Introduction to Choreographies

Fabrizio Montesi
fmontesi@imada.sdu.dk
Department of Mathematics and Computer Science
University of Southern Denmark

Last update: September 18, 2018

Abstract

This document contains lecture notes on choreographies for the course on DM861 Concurrency Theory (2018) at the University of Southern Denmark. It is organised as a working book. Different parts get updated during the course.

The chapters in advance of the lectures are given as a preview on what is to come, but **remember to download the updated version of this book every week to stay up to date!**

Contents

Preface: Alice, Bob, and Choreographies

We live in the era of concurrency and distribution. Concurrency gives us the ability to perform multiple tasks at a time. Distribution allows us to do so at a distance. It is the era of the web, mobile computers, cloud computing, and microservices. The number of computer programs that communicate with other programs over a network is exploding. By 2025, the Internet alone might is expected to connect from 25 to 100 billion devices [OECD, 2016].

Modern computer networks and their applications are key 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.

Modern computer networks and their applications are complex. Services are becoming increasingly dependent on other services. 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 brower, 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 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, because each party needs to know what it is supposed to do such that integration is successful and the whole system works. At the same time, it is important that protocols are as concise as possible: the bigger a protocol, the higher the chance that we made 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 might say that we need a good *notation.*

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

$$A \rightarrow B : M.$$

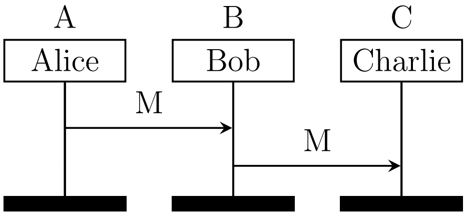
To define a protocol where A sends a message M to B, and then B passes the same message to C, we can just compose communications in a sequence:

$$\begin{gathered} A \rightarrow B : M \\ B \rightarrow C : M. \end{gathered}$$

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 for 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 the case in general: not all notations are equally expressive, as we will see in the rest of this book.[1]

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 is essentially still a protocol, but the emphasis is on detail. Choreographies might include details like the following.

- The kind of data being transmitted, or even the actual functions used to compute the data to be transmitted.
- Explicit cause-effect relations (also known as causal dependencies) between the data being transmitted and the choices made by participants in a protocol.
- The state of participants, e.g., their memory states.

Choreographies are typically written in languages designed to be readable *mechanically*, which also make them amenable to be used by a computer. 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 started emerging 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. In 2011, the global technology standards consortium Object Management Group (OMG) introduced choreographies in their notation for business processes [BPMN]. The recent paradigm of microservices [Dragoni et al., 2017] advocates for 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].

It is the time of Alice and Bob. It is the time of choreographies.

[1] Message sequence charts in particular support many features, including timeouts and alternative behaviours.

This book

This book is an introduction to the theory of choreographies. It explains what choreographies are and how we can model them mathematically. Its primary audiences are computer science students and researchers. However, the book should be approachable by anybody interested in studying choreographies, e.g., for theoretical purposes or the development of choreography-based tools. The aim of this book is to be pedagogical. It is not an aim to be comprehensive. References to alternative notations and techniques to model choreographies are given where appropriate. It is assumed that the reader is familiar with the notion of concurrency and 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 in terms of context-free grammars. To give meaning to choreographies, we are going to use extensively Plotkin's structural approach to operational semantics.

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 not a requirement to read this book: chapter 1 provides an 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 ideas from Milner's seminal calculus of communicating systems [Milner, 1980].

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]

An inference system is a set of *inference rules* (also called rules of inference). An inference rule has the form

$$\frac{\text{Premise 1} \qquad \text{Premise 2} \qquad \cdots \qquad \text{Premise } n}{\text{Conclusion}}$$

