# Introduction to Choreographies
## First edition (DRAFT)

Fabrizio Montesi
*Department of Mathematics and Computer Science*
*University of Southern Denmark*


Last update: November 6, 2019

# Contents

# Preface: Alice, Bob, Concurrency, and Distribution

We live in the era of *concurrency*, the performance of multiple tasks at a time, and *distribution*, the act of computing using communicating connected devices. These aspects have become pervasive in modern computing. Today, even mobile phones and tiny computers like Raspberry Pis feature multiple processing units of different kinds, with purposes that go from generic computing to more specialised ones like graphics and artificial intelligence. Our computer networks are getting bigger than ever, with the rise of the World Wide Web, telecommunications, cloud computing, and the Internet of Things. This transformation is making the number of computer programs that communicate with each other over a network explode. By 2025, the Internet alone is expected to connect from 25 to 100 billion devices [OECD 2016].

On the one hand, modern computer networks and their applications have become the drivers of our technological advancement. They give us better citizen services, a more efficient industry (Industry 4.0), new ways to connect socially, and even better health with smart medical devices. On the other hand, these systems and their software are increasingly complex. Services depend on other services to function. For example, a web store might depend on an external service provided by a bank to carry out customer payments. The web store, the customer's web browser, and the bank service are thus integrated: they communicate with each other to reach the goal of transferring the right amount of money to the right recipient, such that the customer can get the product she wants from the store. In concurrent and distributed systems, the heart of integration is the notion of *protocol*: a document that prescribes the communications that the involved parties should perform in order to reach a goal.

It is important that protocols are clear and precise. If they are ambiguous, we risk that the designers of different parts of the same system interpret them differently, which leads to errors. The consequences of such errors can be dire,

including data getting in the system (deadlocks), data corruption, or data leaks. The more we equip programmers with solid methods for specifying and implementing protocols correctly, the more likely they are to succeed at integrating the different parts of concurrent and distributed systems correctly. The ultimate quest is to increase the reliability of these systems.

It is also important that protocols are as concise as possible: the bigger a protocol or the harder it is to write it down, the higher the chance that we make some mistake in writing it. Computer scientists and mathematicians might get a familiar feeling when presented with the necessity of achieving clarity, precision, and conciseness in writing. A computer scientist could point out that we need a good *language* to write protocols. A mathematician could say that we need a good *notation*. There is no contradiction here, since a language is, in fact, a notation. Both computer scientists and mathematician might say that we could (or should!) be even more ambitious and seek an *elegant* language.

Needham and Schroeder introduced an interesting notation for writing protocols in 1978. A communication of a message $M$ from the participant $A$ to the participant $B$ is written

$$A \mathrel{-\!\!>} B : M \ .$$

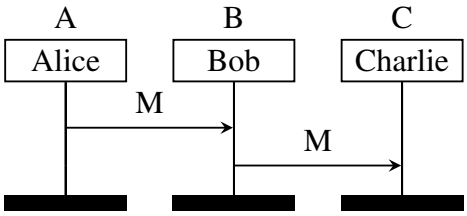
To define a protocol where $A$ sends a message $M$ to $B$, and then $B$ passes the same message to $C$, we can just compose communications in a sequence:

$$\begin{aligned} A &\mathrel{-\!\!>} B : M \\ B &\mathrel{-\!\!>} C : M \ . \end{aligned}$$

This notation is called "Alice and Bob notation", due to a presentational style found in security research whereby the participants $A$ and $B$ represent the fictional characters Alice and Bob, respectively, who use the protocol to perform some task. There might be more participants, like $C$ in our example—typically a shorthand for Carol, or Charlie. The first mention of Alice and Bob appeared in the seminal paper by Rivest et al. [1978] on their public-key cryptosystem:

> "For our scenarios we suppose that $A$ and $B$ (also known as Alice and Bob) are two users of a public-key cryptosystem".

Over the years, researchers and developers created many more protocol notations. Some of these notations are graphical rather than textual, like Message Sequence Charts [International Telecommunication Union 1996]. The message sequence chart of our protocol with Alice, Bob, and Charlie looks as follows.




For our particular example, the graphical representation of our protocol (as a message sequence chart) and our previous textual representation (in Alice and Bob notation) are equivalent, in the sense that they contain the same information. This is not always the case: not all notations are equally expressive.

In the beginning of the 2000s, researchers and practitioners took the idea of protocol notations even further, and developed the idea of *choreography*. A choreography offers more details. For example, some details that might be included are:

- the kind of data being transmitted, or even the actual functions used to compute the data to be transmitted;
- nested protocols, i.e., the ability to call another protocol like a procedure;
- the state of participants, e.g., their memory states.

Choreographies are typically written in languages designed to be readable *mechanically*. This makes them amenable to be used in computer programs and, also, rigorous mathematical reasoning. In this book, we will start with a very simple choreography language and then progressively extend it with more sophisticated features, like parallelism and recursion. We will see that it is possible to define mathematically a *semantics* for choreographies, which gives us an interpretation of what running a protocol means. We will also see that it is possible to translate choreographies to a theoretical model of executable programs, which gives us an interpretation of how choreographies can be correctly implemented in the real world.

Although choreographies still represent a young and active area of research, they have already emerged in many places. In 2005, the World Wide Web Consortium (W3C)—the main international standards organisation for the web—drafted the Web Services Choreography Description Language (WS-CDL), for defining interactions among web services [W3C WS-CDL Working Group 2004]. In 2011, the global technology standards consortium Object Management Group (OMG) introduced choreographies in their notation for business processes [Object Management Group 2011]. The recent paradigm of microservices [Dragoni et al.

2017] advocates the use of choreographies to achieve better scalability. All this momentum is motivating a lot of research on both the theory of choreographies and its application to programming [Ancona et al. 2016, Hüttel et al. 2016]. These days, Alice and Bob surely are in the spotlight.

# This book

This book is an introduction to the basic theory of choreographies. It explains what choreographies are and how we can model them mathematically. Its primary intended audience consists of computer science students and researchers, but it is also designed to be approachable by mathematicians (willing to become) familiar with context-free grammars.

The aim of this book is to be pedagogical, and to equip the reader with a new perspective on how we can abstract, design, and reason about concurrent and distributed systems. It is not an aim to be comprehensive, and we will not present features to capture all possible protocols. References to alternative notations, further developments, and techniques to model choreographies are given where appropriate. It is assumed that the reader is familiar with the notion of concurrency and the basic intuition of how distributed systems are programmed.

**Prerequisites** To read this book, you should be familiar with:

- discrete mathematics and the induction proof method;
- context-free grammars (only basic knowledge is required);
- basic data structures, like trees and graphs;
- concurrent and distributed systems.

These prerequisites are attainable in most computer science B.Sc. degrees, or with three years of relevant experience.

To study choreographies, we are going to define choreography languages and then write choreographies as terms of these languages. The syntax of languages is going to be defined using context-free grammars. To give meaning to choreographies, we are going to use extensively Plotkin's structural approach to operational semantics [Plotkin 2004].

The rules defining the semantics of choreographies are going to be rules of inference, borrowing from deductive systems. Knowing formal systems based on rules of inference is an advantage, but not a requirement for reading this book:

chapter 1 provides a brief introduction to the essential knowledge on these systems that we need for the rest of the book. The reader familiar with inference systems or structural operational semantics can safely skip the first chapter and jump straight to chapter 2.

An important aspect of choreographies is determining how they can be executed correctly in concurrent and distributed systems, in terms of independent programs for *processes*. To model process programs, we will borrow techniques from the area of process calculi. We will introduce the necessary notions on process calculi as we go along, so knowing this area is not a requirement for reading this book. The reader familiar with process calculi will recognise that we borrow many ideas from Milner's seminal calculus of communicating systems [Milner 1980].

Some exercises in this book are marked with !, indicating a higher degree of difficulty. For the exercises marked with $\hookrightarrow$, a solution is given in appendix A.

# Chapter 1

# Inference systems

Before we venture into the study of choreographies, we need to become familiar with the formalism that we are going to use throughout this book: inference systems. Inference systems are widely used in formal logic and the specification of programming languages (our case).[1]

**Definition 1** (Inference systems and inference rules)**.** *An inference system is a set of* inference rules *(also called rules of inference). An inference rule has the form*

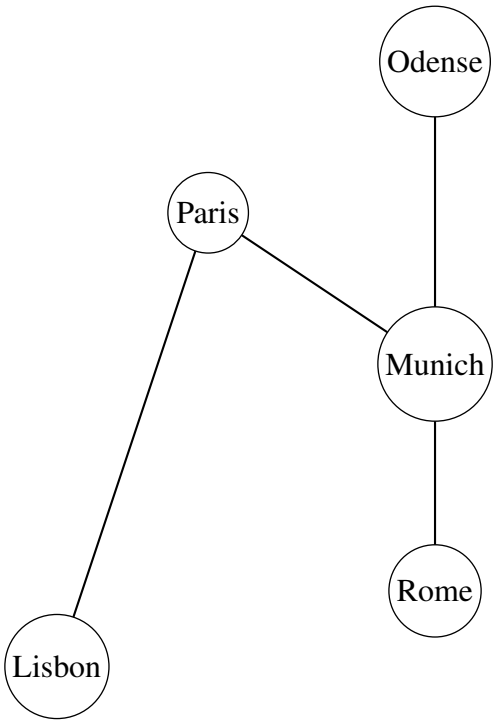
$$\frac{\textit{Premise } 1 \quad \textit{Premise } 2 \quad \cdots \quad \textit{Premise } n}{\textit{Conclusion}}$$

*and reads "If the premises Premise* 1*, Premise* 2*, . . . , and Premise* $n$ *hold, then Conclusion holds". It is perfectly valid for an inference rule to have no premises: we call such rules* axioms*, because their conclusions always hold. An inference rule has always exactly one conclusion.*

## 1.1 Example: Flight connections

Consider the following undirected graph of direct flights between cities.

[1] For further reading on these systems, see the lecture notes by Martin-Löf [1996].




Let $A$, $B$, and $C$ range over cities. We denote that two cities $A$ and $B$ are connected by a direct flight with the *proposition* $\mathsf{conn}(A, B)$. Then, we can represent our graph as the set of axioms below.[2]

$$\frac{}{\mathsf{conn}(\text{Odense}, \text{Munich})} \quad \frac{}{\mathsf{conn}(\text{Munich}, \text{Rome})} \quad \frac{}{\mathsf{conn}(\text{Paris}, \text{Munich})}$$

$$\frac{}{\mathsf{conn}(\text{Paris}, \text{Lisbon})}$$

Notice that our graph is undirected, meaning that all direct flights are available also in the opposite direction. This is not faithfully represented by our axioms: if $\mathsf{conn}(A, B)$, it should also be the case that $\mathsf{conn}(B, A)$. One option to solve this discrepancy is to double the number of our axioms, to include their symmetric versions—e.g., we would add an axiom concluding with $\mathsf{conn}(\text{Munich}, \text{Odense})$. A more concise option is to formalise the general concept of symmetry of connections with a rule of inference, as follows.

$$\frac{\mathsf{conn}(A, B)}{\mathsf{conn}(B, A)}\ \textsc{Sym}$$

The label SYM is the name of our new inference rule; it is just a decoration to remember what the rule does (SYM is a shorthand for symmetry). Rule SYM tells

[2] This example is inspired by Pfenning's lecture notes on inference rules, where he also uses inference rules to model graphs [Pfenning 2012].

$$\frac{}{\mathsf{conn}(\text{Odense}, \text{Munich})} \qquad \frac{}{\mathsf{conn}(\text{Munich}, \text{Rome})} \qquad \frac{}{\mathsf{conn}(\text{Paris}, \text{Munich})}$$

$$\frac{}{\mathsf{conn}(\text{Paris}, \text{Lisbon})}$$

$$\frac{\mathsf{conn}(A, B)}{\mathsf{conn}(B, A)}\ \textsc{Sym} \qquad \frac{\mathsf{conn}(A, B)}{\mathsf{path}(A, B)}\ \textsc{Dir} \qquad \frac{\mathsf{path}(A, B) \quad \mathsf{path}(B, C)}{\mathsf{path}(A, C)}\ \textsc{Trans}$$

Figure 1.1: An inference system for flights.

us that if we have a connection from any $A$ to any $B$ (premise), then we have a connection from $B$ to $A$ (conclusion). In this rule, $A$ and $B$ are *schematic variables*: they can stand for any of our cities. So if we were to add more connections in the future, their symmetric version would also become immediately available thanks to rule SYM.

Now that we are satisfied with how our inference system captures the graph, we can use it to find flight paths from a city to another. Say that for any two cities $A$ and $B$, the proposition $\mathsf{path}(A, B)$ denotes that there is a path from $A$ to $B$. Finding paths is relatively easy. First, we observe that direct connections give us a path. (In the following rule, DIR stands for direct.)

$$\frac{\mathsf{conn}(A, B)}{\mathsf{path}(A, B)}\ \textsc{Dir}$$

Second, we formulate a rule for multi-step paths. If there is a path from $A$ to $B$, and a path from $B$ to $C$, then we have a path from $A$ to $C$. In other words, paths are transitive. (TRANS stands for transitivity.)

$$\frac{\mathsf{path}(A, B) \quad \mathsf{path}(B, C)}{\mathsf{path}(A, C)}\ \textsc{Trans}$$

The whole system is displayed in fig. 1.1.

## 1.2 Derivations

The key feature of an inference system is the capability of performing *derivations*. Suppose that we wanted to answer the following question.

> Is there a flight path from Odense to Rome?

Answering this question in our inference system for flight connections corresponds to showing that $\mathsf{path}(\text{Odense}, \text{Rome})$ holds. To do this, we have to build a *derivation tree*. We will call derivation trees also simply *derivations*, or *proof trees* (since they prove that something can be derived in a given inference system).

**From Odense to Rome** We start looking for our derivation from what we want to conclude with—our conclusion is what we want to prove.

$$\mathsf{path}(\text{Odense}, \text{Rome})$$

Observe that we have only two inference rules that can conclude something of this form: DIR and TRANS. Let us try to apply DIR first, by substituting its schematic variables with the cities that we need:

$$\frac{\mathsf{conn}(\text{Rome}, \text{Odense})}{\mathsf{path}(\text{Odense}, \text{Rome})}\ \textsc{Dir}\ .$$

Our derivation is not complete, because we are left with a premise that we are now responsible for proving: $\mathsf{conn}(\text{Odense}, \text{Rome})$. We can read what we have now as: if Rome is connected to Odense, then there is a path from Odense to Rome. However, proving $\mathsf{conn}(\text{Odense}, \text{Rome})$ seems unfeasible: there is no rule that concludes that, and we cannot hope to conclude it by applying rule SYM, since that would require in turn to prove $\mathsf{conn}(\text{Rome}, \text{Odense})$.

Now that we know that rule DIR is not helpful for our derivation, our only remaining option is to apply rule TRANS. Clearly, we should instantiate $A$ as Odense and $C$ as Rome in the application of TRANS. How should we instantiate $B$, though? (We informally write $B?$ for "we do not know what $B$ should be yet" here.)

$$\frac{\mathsf{path}(\text{Odense}, B?) \quad \mathsf{path}(B?, \text{Rome})}{\mathsf{path}(\text{Odense}, \text{Rome})}\ \textsc{Trans}$$

One way would be to try out all possible cities as our $B$, in search of a city that will allow us to continue our derivation. However, looking at the definition of our graph, it is easy to see that the right choice is to pick Munich, since that city is connected to both Odense and Rome.

$$\frac{\mathsf{path}(\text{Odense}, \text{Munich}) \quad \mathsf{path}(\text{Munich}, \text{Rome})}{\mathsf{path}(\text{Odense}, \text{Rome})}\ \textsc{Trans}$$

What we have above is an instantiation of rule TRANS. (We will call rule instantiations also rule applications in the remainder, since it corresponds to "applying" the rule to the premises in order to reach the conclusion.) The derivation depends on the validity of some premises. We can read it as a proof of the

statement "If $\mathsf{path}(\text{Odense}, \text{Munich})$ holds and $\mathsf{path}(\text{Munich}, \text{Rome})$ holds, then $\mathsf{path}(\text{Odense}, \text{Rome})$ holds". In other words, our derivation of $\mathsf{path}(\text{Odense}, \text{Rome})$ has two hypotheses: $\mathsf{path}(\text{Odense}, \text{Munich})$ and $\mathsf{path}(\text{Munich}, \text{Rome})$. Thus, we have gone from the task of deriving $\mathsf{path}(\text{Odense}, \text{Rome})$ to the tasks of deriving $\mathsf{path}(\text{Odense}, \text{Munich})$ and $\mathsf{path}(\text{Munich}, \text{Rome})$, respectively. Thanks to the notation of inference rules, we can complete these tasks just by "digging deeper" with further rule applications. Since we know that Odense and Munich are connected, we can try using rule DIR.

$$\dfrac{\dfrac{\mathsf{conn}(\text{Odense}, \text{Munich})}{\mathsf{path}(\text{Odense}, \text{Munich})}\ \textsc{Dir} \qquad \mathsf{path}(\text{Munich}, \text{Rome})}{\mathsf{path}(\text{Odense}, \text{Rome})}\ \textsc{Trans}$$

We now have to prove $\mathsf{conn}(\text{Odense}, \text{Munich})$. We have that as the conclusion of one of our axioms, so we can conclude that branch of our derivation by applying the related axiom.

$$\dfrac{\dfrac{\dfrac{}{\mathsf{conn}(\text{Odense}, \text{Munich})}}{\mathsf{path}(\text{Odense}, \text{Munich})}\ \textsc{Dir} \qquad \mathsf{path}(\text{Munich}, \text{Rome})}{\mathsf{path}(\text{Odense}, \text{Rome})}\ \textsc{Trans}$$

We can finish our derivation for the right premise $\mathsf{path}(\text{Munich}, \text{Rome})$ likewise.

$$\dfrac{\dfrac{\dfrac{}{\mathsf{conn}(\text{Odense}, \text{Munich})}}{\mathsf{path}(\text{Odense}, \text{Munich})}\ \textsc{Dir} \qquad \dfrac{\dfrac{}{\mathsf{conn}(\text{Munich}, \text{Rome})}}{\mathsf{path}(\text{Munich}, \text{Rome})}\ \textsc{Dir}}{\mathsf{path}(\text{Odense}, \text{Rome})}\ \textsc{Trans}$$

What we have now is a *complete* derivation, i.e., a derivation without hypotheses. We can tell by the fact that there are no more premises left to prove. Since the derivation is complete, it shows that $\mathsf{path}(\text{Odense}, \text{Rome})$ can always be derived in our inference system, without making any assumptions. In general, we say that a proposition is derivable if there exists at least one complete derivation with such proposition as conclusion.

**The structure of derivations** We now move to the formal definition of derivations.

First, some notation. Let $\mathcal{D}$ (for derivation) range over derivations and $p$ range over propositions. (Sometimes we will also write $\mathcal{P}$, for "proof", to denote a

derivation.) We shall write $\overset{\mathcal{D}}{p}$ for "the derivation $\mathcal{D}$ has conclusion $p$". For example, assume that $\mathcal{D}$ is our latest derivation, as follows.

$$\mathcal{D} \quad \triangleq \quad \dfrac{\dfrac{\overline{\mathsf{conn}(\mathrm{Odense}, \mathrm{Munich})}}{\mathsf{path}(\mathrm{Odense}, \mathrm{Munich})}\ \textsc{Dir} \qquad \dfrac{\overline{\mathsf{conn}(\mathrm{Munich}, \mathrm{Rome})}}{\mathsf{path}(\mathrm{Munich}, \mathrm{Rome})}\ \textsc{Dir}}{\mathsf{path}(\mathrm{Odense}, \mathrm{Rome})}\ \textsc{Trans}$$

The symbol $\triangleq$ in the equation above means "is defined as". Therefore, we have that $\overset{\mathcal{D}}{\mathsf{path}(\mathrm{Odense}, \mathrm{Rome})}$.

