Lecture Notes on Choreographies, Part 1

Fabrizio Montesi
fmontesi@imada.sdu.dk

October 30, 2017

Abstract

This document contains lecture notes for the course on Concurrency Theory (2017) at the University of Southern Denmark.

1 Introduction

In the previous lectures, we have familiarised ourselves with deductive reasoning based on inference rules and, building on those, formal models for processes (process calculi, or process algebras).

Process calculi are interesting because they abstract significantly from the implementation details of concurrent systems, yet they retain enough expressivity to allow for the abstract modelling of interaction structures. They help in reasoning about the design and implementation of concurrent systems, because such systems typically operate in terms of Input/Output actions (I/O for short). Summarising, what we do in this paradigm is: we program each process separately, by defining the I/O that it should perform; then, we compose processes in a network by using the parallel operator $|$. Let us call this the *local-view paradigm*, since the focus is on the local actions performed by processes and how the composition of these lead to process interaction.

The local-view paradigm is adequate for the (abstract) modelling of what *will* happen in the execution of a system. However, another aspect that is just as important is the design of what a system *should* do. If we could mathematically define our expectations for what a system should do, then we could hope for building methods that ensure that what the system *will* do and what the system *should* do match. Exploring these aspects is what we are going to do in these lecture notes.

2 From action to interaction: towards global views

How do we define what a concurrent system should do? There are many possible ways to do this, depending on what kind of properties we are interested in. Arguably, one of the most foundational aspects is "what interactions do we want our concurrent system to implement?".

Let us see a simple example to get a bit more concrete about our question. 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, so it does not serve our purpose of mathematically defining our expectations[1]. However, it gives us an important indication about how a mathematical formalism that supports our objective may look like: our description talks about *multiple* processes and how they interact. In other words, when we describe what we want to happen, we are adopting a global view on all the interactions among the processes that we are interested in. This is in contrast with the local view on the actions performed by each process. Hopefully, we can work towards bridging the two[2].

3 A simple global language

We now move to studying how we can design formal models for a global view of concurrent interactions. Technically, we are going to study languages for choreographies, drawing from a research line that has been particularly productive over the last decade. For reference, two surveys that contain relevant information have been written by Ancona et al. [2016] and Hüttel et al. [2016]. Choreographies are the basis for theories of communication protocols, Multiparty Session Types [Honda et al., 2016] being a prominent example, and emerging programming paradigms, like Choreographic Programming [Montesi, 2013, 2015]. We are going to see more about some of these applications in later lectures.

[1] Software engineers actually specify concurrent systems in similar ways using different kinds of tools, like Message Sequence Charts [International Telecommunication Union, 1996]. The artifacts produced are similar, in essence, to our informal description.

[2] Mathematically, of course!

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

Figure 1: Simple choreographies, syntax.

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

Figure 2: Simple choreographies, semantics.

We start by defining a very simple choreography language, given by the grammar in fig. 1. We use C to range over choreographies. 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. Term $\mathsf{p} \twoheadrightarrow \mathsf{q}; C$ 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 $\mathsf{p} \twoheadrightarrow \mathsf{q}$, 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).

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

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

Note that what we have is actually an abstraction *of what we described in section 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.*

We give a semantics for simple choreographies in terms of a reduction relation $\to$, which is defined as the smallest relation satisfying the rule displayed in fig. 2. Reductions have the form $C \to C'$.

There is only one rule, called Com, which always allows us to reduce interactions. This formalises the fact that if the programmer *wants* an interaction to take place, it will actually happen.

We can formally observe whether a choreography matches our intended behaviour by studying its reduction chains.

Definition 1. *We say that there is a 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 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 2. *The program in example 1 has the following reduction chain:*

$$\mathsf{Buyer} \to \mathsf{Seller}; \mathsf{Seller} \to \mathsf{Buyer}; \mathbf{0} \quad \to \quad \mathsf{Seller} \to \mathsf{Buyer}; \mathbf{0} \quad \to \quad \mathbf{0}$$

. 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 section 2.*

As for process calculi, it will be convenient to consider the transitive and reflective closure of $\to$ for choreographies to define properties of runs with potentially many steps. We denote this closure with $\to^*$.

Definition 2. *We write $C \to^* C'$ if either:*

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

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

Exercise 1. *Prove that $C \to^+ C'$ implies $C \to^* C'$. Does the converse also hold?*

4 A process model with process identifiers

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).

Differently from the process models that we have seen so far, we are going to have processes communicate by referring to their identifiers, inspired by mainstream frameworks based on actors.