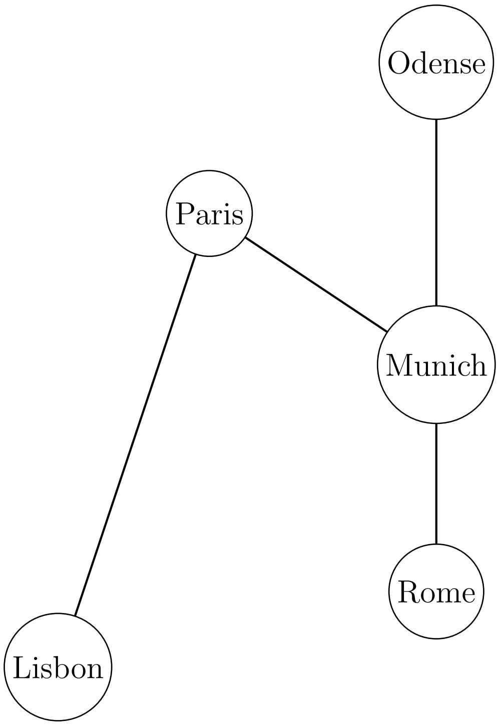
and reads "If the premises Premise 1, Premise 2, …, and Premise n hold, then Conclusion holds". It is perfectly valid for an inference rule to have no premises: we call such rules *axioms*, because their conclusions always hold. An inference rule has always exactly one conclusion.

1.1 Example: Flight connections

This example is inspired by the usage of rules of inference to model graphs by Pfenning [2012].

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.

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

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

Notice that our graph is undirected: in an ideal world, 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., for $\mathsf{conn}(\text{Munich}, \text{Odense})$). A more elegant option is to include a rule of inference for symmetry, 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 us that if we have a connection from any A to any B (premise),

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

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

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

Figure 1.1: An inference system for flights.

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.

Now that we are satisfied with how our rule system captures the graph, we can use it to find flight paths from a city to another. 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. (Dɪʀ 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. (Tʀᴀɴs in the following rule 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 $\mathsf{path}(\mathrm{Odense}, \mathrm{Rome})$ holds. To do this, we build a *derivation tree* (also simply called *derivation*, or *proof tree*). The problem tackled in the following is also known as proof search.

We start 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. Dir cannot be applied: we would need to prove $\mathsf{conn}(\text{Odense}, \text{Rome})$, which is impossible. So our only choice is rule Trans, by instantiating A as Odense and C as Rome. How should we instantiate B in rule Trans, though?

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

We need to guess a correct instantiation for B and hope that it will allow us to continue our derivation. Looking at the definition of our graph, it is easy to see that the right choice is to pick Munich.

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

Our derivation is not complete yet, because we are left with the tasks of proving (deriving) $\mathsf{path}(\text{Odense}, \text{Munich})$ and $\mathsf{path}(\text{Munich}, \text{Rome})$. Thanks to the notation of inference rules, we can just “dig 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.

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

Our derivation is done! We can tell by the fact that there are no more premises left to prove.

If you look carefully, you can see that our derivation is a tree. The conclusion $\mathsf{path}(\text{Odense}, \text{Rome})$ is the root of the tree, and the leaves are empty nodes (the premises of our axioms). Rule applications connect the nodes of the tree. In fact, even our previous partial derivations are all trees. This is a very useful property that we will use throughout the book.

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

1.3 Proof non-existence

Proof, or proof of no proof. There is no try.
- Yoda, to an exhausted Luke

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.

What about showing that a proposition *cannot* be derived? This can be far trickier, because it requires us to reason about all the possible proofs that could, potentially, conclude with our proposition and showing that none of those proofs 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. None of the axioms can be applied for $A = \text{Lisbon}$ and $B = \text{Rome}$, as in our case. We are left with rule Sym: any search of a derivation of $\mathsf{conn}(\text{Lisbon}, \text{Rome})$ must begin as follows.

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

By similar reasoning (there is no rule to apply for $\mathsf{conn}(\text{Rome}, \text{Lisbon})$ but Sym), 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.

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

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.

Another way of proving that $\mathsf{conn}(\text{Lisbon}, \text{Rome})$ is by looking at how proofs are constructed from the top, rather than the bottom. For our system, we can prove the following result. We use $\mathcal{P}$ to range over proofs (derivations).

Proposition 1. *For any natural number n, there exists no proof of $\mathsf{conn}(\mathit{Lisbon}, \mathit{Rome})$ of height n.*

Proof. We prove the stronger result that there exists no proof of $\mathsf{conn}(\text{Lisbon}, \text{Rome})$ and there exists no proof of $\mathsf{conn}(\text{Rome}, \text{Lisbon})$, either.

