Forwarders as Process Compatibility, Logically

Marco Carbone[1], Sonia Marin[2], and Carsten Schürmann[1]

[1] Computer Science Department, IT University of Copenhagen, Denmark
`carbonem | carsten@itu.dk`
[2] School of Computer Science, University of Birmingham, United Kingdom
`s.marin@bham.ac.uk`

Session types, originally proposed by Honda et al. [8], are type annotations that ascribe protocols to processes in a concurrent system and determine how they behave when communicating together. *Binary session types* found a logical justification in *linear logic*, identified by Caires and Pfenning [2] and later by Wadler [13, 14], which establishes the following correspondences: linear logic propositions *as* session types, proofs *as* processes, cut reductions in linear logic *as* reductions in the operational semantics, and duality *as* a notion of compatibility ensuring that two processes communications match.

The situation is not as direct for *multiparty session types* [9, 10], which generalize binary session types to protocols with more than two participants. One of the central observations is that compatibility requires a stronger property than duality, still ensuring that all messages sent by any participating party will eventually be collected by another. In [6], Deniélou and Yoshida first proposed the notion of *multiparty compatibility* as a "new extension to multiparty interactions of the duality condition for binary session type".

In this work, we carve out a fragment of classical linear logic (CLL) that adequately captures the notion of multiparty compatibility. We observe moreover that the processes corresponding to proofs in this fragment act as *forwarders* of messages between participants, generalizing existing proposals, such coherence [5], arbiters [3], and medium processes [1]. Our main result is that the multiparty compatibility of processes can be formalized by the means of a forwarder, and conversely, that any set of processes modelled by a forwarder is compatible.

Processes and types. The language of *processes* we use to represent communicating entities is a standard variant of the π-calculus [12] used in the context of session types: $P, Q ::= x \leftrightarrow y$ (link) $\mid$ $x().P$ (wait) $\mid$ $x[]$ (close) $\mid$ $x(y).P$ (input) $\mid$ $x[y \triangleright P].Q$ (output).

Types, taken to be propositions of CLL, denote the way an endpoint x (a channel's end) must be used at runtime. In this work, we must annotate each operator with another endpoint u (or a list of endpoints $\tilde{u}$ in the case of *gathering* of communications) to make explicit the destination of messages. Therefore, the syntax of types is defined as $A, B ::= a \mid a^\perp \mid \perp^u \mid \mathbf{1}^{\tilde{u}} \mid (A \parr^u B) \mid (A \otimes^{\tilde{u}} B)$, where a denotes atoms.

Multiparty compatibility. Multiparty compatibility [6, 11, 7] is a semantic notion that uses types as abstractions of the processes behaviors and simulates their execution.

In order to give a semantics to types, we introduce FIFO queues defined as $\Psi ::= \epsilon \mid A \cdot \Psi \mid * \cdot \Psi$, capturing messages that are waiting to be delivered. A *message* can be a proposition A or a session termination symbol $*$. Every pair of endpoints (x, y) has an associated queue containing messages in transit from endpoint x to endpoint y. A *queue environment* σ is a mapping from pairs of endpoints to queues: $\sigma : (x, y) \mapsto \Psi$. σ_ϵ denotes the queue environment with only empty queues while $\sigma[(x, y) \mapsto \Psi]$ denotes a new environment where the entry for (x, y) has been updated to Ψ.

We define the *type-context semantics* for context $\Delta = x_1 : B_1, \ldots, x_k : B_k$ given environment σ as the minimum relation on $\Delta \bullet \sigma$ following rules in Fig. 1. Let $\alpha_1, \ldots, \alpha_n$ denote a *path* for context Δ if $\Delta \bullet \sigma \xrightarrow{\alpha_1} \Delta_1 \bullet \sigma_1 \ldots \xrightarrow{\alpha_n} \Delta_n \bullet \sigma_n$. This path is *maximal* if there is no

