Linear Logic and Choreographic Programming

Matteo Acclavio[1] Giulia Manara[2*] Fabrizio Montesi[2]

[1] University of Sussex, [2] University of Southern Denmark

Choreographic programming is a high-level paradigm for specifying the intended global behavior of communicating systems [12]. In this setting, a *choreography* describes the interaction among processes as a single, coherent specification. From a choreography, local process implementations can be automatically derived through *endpoint projection* (EPP), and, under suitable conditions, a choreography can be extracted from a given network of processes. A key property of this paradigm is that choreographies are deadlock-free by design, and so are the process networks generated by EPP. Despite its advantages, a fundamental question remains open:

Which processes can be captured by choreographies? (1)

This connection was anticipated in [5], which relates choreographies to cut elimination in LL. Since cut elimination implies progress [14], this suggests that choreographies correspond to proofs of deadlock freedom. We formalize this idea by developing a general logical methodology for reasoning about process calculi in the style of logic programming, where processes are represented as formulas and executions correspond to derivations.[1] In particular, we extend first-order multiplicative additive linear logic (MALL_1) with a non-commutative connective $\blacktriangleleft$ for prefixing and nominal quantifiers И, Я to model name restriction, yielding the system PiL. This framework supports faithful encodings of various π-calculus dialects (monadic, polyadic, synchronous, asynchronous) and allows one to characterize deadlock freedom in logical terms: a process P is deadlock-free whenever the formula $[\![P]\!]$ is provable.

By establishing a correspondence between proofs and process executions, we obtain a foundation for reasoning about global behavior. We build on this connection to study the expressiveness of choreographies and show that, for a broad class of deadlock-free processes choreographic programming is *complete*.

The system PiL. We consider formulas are generated by countable set of variables $\mathcal{V}$ and a unique constant $\circ$ (called ***unit***), using the multiplicative conjunction ($\otimes$) and disjunction ($⅋$), the additive conjunction (&) and disjunction

*Funded by the European Union (ERC, CHORDS, 101124225)

[1] Our approach follow a longstanding line of work on processes-as-formulas approach inspired by the logic programming paradigm, see, e.g., [10, 9, 4, 8]

$$A, B ::= \circ \quad \text{unit (atom)}$$
$$\mid \langle x!y\rangle \quad \text{atom}$$
$$\mid (x?y) \quad \text{atom}$$
$$\mid A \parr B \quad \text{par}$$
$$\mid A \otimes B \quad \text{tensor}$$
$$\mid A \blacktriangleleft B \quad \text{prec}$$
$$\mid A \oplus B \quad \text{oplus}$$
$$\mid A \,\&\, B \quad \text{with}$$
$$\mid \forall x.A \quad \text{for all}$$
$$\mid \exists x.A \quad \text{exists}$$
$$\mid И x.A \quad \text{new}$$
$$\mid Я x.A \quad \text{ya}$$

$$\mathsf{ax}\ \frac{}{\mathcal{S} \vdash \langle x!y\rangle, (x?y)} \qquad \parr\ \frac{\mathcal{S} \vdash \Gamma, A, B}{\mathcal{S} \vdash \Gamma, A \parr B} \qquad \otimes\ \frac{\mathcal{S}_1 \vdash \Gamma, A \quad \mathcal{S}_2 \vdash B, \Delta}{\mathcal{S}_1, \mathcal{S}_2 \vdash \Gamma, A \otimes B, \Delta}$$

$$\oplus\ \frac{\mathcal{S} \vdash \Gamma, A_k}{\mathcal{S} \vdash \Gamma, \bigoplus_{i=1}^{n} A_i} \text{ for a } k \in \{1, \dots, n\} \qquad \&\ \frac{\mathcal{S} \vdash \Gamma, A_1 \quad \cdots \quad \mathcal{S} \vdash \Gamma, A_n}{\mathcal{S} \vdash \Gamma, \&_{i=1}^{n} A_i}$$

$$\forall\ \frac{\mathcal{S} \vdash \Gamma, A}{\mathcal{S} \vdash \Gamma, \forall x.A}\ \dagger \qquad \exists\ \frac{\mathcal{S} \vdash \Gamma, A[y/x]}{\mathcal{S} \vdash \Gamma, \exists x.A}$$

