Lecture Notes on Choreographies, Part 4

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 Introduction to realisability

Let us revisit the problematic example from the last part.

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

Observe that the choreography states that p and q have to behave differently depending on the branch chosen by the conditional. For p, this is not a problem, because p is the process making the choice and thus "knows" whether we should run the then branch or the else branch. However, q does not have any information that it can use to determine which branch has been chosen. Should then q wait for a message from p (then branch) or simply terminate (else branch)? The problem is thus that the choreography does not specify a sufficient *flow of information* about choices among processes. In the literature, a choreography such as this is said to be *unrealisable*, or *unprojectable*, in the sense that if we project it naïvely as we attempted to do in part 3 we obtain an incorrect (network) implementation. Mentions of this kind of properties (sometimes with different terminology) were already present in the early days of formal methods for choreographies [Fu et al., 2005; Carbone et al., 2007; Qiu et al., 2007; Lanese et al., 2008]. In more expressive choreography models, it is not only conditionals that can make a choreography unprojectable, as we will see later on. For now, let us focus on this particular construct.

There are different and useful theoretical tools that can be adopted to deal with unrealisability.

Detection First of all, we can develop mechanical methods to detect whether a choreography is realisable. Then, we can add realisability as a necessary assumption for the definition and/or the correctness of EPP, i.e., we guarantee that EPP is defined and/or correct only if the choreography given as input is realisable.

Amendment Given an unrealisable choreography due to some insufficient communication flow, we can attempt at automatically fixing the flow by adding extra communications to the choreography.

Smart Projection We can design EPP such that it can also project unrealisable choreographies, by adding extra communications in the generated network that are not defined in the original choreography.

We focus on detection in these notes, and leave amendment and smart projections to later.

2 EPP with detection

We define EPP for choreographies with conditionals, including a requirement that detects unrealisable choreographies. The definition of EPP is the same as the last one, save that we need to add the case for conditionals to behaviour projection, in fig. 1.

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

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

.

The rule for projecting a choreographic conditional includes a check that prevents projecting problematic choreographies. Namely, when we are projecting a conditional $\mathsf{if}\ \mathsf{p}.f\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2$, we just proceed homomorphically if we are projecting the process that evaluates the guard (p)—by homomorphically, we mean that we follow the structure of the choreography and project a corresponding conditional with the same structure. If, instead, we are projecting some other process, we know that this process will not know the choice that p will make between the two branches C_1 and C_2. Thus we require that the behaviour of this uninformed process is *the same* ($[\![C_1]\!]_{\mathsf{r}} = [\![C_2]\!]_{\mathsf{r}}$ in the rule).

$$
\begin{aligned}
[\![\mathsf{p}.f \mathrel{\text{->}} \mathsf{q}.g; C]\!]_{\mathsf{r}} &= \begin{cases} \mathsf{q}!f; [\![C]\!]_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{p} \\ \mathsf{p}?g; [\![C]\!]_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{q} \\ [\![C]\!]_{\mathsf{r}} & \text{otherwise} \end{cases} \\
[\![\mathsf{p}.f; C]\!]_{\mathsf{r}} &= \begin{cases} f; [\![C]\!]_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{p} \\ [\![C]\!]_{\mathsf{r}} & \text{otherwise} \end{cases} \\
[\![\mathsf{if}\ \mathsf{p}.f\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2; C]\!]_{\mathsf{r}} &= \begin{cases} (\mathsf{if}\ f\ \mathsf{then}\ [\![C_1]\!]_{\mathsf{r}}\ \mathsf{else}\ [\![C_2]\!]_{\mathsf{r}})\,; [\![C]\!]_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{p} \\ [\![C_1]\!]_{\mathsf{r}}\,; [\![C]\!]_{\mathsf{r}} \quad \text{if } \mathsf{r} \neq \mathsf{p} \text{ and } [\![C_1]\!]_{\mathsf{r}} = [\![C_2]\!]_{\mathsf{r}} \end{cases} \\
[\![\mathbf{0}; C]\!]_{\mathsf{p}} &= [\![C]\!]_{\mathsf{p}} \\
[\![\mathbf{0}]\!]_{\mathsf{p}} &= \mathbf{0}
\end{aligned}
$$

Figure 1: Behaviour projection for choreographies with conditionals.