Observe that $\mathcal{D}$ can be seen as a tree, by viewing each rule application as a node of the tree, where the root node is the rule application that reaches the conclusion of the derivation. (In the last derivation, the root is the application of rule TRANS.) This is a very useful property that we will use throughout the book, in particular to apply the principle of induction in our formal reasonings about derivations.

**Definition 2** (Derivation)**.** *Let $S$ be an inference system. Then:*

- *Any application of a rule in $S$ is a derivation in $S$.*
- *Let $\mathcal{D}_1$, ..., $\mathcal{D}_n$ be derivations in $S$, for some natural number $n > 0$, with respective conclusions $p_1$, ..., $p_n$. Then, any application of a rule* R *in $S$ with $p_1$, ..., $p_n$ as premises is a derivation in $S$.*

Intuitively, a complete derivation is a derivation where the "leaves" of the derivation tree (the base cases) are all axioms. We can formalise this by tuning slightly the base case of the previous definition. (The difference is emphasised.)

**Definition 3** (Complete Derivation)**.** *Let $S$ be an inference system.*

- *Any application of* an axiom *in $S$ is a complete derivation in $S$.*
- *Let $\mathcal{D}_1$, ..., $\mathcal{D}_n$ be complete derivations in $S$, for some natural number $n > 0$, with respective conclusions $p_1$, ..., $p_n$. Then, any application of a rule* R *in $S$ with $p_1$, ..., $p_n$ as premises is a complete derivation in $S$.*

We impose that derivations are finite: we are interested in proofs that can be checked mechanically in finite time.

**Exercise 1.** *Prove that the following statements hold for the system in fig. 1.1.*

1. *The proposition* $\mathsf{path}(\mathrm{Paris}, \mathrm{Rome})$ *is derivable.*

$$\frac{}{\mathsf{conn}(\text{Odense}, \text{Munich})} \qquad \frac{}{\mathsf{conn}(\text{Munich}, \text{Rome})} \qquad \frac{}{\mathsf{conn}(\text{Paris}, \text{Munich})}$$

$$\frac{}{\mathsf{conn}(\text{Paris}, \text{Lisbon})}$$

$$\frac{\mathsf{conn}(A, B)}{\mathsf{conn}(B, A)}\ \textsc{Sym}$$

$$\frac{\mathsf{conn}(A, B)}{\mathsf{path}(A, B, 1)}\ \textsc{DirW} \qquad \frac{\mathsf{path}(A, B, n) \quad \mathsf{path}(B, C, m)}{\mathsf{path}(A, C, n + m)}\ \textsc{TransW}$$

Figure 1.2: Weighted rules for flight paths.

2. *The proposition* $\mathsf{path}(\text{Odense}, \text{Odense})$ *is derivable. (*Hint: *in rule* Trans*, nothing forbids* $A$ *from being the same as* $C$*.)*

3. *For every* $A$*, the proposition* $\mathsf{path}(A, A)$ *is derivable.*

**Exercise 2.** *Prove the following statements about the system in fig. 1.1.*

1. *For any* $A$*, there are infinitely many derivations that conclude* $\mathsf{path}(A, A)$.

2. *For any* $A$ *and* $B$*, if there exists at least one derivation that concludes* $\mathsf{path}(A, B)$*, then there are infinitely many derivations that conclude* $\mathsf{path}(A, B)$.

**Exercise 3** ($\hookrightarrow$)**.** *Consider the system in fig. 1.2, which replaces rules* Dir *and* Trans *respectively with the alternative rules* DirW *and* TransW*, which measure the length of a path (paths are "weighted", with each connection having weight* 1*).*

*Prove that, for any* $A$ *and* $B$*, if* $\mathsf{path}(A, B)$ *is derivable in the system in fig. 1.1, then there exists* $n$ *such that* $\mathsf{path}(A, B, n)$ *is derivable in the system in fig. 1.2.*

*Suggestion: proceed by structural induction on the proof of* $\mathsf{path}(A, B)$.

## 1.3 Underivable propositions

Showing that a proposition holds requires showing a proof, i.e., a derivation, in the inference systems of interest. Once the proof is shown, then we just need to check that all rules have been applied correctly. If we are convinced that this is the case, then we are convinced that the proof is correct and that the proposition indeed holds.

Knowing what can be derived is paramount to establish the adequacy of an inference system, but it is just as important to check what *cannot* be derived. But how can we prove that something cannot be derived? This can be tricky, because it requires us to reason about all the possible derivations that could, potentially, conclude with our proposition and showing that none of those derivations can actually be built.

Consider a simple example: showing that $\mathsf{conn}(\text{Lisbon}, \text{Rome})$ is not derivable. Intuitively, there is no direct connection between Lisbon and Rome in our graph. But how can we show this formally?

First, we observe that the only rules that can have a conclusion of the form $\mathsf{conn}(A, B)$ for any $A$ and $B$ are our axioms and rule SYM. This observation conveniently restricts the set of rules that we have to consider. None of the axioms can be applied for $A = \text{Lisbon}$ and $B = \text{Rome}$, as in our case. We are left only with rule SYM, so any search of a derivation of $\mathsf{conn}(\text{Lisbon}, \text{Rome})$ must begin as follows.

$$\dfrac{\mathsf{conn}(\text{Rome}, \text{Lisbon})}{\mathsf{conn}(\text{Lisbon}, \text{Rome})}\ \textsc{Sym}$$

Again, by the same argument, there is no rule to apply for $\mathsf{conn}(\text{Rome}, \text{Lisbon})$ but SYM. So we obtain that the proof *must* continue with another application of SYM.

$$\dfrac{\dfrac{\mathsf{conn}(\text{Lisbon}, \text{Rome})}{\mathsf{conn}(\text{Rome}, \text{Lisbon})}\ \textsc{Sym}}{\mathsf{conn}(\text{Lisbon}, \text{Rome})}\ \textsc{Sym}$$

We got back to where we started: we have to prove $\mathsf{conn}(\text{Lisbon}, \text{Rome})$. Since we have only one way to prove it, and we have just shown that it leads to the exact same premise, this points out that our proof search will go on indefinitely and that we will never reach a finite derivation. So, $\mathsf{conn}(\text{Lisbon}, \text{Rome})$ cannot be proven.

Let us be more formal, to convince ourselves that $\mathsf{conn}(\text{Lisbon}, \text{Rome})$ cannot be derived more decisively. Since every derivation is a finite tree, we can measure the height of a derivation as a natural number (it is the height of the tree). Thus, among all the proofs of $\mathsf{conn}(\text{Lisbon}, \text{Rome})$, there is at least one of minimal height, in the sense that there is no other proof of $\mathsf{conn}(\text{Lisbon}, \text{Rome})$ with lower height. We attempt at finding this minimal proof. The reasoning goes as before, and we soon end up seeing that a minimal proof necessarily starts as follows.

$$\dfrac{\dfrac{\mathsf{conn}(\text{Lisbon}, \text{Rome})}{\mathsf{conn}(\text{Rome}, \text{Lisbon})}\ \textsc{Sym}}{\mathsf{conn}(\text{Lisbon}, \text{Rome})}\ \textsc{Sym}$$

$$\frac{}{\mathsf{conn}(\text{Odense}, \text{Munich})} \qquad \frac{}{\mathsf{conn}(\text{Munich}, \text{Rome})} \qquad \frac{}{\mathsf{conn}(\text{Paris}, \text{Lisbon})}$$

$$\frac{\mathsf{conn}(A, B)}{\mathsf{conn}(B, A)}\ \textsc{Sym}$$

$$\frac{\mathsf{conn}(A, B)}{\mathsf{path}(A, B, 1)}\ \textsc{DirW} \qquad \frac{\mathsf{path}(A, B, n) \quad \mathsf{path}(B, C, m)}{\mathsf{path}(A, C, n + m)}\ \textsc{TransW}$$

Figure 1.3: A limited and weighted flight system.

So we have to find some proof of our premise $\mathsf{conn}(\text{Lisbon}, \text{Rome})$ on top. But if such a proof exists, it would be a proof of $\mathsf{conn}(\text{Lisbon}, \text{Rome})$ that is *smaller* than the proof that we are building (because it would not have the first two applications of SYM of our proof). Thus our minimal proof must be bigger than another proof, and we reach a contradiction.

**Exercise 4.** *Consider the system in fig. 1.3, which removes the direct flight from Paris to Munich.*

*Prove that it is not possible to derive* $\mathsf{path}(\textit{Lisbon}, \textit{Munich}, 1)$.

**Exercise 5.** *Prove that it is not possible to derive* $\mathsf{path}(\textit{Lisbon}, \textit{Munich}, 2)$ *using the system in fig. 1.3.*

**Exercise 6** (!)**.** *Prove that there exists no* $n$ *such that* $\mathsf{path}(\textit{Lisbon}, \textit{Munich}, n)$ *is derivable in the system in fig. 1.3.*

## 1.4 Rule derivability and admissibility

Consider the system in fig. 1.4, which replaces rule TRANS from fig. 1.1 with rule STEP. The difference is that rule STEP requires a direct connection from the source $A$ to some city $B$, and then a path from $B$ to the destination $C$.

From an algorithmic perspective, adopting rule TRANS or rule STEP might lead to slightly different search strategies. Searching for a path from $A$ to $C$ using rule STEP roughly corresponds to: look up in our database of direct connections (given by the axioms and their reflexive closure, thanks to rule SYM) from $A$ to some $B$, and then recursively try to find a path from $B$ to $C$; if we fail, we have to try with another $B$, if possible. By contrast, searching for a path from $A$ to $C$ using rule TRANS corresponds to recursively trying to find a path from $A$ to some $B$, and then again recursively trying to find a path from $B$ to $C$; again, if we fail,

$$\frac{}{\mathsf{conn}(\text{Odense}, \text{Munich})} \qquad \frac{}{\mathsf{conn}(\text{Munich}, \text{Rome})} \qquad \frac{}{\mathsf{conn}(\text{Paris}, \text{Munich})}$$

$$\frac{}{\mathsf{conn}(\text{Paris}, \text{Lisbon})}$$

$$\frac{\mathsf{conn}(A, B)}{\mathsf{conn}(B, A)}\ \textsc{Sym} \qquad \frac{\mathsf{conn}(A, B)}{\mathsf{path}(A, B)}\ \textsc{Dir} \qquad \frac{\mathsf{conn}(A, B) \quad \mathsf{path}(B, C)}{\mathsf{path}(A, C)}\ \textsc{Step}$$

Figure 1.4: An alternative way of constructing paths.

we have to try with another $B$, if possible. Of course, these strategies are only for paths not covered already by rule DIR, which covers the case in which $A$ and $C$ are connected directly.

Since the two systems are different, a key question is whether they are *equally powerful*, in the sense that every derivable proposition in one of the two is derivable also in the other.

There are only two kinds of propositions that we can derive in our two systems: $\mathsf{conn}(A, B)$ and $\mathsf{path}(A, B)$. It is easy to see that, for any $A$ and $B$, $\mathsf{conn}(A, B)$ is derivable in the system with rule TRANS if and only if it is derivable in the system with rule STEP, simply because the two systems share exactly the same rules for deriving conn propositions. For propositions of the form $\mathsf{path}(A, B)$, the situation is more complicated. We tackle the two directions separately (from the system with rule STEP to the system with rule TRANS, and vice versa). The exploration of each direction leads to its own useful new concept—derivable rules and admissible rules.

### 1.4.1 Derivable rules

We start by showing that the system with rule TRANS (fig. 1.1) can derive all paths that can be derived in the system with rule STEP (fig. 1.4).

To do this, we prove that adding rule STEP to the system with rule TRANS would not add any new derivable propositions. Recall that rule STEP looks as follows.

$$\frac{\mathsf{conn}(A, B) \quad \mathsf{path}(B, C)}{\mathsf{path}(A, C)}\ \textsc{Step}$$

The key observation here is that we can build a derivation that, from the premises $\mathsf{conn}(A, B)$ and $\mathsf{path}(B, C)$, concludes $\mathsf{path}(A, C)$ by using the rules

in fig. 1.1. Here it is.

$$\dfrac{\dfrac{\mathsf{conn}(A,B)}{\mathsf{path}(A,B)}\,\textsc{Dir} \qquad \mathsf{path}(B,C)}{\mathsf{path}(A,C)}\,\textsc{Trans} \tag{1.1}$$

The derivation above holds for any $A$, $B$, and $C$. Since it is parametric on these schematic variables, a more correct name might be derivation scheme, but we will allow ourselves the abuse of terminology and simply call it a derivation. This derivation is proof that rule STEP is *derivable* in the system in fig. 1.1. In general, we say that a rule is derivable whenever its conclusion can be derived from its premises by using rules that are already in the system. In other words, if we can build a derivation from the premises of the rule to its conclusion, then the rule is derivable.

A very nice and convenient property of derivable rules is that their applications can be rewritten locally inside of derivations, without the need for modifying the derivations of the premises. Let us see an example to understand what this means. From the derivation in eq. (1.1), we now know that we can rewrite every application of rule STEP into a valid derivation in the system in fig. 1.1 as follows. We also show where the derivations of the premises go ($\rightarrow$ here means "is rewritten to"):

$$\dfrac{\overset{\mathcal{D}}{\mathsf{conn}(A,B)} \qquad \overset{\mathcal{E}}{\mathsf{path}(B,C)}}{\mathsf{path}(A,C)}\,\textsc{Step} \;\rightarrow\; \dfrac{\dfrac{\overset{\mathcal{D}}{\mathsf{conn}(A,B)}}{\mathsf{path}(A,B)}\,\textsc{Dir} \qquad \overset{\mathcal{E}}{\mathsf{path}(B,C)}}{\mathsf{path}(A,C)}\,\textsc{Trans} \tag{1.2}$$

We can apply this transformation to any derivation in the system with rule STEP (fig. 1.4). For instance, consider the following derivation of a multi-hop path from Odense to Lisbon:

$$\dfrac{\dfrac{}{\mathsf{conn}(\mathrm{Odense},\mathrm{Munich})} \qquad \dfrac{\dfrac{\dfrac{}{\mathsf{conn}(\mathrm{Paris},\mathrm{Munich})}}{\mathsf{conn}(\mathrm{Munich},\mathrm{Paris})}\,\textsc{Sym} \qquad \dfrac{\dfrac{}{\mathsf{conn}(\mathrm{Paris},\mathrm{Lisbon})}}{\mathsf{path}(\mathrm{Paris},\mathrm{Lisbon})}\,\textsc{Dir}}{\mathsf{path}(\mathrm{Munich},\mathrm{Lisbon})}\,\textsc{Step}}{\mathsf{path}(\mathrm{Odense},\mathrm{Lisbon})}\,\textsc{Step}$$

To translate this derivation into a derivation in the system in fig. 1.1, we can just rewrite each application of rule STEP as indicated by eq. (1.2). The order in which we pick and rewrite these applications does not matter.[3] For example, here is the

[3] In fact, thanks to the locality of our transformation, one could even devise a concurrent algorithm that replaces all occurrences of STEP in parallel.

result of replacing the top-right occurrence of STEP first:

$$\dfrac{\mathsf{conn}(\mathrm{Odense}, \mathrm{Munich}) \qquad \dfrac{\dfrac{\dfrac{\mathsf{conn}(\mathrm{Paris}, \mathrm{Munich})}{\mathsf{conn}(\mathrm{Munich}, \mathrm{Paris})}\,\textsc{Sym}}{\mathsf{path}(\mathrm{Munich}, \mathrm{Paris})}\,\textsc{Dir} \qquad \dfrac{\mathsf{conn}(\mathrm{Paris}, \mathrm{Lisbon})}{\mathsf{path}(\mathrm{Paris}, \mathrm{Lisbon})}\,\textsc{Dir}}{\mathsf{path}(\mathrm{Munich}, \mathrm{Lisbon})}\,\textsc{Trans}}{\mathsf{path}(\mathrm{Odense}, \mathrm{Lisbon})}\,\textsc{Step}\ .$$

And here the result of replacing also the reamining occurrence:

$$\dfrac{\dfrac{\mathsf{conn}(\mathrm{Odense}, \mathrm{Munich})}{\mathsf{path}(\mathrm{Odense}, \mathrm{Munich})}\,\textsc{Dir} \qquad \dfrac{\dfrac{\dfrac{\mathsf{conn}(\mathrm{Paris}, \mathrm{Munich})}{\mathsf{conn}(\mathrm{Munich}, \mathrm{Paris})}\,\textsc{Sym}}{\mathsf{path}(\mathrm{Munich}, \mathrm{Paris})}\,\textsc{Dir} \qquad \dfrac{\mathsf{conn}(\mathrm{Paris}, \mathrm{Lisbon})}{\mathsf{path}(\mathrm{Paris}, \mathrm{Lisbon})}\,\textsc{Dir}}{\mathsf{path}(\mathrm{Munich}, \mathrm{Lisbon})}\,\textsc{Trans}}{\mathsf{path}(\mathrm{Odense}, \mathrm{Lisbon})}\,\textsc{Trans}\ . \tag{1.3}$$

What we have in eq. (1.3) is a complete derivation that is valid in the system in fig. 1.1.

### 1.4.2 Rule admissibility

We now move to the other direction: proving that adding rule TRANS to the system with rule STEP would not add any new derivable propositions.

Recall that rule TRANS is defined as follows.

$$\dfrac{\mathsf{path}(A, B) \qquad \mathsf{path}(B, C)}{\mathsf{path}(A, C)}\,\textsc{Trans}$$

As a first attempt, we could try the same strategy that we followed in section 1.4.1: deriving the conclusion $\mathsf{path}(A, C)$ from the premises $\mathsf{path}(A, B)$ and $\mathsf{path}(B, C)$ using the rules in fig. 1.4.

Unfortunately, we reach a dead end pretty quickly when trying to show that rule TRANS is derivable in the system with rule STEP. The only way to build a path with multiple connections is by using rule STEP, which requires a conn as premise. But our only available premises are path propositions, and we have no rule that allows us to conclude conn from a path.

We resort to a different proof technique and show that rule TRANS is *admissible* in the system with rule STEP (fig. 1.4). An admissible rule is one that does not add any new derivable propositions. All derivable rules are also admissible, but admissible rules are not necessarily derivable (just like our case here with rule

TRANS). It is sometimes convenient to mark admissible rules and their applications explicitly. Here, we will distinguish them by using a dashed horizontal line.

**Theorem 1.** *The rule*

$$\frac{\mathsf{path}(A,B) \qquad \mathsf{path}(B,C)}{\mathsf{path}(A,C)}\ \textsc{Trans}$$

*is admissible in the system in fig. 1.4.*

*Proof.* We need to prove that, for every derivation $\mathcal{D}$ using the rules in fig. 1.4 and rule TRANS, there exists a derivation with the same conclusion that uses only the rules in fig. 1.4. To shorten our text, we shall say that a derivation is TRANS-free when it does not contain any applications of rule TRANS.

We proceed by induction on the size of $\mathcal{D}$. We have a case for each one of the rules that can be applied last in $\mathcal{D}$. For the induction to work, we prove a stronger result: for every derivation $\mathcal{D}$ using the rules in fig. 1.4 and rule TRANS, there exists a derivation with the same conclusion that uses only the rules in fig. 1.4 *and is not bigger than* $\mathcal{D}$. This last remark about size is going to be important in the last case of this proof.