The syntax of simple processes is given in fig. 3. We model systems as networks, ranged over by N. A network N can be: a single process term $\mathsf{p} \triangleright B$, where p is the name of the process and B its behaviour; a parallel composition of two networks N and N', written $N \mid N'$; or the empty network $\mathbf{0}$. A process behaviour, ranged over by B, can be: a send action $\mathsf{p}!; B$, read "send

$$N ::= \mathsf{p} \triangleright B \mid N \mid N \mid \mathbf{0}$$
$$B ::= \mathsf{p}!; B \mid \mathsf{p}?; B \mid \mathbf{0}$$

Figure 3: Simple processes, syntax.

$$\frac{}{\mathsf{p} \triangleright \mathsf{q}!; B \;\mid\; \mathsf{q} \triangleright \mathsf{p}?; B' \quad \to \quad \mathsf{p} \triangleright B \;\mid\; \mathsf{q} \triangleright B'}\ \mathsf{Com}$$

$$\frac{N_1 \to N_1'}{N_1 \mid N_2 \;\to\; N_1' \mid N_2}\ \mathsf{Par} \qquad \frac{N \preceq N_1 \quad N_1 \to N_2 \quad N_2 \preceq N'}{N \to N'}\ \mathsf{Struct}$$

Figure 4: Simple processes, semantics.

a message to process p and then do B"; a receive action $\mathsf{p}?; B$, read "receive a message from process p and then do B"; or the terminated behaviour $\mathbf{0}$.

The semantics of simple processes is given by the reduction rules displayed in fig. 4. Rule Com models communications, by matching a send action by process p towards process q with a receive action at q waiting for a message from p. Each process then proceeds with its respective continuation (B for p, B' for q). Rule Par allows for reductions to happen in a sub-network. Rule Struct closes reductions under the structural precongruence $\preceq$, which is defined as the smallest precongruence satisfying the rules in fig. 5[3]. We $N \equiv N'$ as a shortcut for $N \preceq N'$ and $N' \preceq N$ (this means that the precongruence is symmetric for that case).

The rules defining $\preceq$ are standard, and similar to those seen in previous lectures for process models. Parallel is associative (PA) and commutative (PC). Empty networks can be garbage collected (GCN), and the same for

[3] By precongruence, we mean that it is reflexive ($N \equiv N$ for all N), it is transitive, but it is not necessarily symmetric.

$$\frac{}{(N_1 \mid N_2) \mid N_3 \;\equiv\; N_1 \mid (N_2 \mid N_3)}\ \mathsf{PA}$$

$$\frac{}{N_1 \mid N_2 \;\equiv\; N_2 \mid N_1}\ \mathsf{PC} \qquad \frac{}{N \mid \mathbf{0} \;\preceq\; N}\ \mathsf{GCN} \qquad \frac{}{\mathsf{p} \triangleright \mathbf{0} \;\preceq\; \mathbf{0}}\ \mathsf{GCP}$$

Figure 5: Simple processes, structural precongruence.

processes with terminated behaviour (GCP).

Example 3. *We write a process implementation for the communication scenario informally described in section 2.*

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

The network proceeds as expected, as the following reduction chain shows.

$$\begin{aligned}
&\mathsf{Buyer} \triangleright \mathsf{Seller}!; \mathsf{Seller}?; \mathbf{0} \mid \mathsf{Seller} \triangleright \mathsf{Buyer}?; \mathsf{Buyer}!; \mathbf{0} \\
&\rightarrow \\
&\mathsf{Buyer} \triangleright \mathsf{Seller}?; \mathbf{0} \mid \mathsf{Seller} \triangleright \mathsf{Buyer}!; \mathbf{0} \\
&\rightarrow \\
&\mathsf{Buyer} \triangleright \mathbf{0} \mid \mathsf{Seller} \triangleright \mathbf{0}
\end{aligned}$$

Exercise 2 (!)**.** *Prove the following statement.*

If $N \preceq N'$ and $N \preceq N''$, there exists N''' such that $N' \preceq N'''$ and $N'' \preceq N'''$.

5 From choreographies to processes

The network in example 3 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.

To gain some intuition on how we could develop the method we want, we can look at our examples. Let us see our choreography from example 1 again:

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

. It is evident, albeit informally, that our network from example 3 implements exactly the interactions defined in the choreography:

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

. This informal correspondence is preserved by reductions—remember, reductions model execution, if you think in computational terms. Indeed, whenever we take a step in the reduction chain shown in example 2 (for the choreography), we can "mimic" it by following the reduction chain in example 3

(for the process network), and vice versa (if we take a step for the process network, we can mimic it for the choreography).