So far, all definitions of EPP that we have seen were complete—as in a complete function, in the sense that EPP was defined for all possible choreographies. The check performed by the rule for projecting conditionals makes EPP a partial function instead, since we now have choreographies that may not respect our check. The unrealisable choreography from section 1 is an example of a choreography for which EPP is undefined. We recall its definition here:

$$C_{\mathsf{unproj}} \triangleq (\mathsf{if}\ \mathsf{p}.f\ \mathsf{then}\ \mathsf{p}.\mathtt{true} \mathrel{\text{->}} \mathsf{q}.x; \mathbf{0}\ \mathsf{else}\ \mathbf{0})\,; \mathbf{0} \tag{1}$$

. Then we can prove that it cannot be projected.

Proposition 1. *Let C_{unproj} be the choreography in eq. (1). For all σ, $[\![\langle C_{\mathsf{unproj}}, \sigma\rangle]\!]$ is undefined.*

Proof. For $[\![\langle C_{\mathsf{unproj}}, \sigma\rangle]\!]$ to be defined, we need $[\![C_{\mathsf{unproj}}]\!]_{\mathsf{q}}$ to be defined (by definition 1). Since C_{unproj} is a conditional, the only rule that we can apply for $[\![C_{\mathsf{unproj}}]\!]_{\mathsf{q}}$ is the third one in fig. 1. We proceed by cases according to the definition of the rule. Since $\mathsf{p} \neq \mathsf{q}$, we cannot apply the first case. The only remaining option is the second one, which requires the following.

- $\mathsf{p} \neq \mathsf{q}$. This holds.
- $[\![\mathsf{p}.\mathtt{true} \mathrel{\text{->}} \mathsf{q}.x; \mathbf{0}]\!]_{\mathsf{q}} = [\![\mathbf{0}]\!]_{\mathsf{q}}$. This does not hold, because $[\![\mathsf{p}.\mathtt{true} \mathrel{\text{->}} \mathsf{q}.x; \mathbf{0}]\!]_{\mathsf{q}} = \mathsf{p}?x; \mathbf{0}$ which is not equal to $[\![\mathbf{0}]\!]_{\mathsf{q}} = \mathbf{0}$.

Thus, $[\![C_{\mathsf{unproj}}]\!]_{\mathsf{q}}$ is undefined and, consequently, the thesis follows. □

When the EPP of a choreography is not defined, we say that the choreography is unprojectable.

Observe that the requirement on conditionals may seem strict, but we can still write quite meaningful choreographies. Here is a modified version of C_{unproj} that is projectable.

$$(\mathsf{if}\ \mathsf{p}.f\ \mathsf{then}\ \mathsf{p}.\mathtt{true} \mathrel{\text{->}} \mathsf{q}.x; \mathbf{0}\ \mathsf{else}\ \mathsf{p}.\mathtt{false} \mathrel{\text{->}} \mathsf{q}.x; \mathbf{0})\ ; \mathbf{0} \tag{2}$$

Exercise 1. *Write the EPP of the choreography in eq.* (2)*.*

Intuitively, the choreography in eq. (2) is projectable because q has the same behaviour in both branches of the conditional, i.e., a receive action on variable x. Observe that p, however, can send different values to q (`true` and `false` respectively, in our example), since p knows which branch is chosen. Now that q has this information, it can evaluate it internally with a conditional to know which branch we are in. Then, we could have that q does something different depending on this. For example, we may wish that q sends some money to p when p sends `true`, and no money when p sends `false`. We do this in the following choreography. As usual, we assume that $\mathsf{q}.x$ returns the value stored in x at q. The symbol 0 in the choreography below is just the natural number 0, representing no money.

$$\left(\begin{array}{rl} \mathsf{if}\ \mathsf{p}.f\ \mathsf{then} & \mathsf{p}.\mathtt{true} \mathrel{\text{->}} \mathsf{q}.x; \\ & \left(\begin{array}{rl} \mathsf{if}\ \mathsf{q}.x\ \mathsf{then} & \mathsf{q}.\mathit{money} \mathrel{\text{->}} \mathsf{p}.m; \mathbf{0} \\ \mathsf{else} & \mathsf{q}.0 \mathrel{\text{->}} \mathsf{p}.m; \mathbf{0} \end{array}\right); \mathbf{0} \\ \mathsf{else} & \mathsf{p}.\mathtt{false} \mathrel{\text{->}} \mathsf{q}.x; \\ & \left(\begin{array}{rl} \mathsf{if}\ \mathsf{q}.x\ \mathsf{then} & \mathsf{q}.\mathit{money} \mathrel{\text{->}} \mathsf{p}.m; \mathbf{0} \\ \mathsf{else} & \mathsf{q}.0 \mathrel{\text{->}} \mathsf{p}.m; \mathbf{0} \end{array}\right); \mathbf{0} \end{array}\right); \mathbf{0} \tag{3}$$

