrac{P \triangleright \Psi' \mid \Delta_2, x : A \vdash T \quad Q \triangleright \Psi \mid \Psi' \mid \Delta_1, w : \bullet C \vdash x : \bullet B \mid \Delta_2, x : \bullet B \vdash T \mid \Delta \vdash w : \bullet C}{\overrightarrow{x.\mathsf{r}}\ (P, Q) \triangleright \Psi \mid \Psi' \mid \Delta_1, w : \bullet C \vdash x : \bullet A \oplus B \mid \Delta_2, x : \bullet A \oplus B \vdash T \mid \Delta \vdash w : \bullet C}\ \mathsf{Scope}}{(\boldsymbol{\nu} w)\ \overrightarrow{x.\mathsf{r}}\ (P, Q) \triangleright \Psi \mid \Psi' \mid \Delta, \Delta_1 \vdash x : \bullet A \oplus B \mid \Delta_2, x : \bullet A \oplus B \vdash T}\ \oplus\mathsf{C}_2$$

is equivalent to ($\equiv$)

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

$$[\mathsf{Scope}/\oplus\mathsf{C}_2/\mathsf{L}/\mathsf{R}] \quad (\boldsymbol{\nu} w)\ \overrightarrow{x.\mathsf{r}}\ (P, Q) \equiv \overrightarrow{x.\mathsf{r}}\ ((\boldsymbol{\nu} w)\ P, (\boldsymbol{\nu} w)\ Q) \qquad \Big(w \in \mathsf{fn}(P)\Big)$$

Case 1:

$$\dfrac{\dfrac{P \triangleright \Psi' \mid \Delta_2, x : A \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T' \qquad Q \triangleright \begin{array}{l} \Psi \mid \Psi' \\ \mid\ \Delta_1 \vdash x : \bullet B \mid \Delta_2, x : \bullet B \vdash T \\ \mid\ \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T' \end{array}}{\overrightarrow{x.\mathsf{r}}\ (P, Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \oplus B \mid \Delta_2, x : \bullet A \oplus B \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\ \oplus\mathsf{C}_2}{(\boldsymbol{\nu} w)\ \overrightarrow{x.\mathsf{r}}\ (P, Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \oplus B \mid \Delta_2, x : \bullet A \oplus B \vdash T \mid \Delta, \Delta' \vdash T'}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \begin{array}{l} \Psi' \mid \Delta_2, x : A \vdash T \\ \mid\ \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T' \end{array}}{(\boldsymbol{\nu} w)\ P \triangleright \Psi' \mid \Delta_2, x : A \vdash T \mid \Delta, \Delta' \vdash T'}\ \mathsf{Scope} \qquad \dfrac{Q \triangleright \begin{array}{l} \Psi \mid \Psi' \\ \mid\ \Delta_1 \vdash x : \bullet B \mid \Delta_2, x : \bullet B \vdash T \\ \mid\ \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T' \end{array}}{(\boldsymbol{\nu} w)\ Q \triangleright \begin{array}{l} \Psi \mid \Psi' \\ \mid\ \Delta_1 \vdash x : \bullet B \mid \Delta_2, x : \bullet B \vdash T \\ \mid\ \Delta, \Delta' \vdash T' \end{array}}\ \mathsf{Scope}}{\overrightarrow{x.\mathsf{r}}\ ((\boldsymbol{\nu} w)\ P, (\boldsymbol{\nu} w)\ Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \oplus B \mid \Delta_2, x : \bullet A \oplus B \vdash T \mid \Delta, \Delta' \vdash T'}\ \oplus\mathsf{C}_2$$

Case 2:

$$\dfrac{\dfrac{P \triangleright \Psi' \mid \Delta_2, w : \bullet C, x : A \vdash T \mid \Delta \vdash w : \bullet C \qquad Q \triangleright \begin{array}{l} \Psi \mid \Psi' \\ \mid\ \Delta_1 \vdash x : \bullet B \mid \Delta_2, w : \bullet C, x : \bullet B \vdash T \\ \mid\ \Delta \vdash w : \bullet C \end{array}}{\overrightarrow{x.\mathsf{r}}\ (P, Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \oplus B \mid \Delta_2, w : \bullet C, x : \bullet A \oplus B \vdash T \mid \Delta \vdash w : \bullet C}\ \oplus\mathsf{C}_2}{(\boldsymbol{\nu} w)\ \overrightarrow{x.\mathsf{r}}\ (P, Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \oplus B \mid \Delta, \Delta_2, x : \bullet A \oplus B \vdash T}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \begin{array}{l} \Psi' \mid \Delta_2, x : A \vdash T \\ \mid\ \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T' \end{array}}{(\boldsymbol{\nu} w)\ P \triangleright \Psi' \mid \Delta_2, x : A \vdash T \mid \Delta, \Delta' \vdash T'}\ \mathsf{Scope} \qquad \dfrac{Q \triangleright \begin{array}{l} \Psi \mid \Psi' \\ \mid\ \Delta_1 \vdash x : \bullet B \mid \Delta_2, w : \bullet C, x : \bullet B \vdash T \\ \mid\ \Delta \vdash w : \bullet C \end{array}}{(\boldsymbol{\nu} w)\ Q \triangleright \begin{array}{l} \Psi \mid \Psi' \mid \\ \mid\ \Delta_1 \vdash x : \bullet B \mid \Delta, \Delta_2, x : \bullet B \vdash T \end{array}}\ \mathsf{Scope}}{\overrightarrow{x.\mathsf{r}}\ ((\boldsymbol{\nu} w)\ P, (\boldsymbol{\nu} w)\ Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \oplus B \mid \Delta, \Delta_2, x : \bullet A \oplus B \vdash T}\ \oplus\mathsf{C}_2$$

Case 3:

$$\dfrac{\dfrac{P \triangleright \Psi' \mid \Delta_2, x : A \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T' \qquad Q \triangleright \begin{array}{l} \Psi \mid \Psi' \\ \mid\ \Delta_1 \vdash x : \bullet B \\ \mid\ \Delta_2, x : \bullet B \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T' \end{array}}{\overrightarrow{x.\mathsf{r}}\ (P, Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \oplus B \mid \Delta_2, x : \bullet A \oplus B \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\ \oplus\mathsf{C}_2}{(\boldsymbol{\nu} w)\ \overrightarrow{x.\mathsf{r}}\ (P, Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \oplus B \mid \Delta_2, \Delta', x : \bullet A \oplus B \vdash T'}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \begin{array}{l}\Psi' \mid \Delta_2, x : A \vdash w : \bullet C \\ \mid\ \Delta', w : \bullet C \vdash T'\end{array}}{(\boldsymbol{\nu} w)\ P \triangleright \Psi' \mid \Delta', \Delta_2, x : A \vdash T'}\ \mathsf{Scope} \qquad \dfrac{Q \triangleright \begin{array}{l}\Psi \mid \Psi' \\ \mid\ \Delta_1 \vdash x : \bullet B \mid \Delta_2, x : \bullet B \vdash w : \bullet C \\ \mid\ \Delta', w : \bullet C \vdash T'\end{array}}{(\boldsymbol{\nu} w)\ Q \triangleright \begin{array}{l}\Psi \mid \Psi' \\ \mid\ \Delta_1 \vdash x : \bullet B \mid \Delta', \Delta_2, x : \bullet B \vdash T'\end{array}}\ \mathsf{Scope}}{\overrightarrow{x.\mathsf{r}}\ ((\boldsymbol{\nu} w)\ P, (\boldsymbol{\nu} w)\ Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \oplus B \mid \Delta', \Delta_2, x : \bullet A \oplus B \vdash T'}\ \oplus\mathsf{C}_2$$

Commuting Scope with $\&\mathsf{C}_1$ and $\&\mathsf{C}_2$

$$[\mathsf{Scope}/\&\mathsf{C}_1/\mathsf{L}] \quad (\boldsymbol{\nu} w)\ \overrightarrow{x.\mathsf{l}}\ (P, Q) \equiv \overrightarrow{x.\mathsf{l}}\ ((\boldsymbol{\nu} w)\ P, Q) \qquad \Big(w \notin \mathsf{fn}(Q)\Big)$$

Case 1:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T' \quad Q \triangleright \Psi' \mid \Delta_1 \vdash x : B}{\overrightarrow{x.\mathsf{l}}\ (P, Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \& B \mid \Delta_2, x : \bullet A \& B \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\ \&\mathsf{C}_1}{(\boldsymbol{\nu} w)\ \overrightarrow{x.\mathsf{l}}\ (P, Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \& B \mid \Delta_2, x : \bullet A \& B \vdash T \mid \Delta, \Delta' \vdash T'}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}{(\boldsymbol{\nu} w)\ P \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash T \mid \Delta, \Delta' \vdash T'}\ \mathsf{Scope} \quad Q \triangleright \Psi' \mid \Delta_1 \vdash x : B}{\overrightarrow{x.\mathsf{l}}\ ((\boldsymbol{\nu} w)\ P, Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \& B \mid \Delta_2, x : \bullet A \& B \vdash T \mid \Delta, \Delta' \vdash T'}\ \&\mathsf{C}_1$$

Case 2:

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

is equivalent to ($\equiv$)

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

Case 3:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T' \quad Q \triangleright \Psi' \mid \Delta_1 \vdash x : B}{\overrightarrow{x.\mathsf{l}}\ (P, Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \& B \mid \Delta_2, x : \bullet A \& B \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\ \&\mathsf{C}_1}{(\boldsymbol{\nu} w)\ \overrightarrow{x.\mathsf{l}}\ (P, Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \& B \mid \Delta', \Delta_2, x : \bullet A \& B \vdash T'}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

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

$$\boxed{[\mathsf{Scope}/\&\mathsf{C}_1/\mathsf{L}/\mathsf{R}] \quad (\boldsymbol{\nu} w)\ \overrightarrow{x.\mathsf{l}}\ (P, Q) \equiv \overrightarrow{x.\mathsf{l}}\ ((\boldsymbol{\nu} w)\ P, (\boldsymbol{\nu} w)\ Q) \qquad \Big(w \in \mathsf{fn}(Q)\Big)}$$

Case 1:

$$\dfrac{\dfrac{P \triangleright \begin{array}{l}\Psi \mid \Psi' \\ \quad \mid\ \Delta_1 \vdash x : \bullet A \quad \mid\ \Delta_2, x : \bullet A \vdash T \\ \quad \mid\ \Delta \vdash w : \bullet C \quad \mid\ \Delta', w : \bullet C \vdash T'\end{array} \qquad Q \triangleright \begin{array}{l}\Psi' \mid \Delta_1 \vdash x : B \\ \quad \mid\ \Delta \vdash w : \bullet C \quad \mid\ \Delta', w : \bullet C \vdash T'\end{array}}{\overrightarrow{x.\mathsf{l}}\ (P, Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \& B \mid \Delta_2, x : \bullet A \& B \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\ \&\mathsf{C}_1}{(\boldsymbol{\nu} w)\ \overrightarrow{x.\mathsf{l}}\ (P, Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \& B \mid \Delta_2, x : \bullet A \& B \vdash T \mid \Delta, \Delta' \vdash T'}\ \mathsf{Scope}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \begin{array}{l}\Psi \mid \Psi' \\ \quad \mid\ \Delta_1 \vdash x : \bullet A \quad \mid\ \Delta_2, x : \bullet A \vdash T \\ \quad \mid\ \Delta \vdash w : \bullet C \quad \mid\ \Delta', w : \bullet C \vdash T'\end{array}}{(\boldsymbol{\nu} w)\ P \triangleright \begin{array}{l}\Psi \mid \Psi' \\ \quad \mid\ \Delta_1 \vdash x : \bullet A \quad \mid\ \Delta_2, x : \bullet A \vdash T \\ \quad \mid\ \Delta, \Delta' \vdash T'\end{array}}\ \mathsf{Scope} \qquad \dfrac{Q \triangleright \begin{array}{l}\Psi' \mid \Delta_1 \vdash x : B \\ \quad \mid\ \Delta \vdash w : \bullet C \quad \mid\ \Delta', w : \bullet C \vdash T'\end{array}}{(\boldsymbol{\nu} w)\ Q \triangleright \Psi' \mid \Delta_1 \vdash x : B \quad \mid\ \Delta, \Delta' \vdash T'}\ \mathsf{Scope}}{\overrightarrow{x.\mathsf{l}}\ ((\boldsymbol{\nu} w)\ P, (\boldsymbol{\nu} w)\ Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \& B \mid \Delta_2, x : \bullet A \& B \vdash T \mid \Delta, \Delta' \vdash T'}\ \&\mathsf{C}_1$$

Case 2:

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

is equivalent to ($\equiv$)

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

$$\boxed{[\mathsf{Scope}/\&\mathsf{C}_2/\mathsf{R}] \quad (\boldsymbol{\nu} w)\ \overrightarrow{x.\mathsf{r}}\ (P, Q) \equiv \overrightarrow{x.\mathsf{r}}\ (P, (\boldsymbol{\nu} w)\ Q) \qquad \Big(w \notin \mathsf{fn}(P)\Big)}$$

Similar to $[\mathsf{Scope}/\oplus\mathsf{C}_2/\mathsf{R}]$

$$\boxed{[\mathsf{Scope}/\&\mathsf{C}_2/\mathsf{L}/\mathsf{R}] \quad (\boldsymbol{\nu} w)\ \overrightarrow{x.\mathsf{r}}\ (P, Q) \equiv \overrightarrow{x.\mathsf{r}}\ ((\boldsymbol{\nu} w)\ P, (\boldsymbol{\nu} w)\ Q) \qquad \Big(w \in \mathsf{fn}(P)\Big)}$$

Similar to $[\mathsf{Scope}/\oplus\mathsf{C}_2/\mathsf{R}]$

Commuting Conn with Conn

$$\boxed{[\mathsf{Conn}/\mathsf{Conn}] \quad (P \mid_y Q) \mid_x R \equiv P \mid_y (Q \mid_x R)}$$

Case 1:

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta_1 \vdash y : A \quad Q \triangleright \Psi_2 \mid \Delta_2, y : A \vdash T \mid \Delta_3 \vdash x : B}{P \mid_y Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, y : \bullet A \vdash T \mid \Delta_3 \vdash x : B}\,\mathsf{Conn} \quad R \triangleright \Psi_3 \mid \Delta_4, x : B \vdash T'}{(P \mid_y Q) \mid_x R \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, y : \bullet A \vdash T \mid \Delta_3 \vdash x : \bullet B \mid \Delta_4, x : \bullet B \vdash T'}\,\mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{P \triangleright \Psi_1 \mid \Delta_1 \vdash y : A \quad \dfrac{Q \triangleright \Psi_2 \mid \Delta_2, y : A \vdash T \mid \Delta_3 \vdash x : B \quad R \triangleright \Psi_3 \mid \Delta_4, x : B \vdash T'}{Q \mid_x R \triangleright \Psi_2 \mid \Psi_3 \mid \Delta_2, y : A \vdash T \mid \Delta_3 \vdash x : \bullet B \mid \Delta_4, x : \bullet B \vdash T'}\,\mathsf{Conn}}{P \mid_y (Q \mid_x R) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, y : \bullet A \vdash T \mid \Delta_3 \vdash x : \bullet B \mid \Delta_4, x : \bullet B \vdash T'}\,\mathsf{Conn}$$

Case 2:

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta_1 \vdash y : A \quad Q \triangleright \Psi_2 \mid \Delta_3, y : A \vdash x : B}{P \mid_y Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1 \vdash y : \bullet A \mid \Delta_3, y : \bullet A \vdash x : B}\,\mathsf{Conn} \quad R \triangleright \Psi_3 \mid \Delta_4, x : B \vdash T'}{(P \mid_y Q) \mid_x R \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash y : \bullet A \mid \Delta_3, y : \bullet A \vdash x : \bullet B \mid \Delta_4, x : \bullet B \vdash T'}\,\mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{P \triangleright \Psi_1 \mid \Delta_1 \vdash y : A \quad \dfrac{Q \triangleright \Psi_2 \mid \Delta_3, y : A \vdash x : B \quad R \triangleright \Psi_3 \mid \Delta_4, x : B \vdash T'}{Q \mid_x R \triangleright \Psi_2 \mid \Psi_3 \mid \Delta_3, y : A \vdash x : \bullet B \mid \Delta_4, x : \bullet B \vdash T'}\,\mathsf{Conn}}{P \mid_y (Q \mid_x R) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash y : \bullet A \mid \Delta_3, y : \bullet \vdash x : \bullet B \mid \Delta_4, x : \bullet B \vdash T'}\,\mathsf{Conn}$$

Commuting Conn with 1L (no case for 1R)

$$\boxed{[\mathsf{Conn}/1\mathsf{L}/\mathsf{L}] \quad \mathsf{wait}[x]; P \mid_y Q \equiv \mathsf{wait}[x]; (P \mid_y Q)}$$

Case 1:

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash T' \mid \Delta_1 \vdash y : A}{\mathsf{wait}[x]; P \triangleright \Psi_1 \mid \Delta, x : \mathbf{1} \vdash T' \mid \Delta_1 \vdash y : A}\,1\mathsf{L} \quad Q \triangleright \Psi_2 \mid \Delta_2, y : A \vdash T}{\mathsf{wait}[x]; P \mid_y Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta, x : \mathbf{1} \vdash T' \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, y : \bullet A \vdash T}\,\mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash T' \mid \Delta_1 \vdash y : A \qquad Q \triangleright \Psi_2 \mid \Delta_2, y : A \vdash T}{P \mid_y Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash T' \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, y : \bullet A \vdash T}\,\mathsf{Conn}}{\mathsf{wait}[x]; (P \mid_y Q) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta, x : \mathbf{1} \vdash T' \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, y : \bullet A \vdash T}\,1\mathsf{L}$$

Case 2:

