Lecture Notes on Choreographies, Part 3

Fabrizio Montesi
fmontesi@imada.sdu.dk

November 10, 2017

Abstract

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

1 EPP for Stateful Choreographies

In part 2, we left the definition of EPP for stateful choreographies as an exercise. We develop it in this section, for reference.

Definition 1 (EndPoint Projection (EPP))**.** *The EPP of a configuration $\langle C, \sigma \rangle$, denoted $[\![\langle C, \sigma \rangle]\!]$, is defined as:*

$$[\![\langle C, \sigma \rangle]\!] = \prod_{\mathsf{p} \in \mathsf{procs}(C)} \mathsf{p} \triangleright_{\sigma(\mathsf{p})} [\![C]\!]_{\mathsf{p}}$$

.

We also need to update the definition of behaviour projection—$[\![C]\!]_{\mathsf{p}}$—since the language of stateful choreographies is different from that of simple choreographies. We display the new rules in fig. 1.

Exercise 1. *Formulate the operational correspondence theorem for EPP in the setting of stateful choreographies.*

Exercise 2. *Prove the theorem that you have formulated in exercise 1.*

2 Conditionals

The choreographies that we have seen so far are simple sequences of interactions. What if we wanted to express a choice between alternative behaviours?

$$\llbracket \mathsf{p}.f \,\text{->}\, \mathsf{q}.g; C \rrbracket_{\mathsf{r}} = \begin{cases} \mathsf{q}!f; \llbracket C \rrbracket_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{p} \\ \mathsf{p}?g; \llbracket C \rrbracket_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{q} \\ \llbracket C \rrbracket_{\mathsf{r}} & \text{otherwise} \end{cases}$$

$$\llbracket \mathsf{p}.f; C \rrbracket_{\mathsf{r}} = \begin{cases} f; \llbracket C \rrbracket_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{p} \\ \llbracket C \rrbracket_{\mathsf{r}} & \text{otherwise} \end{cases}$$

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

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

Figure 1: Behaviour projection for stateful choreographies.

$$\begin{aligned} C &::= I; C \mid \mathbf{0} \\ I &::= \mathsf{p}.f \,\text{->}\, \mathsf{q}.g \mid \mathsf{p}.f \mid \mathsf{if}\ \mathsf{p}.f\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2 \mid \mathbf{0} \end{aligned}$$

Figure 2: Choreographies with conditionals, syntax.

For instance, in our example with Buyer and Seller, Buyer may proceeding by deciding whether to buy the book or not depending on the price given by the Seller. In this section, we extend our framework with conditionals that allow to capture this kind of situations.

2.1 Choreographies

Syntax We extend statements in choreographies to be instructions, denoted I, which may also contain conditionals (if-then-else constructs). The new syntax is given in fig. 2.

The new term $\mathsf{if}\ \mathsf{p}.f\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2$ means "process p runs function f, and the choreography proceeds as C_1 if the result is the value `true`, or proceeds as C_2 otherwise". (So we now assume that the set of possible values contains booleans.) The condition used by this kind of terms is also typically called a *guard*—in our case, the only kind of guard that we can have for now is of the form $\mathsf{p}.f$.

$$\frac{f(\sigma(\mathsf{p}))\downarrow v \qquad g(\sigma(\mathsf{q}),v)\downarrow u}{\langle \mathsf{p}.f \mathbin{\texttt{->}} \mathsf{q}.g; C,\sigma\rangle \to \langle C, \sigma[\mathsf{q}\mapsto u]\rangle}\ \textsc{Com} \qquad \frac{f(\sigma(\mathsf{p}))\downarrow v}{\langle \mathsf{p}.f; C,\sigma\rangle \to \langle C, \sigma[\mathsf{p}\mapsto v]\rangle}\ \textsc{Local}$$

$$\frac{i = 1 \text{ if } f(\sigma(\mathsf{p}))\downarrow \texttt{true},\ i=2 \text{ otherwise}}{\langle \mathsf{if}\ \mathsf{p}.f\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2; C,\sigma\rangle \to \langle C_i; C,\sigma\rangle}\ \textsc{Cond}$$

$$\frac{C \preceq C_1 \qquad \langle C_1,\sigma\rangle \to \langle C_2,\sigma'\rangle \qquad C_2 \preceq C'}{\langle C,\sigma\rangle \to \langle C',\sigma'\rangle}\ \textsc{Struct}$$

Figure 3: Choreographies with conditionals, semantics.

