A Linear Logic for Fixpoints and Deadlock Freedom in the Pi-Calculus

Juan C. Jaramillo
University of Groningen

Damiano Mazza
CNRS, LIPN, Université Sorbonne Paris Nord

Jorge A. Pérez
University of Groningen

The connection between linear logic and process calculi has been studied for decades and is now well-established. In ongoing work, we have developed a new type system for the asynchronous pi-calculus based on linear logic, which enforces deadlock and lock freedom without imposing any determinism, termination or acyclic conditions coming from proofs. We introduce *fixpoint replication*, a new process construct that expresses recursion with simple types, like the fixpoint combinator in PCF. Fixpoint replication is justified by a natural generalization of the promotion rule in linear logic. Our work provides a unifying view on approaches to concurrency based on session types and on differential linear logic, in which logical features (cocontraction, fixpoints, logical correctness) correspond to behavioral features (non-determinism, non-termination, acyclicity) and may be modularly combined.

Context The connection between linear logic and process calculi has been investigated for decades and is now well-established. The line of work initiated by Caires and Pfenning relating linear logic propositions with session types [3, 23, 4] is arguably the most developed. Another approach, rooted in old work of Bellin and Scott [2], was suggested by Honda and Laurent [11] and further developed by Ehrhard and Laurent using the differential extension of linear logic [9, 8]. In spite of a mismatch between the non-determinism offered by the standard operational semantics of differential linear logic and the non-determinism of concurrent calculi [16], this viewpoint is not devoid of interest: it has allowed importing intersection types into the π-calculus [5], and structures originating more or less directly from differential linear logic have recently been proposed to model server-client interactions [20] or shared state in session types [21, 22]. Resource calculi related to differential linear logic have also been studied in the context of session-type-based concurrency [18].

This Talk Proposal. This talk proposal is based on ongoing work; our contributions are two-fold, with both conceptual and a technical facets. On the conceptual side, we offer a unifying view on the two lines of research mentioned above. On the technical side, we substantially improve on previous work on type systems for deadlock and lock freedom.

The *conceptual contribution* stems from realizing how the *asynchronous* session-based correspondence with linear logic presented by DeYoung et al. [7] (a variant of the synchronous interpretation originally proposed in [3]) is actually intimately related to Honda and Laurent's correspondence.[1] We may thus leverage the non-determinism of differential linear logic and, simultaneously, previous work based on interpreting asynchronous session types in linear logic, resulting in a modular framework reaching a remarkable level of expressiveness.

On the other hand, the *technical contribution* lies in tackling the problem of ensuring deadlock and lock freedom in cyclic process networks with infinite behaviors. Informally, deadlock freedom ensures

[1] Unbeknownst to the authors, Laurent [15] had already been aware for quite some time that, once reformulated in classical linear logic and augmented with cocontraction, DeYoung et al.'s system is basically an unpolarized version of the system he had introduced with Honda [11]. However, he never published this observation.

Submitted to:
TLLA 2026

that processes "never get stuck"; lock freedom ensures that all the linear input/output actions are used in a reduction.

Deadlock and lock freedom are known to be particularly challenging to enforce compositionally: (dead)locks arise from non-local dependencies between interacting programs, even when such programs are correct and (dead)lock-free in isolation. Our type system ensures these properties by relying on *priorities* to control the dependencies between processes. This approach, first developed by Kobayashi (cf. [12, 13]), was later revisited by Padovani [17], who achieved a streamlined type system with increased expressiveness. Albeit Padovani's system is explicitly based on linearity, its types and typing rules have no apparent relationship with linear logic. Over the past few years, priorities have been integrated in session types, yielding systems [6, 1, 14, 10] in which types are (annotated) linear logic formulas and in which one may type deadlock-free processes with cyclic topologies (e.g. the dining philosophers), beyond the tree-like structure of logical proofs. None of these systems, however, matches Padovani's in expressiveness.

By contrast, our system *subsumes* Padovani's, thus providing a principled, logical explanation of his work while simultaneously broadening its scope. Compared to Padovani's, the above-mentioned systems suffer from limitations given by determinism or termination (or both). Let us explain how we overcome them.