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta_1 \vdash y : A}{\mathsf{wait}[x]; P \triangleright \Psi_1 \mid \Delta_1, x : \mathbf{1} \vdash y : A}\,\mathsf{1L} \qquad Q \triangleright \Psi_2 \mid \Delta_2, y : A \vdash T}{\mathsf{wait}[x]; P \mid_y Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1, x : \mathbf{1} \vdash y : \bullet A \mid \Delta_2, y : \bullet A \vdash T}\,\mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta_1 \vdash y : A \qquad Q \triangleright \Psi_2 \mid \Delta_2, y : A \vdash T}{P \mid_y Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, y : \bullet A \vdash T}\,\mathsf{Conn}}{\mathsf{wait}[x]; (P \mid_y Q) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1, x : \mathbf{1} \vdash y : \bullet A \mid \Delta_2, y : \bullet A \vdash T}\,\mathsf{1L}$$

$$\boxed{[\mathsf{Conn}/\mathsf{1L}/\mathsf{R}] \quad P \mid_y \mathsf{wait}[x]; Q \equiv \mathsf{wait}[x]; (P \mid_y Q)}$$

Case 1:

$$\dfrac{P \triangleright \Psi_1 \mid \Delta_1 \vdash y : A \qquad \dfrac{Q \triangleright \Psi_2 \mid \Delta \vdash T' \mid \Delta_2, y : A \vdash T}{\mathsf{wait}[x]; Q \triangleright \Psi_1 \mid \Delta, x : \mathbf{1} \vdash T' \mid \Delta_2, y : A \vdash T}\,\mathsf{1L}}{\mathsf{wait}[x]; P \mid_y Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta, x : \mathbf{1} \vdash T' \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, y : \bullet A \vdash T}\,\mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta_1 \vdash y : A \qquad Q \triangleright \Psi_2 \mid \Delta \vdash T' \mid \Delta_2, y : A \vdash T}{P \mid_y Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash T' \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, y : \bullet A \vdash T}\,\mathsf{Conn}}{\mathsf{wait}[x]; (P \mid_y Q) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta, x : \mathbf{1} \vdash T' \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, y : \bullet A \vdash T}\,\mathsf{1L}$$

Case 2:

$$\dfrac{P \triangleright \Psi_1 \mid \Delta_1 \vdash y : A \qquad \dfrac{Q \triangleright \Psi_2 \mid \Delta_2, y : A \vdash T}{\mathsf{wait}[x]; Q \triangleright \Psi_1 \mid \Delta_2, x : \mathbf{1}, y : A \vdash T}\,\mathsf{1L}}{\mathsf{wait}[x]; P \mid_y Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, x : \mathbf{1}, y : \bullet A \vdash T}\,\mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta_1 \vdash y : A \qquad Q \triangleright \Psi_2 \mid \Delta_2, y : A \vdash T}{P \mid_y Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, y : \bullet A \vdash T}\,\mathsf{Conn}}{\mathsf{wait}[x]; (P \mid_y Q) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, x : \mathbf{1}, y : \bullet A \vdash T}\,\mathsf{1L}$$

Commuting Conn with $\otimes$R and $\otimes$L

$$\boxed{[\mathsf{Conn}/\otimes\,\mathsf{R}/\mathsf{R}/\mathsf{L}] \quad P \mid_w \overline{x}(y); (Q \mid R) \equiv \overline{x}(y); ((P \mid_w Q) \mid R)}$$

Case 1:

$$\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \qquad \dfrac{Q \triangleright \Psi_2 \mid \Delta_1 \vdash y : A \mid \Delta', w : C \vdash T \qquad R \triangleright \Psi_3 \mid \Delta_2 \vdash x : B}{\overline{x}(y);(Q \mid R) \triangleright \Psi_2 \mid \Psi_1 \mid \Delta_1, \Delta_2 \vdash x : A \otimes B \mid \Delta', w : C \vdash T}\otimes\mathsf{R}}{P \mid_w \overline{x}(y);(Q \mid R) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1, \Delta_2 \vdash x : A \otimes B \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T}\mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \qquad Q \triangleright \Psi_2 \mid \Delta', w : C \vdash T \mid \Delta_1 \vdash y : A}{P \mid_w Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1 \vdash y : A \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T}\mathsf{Conn} \qquad R \triangleright \Psi_3 \mid \Delta_2 \vdash x : B}{\overline{x}(y);((P \mid_w Q) \mid R) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1, \Delta_2 \vdash x : A \otimes B \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T}\otimes\mathsf{R}$$

Case 2:

$$\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \qquad \dfrac{Q \triangleright \Psi_2 \mid \Delta_1, w : C \vdash y : A \qquad R \triangleright \Psi_3 \mid \Delta_2 \vdash x : B}{\overline{x}(y);(Q \mid R) \triangleright \Psi_2 \mid \Psi_1 \mid \Delta_1, \Delta_2, w : C \vdash x : A \otimes B}\otimes\mathsf{R}}{P \mid_w \overline{x}(y);(Q \mid R) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta \vdash w : \bullet C \mid \Delta_1, \Delta_2, w : \bullet C \vdash x : A \otimes B}\mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \qquad Q \triangleright \Psi_2 \mid \Delta_1, w : C \vdash y : A}{P \mid_w Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta_1, w : \bullet C \vdash y : A}\mathsf{Conn} \qquad R \triangleright \Psi_3 \mid \Delta_2 \vdash x : B}{\overline{x}(y);((P \mid_w Q) \mid R) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta \vdash w : \bullet C \mid \Delta_1, \Delta_2, w : \bullet C \vdash x : A \otimes B}\otimes\mathsf{R}$$

$$\boxed{[\mathsf{Conn}/\otimes\mathsf{R}/\mathsf{R}/\mathsf{R}] \quad P \mid_w \overline{x}(y);(Q \mid R) \equiv \overline{x}(y);(Q \mid (P \mid_w R))}$$

Case 1:

$$\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \qquad \dfrac{Q \triangleright \Psi_2 \mid \Delta_1 \vdash y : A \qquad R \triangleright \Psi_3 \mid \Delta_2 \vdash x : B \mid \Delta', w : C \vdash T}{\overline{x}(y);(Q \mid R) \triangleright \Psi_2 \mid \Psi_1 \mid \Delta_1, \Delta_2 \vdash x : A \otimes B \mid \Delta', w : C \vdash T}\otimes\mathsf{R}}{P \mid_w \overline{x}(y);(Q \mid R) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1, \Delta_2 \vdash x : A \otimes B \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T}\mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{Q \triangleright \Psi_2 \mid \Delta_1 \vdash y : A \qquad \dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \qquad R \triangleright \Psi_3 \mid \Delta_2 \vdash x : B \mid \Delta', w : C \vdash T}{P \mid_w R \triangleright \Psi_1 \mid \Psi_3 \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T \mid \Delta_2 \vdash x : B}\mathsf{Conn}}{\overline{x}(y);(Q \mid (P \mid_w R)) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1, \Delta_2 \vdash x : A \otimes B \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T}\otimes\mathsf{R}$$

Case 2:

$$\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \qquad \dfrac{Q \triangleright \Psi_2 \mid \Delta_1 \vdash y : A \qquad R \triangleright \Psi_3 \mid \Delta_2, w : C \vdash x : B}{\overline{x}(y);(Q \mid R) \triangleright \Psi_2 \mid \Psi_1 \mid \Delta_1, \Delta_2, w : C \vdash x : A \otimes B}\otimes\mathsf{R}}{P \mid_w \overline{x}(y);(Q \mid R) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta \vdash w : \bullet C \mid \Delta_1, \Delta_2, w : \bullet C \vdash x : A \otimes B}\mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{Q \triangleright \Psi_2 \mid \Delta_1 \vdash y : A \qquad \dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \qquad R \triangleright \Psi_3 \mid \Delta_2, w : C \vdash x : B}{P \mid_w R \triangleright \Psi_1 \mid \Psi_3 \mid \Delta \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash x : B}\mathsf{Conn}}{\overline{x}(y);(Q \mid (P \mid_w R)) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta \vdash w : \bullet C \mid \Delta_1, \Delta_2, w : \bullet C \vdash x : A \otimes B}\otimes\mathsf{R}$$

$$\boxed{[\mathsf{Conn}/\otimes\mathsf{L}/\mathsf{L}] \quad x(y);P\mid_w Q \equiv x(y);(P\mid_w Q)}$$

Case 1:

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta_1, y:A, x:B \vdash T \mid \Delta_2 \vdash w:C}{x(y);P \triangleright \Psi_1 \mid \Delta_1, x:A\otimes B \vdash T \mid \Delta_2 \vdash w:C}\otimes\mathsf{L} \qquad Q \triangleright \Psi_2 \mid \Delta_3, w:C \vdash T'}{x(y);P\mid_w Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1, x:A\otimes B \vdash T \mid \Delta_2 \vdash w:\bullet C \mid \Delta_3, w:\bullet C \vdash T'}\,\mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta_1, y:A, x:B \vdash T \mid \Delta_2 \vdash w:C \qquad Q \triangleright \Psi_2 \mid \Delta_3, w:C \vdash T'}{P\mid_w Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1, y:A, x:B \vdash T \mid \Delta_2 \vdash w:\bullet C \mid \Delta_3, w:\bullet C \vdash T'}\,\mathsf{Conn}}{x(y);(P\mid_w Q) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1, x:A\otimes B \vdash T \mid \Delta_2 \vdash w:C \mid \Delta_3, w:C \vdash T'}\otimes\mathsf{L}$$

Case 2:

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta_1, y:A, x:B \vdash w:C}{x(y);P \triangleright \Psi_1 \mid \Delta_1, x:A\otimes B \vdash w:C}\otimes\mathsf{L} \qquad Q \triangleright \Psi_2 \mid \Delta_3, w:C \vdash T'}{x(y);P\mid_w Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1, x:A\otimes B \vdash w:\bullet C \mid \Delta_3, w:\bullet C \vdash T'}\,\mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta_1, y:A, x:B \vdash w:C \qquad Q \triangleright \Psi_2 \mid \Delta_3, w:C \vdash T'}{P\mid_w Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1, y:A, x:B \vdash w:\bullet C \mid \Delta_3, w:\bullet C \vdash T'}\,\mathsf{Conn}}{x(y);(P\mid_w Q) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1, x:A\otimes B \vdash w:\bullet C \mid \Delta_3, w:C \vdash T'}\otimes\mathsf{L}$$

$$\boxed{[\mathsf{Conn}/\otimes\mathsf{L}/\mathsf{R}] \quad P\mid_w x(y);Q \equiv x(y);(P\mid_w Q)}$$

Case 1:

$$\dfrac{P \triangleright \Psi_1 \mid \Delta_1 \vdash w:C \qquad \dfrac{Q \triangleright \Psi_2 \mid \Delta_2, y:A, x:B \vdash T \mid \Delta_3, w:C \vdash T'}{x(y);Q \triangleright \Psi_2 \mid \Delta_2, x:A\otimes B \vdash T \mid \Delta_3, w:C \vdash T'}\otimes\mathsf{L}}{P\mid_w x(y);Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1 \vdash w:\bullet C \mid \Delta_2, x:A\otimes B \vdash T \mid \Delta_3, w:\bullet C \vdash T'}\,\mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta_1 \vdash w:C \qquad Q \triangleright \Psi_2 \mid \Delta_2, y:A, x:B \vdash T \mid \Delta_3, w:C \vdash T'}{P\mid_w Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1 \vdash w:\bullet C \mid \Delta_2, y:A, x:B \vdash T \mid \Delta_3, w:\bullet C \vdash T'}\,\mathsf{Conn}}{x(y);(P\mid_w Q) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1 \vdash w:\bullet C \mid \Delta_2, x:A\otimes B \vdash T \mid \Delta_3, w:\bullet C \vdash T'}\otimes\mathsf{L}$$

Case 2:

$$\dfrac{P \triangleright \Psi_1 \mid \Delta_1 \vdash w:C \qquad \dfrac{Q \triangleright \Psi_2 \mid \Delta_2, w:C, y:A, x:B \vdash T}{x(y);Q \triangleright \Psi_2 \mid \Delta_2, w:C, x:A\otimes B \vdash T}\otimes\mathsf{L}}{P\mid_w x(y);Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1 \vdash w:\bullet C \mid \Delta_2, w:\bullet C x:A\otimes B \vdash T}\,\mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta_1 \vdash w:C \qquad Q \triangleright \Psi_2 \mid \Delta_2, w:C, y:A, x:B \vdash T}{P\mid_w Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1 \vdash w:\bullet C \mid \Delta_2, w:\bullet C, y:A, x:B \vdash T}\,\mathsf{Conn}}{x(y);(P\mid_w Q) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1 \vdash w:\bullet C \mid \Delta_2, w:\bullet C, x:A\otimes B \vdash T}\otimes\mathsf{L}$$

Commuting Conn with $\multimap$ R and $\multimap$ L

$$\boxed{[\mathsf{Conn}/\multimap \mathsf{R}/\mathsf{R}] \quad P \mid_w x(y); Q \equiv x(y); (P \mid_w Q)}$$

Case 1:

$$\dfrac{P \triangleright \Psi_1 \mid \Delta_1 \vdash w : C \qquad \dfrac{Q \triangleright \Psi_2 \mid \Delta_2, y : A \vdash x : B \mid \Delta_3, w : C \vdash T}{x(y); Q \triangleright \Psi_2 \mid \Delta_2 \vdash x : A \multimap B \mid \Delta_3, w : C \vdash T}\ \multimap \mathsf{R}}{P \mid_w x(y); Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2 \vdash x : A \multimap B \mid \Delta_3, w : \bullet C \vdash T}\ \mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta_1 \vdash w : C \qquad Q \triangleright \Psi_2 \mid \Delta_2, y : A \vdash x : B \mid \Delta_3, w : C \vdash T}{P \mid_w Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, y : A \vdash x : B \mid \Delta_3, w : \bullet C \vdash T}\ \mathsf{Conn}}{x(y); (P \mid_w Q) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2 \vdash x : A \multimap B \mid \Delta_3, w : \bullet C \vdash T}\ \multimap \mathsf{R}$$

Case 2:

$$\dfrac{P \triangleright \Psi_1 \mid \Delta_1 \vdash w : C \qquad \dfrac{Q \triangleright \Psi_2 \mid \Delta_2, w : C, y : A \vdash x : B}{x(y); Q \triangleright \Psi_2 \mid \Delta_2, w : C \vdash x : A \multimap B}\ \multimap \mathsf{R}}{P \mid_w x(y); Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash x : A \multimap B}\ \mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta_1 \vdash w : C \qquad Q \triangleright \Psi_2 \mid \Delta_2, w : C, y : A \vdash x : B}{P \mid_w Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C, y : A \vdash x : B}\ \mathsf{Conn}}{x(y); (P \mid_w Q) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1 \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash x : A \multimap B}\ \multimap \mathsf{R}$$

$$\boxed{[\mathsf{Conn}/\multimap \mathsf{L}/\mathsf{L}/\mathsf{R}] \quad \overline{x}(y); (P \mid Q) \mid_w R \equiv \overline{x}(y); (P \mid (Q \mid_w R))}$$

Case 1:

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta_1 \vdash y : A \qquad Q \triangleright \Psi_2 \mid \Delta_2, x : B \vdash T \mid \Delta \vdash w : C}{\overline{x}(y); (P \mid Q) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1, \Delta_2, x : A \multimap B \vdash T \mid \Delta \vdash w : C}\ \multimap \mathsf{L} \qquad R \triangleright \Psi_3 \mid \Delta', w : C \vdash T'}{\overline{x}(y); (P \mid Q) \mid_w R \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1, \Delta_2, x : A \multimap B \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\ \mathsf{Conn}$$

is equivalent ($\equiv$)

$$\dfrac{P \triangleright \Psi_1 \mid \Delta_1 \vdash y : A \qquad \dfrac{Q \triangleright \Psi_2 \mid \Delta_2, x : B \vdash T \mid \Delta \vdash w : C \qquad R \triangleright \Psi_3 \mid \Delta', w : C \vdash T'}{Q \mid_w R \triangleright \Psi_2 \mid \Psi_3 \mid \Delta_2, x : B \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\ \mathsf{Conn}}{\overline{x}(y); (P \mid (Q \mid_w R)) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1, \Delta_2, x : A \multimap B \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\ \multimap \mathsf{L}$$

Case 2:

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta_1 \vdash y : A \qquad Q \triangleright \Psi_2 \mid \Delta_2, x : B \vdash w : C}{\overline{x}(y); (P \mid Q) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1, \Delta_2, x : A \multimap B \vdash w : C}\ \multimap \mathsf{L} \qquad R \triangleright \Psi_3 \mid \Delta', w : C \vdash T'}{\overline{x}(y); (P \mid Q) \mid_w R \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1, \Delta_2, x : A \multimap B \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\ \mathsf{Conn}$$

is equivalent ($\equiv$)

$$\dfrac{P \triangleright \Psi_1 \mid \Delta_1 \vdash y : A \qquad \dfrac{Q \triangleright \Psi_2 \mid \Delta_2, x : B \vdash w : C \qquad R \triangleright \Psi_3 \mid \Delta', w : C \vdash T'}{Q \mid_w R \triangleright \Psi_2 \mid \Psi_3 \mid \Delta_2, x : B \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\ \mathsf{Conn}}{\overline{x}(y); (P \mid (Q \mid_w R)) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1, \Delta_2, x : A \multimap B \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\ \multimap \mathsf{L}$$

$$\boxed{[\mathsf{Conn}/\multimap \mathsf{L}/\mathsf{R}/\mathsf{L}] \quad P \mid_w \overline{x}(y); (Q \mid R) \equiv \overline{x}(y); (P \mid_w Q) \mid R)}$$

Case 1:

$$\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \qquad \dfrac{Q \triangleright \Psi_2 \mid \Delta_1 \vdash y : A \mid \Delta', w : C \vdash T \qquad R \triangleright \Psi_3 \mid \Delta_2, x : B \vdash T'}{\overline{x}(y); (Q \mid R) \triangleright \Psi_2 \mid \Psi_3 \mid \Delta_1, \Delta_2, x : A \multimap B \vdash T' \mid \Delta', w : C \vdash T}\ \multimap \mathsf{L}}{P \mid_w \overline{x}(y); (Q \mid R) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta \vdash w : \bullet C \mid \Delta_1, \Delta_2, x : A \multimap B \vdash T' \mid \Delta', w : \bullet C \vdash T}\ \mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \qquad Q \triangleright \Psi_2 \mid \Delta_1 \vdash y : A \mid \Delta', w : C \vdash T}{P \mid_w Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta_1 \vdash y : A \mid \Delta', w : \bullet C \vdash T}\ \mathsf{Conn} \qquad R \triangleright \Psi_3 \mid \Delta_2, x : B \vdash T'}{\overline{x}(y); (P \mid_w Q) \mid R) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta \vdash w : \bullet C \mid \Delta_1, \Delta_2, x : A \multimap B \vdash T' \mid \Delta', w : \bullet C \vdash T}\ \multimap \mathsf{L}$$