We proceed by induction on n.

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

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

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

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

Figure 1.2: Weighted rules for flight paths.

Base case: $n = 1$. All proofs with height 1 must necessarily be an application of an axiom, and there is no axiom that can prove either $\mathsf{conn}(\text{Lisbon}, \text{Rome})$ or $\mathsf{conn}(\text{Rome}, \text{Lisbon})$.

Inductive case: $n = m + 1$ for some natural number m. By induction hypothesis, we know that there is no proof $\mathcal{P}$ such that the height of $\mathcal{P}$ is m and that the conclusion of $\mathcal{P}$ is $\mathsf{conn}(\text{Lisbon}, \text{Rome})$ or $\mathsf{conn}(\text{Rome}, \text{Lisbon})$. Thus, all proofs $\mathcal{Q}$ of height m conclude with either: i) $\mathsf{conn}(A, B)$ for some A and B such that the set $\{A, B\}$ is different from $\{\text{Lisbon}, \text{Rome}\}$; or ii) $\mathsf{path}(A, B)$ for some A and B. For i), we observe that there is no rule that allows us to derive $\mathsf{conn}(\text{Lisbon}, \text{Rome})$ from a premise that is not $\mathsf{conn}(\text{Rome}, \text{Lisbon})$, and likewise for the symmetric case where the premise is $\mathsf{conn}(\text{Rome}, \text{Lisbon})$. For ii), we observe that there is no rule that, given a conn proposition as one of its premises, allows us to conclude $\mathsf{conn}(\text{Lisbon}, \text{Rome})$ or $\mathsf{conn}(\text{Rome}, \text{Lisbon})$.

□

It follows from proposition 1 that there is no proof of $\mathsf{conn}(\text{Lisbon}, \text{Rome})$. Assume that there were such a proof. Since it would have to be finite, it would have a height n. Thus we reach a contradiction, because proposition 1 states that for all n there is no proof of $\mathsf{conn}(\text{Lisbon}, \text{Rome})$.

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

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

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

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

Figure 1.3: A limited and weighted flight system.

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

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

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

Figure 1.4: An alternative way of constructing paths.

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 $\mathsf{path}(A, B)$.

Exercise 2 (!)**.** *Consider the system in fig. 1.3, which removes the direct flight from Paris to Munich.*

Prove that it is not possible to derive $\mathsf{path}(\mathit{Lisbon}, \mathit{Munich}, n)$ for any n.

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, 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).

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.

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

The derivation above 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.

1.4.2 Rule admissibility

Let us look at the other direction: proving that if adding rule Trans to the system with rule Step would not add any new derivable propositions.

Recall that rule Trans is defined as follows.

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

As a first attempt, we could try to adopt 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 for rule Trans). It is sometimes convenient to mark admissible rules explicitly when defining them. We will use dashed horizontal lines to denote admissible inference rules.

Theorem 1. *The rule*

$$\begin{array}{c} \mathsf{path}(A,B) \quad \mathsf{path}(B,C) \\ \hdashline \mathsf{path}(A,C) \end{array}\ \textsc{Trans}$$

is admissible in the system in fig. 1.4.

Proof. Let $\mathcal{P}$ and $\mathcal{Q}$ be the derivations of $\mathsf{path}(A,B)$ and $\mathsf{path}(B,C)$, respectively. We proceed by induction on the structure of the derivation

$$\begin{array}{c} \dfrac{\mathcal{P}}{\mathsf{path}(A,B)} \quad \dfrac{\mathcal{Q}}{\mathsf{path}(B,C)} \\ \hdashline \mathsf{path}(A,C) \end{array}\ \textsc{Trans}$$

and prove that we can build a derivation using the rules in fig. 1.4 *that is not bigger* (this is going to be important for the last case of the proof). We have three cases, depending on the last applied rule in $\mathcal{P}$ (one for each rule that can conclude a path proposition).