$$\circ\ \frac{}{\mathcal{S} \vdash \circ} \qquad \blacktriangleleft\ \frac{\mathcal{S}_1 \vdash \Gamma, A_1, \dots, A_n \quad \mathcal{S}_2 \vdash \Delta, B_1, \dots, B_n}{\mathcal{S}_1, \mathcal{S}_2 \vdash \Gamma, \Delta, A_1 \blacktriangleleft B_1, \dots, A_n \blacktriangleleft B_n}\ n \geq 0$$

$$И_{\circ}\ \frac{\mathcal{S} \vdash \Gamma, A}{\mathcal{S} \vdash \Gamma, И x.A}\ \dagger \qquad И_{\mathsf{load}}\ \frac{\mathcal{S}, x^{И} \vdash \Gamma, A}{\mathcal{S} \vdash \Gamma, И x.A}\ \dagger \qquad И_{\mathsf{pop}}\ \frac{\mathcal{S} \vdash \Gamma, A[y/x]}{\mathcal{S}, y^{И} \vdash \Gamma, Я x.A}$$

$$Я_{\circ}\ \frac{\mathcal{S} \vdash \Gamma, A}{\mathcal{S} \vdash \Gamma, Я x.A}\ \dagger \qquad Я_{\mathsf{load}}\ \frac{\mathcal{S}, x^{Я} \vdash \Gamma, A}{\mathcal{S} \vdash \Gamma, Я x.A}\ \dagger \qquad Я_{\mathsf{pop}}\ \frac{\mathcal{S} \vdash \Gamma, A[y/x]}{\mathcal{S}, y^{Я} \vdash \Gamma, И x.A}$$

Figure 1: Sequent calculus rules, with $\dagger ::= x \notin (\mathsf{free}(\Gamma) \cup \mathcal{S})$.

($\oplus$), the universal ($\forall$) and existential ($\exists$) quantifiers, together with a novel connective ***prec***[2] ($\blacktriangleleft$) and two dual nominal quantifiers called ***new*** ($И x$) and ***ya*** ($Я$). The syntax of formulas is given in Figure 1,

The ***(linear) implication*** $A \multimap B$ is defined as $A^{\perp} \parr B$, where the negation is defined by the following the standard ***De Morgan laws*** for MALL_1, plus the following:

$$\langle x!y\rangle^{\perp} = (x?y) \qquad \circ^{\perp} = \circ \qquad A \blacktriangleleft B^{\perp} = A^{\perp} \blacktriangleleft B^{\perp} \qquad И x.A^{\perp} = Я x.A^{\perp}$$

In Figure 1, we also recall the rules of the ***sequent calculus*** PiL. These rules operate on sequents of the form $\mathcal{S} \vdash \Gamma$, made of a set of occurrences of formulas Γ, and a ***store*** $\mathcal{S}$ of ***nominal variable***, that is, element of the form $x^{И}$ or $x^{Я}$ for a variable $x \in \mathcal{V}$. The rules are divided into two groups: the first group contains the standard rules of MALL_1. The second group contains the rules for the unit $\circ$, the new connective $\blacktriangleleft$ and the nominal quantifiers $И$ and $Я$.

Remark 1 (Rules for the prec connective)**.** *The non-commutative and non-associative connective $\blacktriangleleft$ is governed by a rule $\blacktriangleleft$ introducing multiple $\blacktriangleleft$-formulas simultaneously. The case for $n = 0$ is the standard rule* $\mathsf{mix}\ \dfrac{\mathcal{S}_1 \vdash \Gamma \quad \mathcal{S}_2 \vdash \Delta}{\mathcal{S}_1, \mathcal{S}_2 \vdash \Gamma, \Delta}$*. The case $n = 2$ rules captures the self-duality of $\blacktriangleleft$, by introducing in a single rule the connetive and its dual, following a general pattern observed for multiplicative connectives (see [1, Remark 5]).*

Remark 2 (Rules for the nominal quantifiers)**.** *The store enforces a disciplined use of names by ensuring that each $И$ is matched with at most one $Я$ (or*

[2]The name *prec* is an abbreviation for *preceed*. The name is chosen to suggest that the connective models the prefix operator in process calculi, which is used to model the sending and receiving of messages.

vice versa) along a derivation branch. If a rule $И_\circ$ *(resp.* $Я_\circ$*) is applied, then the nominal quantifier is not linked, reason why the rule reminds the standard universal quantifier rule. Otherwise, either the rule* $И_{\mathsf{load}}$ *(resp.* $Я_{\mathsf{load}}$*) loads a nominal variable in the store, or a rule* $И_{\mathsf{pop}}$ *(resp.* $Я_{\mathsf{pop}}$*) uses a nominal variable (of dual type) occurring in the store as a witness variable.*