Case 2:

$$\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \qquad \dfrac{Q \triangleright \Psi_2 \mid \Delta_1, w : C \vdash y : A \qquad R \triangleright \Psi_3 \mid \Delta_2, x : B \vdash T'}{\overline{x}(y); (Q \mid R) \triangleright \Psi_2 \mid \Psi_3 \mid \Delta_1, \Delta_2, w : C, x : A \multimap B \vdash T'}\ \multimap \mathsf{L}}{P \mid_w \overline{x}(y); (Q \mid R) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta \vdash w : \bullet C \mid \Delta_1, \Delta_2, w : \bullet C, x : A \multimap B \vdash T'}\ \mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \qquad Q \triangleright \Psi_2 \mid \Delta_1, w : C \vdash y : A}{P \mid_w Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta_1, w : \bullet C \vdash y : A}\ \mathsf{Conn} \qquad R \triangleright \Psi_3 \mid \Delta_2, x : B \vdash T'}{\overline{x}(y); (P \mid_w Q) \mid R) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta \vdash w : \bullet C \mid \Delta_1, \Delta_2, w : \bullet C, x : A \multimap B \vdash T'}\ \multimap \mathsf{L}$$

$$\boxed{[\mathsf{Conn}/\multimap \mathsf{L}/\mathsf{R}/\mathsf{R}] \quad P \mid_w \overline{x}(y); (Q \mid R) \equiv \overline{x}(y); (Q \mid (P \mid_w R))}$$

Case 1:

$$\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \qquad \dfrac{Q \triangleright \Psi_2 \mid \Delta_1 \vdash y : A \qquad R \triangleright \Psi_3 \mid \Delta_2, x : B \vdash T' \mid \Delta', w : C \vdash T}{\overline{x}(y); (Q \mid R) \triangleright \Psi_2 \mid \Psi_3 \mid \Delta_1, \Delta_2, x : A \multimap B \vdash T' \mid \Delta', w : C \vdash T}\ \multimap \mathsf{L}}{P \mid_w \overline{x}(y); (Q \mid R) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta \vdash w : \bullet C \mid \Delta_1, \Delta_2, x : A \multimap B \vdash T' \mid \Delta', w : \bullet C \vdash T}\ \mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{Q \triangleright \Psi_2 \mid \Delta_1 \vdash y : A \qquad \dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \qquad R \triangleright \Psi_3 \mid \Delta_2, x : B \vdash T' \mid \Delta', w : C \vdash T}{P \mid_w R \triangleright \Psi_1 \mid \Psi_3 \mid \Delta \vdash w : \bullet C \mid \Delta_2, x : B \vdash T' \mid \Delta', w : \bullet C \vdash T}\ \mathsf{Conn}}{\overline{x}(y); (Q \mid (P \mid_w R)) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta \vdash w : \bullet C \mid \Delta_1, \Delta_2, x : A \multimap B \vdash T' \mid \Delta', w : \bullet C \vdash T}\ \multimap \mathsf{L}$$

Case 2:

$$\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \qquad \dfrac{Q \triangleright \Psi_2 \mid \Delta_1 \vdash y : A \qquad R \triangleright \Psi_3 \mid \Delta_2, w : C, x : B \vdash T'}{\overline{x}(y); (Q \mid R) \triangleright \Psi_2 \mid \Psi_3 \mid \Delta_1, \Delta_2, w : C, x : A \multimap B \vdash T'}\ \multimap \mathsf{L}}{P \mid_w \overline{x}(y); (Q \mid R) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta \vdash w : \bullet C \mid \Delta_1, \Delta_2, w : \bullet C, x : A \multimap B \vdash T'}\ \mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \qquad R \triangleright \Psi_3 \mid \Delta_2, w : C, x : B \vdash T'}{P \mid_w R \triangleright \Psi_1 \mid \Psi_3 \mid \Delta \vdash w : \bullet C \mid \Delta_2, w : \bullet C, x : B \vdash T'}\ \mathsf{Conn} \qquad Q \triangleright \Psi_2 \mid \Delta_1 \vdash y : A}{\overline{x}(y); (Q \mid (P \mid_w R)) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta \vdash w : \bullet C \mid \Delta_1, \Delta_2, w : \bullet C, x : A \multimap B \vdash T'}\ \multimap \mathsf{L}$$

Commuting Conn with $\oplus$R and $\oplus$L

$$\boxed{[\mathsf{Conn}/\oplus \mathsf{R}_1/\mathsf{R}] \quad P \mid_w x.\mathsf{inl}; Q \equiv x.\mathsf{inl}; (P \mid_w Q)}$$

Case 1:

$$\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \qquad \dfrac{Q \triangleright \Psi_2 \mid \Delta', w : C \vdash T \mid \Delta_1 \vdash x : A}{x.\mathsf{inl}; Q \triangleright \Psi_2 \mid \Delta', w : C \vdash T \mid \Delta_1 \vdash x : A \oplus B}\ \oplus \mathsf{R}_1}{P \mid_w x.\mathsf{inl}; Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T \mid \Delta_1 \vdash x : A \oplus B}\ \mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \quad Q \triangleright \Psi_2 \mid \Delta', w : C \vdash T \mid \Delta_1 \vdash x : A}{P \mid_w Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T \mid \Delta_1 \vdash x : A}\ \mathsf{Conn}}{x.\mathsf{inl}; (P \mid_w Q) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T \mid \Delta_1 \vdash x : A \oplus B}\ \mathsf{Conn}$$

Case 2:

$$\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \qquad \dfrac{Q \triangleright \Psi_2 \mid \Delta_1, w : C \vdash x : A}{x.\mathsf{inl}; Q \triangleright \Psi_2 \mid \Delta_1, w : C \vdash x : A \oplus B}\ \oplus \mathsf{R}_1}{P \mid_w x.\mathsf{inl}; Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta_1, w : \bullet C \vdash x : A \oplus B}\ \mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \quad Q \triangleright \Psi_2 \mid \Delta_1, w : C \vdash x : A}{P \mid_w Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta_1, w : \bullet C \vdash x : A}\ \mathsf{Conn}}{x.\mathsf{inl}; (P \mid_w Q) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta_1, w : \bullet C \vdash x : A \oplus B}\ \mathsf{Conn}$$

$$\boxed{[\mathsf{Conn}/\oplus \mathsf{R}_2/\mathsf{R}] \quad P \mid_w x.\mathsf{inr}; Q \equiv x.\mathsf{inr}; (P \mid_w Q)}$$

Similar to $[\mathsf{Conn}/\oplus \mathsf{R}_1/\mathsf{R}]$

$$\boxed{[\mathsf{Conn}/\oplus\mathsf{L}/\mathsf{L}] \quad x.\mathsf{case}(P,Q)\mid_w R \equiv x.\mathsf{case}(P\mid_w R\ ,\ Q\mid_w R)}$$

Case 1:

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta_1, x : A \vdash T \mid \Delta \vdash w : C \quad Q \triangleright \Psi_1 \mid \Delta_1, x : B \vdash T \mid \Delta \vdash w : C}{x.\mathsf{case}(P,Q) \triangleright \Psi_1 \mid \Delta_1, x : A \oplus B \vdash T \mid \Delta \vdash w : C}\ \oplus\mathsf{L} \quad R \triangleright \Psi_2 \mid \Delta', w : C \vdash T'}{x.\mathsf{case}(P,Q)\mid_w R \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1, x : A \oplus B \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\ \mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta_1, x : A \vdash T \Delta \vdash w : C \quad R \triangleright \Psi_2 \mid \Delta_2, w : C \vdash T}{P\mid_w R \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1, x : A \vdash \mid \Delta \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T}\ \mathsf{Conn} \quad \dfrac{Q \triangleright \Psi_1 \mid \Delta_1, x : B \vdash T \mid \Delta \vdash w : C \quad R \triangleright \Psi_2 \mid \Delta', w : C \vdash T}{Q\mid_w R \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1, x : B \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T}\ \mathsf{Conn}}{x.\mathsf{case}(P\mid_w R\ ,\ Q\mid_w R) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1, x : A \oplus B \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T}\ \oplus\mathsf{L}$$

Case 2:

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta_1, x : A \vdash w : C \quad Q \triangleright \Psi_1 \mid \Delta_1, x : B \vdash w : C}{x.\mathsf{case}(P,Q) \triangleright \Psi_1 \mid \Delta_1, x : A \oplus B \vdash w : C}\ \oplus\mathsf{L} \quad R \triangleright \Psi_2 \mid \Delta_2, w : C \vdash T}{x.\mathsf{case}(P,Q)\mid_w R \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1, x : A \oplus B \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T}\ \mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta_1, x : A \vdash w : C \quad R \triangleright \Psi_2 \mid \Delta_2, w : C \vdash T}{P\mid_w R \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1, x : A \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T}\ \mathsf{Conn} \quad \dfrac{Q \triangleright \Psi_1 \mid \Delta_1, x : B \vdash w : C \quad R \triangleright \Psi_2 \mid \Delta_2, w : C \vdash T}{Q\mid_w R \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1, x : B \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T}\ \mathsf{Conn}}{x.\mathsf{case}(P\mid_w R\ ,\ Q\mid_w R) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1, x : A \oplus B \vdash w : \bullet C \mid \Delta_2, w : \bullet C \vdash T}\ \oplus\mathsf{L}$$

$$\boxed{[\mathsf{Conn}/\oplus\mathsf{L}/\mathsf{R}] \quad P\mid_w x.\mathsf{case}(Q,R) \equiv x.\mathsf{case}(P\mid_w Q\ ,\ P\mid_w R)}$$

Case 1:

$$\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \qquad \dfrac{Q \triangleright \Psi_2 \mid \Delta', w : C \vdash T \mid \Delta_1, x : A \vdash T' \quad R \triangleright \Psi_2 \mid \Delta', w : C \vdash T \mid \Delta_1, x : B \vdash T'}{x.\mathsf{case}(Q,R) \triangleright \Psi_2 \mid \Delta', w : C \vdash T \mid \Delta_1, x : A \oplus B \vdash T'}\ \oplus\mathsf{L}}{P\mid_w x.\mathsf{case}(Q,R) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T \mid \Delta_1, x : A \oplus B \vdash T'}\ \mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \quad Q \triangleright \Psi_2 \mid \Delta', w : C \vdash T \mid \Delta_1, x : A \vdash T'}{P\mid_w Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T \mid \Delta_1, x : A \vdash T'}\ \mathsf{Conn} \quad \dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \quad R \triangleright \Psi_2 \mid \Delta', w : C \vdash T \mid \Delta_1, x : B \vdash T'}{P\mid_w R \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T \mid \Delta_1, x : B \vdash T'}\ \mathsf{Conn}}{x.\mathsf{case}(P\mid_w Q\ ,\ P\mid_w R) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T \mid \Delta_1, x : A \oplus B \vdash T'}\ \oplus\mathsf{L}$$

Case 2:

$$\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \qquad \dfrac{Q \triangleright \Psi_2 \mid \Delta_1, w : C, x : A \vdash T' \quad R \triangleright \Psi_2 \mid \Delta_1, w : C, x : B \vdash T'}{x.\mathsf{case}(Q,R) \triangleright \Psi_2 \mid \Delta_1, w : C, x : A \oplus B \vdash T'}\ \oplus\mathsf{L}}{P\mid_w x.\mathsf{case}(Q,R) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta_1, w : \bullet C, x : A \oplus B \vdash T'}\ \mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \quad Q \triangleright \Psi_2 \mid \Delta_1, w : C, x : A \vdash T'}{P \mid_w Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta_1, w : \bullet C, x : A \vdash T'}\ \mathsf{Conn} \quad \dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \quad R \triangleright \Psi_2 \mid \Delta_1, w : C, x : B \vdash T'}{P \mid_w R \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta_1, w : \bullet C, x : B \vdash T'}\ \mathsf{Conn}}{x.\mathsf{case}(P \mid_w Q\ ,\ P \mid_w R) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta_1, w : \bullet C, x : A \oplus B \vdash T'}\ \oplus\mathsf{L}$$

Commuting Conn with &R and &L

$$\boxed{[\mathsf{Conn}/\&\mathsf{R}/\mathsf{R}] \quad P \mid_w x.\mathsf{case}(Q, R) \equiv x.\mathsf{case}((P \mid_w Q)\ ,\ (P \mid_w R))}$$

Case 1:

$$\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \qquad \dfrac{Q \triangleright \Psi_2 \mid \Delta', w : C \vdash T \mid \Delta_1 \vdash x : A \quad R \triangleright \Psi_2 \mid \Delta', w : C \vdash T \mid \Delta_1 \vdash x : B}{x.\mathsf{case}(Q, R) \triangleright \Psi_2 \mid \Delta', w : C \vdash T \mid \Delta_1 \vdash x : A\&B}\ \&\mathsf{R}}{P \mid_w x.\mathsf{case}(Q, R) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T \mid \Delta_1 \vdash x : A\&B}\ \mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \quad Q \triangleright \Psi_2 \mid \Delta', w : C \vdash T \mid \Delta_1 \vdash x : A}{P \mid_w Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T \mid \Delta_1 \vdash x : A}\ \mathsf{Conn} \quad \dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \quad R \triangleright \Psi_2 \mid \Delta', w : C \vdash T \mid \Delta_1 \vdash x : B}{P \mid_w R \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T \mid \Delta_1 \vdash x : B}\ \mathsf{Conn}}{x.\mathsf{case}((P \mid_w Q)\ ,\ (P \mid_w R)) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T \mid \Delta_1 \vdash x : A\&B}\ \&\mathsf{R}$$

Case 2:

$$\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \qquad \dfrac{Q \triangleright \Psi_2 \mid \Delta', w : C \vdash x : A \quad R \triangleright \Psi_2 \mid \Delta', w : C \vdash x : B}{x.\mathsf{case}(Q, R) \triangleright \Psi_2 \mid \Delta', w : C \vdash x : A\&B}\ \&\mathsf{R}}{P \mid_w x.\mathsf{case}(Q, R) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash x : A\&B}\ \mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \quad Q \triangleright \Psi_2 \mid \Delta', w : C \vdash x : A}{P \mid_w Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash x : A}\ \mathsf{Conn} \quad \dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \quad R \triangleright \Psi_2 \mid \Delta', w : C \vdash x : B}{P \mid_w R \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash x : B}\ \mathsf{Conn}}{x.\mathsf{case}((P \mid_w Q)\ ,\ (P \mid_w R)) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash x : A\&B}\ \&\mathsf{R}$$

$$\boxed{[\mathsf{Conn}/\&\mathsf{L}_1/\mathsf{L}] \quad x.\mathsf{inl}; P \mid_w Q \equiv x.\mathsf{inl}; (P \mid_w Q)}$$

Case 1:

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta_1, x : A \vdash T \mid \Delta \vdash w : C}{x.\mathsf{inl}; P \triangleright \Psi_1 \mid \Delta_1, x : A\&B \vdash T \mid \Delta \vdash w : C}\ \&\mathsf{L}_1 \qquad Q \triangleright \Psi_2 \mid \Delta', w : C \vdash T'}{x.\mathsf{inl}; P \mid_w Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1, x : A\&B \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\ \mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta_1, x : A \vdash T \mid \Delta \vdash w : C \quad Q \triangleright \Psi_2 \mid \Delta', w : C \vdash T'}{P \mid_w Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1, x : A \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\,\mathsf{Conn}}{x.\mathsf{inl}; (P \mid_w Q) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1, x : A \& B \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\,\&\mathsf{L}_1$$

Case 2:

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta_1, x : A \vdash w : C}{x.\mathsf{inl}; P \triangleright \Psi_1 \mid \Delta_1, x : A \& B \vdash w : C}\,\&\mathsf{L}_1 \qquad Q \triangleright \Psi_2 \mid \Delta', w : C \vdash T'}{x.\mathsf{inl}; P \mid_w Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1, x : A \& B \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\,\mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta_1, x : A \vdash w : C \quad Q \triangleright \Psi_2 \mid \Delta', w : C \vdash T'}{P \mid_w Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1, x : A \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\,\mathsf{Conn}}{x.\mathsf{inl}; (P \mid_w Q) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1, x : A \& B \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\,\&\mathsf{L}_1$$

$$\boxed{[\mathsf{Conn}/\&\mathsf{L}_1/\mathsf{R}] \quad P \mid_w x.\mathsf{inl}; Q \equiv x.\mathsf{inl}; (P \mid_w Q)}$$

Case 1:

$$\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \qquad \dfrac{Q \triangleright \Psi_2 \mid \Delta', w : C \vdash T \mid \Delta_1, x : A \vdash T'}{x.\mathsf{inl}; Q \triangleright \Psi_2 \mid \Delta', w : C \vdash T \mid \Delta_1, x : A \& B \vdash T'}\,\&\mathsf{L}_1}{P \mid_w x.\mathsf{inl}; Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T \mid \Delta_1, x : A \& B \vdash T'}\,\mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \quad Q \triangleright \Psi_2 \mid \Delta', w : C \vdash T \mid \Delta_1, x : A \vdash T'}{P \mid_w x.\mathsf{inl}; Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T \mid \Delta_1, x : A \vdash T'}\,\mathsf{Conn}}{x.\mathsf{inl}; (P \mid_w Q) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T \mid \Delta_1, x : A \& B \vdash T'}\,\&\mathsf{L}_1$$

Case 2:

$$\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \qquad \dfrac{Q \triangleright \Psi_2 \mid \Delta_1, w : C, x : A \vdash T'}{x.\mathsf{inl}; Q \triangleright \Psi_2 \mid \Delta_1, w : C, x : A \& B \vdash T'}\,\&\mathsf{L}_1}{P \mid_w x.\mathsf{inl}; Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta_1, w : \bullet C, x : A \& B \vdash T'}\,\mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \quad Q \triangleright \Psi_2 \mid \Delta_1, w : C, x : A \vdash T'}{P \mid_w x.\mathsf{inl}; Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta_1, w : \bullet C, x : A \vdash T'}\,\mathsf{Conn}}{x.\mathsf{inl}; (P \mid_w Q) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta_1, w : \bullet C, x : A \& B \vdash T'}\,\&\mathsf{L}_1$$

$$\boxed{[\mathsf{Conn}/\&\mathsf{L}_2/\mathsf{L}] \quad x.\mathsf{inr}; P \mid_w Q \equiv x.\mathsf{inr}; (P \mid_w Q)}$$

Similar to $[\mathsf{Conn}/\&\mathsf{L}_1/\mathsf{L}]$.

$$[\mathsf{Conn}/\&\mathsf{L}_2/\mathsf{R}] \quad P \mid_w x.\mathsf{inr}; Q \equiv x.\mathsf{inr}; (P \mid_w Q)$$

Similar to $[\mathsf{Conn}/\&\mathsf{L}_1/\mathsf{R}]$.