**Case** $\mathcal{D}$ ends with one of the axioms. In these cases $\mathcal{D}$ is trivially TRANS-free (as it consists only of the axiom), so the thesis holds by picking $\mathcal{D}$.

**Case** $\mathcal{D}$ ends with an application of rule SYM:

$$\mathcal{D} \;=\; \frac{\begin{array}{c}\mathcal{D}'\\ \mathsf{conn}(A,B)\end{array}}{\mathsf{conn}(B,A)}\ \textsc{Sym}$$

for some derivation $\mathcal{D}'$.

All rules that can conclude $\mathsf{conn}(A,B)$ are either axioms or rules that have only conn propositions as premises. Since TRANS does not conclude with a conn proposition, $\mathcal{D}'$ is necessarily TRANS-free and so is $\mathcal{D}$.

**Case** $\mathcal{D}$ ends with an application of rule DIR:

$$\mathcal{D} \;=\; \frac{\begin{array}{c}\mathcal{D}'\\ \mathsf{conn}(A,B)\end{array}}{\mathsf{path}(A,B)}\ \textsc{Dir}$$

for some derivation $\mathcal{D}'$.

By following the same reasoning for the previous case, we know that $\mathcal{D}'$ and $\mathcal{D}$ are TRANS-free.

**Case** $\mathcal{D}$ ends with an application of rule TRANS:

$$\mathcal{D} \;=\; \dfrac{\overset{\mathcal{E}}{\mathsf{path}(A,B)} \qquad \overset{\mathcal{F}}{\mathsf{path}(B,C)}}{\mathsf{path}(A,C)}\ \textsc{Trans}$$

for some derivations $\mathcal{E}$ and $\mathcal{F}$.

This is the most interesting case. By induction hypothesis, we know that there exists a TRANS-free derivation $\mathcal{E}'$ such that $\overset{\mathcal{E}'}{\mathsf{path}(A,B)}$.

There are only two possibilities for how $\mathcal{E}'$ ends: either with an application of rule DIR or with an application of rule STEP. This gives us two subcases.

**Case** For some $\mathcal{E}''$,

$$\mathcal{E}' \;=\; \dfrac{\overset{\mathcal{E}''}{\mathsf{conn}(A,B)}}{\mathsf{path}(A,B)}\ \textsc{Dir}\quad.$$

The thesis follows from the derivation

$$\dfrac{\overset{\mathcal{E}''}{\mathsf{conn}(A,B)} \qquad \overset{\mathcal{F}'}{\mathsf{path}(B,C)}}{\mathsf{path}(A,C)}\ \textsc{Step}\,.$$

**Case** For some $B'$, $\mathcal{E}_1$ and $\mathcal{E}_2$,

$$\mathcal{E}' \;=\; \dfrac{\overset{\mathcal{E}'_1}{\mathsf{conn}(A,B')} \qquad \overset{\mathcal{E}'_2}{\mathsf{path}(B',B)}}{\mathsf{path}(A,B)}\ \textsc{Step}\quad.$$

Consider the following derivation $\mathcal{G}$.

$$\mathcal{G} \;\triangleq\; \dfrac{\overset{\mathcal{E}'_2}{\mathsf{path}(B',B)} \qquad \overset{\mathcal{F}}{\mathsf{path}(B,C)}}{\mathsf{path}(B',C)}\ \textsc{Trans}$$

The derivation $\mathcal{G}$ is smaller than $\mathcal{D}$: $\mathcal{E}'_2$ is part of $\mathcal{E}'$, which by induction hypothesis is not bigger than $\mathcal{E}$; and $\mathcal{F}$ is part of $\mathcal{D}$. Thus, we can invoke the induction hypothesis on $\mathcal{G}$ and get that there exists a TRANS-free derivation $\mathcal{G}'$ with the same conclusion: $\overset{\mathcal{G}'}{\mathsf{path}(B',C)}$.

The thesis now follows by:

$$\dfrac{\overset{\mathcal{E}'_1}{\mathsf{conn}(A,B')} \qquad \overset{\mathcal{G}'}{\mathsf{path}(B',C)}}{\mathsf{path}(A,C)}\ \textsc{Step}\,.$$

□

**Example 1.** *To the reader not familiar with this kind of proofs, the last case might look like a bit of a magic trick! How are we manipulating the derivation, exactly?*

*Let us look at a concrete, nontrivial example of a proof $\mathcal{D}$ containing applications of* TRANS *(to help readability, we use the same derivation names as in the proof of theorem 1):*

$$\mathcal{D} \triangleq \dfrac{\overset{\mathcal{E}}{\mathsf{path}(\mathit{Odense}, \mathit{Paris})} \quad \overset{\mathcal{F}}{\mathsf{path}(\mathit{Paris}, \mathit{Lisbon})}}{\mathsf{path}(\mathit{Odense}, \mathit{Lisbon})}\ \textsc{Trans}$$

*where*

$$\mathcal{E} \triangleq \dfrac{\dfrac{\overline{\mathsf{conn}(\mathit{Odense}, \mathit{Munich})}}{\mathsf{path}(\mathit{Odense}, \mathit{Munich})}\ \textsc{Dir} \quad \dfrac{\dfrac{\overline{\mathsf{conn}(\mathit{Paris}, \mathit{Munich})}}{\mathsf{conn}(\mathit{Munich}, \mathit{Paris})}\ \textsc{Sym}}{\mathsf{path}(\mathit{Munich}, \mathit{Paris})}\ \textsc{Dir}}{\mathsf{path}(\mathit{Odense}, \mathit{Paris})}\ \textsc{Trans} \quad .$$

$$\mathcal{F} \triangleq \dfrac{\overline{\mathsf{conn}(\mathit{Paris}, \mathit{Lisbon})}}{\mathsf{path}(\mathit{Paris}, \mathit{Lisbon})}\ \textsc{Dir}$$

*We now "run" the proof of theorem 1 on $\mathcal{D}$. We are clearly in the last case, so we start by obtaining a* TRANS*-free derivation $\mathcal{E}'$ that has the same conclusion as $\mathcal{E}$:*

$$\mathcal{E}' = \dfrac{\overline{\mathsf{conn}(\mathit{Odense}, \mathit{Munich})} \quad \dfrac{\dfrac{\overline{\mathsf{conn}(\mathit{Paris}, \mathit{Munich})}}{\mathsf{conn}(\mathit{Munich}, \mathit{Paris})}\ \textsc{Sym}}{\mathsf{path}(\mathit{Munich}, \mathit{Paris})}\ \textsc{Dir}}{\mathsf{path}(\mathit{Odense}, \mathit{Paris})}\ \textsc{Step} \quad .$$

*$\mathcal{E}'$ is not bigger than $\mathcal{E}$ (it is actually smaller), as expected. We have that*

$$B' = \mathit{Munich}$$

$$\mathcal{E}'_1 = \overline{\mathsf{conn}(\mathit{Odense}, \mathit{Munich})}$$

$$\mathcal{E}'_2 = \dfrac{\dfrac{\overline{\mathsf{conn}(\mathit{Paris}, \mathit{Munich})}}{\mathsf{conn}(\mathit{Munich}, \mathit{Paris})}\ \textsc{Sym}}{\mathsf{path}(\mathit{Munich}, \mathit{Paris})}\ \textsc{Dir}$$

$$\mathcal{G} = \dfrac{\overset{\mathcal{E}'_2}{\mathsf{path}(\mathit{Munich}, \mathit{Paris})} \quad \overset{\mathcal{F}}{\mathsf{path}(\mathit{Paris}, \mathit{Lisbon})}}{\mathsf{path}(\mathit{Munich}, \mathit{Lisbon})}\ \textsc{Trans}$$

$$\frac{}{\mathsf{conn}(\text{Odense}, \text{Munich})} \quad \frac{}{\mathsf{conn}(\text{Munich}, \text{Rome})} \quad \frac{}{\mathsf{conn}(\text{Paris}, \text{Munich})}$$

$$\frac{}{\mathsf{conn}(\text{Paris}, \text{Lisbon})}$$

$$\frac{\mathsf{conn}(A, B)}{\mathsf{conn}(B, A)}\ \textsc{Sym}$$

$$\frac{\mathsf{conn}(A, B)}{\mathsf{path}(A, B, 1)}\ \textsc{DirW} \qquad \frac{\mathsf{conn}(A, B) \quad \mathsf{path}(B, C, n)}{\mathsf{path}(A, C, n+1)}\ \textsc{StepW}$$

Figure 1.5: A weighted version of the system in fig. 1.4.

*To obtain $\mathcal{G}'$, we apply recursively the same reasoning to $\mathcal{G}$, obtaining:*

$$\mathcal{G}' = \dfrac{\dfrac{}{\mathsf{conn}(\mathit{Munich}, \mathit{Paris})} \quad \dfrac{\dfrac{}{\mathsf{conn}(\mathit{Paris}, \mathit{Munich})}\ \textsc{Sym}}{\mathsf{conn}(\mathit{Munich}, \mathit{Paris})}\ \textsc{Dir} \quad \mathsf{path}(\mathit{Munich}, \mathit{Paris})}{\mathsf{path}(\mathit{Munich}, \mathit{Lisbon})}\ \textsc{Step} .$$

*The final* TRANS*-free result is:*

$$\mathcal{D}' = \dfrac{\overset{\mathcal{E}'_1}{\mathsf{conn}(\mathit{Odense}, \mathit{Munich})} \quad \overset{\mathcal{G}'}{\mathsf{path}(\mathit{Munich}, \mathit{Lisbon})}}{\mathsf{path}(\mathit{Odense}, \mathit{Lisbon})}\ \textsc{Step} .$$

**Exercise 7** (!)**.** *Prove that if* $\mathsf{path}(A, B)$ *is derivable in the system in fig. 1.4, then there exists a nonempty sequence of derivable propositions* $\mathsf{conn}(C_1, C'_1)$, …, $\mathsf{conn}(C_n, C'_n)$ *for some* $C_1$, …, $C_n$ *such that* $n > 0$, $C_1 = A$ *(the sequence starts from* $A$*), and* $C'_n = B$ *(the sequence ends at* $B$*).*

*Prove that if* $\mathsf{path}(A, B)$ *is derivable in the system in fig. 1.1, then there exists a nonempty sequence of derivable propositions* $\mathsf{conn}(C_1, C'_1)$, …, $\mathsf{conn}(C_n, C'_n)$ *for some* $C_1$, …, $C_n$ *such that* $n > 0$, $C_1 = A$ *(the sequence starts from* $A$*), and* $C'_n = B$ *(the sequence ends at* $B$*).*

**Exercise 8.** *Prove that, for any* $A$ *and* $B$*, if* $\mathsf{path}(A, B)$ *is derivable in the system in fig. 1.4 then there exists* $n$ *such that* $\mathsf{path}(A, B, n)$ *is derivable in the system in fig. 1.5.*

**Exercise 9.** *Prove that, for any $A$ and $B$, if* $\mathsf{path}(A, B)$ *is derivable in the system in fig. 1.1 then there exists $n$ such that* $\mathsf{path}(A, B, n)$ *is derivable in the system in fig. 1.5.*

Hint: *Use theorem 1 and the result of exercise 8.*

# Chapter 2

# Simple choreographies

Now that we have familiarised ourselves with formal systems based on inference rules, we can proceed to using them for the study of concurrency. We start in this chapter by building our first, and very simple, choreography model. Our aim is to design the simplest possible choreography language that captures the essence of what a choreography model is and how we can use it. We will add further features later on, by building on the basic concepts that we shall establish in this chapter. In a sense, you could consider the material in this chapter a sort of technical preliminaries.

The cornerstone of our study will be the notion of *process*, an independent agent that can perform local computation and communicate with other agents by means of message passing (Input/Output, or I/O for short). Processes are abstract representations of computer programs executed concurrently, each possessing an independent control state and memory. Essentially, what we are going to do is to use inference systems to model concurrent systems that consist of processes communicating with each other.

**Example 2.** *As guiding example for this chapter, suppose that we want to define a system that consists of two processes, called* Buyer *and* Seller. *Suppose also that we want these two processes to interact as follows:*

1. Buyer *sends the title of a book she wishes to buy to* Seller*;*

2. Seller *replies to* Buyer *with the price of the book.*

The description above is informal. However, it gives us some important indications on how a mathematical formalism for choreographies might look like: our description talks about *multiple* processes and how they interact. We are adopting a global view on all the interactions among the processes that we are interested in. More specifically: each step of our protocol talks about *both* the sender and the

$$C ::= \mathsf{p} \twoheadrightarrow \mathsf{q}; C \mid \mathbf{0}$$

Figure 2.1: Simple choreographies, syntax.

receiver of the communication; and we are explicitly ordering communications (as in the "Alice and Bob notation" from the Preface).

## 2.1 Syntax

Our first choreography model is called simple choreographies. We start by formalising its *syntax*, which defines the set of (choreographic) programs that can be written in this model.

We refer to processes by using process names. Let $Pid$ be an infinite set of *process names*, ranged over by $\mathsf{p}$, $\mathsf{q}$, ..., $\mathsf{t}$. We call a process names also *process identifier*, *pid* for short, recalling the nomenclature from operating systems. The syntax of simple choreographies is given by the grammar in fig. 2.1, where $C$ ranges over a choreography. Let $SimpleChor$ be the language of that grammar, i.e., the set of its derivable terms. In other words, $SimpleChor$ is the set of all simple choreographies. From the grammar, we can see that there are two forms that a choreography can take.

- The empty choreography $\mathbf{0}$ (also called the terminated choreography), which describes a terminated protocol with no further actions to perform. You can think of it as the end of a choreographic program.
- An interaction term $\mathsf{p} \twoheadrightarrow \mathsf{q}; C$, which means that a message is communicated from process $\mathsf{p}$ to process $\mathsf{q}$. After this interaction is performed, the choreography proceeds as defined by the continuation $C$.

The syntax of simple choreographies is minimalistic, and an interaction between two processes is meant as an atomic action—communications are synchronous.[1] We assume that communications are always between different processes, i.e., whenever we write $\mathsf{p} \twoheadrightarrow \mathsf{q}$, we require $\mathsf{p} \neq \mathsf{q}$.

**Example 3.** *The following choreography defines the behaviour that we informally described in example 2.*

$$\mathsf{Buyer} \twoheadrightarrow \mathsf{Seller}; \mathsf{Seller} \twoheadrightarrow \mathsf{Buyer}; \mathbf{0}$$

[1] We shall see how to deal with asynchronous communications later, in **??**.

*Note that what we have is actually a rather coarse* abstraction *of what we described in example 2, because we are not formalising* what *is being sent from a process to another. For example, the informal description stated that* Buyer *sends "the title of a book she wishes to buy" to* Seller *in the first interaction, but our choreography above does not define this part. It simply states that* Buyer *sends some unspecified message to* Seller*, and that* Seller *replies to* Buyer *afterwards. We are going to add the possibility to specify the content of messages later on.*

## 2.2 Semantics

Now that we can write simple choreographies, we give them a semantics in terms of an operational interpretation. The objective is to formalise abstractly the execution of a choreography. We use one of the most established approaches to define this kind of interpretation (perhaps *the* most established approach), called labelled transition systems.

**Definition 4** (Labelled transition system)**.** *A labelled transition system (lts) is a triple* $(S, L, \longrightarrow)$ *where* $S$ *is the set of* states*,* $L$ *is the set of* labels*, and* $\longrightarrow \subseteq S \times L \times S$ *is the* transition relation*.*

We will range over labels with $\mu$, $\mu'$, etc. Intuitively, labels represent what we can observe about the step from one state into another. For this reason, they are also sometimes called *actions* or *observables*. We adopt the standard shorthand notation for transitions: we write $s_1 \xrightarrow{\mu} s_2$ whenever $(s_1, \mu, s_2) \in \longrightarrow$, for some states $s_1$ and $s_2$ in $S$ and label $\mu$ in $L$.

To define an lts for simple choreographies, we follow the structural operational semantics approach by Plotkin [2004]: we define the behaviour of a (choreographic) program in terms of the behaviours of its parts.

**Definition 5** (Lts for simple choreographies)**.** *The lts of simple choreographies is the lts* $(S, L, \longrightarrow)$*, where:*

- *the set of states* $S$ *is the set of all choreographies, i.e.,* $S = SimpleChor$*;*
- *the set of labels* $L$ *is defined as the set of all possible communications between any two processes, i.e.,* $L = \{\mathsf{p} \texttt{ -> } \mathsf{q} \mid \mathsf{p}, \mathsf{q} \in Pid\}$*;*
- *the transition relation* $\longrightarrow$ *is the smallest relation satisfying the rule in fig. 2.2.*

Formally, the transition relation $\longrightarrow$ of simple choreographies is defined as the smallest relation satisfying the rule displayed in fig. 2.2. There is only one rule,

$$\frac{}{\mathsf{p} \to \mathsf{q}; C \xrightarrow{\mathsf{p}\to\mathsf{q}} C}\ \textsc{Com}$$

Figure 2.2: Simple choreographies, semantics.

called Com, which is an axiom: it always allows us to execute interactions—if a programmer wishes for an interaction to take place, it always will. In the rule, p, q, and $C$ are all schematic variables, on which we impose no conditions. So the rule works for all process names and choreographies. (Recall, however, that we assumed $\mathsf{p} \neq \mathsf{q}$ in communication terms, so this is assumed also here.)

**Example 4.** *Let $C$ be the program from example 3:*

$$C \triangleq \mathsf{Buyer} \to \mathsf{Seller}; \mathsf{Seller} \to \mathsf{Buyer}; \mathbf{0}\,.$$

*We have the following transition*

$$C \xrightarrow{\mathsf{Buyer}\to\mathsf{Seller}} \mathsf{Seller} \to \mathsf{Buyer}; \mathbf{0}$$

*which formally tells us that $C$ can execute a communication from* Buyer *to* Seller *(from the label* Buyer $\to$ Seller*). The transition transforms the program into its continuation, from which we can observe a further transition:*

$$\mathsf{Seller} \to \mathsf{Buyer}; \mathbf{0} \xrightarrow{\mathsf{Seller}\to\mathsf{Buyer}} \mathbf{0}\,.$$

*There are no further transitions, since we reached term* $\mathbf{0}$*, for which there are no transition rules.*

*So the execution of $C$ is that we first have a communication from* Buyer *to* Seller *and then a communication from* Seller *to* Buyer*, which is exactly the communication flow that we wanted in example 2.*

**Conventions on labelled transition systems for programs** In this chapter, we have defined all the components of the lts of simple choreographies explicitly: the set of states, the set of labels, and the transition relation. However, once a grammar like that in fig. 2.1 and a set of inference rules for the transition relation like that in fig. 2.2 are given, this level of detail is unnecessary with a few conventions. These conventions will save us quite a bit of ink in the next chapters, since we will update the syntax and semantics of choreographies a few times to explore different directions and make them more powerful.

First, we can just assume that the set of states of a choreography model is the language of its grammar. Second, let the set of labels for a choreography model be the union of all labels that can be instantiated by replacing the schematic variables

in the inference system for its semantics. Third, let the transition relation always be the smallest relation satisfying the rules of the inference system given for the choreography model.

With these conventions in place, we just need to give the syntax and transition inference system for a choreography model and we will immediately know its intended lts. For example, for simple choreographies, we would just need to present the syntax in fig. 2.1 and the rules in fig. 2.2. We follow these conventions in the remainder.

**Other notations and terminologies for labelled transition systems** We define some useful notations and terminologies for labelled transition systems, adapting some from Sangiorgi [2011].