The rules for nominal quantifier are designed to ensure four key properties:

1. *Equivariance:* $\eth x.\eth y.A$ *is logically equivalent to* $\eth y.\eth x.A$*.*
2. *Non-diagonality:* $\eth x.\eth y.A(x,y)$ *does not entail* $\eth z.A(z,z)$*, and vice versa.*[3]
3. *Scope extrusion:* $(\eth x.A) \parr B$ *and* $\eth x.(A \parr B)$ *are equivalent when* x *does not occur in* B*, reflecting the behavior of restriction in parallel composition.*
4. *Name-choice: if* $\odot \in \{\oplus, \&\}$*, then* $\eth x.A \odot \eth x.B$ *is equivalent to* $\eth x.(A \odot B)$*.*

Theorem 3. *The rule* $\mathsf{cut}\dfrac{\mathcal{S}_1 \vdash \Gamma, A \quad \mathcal{S}_2 \vdash A^\perp, \Delta}{\mathcal{S}_1, \mathcal{S}_2 \vdash \Gamma, \Delta}$ *is admissible in* PiL*.*

Proof. The proof is obtained by extending the cut-elimination procedure for MALL_1. Handling the (multiplicative) connective $\blacktriangleleft$ is straightforward. However, for nominal quantifiers, the proof requires more care due to the implicit links between store and nominal quantifier rules. This requires to use the auxiliary rule: $\mathcal{S}\text{-}\mathsf{cut}\dfrac{\mathcal{S}, x^{И}, x^{Я} \vdash \Gamma}{\mathcal{S} \vdash \Gamma}$ which is introduced and eliminated during the cut-elimination procedure. For details, see [2]. □

Choreographies as proofs (of deadlock-freedom)

We illustrate how PiL can be used to characterize deadlock freedom in the race-free[4] fragment of the synchronous π-calculus. This allows us to establish a *choreographies-as-proofs* correspondence, proving the completeness of choreographies with respect to deadlock-free race-free finite networks.

The syntax and operational semantics of the π-calculus are recalled in Figure 2. An execution of a process P is *successful* if it terminates in Nil. The ***execution tree*** of a process P is a tree whose nodes are labeled with the processes that can be executed from P, and whose edges are labeled with the reduction steps that lead to the child nodes.

The translation of processes into formulas is defined as follows:

$$\begin{gathered}
[\![\mathsf{Nil}]\!] = \circ \qquad [\![P \mid Q]\!] = [\![P]\!] \parr [\![Q]\!] \qquad [\![(\nu x)(P)]\!] = И x.[\![P]\!] \\
[\![x!\langle y\rangle.P]\!] = \langle x!y\rangle \blacktriangleleft [\![P]\!] \qquad [\![x?(y).P]\!] = \exists y.((x?y) \blacktriangleleft [\![P]\!]) \\
[\![x \triangleleft \{\ell : P_\ell\}_{\ell\in L}]\!] = \mathop{\&}_{\ell\in L} (\langle x!\ell\rangle \blacktriangleleft [\![P_\ell]\!]) \qquad [\![x \triangleright \{\ell : P_\ell\}_{\ell\in L}]\!] = \bigoplus_{\ell\in L} ((x?\ell) \blacktriangleleft [\![P_\ell]\!])
\end{gathered} \tag{2}$$

[3] This fails for self-dual nominal quantifiers, as in [13, 7, 6, 11].

[4] A process is race-free when no two sub-processes ever compete for the same resource, i.e., no reachable term contains parallel sends or receives (messages or labels) on the same channel.

$$
\begin{array}{l|l}
P, Q, R \coloneqq \mathsf{Nil} & \text{nil} \\
\quad \mid\ x!\langle y\rangle.P & \text{send } (y \text{ on } x) \\
\quad \mid\ x?(y).P & \text{receive } (y \text{ on } x) \\
\quad \mid\ P \mid Q & \text{parallel} \\
\quad \mid\ (\nu x)P & \text{nu} \\
\quad \mid\ x \triangleleft \{\ell : P_\ell\}_{\ell \in L} & \text{label send (on } x) \\
\quad \mid\ x \triangleright \{\ell : P_\ell\}_{\ell \in L} & \text{label receive (on } x)
\end{array}
$$