Case Dir $\mathcal{P}$ ends with an application of rule Dir. Thus, for some $\mathcal{P}'$ that does not contain any applications of rule Trans, we are in the following situation.

$$\dfrac{\dfrac{\dfrac{\mathcal{P}'}{\mathsf{conn}(A,B)}}{\mathsf{path}(A,B)}\,\text{Dir}\qquad\dfrac{\mathcal{Q}}{\mathsf{path}(B,C)}}{\mathsf{path}(A,C)}\,\text{Trans}$$

We rewrite the proof as follows.

$$\dfrac{\dfrac{\mathcal{P}'}{\mathsf{conn}(A,B)}\qquad\dfrac{\mathcal{Q}}{\mathsf{path}(B,C)}}{\mathsf{path}(A,C)}\,\text{Step}$$

If $\mathcal{Q}$ does not contain applications of rule Trans, then this is the base case and the thesis follows immediately. Otherwise, the thesis follows by induction hypothesis.

Case Step $\mathcal{P}$ ends with an application of rule Step. Thus, for some $\mathcal{P}'$ and $\mathcal{P}''$, we are in the following situation.

$$\dfrac{\dfrac{\dfrac{\mathcal{P}'}{\mathsf{conn}(A,B')}\qquad\dfrac{\mathcal{P}''}{\mathsf{path}(B',B)}}{\mathsf{path}(A,B')}\,\text{Step}\qquad\dfrac{\mathcal{Q}}{\mathsf{path}(B,C)}}{\mathsf{path}(A,C)}\,\text{Trans}$$

We rewrite the proof as follows.

$$\dfrac{\dfrac{\mathcal{P}'}{\mathsf{conn}(A,B')}\qquad\dfrac{\dfrac{\mathcal{P}''}{\mathsf{path}(B',B)}\qquad\dfrac{\mathcal{Q}}{\mathsf{path}(B,C)}}{\mathsf{path}(B',C)}\,\text{Trans}}{\mathsf{path}(A,C)}\,\text{Step}$$

The thesis now follows by induction hypothesis.

Case Trans $\mathcal{P}$ ends with an application of rule Trans. Thus, for some $\mathcal{P}'$ and $\mathcal{P}''$, we are in the following situation.

$$\dfrac{\dfrac{\dfrac{\mathcal{P}'}{\mathsf{path}(A,B')}\qquad\dfrac{\mathcal{P}''}{\mathsf{path}(B',B)}}{\mathsf{path}(A,B)}\,\text{Trans}\qquad\dfrac{\mathcal{Q}}{\mathsf{path}(B,C)}}{\mathsf{path}(A,C)}\,\text{Trans}$$

From the proof of the left-hand premise and the induction hypothesis, we know that there exists a proof $\mathcal{R}$ that does not contain applications of rule Trans, ends with $\mathsf{path}(A, B)$, and is not bigger than the left-hand side proof

$$\dfrac{\dfrac{\mathcal{P}'}{\mathsf{path}(A, B')} \qquad \dfrac{\mathcal{P}''}{\mathsf{path}(B', B)}}{\mathsf{path}(A, B)}\ \textsc{Trans}.$$

Thus we can build the following smaller derivation, by eliminating at least one application of rule Trans.

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

Since the derivation is smaller than the one we started with, we can conclude by invoking the induction hypothesis on this derivation.

$\square$

Exercise 3 (!)**.** *Prove that if* $\mathsf{path}(A, B)$ *is provable in the system in fig. 1.4, then there exists a nonempty sequence of provable 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 provable in the system in fig. 1.1, then there exists a nonempty sequence of provable 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*).*

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.

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

$$C ::= \mathsf{p} \text{ -> } \mathsf{q}; C \mid \mathbf{0}$$

Figure 2.1: Simple choreographies, syntax.

2.1 Syntax

Our first choreography model is called simple choreographies. The syntax of simple choreographies is given by the grammar in fig. 2.1, where C ranges over a choreography. Choreographies describe interactions between processes. We refer to processes by using process names, ranged over by p, q.