Commuting Conn with 1C

$$[\mathsf{Conn}/1\mathsf{C}/\mathsf{L}] \quad \overrightarrow{\mathsf{close}}[x]; P \mid_w Q \equiv \overrightarrow{\mathsf{close}}[x]; (P \mid_w Q)$$

Case 1:

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta_1 \vdash T \mid \Delta \vdash w : C}{\overrightarrow{\mathsf{close}}[x]; P \triangleright \Psi_1 \mid \cdot \vdash x : \bullet\mathbf{1} \mid \Delta_1, x : \bullet\mathbf{1} \vdash T \mid \Delta \vdash w : C}\,1\mathsf{C} \qquad Q \triangleright \Psi_2 \mid \Delta', w : C \vdash T'}{\overrightarrow{\mathsf{close}}[x]; P \mid_w Q \triangleright \Psi_1 \mid \Psi_2 \mid \cdot \vdash x : \bullet\mathbf{1} \mid \Delta_1, x : \bullet\mathbf{1} \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\,\mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta_1 \vdash T \mid \Delta \vdash w : C \qquad Q \triangleright \Psi_2 \mid \Delta', w : C \vdash T'}{P \mid_w Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1 \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\,\mathsf{Conn}}{\overrightarrow{\mathsf{close}}[x]; (P \mid_w Q) \triangleright \Psi_1 \mid \Psi_2 \mid \cdot \vdash x : \bullet\mathbf{1} \mid \Delta_1, x : \bullet\mathbf{1} \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\,\mathsf{Conn}$$

Case 2:

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C}{\overrightarrow{\mathsf{close}}[x]; P \triangleright \Psi_1 \mid \cdot \vdash x : \bullet\mathbf{1} \mid \Delta, x : \bullet\mathbf{1} \vdash w : C}\,1\mathsf{C} \qquad Q \triangleright \Psi_2 \mid \Delta', w : C \vdash T'}{\overrightarrow{\mathsf{close}}[x]; P \mid_w Q \triangleright \Psi_1 \mid \Psi_2 \mid \cdot \vdash x : \bullet\mathbf{1} \mid \Delta, x : \bullet\mathbf{1} \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\,\mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \qquad Q \triangleright \Psi_2 \mid \Delta', w : C \vdash T'}{P \mid_w Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\,\mathsf{Conn}}{\overrightarrow{\mathsf{close}}[x]; (P \mid_w Q) \triangleright \Psi_1 \mid \Psi_2 \mid \cdot \vdash x : \bullet\mathbf{1} \mid \Delta, x : \bullet\mathbf{1} \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\,\mathsf{Conn}$$

$$[\mathsf{Conn}/1\mathsf{C}/\mathsf{R}] \quad P \mid_w \overrightarrow{\mathsf{close}}[x]; Q \equiv \overrightarrow{\mathsf{close}}[x]; (P \mid_w Q)$$

Case 1:

$$\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \qquad \dfrac{Q \triangleright \Psi_2 \mid \Delta_1 \vdash T \mid \Delta', w : C \vdash T'}{\overrightarrow{\mathsf{close}[x]}; Q \triangleright \Psi_2 \mid \cdot \vdash x : \bullet\mathbf{1} \mid \Delta_1, x : \bullet\mathbf{1} \vdash T \mid \Delta', w : C \vdash T'}\,\mathsf{1C}}{P \mid_w \overrightarrow{\mathsf{close}[x]}; Q \triangleright \Psi_1 \mid \Psi_2 \mid \cdot \vdash x : \bullet\mathbf{1} \mid \Delta_1, x : \bullet\mathbf{1} \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\,\mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \qquad Q \triangleright \Psi_2 \mid \Delta_1 \vdash T \mid \Delta', w : C \vdash T'}{P \mid_w Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1 \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\,\mathsf{Conn}}{\overrightarrow{\mathsf{close}[x]}; (P \mid_w Q) \triangleright \Psi_1 \mid \Psi_2 \mid \cdot \vdash x : \bullet\mathbf{1} \mid \Delta_1, x : \bullet\mathbf{1} \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\,\mathsf{Conn}$$

Case 2:

$$\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \qquad \dfrac{Q \triangleright \Psi_2 \mid \Delta', w : C \vdash T'}{\overrightarrow{\mathsf{close}[x]}; Q \triangleright \Psi_2 \mid \cdot \vdash x : \bullet\mathbf{1} \mid \Delta', x : \bullet\mathbf{1}, w : C \vdash T'}\,\mathsf{1C}}{P \mid_w \overrightarrow{\mathsf{close}[x]}; Q \triangleright \Psi_1 \mid \Psi_2 \mid \cdot \vdash x : \bullet\mathbf{1} \mid \Delta \vdash w : \bullet C \mid \Delta', x : \bullet\mathbf{1}, w : \bullet C \vdash T'}\,\mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \qquad Q \triangleright \Psi_2 \mid \Delta', w : C \vdash T'}{P \mid_w Q \triangleright \Psi_1 \mid \Psi_2 \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\,\mathsf{Conn}}{\overrightarrow{\mathsf{close}[x]}; (P \mid_w Q) \triangleright \Psi_1 \mid \Psi_2 \mid \cdot \vdash x : \bullet\mathbf{1} \mid \Delta \vdash w : \bullet C \mid \Delta', x : \bullet\mathbf{1}, w : \bullet C \vdash T'}\,\mathsf{Conn}$$

Commuting Conn with $\otimes$C

$$\boxed{[\mathsf{Conn}/\otimes\mathsf{C}/\mathsf{L}] \quad \overrightarrow{x(y)}; P \mid_w Q \equiv \overrightarrow{x(y)}; (P \mid_w Q) \qquad \Big(y \notin \mathsf{fn}(Q)\Big)}$$

Case 1:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2 \vdash x : \bullet B \mid \Delta_3, y : \bullet A, x : \bullet B \vdash T'' \mid \Delta_4 \vdash w : C}{\overrightarrow{x(y)}; P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash \bullet x : A \otimes B \mid \Delta_3, \bullet x : A \otimes B \vdash T'' \mid \Delta_4 \vdash w : C}\,\otimes\mathsf{C} \qquad Q \triangleright \Psi' \mid \Delta_5, w : C \vdash T'}{\overrightarrow{x(y)}; P \mid_w Q \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta_2 \vdash x : \bullet A \otimes B \mid \Delta_3, x : \bullet A \otimes B \vdash T'' \mid \Delta_4 \vdash w : \bullet C \mid \Delta_5, w : \bullet C \vdash T'}\,\mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2 \vdash x : \bullet B \mid \Delta_3, y : \bullet A, x : \bullet B \vdash T'' \mid \Delta_4 \vdash w : C \qquad Q \triangleright \Psi' \mid \Delta_5, w : C \vdash T'}{P \mid_w Q \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2 \vdash x : \bullet B \mid \Delta_3, y : \bullet A^y, x : \bullet B \vdash T'' \mid \Delta_4 \vdash w : \bullet C \mid \Delta_5, w : \bullet C \vdash T'}\,\mathsf{Conn}}{\overrightarrow{x(y)}; (P \mid_w Q) \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta_2 \vdash x : \bullet A \otimes B \mid \Delta_3, \bullet x : A \otimes B \vdash T'' \mid \Delta_4 \vdash w : \bullet C \mid \Delta_5, w : \bullet C \vdash T'}\,\otimes\mathsf{C}$$

Case 2:

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2 \vdash x : \bullet B \mid \Delta_3, y : \bullet A, x : \bullet B \vdash w : C}{\overrightarrow{x(y)}; P \triangleright \Psi \mid \Delta_1, \Delta_2 \vdash x : \bullet A \otimes B \mid \Delta_3, x : \bullet A \otimes B \vdash w : C}\,\otimes\mathsf{C} \qquad Q \triangleright \Psi' \mid \Delta_5, w : C \vdash T'}{\overrightarrow{x(y)}; P \mid_w Q \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta_2 \vdash x : \bullet A \otimes B \mid \Delta_3, x : \bullet A \otimes B \vdash w : \bullet C \mid \Delta_5, w : \bullet C \vdash T'}\,\mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright\ \Psi \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2 \vdash x : \bullet B \mid \Delta_3, y : \bullet A, x : \bullet B \vdash w : D \qquad Q \triangleright\ \Psi' \mid \Delta_5, w : D \vdash T'}{P \mid_w Q \quad \triangleright\ \Psi \mid \Psi' \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2 \vdash x : \bullet B \mid \Delta_3, y : \bullet A, x : \bullet B \vdash w : \bullet C \mid \Delta_5, w : \bullet C \vdash T'}\ \mathsf{Conn}}{\overrightarrow{x(y)}; (P \mid_w Q) \triangleright\ \Psi \mid \Psi' \mid \Delta_1, \Delta_2 \vdash x : \bullet A \otimes B \mid \Delta_3, x : \bullet A \otimes B \vdash w : \bullet C \mid \Delta_5, w : \bullet C \vdash T'}\ \otimes\mathsf{C}$$

$$[\mathsf{Conn}/\otimes\mathsf{C}/\mathsf{R}] \quad P \mid_w \overrightarrow{x(y)}; Q \equiv \overrightarrow{x(y)}; (P \mid_w Q) \qquad \Big(y \notin \mathsf{fn}(P)\Big)$$

Case 1:

$$\dfrac{P \triangleright\ \Psi' \mid \Delta_5 \vdash w : C \qquad \dfrac{Q \triangleright\ \Psi \mid \Delta_4, w : C \vdash T' \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2 \vdash x : \bullet B \mid \Delta_3, y : \bullet A, x : \bullet B \vdash T''}{\overrightarrow{x(y)}; Q \triangleright\ \Psi \mid \Delta_4, w : C \vdash T' \mid \Delta_1, \Delta_2 \vdash x : \bullet A \otimes B \mid \Delta_3, x : \bullet A \otimes B \vdash T''}\ \otimes\mathsf{C}}{P \mid_w \overrightarrow{x(y)}; Q \triangleright\ \Psi \mid \Psi' \mid \Delta_5 \vdash w : \bullet C \mid \Delta_4, w : \bullet C \vdash T' \mid \Delta_1, \Delta_2 \vdash x : \bullet A \otimes B \mid \Delta_3, x : \bullet A \otimes B \vdash T''}\ \mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright\ \Psi' \mid \Delta_5 \vdash w : C \qquad Q \triangleright\ \Psi \mid \Delta_4, w : C \vdash T' \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2 \vdash x : \bullet B \mid \Delta_3, y : \bullet A, x : \bullet B \vdash T''}{P \mid_w Q \triangleright\ \Psi \mid \Psi' \mid \Delta_5 \vdash w : \bullet C \mid \Delta_4, w : \bullet C \vdash T' \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2 \vdash x : \bullet B \mid \Delta_3, y : \bullet A, x : \bullet B \vdash T''}\ \mathsf{Conn}}{\overrightarrow{x(y)}; (P \mid_w Q) \triangleright\ \Psi \mid \Psi' \mid \Delta_5 \vdash w : \bullet C \mid \Delta_4, w : \bullet C \vdash T' \mid \Delta_1, \Delta_2 \vdash x : \bullet A \otimes B \mid \Delta_3, x : \bullet A \otimes B \vdash T''}\ \otimes\mathsf{C}$$

Case 2:

$$\dfrac{P \triangleright\ \Psi' \mid \Delta_5 \vdash w : C \qquad \dfrac{Q \triangleright\ \Psi \mid \Delta_1, w : C \vdash y : \bullet A \mid \Delta_2 \vdash x : \bullet B \mid \Delta_3, y : \bullet A, x : \bullet B \vdash T''}{\overrightarrow{x(y)}; Q \triangleright\ \Psi \mid \Delta_1, \Delta_2, w : C \vdash x : \bullet A \otimes B \mid \Delta_3, x : \bullet A \otimes B \vdash T''}\ \otimes\mathsf{C}}{P \mid_w \overrightarrow{x(y)}; Q \triangleright\ \Psi \mid \Psi' \mid \Delta_5 \vdash w : \bullet C \mid \Delta_1, \Delta_2, w : \bullet C \vdash x : \bullet A \otimes B \mid \Delta_3, x : \bullet A \otimes B \vdash T''}\ \mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright\ \Psi' \mid \Delta_5 \vdash w : C \qquad Q \triangleright\ \Psi \mid \Delta_1, w : C \vdash y : \bullet A \mid \Delta_2 \vdash x : \bullet B \mid \Delta_3, y : \bullet A, x : \bullet B \vdash T''}{P \mid_w Q \triangleright\ \Psi \mid \Psi' \mid \Delta_5 \vdash w : \bullet C \mid \Delta_1, w : \bullet C \vdash y : \bullet A \mid \Delta_2 \vdash x : \bullet B \mid \Delta_3, y : \bullet A, x : \bullet B \vdash T''}\ \mathsf{Conn}}{\overrightarrow{x(y)}; (P \mid_w Q) \triangleright\ \Psi \mid \Psi' \mid \Delta_5 \vdash w : \bullet C \mid \Delta_1, \Delta_2, w : \bullet C \vdash x : \bullet A \otimes B \mid \Delta_3, x : \bullet A \otimes B \vdash T''}\ \otimes\mathsf{C}$$

Case 3:

$$\dfrac{P \triangleright\ \Psi' \mid \Delta_5 \vdash w : C \qquad \dfrac{Q \triangleright\ \Psi \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, w : C \vdash x : \bullet B \mid \Delta_3, y : \bullet A, x : \bullet B \vdash T''}{\overrightarrow{x(y)}; Q \triangleright\ \Psi \mid \Delta_1, \Delta_2, w : C \vdash x : \bullet A \otimes B \mid \Delta_3, x : \bullet A \otimes B \vdash T''}\ \otimes\mathsf{C}}{P \mid_w \overrightarrow{x(y)}; Q \triangleright\ \Psi \mid \Psi' \mid \Delta_5 \vdash w : \bullet C \mid \Delta_1, \Delta_2, w : \bullet \vdash x : \bullet A \otimes B \mid \Delta_3, x : \bullet A \otimes B \vdash T''}\ \mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright\ \Psi' \mid \Delta_5 \vdash w : C \qquad Q \triangleright\ \Psi \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, w : C \vdash x : \bullet B \mid \Delta_3, y : \bullet A, x : \bullet B \vdash T''}{P \mid_w Q \triangleright\ \Psi \mid \Psi' \mid \Delta_5 \vdash w : \bullet C \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, w : \bullet C \vdash x : \bullet B \mid \Delta_3, y : \bullet A, x : \bullet B \vdash T''}\ \mathsf{Conn}}{\overrightarrow{x(y)}; (P \mid_w Q) \triangleright\ \Psi \mid \Psi' \mid \Delta_5 \vdash w : \bullet C \mid \Delta_1, \Delta_2, w : \bullet C \vdash x : \bullet A \otimes B \mid \Delta_3, x : \bullet A \otimes B \vdash T''}\ \otimes\mathsf{C}$$

Case 4:

$$\dfrac{\dfrac{P \triangleright\ \Psi' \mid \Delta_5 \vdash w : C \qquad Q \triangleright\ \Psi \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2 \vdash x : \bullet B \mid \Delta_3, w : C, y : \bullet A, x : \bullet B \vdash T''}{\overrightarrow{x(y)}; Q \triangleright\ \Psi \mid \Delta_1, \Delta_2 \vdash x : \bullet A \otimes B \mid \Delta_3, w : C, x : \bullet A \otimes B \vdash T''}\otimes\mathsf{C}}{P \mid_w \overrightarrow{x(y)}; Q \triangleright\ \Psi \mid \Psi' \mid \Delta_5 \vdash w : \bullet C \mid \Delta_1, \Delta_2 \vdash x : \bullet A \otimes B \mid \Delta_3, w : \bullet C, x : \bullet A \otimes B \vdash T''}\mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright\ \Psi' \mid \Delta_5 \vdash D \qquad Q \triangleright\ \Psi \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2 \vdash x : \bullet B \mid \Delta_3, w : C, y : \bullet A, x : \bullet B \vdash T''}{P \mid_w Q \triangleright\ \Psi \mid \Psi' \mid \Delta_5 \vdash w : \bullet C \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2 \vdash x : \bullet B \mid \Delta_3, w : \bullet C, y : \bullet A, x : \bullet B \vdash T''}\mathsf{Conn}}{\overrightarrow{x(y)}; (P \mid_w Q) \triangleright\ \Psi \mid \Psi' \mid \Delta_5 \vdash w : \bullet C \mid \Delta_1, \Delta_2 \vdash x : \bullet A \otimes B \mid \Delta_3, w : \bullet C, x : \bullet A \otimes B \vdash T''}\otimes\mathsf{C}$$

Commuting Conn with $\multimap$ C

$$\boxed{[\mathsf{Conn}/\multimap\mathsf{C}/\mathsf{L}] \quad \overrightarrow{x(y)}; P \mid_w Q \equiv \overrightarrow{x(y)}; (P \mid_w Q) \qquad \Big(y \notin \mathsf{fn}(Q)\Big)}$$