Non-determinism via Cocontraction The main feature of differential linear logic is the symmetry between the two so-called exponential modalities. In linear logic, the $?(-)$ modality is governed by three rules, called *dereliction*, *weakening*, and *contraction*. In process calculi, these correspond to a client invoking a server, to the absence of clients and to the presence of multiple clients, respectively. Next to these, differential linear logic adds three symmetric rules for the $!(-)$ modality, called *codereliction*, *coweakening*, and *cocontraction*. The latter is a key element of our system: it corresponds to the concurrent presence of multiple servers on the same channel, thus introducing races and non-determinism—our types do *not* enforce confluence.

Codereliction and coweakening, which would correspond to a "single-use server" and a "non-responding server", are excluded from our system. For the acquainted reader, this avoids proofs reducing to the proof with empty semantics (the "zero proof"), which is crucial for deadlock freedom.

Non-termination via Fixpoint Replication Linear logic with cocontraction enjoys normalization, so a type system based on it enforces termination on typable processes. The situation is similar to the simply-typed λ-calculus, which is strongly normalizing by virtue of its correspondence with intuitionistic logic. The simply-typed λ-calculus may be promoted to a Turing-complete functional language by adding fixpoints and conditionals, as exemplified by Plotkin's PCF [19]. Albeit PCF programs may not terminate, types still ensure that they "do not go wrong", in the sense of absence of runtime errors.

Since our type system is grounded in linear logic, we may follow the same recipe. Conditionals are given by additive connectives. For fixpoints, all we need to do is express them in linear logic and read back what we obtain in process calculus syntax. The result is **fixpoint replication**, a new form of replication—a key technical novelty, which we proceed to illustrate.

Consider the usual replication construct, denoted $!x(a).P$, which denotes a persistent input on x (a server), with behavior P. This process is typable using the promotion rule

$$\frac{P \vdash ?\Gamma, a : A}{!x(a).P \vdash ?\Gamma, x : !A}\ \textsc{Prom}$$

Such server interacts with a request from a client $\overline{x}\langle b\rangle \mid Q$ as follows:

$$\overline{x}\langle b\rangle \mid Q \mid !x(a).P \longrightarrow P\{^b/_a\} \mid Q \mid !x(a).P \qquad (\star)$$

where $P\{^b/_a\}$ denotes the spawned instance of the server, now available locally to Q. Observe that while standard promotion induces servers with an "in width" infinite behavior (parallel copies of the same process), it cannot express recursive behaviors, i.e., "in depth" infinite behavior.

Our new fixpoint replication construct has the form $!_z x(a).P$, where z is a name that is bound in P. As customary, this may be seen as a server. The interaction with a client is now defined by the rule

$$\overline{x}\langle b\rangle \mid Q \mid !_z x(a).P \longrightarrow (\nu x')\big(P\{^b/_a, ^{x'}/_z\} \mid Q \mid !_z x'(a).P\big) \mid !_z x(a).P \qquad (\star\star)$$

Thus, the client spawns an instance of P and *two copies* of the server: a global copy (blue), available to other clients, and a local copy (orange), available only to the spawned instance of P. Intuitively, it is this local copy that allows us to express recursive definitions, via client requests along the freshly created name x'. Notice that if z is not free in P then the local copy is inaccessible and we fall back on the usual reduction rule for replication $(\star)$.

Example 1. Consider the server $S \triangleq !_z x(a).\overline{z}\langle a\rangle$, which receives a name on x and forwards it along z. The server can be invoked by the clients $C_b \triangleq \overline{x}\langle b\rangle$ and $C_c \triangleq \overline{x}\langle c\rangle$. The following reduction shows an interaction between S and C_b:

$$!_z x(a).\overline{z}\langle a\rangle \mid \overline{x}\langle b\rangle \mid \overline{x}\langle c\rangle \longrightarrow (\nu x')(\overline{x'}\langle b\rangle \mid !_z x'(a).\overline{z}\langle a\rangle) \mid !_z x(a).\overline{z}\langle a\rangle \mid \overline{x}\langle c\rangle$$

Note that $\overline{x'}\langle b\rangle$, a copy of the body of the server, is recursively requesting $!_z x'(a).\overline{z}\langle a\rangle$. The reduction also spawns $!_z x(a).\overline{z}\langle a\rangle$, which is ready to interact with C_c.