Whenever $C \xrightarrow{\mu} C'$, we say that $C'$ is a *$\mu$-derivative* of $C$, or sometimes simply a *derivative* of $C$.

It can be useful to combine transitions. Let $\vec{\mu}$ be a shorthand for a sequence of labels $\mu_1, \ldots, \mu_n$, for some $n$; then $C \xrightarrow{\vec{\mu}} C'$ holds whenever there exist $C_1, \ldots, C_{n-1}$ such that $C \xrightarrow{\mu_1} C_1 \cdots C_{n-1} \xrightarrow{\mu_n} C'$. When $C \xrightarrow{\vec{\mu}} C'$ for some choreographies $C$ and $C'$ and sequence of labels $\vec{\mu}$, we say that $C'$ is a derivative of $C$ under $\vec{\mu}$, or simply a *multi-step derivative* of $C$. We also write $C \xrightarrow{\vec{\mu}}\xrightarrow{\mu} C'$ whenever there exists $C''$ such that $C \xrightarrow{\vec{\mu}} C''$ and $C'' \xrightarrow{\mu} C'$.

We write $C \xrightarrow{\mu}$ (read "$C$ can make a transition with label $\mu$") whenever there exists some $C'$ such that $C \xrightarrow{\mu} C'$. Likewise, we write $C \not\xrightarrow{\mu}$ (read "$C$ cannot make a transition with label $\mu$") whenever there exists no $C'$ such that $C \xrightarrow{\mu} C'$.

For example, $\mathbf{0} \not\xrightarrow{\mu}$ for any label $\mu$, which formalises that $\mathbf{0}$ represents the terminated choreography.

**Example 5.** *With our new notation in place, we can show a complete execution trace of the choreography $C$ from example 3 in a single shot. Recall the choreography:*

$$C \triangleq \mathsf{Buyer} \to \mathsf{Seller}; \mathsf{Seller} \to \mathsf{Buyer}; \mathbf{0}\,.$$

*We have the following transitions:*

$$C \xrightarrow{\mathsf{Buyer}\to\mathsf{Seller}} \xrightarrow{\mathsf{Seller}\to\mathsf{Buyer}} \mathbf{0}\,.$$

*So we first have an interaction where* Buyer *sends a message to* Seller *and then we have an interaction where* Seller *sends a message to* Buyer*, which is exactly the communication flow that we wanted in example 2. Thus, the formal semantics of $C$ matches our informal description.*

**Proposition 1** (Strong termination for simple choreographies)**.** *For any simple choreography $C$ such that $C \neq \mathbf{0}$ ($C$ is not the terminated choreography), there exists a sequence of labels $\vec{\mu}$ such that $C \xrightarrow{\vec{\mu}} \mathbf{0}$.*

**Exercise 10** ($\hookrightarrow$)**.** *Prove proposition 1.* Hint: *proceed by induction on the structure of the choreography $C$.*

# Chapter 3

# From choreographic programs to process programs

Simple choreographies is like a high-level programming language, providing us with a useful abstraction (the communication term) to talk about what we are interested in. In our case, what we are interested in is defining the interactions that we want to take place among our processes—we want to write protocols. However, high-level programs are not particularly useful if we do not know how they can be implemented in practice. Other high-level languages face the same situation: they offer useful abstractions, like functions and objects, but their programs need to be converted to something that the computer can actually execute, like machine code. Here, we are not interested in reaching the detail of machine code, but in bridging the conceptual difference between choreographies and the typical programs that can be executed in concurrent systems.

In a concurrent and distributed system, each process has its own program that is run by the computer that hosts it. To communicate, processes can then send and receive messages to/from each other. A process does not (necessarily) know what programs the other processes are running, only its own. This is different from choreographies, where the behaviour of multiple processes is defined from a global viewpoint.

**Example 6.** *To implement our scenario from example 2 following the standard methodology for programming concurrent and distributed systems, we would have to produce two programs: one for process* Buyer *and one for process* Seller.

*Informally, a program for* Buyer *could look as follows.*

- *Send the title of the book to buy to* Seller*;*
- *receive the price of the book from* Seller.

*For* Seller*, we could use the following (abstract) program.*

$$P, Q, R ::= \mathsf{p}!; P \mid \mathsf{p}?; P \mid \mathbf{0}$$

Figure 3.1: Simple processes, syntax.

- *Receive the title of the book from* Buyer*;*
- *send the price of the book to* Buyer.

In a nutshell, choreographies give us a global view on the communications to be enacted by the system; whereas realistic concurrent and distributed systems expect to have a program for each process (we call these *process programs*), based on the local view of that process and using send and receive actions (also called Input/Output, or I/O) to interact with the other processes.

Thus, if we want to make choreographies useful, we need the following two elements.

- A formal model for process programs, or *process model*.
- A way to relate our choreography model (simple choreographies) to the process model or, more concretely, choreography programs to process programs.

## 3.1 Simple processes

In this section we define a simple process model to describe system implementations. We will use it later to build a notion of *correspondence* between what should happen (given as a choreography) and what the system actually does (given as a term in this process model).

We assume that each process is uniquely identified by its name on the network, and that process names can be used to direct messages from one process to each other (similarly to URLs on the web). based on actors.

### 3.1.1 Syntax

The syntax of simple processes is given in fig. 3.1. A process term $P$ can be:

- a send action $\mathsf{p}!; P$, read "send a message to process $\mathsf{p}$ and then do $P$";
- a receive action $\mathsf{p}?; P$, read "receive a message from process $\mathsf{p}$ and then do $P$";

- or the inactive process $\mathbf{0}$.

We write $SimpleProc$ for the set of all process terms in simple processes. We call process terms also process programs, or simply processes.

Notice that we are using $\mathbf{0}$ to represent both inactive choreographies and processes, so our notation is overloaded. This is intuitive because they are both "no-op" terms, i.e., they perform no actions.

We model systems where processes can interact as functions that map process names to process terms, called *networks*. A preliminary ingredient is required to reach the formal definition. Let $N$ be a function from process names to terms:

$$N : Pid \longrightarrow SimpleProc\,.$$

The *support* of $N$, written $\mathsf{supp}(N)$, is the subset of the domain $Pid$ containing those process names that are not mapped to $\mathbf{0}$; in set-builder notation:

$$\mathsf{supp}(N) \;=\; \{\mathsf{p} \mid N(\mathsf{p}) \neq \mathbf{0}\}\,.$$

We say that a function $N$ has *finite support* if the set $\mathsf{supp}(N)$ is finite.

**Definition 6** (Network)**.** *A network $N$ is a function with finite support from process names to process terms:*

$$N : Pid \longrightarrow SimpleProc\,.$$

**Notation for networks** Since networks have finite support, we can define any network precisely using a finite representation. To do this, we introduce a notation inspired by other works on process models [Milner 1980, Hennessy 2007]. Intuitively, we write $\mathsf{p}[P]$ for "process $\mathsf{p}$ runs the program $P$", and use the *parallel operator* "|" to compose such processes in systems where all processes run in parallel.

Formally, let $\mathsf{p} \in Pid$. Then, we write $\mathsf{p}[P]$ for the network $N$ such that

$$N(\mathsf{q}) \;=\; \begin{cases} P & \text{if } \mathsf{q} = \mathsf{p} \\ \mathbf{0} & \text{otherwise} \end{cases}$$

Given any two sets $S_1$ and $S_2$, we write $S_1 \# S_2$ whenever $S_1 \cap S_2 = \emptyset$, that is, the two sets are disjoint (they do not share any elements). Let $N$ and $M$ be networks with disjoint supports, i.e., $\mathsf{supp}(N) \# \mathsf{supp}(M)$. Then, we write $N \mid M$ for the network obtained by composing $N$ and $M$:

$$(N \mid M)\,(\mathsf{p}) \;=\; \begin{cases} N(\mathsf{p}) & \text{if } \mathsf{p} \in \mathsf{supp}(N) \\ M(\mathsf{p}) & \text{otherwise} \end{cases}\,.$$

The *terminated network* $\mathbf{0}$ is the network such that:

$$\mathbf{0}(\mathsf{p}) = \mathbf{0} \quad \text{for all } \mathsf{p} \in \mathit{Pid}\,.$$

Again, there is no ambiguity in using the same symbol $\mathbf{0}$, as we did for the terminated choreography and process term, since the context will always clarify what we are referring to. On the other hand, $\mathbf{0}$ gives us the useful mnemonic of an "inactive" object.

**Example 7.** *The two process programs informally described in example 6 can be formalised as the following network.*

$$\mathsf{Buyer}[\mathsf{Seller}!; \mathsf{Seller}?; \mathbf{0}] \mid \mathsf{Seller}[\mathsf{Buyer}?; \mathsf{Buyer}!; \mathbf{0}]$$

**Exercise 11** (Unit of parallel composition)**.** *Prove that* $\mathbf{0}$ *is the unit of parallel composition, that is, that for all* $N$*,*

$$\mathbf{0} \mid N = N\,.$$

**Exercise 12** (Commutativity of parallel composition)**.** *Prove that the parallel operator is commutative, that is, for all* $N$ *and* $M$*,*

$$N \mid M = M \mid N\,.$$

Hint: *Recall that* $N \mid M$ *is defined only when* $\mathsf{supp}(N) \# \mathsf{supp}(M)$.

**Exercise 13** (Associativity of parallel composition)**.** *Prove that the parallel operator is associative, that is, for all* $N_1$*,* $N_2$*, and* $N_3$*,*

$$N_1 \mid (N_2 \mid N_3) \;=\; (N_1 \mid N_2) \mid N_3\,.$$

### 3.1.2 Semantics

Similarly to what we have done for simple choreographies, we equip simple processes with a labelled transition system semantics. The idea is that each transition models a step in the execution of a network.

Formally, the lts of simple processes is given by the inference system displayed in fig. 3.2—recall our convention for defining an lts from an inference system from section 2.2. There are two rules. The first rule, called COM, is an axiom:

$$\frac{}{\mathsf{p}[\mathsf{q}!; P] \mid \mathsf{q}[\mathsf{p}?; Q] \xrightarrow{\mathsf{p}\to\mathsf{q}} \mathsf{p}[P] \mid \mathsf{q}[Q]}\ \textsc{Com}\,.$$

Rule COM matches a send action from a process $\mathsf{p}$ to a process $\mathsf{q}$ with the compatible receive action by process $\mathsf{q}$ from $\mathsf{p}$. The two processes then proceed with

$$\frac{}{\mathsf{p}[\mathsf{q}!; P] \mid \mathsf{q}[\mathsf{p}?; Q] \xrightarrow{\mathsf{p}\to\mathsf{q}} \mathsf{p}[P] \mid \mathsf{q}[Q]}\ \textsc{Com} \qquad \frac{N \xrightarrow{\mu} N'}{N \mid M \xrightarrow{\mu} N' \mid M}\ \textsc{Par}$$

Figure 3.2: Simple processes, semantics.

their respective continuations. The second rule, called PAR, states that if a part of a network can perform a transition, then this can be observed also in a larger context:

$$\frac{N \xrightarrow{\mu} N'}{N \mid M \xrightarrow{\mu} N' \mid M}\ \textsc{Par} .$$

Observe that the part of the network not affected by the transition remains ($M$) remains unaffected. Also, since the parallel operator for networks is commutative and associative (**??** 12**??** 13), the rule supports performing transitions in any part of the network.

**Remark 1.** *We use the same name (*COM*) for the communication rules of both choreographies and processes, as a mnemonic that the two rules intuitively model the same concept. This causes no ambiguity, since the two rules act on different syntactic terms, so their usages will be clear from the context.*

**Example 8.** *Let $N$ be the network from example 7:*

$$\mathsf{Buyer}[\mathsf{Seller}!; \mathsf{Seller}?; \mathbf{0}] \mid \mathsf{Seller}[\mathsf{Buyer}?; \mathsf{Buyer}!; \mathbf{0}] .$$

*We can observe the intended communication between* Buyer *and* Seller *by rule* COM*:*

$$\frac{}{\begin{array}{l}\mathsf{Buyer}[\mathsf{Seller}!; \mathsf{Seller}?; \mathbf{0}] \mid \\ \mathsf{Seller}[\mathsf{Buyer}?; \mathsf{Buyer}!; \mathbf{0}]\end{array} \xrightarrow{\mathsf{Buyer}\to\mathsf{Seller}} \begin{array}{l}\mathsf{Buyer}[\mathsf{Seller}?; \mathbf{0}] \mid \\ \mathsf{Seller}[\mathsf{Buyer}!; \mathbf{0}]\end{array}}\ \textsc{Com} .$$

*Let $N'$ be the derivative above. We can proceed further and observe that $N'$ can terminate by performing the expected second communication.*

$$\frac{}{N' \xrightarrow{\mathsf{Seller}\to\mathsf{Buyer}} \mathsf{Seller}[\mathbf{0}] \mid \mathsf{Buyer}[\mathbf{0}]}\ \textsc{Com} .$$

*Observe that the derivative above is equal to the terminated network, $\mathbf{0}$:*

$$\mathsf{Seller}[\mathbf{0}] \mid \mathsf{Buyer}[\mathbf{0}] \ = \ \mathbf{0} .$$

*In other words, we have that $\mathbf{0}$ is a derivative of $N$ under* Buyer -> Seller, Seller -> Buyer*, or more concisely:*

$$N \xrightarrow{\mathsf{Buyer}\to\mathsf{Seller}} \xrightarrow{\mathsf{Seller}\to\mathsf{Buyer}} \mathbf{0} .$$

**Example 9.** *Let $N$ be the following network of three processes, for some* $\mathsf{p}$, $\mathsf{q}$, *and* $\mathsf{r}$. *The idea is that* $\mathsf{q}$ *forwards a message from* $\mathsf{p}$ *to* $\mathsf{r}$.

$$N \triangleq \mathsf{p}[\mathsf{q}!;\mathbf{0}] \mid \mathsf{q}[\mathsf{p}?;\mathsf{r}!;\mathbf{0}] \mid \mathsf{r}[\mathsf{q}?;\mathbf{0}]$$

*We can derive the first communication from* $\mathsf{p}$ *to* $\mathsf{q}$ *by using rule* PAR*:*

$$\dfrac{\dfrac{}{\mathsf{p}[\mathsf{q}!;\mathbf{0}] \mid \mathsf{q}[\mathsf{p}?;\mathsf{r}!;\mathbf{0}] \xrightarrow{\mathsf{p}\to\mathsf{q}} \mathsf{q}[\mathsf{r}!;\mathbf{0}]}\ \textsc{Com}}{\mathsf{p}[\mathsf{q}!;\mathbf{0}] \mid \mathsf{q}[\mathsf{p}?;\mathsf{r}!;\mathbf{0}] \mid \mathsf{r}[\mathsf{q}?;\mathbf{0}] \xrightarrow{\mathsf{p}\to\mathsf{q}} \mathsf{q}[\mathsf{r}!;\mathbf{0}] \mid \mathsf{r}[\mathsf{q}?;\mathbf{0}]}\ \textsc{Par}\ .$$

*By rule* COM*, the derivative above can terminate.*

$$\dfrac{}{\mathsf{q}[\mathsf{r}!;\mathbf{0}] \mid \mathsf{r}[\mathsf{q}?;\mathbf{0}] \xrightarrow{\mathsf{q}\to\mathsf{r}} \mathbf{0}}\ \textsc{Com}$$

*Hence,* $N \xrightarrow{\mathsf{p}\to\mathsf{q}}\xrightarrow{\mathsf{q}\to\mathsf{r}} \mathbf{0}$.

**Exercise 14.** *Let $N$ be defined as:*

$$N \triangleq \mathsf{p}[\mathsf{q}?;\mathsf{r}?;\mathsf{r}!;\mathbf{0}] \mid \mathsf{q}[\mathsf{p}!;\mathbf{0}] \mid \mathsf{r}[\mathsf{p}!;\mathsf{p}?;\mathsf{s}!;\mathbf{0}] \mid \mathsf{s}[\mathsf{r}?;\mathbf{0}]\ .$$

*Prove that there exists a sequence of labels $\vec{\mu}$ such that $N \xrightarrow{\vec{\mu}} \mathbf{0}$.*

Hint: *use the commutativity and associativity properties of parallel composition to apply rule* PAR *correctly in your derivations.*

Differently from choreographies, it is possible to write networks that can get "stuck", in the sense that they may not terminate. In fact, there is no equivalent of proposition 1 for simple processes, since there exist networks that might not terminate.

**Exercise 15.** *Prove the following propositions.*

1. $\mathsf{p}[\mathsf{q}!;\mathbf{0}] \overset{\mu}{\not\to}$ *for any* $\mu$.
2. $\mathsf{p}[\mathsf{q}!;\mathbf{0}] \mid \mathsf{q}[\mathsf{p}!;\mathbf{0}] \overset{\mu}{\not\to}$ *for any* $\mu$.
3. *There is no* $\vec{\mu}$ *such that*

$$\mathsf{p}[\mathsf{q}?;\mathsf{r}!;\mathsf{q}?;\mathbf{0}] \mid \mathsf{q}[\mathsf{p}!;\mathsf{p}!;\mathbf{0}] \mid \mathsf{r}[\mathsf{q}?;\mathsf{p}?;\mathbf{0}] \xrightarrow{\vec{\mu}} \mathbf{0}\ .$$

**Lemma 1.** *For any $N$, $\mu$, and $N'$, if $N \xrightarrow{\mu} N'$ then $N(\mathsf{r}) = N'(\mathsf{r})$ for all* $\mathsf{r}$ *such that* $\mathsf{r} \notin \mathsf{pn}(\mu)$.

**Exercise 16 ($\hookrightarrow$).** *Prove lemma 1.*

Hint: *Proceed by induction on the derivation of the transition $N \xrightarrow{\mu} N'$.*

## 3.2 Endpoint Projection (EPP)

The network in example 7 works as intended, but we had to come up with it manually. If we could figure out a mechanical method of going from a choreography (which formalises what we want) to a network (which formalises an implementation), we would save time. If we could also prove that such method *always gives us a correct result*, we would also save ourselves the potential mistakes that come from the manual activity of writing a process network that should implement what we want.

### 3.2.1 From choreographies to processes

**An operational intuition** To gain some intuition on how we could develop a method to translate choreographies into processes, we can look at our examples. Let $C$ be the choreography from example 3, i.e.,

$$C \triangleq \mathsf{Buyer} \to \mathsf{Seller}; \mathsf{Seller} \to \mathsf{Buyer}; \mathbf{0} \ ,$$

and let $N$ be the network from example 7, i.e.,

$$N \triangleq \mathsf{Buyer}[\mathsf{Seller}!; \mathsf{Seller}?; \mathbf{0}] \mid \mathsf{Seller}[\mathsf{Buyer}?; \mathsf{Buyer}!; \mathbf{0}] \ .$$

Intuitively, $N$ is a correct process implementation of $C$, in the sense that $N$ performs exactly the communications prescribed by $C$. We can check for this by looking at the respective transitions of the choreography and the network. The only transition that $C$ can perform is

$$C \xrightarrow{\mathsf{Buyer}\to\mathsf{Seller}} \mathsf{Seller} \to \mathsf{Buyer}; \mathbf{0} \ ,$$

and the only transition that $N$ can perform is

$$\begin{array}{l}\mathsf{Buyer}[\mathsf{Seller}!; \mathsf{Seller}?; \mathbf{0}] \mid \\ \mathsf{Seller}[\mathsf{Buyer}?; \mathsf{Buyer}!; \mathbf{0}]\end{array} \xrightarrow{\mathsf{Buyer}\to\mathsf{Seller}} \begin{array}{l}\mathsf{Buyer}[\mathsf{Seller}?; \mathbf{0}] \mid \\ \mathsf{Seller}[\mathsf{Buyer}!; \mathbf{0}]\end{array} \ .$$

These two transitions have the same label, so they correspond to each other. If we look at the derivatives of $C$ and $N$, they have just one possible transition each, which also correspond to each other:

$$\begin{array}{c}\mathsf{Seller} \to \mathsf{Buyer}; \mathbf{0} \xrightarrow{\mathsf{Seller}\to\mathsf{Buyer}} \mathbf{0} \\ \mathsf{Buyer}[\mathsf{Seller}?; \mathbf{0}] \mid \mathsf{Seller}[\mathsf{Buyer}!; \mathbf{0}] \xrightarrow{\mathsf{Seller}\to\mathsf{Buyer}} \mathsf{Buyer}[\mathbf{0}] \mid \mathsf{Seller}[\mathbf{0}]\end{array} \ .$$

Now the choreography has terminated and the network cannot make any further transitions with observable labels.

$$\llbracket \mathsf{p} \to \mathsf{q}; C \rrbracket_{\mathsf{r}} = \begin{cases} \mathsf{q}!; \llbracket C \rrbracket_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{p} \\ \mathsf{p}?; \llbracket C \rrbracket_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{q} \\ \llbracket C \rrbracket_{\mathsf{r}} & \text{otherwise} \end{cases}$$

$$\llbracket \mathbf{0} \rrbracket_{\mathsf{p}} = \mathbf{0}$$

Figure 3.3: Process projection for simple choreographies.