$$
\begin{array}{lcl}
x : a^{\perp}, y : a \bullet \sigma_{\epsilon} & \xrightarrow{x \leftrightarrow y} & \varnothing \bullet \sigma_{\epsilon} \\
x : \mathbf{1}^{\tilde{y}} \bullet \ \sigma_{\epsilon}[\{(y_i,x) \mapsto *\}_i] & \xrightarrow{\tilde{y}\mathbf{1}x} & \varnothing \bullet \sigma_{\epsilon} \\
\Delta, x : \perp^{y} \bullet \ \sigma[(x,y) \mapsto \Psi] & \xrightarrow{x \perp y} & \Delta \bullet \sigma[(x,y) \mapsto \Psi \cdot *] \\
\Delta, x : A \parr^{y} B \bullet \sigma[(x,y) \mapsto \Psi] & \xrightarrow{x \parr y} & \Delta, x : B \bullet \sigma[(x,y) \mapsto \Psi \cdot A] \\
\Delta, x : A \otimes^{\tilde{y}} B \bullet \sigma[\{(y_i,x) \mapsto A_i \cdot \Psi_i\}_i] & \xrightarrow{\tilde{y} \otimes x[A,\{A_i\}_i]} & \Delta, x : B \bullet \sigma[\{(y_i,x) \mapsto \Psi_i\}_i]
\end{array}
$$

Figure 1: Type-Context Semantics

$$
\frac{}{x \leftrightarrow y \Vdash x : a^{\perp}, y : a}\ \mathrm{Ax}
\qquad
\frac{P \Vdash \Gamma, [\![\Psi]\!][^{u}*]x : \cdot}{x().P \Vdash \Gamma, [\![\Psi]\!]x : \perp^{u}}\ \perp
\qquad
\frac{}{x[] \Vdash \{[^{x}*]u_i : \cdot\}_i, x : \mathbf{1}^{\tilde{u}}}\ \mathbf{1}\ (\tilde{u} \neq \varnothing)
$$

$$
\frac{P \Vdash \Gamma, [\![\Psi]\!][^{u}y : A]x : B}{x(y).P \Vdash \Gamma, [\![\Psi]\!]x : A \parr^{u} B}\ \parr
\qquad
\frac{P \Vdash \left\{y_i : A_i\right\}_i, y : A \quad Q \Vdash \Gamma, \left\{[\![\Psi_i]\!]u_i : A_i\right\}_i, [\![\Psi]\!]x : B}{x[y \triangleright P].Q \Vdash \Gamma, \left\{[^{x}y_i : A_i][\![\Psi_i]\!]u_i : C_i\right\}_i, [\![\Psi]\!]x : A \otimes^{\tilde{u}} B}\ \otimes\ (\tilde{u} \neq \varnothing)
$$

Figure 2: Proof System for Forwarders

$\Delta_{n+1}, \sigma_{n+1}, \alpha_{n+1}$ such that $\Delta_n \bullet \sigma_n \xrightarrow{\alpha_{n+1}} \Delta_{n+1} \bullet \sigma_{n+1}$ and it is *live* if $\Delta_n = \varnothing$ and $\sigma_n = \sigma_\epsilon$, meaning that it progresses without reaching an error.

Definition 1. *A context* Δ *is* multiparty compatible *if all maximal paths* $\alpha_1, \ldots, \alpha_n$ *for* Δ *are live and such that, if* $\alpha_p = \tilde{y} \otimes x[A, \{A_i\}_i]$*, then* $x : A^{\perp}, \left\{y_i : A_i^{\perp}\right\}_i$ *is also multiparty compatible.*

Forwarders. To capture multiparty compatibility, forwarders are designed as a subclass of derivations in classical linear logic that satisfies three conditions: i) anything received must be forwarded, ii) anything that is forwarded must have been previously received, and iii) the order of messages between two endpoints must be preserved.

In order to enforce these properties, the logic of forwarders is defined on judgements of the form $P \Vdash \Gamma$ where P is a process and $\Gamma ::= \varnothing \mid \Gamma, [\![\Psi]\!]x : B \mid \Gamma, [\![\Psi]\!]x : \cdot$ is an extended type context which associates to each endpoint x a priority queue $[\![\Psi]\!]$ consisting of any yet to process messages originating from x.