Case 1:

$$\dfrac{\dfrac{P \triangleright\ \Psi \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, x : \bullet B \vdash T'' \mid \Delta_3, y : \bullet A \vdash x : \bullet B \mid \Delta_4 \vdash w : C}{\overrightarrow{x(y)}; P \triangleright\ \Psi \mid \Delta_1, \Delta_2, x : \bullet A \multimap B \vdash T'' \mid \Delta_3 \vdash x : \bullet A \multimap B \mid \Delta_4 \vdash w : C}\multimap\mathsf{C} \qquad Q \triangleright\ \Psi' \mid \Delta_5, w : C \vdash T'}{\overrightarrow{x(y)}; P \mid_w Q \triangleright\ \Psi \mid \Psi' \mid \Delta_1, \Delta_2, x : \bullet A \multimap B \vdash T'' \mid \Delta_3 \vdash x : \bullet A \multimap B \mid \Delta_4 \vdash w : \bullet C \mid \Delta_5, w : \bullet C \vdash T'}\mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright\ \Psi \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, x : \bullet B \vdash T'' \mid \Delta_3, y : \bullet A \vdash x : \bullet B \mid \Delta_4 \vdash w : C \qquad Q \triangleright\ \Psi' \mid \Delta_5, w : C \vdash T'}{P \mid_w Q \triangleright\ \Psi \mid \Psi' \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, x : \bullet B \vdash T'' \mid \Delta_3, y : \bullet A \vdash x : \bullet B \mid \Delta_4 \vdash w : \bullet C \mid \Delta_5, w : \bullet C \vdash T'}\mathsf{Conn}}{\overrightarrow{x(y)}; (P \mid_w Q) \triangleright\ \Psi \mid \Psi' \mid \Delta_1, \Delta_2, x : \bullet A \multimap B \vdash T'' \mid \Delta_3 \vdash x : \bullet A \multimap B \mid \Delta_4 \vdash w : \bullet C \mid \Delta_5, w : \bullet C \vdash T'}\multimap\mathsf{C}$$

Case 2

$$\dfrac{\dfrac{P \triangleright\ \Psi \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, x : \bullet B \vdash D \mid \Delta_3, y : \bullet A \vdash x : \bullet B}{\overrightarrow{x(y)}; P \triangleright\ \Psi \mid \Delta_1, \Delta_2, x : \bullet A \multimap B \vdash w : C \mid \Delta_3 \vdash x : \bullet A \multimap B}\multimap\mathsf{C} \qquad Q \triangleright\ \Psi' \mid \Delta_5, w : C \vdash T'}{\overrightarrow{x(y)}; P \mid_w Q \triangleright\ \Psi \mid \Psi' \mid \Delta_1, \Delta_2, x : \bullet A \multimap B \vdash w : \bullet C \mid \Delta_3 \vdash x : \bullet A \multimap B \mid \Delta_5, w : \bullet C \vdash T'}\mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright\ \Psi \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, x : \bullet B \vdash w : C \mid \Delta_3, y : \bullet A \vdash x : \bullet B \qquad Q \triangleright\ \Psi' \mid \Delta_5, w; C \vdash T'}{P \mid_w Q \triangleright\ \Psi \mid \Psi' \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, x : \bullet B \vdash w : \bullet C \mid \Delta_3, y : \bullet A \vdash x : \bullet B \mid \Delta_5, w : \bullet C \vdash T'}\mathsf{Conn}}{\overrightarrow{x(y)}; (P \mid_w Q) \triangleright\ \Psi \mid \Psi' \mid \Delta_1, \Delta_2, x : \bullet A \multimap B \vdash w : \bullet C \mid \Delta_3 \vdash x : \bullet A \multimap B \mid \Delta_5, w : \bullet C \vdash T'}\multimap\mathsf{C}$$

$$[\mathsf{Conn}/\multimap \mathsf{C}/\mathsf{R}] \quad P \mid_w \overrightarrow{x(y)}; Q \equiv \overrightarrow{x(y)}; (P \mid_w Q) \qquad \Big(y \notin \mathsf{fn}(P)\Big)$$

Case 1:

$$\dfrac{P \triangleright \Psi' \mid \Delta_5 \vdash w : C \qquad \dfrac{Q \triangleright \Psi \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, x : \bullet B \vdash T'' \mid \Delta_3, y : \bullet A \vdash x : \bullet B \mid \Delta_4, w : C \vdash T'}{\overrightarrow{x(y)}; Q \triangleright \Psi \mid \Delta_1, \Delta_2, x : \bullet A \multimap B \vdash T'' \mid \Delta_3 \vdash x : \bullet A \multimap B \mid \Delta_4, w : C \vdash T'}\,\multimap \mathsf{C}}{P \mid_w \overrightarrow{x(y)}; Q \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta_2, x : \bullet A \multimap B \vdash T'' \mid \Delta_3 \vdash x : \bullet A \multimap B \mid \Delta_5 \vdash w : \bullet C \mid \Delta_4, w : \bullet C \vdash T'}\,\mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi' \mid \Delta_5 \vdash w : C \qquad Q \triangleright \Psi \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, x : \bullet B \vdash T'' \mid \Delta_3, y : \bullet A \vdash x : \bullet B \mid \Delta_4, w : C \vdash T'}{P \mid_w Q \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, x : \bullet B \vdash T'' \mid \Delta_3, y : \bullet A \vdash x : \bullet B \mid \Delta_4 \vdash w : \bullet C \mid \Delta_5, w : \bullet C \vdash T'}\,\mathsf{Conn}}{\overrightarrow{x(y)}; (P \mid_w Q) \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta_2, x : \bullet A \multimap B \vdash T'' \mid \Delta_3 \vdash x : \bullet A \multimap B \mid \Delta_5 \vdash w : \bullet C \mid \Delta_4, w : \bullet C \vdash T'}\,\multimap \mathsf{C}$$

Case 2:

$$\dfrac{P \triangleright \Psi' \mid \Delta_5 \vdash w : C \qquad \dfrac{Q \triangleright \Psi \mid \Delta_1, w : C \vdash y : \bullet A \mid \Delta_2, x : \bullet B \vdash T'' \mid \Delta_3, y : \bullet A \vdash x : \bullet B}{\overrightarrow{x(y)}; Q \triangleright \Psi \mid \Delta_1, \Delta_2, w : C, x : \bullet A \multimap B \vdash T'' \mid \Delta_3 \vdash x : \bullet A \multimap B}\,\multimap \mathsf{C}}{P \mid_w \overrightarrow{x(y)}; Q \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta_2, w : \bullet C, x : \bullet A \multimap B \vdash T'' \mid \Delta_3 \vdash x : \bullet A \multimap B \mid \Delta_5 \vdash w : \bullet C}\,\mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi' \mid \Delta_5 \vdash w : C \qquad Q \triangleright \Psi \mid \Delta_1, w : C \vdash y : \bullet A \mid \Delta_2, x : \bullet B \vdash T'' \mid \Delta_3, y : \bullet A \vdash x : \bullet B}{P \mid_w Q \triangleright \Psi \mid \Psi' \mid \Delta_1, w : \bullet C \vdash y : \bullet A \mid \Delta_2, x : \bullet B \vdash T'' \mid \Delta_3, y : \bullet A \vdash x : \bullet B \mid \Delta_5 \vdash w : C}\,\mathsf{Conn}}{\overrightarrow{x(y)}; (P \mid_w Q) \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta_2, w : \bullet C, x : \bullet A \multimap B \vdash T'' \mid \Delta_3 \vdash x : \bullet A \multimap B \mid \Delta_5 \vdash w : \bullet C}\,\multimap \mathsf{C}$$

Case 3:

$$\dfrac{P \triangleright \Psi' \mid \Delta_5 \vdash w : C \qquad \dfrac{Q \triangleright \Psi \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, w : C, x : \bullet B \vdash T'' \mid \Delta_3, y : \bullet A \vdash x : \bullet B}{\overrightarrow{x(y)}; Q \triangleright \Psi \mid \Delta_1, \Delta_2, w : C, x : \bullet A \multimap B \vdash T'' \mid \Delta_3 \vdash x : \bullet A \multimap B}\,\multimap \mathsf{C}}{P \mid_w \overrightarrow{x(y)}; Q \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta_2, w : \bullet C, x : \bullet A \multimap B \vdash T'' \mid \Delta_3 \vdash x : \bullet A \multimap B \mid \Delta_5 \vdash w : \bullet C}\,\mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi' \mid \Delta_5 \vdash w : C \qquad Q \triangleright \Psi \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, w : C, x : \bullet B \vdash T'' \mid \Delta_3, y : \bullet A \vdash x : \bullet B}{P \mid_w Q \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, w : \bullet C, x : \bullet B \vdash T'' \mid \Delta_3, y : \bullet A \vdash x : \bullet B \mid \Delta_5 \vdash w : \bullet C}\,\mathsf{Conn}}{\overrightarrow{x(y)}; (P \mid_w Q) \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta_2, w : \bullet C, x : \bullet A \multimap B \vdash T'' \mid \Delta_3 \vdash x : \bullet A \multimap B \mid \Delta_5 \vdash w : \bullet C}\,\multimap \mathsf{C}$$

Case 4:

$$\dfrac{P \triangleright \Psi' \mid \Delta_5 \vdash w : C \qquad \dfrac{Q \triangleright \Psi \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, x : \bullet B \vdash T'' \mid \Delta_3, w : C, y : \bullet A \vdash x : \bullet B}{\overrightarrow{x(y)}; Q \triangleright \Psi \mid \Delta_1, \Delta_2, x : \bullet A \multimap B \vdash T'' \mid \Delta_3, w : C \vdash x : \bullet A \multimap B}\,\multimap \mathsf{C}}{P \mid_w \overrightarrow{x(y)}; Q \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta_2, x : \bullet A \multimap B \vdash T'' \mid \Delta_3, w : \bullet C \vdash x : \bullet A \multimap B \mid \Delta_5 \vdash w : \bullet C}\,\mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi' \mid \Delta_5 \vdash w : C \qquad Q \triangleright \Psi \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, x : \bullet B \vdash T'' \mid \Delta_3, w; C, y : \bullet A \vdash x : \bullet B}{P \mid_w Q \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, x : \bullet B \vdash T'' \mid \Delta_3, w : \bullet C, y : \bullet A \vdash x : \bullet B \mid \Delta_5 \vdash w : \bullet C}\,\mathsf{Conn}}{\overrightarrow{x(y)}; (P \mid_w Q) \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta_2, x : \bullet A \multimap B \vdash T'' \mid \Delta_3, w : \bullet C \vdash x : \bullet A \multimap B \mid \Delta_5 \vdash w : \bullet C}\,\multimap \mathsf{C}$$

Commuting Conn with $\oplus$C

$$[\mathsf{Conn}/\oplus \mathsf{C}_1/\mathsf{L}] \quad \overrightarrow{x.\mathsf{I}}\ (P,Q) \mid_w R \equiv \overrightarrow{x.\mathsf{I}}\ ((P \mid_w R)\ ,\ (Q \mid_w R)) \qquad \Big(w \in \mathsf{fn}(P) \cap \mathsf{fn}(Q)\Big)$$

Case 1:

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1 \vdash x : A \mid \Delta_2, x : A \vdash T \mid \Delta \vdash w : C \quad Q \triangleright \Psi_2 \mid \Delta_2, x : B \vdash T \mid \Delta \vdash w : C}{\overrightarrow{x.\mathsf{I}}\ (P,Q) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1 \vdash x : A \oplus B \mid \Delta_2, x : A \oplus B \vdash T \mid \Delta \vdash w : C}\ \oplus\mathsf{C}_1 \quad R \triangleright \Psi_3 \mid \Delta', w : C \vdash T'}{\overrightarrow{x.\mathsf{I}}\ (P,Q) \mid_w R \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash x : A \oplus B \mid \Delta_2, x : A \oplus B \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\ \mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1 \vdash x : A \mid \Delta_2, x : A \vdash T \mid \Delta \vdash w : C \quad R \triangleright \Psi_3 \mid \Delta', w : C \vdash T'}{P \mid_w R \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash x : A \mid \Delta_2, x : A \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\ \mathsf{Conn} \quad \dfrac{Q \triangleright \Psi_2 \mid \Delta_2, x : B \vdash T \mid \Delta \vdash w : C \quad R \triangleright \Psi_3 \mid \Delta', w : C \vdash T'}{Q \mid_w R \triangleright \Psi_2 \mid \Psi_3 \mid \Delta_2, x : B \vdash T \mid \Delta \vdash w : \bullet C \Delta', w : \bullet C \vdash T'}\ \mathsf{Conn}}{\overrightarrow{x.\mathsf{I}}\ ((P \mid_w R)\ ,\ (Q \mid_w R)) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash x : A \oplus B \mid \Delta_2, x : A \oplus B \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\ \oplus\mathsf{C}_1$$

The other case is similar.

$$[\mathsf{Conn}/\oplus \mathsf{C}_1/\mathsf{R}] \quad P \mid_w \overrightarrow{x.\mathsf{I}}\ (Q,R) \equiv \overrightarrow{x.\mathsf{I}}\ ((P \mid_w Q)\ ,\ (P \mid_w R)) \qquad \Big(w \in \mathsf{fn}(Q) \cap \mathsf{fn}(R)\Big)$$

Case 1:

$$\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : D \qquad \dfrac{Q \triangleright \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash T \mid \Delta', w : D \vdash T' \quad R \triangleright \Psi_3 \mid \Delta_2, x : B \vdash T \mid \Delta', w : D \vdash T'}{\overrightarrow{x.\mathsf{I}}\ (Q,R) \triangleright \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash x : \bullet A \oplus B \mid \Delta_2, x : \bullet A \oplus B \vdash T \mid \Delta', w : D \vdash T'}\ \oplus\mathsf{C}_1}{P \mid_w \overrightarrow{x.\mathsf{I}}\ (Q,R) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash x : \bullet A \oplus B \mid \Delta_2, x : \bullet A \oplus B \vdash T \mid \Delta \vdash w : \bullet D \mid \Delta', w : \bullet D \vdash T'}\ \mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : D \quad Q \triangleright \Psi_2 \mid \Psi_3 \mid \Delta', w : D \vdash T' \mid \Delta_1 \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash T}{P \mid_w Q \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta \vdash w : \bullet D \mid \Delta', w : \bullet D \vdash T' \mid \Delta_1 \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash T}\ \mathsf{Conn} \quad \dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : D \quad R \triangleright \Psi_3 \mid \Delta_2, x : B \vdash T \mid \Delta', w : D \vdash T'}{P \mid_w R \triangleright \Psi_1 \mid \Psi_3 \mid \Delta \vdash w : \bullet D \mid \Delta', w : \bullet D \vdash T' \mid \Delta_2, x : B \vdash T}\ \mathsf{Conn}}{\overrightarrow{x.\mathsf{I}}\ ((P \mid_w Q)\ ,\ (P \mid_w R)) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta \vdash w : \bullet D \mid \Delta', w : \bullet D \vdash T' \mid \Delta_1 \vdash x : \bullet A \oplus B \mid \Delta_2, x : \bullet A \oplus B \vdash T}\ \oplus\mathsf{C}_1$$

The other cases are similar.

$$\boxed{[\mathsf{Conn}/\oplus \mathsf{C}_1/\mathsf{R}/\mathsf{L}] \quad P \mid_w \overrightarrow{x.\mathsf{l}}\ (Q, R) \equiv \overrightarrow{x.\mathsf{l}}\ ((P \mid_w Q)\ , R) \qquad \Big(w \in \mathsf{fn}(Q), w \notin \mathsf{fn}(R)\Big)}$$

Case 1:

$$\dfrac{\dfrac{P \triangleright \Psi'' \mid \Delta_4 \vdash w : D \qquad \overrightarrow{x.\mathsf{l}}\ (Q, R) \triangleright \Psi \mid \Psi' \mid \Delta_3, w : D \vdash T' \mid \Delta_1 \vdash x : \bullet A \oplus B \mid \Delta_2, x : \bullet A \oplus B \vdash T}{P \mid_w \overrightarrow{x.\mathsf{l}}\ (Q, R) \triangleright \Psi \mid \Psi' \mid \Psi'' \mid \Delta_4 \vdash w : \bullet D \mid \Delta_3, w : \bullet D \vdash T' \mid \Delta_1 \vdash x : \bullet A \oplus B \mid \Delta_2, x : \bullet A \oplus B \vdash T}\ \mathsf{Conn} \qquad R \triangleright \Psi' \mid \Delta_2, x : B \vdash T}{\overrightarrow{x.\mathsf{l}}\ ((P \mid_w Q)\ , R) \triangleright \Psi \mid \Psi' \mid \Psi'' \mid \Delta_4 \vdash w : \bullet D \mid \Delta_3, w : \bullet D \vdash T' \mid \Delta_1 \vdash x : \bullet A \oplus B \mid \Delta_2, x : \bullet A \oplus B \vdash T}\ \oplus\mathsf{C}_1$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi'' \mid \Delta_4 \vdash w : D \qquad Q \triangleright \Psi \mid \Psi' \mid \Delta_3, w : D \vdash T' \mid \Delta_1 \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash T}{P \mid_w Q \triangleright \Psi \mid \Psi' \mid \Psi'' \mid \Delta_4 \vdash w : \bullet D \mid \Delta_3, w : \bullet D \vdash T' \mid \Delta_1 \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash T}\ \mathsf{Conn} \qquad R \triangleright \Psi' \mid \Delta_2, x : B \vdash T}{\overrightarrow{x.\mathsf{l}}\ ((P \mid_w Q)\ , R) \triangleright \Psi \mid \Psi' \mid \Psi'' \mid \Delta_4 \vdash w : \bullet D \mid \Delta_3, w : \bullet D \vdash T' \mid \Delta_1 \vdash x : \bullet A \oplus B \mid \Delta_2, x : \bullet A \oplus B \vdash T}\ \oplus\mathsf{C}_1$$

The other cases are similar.

$$\boxed{[\mathsf{Conn}/\oplus \mathsf{C}_2/\mathsf{L}] \quad \overrightarrow{x.\mathsf{r}}\ (P, Q) \mid_w R \equiv \overrightarrow{x.\mathsf{r}}\ ((P \mid_w R)\ ,\ (Q \mid_w R)) \qquad \Big(w \in \mathsf{fn}(P) \cap \mathsf{fn}(Q)\Big)}$$

Similar to $[\mathsf{Conn}/\oplus \mathsf{C}_1/\mathsf{L}]$.

$$\boxed{[\mathsf{Conn}/\oplus \mathsf{C}_2/\mathsf{R}] \quad P \mid_w \overrightarrow{x.\mathsf{r}}\ (Q, R) \equiv \overrightarrow{x.\mathsf{r}}\ ((P \mid_w Q)\ ,\ (P \mid_w R)) \qquad \Big(w \in \mathsf{fn}(Q) \cap \mathsf{fn}(R)\Big)}$$

Similar to $[\mathsf{Conn}/\oplus \mathsf{C}_1/\mathsf{R}]$.

$$\boxed{[\mathsf{Conn}/\oplus \mathsf{C}_2/\mathsf{R}/\mathsf{R}] \quad P \mid_w \overrightarrow{x.\mathsf{r}}\ (Q, R) \equiv \overrightarrow{x.\mathsf{r}}\ (Q\ ,\ (P \mid_w R)) \qquad \Big(w \notin \mathsf{fn}(Q), w \in \mathsf{fn}(R)\Big)}$$

Similar to $[\mathsf{Conn}/\oplus \mathsf{C}_1/\mathsf{R}/\mathsf{L}]$.

