Choreographies in Coq

Luís Cruz-Filipe, Fabrizio Montesi, and Marco Peressotti[*]

Department of Mathematics and Computer Science, University of Southern Denmark
{lcf,fmontesi,peressotti}@imada.sdu.dk

Choreographic Programming is a paradigm for specifying concurrent systems based on message-passing where communications are written in an Alice-to-Bob notation. Every program (choreography) can then be mechanically translated into a distributed process-calculus implementation that is guaranteed to be bisimilar to the original choreography. Thanks to this methodology, such implementations are guaranteed never to suffer from mismatched communications. More generally, they cannot reach a deadlock state, since the original choreography language does not allow deadlocks.

Example 1. *The following choreography models a scenario where Alice (*a*) buys a book from a seller (*s*) routing the payment through her bank (*b*).*

$$\begin{aligned}&\mathsf{a}.\mathit{title} \to \mathsf{s};\ \mathsf{s}.\mathit{price} \to \mathsf{a};\ \mathsf{s}.\mathit{price} \to \mathsf{b};\\ &\mathit{if}\ \mathsf{b}.\mathit{ok}\ \mathit{then}\ \mathsf{b} \to \mathsf{s}[\mathit{ok}];\ \mathsf{b} \to \mathsf{a}[\mathit{ok}];\ \mathsf{s}.\mathit{book} \to \mathsf{a}\\ &\qquad\quad \mathit{else}\ \mathsf{b} \to \mathsf{s}[\mathit{ko}];\ \mathsf{b} \to \mathsf{a}[\mathit{ko}]\end{aligned}$$

First, Alice sends the title of the book to the seller, which then quotes the price to both Alice and the bank. If the bank confirms the transaction, it sends an acknowledgement to both Alice and the seller, and the latter proceeds to send the book. Otherwise, the bank sends a cancellation to both parties.

The hallmark of Choreographic Programming is the EPP Theorem, which guarantees a precise operational correspondence between a choreography and its generated implementation (EndPoint Projection).

Example 2. *The previous choreography can be projected into the following distributed protocol.*

$$\begin{aligned}\mathsf{a}\ &\triangleright\ \mathsf{s}!\mathit{title};\ \mathsf{s}?;\ \mathsf{b}\&\{\mathit{ok}:\ \mathsf{s}?\mid \mathit{ko}:\}\\ \mathsf{b}\ &\triangleright\ \mathsf{s}?;\ \mathit{if}\ \mathit{ok}\ \mathit{then}\ (\mathsf{s}[\mathit{ok}];\ \mathsf{a}[\mathit{ok}])\ \mathit{else}\ (\mathsf{s}[\mathit{ko}];\ \mathsf{a}[\mathit{ko}])\\ \mathsf{s}\ &\triangleright\ \mathsf{a}?;\ \mathsf{a}!\mathit{price};\ \mathsf{b}!\mathit{price};\ \mathsf{b}\&\{\mathit{ok}:\ \mathsf{a}!\mathit{book}\mid \mathit{ko}:\}\end{aligned}$$

The protocol for Alice is thus: send a title to the seller and wait for a reply; then wait for either confirmation from the bank, in which case the seller will send the book, or cancellation, in which case the protocol ends. The protocol for the seller is similar. In turn, the bank initially waits for a message from the seller, and then decides whether to send confirmation or cancellation to both the seller and Alice.

In the examples above, the participants exchange two kinds of messages: data messages (e.g. $\mathsf{a}.\mathit{title} \to \mathsf{s}$) or signals (e.g. $\mathsf{b} \to \mathsf{s}[\mathit{ok}]$), which are exclusively meant to dictate control flow. The need for this distinction has to do with propagating local choices, in our example the decision by the bank on whether to authorize payment or not.

The EPP Theorem guarantees that the choregraphy in Example 1 behaves exactly as the three communicating processes in Example 2. However, the proof of this theorem even for

[*]This work was partially supported by the Independent Research Fund Denmark, grant no. DFF-7014-00041.

simple choreography languages is complex, due to the high number of cases that need to be considered and to the multitude of rules in the semantics of both choreography and process languages. Such proofs are known to be prone to errors when designed and checked by humans: a previous attempt to formalize a publication on a higher-order process calculus [3] turned up a number of problems in the original proofs [4].

Choreographic Programming is closely related to Multiparty Session Types, a typing discipline for concurrent programming that also guarantees desirable properties. The main difference between these two approaches is methodological: Multiparty Session Types work bottom-up, starting from an implementation and trying to find a type; Choreographic Programming works top-down, starting from a choreography (which can be thought of as a type with additional computational abilities and information on the data being communicated) and generating the implementation. It has recently been discovered that a significant number of published results in Multiparty Session Types were wrong, in the sense that not only did the published proofs contain errors, but also the stated results did not hold [5, Chapter 8.1]. Here again, the problem is the complexity of the proofs involved, both in terms of number of cases to be checked and technical complexity of checking each individual case.

In order to establish solid foundations for Choreographic Programming, we propose to formalize the core choreography calculus from [1] using the Coq theorem prover. This calculus was proposed originally as a minimal calculus that already embodies the characteristic features of Choreographic Programming. As such, it provides a good benchmark both to evaluate the feasibility of a full formalization of a model for Choreographic Programming and to verify its correctness by certified means. Moreover, this calculus already includes the major challenges that have to be dealt with in this theory, namely: finite sets and functions on finite sets; partial functions; syntactic binders.

Furthermore, [1] also includes a proof that this choreography model is Turing-complete. Formalizing this proof also requires formalizing Kleene's theory of partial recursive functions [2], which again deals with partiality and finite sets, but also poses some additional problems related to induction over dependent types.

Currently our formalization covers the fragment of the choreography language that does not include recursion (infinite behaviour). This fragment already requires treating finite sets and functions (as the semantics of choreographies is defined by means of a function assigning each process to the value it stores), as well as partial functions (even without recursion, projecting a choreography to a process implementation is not always possible). Furthermore, in this fragment we can already encode a subset of partial recursive functions. As such, this work is already illustrative of the challenges encountered and the solutions that can be put in place.

References

[1] Luís Cruz-Filipe and Fabrizio Montesi. A core model for choreographic programming. In Olga Kouchnarenko and Ramin Khosravi, editors, *FACS*, volume 10231 of *LNCS*, pages 17–35. Springer, 2017.

[2] S.C. Kleene. *Introduction to Metamathematics*, volume 1. North-Holland Publishing Co., 1952.

[3] Ivan Lanese, Jorge A. Pérez, Davide Sangiorgi, and Alan Schmitt. On the expressiveness and decidability of higher-order process calculi. *Inf. Comput.*, 209(2):198–226, 2011.

[4] Petar Maksimovic and Alan Schmitt. HOCore in Coq. In Christian Urban and Xingyuan Zhang, editors, *ITP*, volume 9236 of *LNCS*, pages 278–293. Springer, 2015.

[5] Alceste Scalas and Nobuko Yoshida. Less is more: multiparty session types revisited. *PACMPL*, 3(POPL):30:1–30:29, 2019.