Extending function procs to the new syntax is easy:

$$\begin{aligned}
\mathsf{procs}\,(I;C) &= \mathsf{procs}(I)\cup\mathsf{procs}(C)\\
\mathsf{procs}(\mathbf{0}) &= \emptyset\\
\mathsf{procs}\,(\mathsf{p}.f \mathbin{\texttt{->}} \mathsf{q}.g) &= \{\mathsf{p},\mathsf{q}\}\\
\mathsf{procs}\,(\mathsf{p}.f) &= \{\mathsf{p}\}\\
\mathsf{procs}\,(\mathsf{if}\ \mathsf{p}.f\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2) &= \{\mathsf{p}\}\cup\mathsf{procs}(C_1)\cup\mathsf{procs}(C_2)
\end{aligned}$$

.

Semantics The reduction semantics of choreographies with conditionals is given by the rules in fig. 3.

The new rule Cond formalises the intended meaning of conditionals, choosing the right branch depending on the result of the guard.

Updating structural precongruence is a bit more involved. Let us do the easy part first. Observe that an I can be $\mathbf{0}$. This is necessary for our semantics to be defined, since a conditional may contain a $\mathbf{0}$ branch. Consider the choreography $\mathsf{if}\ \mathsf{p}.\texttt{true}\ \mathsf{then}\ \mathbf{0}\ \mathsf{else}\ \mathbf{0}; C$. By rule Cond, this reduces to $\mathbf{0}; C$. That $\mathbf{0}$ is now “garbage”, in the sense that it does not specify any behaviour so we should just get rid of it. For this purpose, we introduce the following rule.

$$\frac{}{\mathbf{0}; C \preceq C}\ \textsc{GCNil}$$

Now for the more sophisticated part. Consider the following choreography.

$$(\mathsf{if}\ \mathsf{p}.f\ \mathsf{then}\ (\mathsf{if}\ \mathsf{q}.g\ \mathsf{then}\ C_{11}\ \mathsf{else}\ C_{12})\ \mathsf{else}\ (\mathsf{if}\ \mathsf{q}.g\ \mathsf{then}\ C_{21}\ \mathsf{else}\ C_{22}))\,;\mathbf{0}$$

$$\frac{\mathsf{procs}(I) \# \mathsf{procs}(I')}{I; I' \quad \equiv \quad I'; I} \ \textsc{I-I}$$

$$\frac{\mathsf{p} \notin \mathsf{procs}(I)}{I; \mathsf{if}\ \mathsf{p}.f\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2 \quad \equiv \quad \mathsf{if}\ \mathsf{p}.f\ \mathsf{then}\ (I; C_1)\ \mathsf{else}\ (I; C_2)} \ \textsc{I-Cond}$$

$$\frac{\mathsf{p} \neq \mathsf{q}}{\begin{array}{c}\mathsf{if}\ \mathsf{p}.f\ \mathsf{then}\ (\mathsf{if}\ \mathsf{q}.g\ \mathsf{then}\ C_1^1\ \mathsf{else}\ C_2^1)\ \mathsf{else}\ (\mathsf{if}\ \mathsf{q}.g\ \mathsf{then}\ C_1^2\ \mathsf{else}\ C_2^2) \\ \equiv \\ \mathsf{if}\ \mathsf{q}.g\ \mathsf{then}\ (\mathsf{if}\ \mathsf{p}.f\ \mathsf{then}\ C_1^1\ \mathsf{else}\ C_1^2)\ \mathsf{else}\ (\mathsf{if}\ \mathsf{p}.f\ \mathsf{then}\ C_2^1\ \mathsf{else}\ C_2^2)\end{array}} \ \textsc{Cond-Cond}$$

$$\frac{}{\mathbf{0}; C \preceq C} \ \textsc{GCNil}$$

Figure 4: Choreographies with conditionals, structural precongruence.