Commuting Conn with &C

$$[\mathsf{Conn}/\&\mathsf{C}_1/\mathsf{L}] \quad \overrightarrow{x.\mathsf{I}}\ (P,Q) \mid_w R \equiv \overrightarrow{x.\mathsf{I}}\ ((P \mid_w R),\ Q) \qquad \Big(w \in \mathsf{fn}(P), w \notin \mathsf{fn}(Q)\Big)$$

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1 \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash w : C \quad Q \triangleright \Psi_2 \mid \Delta_1 \vdash x : \bullet B}{\overrightarrow{x.\mathsf{I}}\ (P,Q) \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1 \vdash x : \bullet A\&B \mid \Delta_2, x : \bullet A\&B \vdash w : C}\ \&\mathsf{C}_1 \qquad R \triangleright \Psi_3 \mid \Delta, w : C \vdash T}{\overrightarrow{x.\mathsf{I}}\ (P,Q) \mid_w R \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash x : \bullet A\&B \mid \Delta_2, x : \bullet A\&B \vdash w : \bullet C \mid \Delta, w : \bullet C \vdash T}\ \mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_1 \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash w : C \quad R \triangleright \Psi_3 \mid \Delta, w : C \vdash T}{P \mid_w R \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash w : \bullet C \mid \Delta, w : \bullet C \vdash T}\ \mathsf{Conn} \qquad Q \triangleright \Psi_2 \mid \Delta_1 \vdash x : \bullet B}{\overrightarrow{x.\mathsf{I}}\ ((P \mid_w R),\ Q) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash x : \bullet A\&B \mid \Delta_2, x : \bullet A\&B \vdash w : \bullet C \mid \Delta, w : \bullet C \vdash T}\ \&\mathsf{C}_1$$

$$[\mathsf{Conn}/\&\mathsf{C}_1/\mathsf{R}] \quad P \mid_w \overrightarrow{x.\mathsf{I}}\ (Q,R) \equiv \overrightarrow{x.\mathsf{I}}\ (P \mid_w Q, P \mid_w R) \qquad \Big(w \in \mathsf{fn}(Q) \cap \mathsf{fn}(R)\Big)$$

Case 1:

$$\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \qquad \dfrac{Q \triangleright \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash T \mid \Delta', w : C \vdash T' \quad R \triangleright \Psi_3 \mid \Delta_1 \vdash x : B \mid \Delta', w : C \vdash T'}{\overrightarrow{x.\mathsf{I}}\ (Q,R) \triangleright \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash x : \bullet A\&B \mid \Delta_2, x : \bullet A\&B \vdash T \mid \Delta', w : C \vdash T'}\ \&\mathsf{C}_1}{P \mid_w \overrightarrow{x.\mathsf{I}}\ (Q,R) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash x : \bullet A\&B \mid \Delta_2, x : \bullet A\&B \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\ \mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \quad Q \triangleright \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash T \mid \Delta', w : C \vdash T'}{P \mid_w Q \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\ \mathsf{Conn} \qquad \dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \quad R \triangleright \Psi_3 \mid \Delta_1 \vdash x : B \mid \Delta', w : C \vdash T'}{P \mid_w R \triangleright \Psi_1 \mid \Psi_3 \mid \Delta_1 \vdash x : B \mid \Delta \vdash w : \bullet C\Delta', w : \bullet C \vdash T'}\ \mathsf{Conn}}{\overrightarrow{x.\mathsf{I}}\ ((P \mid_w Q)\ ,\ (P \mid_w R)) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash x : \bullet A\&B \mid \Delta_2, x : \bullet A\&B \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\ \oplus\mathsf{C}_1$$

The other cases are similar.

$$[\mathsf{Conn}/\&\mathsf{C}_1/\mathsf{R}/\mathsf{L}] \quad P \mid_w \overrightarrow{x.\mathsf{I}}\ (Q,R) \equiv \overrightarrow{x.\mathsf{I}}\ ((P \mid_w Q)\ ,\ R) \qquad \Big(w \in \mathsf{fn}(Q), w \notin \mathsf{fn}(R)\Big)$$

Case 1:

$$\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \qquad \dfrac{Q \triangleright \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash T \mid \Delta', w : C \vdash T' \quad R \triangleright \Psi_3 \mid \Delta_1 \vdash x : B \mid}{\overrightarrow{x.\mathsf{l}}\ (Q, R) \triangleright \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash x : \bullet A \& B \mid \Delta_2, x : \bullet A \& B \vdash T \mid \Delta', w : C \vdash T'}\ \&\mathsf{C}_1}{P \mid_w \overrightarrow{x.\mathsf{l}}\ (Q, R) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash x : \bullet A \& B \mid \Delta_2, x : \bullet A \& B \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\ \mathsf{Conn}$$

is equivalent to ($\equiv$)

$$\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta \vdash w : C \quad Q \triangleright \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash T \mid \Delta', w : C \vdash T'}{P \mid_w Q \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\ \mathsf{Conn} \qquad R \triangleright \Psi_3 \mid \Delta_1 \vdash x : B}{\overrightarrow{x.\mathsf{l}}\ ((P \mid_w Q),\ R) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash x : \bullet A \& B \mid \Delta_2, x : \bullet A \& B \vdash T \mid \Delta \vdash w : \bullet C \mid \Delta', w : \bullet C \vdash T'}\ \oplus\mathsf{C}_1$$

The other cases are similar.

$$[\mathsf{Conn}/\&\mathsf{C}_2/\mathsf{L}] \quad \overrightarrow{x.\mathsf{r}}\ (P, Q) \mid_w R \equiv \overrightarrow{x.\mathsf{r}}\ (P\ ,\ (Q \mid_w R)) \qquad \Big(w \notin \mathsf{fn}(P), w \in \mathsf{fn}(Q)\Big)$$

Similar to $[\mathsf{Conn}/\&\mathsf{C}_1/\mathsf{L}]$.

$$[\mathsf{Conn}/\&\mathsf{C}_2/\mathsf{R}] \quad P \mid_w \overrightarrow{x.\mathsf{r}}\ (Q, R) \equiv \overrightarrow{x.\mathsf{r}}\ (P \mid_w Q, P \mid_w R) \qquad \Big(w \in \mathsf{fn}(Q) \cap \mathsf{fn}(R)\Big)$$

Similar to $[\mathsf{Conn}/\&\mathsf{C}_1/\mathsf{R}]$.

$$[\mathsf{Conn}/\&\mathsf{C}_2/\mathsf{R}/\mathsf{L}] \quad P \mid_w \overrightarrow{x.\mathsf{r}}\ (Q, R) \equiv \overrightarrow{x.\mathsf{r}}\ (P\ ,\ (Q \mid_w R)\ ) \qquad \Big(w \notin \mathsf{fn}(Q), w \in \mathsf{fn}(Q)\Big)$$

Similar to $[\mathsf{Conn}/\&\mathsf{C}_1/\mathsf{R}/\mathsf{L}]$.

C.2 Reductions

Reductions for the Action Fragment

$$[\beta_{\mathbf{1}}] \qquad (\boldsymbol{\nu} x)\,(\mathsf{close}[x] \mid_x \mathsf{wait}[x]; Q) \xrightarrow{x} Q$$

$$\dfrac{\dfrac{}{\mathsf{close}[x] \triangleright \cdot \vdash \mathbf{1}}\,\mathsf{1R} \quad \dfrac{Q \triangleright \Psi \mid \Delta \vdash T}{\mathsf{wait}[x]; Q \triangleright \Psi \mid \Delta, \mathbf{1} \vdash T}\,\mathsf{1L}}{\dfrac{\mathsf{close}[x] \mid_x \mathsf{wait}[x]; Q \triangleright \Psi \mid \cdot \vdash x : \bullet\mathbf{1} \mid \Delta, x : \bullet\mathbf{1} \vdash T}{(\boldsymbol{\nu} x)\,(\mathsf{close}[x] \mid_x \mathsf{wait}[x]; Q) \triangleright \Psi \mid \Delta \vdash T}\,\mathsf{Scope}^{x}}\,\mathsf{Conn}^{x}$$

$$\xrightarrow{x}$$

$$Q \triangleright \Psi \mid \Delta \vdash T$$

$$[\beta_{\otimes}] \qquad (\boldsymbol{\nu} x)\,(\overline{x}(y); (P \mid Q) \mid_x x(y); R) \xrightarrow{x} (\boldsymbol{\nu} y)\,(\boldsymbol{\nu} x)\,\left(P \mid_y (Q \mid_x R)\right)$$

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

$$\xrightarrow{x}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{P \triangleright \Psi_1 \mid \Delta_1 \vdash y : A} \quad \dfrac{Q \triangleright \Psi_2 \mid \Delta_2 \vdash x : B \quad R \triangleright \Psi_3 \mid \Delta_3, y : A, x : B \vdash T}{Q \mid_x R \triangleright \Psi_2 \mid \Psi_3 \mid \Delta_2 \vdash x : \bullet B \mid \Delta_3, y : A, x : \bullet B \vdash T}\,\mathsf{Conn}^{x}}{P \mid_y (Q \mid_x R) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2 \vdash x : \bullet B \mid \Delta_3, y : \bullet A, x : \bullet B \vdash T}\,\mathsf{Conn}^{y}}{(\boldsymbol{\nu} x)\,(P \mid_y (Q \mid_x R)) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, \Delta_3, y : \bullet A \vdash T}\,\mathsf{Scope}^{x}}{(\boldsymbol{\nu} y)\,(\boldsymbol{\nu} x)\,(P \mid_y (Q \mid_x R)) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1, \Delta_2, \Delta_3 \vdash T}\,\mathsf{Scope}^{y}$$

$$[\beta_{\multimap}] \qquad (\boldsymbol{\nu} x)\,(x(y); P \mid_x \overline{x}(y); (Q \mid R)) \xrightarrow{x} (\boldsymbol{\nu} x)\,(\boldsymbol{\nu} y)\,\left((Q \mid_y P) \mid_x R\right)$$

$$\dfrac{\dfrac{\dfrac{\dfrac{P \triangleright \Psi_1 \mid \Delta_1, y : A \vdash x : B}{x(y); P \triangleright \Psi_1 \mid \Delta_1 \vdash x : A \multimap B}\multimap \mathsf{R} \quad \dfrac{Q \triangleright \Psi_2 \mid \Delta_2 \vdash y : A \quad R \triangleright \Psi_3 \mid \Delta_3, x : B \vdash T}{\overline{x}(y); (Q \mid R) \triangleright \Psi_2 \mid \Psi_3 \mid \Delta_2, \Delta_3, x : A \multimap B \vdash T}\multimap \mathsf{L}}{x(y); P \mid_x \overline{x}(y); (Q \mid R) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1, x : \bullet A \multimap B \vdash T \mid \Delta_2, \Delta_3 \vdash x : \bullet A \multimap B}\mathsf{Conn}^x}{(\boldsymbol{\nu} x)\,(x(y); P \mid_x \overline{x}(y); (Q \mid R)) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1, \Delta_2, \Delta_3 \vdash T}\mathsf{Scope}^x$$

$$\xrightarrow{x}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{Q \triangleright \Psi_2 \mid \Delta_2 \vdash y : A \quad P \triangleright \Psi_1 \mid \Delta_1, y : A \vdash x : B}{Q \mid_y P \triangleright \Psi_1 \mid \Psi_2 \mid \Delta_2 \vdash y : \bullet A \mid \Delta_1, y : \bullet A \vdash x : B}\mathsf{Conn}^y \quad R \triangleright \Psi_3 \mid \Delta_3, x : B \vdash T}{(Q \mid_y P) \mid_x R \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_2 \vdash y : \bullet A \mid \Delta_1, y : \bullet A \vdash x : \bullet B \mid \Delta_3, x : \bullet B \vdash T}\mathsf{Conn}_x}{(\boldsymbol{\nu} y)\,((Q \mid_y P) \mid_x R) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1, \Delta_2 \vdash x : \bullet B \mid \Delta_3, x : \bullet B \vdash T}\mathsf{Scope}^y}{(\boldsymbol{\nu} x)\,(\boldsymbol{\nu} y)\,((Q \mid_y P) \mid_x R) \triangleright \Psi_1 \mid \Psi_2 \mid \Psi_3 \mid \Delta_1, \Delta_2, \Delta_3 \vdash T}\mathsf{Scope}^x$$

$$[\beta_{\&_1}] \qquad (\boldsymbol{\nu} x)\,(x.\mathsf{case}(P, Q) \mid_x x.\mathsf{inl}; R) \xrightarrow{x} (\boldsymbol{\nu} x)\,(P \mid_x R)$$

$$\dfrac{\dfrac{\dfrac{P \triangleright \Psi \mid \Delta \vdash x : A \quad Q \triangleright \Psi \mid \Delta \vdash x : B}{x.\mathsf{case}(P, Q) \triangleright \Psi \mid \Delta \vdash x : A \& B}\&\mathsf{R} \quad \dfrac{R \triangleright \Psi' \mid \Delta', x : A \vdash T}{x.\mathsf{inl}; R \triangleright \Psi' \mid \Delta', x : A \& B \vdash T}\&\mathsf{L}_1}{x.\mathsf{case}(P, Q) \mid_x x.\mathsf{inl}; R \triangleright \Psi \mid \Psi' \mid \Delta \vdash x : \bullet A \& B \mid \Delta', x : \bullet A \& B \vdash T}\mathsf{Conn}^x}{(\boldsymbol{\nu} x)\,(x.\mathsf{case}(P, Q) \mid_x x.\mathsf{inl}; R) \triangleright \Psi \mid \Psi' \mid \Delta, \Delta' \vdash T}\mathsf{Scope}^x$$

$$\xrightarrow{x}$$

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

$$[\beta_{\&_2}] \qquad (\boldsymbol{\nu} x)\,(x.\mathsf{case}(P, Q) \mid_x x.\mathsf{inr}; R) \xrightarrow{x} (\boldsymbol{\nu} x)\,(Q \mid_x R)$$

$$\dfrac{\dfrac{\dfrac{P \triangleright \Psi \mid \Delta \vdash x : A \quad Q \triangleright \Psi \mid \Delta \vdash x : B}{x.\mathsf{case}(P, Q) \triangleright \Psi \mid \Delta \vdash x : A \& B}\&\mathsf{R} \quad \dfrac{R \triangleright \Psi' \mid \Delta', x : B \vdash T}{x.\mathsf{inr}; R \triangleright \Psi' \mid \Delta', x : A \& B \vdash T}\&\mathsf{L}_2}{x.\mathsf{case}(P, Q) \mid_x x.\mathsf{inr}; R \triangleright \Psi \mid \Psi' \mid \Delta \vdash x : \bullet A \& B \mid \Delta', x : \bullet A \& B \vdash T}\mathsf{Conn}^x}{(\boldsymbol{\nu} x)\,(x.\mathsf{case}(P, Q) \mid_x x.\mathsf{inr}; R) \triangleright \Psi \mid \Psi' \mid \Delta, \Delta' \vdash T}\mathsf{Scope}^x$$

$$\xrightarrow{x}$$

$$\dfrac{\dfrac{Q \triangleright \Psi \mid \Delta \vdash x : B \quad R \triangleright \Psi' \mid \Delta', x : B \vdash T}{Q \mid_x R \triangleright \Psi \mid \Psi' \mid \Delta \vdash x : \bullet B \mid \Delta', x : \bullet B \vdash T}\mathsf{Conn}^x}{(\boldsymbol{\nu} x)\,(Q \mid_x R) \triangleright \Psi \mid \Psi' \mid \Delta, \Delta' \vdash T}\mathsf{Scope}^x$$

$$[\beta_{\oplus_1}] \qquad (\boldsymbol{\nu} x)\ (x.\mathsf{inl}; P \mid_x x.\mathsf{case}(Q, R)) \xrightarrow{x} (\boldsymbol{\nu} x)\ (P \mid_w Q)$$

$$\dfrac{\dfrac{\dfrac{P \triangleright \Psi \mid \Delta \vdash x : A}{x.\mathsf{inl}; P \triangleright \Psi \mid \Delta \vdash x : A \oplus B}\ \oplus\mathsf{R}_1 \quad \dfrac{Q \triangleright \Psi' \mid \Delta', x : A \vdash T \quad R \triangleright \Psi' \mid \Delta', x : B \vdash T}{x.\mathsf{case}(Q, R) \triangleright \Psi' \mid \Delta', x : A \oplus B \vdash T}\ \oplus\mathsf{L}}{x.\mathsf{inl}; P \mid_x x.\mathsf{case}(Q, R) \triangleright \Psi \mid \Psi' \mid \Delta \vdash x : \bullet A \oplus B \mid \Delta', x : \bullet A \oplus B \vdash T}\ \mathsf{Conn}^x}{(\boldsymbol{\nu} x)\ (x.\mathsf{inl}; P \mid_x x.\mathsf{case}(Q, R)) \triangleright \Psi \mid \Psi' \mid \Delta, \Delta' \vdash T}\ \mathsf{Scope}^x$$

$$\xrightarrow{x}$$

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta \vdash x : A \quad Q \triangleright \Psi' \mid \Delta', x : A \vdash T}{P \mid_w Q \triangleright \Psi \mid \Psi' \mid \Delta \vdash x : \bullet A \mid \Delta', x : \bullet A \vdash T}\ \mathsf{Conn}^x}{(\boldsymbol{\nu} x)\ (P \mid_w Q) \triangleright \Psi \mid \Psi' \mid \Delta, \Delta' \vdash T}\ \mathsf{Scope}^x$$

$$[\beta_{\oplus_2}] \qquad (\boldsymbol{\nu} x)\ (x.\mathsf{inr}; P \mid_x x.\mathsf{case}(Q, R)) \xrightarrow{x} (\boldsymbol{\nu} x)\ (P \mid_x R)$$

$$\dfrac{\dfrac{\dfrac{P \triangleright \Psi \mid \Delta \vdash x : B}{x.\mathsf{inr}; P \triangleright \Psi \mid \Delta \vdash x : A \oplus B}\ \oplus\mathsf{R}_2 \quad \dfrac{Q \triangleright \Psi' \mid \Delta', x : A \vdash T \quad R \triangleright \Psi' \mid \Delta', x : B \vdash T}{x.\mathsf{case}(Q, R) \triangleright \Psi' \mid \Delta', x : A \oplus B \vdash T}\ \oplus\mathsf{L}}{x.\mathsf{inr}; P \mid_x x.\mathsf{case}(Q, R) \triangleright \Psi \mid \Psi' \mid \Delta \vdash x : \bullet A \oplus B \mid \Delta', x : \bullet A \oplus B \vdash T}\ \mathsf{Conn}^x}{(\boldsymbol{\nu} x)\ (x.\mathsf{inr}; P \mid_x x.\mathsf{case}(Q, R)) \triangleright \Psi \mid \Psi' \mid \Delta, \Delta' \vdash T}\ \mathsf{Scope}^x$$

$$\xrightarrow{x}$$

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta \vdash x : B \quad R \triangleright \Psi' \mid \Delta', x : B \vdash T}{P \mid_x R \triangleright \Psi \mid \Psi' \mid \Delta \vdash x : \bullet B \mid \Delta', x : \bullet B \vdash T}\ \mathsf{Conn}^x}{(\boldsymbol{\nu} x)\ (P \mid_x R) \triangleright \Psi \mid \Psi' \mid \Delta, \Delta' \vdash T}\ \mathsf{Scope}^x$$