Key idea: A Generalized Promotion Rule In an untyped context, the behavior expressed by rule $(\star\star)$ is encodable using standard replication, just as fixpoint combinators are encodable in the untyped λ-calculus. The key point, however, is that our new fixpoint replication is simply-typable in a way compatible with the above reduction, via a generalized version of the *promotion rule* of linear logic:[2]

$$\frac{P \vdash ?\Gamma, z : ?A^{\perp}, a : A}{!_z x(a).P \vdash ?\Gamma, x : !A}\ \textsc{GenProm}$$

Again, if z is not free in P, then its type $?A^{\perp}$ is introduced by weakening, and the above rule is equivalent to the standard promotion rule. This rule reveals the origins of fixpoint replication: since an intuitionistic sequent $\Gamma \vdash A$ translates in linear logic as $\vdash ?\Gamma^{\perp}, A$, this is just the linear logic version of a rule for typing fixpoints in PCF, with premise $\Gamma, z : A \vdash M : A$ and conclusion $\Gamma \vdash \mathsf{Y}(\lambda z.M) : A$, where Y is the fixpoint combinator. There is a direct connection between fixpoint replication and PCF-like fixpoints, which we formally elucidate by means of an encoding.

Giving up Logical Correctness: Deadlock Freedom via Priorities What we have so far is a "logically correct" simply-typed process calculus, capable of expressing all computable functions and arbitrary non-determinism. Basically, it may be seen either as DeYoung et al.'s asynchronous π-calculus [7] with cocontraction and recursive definitions, or as an unpolarized version of Honda and Laurent's calculus [11]

[2] Upon discussing our work with Thomas Ehrhard, we learned that this rule is known as *Danos's fixpoint*.

with recursive definitions. Such a calculus is still limited in terms of concurrency: typable processes are essentially π-calculus encodings of classical (in the sense of the $\lambda\mu$-calculus) non-deterministic PCF programs, much like the original session-typed systems [3, 23] essentially yield π-calculus encodings of $\lambda\mu$-terms. The system does ensure deadlock freedom, but in an uninteresting way: it only types tree-like network topologies arising from purely functional programs.

Our next move then is to abandon logical correctness, in the footsteps of the previously mentioned authors [6, 14, 10]. Technically, we decorate our types, which are just linear logic formulas, with elements from a totally ordered set of *priorities* (in fact, integers) and we consider a unary cut rule, which may create cyclic configurations. We rely on priorities to exclude deadlocks, as in the following example.

Consider the process $P \triangleq x(z_1,z_2).\bar{y}\langle u_1,u_2\rangle$ (an input on x followed by an output on y). Following [3, 23], we interpret $\parr$ as input and $\otimes$ as output; as in prior works [6, 14, 10] here we annotate these connectives with priorities (denoted $\kappa,\rho,\ldots$). This way, if $x_i : A_i^\perp$ and if $\Gamma = u_1 : B_1, u_2 : B_2$, then the judgment $P \vdash \Gamma, x : A_1^\perp \parr^\rho A_2^\perp, y : B_1 \otimes^\kappa B_2$ will be derivable only if $\rho < \kappa$, thus specifying that the input on x is blocking and must happen before the output on y. Similarly, if $\Gamma' = t_1 : A_1, t_2 : A_2$ and $Q \triangleq y(z_1,z_2).\bar{x}\langle t_1,t_2\rangle$ then a dual judgment $Q \vdash \Gamma', x : A_1 \otimes^{\rho'} A_2, y : B_1^\perp \parr^{\kappa'} B_2^\perp$ will require $\kappa' < \rho'$. Clearly, although P and Q implement complementary actions, they cannot synchronize—there is a deadlock: since duality preserves priorities, typing $P \mid Q$ would force $\rho = \rho'$ and $\kappa = \kappa'$, implying contradictory inequalities $\rho < \kappa$ and $\kappa < \rho$.