What we learn from this example is that a network "implements" a choreography correctly if the two can "mimic" each other transitions, using the same labels. In other words, executing a network that implements a choreography correctly gives rise to the interactions described in the choreography.

**Formal definition** We can now move to formally defining our desired method for translating choreographies into correct process implementations, as a function from choreographies to networks. This function is commonly called *EndPoint Projection* (EPP for short), since it projects each interaction in the choreography to the local action that each process (an endpoint) should perform in the network [Qiu et al. 2007, Lanese et al. 2008, Carbone et al. 2012]. Indeed, we can think of an interaction like Buyer -> Seller as consisting of two parts, i.e., the send action by Buyer and the receive action by Seller. So the send action that the process implementing Buyer should perform is the first component of the interaction, and the receive action by Seller is the second component.

The key part of the definition of EPP is a function called *process projection*, written $\llbracket \bullet \rrbracket_{\bullet}$. Intuitively, given a choreography $C$ and a process name $\mathsf{p}$, $\llbracket C \rrbracket_{\mathsf{p}}$ returns the process term that defines the actions that $\mathsf{p}$ should execute in order to play its part as specified by $C$. Process projection is defined by structural recursion on choreographies, as follows.

**Definition 7** (Process projection)**.** *Let $C$ be a choreography and $\mathsf{p}$ a process name ($\mathsf{p} \in Pid$). The projection of $C$ for $\mathsf{p}$, written $\llbracket C \rrbracket_{\mathsf{p}}$, is defined by the rules given in fig. 3.3.*

**Example 10.** *Let $C_{\mathsf{BuyerSeller}}$ be the choreography in example 3:*

$$C_{\mathsf{BuyerSeller}} \triangleq \mathsf{Buyer} \to \mathsf{Seller}; \mathsf{Seller} \to \mathsf{Buyer}; \mathbf{0}\,.$$

*If we construct a network consisting of the process projections for* Buyer *and* Seller*, we obtain the correct implementation of $C_{\mathsf{BuyerSeller}}$ that we have manually written before:*

$$\mathsf{Buyer}\left[\llbracket C \rrbracket_{\mathsf{Buyer}}\right] \mid \mathsf{Seller}[\llbracket C \rrbracket_{\mathsf{Seller}}] = \begin{array}{l} \mathsf{Buyer}[\mathsf{Seller}!; \mathsf{Seller}?; \mathbf{0}] \mid \\ \mathsf{Seller}[\mathsf{Buyer}?; \mathsf{Buyer}!; \mathbf{0}] \end{array}\,.$$

The definition of EPP is obtained by generalising the idea to map each process name to the related process projection for the originating choreography. We write $[\![C]\!]$ for the EPP of a choreography $C$. Formally, $[\![\bullet]\!]$ is a function from choreographies to networks defined as follows.

**Definition 8** (EndPoint Projection (EPP))**.** *Let $C$ be a choreography. The endpoint projection (EPP) of $C$, written $[\![C]\!]$, is the network such that, for all $\mathsf{p} \in Pid$,*

$$[\![C]\!](\mathsf{p}) = [\![C]\!]_{\mathsf{p}} \ .$$

**Example 11.** *Let $C_{\mathsf{BuyerSeller}}$ be the choreography in example 3:*

$$C_{\mathsf{BuyerSeller}} \quad \triangleq \quad \mathsf{Buyer} \to \mathsf{Seller}; \mathsf{Seller} \to \mathsf{Buyer}; \mathbf{0} \ .$$

*The EPP of $C_{\mathsf{BuyerSeller}}$ is exactly the network that we defined in example 7:*

$$[\![C_{\mathsf{BuyerSeller}}]\!] \ = \ \mathsf{Buyer}[\mathsf{Seller}!; \mathsf{Seller}?; \mathbf{0}] \ \mid \ \mathsf{Seller}[\mathsf{Buyer}?; \mathsf{Buyer}!; \mathbf{0}] \ .$$

**Exercise 17** ($\hookrightarrow$)**.** *Write the EPP of the choreography*

$$\mathsf{p} \to \mathsf{q}; \mathsf{r} \to \mathsf{q}; \mathsf{r} \to \mathsf{s}; \mathsf{q} \to \mathsf{p}; \mathbf{0} \ .$$

**A sanity check on EPP** Definition 8 claims that EPP produces a network, and definition 6 requires every network to have finite support. Thus, as a sanity check, we should check that $[\![C]\!]$ has finite support for all choreographies $C$.

By definition of support, the support of $[\![C]\!]$ is defined as

$$\mathsf{supp}([\![C]\!]) \ = \ \{\mathsf{p} \mid [\![C]\!](\mathsf{p}) \neq \mathbf{0}\} \ .$$

Since $[\![C]\!](\mathsf{p}) = [\![C]\!]_{\mathsf{p}}$, we can rewrite the equation above as

$$\mathsf{supp}([\![C]\!]) \ = \ \{\mathsf{p} \mid [\![C]\!]_{\mathsf{p}} \neq \mathbf{0}\} \ .$$

The question is thus whether the set

$$\{\mathsf{p} \mid [\![C]\!]_{\mathsf{p}} \neq \mathbf{0}\}$$

is finite. Intuitively, it is, since a choreography can only mention a finite number of processes. Let us prove this formally.

First, we define a function, $\mathsf{pn}$, that maps each choreography to the set of process names that it mentions. We define $\mathsf{pn}$ by recursion on the structure of the choreography, as follows.

$$\begin{aligned} \mathsf{pn}(\mathbf{0}) &= \emptyset \\ \mathsf{pn}(\mathsf{p} \to \mathsf{q}; C) &= \{\mathsf{p}, \mathsf{q}\} \cup \mathsf{pn}(C) \end{aligned}$$

Given any choreography $C$, it is easy to prove that $\mathsf{pn}(C)$ is a finite set.

**Proposition 2.** *For any $C$, $\mathsf{pn}(C)$ is a finite set.*

*Proof.* By induction on $C$.

**Case** $C = \mathbf{0}$. In this case, $\mathsf{pn}(\mathbf{0}) = \emptyset$, which is a finite set.

**Case** $C = \mathsf{p} \rightarrow \mathsf{q}; C'$ for some $\mathsf{p}$, $\mathsf{q}$, and $C'$. By induction hypothesis, we know that $\mathsf{pn}(C')$ is a finite set. By definition of $\mathsf{pn}$, $\mathsf{pn}(C) = \{\mathsf{p}, \mathsf{q}\} \cup C'$. So $\mathsf{pn}(C)$ is the union of two finite sets, which is also a finite set.

□

Now we prove that, for any $C$, if a process name does not appear in $\mathsf{pn}(C)$, then $[\![C]\!]_{\mathsf{p}} = \mathbf{0}$.

**Lemma 2.** *For any $C$ and $\mathsf{p}$, $\mathsf{p} \notin \mathsf{pn}(C)$ implies $[\![C]\!]_{\mathsf{p}} = \mathbf{0}$.*

*Proof.* By induction on $C$.

**Case** $C = \mathbf{0}$. In this case, the thesis follows trivially because $[\![C]\!]_{\mathsf{p}} = \mathbf{0}$ for all $\mathsf{p}$.

**Case** $C = \mathsf{q} \rightarrow \mathsf{r}; C'$ for some $\mathsf{q}$, $\mathsf{r}$, and $C'$. By definition of $\mathsf{pn}$,

$$\mathsf{pn}(C) = \{\mathsf{q}, \mathsf{r}\} \cup C' \,.$$

Hence, by the assumption $\mathsf{p} \notin \mathsf{pn}(C)$, we have that (1) $\mathsf{p} \notin \{\mathsf{q}, \mathsf{r}\}$ and (2) $\mathsf{p} \notin \mathsf{pn}(C')$. From (1) and the definition of EPP, we get $[\![C]\!]_{\mathsf{p}} = [\![C']\!]_{\mathsf{p}}$. From (2), we can apply the induction hypothesis to $[\![C']\!]$ and obtain that $[\![C']\!]_{\mathsf{p}} = \mathbf{0}$.

□

**Corollary 1.** *For any $C$, $[\![C]\!]$ has finite support.*

*Proof.* Direct consequence of proposition 2 and lemma 2. □

**Exercise 18.** *Prove that, for any $C$, $\mathsf{supp}([\![C]\!]) = \mathsf{pn}(C)$.*

**Exercise 19** (!)**.** *Another way of proving the result of corollary 1 is to proceed by structural induction on the choreography $C$ and build directly the support of $[\![C]\!]$. Develop such proof.*

Hint: *follow the structure of the proof of proposition 2.*

### 3.2.2 Towards a correct EPP

That EPP works for our simple buyer-seller example is a good sign. But does EPP work *in general*? That is, are all choreographies projected to a correct network implementation that does exactly what the originating choreography prescribes?

Unfortunately, we can easily find a counterexample where the EPP of a choreography performs transitions that are not allowed for by the choreography. The counterexample is simple. Take the following choreography:

$$C_{\mathsf{problem}} \triangleq \mathsf{p} \to \mathsf{q}; \mathsf{r} \to \mathsf{s}; \mathbf{0} \,. \tag{3.1}$$

The EPP of this choreography is:

$$[\![C_{\mathsf{problem}}]\!] = \mathsf{p}[\mathsf{q}!; \mathbf{0}] \mid \mathsf{q}[\mathsf{p}?; \mathbf{0}] \mid \mathsf{r}[\mathsf{s}!; \mathbf{0}] \mid \mathsf{s}[\mathsf{r}?; \mathbf{0}] \,.$$

We have the following transition for this network, by synchronising $\mathsf{r}$ with $\mathsf{s}$:

$$\mathsf{p}[\mathsf{q}!; \mathbf{0}] \mid \mathsf{q}[\mathsf{p}?; \mathbf{0}] \mid \mathsf{r}[\mathsf{s}!; \mathbf{0}] \mid \mathsf{s}[\mathsf{r}?; \mathbf{0}] \quad \xrightarrow{\mathsf{r}\to\mathsf{s}} \quad \mathsf{p}[\mathsf{q}!; \mathbf{0}] \mid \mathsf{q}[\mathsf{p}?; \mathbf{0}] \mid \mathsf{r}[\mathbf{0}] \mid \mathsf{s}[\mathbf{0}] \,.$$

However, $C_{\mathsf{problem}}$ cannot mimic this transition: $C_{\mathsf{problem}} \not\xrightarrow{\mathsf{r}\to\mathsf{s}}$. Instead, it can only perform the first interaction between $\mathsf{p}$ and $\mathsf{q}$ according to the semantics of simple choreographies.

Choreography $C_{\mathsf{problem}}$ exemplifies a general problem: our framework is not sound yet, because the projection of a choreography can perform "extra" transitions with respect to the choreography. If we look closely though, the communication $\mathsf{r} \to \mathsf{s}$ executed by the EPP of $C_{\mathsf{problem}}$ *is* indeed defined in the choreography; it is just happening *too early*. There are two ways to fix this: forbidding independent sequences of interactions, or allowing for out of order execution in choreographies.

**Forbidding independent sequences of interactions** One way is to say that choreographies like $C_{\mathsf{problem}}$ are "forbidden", because the sequence of interactions $\mathsf{p} \to \mathsf{q}; \mathsf{r} \to \mathsf{s}$ is not enforced by any causality relation between the two interactions. More specifically, the processes $\mathsf{p}$, $\mathsf{q}$, $\mathsf{r}$, and $\mathsf{s}$ are all different, hence they operate independently—as the transition for the projection of $C_{\mathsf{problem}}$ shows. However, if the process names of the two interactions overlapped in any way, this problem would not appear. For example, consider the choreography $\mathsf{p} \to \mathsf{q}; \mathsf{r} \to \mathsf{q}; \mathbf{0}$. Its projection executes as expected by the choreography because process $\mathsf{q}$ necessarily needs to complete the first interaction before participating in the second.

**Exercise 20** ($\hookrightarrow$)**.** *Check that the choreography* $\mathsf{p} \to \mathsf{q}; \mathsf{r} \to \mathsf{q}; \mathbf{0}$ *can mimic the transitions of its EPP.*

$$C ::= \mathsf{p} \to \mathsf{q}; C \mid \mathbf{0}$$

Figure 3.4: Simple concurrent choreographies, syntax.

Detecting sequences of interactions that do not have causal dependencies can be done mechanically. Thus, it is possible to automatically detect whether a choreography respects the condition of not having sequences of independent interactions, as in $C_{\mathsf{problem}}$. Forbidding programmers to write choreographies like $C_{\mathsf{problem}}$ was a popular approach (and still is in some works, when useful) in the first studies on choreographies, like [Fu et al. 2005, Qiu et al. 2007, Carbone et al. 2007].

**Out of order execution** The other way to solve our issue with sequences of independent interactions is to extend the semantics of choreographies to correctly capture the parallel nature of processes. Going back to $C_{\mathsf{problem}}$ again, if we could somehow design a semantics that allowed for the the transition

$$\mathsf{p} \to \mathsf{q}; \mathsf{r} \to \mathsf{s}; \mathbf{0} \quad \xrightarrow{\mathsf{r}\to\mathsf{s}} \quad \mathsf{p} \to \mathsf{q}; \mathbf{0}$$

then we would be fine, because that transition would match the problematic one that the EPP of the choreography can do. Observe that this transition does not respect the order in which instructions are given in the choreography. This kind of semantics is typically called "out-of-order execution". The idea is widespread in many domains. For example, modern CPUs and/or language runtimes may change the order in which instructions are executed to increase performance, when it is safe to do so—a typical example is the single-threaded imperative code `x++; y++;`, where the order in which the two increments are done is ininfluential and the runtime may thus decide to parallelise it.

Since the inception of out-of-order execution for choreographies [Carbone and Montesi 2013], similar ideas have been adopted in different works [Deniélou and Yoshida 2013, Honda et al. 2016, Cruz-Filipe and Montesi 2019].

In this book, we follow the out of order approach.

### 3.2.3 Simple Concurrent Choreographies

We update our model of simple choreographies to capture concurrent execution of different processes.

The syntax of choreographies remains unchanged. It is displayed in fig. 3.4.

$$\frac{}{\mathsf{p} \to \mathsf{q}; C \xrightarrow{\mathsf{p}\to\mathsf{q}} C}\ \textsc{Com} \qquad \frac{C \xrightarrow{\mu} C' \quad \{\mathsf{p},\mathsf{q}\} \# \mathsf{pn}(\mu)}{\mathsf{p} \to \mathsf{q}; C \xrightarrow{\mu} \mathsf{p} \to \mathsf{q}; C'}\ \textsc{Delay}$$

Figure 3.5: Simple concurrent choreographies, semantics.

The semantics of choreographies (fig. 3.5), instead, has an extra rule:

$$\frac{C \xrightarrow{\mu} C' \quad \{\mathsf{p},\mathsf{q}\} \# \mathsf{pn}(\mu)}{\mathsf{p} \to \mathsf{q}; C \xrightarrow{\mu} \mathsf{p} \to \mathsf{q}; C'}\ \textsc{Delay} \ .$$

Rule DELAY allows us to *delay* the execution of a communication prescribed by the choreography, and let us observe a transition that comes from its continuation instead, *assuming that such transition does not involve any of the processes in the communication term*. This condition is formalised by the premise $\{\mathsf{p},\mathsf{q}\} \# \mathsf{pn}(\mu)$, where function $\mathsf{pn}$ is a function from labels to sets of process names defined as follows:

$$\mathsf{pn}\,(\mathsf{p} \to \mathsf{q}) \ = \ \{\mathsf{p},\mathsf{q}\}\ .$$

**Example 12.** *This is the transition we needed for the choreography in section 3.2.2.*

$$\mathsf{p} \to \mathsf{q}; \mathsf{r} \to \mathsf{s}; \mathbf{0} \quad \xrightarrow{\mathsf{r}\to\mathsf{s}} \quad \mathsf{p} \to \mathsf{q}; \mathbf{0}\ .$$

*We can now perform it with our new semantics. Here is the derivation:*

$$\frac{\dfrac{}{\mathsf{r} \to \mathsf{s}; \mathbf{0} \xrightarrow{\mathsf{r}\to\mathsf{s}} \mathbf{0}}\ \textsc{Com} \qquad \{\mathsf{p},\mathsf{q}\} \#\{\mathsf{r},\mathsf{s}\}}{\mathsf{p} \to \mathsf{q}; \mathsf{r} \to \mathsf{s}; \mathbf{0} \xrightarrow{\mathsf{r}\to\mathsf{s}} \mathsf{p} \to \mathsf{q}; \mathbf{0}}\ \textsc{Delay} \ .$$

**Exercise 21.** *Show all the possible multi-step derivatives of the following choreography.*

$$\mathsf{p} \to \mathsf{q}; \mathsf{r} \to \mathsf{s}; \mathsf{q} \to \mathsf{r}; \mathbf{0}$$

**A note on premises: side conditions** The reader might be puzzled by the fact that the second premise of rule DELAY, $\{\mathsf{p},\mathsf{q}\} \# \mathsf{pn}(\mu)$, has a form for which we have not specified any inference rules. Indeed, in example 12, we have not derived the required premise $\{\mathsf{p},\mathsf{q}\} \#\{\mathsf{r},\mathsf{s}\}$, but simply stated it.

This kind of premises are sometimes called *side conditions*, since they merely express constraints on the schematic variables (in this case $\mathsf{p}$, $\mathsf{q}$, and $\mu$) of a rule. The rule can be applied only if these constraints are respected. If one goes down this road, then side conditions are not exactly premises "morally", since they do not require a derivation.

Another way to look at it is that we certainly *could* define an inference system for deriving basic facts such as $\{\mathsf{p},\mathsf{q}\}\#\{\mathsf{r},\mathsf{s}\}$. It is just that these facts are so elementary that analysing them to such a level of detail is uninteresting for our purposes here (but definitely not in general, and in fact inference systems for this kind of statements do exist).

We take the second viewpoint here. That is, whenever we specify a rule with some premise for which we have not specified an inference system, we are going to assume that such an inference system exists. In our examples, we are simply going to state these premises and assume that there exists a derivation that proves it. We shall adopt this "shortcut" only for basic propositions about sets.