Reductions for the Interaction Fragment

$$[\beta_{1\mathsf{C}}] \qquad (\boldsymbol{\nu} x)\ \overrightarrow{\mathsf{close}}[x]; P \xrightarrow{\bullet x} P$$

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta \vdash T}{\overrightarrow{\mathsf{close}}[x]; P \triangleright \Psi \mid \cdot \vdash x : \bullet\mathbf{1} \mid \Delta, x : \bullet\mathbf{1} \vdash T}\ 1\mathsf{C}^x}{(\boldsymbol{\nu} x)\ \overrightarrow{\mathsf{close}}[x]; P \triangleright \Psi \mid \Delta \vdash T}\ \mathsf{Scope}^x \qquad \xrightarrow{\bullet x} \qquad P \triangleright \Psi \mid \Delta \vdash T$$

$$[\beta_{\otimes\mathsf{C}}] \qquad (\boldsymbol{\nu} x)\ \overrightarrow{x(y)}; P \xrightarrow{\bullet x} (\boldsymbol{\nu} y)\ (\boldsymbol{\nu} x)\ P$$

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

$$\xrightarrow{\bullet x}$$

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash x : \bullet B \mid \Delta_2 \vdash y : \bullet A \mid \Delta_3, y : \bullet A, x : \bullet B \vdash T}{(\boldsymbol{\nu} x)\ P \triangleright \Psi \mid \Delta_2 \vdash y : \bullet A \mid \Delta_1, \Delta_3, y : \bullet A \vdash T}\ \mathsf{Scope}^x}{(\boldsymbol{\nu} y)\ (\boldsymbol{\nu} x)\ P \triangleright \Psi \mid \Delta_1, \Delta_2, \Delta_3 \vdash T}\ \mathsf{Scope}^y$$

$$[\beta_{\multimap \mathsf{C}}] \qquad (\boldsymbol{\nu} x)\ \overrightarrow{x(y)}; P \xrightarrow{\bullet x} (\boldsymbol{\nu} y)\ (\boldsymbol{\nu} x)\ P$$

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

$$\xrightarrow{\bullet x}$$

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, x : \bullet B \vdash T \mid \Delta_3, y : \bullet A \vdash x : \bullet B}{(\boldsymbol{\nu} x)\ P \triangleright \Psi \mid \Delta_1 \vdash y : \bullet A \mid \Delta_2, \Delta_3, y : \bullet A \vdash T}\ \mathsf{Scope}^x}{(\boldsymbol{\nu} y)\ (\boldsymbol{\nu} x)\ P \triangleright \Psi \mid \Delta_1, \Delta_2, \Delta_3 \vdash T}\ \mathsf{Scope}^y$$

$$[\beta_{\&\mathsf{C}_1}] \qquad (\boldsymbol{\nu} x)\ \overrightarrow{x.\mathsf{l}}\ (P, Q) \xrightarrow{\bullet x} (\boldsymbol{\nu} x)\ P$$

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash T \quad Q \triangleright \Psi' \mid \Delta_1 \vdash x : B}{\overrightarrow{x.\mathsf{l}}\ (P, Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \& B \mid \Delta_2, x : \bullet A \& B \vdash T}\ \&\mathsf{C}_1^x}{(\boldsymbol{\nu} x)\ \overrightarrow{x.\mathsf{l}}\ (P, Q) \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta_2 \vdash T}\ \mathsf{Scope}^x$$

$$\xrightarrow{\bullet x}$$

$$\dfrac{P \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash T}{(\boldsymbol{\nu} x)\ P \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta_2 \vdash T}\ \mathsf{Scope}^x$$

$$[\beta_{\&\mathsf{C}_2}] \qquad (\boldsymbol{\nu} x)\ \overrightarrow{x.\mathsf{r}}\ (P, Q) \xrightarrow{x} (\boldsymbol{\nu} x)\ Q$$

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_1 \vdash x : A \quad Q \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet B \mid \Delta_2, x : \bullet B \vdash T}{\overrightarrow{x.\mathsf{r}}\ (P, Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \& B \mid \Delta_2, x : \bullet A \& B \vdash T}\ \&\mathsf{C}_2^x}{(\boldsymbol{\nu} x)\ \overrightarrow{x.\mathsf{r}}\ (P, Q) \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta_2 \vdash T}\ \mathsf{Scope}^x$$

$$\xrightarrow{\bullet x}$$

$$\dfrac{Q \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet B \mid \Delta_2, x : \bullet B \vdash T}{(\boldsymbol{\nu} x)\ Q \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta_2 \vdash T}\ \mathsf{Scope}^x$$

$$[\beta_{\oplus \mathsf{C}_1}] \qquad (\boldsymbol{\nu} x)\ \overrightarrow{x.\mathsf{l}}\ (P, Q) \xrightarrow{x} (\boldsymbol{\nu} x)\ P$$

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash T \quad Q \triangleright \Psi' \mid \Delta_2, x : B \vdash T}{\overrightarrow{x.\mathsf{l}}\ (P, Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \oplus B \mid \Delta_2, x : \bullet A \oplus B \vdash T}\ \oplus \mathsf{C}_1^x}{(\boldsymbol{\nu} x)\ (\overrightarrow{x.\mathsf{l}}\ (P, Q)) \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta_2 \vdash T}\ \mathsf{Scope}^x$$

$$\xrightarrow{\bullet x}$$

$$\dfrac{P \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \mid \Delta_2, x : \bullet A \vdash T}{(\boldsymbol{\nu} x)\ P \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta_2 \vdash T}\ \mathsf{Scope}^x$$

$$[\beta_{\oplus \mathsf{C}_2}] \qquad (\boldsymbol{\nu} x)\ \overrightarrow{x.\mathsf{r}}\ (P, Q) \xrightarrow{x} (\boldsymbol{\nu} x)\ Q$$

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta_2, x : A \vdash T \quad Q \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet B \mid \Delta_2, x : \bullet B \vdash T}{\overrightarrow{x.\mathsf{r}}\ (P, Q) \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet A \oplus B \mid \Delta_2, x : \bullet A \oplus B \vdash T}\ \oplus \mathsf{C}_2^x}{(\boldsymbol{\nu} x)\ \ \overrightarrow{x.\mathsf{r}}\ (P, Q) \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta_2 \vdash T}\ \mathsf{Scope}^x$$

$$\xrightarrow{\bullet x}$$

$$\dfrac{Q \triangleright \Psi \mid \Psi' \mid \Delta_1 \vdash x : \bullet B \mid \Delta_2, x : \bullet B \vdash T}{(\boldsymbol{\nu} x)\ Q \triangleright \Psi \mid \Psi' \mid \Delta_1, \Delta_2 \vdash T}\ \mathsf{Scope}^x$$

C.3 Abstraction and Concretisation

$$[\alpha\gamma_{\mathbf{1}}] \qquad \mathsf{close}[x] \mid_x \mathsf{wait}[x]; P \overset{x}{\dashrightarrow} \overrightarrow{\mathsf{close}[x]; P}$$

$$\dfrac{\dfrac{}{\mathsf{close}[x] \triangleright \cdot \vdash x : \mathbf{1}}\,\mathsf{1R} \quad \dfrac{P \triangleright \Psi \mid \Delta \vdash T}{\mathsf{wait}[x]; P \triangleright \Psi \mid \Delta, x : \mathbf{1} \vdash T}\,\mathsf{1L}}{\mathsf{close}[x] \mid_x \mathsf{wait}[x]; P \triangleright \Psi \mid \cdot \vdash x : \bullet\mathbf{1} \mid \Delta, x : \bullet\mathbf{1} \vdash T}\,\mathsf{Conn}$$

$$\overset{x}{\dashleftarrow}$$

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

$$[\alpha\gamma_{\otimes}] \qquad \overline{x}(y); (P \mid Q) \mid_x x(y); R \overset{x}{\dashrightarrow} \overrightarrow{x(y); \left(P \mid_y (Q \mid_x R)\right)}$$

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

$$\overset{x}{\dashleftarrow}$$

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

$$[\alpha\gamma_{\multimap}] \qquad x(y); P \mid_x \overline{x}(y); (Q \mid R) \overset{x}{\dashrightarrow} \overrightarrow{x(y); \left((Q \mid_y P) \mid_x R\right)}$$

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

$$\overset{x}{\dashleftarrow}$$

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

$$[\alpha\gamma_{\&_1}] \qquad x.\mathsf{case}(P,Q) \mid_x x.\mathsf{inl}; R \overset{x}{\underset{\leftarrow}{\dashrightarrow}} \overrightarrow{x.\mathsf{l}}\ ((P \mid_x R), Q)$$

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta \vdash x : A \quad Q \triangleright \Psi \mid \Delta \vdash x : B}{x.\mathsf{case}(P,Q) \triangleright \Psi \mid \Delta \vdash x : A\&B}\ \&\mathsf{R} \quad \dfrac{R \triangleright \Psi' \mid \Delta', x : A \vdash T}{x.\mathsf{inl}; R \triangleright \Psi' \mid \Delta', x : A\&B \vdash T}\ \&\mathsf{L}_1}{x.\mathsf{case}(P,Q) \mid_x x.\mathsf{inl}; R \triangleright \Psi \mid \Psi' \mid \Delta \vdash x : \bullet A\&B \mid \Delta', x : \bullet A\&B \vdash T}\ \mathsf{Conn}^x$$

$$\overset{x}{\underset{\leftarrow}{\dashrightarrow}}$$

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta \vdash x : A \quad R \triangleright \Psi' \mid \Delta', x : A \vdash T}{P \mid_x R \triangleright \Psi \mid \Psi' \mid \Delta \vdash x : \bullet A \mid \Delta', x : \bullet A \vdash T}\ \mathsf{Conn}^x \quad Q \triangleright \Psi \mid \Delta \vdash x : B}{\overrightarrow{x.\mathsf{l}}\ ((P \mid_x R), Q) \triangleright \Psi \mid \Psi' \mid \Delta \vdash x : \bullet A\&B \mid \Delta', x : \bullet A\&B \vdash T}\ \&\mathsf{C}_1^x$$

$$[\alpha\gamma_{\&_2}] \qquad x.\mathsf{case}(P,Q) \mid_x x.\mathsf{inr}; R \overset{x}{\underset{\leftarrow}{\dashrightarrow}} \overrightarrow{x.\mathsf{r}}\ (P\ ,\ Q \mid_x R)$$

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta \vdash x : A \quad \triangleright Q\,\Psi \mid \Delta \vdash x : B}{x.\mathsf{case}(P,Q) \triangleright \Psi \mid \Delta \vdash x : A\&B}\ \&\mathsf{R} \quad \dfrac{R \triangleright \Psi' \mid \Delta', x : B \vdash T}{x.\mathsf{inr}; R \triangleright \Psi' \mid \Delta', x : A\&B \vdash T}\ \&\mathsf{L}_2}{x.\mathsf{case}(P,Q) \mid_x x.\mathsf{inr}; R \triangleright \Psi \mid \Psi' \mid \Delta \vdash x : \bullet A\&B \mid \Delta', x : \bullet A\&B \vdash T}\ \mathsf{Conn}^x$$

$$\overset{x}{\underset{\leftarrow}{\dashrightarrow}}$$

$$\dfrac{P \triangleright \Psi \mid \Delta \vdash x : A \quad \dfrac{Q \triangleright \Psi \mid \Delta \vdash x : B \quad R \triangleright \Psi' \mid \Delta', x : B \vdash T}{Q \mid_x R \triangleright \Psi \mid \Psi' \mid \Delta \vdash x : \bullet B \mid \Delta', x : \bullet B \vdash T}\ \mathsf{Conn}^x}{\overrightarrow{x.\mathsf{r}}\ (P\ ,\ (Q \mid_x R)) \triangleright \Psi \mid \Psi' \mid \Delta \vdash \bullet x : A\&B \mid \Delta', \bullet x : A\&B \vdash T}\ \&\mathsf{C}_2^x$$

$$[\alpha\gamma_{\oplus_1}] \qquad x.\mathsf{inl}; P \mid_x x.\mathsf{case}(Q,R) \overset{x}{\underset{\leftarrow}{\dashrightarrow}} \overrightarrow{x.\mathsf{l}}\ ((P \mid_x Q)\ ,\ R)$$

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta \vdash x : A}{x.\mathsf{inl}; P \triangleright \Psi \mid \Delta \vdash x : A \oplus B}\ \oplus\mathsf{R}_1 \quad \dfrac{Q \triangleright \Psi' \mid \Delta', x : A \vdash T \quad R \triangleright \Psi' \mid \Delta', x : B \vdash T}{x.\mathsf{case}(Q,R) \triangleright \Psi' \mid \Delta', x : A \oplus B \vdash T}\ \oplus\mathsf{L}}{x.\mathsf{inl}; P \mid_x x.\mathsf{case}(Q,R) \triangleright \Psi \mid \Psi' \mid \Delta \vdash x : \bullet A \oplus B \mid \Delta', x : \bullet A \oplus B \vdash T}\ \mathsf{Conn}^x$$

$$\overset{x}{\underset{\leftarrow}{\dashrightarrow}}$$

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta \vdash x : A \quad Q \triangleright \Psi' \mid \Delta', x : A \vdash T}{P \mid_x Q \triangleright \Psi \mid \Psi' \mid \Delta \vdash x : \bullet A \mid \Delta', x : \bullet A \vdash T}\ \mathsf{Conn}^x \quad R \triangleright \Psi' \mid \Delta', x : B \vdash T}{\overrightarrow{x.\mathsf{l}}\ ((P \mid_x Q)\ ,\ R) \triangleright \Psi \mid \Psi' \mid \Delta \vdash x : \bullet A \oplus B \mid \Delta', x : \bullet A \oplus B \vdash T}\ \oplus\mathsf{C}_1^x$$

$[\alpha\gamma_{\oplus_2}]$ $\quad x.\mathsf{inr}; P \mid_x x.\mathsf{case}(Q, R) \overset{x}{\dashrightarrow} \overrightarrow{x.\mathsf{r}}\ (Q\ ,\ (P \mid_x R))$

$$\dfrac{\dfrac{P \triangleright \Psi \mid \Delta \vdash x : B}{x.\mathsf{inr}; P \triangleright \Psi \mid \Delta \vdash x : A \oplus B}\oplus\mathsf{R}_2 \quad \dfrac{Q \triangleright \Psi' \mid \Delta', x : A \vdash T \quad R \triangleright \Psi' \mid \Delta', x : B \vdash T}{x.\mathsf{case}(Q, R) \triangleright \Psi' \mid \Delta', x : A \oplus B \vdash T}\oplus\mathsf{L}}{x.\mathsf{inr}; P \mid_x x.\mathsf{case}(Q, R) \triangleright \Psi \mid \Psi' \mid \Delta \vdash x : \bullet A \oplus B \mid \Delta', x : \bullet A \oplus B \vdash T}\ \mathsf{Conn}^x$$

$$\overset{x}{\dashrightarrow}$$

$$\dfrac{Q \triangleright \Psi' \mid \Delta', x : A \vdash T \quad \dfrac{P \triangleright \Psi \mid \Delta \vdash x : B \quad R \triangleright \Psi' \mid \Delta', x : B \vdash T}{P \mid_x R \triangleright \Psi \mid \Psi' \mid \Delta \vdash x : \bullet B \mid \Delta', x : \bullet B \vdash T}\ \mathsf{Conn}^x}{\overrightarrow{x.\mathsf{r}}\ (Q\ ,\ (P \mid_x R)) \triangleright \Psi \mid \Psi' \mid \Delta \vdash x : \bullet A \oplus B \mid \Delta', x : \bullet A \oplus B \vdash T}\oplus\mathsf{C}_2^x$$

Bibliography

[1] Business Process Model and Notation. http://www.omg.org/spec/BPMN/2.0/. (Cited on pages 4, 5, 11, 75, 127 and 156.)