The syntax of simple choreographies is minimalistic. A choreography can either be a term p -> $\mathsf{q}; C$ or term $\mathbf{0}$. Term p -> $\mathsf{q}; C$ is an interaction and reads "process p sends a message to process q; then, the choreography proceeds as C". We always assume that p and q are different in interactions p -> q, i.e., $\mathsf{p} \neq \mathsf{q}$ (meaning that a process cannot send a message to itself). Term $\mathbf{0}$ is the terminated choreography (no interactions, or end of program, if you like). Sometimes, we refer to terms as *programs* in the remainder.

Example 2. *The following choreography defines the behaviour that we informally described in example 1.*

$$\mathsf{Buyer} \text{ -> } \mathsf{Seller}; \mathsf{Seller} \text{ -> } \mathsf{Buyer}; \mathbf{0}$$

Note that what we have is actually a rather coarse abstraction *of what we described in example 1, 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 semantic interpretation. We use a relation to do that. Recall from set theory that a relation $\mathcal{R}$ on two sets S and S' is a subset of the product $S \times S'$ (the set of all pairs of elements from S and S'). Given an element $s \in S$ and an element $s' \in S'$, we write $s\mathcal{R}s'$ for $(s, s') \in \mathcal{R}$.

$$\frac{}{\mathsf{p} \text{ -> } \mathsf{q}; C \to C}\ \textsc{Com}$$

Figure 2.2: Simple choreographies, semantics.

Formally, the semantics for simple choreographies is given in terms of a *reduction relation* $\to$, which is defined as the smallest relation satisfying the rule displayed in fig. 2.2. Whenever two choreographies C and C' are related by $\to$, written $C \to C'$, we say that there is a reduction from C to C'. A reduction models executing a step of a choreography.

There is only one rule, called Com, which is an axiom: it always allows us to reduce 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 true also here.)

The sets over which the relation $\to$ is defined are given implicitly: they are evident from the inference rules that define the relation. Specifically, let *SimpleChor* be the set of all choreography terms in our grammar for simpler choreographies—or, equivalently, the *language* generated by that grammar. There is only one inference rule in fig. 2.2, rule Com. The rule relates any choreography of the form $\mathsf{p} \text{ -> } \mathsf{q}; C$ to the choreography C. Thus, we know that $\to \subseteq \mathit{SimpleChor} \times \mathit{SimpleChor}$.

It is important that the set of departure of relation $\to$ ($\mathit{SimpleChor}$) is the same as its set of destination ($\mathit{SimpleChor}$), i.e., that $\to$ goes from choreographies to choreographies. This property allows us to define the useful concept of strong reduction chain, which permits to observe multiple steps of execution. The “strong” qualifier indicates that there is at least one step in the chain.

Definition 1 (Strong reduction chain)**.** *We say that there is a strong reduction chain from C_1 to C_n whenever there exists a sequence of choreographies $(C_1, \ldots, C_n)$ such that $C_i \to C_{i+1}$ for each $i \in [1, n-1]$.*

When there is a strong reduction chain from C_1 to C_n, we write $C_1 \to^+ C_n$ (when showing the intermediate steps is not necessary, only their existance) or $C_1 \to \cdots \to C_n$ (when we want to show the intermediate steps).

Example 3. *The program in example 2 has the following strong reduction chain:*

$$\mathsf{Buyer} \text{ -> } \mathsf{Seller}; \mathsf{Seller} \text{ -> } \mathsf{Buyer}; \mathbf{0} \quad \to \quad \mathsf{Seller} \text{ -> } \mathsf{Buyer}; \mathbf{0} \quad \to \quad \mathbf{0}.$$

So we first reduce an interaction where Buyer *sends a message to* Seller *and then we reduce an interaction where* Seller *sends a message to* Buyer*, which is exactly the communication flow that we wanted in example 1.*

Another way to define relation $\rightarrow^+$ is: $\rightarrow^+$ is the transitive closure of $\rightarrow$. In the next sections, we will also use the reflexive and transitive closure of $\rightarrow$, written $\rightarrow^*$ and defined as follows. Whenever $C \rightarrow^* C'$ for some C and C', we say that there is a weak reduction chain from C to C'.