## 3.3 Correctness of EPP

We are now in possession of all the necessary ingredients to prove that EPP is correct in general, meaning that it yields the right result for all simple concurrent choreographies. Formally, we are going to prove two results.

- *(Completeness) For any $C$, $\mu$, and $C'$, $C \xrightarrow{\mu} C'$ implies $[\![C]\!] \xrightarrow{\mu} [\![C']\!]$.*
- *(Soundness) For any $C$, $\mu$, and $N$, $[\![C]\!] \xrightarrow{\mu} N$ implies $C \xrightarrow{\mu} C'$ for some $C'$ such that $N = [\![C']\!]$.*

Intuitively, the completeness part means that the network generated by the EPP of a choreography does *all* that the choreography says. Conversely, the soundness part means that the network generated by the EPP of a choreography does *only* what the choreography says. The two parts combined give us correctness: the network generated by EPP does exactly what is defined in the originating choreography. So what happens is what we want!

A useful property for our proof will be that process projection remains invariant under orthogonal transitions.

**Lemma 3.** *If $C \xrightarrow{\mu} C'$, then $[\![C]\!]_{\mathsf{r}} = [\![C']\!]_{\mathsf{r}}$ for all $\mathsf{r} \notin \mathsf{pn}(\mu)$.*

**Exercise 22 ($\hookrightarrow$).** *Prove lemma 3.*

Hint: *Proceed by induction on the derivation of the transition $C \xrightarrow{\mu} C'$.*

**Notation for the proof** Before proceeding to the proof of the theorem, it is convenient to introduce some additional notation on networks.

First, let $C$ be a choreography and $S = \{\mathsf{p}_1, \ldots, \mathsf{p}_n\}$ be a finite set of process names. The *restriction* of a network $N$ to $S$, written $N\restriction_S$, is defined as:

$$N\restriction_S \;=\; \mathsf{p}_1[N(\mathsf{p}_1)] \mid \cdots \mid \mathsf{p}_n[N(\mathsf{p}_n)] \,.$$

We shall write $N\restriction_{\vec{\mathsf{p}}}$ as abbreviation for $N\restriction_{\{\vec{\mathsf{p}}\}}$ (since the ordering of the sequence $\vec{\mathsf{p}}$ does not matter).

We define also a notation to "erase" processes in a network. For any network $N$ and process name $\mathsf{p}$, we write $N \setminus \mathsf{p}$ for the network such that:

$$(N \setminus \mathsf{p})(\mathsf{q}) \;=\; \begin{cases} \mathbf{0} & \text{if } \mathsf{q} = \mathsf{p} \\ N(\mathsf{q}) & \text{otherwise} \end{cases} .$$

The notation is generalised to sequences of process names in the expected way, that is, for any non-empty sequence $\vec{\mathsf{q}}$:

$$N \setminus \mathsf{p}, \vec{\mathsf{q}} \;=\; (N \setminus \mathsf{p}) \setminus \vec{\mathsf{q}} .$$

We give $\setminus$ a higher priority than $\mid$, in the sense that

$$N \setminus \vec{\mathsf{p}} \mid M \;=\; (N \setminus \vec{\mathsf{p}}) \mid M$$

for all $N$, $M$, and $\vec{\mathsf{p}}$.

The notation supports the following property.

**Proposition 3.** *For any $N$ and $\vec{\mathsf{p}}$,*

$$N \;=\; (N \setminus \vec{\mathsf{p}}) \;\mid\; N\restriction_{\vec{\mathsf{p}}}$$

Proposition 3 is going to be useful for focusing on some processes of interest. We are going to use it often in the remainder, sometimes implicitly.

For the upcoming proof specifically, by the definition of EPP, we can reformulate proposition 3 to obtain a useful property about choreography projections.

**Proposition 4.** *For any $C$ and $\vec{\mathsf{p}} = \mathsf{p}_1, \ldots, \mathsf{p}_n$,*

$$[\![C]\!] \;=\; ([\![C]\!] \setminus \vec{\mathsf{p}}) \;\mid\; \mathsf{p}_1\Big[[\![C]\!]_{\mathsf{p}_1}\Big] \mid \cdots \mid \mathsf{p}_n\Big[[\![C]\!]_{\mathsf{p}_n}\Big] .$$

**Exercise 23.** *Prove that $N \setminus \mathsf{supp}(N) = \mathbf{0}$.*

Our notation also supports an important semantic property: in network transitions, only the processes mentioned in a label are required.

**Lemma 4.** *If $N \xrightarrow{\mu} N'$ and $\mathsf{p} \notin \mathsf{pn}(\mu)$, then $N \setminus \mathsf{p} \xrightarrow{\mu} N' \setminus \mathsf{p}$.*

**Exercise 24.** *Prove lemma 4.*

By iterating applications of lemma 4, one can additionally prove the following result on the minimal network required to perform a given transition.

**Corollary 2.** *If $N \xrightarrow{\mu} N'$, then $N\restriction_{\mathsf{pn}(\mu)} \xrightarrow{\mu} N'\restriction_{\mathsf{pn}(\mu)}$.*

We can now prove the results about EPP. We prove each one as a separate lemma, and combine these results later in a single theorem.

**The proof** We prove completeness first.

**Lemma 5** (Completeness of EPP)**.** *For any $C$, $\mu$, and $C'$,*

$$C \xrightarrow{\mu} C' \textit{ implies } [\![C]\!] \xrightarrow{\mu} [\![C']\!] \ .$$

*Proof.* We proceed by induction on the derivation of the transition $C \xrightarrow{\mu} C'$.

**Case** The derivation of the transition ends with an application of rule COM:

$$\frac{}{\mathsf{p} \to \mathsf{q}; C' \xrightarrow{\mathsf{p}\to\mathsf{q}} C'} \ \textsc{Com} \ .$$

By the definition of EPP,

$$[\![C]\!] \ = \ [\![C]\!] \setminus \mathsf{p},\mathsf{q} \mid \mathsf{p}\left[\mathsf{q}!; [\![C']\!]_{\mathsf{p}}\right] \mid \mathsf{q}\left[\mathsf{p}?; [\![C']\!]_{\mathsf{q}}\right] .$$

By lemma 3, $[\![C]\!] \setminus \mathsf{p},\mathsf{q} = [\![C']\!] \setminus \mathsf{p},\mathsf{q}$.

Therefore, we can rewrite $[\![C]\!]$ from above as:

$$[\![C]\!] \ = \ [\![C']\!] \setminus \mathsf{p},\mathsf{q} \mid \mathsf{p}\left[\mathsf{q}!; [\![C']\!]_{\mathsf{p}}\right] \mid \mathsf{q}\left[\mathsf{p}?; [\![C']\!]_{\mathsf{q}}\right] .$$

By rules PAR and COM for networks, $[\![C]\!]$ can mimic the choreographic transition:

$$[\![C]\!] \xrightarrow{\mathsf{p}\to\mathsf{q}} [\![C']\!] \setminus \mathsf{p},\mathsf{q} \mid \mathsf{p}\left[[\![C']\!]_{\mathsf{p}}\right] \mid \mathsf{q}\left[[\![C']\!]_{\mathsf{q}}\right] .$$

Notice that the derivative of $[\![C]\!]$ above is exactly the EPP of $C'$, hence $[\![C]\!] \xrightarrow{\mathsf{p}\to\mathsf{q}} [\![C']\!]$.

**Case** The derivation of the transition ends with an application of rule DELAY:

$$\frac{C_1 \xrightarrow{\mu} C_2 \quad \{\mathsf{p},\mathsf{q}\} \# \mathsf{pn}(\mu)}{\mathsf{p} \to \mathsf{q}; C_1 \xrightarrow{\mu} \mathsf{p} \to \mathsf{q}; C_2} \ \textsc{Delay} \ .$$

By definition of EPP,

$$[\![C]\!] \ = \ [\![C_1]\!] \setminus \mathsf{p},\mathsf{q} \mid \mathsf{p}\left[\mathsf{q}!; [\![C_1]\!]_{\mathsf{p}}\right] \mid \mathsf{q}\left[\mathsf{p}?; [\![C_1]\!]_{\mathsf{q}}\right] .$$

By induction hypothesis, $[\![C_1]\!] \xrightarrow{\mu} [\![C_2]\!]$. Since $\mathsf{p}$ and $\mathsf{q}$ do not appear in $\mu$ (premise of DELAY above), by lemma 4 we obtain $[\![C_1]\!] \setminus \mathsf{p},\mathsf{q} \xrightarrow{\mu} [\![C_2]\!] \setminus \mathsf{p},\mathsf{q}$. We use this to make the following derivation:

$$\frac{[\![C_1]\!] \setminus \mathsf{p},\mathsf{q} \xrightarrow{\mu} [\![C_2]\!] \setminus \mathsf{p},\mathsf{q}}{[\![C_1]\!] \setminus \mathsf{p},\mathsf{q} \mid \mathsf{p}\left[\mathsf{q}!; [\![C_1]\!]_{\mathsf{p}}\right] \mid \mathsf{q}\left[\mathsf{p}?; [\![C_1]\!]_{\mathsf{q}}\right] \xrightarrow{\mu} [\![C_2]\!] \setminus \mathsf{p},\mathsf{q} \mid \mathsf{p}\left[\mathsf{q}!; [\![C_1]\!]_{\mathsf{p}}\right] \mid \mathsf{q}\left[\mathsf{p}?; [\![C_1]\!]_{\mathsf{q}}\right]} \ \textsc{Par} \ . \tag{3.2}$$

From the premises of rule DELAY at the beginning and lemma 3, we obtain that $[\![C_1]\!]_{\mathsf{p}} = [\![C_2]\!]_{\mathsf{p}}$ and $[\![C_1]\!]_{\mathsf{q}} = [\![C_2]\!]_{\mathsf{q}}$. Therefore the derivative we obtained with rule PAR is equal to $[\![\mathsf{p} \to \mathsf{q}; C_2]\!]$, and the thesis is proven. We summarise the proof of this case:

$$\begin{array}{lcl} [\![C]\!] & = & \text{(definition 8)} \\ [\![C_1]\!] \setminus \mathsf{p},\mathsf{q} \mid \mathsf{p}\left[\mathsf{q}!; [\![C_1]\!]_{\mathsf{p}}\right] \mid \mathsf{q}\left[\mathsf{p}?; [\![C_1]\!]_{\mathsf{q}}\right] & \xrightarrow{\mu} & \text{(eq. (3.2))} \\ [\![C_2]\!] \setminus \mathsf{p},\mathsf{q} \mid \mathsf{p}\left[\mathsf{q}!; [\![C_1]\!]_{\mathsf{p}}\right] \mid \mathsf{q}\left[\mathsf{p}?; [\![C_1]\!]_{\mathsf{q}}\right] & = & \text{(lemma 3)} \\ [\![C_2]\!] \setminus \mathsf{p},\mathsf{q} \mid \mathsf{p}\left[\mathsf{q}!; [\![C_2]\!]_{\mathsf{p}}\right] \mid \mathsf{q}\left[\mathsf{p}?; [\![C_2]\!]_{\mathsf{q}}\right] & = & \text{(definition 8)} \\ [\![\mathsf{p} \to \mathsf{q}; C_2]\!] \ . & & \end{array}$$

□

We now move on to proving the soundness result.

**Lemma 6** (Soundness of EPP)**.** *For any $C$, $\mu$, and $N$,*

$$[\![C]\!] \xrightarrow{\mu} N \textit{ implies } C \xrightarrow{\mu} C' \textit{ for some } C' \textit{ such that } N = [\![C']\!] \ .$$

*Proof.* We proceed by induction on $C$ and consider all the possible transitions that $N$ can have.

**Case** $C = \mathbf{0}$. In this case, $[\![C]\!] = \mathbf{0}$ so there are no transitions to consider.

**Case** $C = \mathsf{p} \to \mathsf{q}; C_1$ for some $C_1$. By definition of EPP,

$$[\![C]\!] \ = \ [\![C_1]\!] \setminus \mathsf{p},\mathsf{q} \mid \mathsf{p}\left[\mathsf{q}!; [\![C_1]\!]_{\mathsf{p}}\right] \mid \mathsf{q}\left[\mathsf{p}?; [\![C_1]\!]_{\mathsf{q}}\right] \ .$$

Now, if $[\![C]\!] \xrightarrow{\mu} N$, we have two cases: either $\mu = \mathsf{p} \to \mathsf{q}$ (the transition is the execution of the communication between $\mathsf{p}$ and $\mathsf{q}$), or $\mathsf{p}$ and $\mathsf{q}$ do not appear in $\mu$ ($\{\mathsf{p},\mathsf{q}\} \# \mathsf{pn}(\mu)$).

In the first case, by the semantics of networks and lemma 1 we obtain

$$N \ = \ [\![C_1]\!] \setminus \mathsf{p},\mathsf{q} \mid \mathsf{p}\left[[\![C_1]\!]_{\mathsf{p}}\right] \mid \mathsf{q}\left[[\![C_1]\!]_{\mathsf{q}}\right] \ = \ [\![C_1]\!] \ .$$

Therefore, $C$ can mimic the transition by $[\![C]\!]$ by rule COM:

$$\mathsf{p} \to \mathsf{q}; C_1 \xrightarrow{\mathsf{p}\to\mathsf{q}} C_1 \ .$$

In the second case, by lemma 4 we have that

$$[\![C_1]\!] \setminus \mathsf{p},\mathsf{q} \xrightarrow{\mu} N \setminus \mathsf{p},\mathsf{q} \ .$$

From this, we can derive the following transition by using rule PAR:

$$[\![C_1]\!] \xrightarrow{\mu} N \setminus \mathsf{p},\mathsf{q} \mid \mathsf{p}\Big[[\![C_1]\!]_{\mathsf{p}}\Big] \mid \mathsf{q}\Big[[\![C_1]\!]_{\mathsf{q}}\Big] \ .$$

By induction hypothesis, we obtain that there exists $C_2$ such that

$$C_1 \xrightarrow{\mu} C_2 \text{ such that } [\![C_2]\!] = N \setminus \mathsf{p},\mathsf{q} \mid \mathsf{p}\Big[[\![C_1]\!]_{\mathsf{p}}\Big] \mid \mathsf{q}\Big[[\![C_1]\!]_{\mathsf{q}}\Big] \ . \tag{3.3}$$

Equation (3.3) reveals that $N \setminus \mathsf{p},\mathsf{q} = [\![C_2]\!] \setminus \mathsf{p},\mathsf{q}$. Also, since $[\![C]\!] \xrightarrow{\mu} N$, lemma 3 tells us that $N\restriction_{\mathsf{p},\mathsf{q}} = [\![C]\!]\restriction_{\mathsf{p},\mathsf{q}}$. Therefore, given that $C = \mathsf{p}\ \texttt{->}\ \mathsf{q}; C_1$,

$$N = [\![C_2]\!] \setminus \mathsf{p},\mathsf{q} \mid [\![\mathsf{p}\ \texttt{->}\ \mathsf{q}; C_1]\!]\restriction_{\mathsf{p},\mathsf{q}} \ . \tag{3.4}$$

We can now move to mimicking the transition on the choreography level. Since we are in the case where $\mathsf{p}$ and $\mathsf{q}$ do not appear in $\mu$, we can use the transition $C_1 \xrightarrow{\mu} C_2$ from eq. (3.3) to apply rule DELAY as follows.

$$\frac{C_1 \xrightarrow{\mu} C_2 \quad \mathsf{pn}(\mu) \# \{\mathsf{p},\mathsf{q}\}}{\mathsf{p}\ \texttt{->}\ \mathsf{q}; C_1 \xrightarrow{\mu} \mathsf{p}\ \texttt{->}\ \mathsf{q}; C_2}\ \textsc{Delay}$$

We are left with having to prove that $[\![\mathsf{p}\ \texttt{->}\ \mathsf{q}; C_2]\!] = N$. According to eq. (3.4), this is true if $[\![\mathsf{p}\ \texttt{->}\ \mathsf{q}; C_1]\!]\restriction_{\mathsf{p},\mathsf{q}} = [\![\mathsf{p}\ \texttt{->}\ \mathsf{q}; C_2]\!]\restriction_{\mathsf{p},\mathsf{q}}$, which follows from the above derivation and lemma 3.

□

By combining lemmas 5 and 6, we obtain the following theorem.

**Theorem 2** (Correctness of EPP for simple concurrent choreographies)**.**

- *(Completeness) For any $C$, $\mu$, and $C'$, $C \xrightarrow{\mu} C'$ implies $[\![C]\!] \xrightarrow{\mu} [\![C']\!]$.*
- *(Soundness) For any $C$, $\mu$, and $N$, $[\![C]\!] \xrightarrow{\mu} N$ implies $C \xrightarrow{\mu} C'$ for some $C'$ such that $N = [\![C']\!]$.*

The correctness of EPP gives us a powerful result for free: the EPP of a choreography never gets stuck and can always terminate. Intuitively, this works because interactions in choreographies are written atomically, in the sense that they specify both the send and receive actions needed for the communication in a single term. EPP then projects the send and receive actions correctly into the process terms of sender and receiver, so that they are always well-matched. This prevents mismatched communications, which are possible to write in simple processes as we have seen in exercise 15.

We now combine proposition 1 with the completeness part of theorem 2, which respectively say that "a choreography can always reduce until it terminates" and "the EPP of a choreography can always do what the choreography does". This means that "the EPP of a choreography can always reduce until it terminates", as formalised below.

**Theorem 3** (Strong termination for EPP)**.** *For any $C$, if $N$ is a multi-step derivative of $[\![C]\!]$, then there exists $\vec{\mu}$ (possibly empty) such that $N \xrightarrow{\vec{\mu}} \mathbf{0}$.*

*Proof (sketch).* If $C = \mathbf{0}$, then $[\![C]\!] = \mathbf{0}$. Since $\mathbf{0}$ has no derivatives we necessarily have that $N = \mathbf{0}$ and the thesis follows.

Otherwise ($C \neq \mathbf{0}$), by theorem 2 we know that there exists $C'$ that is a multi-step derivative of $C$ such that $N = [\![C']\!]$. By proposition 1, $C'$ can make a sequence of transitions $\vec{\mu}$ such that $C' \xrightarrow{\vec{\mu}} \mathbf{0}$. By the completeness part of theorem 2, $N$ can match these transitions and reach the EPP of $\mathbf{0}$, which is $\mathbf{0}$. □

# Chapter 4

# Local computation

Now that we know the basic principles of choreographies and EPP, we can begin to extend our model to capture more interesting—and realistic—examples.

In this chapter, we shall equip our choreographic and process models with the capability of storing values and performing local computation at processes. This will enable us to capture our initial scenario from example 2 more precisely, i.e., we want to define the *content* of the messages exchanged by Buyer and Seller.

## 4.1 Stores, expressions, and evaluation

**Stores** Let $Var$ be an infinite set of *variable names*, or variables, ranged over by $x$, $y$, $z$. Also, let $Val$ be an infinite set of *values*, ranged over by $v$, $u$. While variables are just names (identifiers), we assume that $Val$ contains all data that we might be interesting in modelling, including integers, strings, and lists.

A *process store* $\sigma$ maps variables to their respective values. Formally, a process store $\sigma$ is a function from variables to values:

$$\sigma : Var \longrightarrow Val \, .$$

We write $PStore$ for the set of all process stores. Also, we shall write $\sigma\left[x \mapsto v\right]$ for the update of store $\sigma$ with the new mapping $x \mapsto v$:

$$\sigma\left[y \mapsto v\right](x) \;=\; \begin{cases} v & \text{if } x = y \\ \sigma(x) & \text{otherwise} \end{cases} \, .$$

A *choreographic store* $\Sigma$ maps process names to their respective process stores. That is, a choreographic store $\Sigma$ is a function from process names to process stores:

$$\Sigma : Pid \longrightarrow PStore \, .$$

$$e ::= v \mid x \mid f(\vec{e})$$

Figure 4.1: Expressions.