Exercise 2. *Write the behaviour projection of* q *for the choreography in eq.* (3)*.*

A few remarks There are two problems that become pretty evident when looking at choreographies such as that in eq. (3). First, the choreography is repetitive and tedious to write, because we have to copy-paste exactly the same code for q in both branches, to respect our condition for projecting conditionals. So it is much "bigger" than we would wish for. Second, we now have a problem with p inside of the conditional evaluated by q. Namely, since p does not know which branch q chooses, we had to insert a communication of 0 from q to p to represent the act of giving no money. This seems silly: from the choreography, we know that if p sends `false`, then p is not due any money. So sending a 0 from q to p is a waste of a communication: it

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

Figure 2: Choreographies with selections, syntax.

would be better if we could write a choreography where q simply does not send anything when no money is due.

Summing up our considerations, we would like to be able to write (and project!) a choreography that looks like the following, which is a direct formalisation of what we would like to happen: if p wants some money, then q sends it to p, otherwise nothing happens.

$$(\mathsf{if}\ \mathsf{p}.f\ \mathsf{then}\ \mathsf{q}.\mathit{money} \,\texttt{->}\, \mathsf{p}.m; \mathbf{0}\ \mathsf{else}\ \mathbf{0})\,; \mathbf{0} \tag{4}$$

Of course, the choreography in eq. (4) is unprojectable according to our rules so far. Our aim in the next section is to develop a choreography model that allows us to write things that are nearly as concise as this choreography—meaning that we avoid the two problems mentioned above—and are also projectable!

3 Selections

We add a new primitive to our choreography model, which can be used to explicitly (and efficiently) propagate information among processes about which branches have been chosen in conditionals.

3.1 Choreographies

Syntax The updated syntax of choreographies is given in fig. 2. The new primitive is $\mathsf{p} \,\texttt{->}\, \mathsf{q}[l]$, read "process p sends to process q the selection of label l". Labels, ranged over by l, are picked from an infinite set of label names. Intuitively, when we write a selection $\mathsf{p} \,\texttt{->}\, \mathsf{q}[l]$, p is *selecting* one of the behaviours (l in this case) that q offers. In programming languages, we can think of labels as abstractions of method names in object-oriented programming, or operations in service-oriented computing.

Example Before we dive into the formal details of selections, let us see how they can help us with our example in eq. (4), where p needs to inform

q of whether we chose the left or the right branch of the conditional, such that q can behave accordingly. We can choose two labels, say PAY and NO, and imagine that q should offer p a choice between the two behaviours in the respective branches of the conditional. When q receives a selection for PAY, then q knows that it should behave as specified in the left branch of the conditional and pay some money—$\mathsf{q}.\mathit{money} \mathsf{\text{->}}\, \mathsf{p}.m; \mathbf{0}$. Instead, when q receives a selection for NO, then it knows that it should behave as specified in the right branch and hence do nothing—$\mathbf{0}$. We can formalise this intuition as the following choreography.

$$\Big(\ \mathsf{if}\ \mathsf{p}.f\ \mathsf{then}\ \ \mathsf{p} \text{->} \mathsf{q}[\textsc{pay}]; \mathsf{q}.\mathit{money} \text{->} \mathsf{p}.m; \mathbf{0}\ \ \mathsf{else}\ \ \mathsf{p} \text{->} \mathsf{q}[\textsc{no}]; \mathbf{0}\ \ \Big)\,; \mathbf{0} \tag{5}$$

In the choreography in eq. (5), p makes a choice as before (evaluating the conditional). Differently from before, however, right after making this choice p now communicates a label to q. In the left branch, q receives label PAY. In the right branch, q receives label NO. We then assume that, thanks to the fact that q receives a *different* label for the two branches, it is able to use this information to know how to behave in the two branches.