The intuition that we can gain from our examples is that a network "implements" a choreography if the actions performed by processes give rise to the interactions described in the choreography. Therefore, an automatic method that produces networks from choreographies should generate a network that consists of the processes described in the choreography, and the behaviour of each of these processes should be the actions that the process needs to perform to implement the interactions that it is involved in in the choreography.

We can now move to formally defining our desired method, 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 $\mathsf{Buyer} \to \mathsf{Seller}$ 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.

Let $\mathsf{procs}(C)$ be the set of process names used in C. We can define this function inductively on the structure of C, as follows.

$$\mathsf{procs}\left(\mathsf{p} \to \mathsf{q}; C\right) = \{\mathsf{p}, \mathsf{q}\} \cup \mathsf{procs}(C)$$
$$\mathsf{procs}(\mathbf{0}) = \emptyset$$

We write $[\![C]\!]$ for the EPP of a choreography C.

Definition 3 (EndPoint Projection (EPP))**.** *The EPP of a choreography C, denoted $[\![C]\!]$, is defined as:*

$$[\![C]\!] = \prod_{\mathsf{p} \in \mathsf{procs}(C)} \mathsf{p} \triangleright [\![C]\!]_{\mathsf{p}}$$

.

In definition 3, the notation $\prod_{\mathsf{p} \in \mathsf{procs}(C)} \mathsf{p} \triangleright [\![C]\!]_{\mathsf{p}}$ stands for "the parallel composition of all $\mathsf{p} \triangleright [\![C]\!]_{\mathsf{p}}$ such that p is in $\mathsf{procs}(C)$". The auxiliary function $[\![C]\!]_{\mathsf{p}}$—not to be confused with $[\![C]\!]$—projects the actions that process p should perform in order to implement its part in choreography C. We call $[\![C]\!]_{\mathsf{p}}$ a behaviour projection, since it outputs a behaviour B. It is inductively defined by the rules given in fig. 6.

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

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

Figure 6: Behaviour projection for simple choreographies.

Example 4. *Let $C_{\mathsf{BuyerSeller}}$ be the choreography in example 1. We recall it here ($\triangleq$ stands for "defined as"):*

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

. The process names in $C_{\mathsf{BuyerSeller}}$ are:

$$\mathsf{procs}(C_{\mathsf{BuyerSeller}}) \quad = \quad \{\mathsf{Buyer}, \mathsf{Seller}\}$$

. The behaviour projection for Buyer *is:*

$$\llbracket C_{\mathsf{BuyerSeller}} \rrbracket_{\mathsf{Buyer}} \quad = \quad \mathsf{Seller}!; \mathsf{Seller}?; \mathbf{0}$$

. The EPP of $C_{\mathsf{BuyerSeller}}$ is exactly the network that we defined in example 3:

$$\llbracket C_{\mathsf{BuyerSeller}} \rrbracket \quad = \quad \mathsf{Buyer} \triangleright \mathsf{Seller}!; \mathsf{Seller}?; \mathbf{0} \;\mid\; \mathsf{Seller} \triangleright \mathsf{Buyer}?; \mathsf{Buyer}!; \mathbf{0}$$

.

Exercise 3. *Write the outputs of* procs *and $\llbracket\rrbracket$ (EPP) for the choreography*

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

.

Exercise 4. *What are the reduction chains for the choreography in exercise 3 and its EPP? Do you think that they informally correspond to one another? (We have not formally defined correspondence yet.)*

Exercise 5 (!)**.** *Is the following statement true?*

Let $N = \llbracket C \rrbracket$. If $N \to N'$ for some N', then there exists C' such that $C \to C'$ and $N' \preceq \llbracket C' \rrbracket$.

If it is not true, how would you change the choreography model to fix this?

References

D. Ancona, V. Bono, M. Bravetti, J. Campos, G. Castagna, P. Deniélou, S. J. Gay, N. Gesbert, E. Giachino, R. Hu, E. B. Johnsen, F. Martins, V. Mascardi, F. Montesi, R. Neykova, N. Ng, L. Padovani, V. T. Vasconcelos, and N. Yoshida. Behavioral types in programming languages. *Foundations and Trends in Programming Languages*, 3(2-3):95–230, 2016.

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

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

H. Hüttel, I. Lanese, V. T. Vasconcelos, L. Caires, M. Carbone, P. Deniélou, D. Mostrous, L. Padovani, A. Ravara, E. Tuosto, H. T. Vieira, and G. 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.

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

F. Montesi. *Choreographic Programming.* Ph.D. Thesis, IT University of Copenhagen, 2013. URL `http://fabriziomontesi.com/files/choreographic_programming.pdf`.

F. Montesi. Kickstarting choreographic programming. In *WS-FM*, volume 9421 of *Lecture Notes in Computer Science*, pages 3–10. Springer, 2015.

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