One can think of a process store as the memory of a process, and of a choreographic store as the global view over the memory of an entire system of processes.

We shall write $\Sigma\,[\mathsf{p}.x \mapsto v]$ for the update of store $\Sigma$ such that $\mathsf{p}$'s local variable $x$ is mapped to $v$:

$$\Sigma\,[\mathsf{q}.x \mapsto v]\,(\mathsf{p}) \;=\; \begin{cases} \Sigma(\mathsf{p})\,[x \mapsto v] & \text{if } \mathsf{p} = \mathsf{q} \\ \Sigma(\mathsf{p}) & \text{otherwise} \end{cases} .$$

**Expressions** Processes compute new values by means of *expressions*, ranged over by $e$. The syntax of expressions is displayed in fig. 4.1.

There are three forms that an expression can take:

- some constant value $v$ (for any possible $v$);
- a variable $x$;
- a function call $f(\vec{e})$ (also called function invocation), where $f$ is a *function identifier*.

In a function invocation $f(\vec{e})$, $\vec{e}$ is an abbreviation for a sequence of expressions $e_1, \ldots, e_n$, which we call the *arguments* of $f$. The identifier $f$ is a reference to a known *local function* over values, i.e., a function that a process can execute locally without necessitating to communicate with other processes.

We will abuse expression notation by adopting some syntactic sugar in the remainder, whenever the intuition is obvious. For example, $1 + 2$ (the sum of the natural numbers 1 and 2) is an abbreviation the function invocation $plus(1, 2)$, where $plus$ is the function such that $plus(x, y) = x + y$.

**Evaluation** Evaluation models the computation of an expression, which returns a value. We write the proposition $\sigma \vdash e \downarrow v$ for "$e$ is evaluated as $v$ under the process store $\sigma$". The idea is that we might need to read the process store in order to compute an expression, since expressions can contain variables whose values is defined in the store.

The inference system for evaluation is displayed in fig. 4.2.

Rule VAL says that a constant value $v$ is always evaluated to itself:

$$\frac{}{\sigma \vdash v \downarrow v}\ \textsc{Val}\ .$$

$$\frac{}{\sigma \vdash v \downarrow v}\ \textsc{Val} \qquad \frac{}{\sigma \vdash x \downarrow \sigma(x)}\ \textsc{Var}$$

$$\frac{\sigma \vdash e_1 \downarrow v_1 \cdots \sigma \vdash e_n \downarrow v_n \quad \vdash f(\vec{v}) \downarrow v}{\sigma \vdash f(\vec{e}) \downarrow v}\ \textsc{Call}$$

Figure 4.2: Expression evaluation.

Rule VAR evaluates a variable to the value that it is mapped to in the process store:

$$\frac{}{\sigma \vdash x \downarrow \sigma(x)}\ \textsc{Var}\ .$$

Finally, rule CALL evaluates a function call:

$$\frac{\sigma \vdash e_1 \downarrow v_1 \cdots \sigma \vdash e_n \downarrow v_n \quad \vdash f(\vec{v}) \downarrow v}{\sigma \vdash f(\vec{e}) \downarrow v}\ \textsc{Call}\ .$$

In the rule, $\sigma \vdash e_1 \downarrow v_1 \cdots \sigma \vdash e_n \downarrow v_n$ means that there is a premise for each argument $e_i$ in $\vec{e}$. This means that the number of premises of this rule depends on the number of arguments in the function call. These premises evaluate all the expressions that we are passing as arguments to the function. Then, the proposition in the right premise, $\vdash f(\vec{v}) \downarrow v$, reads "invoking function $f$ with arguments $\vec{v}$ evaluates to the value $v$". Value $v$ is thus also the value that we assign to the entire evaluation in the conclusion.

We do not specify how propositions of the kind $\vdash f(\vec{v}) \downarrow v$ can be derived, since this would depend on the specifics of local functions, which we chose to abstract from. We just assume that such a system exists, and that for any $f$ and $\vec{v}$, it is always possible to find some $v$ such that $\vdash f(\vec{v}) \downarrow v$ in *finite time*.

**Remark 2.** *Our rule* CALL *evaluates the expressions passed as arguments* before *evaluating the function invocation. This is sometimes called an* early *semantics, since all arguments are evaluated regardless of whether the function that we are invoking will actually use them. Early evaluation is what many mainstream programming languages use, like Java and C.*

*Some programming languages, like Haskell, use* lazy *evaluation instead, whereby expressions are passed as they are in function invocations, and are eventually evaluated only if and when the function body actually needs them. This means that the set of values $Val$ includes also the set of all expressions.*

**Remark 3.** *In many programming languages, the evaluation of an expression might not terminate, e.g., due to an infinite loop. This is not a problem for our theory: in practice, we could enforce a timeout to all evaluations, and assume that*

$$C ::= I; C \mid \mathbf{0}$$
$$I ::= \mathsf{p}.e \twoheadrightarrow \mathsf{q}.x \mid \mathsf{p}.x \texttt{ := } e$$

Figure 4.3: Stateful choreographies, syntax.

*a special "failure" value is returned if the timeout is reached before the evaluation terminated.*

*Another practical problem is that evaluating a function invocation might require typing constraints. For example, $plus$ might be invocable only by passing numbers as arguments. What if we pass two oranges instead? Again, we can assume that a special failure value is returned in these cases too. Cruz-Filipe and Montesi [2017] showed that choreographies can be easily combined with static typing systems that avoid these problems.*

## 4.2 Stateful Choreographies

We enhance choreographies with our new notions for local computation. The key idea is that now processes are equipped with their own local memories, which they can manipulate during execution.

### 4.2.1 Syntax

The syntax of stateful choreographies is given in fig. 4.3.

In the syntax, $I$ stands for instruction. A choreography is either a term $I; C$, read "run the instruction $I$ and then do $C$", or the terminated choreography $\mathbf{0}$. An instruction $I$ can be:

- a communication $\mathsf{p}.e \twoheadrightarrow \mathsf{q}.x$, where process $\mathsf{p}$ evaluates expression $e$ locally (according to its local store) and sends the resulting value to process $\mathsf{q}$, which stores the value in its local variable $x$;
- a local assignment $\mathsf{p}.x$ **:=** $e$, where $\mathsf{p}$ evaluates expression $e$ locally and sets its variable $x$ to the resulting value.

We write $\mathsf{pn}(I)$ for the set of process names involved in $I$, defined as follows.

$$\mathsf{pn}\left(\mathsf{p}.e \twoheadrightarrow \mathsf{q}.x\right) = \{\mathsf{p}, \mathsf{q}\}$$
$$\mathsf{pn}\left(\mathsf{p}.e \texttt{ := } x\right) = \{\mathsf{p}\}$$

$$\frac{\Sigma(\mathsf{p}) \vdash e \downarrow v}{\langle \mathsf{p}.x := e; C, \Sigma\rangle \xrightarrow{\tau@\mathsf{p}} \langle C, \Sigma\,[\mathsf{p}.x \mapsto v]\rangle}\ \textsc{Local}$$

$$\frac{\Sigma(\mathsf{p}) \vdash e \downarrow v}{\langle \mathsf{p}.e \twoheadrightarrow \mathsf{q}.x; C, \Sigma\rangle \xrightarrow{\mathsf{p}.v\twoheadrightarrow\mathsf{q}} \langle C, \Sigma\,[\mathsf{q}.x \mapsto v]\rangle}\ \textsc{Com}$$

$$\frac{\langle C, \Sigma\rangle \xrightarrow{\mu} \langle C', \Sigma'\rangle \qquad \mathsf{pn}(I) \# \mathsf{pn}(\mu)}{\langle I; C, \Sigma\rangle \xrightarrow{\mu} \langle I; C', \Sigma'\rangle}\ \textsc{Delay}$$

Figure 4.4: Stateful choreographies, semantics.

**Example 13.** *We can finally give a precise choreography for our buyer-seller example (example 2), including computation and message contents as well. Recall the informal description of the example:*

1. Buyer *sends the title of a book she wishes to buy to* Seller*;*
2. Seller *replies to* Buyer *with the price of the book.*

*A corresponding choreography that defines this behaviour is*

$$\mathsf{Buyer}.\mathit{title} \twoheadrightarrow \mathsf{Seller}.x; \mathsf{Seller}.\mathit{cat}(x) \twoheadrightarrow \mathsf{Buyer}.\mathit{price}; \mathbf{0}$$

*where $\mathit{title}$ is a variable, $\mathit{cat}$ is a function that given a book title returns the price for it (cat stands for catalogue, if you like), and $\mathit{price}$ is a variable.*

### 4.2.2 Semantics

Now that processes may read from and write to their local memories (stores), the semantics of choreographies needs to keep track of the stores of all processes. We achieve this by generalising the labelled transition system for choreographies to pairs that consist of a choreography and a choreographic store.

Formally, the semantics of stateful choreographies is defined by the rules given in fig. 4.4. Observe that transitions now have the form $\langle C, \Sigma\rangle \xrightarrow{\mu} \langle C', \Sigma'\rangle$, read "the choreography $C$ equipped with store $\Sigma$ has a transition with label $\mu$ to the choreography $C'$ with store $\Sigma'$". We call a pair $\langle C, \Sigma\rangle$ a *configuration*.

Rule Local models a local assignment:

$$\frac{\Sigma(\mathsf{p}) \vdash e \downarrow v}{\langle \mathsf{p}.x := e; C, \Sigma\rangle \xrightarrow{\tau@\mathsf{p}} \langle C, \Sigma\,[\mathsf{p}.x \mapsto v]\rangle}\ \textsc{Local}\ .$$

The premise $\Sigma(\mathsf{p}) \vdash e \downarrow v$ evaluates the expression $e$ under the store of the evaluating process $\mathsf{p}$, $\Sigma(\mathsf{p})$. In the derivative, we update the store of process $\mathsf{p}$ with the resulting value $v$. The new label $\tau@\mathsf{p}$ indicates that process $\mathsf{p}$ has performed an *internal action*, i.e., an action that does not require communications. We symbolise internal actions with $\tau$.

We also update rule COM to consider that messages now contain values:

$$\frac{\Sigma(\mathsf{p}) \vdash e \downarrow v}{\langle \mathsf{p}.e \twoheadrightarrow \mathsf{q}.x; C, \Sigma\rangle \xrightarrow{\mathsf{p}.v \twoheadrightarrow \mathsf{q}} \langle C, \Sigma\,[\mathsf{q}.x \mapsto v]\rangle}\ \textsc{Com}\ .$$

The premise evaluates the expression of the sender $\mathsf{p}$ under its store. The resulting value $v$ is then sent to the receiver $\mathsf{q}$, and in the derivative we update the value of the variable $x$ used by the receiver in its store. Note that the transition label now contains the sent value $v$.

Finally, we have to update rule DELAY to consider our new instructions:

$$\frac{\langle C, \Sigma\rangle \xrightarrow{\mu} \langle C', \Sigma'\rangle \qquad \mathsf{pn}(I) \# \mathsf{pn}(\mu)}{\langle I; C, \Sigma\rangle \xrightarrow{\mu} \langle I; C', \Sigma'\rangle}\ \textsc{Delay}\ .$$

In the rule, the continuation $C$ in a choreography $I; C$ is allowed to perform a transition with label $\mu$ if $\mu$ does not involve any of the processes in $I$. We update the definition of $\mathsf{pn}(\mu)$ as follows.

$$\mathsf{pn}\,(\mathsf{p}.v \twoheadrightarrow \mathsf{q}) = \{\mathsf{p}, \mathsf{q}\}$$
$$\mathsf{pn}\,(\tau@\mathsf{p}) = \{\mathsf{p}\}$$

**Exercise 25.** *Let $\Sigma$ be such that $\Sigma(\mathsf{p})(title) =$ "Flowers for Algernon" and $cat$ be a function such that $cat($"Flowers for Algernon"$) = 100$. Show the transitions of the choreography from example 13 under $\Sigma$:*

$$\langle \mathsf{Buyer}.title \twoheadrightarrow \mathsf{Seller}.x; \mathsf{Seller}.cat(x) \twoheadrightarrow \mathsf{Buyer}.price; \mathbf{0}, \Sigma\rangle\ .$$

## 4.3 Stateful Processes

Since we updated our choreography model, we also need to update our process model to describe the implementations of choreographies. We apply a similar extension to what we have seen for choreographies, i.e., we pair networks with stores.

$$P ::= \mathsf{p}!e; P \mid \mathsf{p}?x; P \mid x \texttt{ := } e; P \mid \mathbf{0}$$

Figure 4.5: Stateful processes, syntax.

### 4.3.1 Syntax

The syntax of stateful processes is given in fig. 4.5.

A process term $P$ can be:

- a send action $\mathsf{p}!e; P$, read "send the result of evaluating expression $e$ to $\mathsf{p}$, and then proceed as $P$";
- a receive action $\mathsf{p}?x; P$, read "receive a value from process $\mathsf{p}$, store it in variable $x$, and then proceed as $P$";
- an assignment $x$ **:=** $e$, read "store the value obtained by evaluating expression $e$ in variable $x$".

We write $StatefulProc$ for the set of all stateful processes.

**Definition 9** (Stateful network)**.** *A stateful network $N$ is a (total) function from process names to process terms:*

$$N : Pid \longrightarrow StatefulProc\,.$$

### 4.3.2 Semantics

The semantics of stateful processes is an extension of the semantics of simple processes, which recalls the development of the semantics of stateful choreographies. In particular, we extend networks to *network configurations* $\langle N, \Sigma \rangle$, like we did for stateful choreographies. The rules for transitions are given in fig. 4.6.

### 4.3.3 EndPoint Projection

We have to update our definition of EPP to our new language model, which now includes primitives for local computation. The general principle of distributing code over process terms remains the same: we just need to update the definition of process projection to handle our new primitives.

First, let us gain some intuition. The network implementation of the choreography given in example 13 should look like the following.

$$\begin{array}{l} \mathsf{Buyer}[\mathsf{Seller}!title; \mathsf{Seller}?price; \mathbf{0}] \\ \mid \\ \mathsf{Seller}[\mathsf{Buyer}?x; \mathsf{Buyer}!cat(x); \mathbf{0}] \end{array}$$

$$\frac{\Sigma(\mathsf{p}) \vdash e \downarrow v}{\langle \mathsf{p}[\mathsf{q}!e; P] \mid \mathsf{q}[\mathsf{p}?x; Q], \Sigma\rangle \xrightarrow{\mathsf{p}.v \to \mathsf{q}} \langle \mathsf{p}[P] \mid \mathsf{q}[Q], \Sigma\,[\mathsf{q}.x \mapsto v]\rangle}\ \textsc{Com}$$

$$\frac{\Sigma(\mathsf{p}) \vdash e \downarrow v}{\langle \mathsf{p}[x := e; P], \Sigma\rangle \xrightarrow{\tau@\mathsf{p}} \langle \mathsf{p}[P], \Sigma\,[\mathsf{p}.x \mapsto v]\rangle}\ \textsc{Local}$$

$$\frac{\langle N, \Sigma\rangle \xrightarrow{\mu} \langle N', \Sigma'\rangle}{\langle N \mid M, \Sigma\rangle \xrightarrow{\mu} \langle N' \mid M, \Sigma'\rangle}\ \textsc{Par}$$

Figure 4.6: Stateful processes, semantics.

$$[\![\mathsf{p}.e \to \mathsf{q}.x; C]\!]_{\mathsf{r}} = \begin{cases} \mathsf{q}!e; [\![C]\!]_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{p} \\ \mathsf{p}?x; [\![C]\!]_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{q} \\ [\![C]\!]_{\mathsf{r}} & \text{otherwise} \end{cases}$$

$$[\![\mathsf{p}.x := e; C]\!]_{\mathsf{r}} = \begin{cases} x := e; [\![C]\!]_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{p} \\ [\![C]\!]_{\mathsf{r}} & \text{otherwise} \end{cases}$$

$$[\![\mathbf{0}]\!]_{\mathsf{p}} = \mathbf{0}$$

Figure 4.7: Behaviour projection for stateful choreographies.

We generalise this intuition in our updated definition for process projection, which is displayed in fig. 4.7.

The definition of EPP remains the same (definition 8), modulo the usage of our updated process projection. We report the definition again, for convenience.

**Definition 10** (EndPoint Projection (EPP))**.** *Let $C$ be a choreography. The endpoint projection (EPP) of $C$, written $[\![C]\!]$, is the network such that, for all $\mathsf{p} \in Pid$,*

$$[\![C]\!]\,(\mathsf{p}) = [\![C]\!]_{\mathsf{p}}\ .$$

## 4.4 Correctness of EPP

To state the correctness result of EPP for stateful choreographies we now need to take stores into consideration. We shall establish that, for any store $\Sigma$, the choreographic configuration $\langle C, \Sigma\rangle$ mimicks and is mimicked by the network configuration $\langle [\![C]\!], \Sigma\rangle$.

The proof of this result follows the same structure as that of theorem 2. We will need to invoke similar properties to those we used for the simpler setting in

the previous chapters. Specifically, we have to update the forms of lemma 3 and lemma 4 to deal with stores.

**Lemma 7.** *If* $\langle C, \Sigma\rangle \xrightarrow{\mu} \langle C', \Sigma'\rangle$*, then* $[\![C]\!]_{\mathsf{r}} = [\![C']\!]_{\mathsf{r}}$ *for all* $\mathsf{r} \notin \mathsf{pn}(\mu)$.

**Lemma 8.** *If* $\langle N, \Sigma\rangle \xrightarrow{\mu} \langle N', \Sigma'\rangle$ *and* $\mathsf{p} \notin \mathsf{pn}(\mu)$*, then* $\langle N \setminus \mathsf{p}, \Sigma\rangle \xrightarrow{\mu} \langle N' \setminus \mathsf{p}, \Sigma'\rangle$.

**Exercise 26.** *Prove lemma 7.*

**Exercise 27.** *Prove lemma 8.*

**Theorem 4** (Correctness of EPP for stateful choreographies)**.**

- *(Completeness) For any* $C$*,* $\Sigma$*,* $\mu$*,* $C'$*, and* $\Sigma'$*,* $\langle C, \Sigma\rangle \xrightarrow{\mu} \langle C', \Sigma'\rangle$ *implies* $\langle [\![C]\!], \Sigma\rangle \xrightarrow{\mu} \langle [\![C']\!], \Sigma'\rangle$.
- *(Soundness) For any* $C$*,* $\Sigma$*,* $\mu$*,* $N$*, and* $\Sigma'$*,* $\langle [\![C]\!], \Sigma\rangle \xrightarrow{\mu} \langle N, \Sigma'\rangle$ *implies* $\langle C, \Sigma\rangle \xrightarrow{\mu} C'$ *for some* $C'$ *such that* $N = \langle [\![C']\!], \Sigma'\rangle$.

*Proof (Completeness part).* We proceed by induction on the derivation of the transition $C \xrightarrow{\mu} C'$.