Semantics The semantics of selections is straightforward, since they do not change the memory of any process. We just need to add the following rule.

$$\frac{}{\langle \mathsf{p} \text{->} \mathsf{q}[l]; C, \sigma\rangle \to \langle C, \sigma\rangle}\ \textsc{Sel}$$

The complete set of rules that we obtain for choreographies with selections is displayed in fig. 3.

The rules defining structural precongruence are the same (see fig. 4), but since now an I can be a selection term—$\mathsf{p} \text{->} \mathsf{q}[l]$—we need to update the definition of procs as follows.

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

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

$$\frac{f(\sigma(\mathsf{p}))\downarrow v}{\langle \mathsf{p}.f; C, \sigma\rangle \to \langle C, \sigma[\mathsf{p} \mapsto v]\rangle}\ \textsc{Local}$$

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

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

Figure 3: Choreographies with selections, semantics.

3.2 Processes

Regarding the process model, we need to add two primitives: one for sending selections, and one for receiving them. The updated syntax and semantics are given by the rules in figs. 5 to 7.

The new primitives are $\mathsf{p} \oplus l; B$ and $\mathsf{p}\,\&\{l_i : B_i\}_{i\in I}; B$. A term $\mathsf{p} \oplus l; B$ sends the choice of a label l to p and then proceeds as B. Dually, a term $\mathsf{p}\,\&\{l_i : B_i\}_{i\in I}; B$ (also called *branching term*, or simply a branching) offers the possibility to choose from many behaviours $\{B_i\}_{i\in I}$ for some finite set I. When a branching term receives a label, it runs the behaviour that the label is associated to. For example, we read $\mathsf{p}\,\&\{l_1 : B_1, l_2 : B_2\}; B$ as "receive a label, and then run $B_1; B$ if the received label is l_1, or run $B_2; B$ if the received label is l_2". This intuition is formalised by rule Sel in fig. 6, where the sender selects a label among those offered by the receiver ($j \in I$).

3.3 EPP for choreographies with selections

Consider again the choreography in eq. (5), which is again given below, for convenience.

$$\big(\ \mathsf{if}\ \mathsf{p}.f\ \mathsf{then}\ \ \mathsf{p} \text{ -> } \mathsf{q}[\textsc{pay}]; \mathsf{q}.\mathit{money} \text{ -> } \mathsf{p}.m; \mathbf{0}\ \ \mathsf{else}\ \ \mathsf{p} \text{ -> } \mathsf{q}[\textsc{no}]; \mathbf{0}\ \ \big); \mathbf{0}$$

Given any σ, we can now manually write an operationally-equivalent net-

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

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

$$\frac{\mathsf{p} \notin \mathsf{procs}(I) \qquad I \neq \mathbf{0}}{\mathsf{if}\ \mathsf{p}.f\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2; I \quad \equiv \quad \mathsf{if}\ \mathsf{p}.f\ \mathsf{then}\ (C_1; I)\ \mathsf{else}\ (C_2; I)} \textsc{ Cond}\text{-I}$$

$$\frac{}{\mathbf{0}; C \preceq C} \text{ GC}\textsc{Nil}$$

Figure 4: Choreographies with selections, structural precongruence.

$$\begin{aligned} N &::= \mathsf{p} \triangleright_v B \mid N \mid N \mid \mathbf{0} \\ B &::= \mathsf{p}!f; B \mid \mathsf{p}?f; B \mid \mathsf{p} \oplus l; B \mid \mathsf{p}\,\&\{l_i : B_i\}_{i \in I}; B \\ &\quad \mid \mathsf{if}\ f\ \mathsf{then}\ B_1\ \mathsf{else}\ B_2; B \mid \mathbf{0}; B \mid \mathbf{0} \end{aligned}$$

Figure 5: Processes with selections, syntax.

work, as follows.

$$\begin{array}{ll} \mathsf{p} \triangleright_{\sigma(\mathsf{p})} & \mathsf{if}\ f\ \mathsf{then}\ \mathsf{q} \oplus \textsc{pay}; \mathsf{q}?m; \mathbf{0}\ \mathsf{else}\ \mathsf{q} \oplus \textsc{no}; \mathbf{0} \\ \mid & \\ \mathsf{q} \triangleright_{\sigma(\mathsf{q})} & \mathsf{p}\,\&\{\textsc{pay} : \mathsf{p}!\mathit{money}; \mathbf{0},\ \textsc{no} : \mathbf{0}\}; \mathbf{0} \end{array} \tag{6}$$