Following the intuitions described above, we define a type system where types are decorated with priorities. For instance Rule GenProm decorated with priorities is defined as follows:

$$\frac{P \vdash ?^{\rho_i}\Gamma, z : ?^{\rho}A^\perp, a : \uparrow^{\kappa}A \quad \rho_i > \rho}{!_z x(a).P \vdash ?^{\rho_i}\Gamma, x : !^{\rho}A}\ \textsc{GenPromPri}$$

where $?^{\rho_i}\Gamma$ means that each type is of the form $?^{\rho_i}A_i$, and $\uparrow^k A$, read as A shifted by κ, equals A where its priorities are inductively incremented by κ. Thus, for instance $\uparrow^1(B_1 \otimes^\kappa B_2) = \uparrow^1 B_1 \otimes^{\kappa+1} \uparrow^1 B_2$. This way we allow a and z to have dual behaviors with different priorities. The side condition $\rho_i > \rho$ ensures that x has the smallest priority, thus specifying that the replicated input along x is blocking the rest of actions in P. Moreover, note that the types of x and z are their exact duals, thus allowing to type recursive definitions.

We show how priorities are liberal enough to allow cyclic network topologies. These, however, need one further technical ingredient, provided by (equi)recursive types. As in PCF, these are enough to express the type of natural numbers because we have sum types (the additive connectives).

As mentioned above, there is in fact an encoding of Padovani's linear types [17] to ours and a map from his type derivations to ours which respects the encoding. This exhibits the system of [17] as, essentially, our system in which the priorities on exponential modalities are forgotten. Once the logical nature of the types of [17] is unveiled, it is quite natural that priorities apply to exponential modalities as well. This, it turns out, allows to type forms of guarded servers not typable in [17].

References

[1] Stephanie Balzer, Bernardo Toninho & Frank Pfenning (2019): *Manifest Deadlock-Freedom for Shared Session Types*. In Luís Caires, editor: *Programming Languages and Systems - 28th European Symposium on Programming, ESOP 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*, *Lecture Notes in Computer Science* 11423, Springer, pp. 611–639, doi:10.1007/978-3-030-17184-1_22. Available at `https://doi.org/10.1007/978-3-030-17184-1_22`.

[2] Gianluigi Bellin & Philip J. Scott (1994): *On the pi-Calculus and Linear Logic*. *Theor. Comput. Sci.* 135(1), pp. 11–65, doi:10.1016/0304-3975(94)00104-9. Available at `https://doi.org/10.1016/0304-3975(94)00104-9`.

[3] Luís Caires & Frank Pfenning (2010): *Session Types as Intuitionistic Linear Propositions*. In Paul Gastin & François Laroussinie, editors: *CONCUR 2010 - Concurrency Theory, 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings*, *Lecture Notes in Computer Science* 6269, Springer, pp. 222–236, doi:10.1007/978-3-642-15375-4_16.

[4] Luís Caires, Frank Pfenning & Bernardo Toninho (2016): *Linear logic propositions as session types*. *Math. Struct. Comput. Sci.* 26(3), pp. 367–423, doi:10.1017/S0960129514000218.

[5] Ugo Dal Lago, Marc de Visme, Damiano Mazza & Akira Yoshimizu (2019): *Intersection types and runtime errors in the pi-calculus*. *Proc. ACM Program. Lang.* 3(POPL), pp. 7:1–7:29, doi:10.1145/3290320. Available at `https://doi.org/10.1145/3290320`.

[6] Ornela Dardha & Simon J. Gay (2018): *A New Linear Logic for Deadlock-Free Session-Typed Processes*. In Christel Baier & Ugo Dal Lago, editors: *Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, *Lecture Notes in Computer Science* 10803, Springer, pp. 91–109, doi:10.1007/978-3-319-89366-2_5. Available at `https://doi.org/10.1007/978-3-319-89366-2_5`.