**Case** The derivation of the transition ends with an application of rule Com:

$$\frac{\Sigma(\mathsf{p}) \vdash e \downarrow v}{\langle \mathsf{p}.e \twoheadrightarrow \mathsf{q}.x; C', \Sigma\rangle \xrightarrow{\mathsf{p}.v \to \mathsf{q}} \langle C', \Sigma\,[\mathsf{q}.x \mapsto v]\rangle}\ \textsc{Com}\ .$$

By the definition of EPP,

$$[\![C]\!] \;=\; [\![C]\!] \setminus \mathsf{p}, \mathsf{q} \;\mid \mathsf{p}\Big[\mathsf{q}!e; [\![C']\!]_{\mathsf{p}}\Big] \mid \mathsf{q}\Big[\mathsf{p}?x; [\![C']\!]_{\mathsf{q}}\Big]\ .$$

By lemma 7, $[\![C]\!] \setminus \mathsf{p}, \mathsf{q} = [\![C']\!] \setminus \mathsf{p}, \mathsf{q}$.

Therefore, we can rewrite $[\![C]\!]$ from above as:

$$[\![C]\!] \;=\; [\![C']\!] \setminus \mathsf{p}, \mathsf{q} \;\mid \mathsf{p}\Big[\mathsf{q}!e; [\![C']\!]_{\mathsf{p}}\Big] \mid \mathsf{q}\Big[\mathsf{p}?x; [\![C']\!]_{\mathsf{q}}\Big]\ .$$

By rules Par and Com for networks, $\langle [\![C]\!], \Sigma\rangle$ can mimic the choreographic transition:

$$\langle [\![C]\!], \Sigma\rangle \xrightarrow{\mathsf{p}.v \to \mathsf{q}} \Big\langle [\![C']\!] \setminus \mathsf{p}, \mathsf{q} \;\mid \mathsf{p}\Big[[\![C']\!]_{\mathsf{p}}\Big] \mid \mathsf{q}\Big[[\![C']\!]_{\mathsf{q}}\Big], \Sigma\,[\mathsf{q}.x \mapsto v]\Big\rangle\ .$$

Notice that the choreography in the derivative of $\langle [\![C]\!], \Sigma\rangle$ above is exactly the EPP of $C'$, hence $\langle [\![C]\!], \Sigma\rangle \xrightarrow{\mathsf{p}.v \to \mathsf{q}} \langle [\![C']\!], \Sigma\,[\mathsf{q}.x \mapsto v]\rangle$.

**Case** The derivation of the transition ends with an application of rule DELAY:

$$\frac{\langle C_1, \Sigma\rangle \xrightarrow{\mu} \langle C_2, \Sigma'\rangle \qquad \mathsf{pn}(I) \# \mathsf{pn}(\mu)}{\langle I; C_1, \Sigma\rangle \xrightarrow{\mu} \langle I; C_2, \Sigma'\rangle}\ \textsc{Delay} \ .$$

We have two cases, depending on whether $I$ is a local computation ($I = \mathsf{p}.x \texttt{ := } e$) or a communication ($I = \mathsf{p}.e \texttt{ -> } \mathsf{q}.x$). We deal with the first case, and leave the second to the reader (the reasoning is similar).

By definition of EPP,

$$[\![C]\!] \ = \ [\![C_1]\!] \setminus \mathsf{p} \mid \mathsf{p}\Big[x \texttt{ := } e; [\![C_1]\!]_{\mathsf{p}}\Big] \ .$$

By induction hypothesis,

$$\langle [\![C_1]\!], \Sigma\rangle \xrightarrow{\mu} \langle [\![C_2]\!], \Sigma'\rangle \ .$$

Since $\mathsf{p}$ does not appear in $\mu$ (premise of DELAY above), by lemma 8 we obtain

$$\langle [\![C_1]\!] \setminus \mathsf{p}, \Sigma\rangle \xrightarrow{\mu} \langle [\![C_2]\!] \setminus \mathsf{p}, \Sigma'\rangle \ .$$

We use this to make the following derivation:

$$\frac{\langle [\![C_1]\!] \setminus \mathsf{p}, \Sigma\rangle \xrightarrow{\mu} \langle [\![C_2]\!] \setminus \mathsf{p}, \Sigma'\rangle}{\Big\langle [\![C_1]\!] \setminus \mathsf{p} \mid \mathsf{p}\Big[x \texttt{ := } e; [\![C_1]\!]_{\mathsf{p}}\Big], \Sigma\Big\rangle \xrightarrow{\mu} \Big\langle [\![C_2]\!] \setminus \mathsf{p} \mid \mathsf{p}\Big[x \texttt{ := } e; [\![C_1]\!]_{\mathsf{p}}\Big], \Sigma'\Big\rangle}\ \textsc{Par} \ . \tag{4.1}$$

From the premises of rule DELAY at the beginning and lemma 7, we have that $[\![C_1]\!]_{\mathsf{p}} = [\![C_2]\!]_{\mathsf{p}}$. Therefore the derivative we obtained with rule PAR is equal to $[\![\mathsf{p}.x \texttt{ := } e; C_2]\!]$, and the thesis is proven. We summarise the proof of this case:

$$\begin{array}{lcl}
[\![C]\!] & = & \text{(definition 8)} \\
[\![C_1]\!] \setminus \mathsf{p}, \mathsf{q} \mid \mathsf{p}\Big[\mathsf{q}!; [\![C_1]\!]_{\mathsf{p}}\Big] \mid \mathsf{q}\Big[\mathsf{p}?; [\![C_1]\!]_{\mathsf{q}}\Big] & \xrightarrow{\mu} & \text{(eq. (3.2))} \\
[\![C_2]\!] \setminus \mathsf{p}, \mathsf{q} \mid \mathsf{p}\Big[\mathsf{q}!; [\![C_1]\!]_{\mathsf{p}}\Big] \mid \mathsf{q}\Big[\mathsf{p}?; [\![C_1]\!]_{\mathsf{q}}\Big] & = & \text{(lemma 3)} \\
[\![C_2]\!] \setminus \mathsf{p}, \mathsf{q} \mid \mathsf{p}\Big[\mathsf{q}!; [\![C_2]\!]_{\mathsf{p}}\Big] \mid \mathsf{q}\Big[\mathsf{p}?; [\![C_2]\!]_{\mathsf{q}}\Big] & = & \text{(definition 8)} \\
[\![\mathsf{p}.x \texttt{ := } e; C_2]\!] \ . & &
\end{array}$$

□

**Exercise 28** (!). *Prove the soundness part of theorem 4.*

Hint: *Follow the structure of the proof of lemma 6, paying attention to the difference between the definitions of syntax, semantics, and EPP between chapters 2 and 3 and this chapter.*

# Appendix A

# Solution to selected exercises

We give the solutions to some selected exercises, pointing out the main aspects. Some solutions are given in full detail, to serve as examples of exposition.

*Solution of exercise 3.* We prove only the direction from the system in fig. 1.1 to the system in fig. 1.2.

To prove this direction, we actually prove the stronger statement:

- if $\mathsf{conn}(A, B)$ is derivable in the system in fig. 1.1, then $\mathsf{conn}(A, B)$ is derivable in the system in fig. 1.2;
- if $\mathsf{path}(A, B)$ is derivable in the system in fig. 1.1, then there exists a natural number $n$ such that $\mathsf{path}(A, B, n)$ is derivable in the system in fig. 1.2.

We proceed by induction on the structure of the derivation of $\mathsf{conn}(A, B)$ or $\mathsf{path}(A, B)$ in the system in fig. 1.1. We get a case for each rule that could be applied last in the derivation.

**Base cases (axioms)** If the last applied rule is an axiom, then the proof is valid also in the other system directly, since the two systems share the same axioms.

**Case SYM** The derivation has this shape:

$$\dfrac{\begin{array}{c}\mathcal{D}\\ \mathsf{conn}(B, A)\end{array}}{\mathsf{conn}(A, B)}\ \textsc{Sym}\ .$$

By induction hypothesis on the sub-derivation $\mathcal{D}$, we know that there exists $\mathcal{D}'$ in the other system such that:

$$\begin{array}{c}\mathcal{D}'\\ \mathsf{conn}(B, A)\ .\end{array}$$

The thesis follows by applying rule SYM:

$$\dfrac{\overset{\mathcal{D}'}{\mathsf{conn}(B, A)}}{\mathsf{conn}(A, B)}\ \textsc{Sym}\ .$$

**Case DIR** The derivation has this shape:

$$\dfrac{\overset{\mathcal{D}}{\mathsf{conn}(A, B)}}{\mathsf{path}(A, B)}\ \textsc{Dir}\ .$$

By induction hypothesis, we know that there exists $\mathcal{D}'$ in the other system such that:

$$\overset{\mathcal{D}'}{\mathsf{conn}(A, B)}\ .$$

The thesis follows by applying rule DIRW.

$$\dfrac{\overset{\mathcal{D}'}{\mathsf{conn}(A, B)}}{\mathsf{path}(A, B, 1)}\ \textsc{DirW}\ .$$

**Case TRANS** The derivation has this shape:

$$\dfrac{\overset{\mathcal{D}}{\mathsf{path}(A, B)} \qquad \overset{\mathcal{E}}{\mathsf{path}(B, C)}}{\mathsf{path}(A, C)}\ \textsc{Trans}\ .$$

By induction hypothesis on $\mathcal{D}$ and by induction hypothesis on $\mathcal{E}$, we know that there exist natural numbers $n$ and $m$, and derivations $\mathcal{D}'$ and $\mathcal{E}'$ in the other system such that:

$$\overset{\mathcal{D}'}{\mathsf{path}(A, B, n)} \qquad \overset{\mathcal{E}'}{\mathsf{path}(A, B, m)}\ .$$

The thesis follows by applying rule TRANSW.

$$\dfrac{\overset{\mathcal{D}'}{\mathsf{path}(A, B, n)} \qquad \overset{\mathcal{E}'}{\mathsf{path}(A, B, m)}}{\mathsf{path}(A, B, n + m)}\ \textsc{TransW}\ .$$

■

*Solution of exercise 10.* We prove the stronger result that, for any choreography $C$, either:

1. $C = \mathbf{0}$; or,
2. $C \neq \mathbf{0}$, in which case there exists $\vec{\mu}$ such that $C \xrightarrow{\vec{\mu}} \mathbf{0}$.

We proceed by induction on the structure of $C$.

**Case** $C = \mathbf{0}$. In this case, the thesis follows trivially.

**Case** $C = \mathsf{p} \to \mathsf{q}; C'$ for some $C'$. By rule COM, we obtain $C \xrightarrow{\mathsf{p}\to\mathsf{q}} C'$.

If $C' = \mathbf{0}$, then the thesis follows by $\vec{\mu} = \mathsf{p} \to \mathsf{q}$. Otherwise, if $C' \neq \mathbf{0}$, by induction hypothesis there exists $\vec{\mu}'$ such that $C' \xrightarrow{\vec{\mu}'} \mathbf{0}$. Hence, the thesis follows for the sequence of labels $\vec{\mu} = \mathsf{p} \to \mathsf{q}, \vec{\mu}'$.

■

*Solution of exercise 16.* By induction on the derivation of the transition $N \xrightarrow{\mu} N'$.

**Case** The derivation of the transition ends with an application of rule COM:

$$\frac{}{\mathsf{p}[\mathsf{q}!; P] \mid \mathsf{q}[\mathsf{p}?; Q] \xrightarrow{\mathsf{p}\to\mathsf{q}} \mathsf{p}[P] \mid \mathsf{q}[Q]}\ \textsc{Com},$$

for $N = \mathsf{p}[\mathsf{q}!; P] \mid \mathsf{q}[\mathsf{p}?; Q]$.

Trivially, in this case $N(\mathsf{r}) = \mathbf{0} = N'(\mathsf{r})$ for all $\mathsf{r}$ different than $\mathsf{p}$ and $\mathsf{q}$.

**Case** The derivation of the transition ends with an application of rule PAR:

$$\frac{N_1 \xrightarrow{\mu} N_2}{N_1 \mid M \xrightarrow{\mu} N_2 \mid M}\ \textsc{Par},$$

for $N = N_1 \mid M$.

By induction hypothesis, $N_1(\mathsf{r}) = N_2(\mathsf{r})$ for all $\mathsf{r}$ that do not appear in $\mu$. The thesis follows because parallel composition requires disjoint supports and the context $M$ does not change in the derivative.

■

*Solution of exercise 22.* By induction on the derivation of the transition $C \xrightarrow{\mu} C'$.

**Case** The derivation of the transition ends with an application of rule COM:

$$\frac{}{\mathsf{p} \to \mathsf{q}; C' \xrightarrow{\mathsf{p}\to\mathsf{q}} C'}\ \textsc{Com}.$$

The thesis follows by definition 7, since for all $\mathsf{r} \notin \{\mathsf{p}, \mathsf{q}\}$, $[\![\mathsf{p} \to \mathsf{q}; C']\!]_{\mathsf{r}} = [\![C']\!]_{\mathsf{r}}$.

**Case** The derivation of the transition ends with an application of rule DELAY:

$$\frac{C_1 \xrightarrow{\mu} C_2 \quad \{\mathsf{p},\mathsf{q}\} \# \mathsf{pn}(\mu)}{\mathsf{p} \text{ -> } \mathsf{q}; C_1 \xrightarrow{\mu} \mathsf{p} \text{ -> } \mathsf{q}; C_2}\ \textsc{Delay} \ .$$

By induction hypothesis, $[\![C_1]\!]_{\mathsf{r}} = [\![C_2]\!]_{\mathsf{r}}$. We have three cases, depending on whether $\mathsf{r}$ is $\mathsf{p}$, $\mathsf{q}$, or another process name.

For $\mathsf{r} = \mathsf{p}$, by definition 7 we obtain

$$[\![\mathsf{p} \text{ -> } \mathsf{q}; C_1]\!]_{\mathsf{p}} = \mathsf{q}!; [\![C_1]\!]_{\mathsf{p}} = \mathsf{q}!; [\![C_2]\!]_{\mathsf{p}} = [\![\mathsf{p} \text{ -> } \mathsf{q}; C_2]\!]_{\mathsf{p}} \ .$$

Similarly, for $\mathsf{r} = \mathsf{q}$, by definition 7 we obtain

$$[\![\mathsf{p} \text{ -> } \mathsf{q}; C_1]\!]_{\mathsf{q}} = \mathsf{p}?; [\![C_1]\!]_{\mathsf{q}} = \mathsf{p}?; [\![C_2]\!]_{\mathsf{q}} = [\![\mathsf{p} \text{ -> } \mathsf{q}; C_2]\!]_{\mathsf{q}} \ .$$

When $\mathsf{r} \notin \{\mathsf{p},\mathsf{q}\}$ we have the simplest case. Again, by definition 7,

$$[\![\mathsf{p} \text{ -> } \mathsf{q}; C_1]\!]_{\mathsf{r}} = [\![C_1]\!]_{\mathsf{r}} = [\![C_2]\!]_{\mathsf{q}} = [\![\mathsf{p} \text{ -> } \mathsf{q}; C_2]\!]_{\mathsf{r}} \ .$$

■

# List of Figures

# List of Notations

$C$ A choreography. 30

$N$ A network. 37

$[\![C]\!]_{\mathsf{p}}$ The projection of the behaviour of process $\mathsf{p}$ in choreography $C$. 42

$[\![C]\!]$ The EndPoint Projection (EPP) of choreography $C$. 43

$\mu$ A label of a labelled transition system. 31

$\tau$ The internal action of a labelled transition system. 60

$\longrightarrow$ The transition relation. 31

**!** A (possibly) difficult exercises. 10

$\hookrightarrow$ An exercise for which a solution is provided in appendix A. 10

$\mathsf{p}$ A process name, also called process identifier (pid for short). 30

$\mathsf{pn}$ The function that computes the set of process names in a term. 43

$\mathsf{supp}$ The support of a function. 37

# Index

# Bibliography


Davide Ancona, Viviana Bono, Mario Bravetti, Joana Campos, Giuseppe Castagna, Pierre-Malo Deniélou, Simon J. Gay, Nils Gesbert, Elena Giachino, Raymond Hu, Einar Broch Johnsen, Francisco Martins, Viviana Mascardi, Fabrizio Montesi, Rumyana Neykova, Nicholas Ng, Luca Padovani, Vasco T. Vasconcelos, and Nobuko Yoshida. Behavioral types in programming languages. *Foundations and Trends in Programming Languages*, 3(2-3):95–230, 2016.

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

Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured communication-centred programming for web services. In Rocco De Nicola, editor, *ESOP*, volume 4421 of *LNCS*, pages 2–17. Springer, 2007. doi: 10.1007/978-3-540-71316-6_2.

Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured communication-centered programming for web services. *ACM Trans. Program. Lang. Syst.*, 34 (2):8, 2012.

Luís Cruz-Filipe and Fabrizio Montesi. Procedural choreographic programming. In *FORTE*, LNCS. Springer, 2017.

Luís Cruz-Filipe and Fabrizio Montesi. A core model for choreographic programming. *Theoretical Computer Science*, 2019. ISSN 0304-3975. doi: https://doi.org/10.1016/j.tcs.2019.07.005. URL http://www.sciencedirect.com/science/article/pii/S0304397519304311.

Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types. In *ICALP (2)*, pages 174–186, 2013.

Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. Microservices: yester-

day, today, and tomorrow. In *Present and Ulterior Software Engineering*, pages 195–216. Springer, 2017.

Xiang Fu, Tevfik Bultan, and Jianwen Su. Realizability of conversation protocols with message contents. *International Journal on Web Service Res.*, 2(4):68–93, 2005.

Matthew Hennessy. *A distributed Pi-calculus*. Cambridge University Press, 2007.

Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty Asynchronous Session Types. *J. ACM*, 63(1):9, 2016. doi: 10.1145/2827695. URL http://doi.acm.org/10.1145/2827695.

Hans Hüttel, Ivan Lanese, Vasco T. Vasconcelos, Luís Caires, Marco Carbone, Pierre-Malo Deniélou, Dimitris Mostrous, Luca Padovani, António Ravara, Emilio Tuosto, Hugo Torres Vieira, and Gianluigi Zavattaro. Foundations of session types and behavioural contracts. *ACM Comput. Surv.*, 49(1):3:1–3:36, 2016.

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

Ivan Lanese, Claudio Guidi, Fabrizio Montesi, and Gianluigi Zavattaro. Bridging the gap between interaction- and process-oriented choreographies. In *SEFM*, pages 323–332, 2008.

Per Martin-Löf. On the meanings of the logical constants and the justifications of the logical laws. *Nordic journal of philosophical logic*, 1(1):11–60, 1996.

Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer, Berlin, 1980.

R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, December 1978. ISSN 0001-0782. doi: 10.1145/359657.359659.

Object Management Group. Business Process Model and Notation. http://www.omg.org/spec/BPMN/2.0/, 2011.

OECD. Horizon scan of megatrends and technology trends in the context of future research policy, 2016. http://ufm.dk/en/publications/2016/an-oecd-horizon-scan-of-megatrends-and-technology-trends-in-the-c ontext-of-future-research-policy.

## BIBLIOGRAPHY

F. Pfenning. Lecture Notes on Deductive Inference, 2012. https://www.cs.cmu.edu/ fp/courses/15816-s12/lectures/01-inference.pdf.

Gordon D. Plotkin. A structural approach to operational semantics. *J. Log. Algebr. Program.*, 60-61:17–139, 2004.

Z. Qiu, X. Zhao, C. Cai, and H. Yang. Towards the theoretical foundation of choreography. In *WWW*, pages 973–982. ACM, 2007.

R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978. ISSN 0001-0782. doi: 10.1145/359340.359342. URL http://doi.acm.org/10.1145/359340.359342.

Davide Sangiorgi. *Introduction to Bisimulation and Coinduction*. Cambridge University Press, 2011. doi: 10.1017/CBO9780511777110.

W3C WS-CDL Working Group. Web services choreography description language version 1.0. http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/, 2004.