$$
\begin{array}{lll}
\text{Com:} & x!\langle a\rangle.P \mid x?(b).Q \to P \mid Q[a/b] & \\
\text{Choice:} & x \triangleleft \{\ell : P_\ell\}_{\ell \in L} \to x \triangleleft \{\ell_k : P_{\ell_k}\} & \text{if } \ell_k \in L \\
\text{Label:} & x \triangleleft \{\ell_k : P_{\ell_k}\} \mid x \triangleright \{\ell : Q_\ell\}_{\ell \in L} \to P_{\ell_k} \mid Q_{\ell_k} & \text{if } \ell_k \in L \\
\text{Res:} & (\nu x)P \to (\nu x)P' & P \to P' \\
\text{Par:} & P \mid Q \to P' \mid Q & P \to P'
\end{array}
$$

$$
C, C_\ell \coloneqq \underset{\text{end}}{\mathbf{0}} \quad \mid \quad \underset{\text{communication}}{[\mathsf{p}.x \to \mathsf{q}.y]; C} \quad \mid \quad \underset{\text{choice}}{\mathsf{p}.L \to \mathsf{q}.L' : k \left\{ \begin{array}{l|l} \ell : C_\ell & \ell \in L \\ \ell : S_\ell & \ell \in L' \setminus L \end{array} \right\}} \quad \mid \quad \underset{\text{restriction}}{(\nu x)C^x} \text{ (with } C^x \text{ containing no } (\nu x))
$$

Figure 2: **Above:** syntax and reduction semantics for the π-calculus. **Below:** syntax for choreographies.

The translation extends with minor adjustments to asynchronous and polyadic variants. This latter requires n-ary send/receive predicates in PiL.

Remark 4. *The terms $x \triangleleft \{\ell : P_\ell\}_{\ell \in L}$ and $x \triangleright \{\ell : P_\ell\}_{\ell \in L}$ model* internal *and* external *choice, respectively. Internal choice, made by the process itself, is encoded using the connective &, while external choice, determined by the context, corresponds to $\oplus$. The key distinction lies in how these connectives behave during proof search: & introduces branching by duplicating the context, enabling multiple execution paths to be explored, whereas $\oplus$ selects a single branch. This branching mechanism is essential for representing non-determinism and reasoning about deadlock freedom.*

The translation of processes execution trees (resp. successful execution trees) into open derivations (resp. derivations) in PiL is defined by induction on the structure of the trees is given by considering the following translation of the operational semantics reduction steps into open derivations in PiL. Details are provided in [3, 2]. See Figure 3 for an example of an execution of a process and its corresponding derivation. Thanks to this translation, we can formulate a correspondence between the executions of a process and the derivations of its translation in PiL.

Theorem 5. *Let P be a race-free process. Then P is deadlock-free iff $\vdash_{\mathsf{PiL}} P$.*

Proof. If P is deadlock-free, then all its executions terminate successfully, and the translation of the execution tree of P is a proof in PiL. Conversely, given a proof of $\vdash [\![P]\!]$ in PiL, we can independent use rule permutations [5] to transform a given derivation of $[\![P]\!]$ into a derivation made of blocks, each of which corresponds to a reduction step of the operational semantics. Then translate such derivation into a successful execution tree of P. Details are provided in [3]. □

Such correspondence establishee the foundation for relating choreographies to proofs.

[5] As the ones usually used during cut-elimination to ensure that the active formula of a cut is principal for a rule.