Exercise 3. *Write the reduction chains of the choreography in eq.* (5) *and the network in eq.* (6). *Do they mimic each other?*

Tuning EPP to mechanically produce this kind of results requires adding a rule for projecting selections and updating the rule for projecting conditionals as follows.

$$[\![\mathsf{p} \text{->} \mathsf{q}[l]; C]\!]_{\mathsf{r}} = \begin{cases} \mathsf{q} \oplus l; [\![C]\!]_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{p} \\ \mathsf{p}\,\&\{l : [\![C]\!]_{\mathsf{r}}\}; \mathbf{0} & \text{if } \mathsf{r} = \mathsf{q} \\ [\![C]\!]_{\mathsf{r}} & \text{otherwise} \end{cases}$$

$$[\![\mathsf{if}\ \mathsf{p}.f\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2; C]\!]_{\mathsf{r}} = \begin{cases} (\mathsf{if}\ f\ \mathsf{then}\ [\![C_1]\!]_{\mathsf{r}}\ \mathsf{else}\ [\![C_2]\!]_{\mathsf{r}}) ; [\![C]\!]_{\mathsf{r}} & \text{if } \mathsf{r} = \mathsf{p} \\ ([\![C_1]\!]_{\mathsf{r}} \sqcup [\![C_2]\!]_{\mathsf{r}}) ; [\![C]\!]_{\mathsf{r}} & \text{otherwise} \end{cases}$$

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

$$\frac{j\in I}{\mathsf{p}\triangleright_v \mathsf{q}\oplus l_j;B \ \mid\ \mathsf{q}\triangleright_u \mathsf{p}\&\{l_i : B_i\}_{i\in I};B' \quad\to\quad \mathsf{p}\triangleright_v B \ \mid\ \mathsf{q}\triangleright_u B_j;B'}\ \textsc{Sel}$$

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

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

Figure 6: Processes with selections, semantics.

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

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

Figure 7: Processes with selections, structural precongruence.

$$
\begin{array}{rcl}
\mathsf{p}!f; B_1 \sqcup \mathsf{p}!f; B_2 & = & \mathsf{p}!f; (B_1 \sqcup B_2) \\
\mathsf{p}?f; B_1 \sqcup \mathsf{p}?f; B_2 & = & \mathsf{p}?f; (B_1 \sqcup B_2) \\
\mathsf{p} \oplus l; B_1 \sqcup \mathsf{p} \oplus l; B_2 & = & \mathsf{p} \oplus l; (B_1 \sqcup B_2) \\
\begin{array}{c} \mathsf{if}\ f\ \mathsf{then}\ B_1\ \mathsf{else}\ B_1'; B_1'' \\ \sqcup \\ \mathsf{if}\ f\ \mathsf{then}\ B_2\ \mathsf{else}\ B_2'; B_2'' \end{array} & = & \mathsf{if}\ f\ \mathsf{then}\ (B_1 \sqcup B_2)\ \mathsf{else}\ (B_1' \sqcup B_2')\,; (B_1'' \sqcup B_2'') \\
\mathbf{0}; B_1 \sqcup \mathbf{0}; B_2 & = & \mathbf{0}; (B_1 \sqcup B_2) \\
\mathbf{0} \sqcup \mathbf{0} & = & \mathbf{0} \\
\mathsf{p}\,\&\{l_i : B_i\}_{i \in I}; B_1 \sqcup \mathsf{p}\,\&\{l_j : B_j'\}_{j \in J}; B_2 & = & \\
\multicolumn{3}{r}{\mathsf{p}\,\&\left(\ \{l_k : (B_k \sqcup B_k')\}_{k \in I \cap J} \cup \{l_i : B_i\}_{i \in I \setminus J} \cup \{l_j : B_j'\}_{j \in J \setminus I}\ \right); (B_1 \sqcup B_2)}
\end{array}
$$

Figure 8: Merging operator for processes with selections.