Formally, $[\![\Psi]\!] ::= \epsilon \mid [^{u}*][\![\Psi]\!] \mid [^{u}y : A][\![\Psi]\!]$ where $[^{u}y : A]$ expresses that a new y of type A has been received and will need to be forwarded later to endpoint u and, similarly, $[^{u}*]$ indicates that a request for closing a session has been received and must be forwarded to u.

Forwarders behave asynchronously: the order of messages needing to be forwarded to *independent* endpoints is irrelevant unless they originate at the same x. This follows the idea of having a queue for every ordered pair of endpoints in the type-context semantics above.

Definition 2. *A process* P *is a* forwarder *for* Δ *if the judgement* $P \Vdash \Delta$ *is derivable using the rules reported in Fig. 2.*

Correspondence. We can conclude this abstract with the statement of the theorem that forwarders characterize multiparty compatibility.

Theorem 3. Δ *is multiparty compatible iff there exists a forwarder for* Δ*, i.e., a process* P *such that* $P \Vdash \Delta$*.*

The proof, as well as more details on forwarders including additives for branching, exponentials to model servers and clients, internal cut-elimination for forwarder composition, and applications to multiparty communication *as* multi-cut elimination, can be found in [4].

References

[1] Luís Caires and Jorge A. Pérez. Multiparty session types within a canonical binary theory, and beyond. In Elvira Albert and Ivan Lanese, editors, *Formal Techniques for Distributed Objects, Components, and Systems - 36th IFIP WG 6.1 International Conference, FORTE 2016, Held as Part of the 11th International Federated Conference on Distributed Computing Techniques, DisCoTec 2016, Heraklion, Crete, Greece, June 6-9, 2016, Proceedings*, volume 9688 of *Lecture Notes in Computer Science*, pages 74–95. Springer, 2016.

[2] Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In *CONCUR*, pages 222–236, 2010.

[3] Marco Carbone, Sam Lindley, Fabrizio Montesi, Carsten Schürmann, and Philip Wadler. Coherence generalises duality: A logical explanation of multiparty session types. In Josée Desharnais and Radha Jagadeesan, editors, *27th International Conference on Concurrency Theory, CONCUR 2016, August 23-26, 2016, Québec City, Canada*, volume 59 of *LIPIcs*, pages 33:1–33:15, Germany, 2016. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[4] Marco Carbone, Sonia Marin, and Carsten Schürmann. Forwarders as process compatibility, logically. *CoRR*, abs/2112.07636, 2021.

[5] Marco Carbone, Fabrizio Montesi, Carsten Schürmann, and Nobuko Yoshida. Multiparty session types as coherence proofs. In *CONCUR*, pages 412–426, 2015.

[6] Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty session types meet communicating automata. In Helmut Seidl, editor, *Programming Languages and Systems - 21st European Symposium on Programming, ESOP 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, volume 7211 of *Lecture Notes in Computer Science*, pages 194–213. Springer, 2012.

[7] Silvia Ghilezan, Jovanka Pantovic, Ivan Prokic, Alceste Scalas, and Nobuko Yoshida. Precise subtyping for asynchronous multiparty sessions. *Proc. ACM Program. Lang.*, 5(POPL):1–28, 2021.

[8] Kohei Honda, Vasco Vasconcelos, and Makoto Kubo. Language primitives and type disciplines for structured communication-based programming. In *ESOP*, pages 22–138, 1998.

[9] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In George C. Necula and Philip Wadler, editors, *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*, pages 273–284. ACM, 2008.

[10] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. *JACM*, 63(1):9, 2016. Also: POPL, 2008, pages 273–284.

[11] Julien Lange and Nobuko Yoshida. Verifying asynchronous interactions via communicating session automata. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*, volume 11561 of *Lecture Notes in Computer Science*, pages 97–117. Springer, 2019.

[12] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–40,41–77, September 1992.

[13] Philip Wadler. Propositions as sessions. In *ICFP*, pages 273–286, 2012.

[14] Philip Wadler. Propositions as sessions. *Journal of Functional Programming*, 24(2–3):384–418, 2014. Also: ICFP, pages 273–286, 2012.