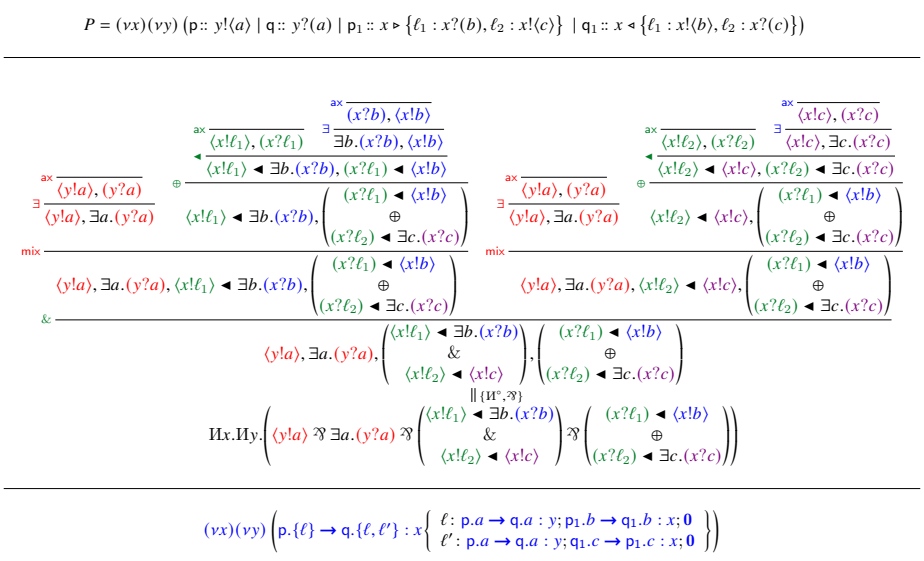

Figure 3: The process P, the derivation corresponding to two possible executions, and the choreography extracted from the derivation.

Theorem 6. *Let P be a race-free flat process, then*

$$P \text{ is deadlock-free} \iff \text{there is a choreography } C \text{ such that } \mathsf{EPP}(C) = P.$$

Proof. It follows from the proof of Theorem 5: each of the blocks corresponding to the reduction step in the reduction semantics can be seen as a choreographic instruction. □

An example of a choreography extracted from a proof in PiL is shown in Figure 3.

References

[1] Acclavio, M.: Sequent systems on undirected graphs. In: Benzmüller, C., Heule, M.J., Schmidt, R.A. (eds.) Automated Reasoning. pp. 216–236. Springer Nature Switzerland, Cham (2024)

[2] Acclavio, M., Manara, G.: Proofs as execution trees for the π-calculus (2025), `https://arxiv.org/abs/2411.08847`

[3] Acclavio, M., Manara, G., Montesi, F.: Formulas as processes, deadlock-freedom as choreographies. In: Vafeiadis, V. (ed.) Programming Languages and Systems. pp. 23–55. Springer Nature Switzerland, Cham (2025)

[4] Bruscoli, P.: A purely logical account of sequentiality in proof search. In: International Conference on Logic Programming. pp. 302–316. Springer (2002)

[5] Carbone, M., Montesi, F., Schürmann, C.: Choreographies, logically. Distributed Comput. **31**(1), 51–67 (2018). https://doi.org/10.1007/S00446-017-0295-1, https://doi.org/10.1007/s00446-017-0295-1

[6] Cheney, J.: A simpler proof theory for nominal logic. In: Sassone, V. (ed.) Foundations of Software Science and Computational Structures. pp. 379–394. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)

[7] Gabbay, M.J., Pitts, A.M.: A new approach to abstract syntax with variable binding. Form. Asp. Comput. **13**(3–5), 341–363 (jul 2002). https://doi.org/10.1007/s001650200016, https://doi.org/10.1007/s001650200016

[8] Horne, R.: The consistency and complexity of multiplicative additive system virtual. Sci. Ann. Comput. Sci. **25**(2), 245–316 (2015). https://doi.org/10.7561/SACS.2015.2.245, https://doi.org/10.7561/SACS.2015.2.245

[9] Kobayashi, N.: Concurrent Linear Logic Programming. Ph.D. thesis, University of Tokyo (1996)

[10] Miller, D.: The π-calculus as a theory in linear logic: Preliminary results. In: Lamma, E., Mello, P. (eds.) Extensions of Logic Programming. pp. 242–264. Springer Berlin Heidelberg, Berlin, Heidelberg (1993)

[11] Miller, D., Tiu, A.: A proof theory for generic judgments. ACM Trans. Comput. Logic **6**(4), 749–783 (oct 2005). https://doi.org/10.1145/1094622.1094628, https://doi.org/10.1145/1094622.1094628

[12] Montesi, F.: Introduction to Choreographies. Cambridge University Press (2023). https://doi.org/10.1017/9781108981491

[13] Pitts, A.M.: Nominal logic, a first order theory of names and binding. Information and Computation **186**(2), 165–193 (2003). https://doi.org/https://doi.org/10.1016/S0890-5401(03)00138-X, https://www.sciencedirect.com/science/article/pii/S089054010300138X, theoretical Aspects of Computer Software (TACS 2001)

[14] Wadler, P.: Propositions as sessions. In: Proceedings of the 17th ACM SIGPLAN International Conference on Functional Programming. p. 273–286. ICFP '12, Association for Computing Machinery, New York, NY, USA (2012). https://doi.org/10.1145/2364527.2364568, https://doi.org/10.1145/2364527.2364568