The projection of a selection $\mathsf{p} \rightarrow \mathsf{q}[l]$ is simple: the sender is projected to the sending of the label—$\mathsf{q} \oplus l$—whereas the receiver is projected to a branching with a single branch with label l—$\mathsf{p}\,\&\{l : [\![C]\!]_{\mathsf{r}}\}$. Observe that we do not require equality of projections for the processes that do not evaluate the conditional. Instead, we have a new ingredient, the *merge operator* $\sqcup$. This is a partial operator on behaviours—meaning that it is not always defined—so EPP is still partial. However, it is now much more expressive, since we can define $\sqcup$ to take advantage of selections. Formally, $B_1 \sqcup B_2$ is defined inductively on the structure of B_1 and B_2. The rules defining $\sqcup$ are displayed in fig. 8. These are an adaptation to our model from the definition originally given by Carbone et al. [2007].

Merging proceeds homomorphically, requiring the two behaviours to be merged to have the same structure. For all terms but branchings, we require the identity (the two merged terms are the same). For branchings (last rule), we apply the following reasoning: all branches with the same label ($k \in I \cap J$) are merged, whereas branches with different labels are simply added to the result with no requirements ($i \in I \setminus J$ and $j \in J \setminus I$).

Example 1. *The network in eq.* (6) *is the EPP of the choreography in eq.* (5).

Exercise 4. *Write the behaviour projection for process* q *in the following choreography.*

$$
\left(\begin{array}{rl}
\mathsf{if}\ \mathsf{p}.f\ \mathsf{then} & \mathsf{p} \rightarrow \mathsf{q}[\textsc{pay}]; \mathsf{q}.\mathit{money} \rightarrow \mathsf{p}.m; \mathbf{0} \\
\mathsf{else} & \\
 & \left(\begin{array}{rl} \mathsf{if}\ \mathsf{p}.g\ \mathsf{then} & \mathsf{p} \rightarrow \mathsf{q}[\textsc{reimburse}]; \mathsf{p}.\mathit{money} \rightarrow \mathsf{q}.m; \mathbf{0} \\ \mathsf{else} & \mathsf{p} \rightarrow \mathsf{q}[\textsc{no}]; \mathbf{0} \end{array}\right); \mathbf{0}
\end{array}\right); \mathbf{0}
$$

3.4 Operational correspondence for EPP with selections

Selections make stating the operational correspondence theorem for EPP trickier. Consider the following choreography.

$$C_{\mathsf{cond}} \triangleq (\mathsf{if}\ \mathsf{p}.f\ \mathsf{then}\ \mathsf{p} \mathbin{\texttt{->}} \mathsf{q}[\textsc{left}]; \mathbf{0}\ \mathsf{else}\ \mathsf{p} \mathbin{\texttt{->}} \mathsf{q}[\textsc{right}]; \mathbf{0})\ ; \mathbf{0}$$

Its projection is the following, for any σ.

$$[\![\langle C_{\mathsf{cond}}, \sigma\rangle]\!] = \begin{array}{l} \mathsf{p} \triangleright_{\sigma(\mathsf{p})} (\mathsf{if}\ f\ \mathsf{then}\ \mathsf{q} \oplus \textsc{left}; \mathbf{0}\ \mathsf{else}\ \mathsf{q} \oplus \textsc{right}; \mathbf{0})\ ; \mathbf{0} \\ \mid \\ \mathsf{q} \triangleright_{\sigma(\mathsf{q})} \mathsf{p} \,\&\{\textsc{left} : \mathbf{0}, \textsc{right} : \mathbf{0}\}; \mathbf{0}; \mathbf{0} \end{array}$$

Let σ be such that $f(\sigma(\mathsf{p}))\downarrow \texttt{true}$ (a similar reasoning to the one that follows holds for the case in which $f(\sigma(\mathsf{p}))$ does not evaluate to $\texttt{true}$). Then, by rule COND for choreographies we can execute the conditional at p and obtain the following reduction:

$$\langle C_{\mathsf{cond}}, \sigma\rangle \to \langle \mathsf{p} \mathbin{\texttt{->}} \mathsf{q}[\textsc{left}]; \mathbf{0}; \mathbf{0}, \sigma\rangle \qquad (7)$$

. The corresponding reduction in $[\![\langle C_{\mathsf{cond}}, \sigma\rangle]\!]$, of course, should execute the same conditional at p. This yields the following reduction:

$$[\![\langle C_{\mathsf{cond}}, \sigma\rangle]\!] \to \begin{array}{l} \mathsf{p} \triangleright_{\sigma(\mathsf{p})} \mathsf{q} \oplus \textsc{left}; \mathbf{0}; \mathbf{0} \\ \mid \\ \mathsf{q} \triangleright_{\sigma(\mathsf{q})} \mathsf{p} \,\&\{\textsc{left} : \mathbf{0}, \textsc{right} : \mathbf{0}\}; \mathbf{0}; \mathbf{0} \end{array} \qquad (8)$$