Definition 2 (Weak reduction chain)**.** *We write $C \rightarrow^* C'$ if either:*

Base case: *$C = C'$; or,*

Inductive case: *there exists C'' such that $C \rightarrow C''$ and $C'' \rightarrow^* C'$.*

If you are familiar with regular expressions, the use of the symbols $+$ and $*$ should ring a bell: $\rightarrow^+$ means "one or more reductions" and $\rightarrow^*$ means "zero or more reductions".

Exercise 4. *Prove that $C \rightarrow^+ C'$ implies $C \rightarrow^* C'$. Prove that the converse does not hold.*

Exercise 5. *Prove the following statements.*

1. *For all $C \neq \mathbf{0}$ (all C that are not $\mathbf{0}$), there exists C' such that $C \rightarrow C'$.*

2. *For all $C \neq \mathbf{0}$, there exists C' such that $C \rightarrow^* C'$.*

3. *For all $C \neq \mathbf{0}$, $C \rightarrow^+ \mathbf{0}$.*

4. *For all C, $C \rightarrow^* \mathbf{0}$.*

Appendix A

Solution to selected exercises

We give the solutions to some selected exercises. Some solutions are given in full detail, to serve as examples of exposition. All solutions should still provide enough information for the reader to figure out the remaining parts.

Solution of exercise 1. 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:

- $\mathsf{conn}(A, B)$ provable in fig. 1.1 implies $\mathsf{conn}(A, B)$ provable in fig. 1.2;
- $\mathsf{path}(A, B)$ provable in fig. 1.1 implies $\mathsf{path}(A, B, n)$ for some n provable in fig. 1.2.

The proof is by induction on the structure of the derivation of $\mathsf{conn}(A, B)$ or $\mathsf{path}(A, B)$. We proceed by cases on the last applied rule of the derivation.

Base cases (axioms) If the last applied rule is an axiom, then the proof is valid also in the other system.

Case Sym The derivation has this shape:

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

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

$$\dfrac{\mathcal{P}'}{\mathsf{conn}(B, A).}$$

The thesis follows by applying rule $\textsc{Sym}$.

$$\cfrac{\cfrac{\mathcal{P}'}{\mathsf{conn}(B, A)}}{\mathsf{conn}(A, B)}\ \textsc{Sym}\,.$$

Case Dir The derivation has this shape:

$$\cfrac{\cfrac{\mathcal{P}}{\mathsf{conn}(A, B)}}{\mathsf{path}(A, B)}\ \textsc{Dir}\,.$$

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

$$\frac{\mathcal{P}'}{\mathsf{conn}(A, B).}$$

The thesis follows by applying rule $\textsc{Dir}$W.

$$\cfrac{\cfrac{\mathcal{P}'}{\mathsf{conn}(A, B)}}{\mathsf{path}(A, B, 1)}\ \textsc{Dir}\mathrm{W}\,.$$

Case Trans The derivation has this shape:

$$\cfrac{\cfrac{\mathcal{P}}{\mathsf{path}(A, B)} \quad \cfrac{\mathcal{Q}}{\mathsf{path}(B, C)}}{\mathsf{path}(A, C)}\ \textsc{Trans}\,.$$

By induction hypothesis on $\mathcal{P}$ and by induction hypothesis on $\mathcal{Q}$, we know that there exist n and m, $\mathcal{P}'$ and $\mathcal{Q}'$ in the other system such that:

$$\frac{\mathcal{P}'}{\mathsf{path}(A, B, n)} \qquad \frac{\mathcal{Q}'}{\mathsf{path}(A, B, m).}$$

The thesis follows by applying rule $\textsc{Trans}$W.

$$\cfrac{\cfrac{\mathcal{P}'}{\mathsf{path}(A, B, n)} \quad \cfrac{\mathcal{Q}'}{\mathsf{path}(A, B, m)}}{\mathsf{path}(A, B, n + m)}\ \textsc{Trans}\mathrm{W}\,.$$

$\blacksquare$

List of Figures

List of Notations

$\mathcal{P}$ A derivation in an inference system. 12

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.

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

Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. Microservices: yesterday, today, and tomorrow. In *Present and Ulterior Software Engineering*, pages 195–216. Springer, 2017.

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.

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.

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.

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.

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`.