[2] D-Bus website. http://www.freedesktop.org/wiki/Software/dbus/. (Cited on page 107.)

[3] Google SPDY. https://developers.google.com/speed/spdy/. (Cited on page 141.)

[4] Google Web Toolkit. http://code.google.com/webtoolkit/. (Cited on pages 132 and 139.)

[5] Internet Relay Chat Protocol. http://tools.ietf.org/html/rfc1459. (Cited on page 109.)

[6] italianaSoftware s.r.l. http://www.italianasoftware.com/. (Cited on pages 12, 52 and 140.)

[7] JavaScript Object Notation. http://www.json.org/. (Cited on page 132.)

[8] Jolie HTTP extension. https://jolie.svn.sourceforge.net/svnroot/jolie/trunk/extensions/http. (Cited on page 140.)

[9] Leonardo Web Server. http://www.sourceforge.net/projects/leonardo/. (Cited on page 134.)

[10] SOAP Specifications. http://www.w3.org/TR/soap/. (Cited on pages 130 and 148.)

[11] Web Services Addressing. http://www.w3.org/TR/ws-addr-core/. (Cited on page 149.)

[12] Web Services Architecture. http://www.w3.org/TR/ws-arch/. (Cited on page 107.)

[13] Web Services Description Language. http://www.w3.org/TR/wsdl. (Cited on pages 110, 129 and 140.)

[14] Workflow Patterns. http://www.workflowpatterns.com/. (Cited on page 127.)

[15] XML-RPC. http://www.xmlrpc.com/. (Cited on page 130.)

[16] A. Avron. Hypersequents, logical consequence and intermediate logics for concurrency. *Ann. Math. Artif. Intell.*, 4:225–248, 1991. (Cited on page 79.)

[17] P. Baltazar, L. Caires, V. T. Vasconcelos, and H. T. Vieira. A type system for flexible role assignment in multiparty communicating systems. In *TGC*, pages 82–96, 2012. (Cited on pages 49, 72 and 102.)

[18] S. Basu and T. Bultan. Choreography conformance via synchronizability. In *WWW*, pages 795–804, 2011. (Cited on pages 4, 75 and 102.)

[19] G. Bellin and P. J. Scott. On the pi-calculus and linear logic. *Theor. Comput. Sci.*, 135(1):11–65, 1994. (Cited on page 103.)

[20] J. Bengtson, M. Johansson, J. Parrow, and B. Victor. Psi-calculi: a framework for mobile processes with nominal data and logic. *Logical Methods in Computer Science*, 7(1), 2011. (Cited on page 4.)

[21] L. Bettini, M. Coppo, L. D'Antoni, M. D. Luca, M. Dezani-Ciancaglini, and N. Yoshida. Global progress in dynamically interleaved multiparty sessions. In *CONCUR*, volume 5201 of *LNCS*, pages 418–433. Springer, 2008. (Cited on pages 11, 13, 20, 32, 35, 42, 47, 48, 59, 62, 72, 102 and 181.)

[22] K. Bhargavan, R. Corin, P.-M. Deniélou, C. Fournet, and J. J. Leifer. Cryptographic protocol synthesis and verification for multiparty sessions. In *Proc. of CSF*, pages 124–140, 2009. (Cited on pages 47 and 49.)

[23] G. Boudol, Z. Luo, T. Rezk, and M. Serrano. Reasoning about web applications: An operational semantics for hop. *ACM Trans. Program. Lang. Syst.*, 34(2):10, 2012. (Cited on page 139.)

[24] R. Braden. Requirements for Internet Hosts - Communication Layers. RFC 1122 (INTERNET STANDARD), Oct. 1989. Updated by RFCs 1349, 4379, 5884, 6093, 6298, 6633, 6864. (Cited on pages 1 and 107.)

[25] S. Briais and U. Nestmann. A formal semantics for protocol narrations. *Theor. Comput. Sci.*, 389(3):484–511, 2007. (Cited on pages 11, 47 and 49.)

[26] N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, and G. Zavattaro. Choreography and orchestration conformance for system design. In *Proc. of Coordination*, volume 4038 of *LNCS*, pages 63–81. Springer-Verlag, 2006. (Cited on pages 11, 48 and 72.)

[27] L. Caires and F. Pfenning. Session types as intuitionistic linear propositions. In *CONCUR*, pages 222–236, 2010. (Cited on pages 5, 75, 76, 77, 78, 79, 96, 101, 102 and 103.)

[28] L. Caires and H. T. Vieira. Conversation types. *Theoretical Computer Science*, 411(51-52):4399–4440, 2010. (Cited on page 49.)

[29] C. Caleiro, L. Viganò, and D. A. Basin. On the semantics of Alice&Bob specifications of security protocols. *Theor. Comput. Sci.*, 367(1-2):88–122, 2006. (Cited on pages 47 and 49.)

[30] M. Carbone. Session-based choreography with exceptions. In *Proc. of PLACES*, volume 241, pages 35–55. ENTCS, 2008. (Cited on page 49.)

[31] M. Carbone, K. Honda, and N. Yoshida. Structured communication-centered programming for web services. *ACM Trans. Program. Lang. Syst.*, 34(2):8, 2012. (Cited on pages 3, 4, 11, 12, 30, 36, 40, 42, 47, 51, 52, 62, 68, 69, 71, 72, 79, 97, 101, 102, 127, 187, 193 and 195.)

[32] L. Cesari, A. Lapadula, R. Pugliese, and F. Tiezzi. A Tool for Rapid Development of WS-BPEL applications. In *SAC*, pages 2438–2442, 2010. (Cited on page 126.)

[33] D. A. Chappell. *Enterprise Service Bus - Theory in practice*. O'Reilly, 2004. (Cited on pages 127 and 128.)

[34] Chor. Programming Language. `http://www.chor-lang.org/`. (Cited on pages 143 and 150.)

[35] E. G. Coffman, M. Elphick, and A. Shoshani. System deadlocks. *ACM Comput. Surv.*, 3(2):67–78, June 1971. (Cited on page 1.)

[36] P.-M. Deniélou and N. Yoshida. Dynamic multirole session types. In *POPL*, pages 435–446. ACM, 2011. (Cited on pages 47, 49 and 72.)

[37] P.-M. Deniélou and N. Yoshida. Multiparty session types meet communicating automata. In *Proc. of ESOP*, LNCS, pages 194–213. Springer-Verlag, 2012. (Cited on page 47.)

[38] P.-M. Deniélou and N. Yoshida. Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types. In *ICALP (2)*, pages 174–186, 2013. (Cited on page 47.)

[39] M. Dumas, W. M. P. van der Aalst, and A. H. M. ter Hofstede. *Process-Aware Information Systems: Bridging People and Software Through Process Technology*. Wiley, 2005. (Cited on page 127.)

[40] Eclipse. The Eclipse IDE. `http://www.eclipse.org/`. (Cited on pages 143 and 147.)

[41] Enea. ENEA: Italian National agency for new technologies, Energy and sustainable economic development. `http://www.enea.it/`. (Cited on page 12.)

[42] J. Esparza and A. Podelski. Efficient algorithms for pre* and post* on interprocedural parallel flow graphs. In *POPL*, pages 1–11, 2000. (Cited on page 1.)

[43] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Updated by RFCs 2817, 5785, 6266, 6585. (Cited on page 107.)

[44] X. Fu, T. Bultan, and J. Su. Realizability of conversation protocols with message contents. *International Journal on Web Service Res.*, 2(4):68–93, 2005. (Cited on page 47.)

[45] S. J. Gay, V. T. Vasconcelos, A. Ravara, N. Gesbert, and A. Z. Caldeira. Modular session types for distributed object-oriented programming. In *POPL*, pages 299–312, 2010. (Cited on pages 127 and 141.)

[46] J.-Y. Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987. (Cited on page 5.)

[47] J. N. Gray. *Notes on data base operating systems*. Springer, 1978. (Cited on page 149.)

[48] C. Guidi. *Formalizing languages for Service Oriented Computing*. PhD. thesis, University of Bologna, 2007. `http://www.cs.unibo.it/pub/TR/UBLCS/2007/2007-07.pdf`. (Cited on pages 127 and 128.)

[49] C. Guidi, I. Lanese, F. Montesi, and G. Zavattaro. Dynamic Error Handling in Service Oriented Applications. *Fundam. Inform.*, 95(1):73–102, 2009. (Cited on page 123.)

[50] C. Guidi and R. Lucchi. Formalizing mobility in service oriented computing. *JSW*, 2(1):1–13, 2007. (Cited on page 140.)

[51] C. Guidi, R. Lucchi, G. Zavattaro, N. Busi, and R. Gorrieri. SOCK: a calculus for service oriented computing. In *ICSOC*, volume 4294 of *LNCS*, pages 327–338, Heidelberg, Germany, 2006. Springer-Verlag. (Cited on pages 112, 117 and 125.)

[52] D. Hirschkoff, J.-M. Madiot, and D. Sangiorgi. Duality and i/o-types in the -calculus. In *CONCUR*, pages 302–316, 2012. (Cited on page 141.)

[53] K. Honda, A. Mukhamedov, G. Brown, T.-C. Chen, and N. Yoshida. Scribbling interactions with a formal foundation. In *ICDCIT*, volume 6536 of *LNCS*, pages 55–75. Springer, 2011. (Cited on pages 11 and 127.)

[54] K. Honda, V. Vasconcelos, and M. Kubo. Language primitives and type disciplines for structured communication-based programming. In *ESOP'98*, volume 1381 of *LNCS*, pages 22–138, Heidelberg, Germany, 1998. Springer-Verlag. (Cited on pages 43, 65, 75, 77, 96, 126, 140 and 141.)

[55] K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *POPL*, volume 43(1), pages 273–284. ACM, 2008. (Cited on pages 11, 12, 13, 15, 20, 24, 30, 32, 38, 42, 43, 47, 48, 52, 59, 64, 65, 72, 103, 126 and 143.)

[56] R. Hu, D. Kouzapas, O. Pernet, N. Yoshida, and K. Honda. Type-safe eventful sessions in java. In *ECOOP*, pages 329–353, 2010. (Cited on pages 127 and 141.)

[57] R. Hu, N. Yoshida, and K. Honda. Session-based distributed programming in java. In *ECOOP*, pages 516–541, 2008. (Cited on pages 126 and 150.)

[58] IETF. WebSocket protocol. `http://tools.ietf.org/html/rfc6455`. (Cited on page 141.)

[59] International Telecommunication Union. Recommendation Z.120: Message sequence chart, 1996. (Cited on pages 2 and 47.)

[60] Jolie. Programming Language. `http://www.jolie-lang.org/`. (Cited on pages 6, 73, 108, 128 and 130.)

[61] 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. (Cited on pages 3, 11, 51, 68, 71, 72 and 97.)

[62] I. Lanese, C. A. Mezzina, and J.-B. Stefani. Reversing higher-order pi. In *CONCUR*, pages 478–493, 2010. (Cited on page 141.)

[63] J. Lange and E. Tuosto. Synthesising choreographies from local session types. In *CONCUR*, pages 225–239, 2012. (Cited on pages 4 and 102.)

[64] A. Lapadula, R. Pugliese, and F. Tiezzi. A calculus for orchestration of web services. In *ESOP*, pages 33–47, 2007. (Cited on pages 126 and 127.)

[65] A. Lapadula, R. Pugliese, and F. Tiezzi. A Formal Account of WS-BPEL. In *Proceedings of COORDINATION 2008*, pages 199–215, 2008. (Cited on page 126.)

[66] S. Lu, S. Park, E. Seo, and Y. Zhou. Learning from mistakes: a comprehensive study on real world concurrency bug characteristics. *ACM SIGARCH Computer Architecture News*, 36(1):329–339, 2008. (Cited on page 2.)

[67] J. Mendling and M. Hafner. From inter-organizational workflows to process execution: Generating bpel from ws-cdl. In *OTM Workshops*, pages 506–515, 2005. (Cited on page 3.)

[68] M. Merro and D. Sangiorgi. On asynchrony in name-passing calculi. *Mathematical Structures in Computer Science*, 14(5):715–767, 2004. (Cited on pages 47 and 72.)

[69] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–40,41–77, Sept. 1992. (Cited on pages 4, 5, 52, 75, 108, 126 and 140.)

[70] F. Montesi. OpenID implementation with Correlation Sets. `http://www.jolie-lang.org/files/icsoc2011/openid.zip`. (Cited on page 124.)

[71] F. Montesi. Jolie: a Service-oriented Programming Language. Master's thesis, University of Bologna, Department of Computer Science, 2010. (Cited on pages 123, 128, 130, 131, 133 and 138.)

[72] F. Montesi. Compositional Choreographies: Additional Resources. `http://www.fabriziomontesi.com/papers/compchor/`, 2013. (Cited on page 52.)

[73] F. Montesi, C. Guidi, I. Lanese, and G. Zavattaro. Dynamic Fault Handling Mechanisms for Service-Oriented Applications. In *ECOWS*, pages 225–234, 2008. (Cited on pages 123 and 130.)

[74] F. Montesi, C. Guidi, and G. Zavattaro. Composing Services with JOLIE. In *ECOWS*, pages 13–22, 2007. (Cited on pages 108, 126, 127, 128 and 143.)

[75] D. Mostrous, N. Yoshida, and K. Honda. Global principal typing in partially commutative asynchronous sessions. In *ESOP*, pages 316–332, 2009. (Cited on pages 30 and 187.)

[76] Mozilla Foundation. Mozilla Project. `http://www.mozilla.org/`. (Cited on page 2.)

[77] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, Dec. 1978. (Cited on page 3.)

[78] R. H. Netzer and B. P. Miller. What are race conditions?: Some issues and formalizations. *ACM Letters on Programming Languages and Systems (LOPLAS)*, 1(1):74–88, 1992. (Cited on page 1.)

[79] OASIS. Web Services Business Process Execution Language. `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html`. (Cited on pages 3, 4, 5, 75, 107, 125, 127, 128, 134, 139 and 144.)

[80] OOI. Ocean Observatories Initiative. `http://www.oceanobservatories.org`. (Cited on pages 12, 144 and 153.)

[81] OpenID. Specifications. `http://openid.net/developers/specs/`. (Cited on pages 45, 108, 124 and 150.)

[82] Oracle Corporation. Java Enterprise Edition. `http://www.oracle.com/technetwork/java/javaee/overview/index.html`. (Cited on pages 4 and 107.)

[83] Oracle Corporation. MySQL. `http://www.mysql.com/`. (Cited on page 2.)

[84] C. Pautasso and E. Wilde. Push-enabling restful business processes. In *ICSOC*, pages 32–46, 2011. (Cited on page 139.)

[85] J. A. Pérez, L. Caires, F. Pfenning, and B. Toninho. Linear logical relations for session-based concurrency. In *ESOP*, pages 539–558, 2012. (Cited on page 103.)

[86] PHP. PHP: Hypertext Preprocessor. `http://www.php.net/`. (Cited on pages 4 and 107.)

[87] PI4SOA. http://www.pi4soa.org, 2008. (Cited on page 11.)

[88] B. C. Pierce. *Types and Programming Languages*. MIT Press, MA, USA, 2002. (Cited on pages 21, 29 and 30.)

[89] Z. Qiu, X. Zhao, C. Cai, and H. Yang. Towards the theoretical foundation of choreography. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 973–982, New York, NY, USA, 2007. ACM. (Cited on pages 3, 4, 71 and 72.)

[90] Qualit-E. Funded by Cognizant. `http://www.cognizant.com/`. (Cited on page 12.)

[91] G. Ramalingam. Context-sensitive synchronization-sensitive analysis is undecidable. *ACM Trans. Program. Lang. Syst.*, 22(2):416–430, Mar. 2000. (Cited on pages 1 and 2.)

[92] D. Rossi and E. Turrini. Designing and architecting process-aware web applications with epml. In *SAC*, pages 2409–2414, 2008. (Cited on page 139.)

[93] D. Sangiorgi. pi-calculus, internal mobility, and agent-passing calculi. *Theor. Comput. Sci.*, 167(1&2):235–274, 1996. (Cited on page 94.)

[94] D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001. (Cited on pages 2 and 140.)

[95] Savara. JBoss Community. `http://www.jboss.org/savara/`. (Cited on pages 11 and 12.)

[96] M. Serrano, E. Gallesio, and F. Loitsch. Hop: a language for programming the web 2.0. In *OOPSLA Companion*, pages 975–985, 2006. (Cited on page 139.)

[97] USB Implementers Forum. USB Specifications. `http://www.usb.org/developers/docs/`. (Cited on page 1.)

[98] W. M. P. van der Aalst. Verification of workflow nets. In *ICATPN*, pages 407–426, 1997. (Cited on page 127.)

[99] W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede, N. Russell, H. M. W. E. Verbeek, and P. Wohed. Life after bpel? In *EPEW/WS-FM*, pages 35–50, 2005. (Cited on page 4.)

[100] W. M. P. van der Aalst and A. H. M. ter Hofstede. Yawl: yet another workflow language. *Inf. Syst.*, 30(4):245–275, 2005. (Cited on pages 127 and 139.)

[101] V. Vasconcelos and N. Yoshida. Language primitives and type discipline for structured communication-based programming revisited: Two systems for higher-order session communication. *Electr. Notes Theor. Comput. Sci.*, 171(4):73–93, 2007. (Cited on page 161.)

[102] W3C WS-CDL Working Group. Web services choreography description language version 1.0. http://www.w3.org/TR/ws-cdl-10/, 2004. (Cited on pages 3, 11, 12, 127, 139 and 156.)

[103] P. Wadler. Propositions as sessions. In *ICFP*, pages 273–286, 2012. (Cited on pages 75, 96, 101 and 103.)

[104] Xtext. The Xtext Framework. `http://www.eclipse.org/Xtext/`. (Cited on page 147.)

[105] N. Yoshida, P.-M. Deniélou, A. Bejleri, and R. Hu. Parameterised multiparty session types. In *FOSSACS'10*, volume 6014 of *LNCS*, pages 128–145, 2010. (Cited on pages 43, 60 and 61.)