. Let C_{left} be the choreography on the right-hand side in eq. (7) (the choreography after the reduction) and N_{left} be the network on the right-hand side in eq. (8) (the network after the reduction). Ideally, following our previous statements of operational correspondence, we would expect that $N_{\mathsf{left}} \preceq [\![\langle C_{\mathsf{left}}, \sigma\rangle]\!]$. But this is not the case, since we can see that the EPP of C_{left} is different:

$$[\![\langle C_{\mathsf{left}}, \sigma\rangle]\!] = [\![\langle \mathsf{p} \mathbin{\texttt{->}} \mathsf{q}[\textsc{left}]; \mathbf{0}; \mathbf{0}, \sigma\rangle]\!] = \begin{array}{l} \mathsf{p} \triangleright_{\sigma(\mathsf{p})} \mathsf{q} \oplus \textsc{left}; \mathbf{0} \\ \mid \\ \mathsf{q} \triangleright_{\sigma(\mathsf{q})} \mathsf{p} \,\&\{\textsc{left} : \mathbf{0}\}; \mathbf{0} \end{array}$$

. Notice that the difference is in the branching term at process q: the projection of C_{left} has only the LEFT branch, whereas in N_{left} we still have both the LEFT and RIGHT branches. Generally speaking, this happens because when a conditional like $\mathsf{if}\ \mathsf{p}.f\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2$ in a choreography is reduced, we "cut off" either C_1 or C_2 in a single step and they may contain code that involves many processes, not just p. Instead, in the process calculus, executing a conditional at process p removes code for p only, so the other processes may still have extra branches in their branching terms (as in our example here).

We now move to formalising our observations in a general way, obtaining a new statement for operational correspondence. First, we define formally what it means to have a network with "extra branches".

Definition 2. *We write $N \sqsupseteq N'$ when N has at least as many branches in branchings as N'. Formally, $\sqsupseteq$ is defined inductively as follows.*

- $\mathbf{0} \sqsupseteq \mathbf{0}$*;*
- $\mathsf{p} \triangleright_v B \sqsupseteq \mathsf{p} \triangleright_v B'$ *if* $B \sqcup B' = B$*;*
- $N_1 \mid N_2 \sqsupseteq N_1' \mid N_2'$ *if* $N_1 \sqsupseteq N_1'$ *and* $N_2 \sqsupseteq N_2'$*.*

Then, we can reformulate operational correspondence appropriately.

Theorem 1 (Operational Correspondence)**.** *Let* $[\![\langle C, \sigma\rangle]\!] = N$*. Then,*

Completeness *If* $\langle C, \sigma\rangle \to \langle C', \sigma'\rangle$ *for some* C' *and* σ'*, then there exists* N' *such that* $N \to N'$ *and* $N' \preceq \sqsupseteq [\![\langle C', \sigma'\rangle]\!]$*.*

Soundness *If* $N \to N'$ *for some* N'*, then there exists* C' *and* σ' *such that* $\langle C, \sigma\rangle \to \langle C', \sigma'\rangle$ *and* $N' \preceq \sqsupseteq [\![\langle C', \sigma'\rangle]\!]$*.*

Exercise 5 (!)**.** *Prove that relation $\sqsupseteq$ is a preorder. We recall what this means in the following items.*

- *Reflexivity:* $N \sqsupseteq N$ *for all* N*.*
- *Transitivity:* $N \sqsupseteq N'$ *and* $N' \sqsupseteq N''$ *imply* $N \sqsupseteq N''$ *for all* N*,* N'*, and* N''*.*

Exercise 6 (!)**.** *Prove theorem 1.*

References

M. Carbone, K. Honda, and N. Yoshida. Structured communication-centred programming for web services. In R. D. Nicola, editor, *ESOP*, volume 4421 of *LNCS*, pages 2–17. Springer, 2007. doi: 10.1007/978-3-540-71316-6_2.

X. Fu, T. Bultan, and J. Su. Realizability of conversation protocols with message contents. *Int. J. Web Service Res.*, 2(4):68–93, 2005.

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

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