Since p and q are different, there is no causal dependency that gives an ordering in which the two conditionals to be executed! This means that we should define precongruence rules that capture out-of-order execution of conditionals, too. Another revealing example is the following.

$$\mathsf{p}.f \texttt{->} \mathsf{q}.g; \mathsf{if}\ \mathsf{r}.f'\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2; \mathbf{0}$$

Since p, q, and r are all different, it may happen that r evaluates its conditional before p and q interact. Our structural precongruence should capture this kind of out-of-order behaviour too.

The new rules for structural precongruence that follow the intuition that we have just built are displayed in fig. 4.

Rule Cond-I allows us to swap instructions inside and outside of conditionals, in case that they do not involve the process in the guard. Observe that when we bring a term inside it gets duplicated in both branches of the conditionals, since we need to ensure that it will be executed regardless of which branch is chosen at runtime.

Rule Cond-Cond allows us to swap two independent conditionals. Notice that branches C_2^1 and C_1^2 get exchanged, in order to preserve the combined effects of evaluating the two guards. We can check that this exchange makes sense with the following exercise, which verifies that swapping the two conditionals does not introduce new behaviour.

Exercise 3. *Prove the following statement.*

$$N ::= \mathsf{p} \triangleright_v B \mid N \mid N \mid \mathbf{0}$$
$$B ::= \mathsf{p}!f; B \mid \mathsf{p}?f; B \mid f; B \mid \mathsf{if}\ f\ \mathsf{then}\ B_1\ \mathsf{else}\ B_2; B \mid \mathbf{0}; B \mid \mathbf{0}$$

Figure 5: Processes with conditionals, syntax.

$$\frac{f(v) \downarrow v' \qquad g(u, v') \downarrow u'}{\mathsf{p} \triangleright_v \mathsf{q}!f; B \mid \mathsf{q} \triangleright_u \mathsf{p}?g; B' \quad \to \quad \mathsf{p} \triangleright_v B \mid \mathsf{q} \triangleright_{u'} B'}\ \textsc{Com}$$

$$\frac{f(v) \downarrow u}{\mathsf{p} \triangleright_v f; B \ \to\ \mathsf{p} \triangleright_u B}\ \textsc{Local}$$

$$\frac{i = 1 \text{ if } f(v) \downarrow \texttt{true},\ i = 2 \text{ otherwise}}{\mathsf{p} \triangleright_v (\mathsf{if}\ f\ \mathsf{then}\ B_1\ \mathsf{else}\ B_2)\,; B \to \mathsf{p} \triangleright_v B_i; B}\ \textsc{Cond}$$

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

Figure 6: Processes with conditionals, semantics.

Let σ be a global memory state. We have the following reduction chain without using rule Struct *for some j and i in $\{1, 2\}$*

$$\left\langle \mathsf{if}\ \mathsf{p}.f\ \mathsf{then}\ \left(\mathsf{if}\ \mathsf{q}.g\ \mathsf{then}\ C_1^1\ \mathsf{else}\ C_2^1\right)\ \mathsf{else}\ \left(\mathsf{if}\ \mathsf{q}.g\ \mathsf{then}\ C_1^2\ \mathsf{else}\ C_2^2\right), \sigma \right\rangle \to\!\!\to \left\langle C_i^j, \sigma \right\rangle$$

if and only if we have also the following reduction chain

$$\left\langle \mathsf{if}\ \mathsf{q}.g\ \mathsf{then}\ \left(\mathsf{if}\ \mathsf{p}.f\ \mathsf{then}\ C_1^1\ \mathsf{else}\ C_1^2\right)\ \mathsf{else}\ \left(\mathsf{if}\ \mathsf{p}.f\ \mathsf{then}\ C_2^1\ \mathsf{else}\ C_2^2\right), \sigma \right\rangle \to\!\!\to \left\langle C_i^j, \sigma \right\rangle$$

.

Suggestion: proceed by cases on $f(\sigma(\mathsf{p})) \downarrow v$ (what can v be?) and $g(\sigma(\mathsf{q})) \downarrow u$ (what can u be?) and their consequences on the reduction chains.

2.2 Processes

Introducing conditionals to our process calculus is straightforward, and follows the same principles as for choreographies. The new syntax and semantics are given by the rules in figs. 5 to 7.

$$\frac{}{(N_1 \mid N_2) \mid N_3 \;\equiv\; N_1 \mid (N_2 \mid N_3)}\ \text{PA} \qquad \frac{}{\mathbf{0}; B \preceq B}\ \text{GCB}$$

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

Figure 7: Processes with conditionals, structural precongruence.

2.3 EndPoint Projection

Adding conditionals to choreographies has intriguing consequences for EPP. Therefore, before we dive into a general definition, it is useful to look at an example. Consider the following choreography.

$$C_{\mathsf{unproj}} \triangleq \;(\mathsf{if}\ \mathsf{p}.f\ \mathsf{then}\ \mathsf{p}.\mathtt{true} \to \mathsf{q}.x; \mathbf{0}\ \mathsf{else}\ \mathbf{0})\,; \mathbf{0}$$

If p chooses the left branch in the conditional, then it sends the value $\mathtt{true}$ to q. Otherwise, the choreography terminates. What should the EPP of C_{unproj} look like? It seems obvious that the resulting network should consist of two processes, p and q, so for any σ we get:

$$[\![\langle C_{\mathsf{unproj}}, \sigma\rangle]\!] = \;\mathsf{p} \triangleright_{\sigma(\mathsf{p})} [\![C_{\mathsf{unproj}}]\!]_{\mathsf{p}} \mid \mathsf{q} \triangleright_{\sigma(\mathsf{q})} [\![C_{\mathsf{unproj}}]\!]_{\mathsf{q}} \qquad (1)$$

. The projection for p seems easy to achieve:

$$[\![C_{\mathsf{unproj}}]\!]_{\mathsf{p}} = \;(\mathsf{if}\ f\ \mathsf{then}\ \mathsf{q}!\mathtt{true}; \mathbf{0}\ \mathsf{else}\ \mathbf{0})\,; \mathbf{0}$$

. Instead, we run into trouble for projecting q. Process q is not involved in evaluating the conditional (since that is local at p), so its projection should just “skip” it. Indeed the only piece of code in the choreography that involves q is $\mathsf{p}.\mathtt{true} \to \mathsf{q}.x$. If we choose to project that (and we should, since the choreography contains it), we obtain:

$$[\![C_{\mathsf{unproj}}]\!]_{\mathsf{q}} = \;\mathsf{p}?x; \mathbf{0}; \mathbf{0} \qquad (2)$$

. If, instead, we choose *not to* project the receive action by q, we obtain:

$$[\![C_{\mathsf{unproj}}]\!]_{\mathsf{q}} = \;\mathbf{0}; \mathbf{0} \qquad (3)$$

. Neither of these decisions gives us a correct EPP, as we can check with two exercises.

Exercise 4. *Show that, if we adopt the behaviour projection in eq.* (2)*, the network in eq.* (1) *may reduce to a network that is not the EPP of what* C_{unproj} *reduces to.*

Hint: consider the case of $\langle C_{\mathsf{unproj}}, \sigma\rangle$ *for some* σ *such that* $f(\sigma(\mathsf{p})) \downarrow \mathtt{false}$.

Exercise 5. *Show that, if we adopt the behaviour projection in eq.* (3)*, the network in eq.* (1) *may reduce to a network that is not the EPP of what* C_{unproj} *reduces to.*

Hint: consider the case of $\langle C_{\mathsf{unproj}}, \sigma \rangle$ *for some* σ *such that* $f(\sigma(\mathsf{p})) \downarrow \mathtt{true}$*.*

Exercise 6. *Think about how you would define the behaviour projection for a conditional in a choreography. We are going to delve into this problem in the next part.*