[7] Henry DeYoung, Luís Caires, Frank Pfenning & Bernardo Toninho (2012): *Cut Reduction in Linear Logic as Asynchronous Session-Typed Communication*. In Patrick Cégielski & Arnaud Durand, editors: *CSL*, *LIPIcs* 16, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 228–242.

[8] Thomas Ehrhard & Olivier Laurent (2010): *Acyclic Solos and Differential Interaction Nets*. *Log. Methods Comput. Sci.* 6(3). Available at `http://arxiv.org/abs/1007.0120`.

[9] Thomas Ehrhard & Olivier Laurent (2010): *Interpreting a finitary pi-calculus in differential interaction nets*. *Inf. Comput.* 208(6), pp. 606–633, doi:10.1016/J.IC.2009.06.005. Available at `https://doi.org/10.1016/j.ic.2009.06.005`.

[10] Bas van den Heuvel & Jorge A. Pérez (2024): *Asynchronous Session-Based Concurrency: Deadlock-freedom in Cyclic Process Networks*. *Log. Methods Comput. Sci.* 20(4), doi:10.46298/LMCS-20(4:6)2024. Available at `https://doi.org/10.46298/lmcs-20(4:6)2024`.

[11] Kohei Honda & Olivier Laurent (2010): *An exact correspondence between a typed pi-calculus and polarised proof-nets*. *Theor. Comput. Sci.* 411(22-24), pp. 2223–2238, doi:10.1016/J.TCS.2010.01.028. Available at `https://doi.org/10.1016/j.tcs.2010.01.028`.

[12] Naoki Kobayashi (2002): *Type Systems for Concurrent Programs*. In Bernhard K. Aichernig & T. S. E. Maibaum, editors: *Formal Methods at the Crossroads. From Panacea to Foundational Support, 10th Anniversary Colloquium of UNU/IIST, the International Institute for Software Technology of The United Nations University, Lisbon, Portugal, March 18-20, 2002, Revised Papers*, *Lecture Notes in Computer Science* 2757, Springer, pp. 439–453, doi:10.1007/978-3-540-40007-3_26. Available at `https://doi.org/10.1007/978-3-540-40007-3_26`.

[13] Naoki Kobayashi (2006): *A New Type System for Deadlock-Free Processes*. In: *CONCUR'06*, *LNCS* 4137, pp. 233–247.

[14] Wen Kokke & Ornela Dardha (2023): *Prioritise the Best Variation*. *Log. Methods Comput. Sci.* 19(4), doi:10.46298/LMCS-19(4:28)2023. Available at `https://doi.org/10.46298/lmcs-19(4:28)2023`.

[15] Olivier Laurent (2013): *A Linear Logic Typing System for a π-Calculus*. Unpublished manuscript, personally communicated to the authors.

[16] Damiano Mazza (2018): *The true concurrency of differential interaction nets*. *Mathematical Structures in Computer Science* 28(7), pp. 1097–1125.

[17] Luca Padovani (2014): *Deadlock and Lock Freedom in the Linear π-Calculus*. In: *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, CSL-LICS '14, Association for Computing Machinery, New York, NY, USA. Available at https://doi.org/10.1145/2603088.2603116.

[18] Joseph W. N. Paulus, Daniele Nantes-Sobrinho & Jorge A. Pérez (2023): *Non-Deterministic Functions as Non-Deterministic Processes (Extended Version)*. *Log. Methods Comput. Sci.* 19(4), doi:10.46298/LMCS-19(4:1)2023. Available at https://doi.org/10.46298/lmcs-19(4:1)2023.

[19] Gordon D. Plotkin (1977): *LCF Considered as a Programming Language*. *Theor. Comput. Sci.* 5(3), pp. 223–255, doi:10.1016/0304-3975(77)90044-5. Available at https://doi.org/10.1016/0304-3975(77)90044-5.

[20] Zesen Qian, G. A. Kavvos & Lars Birkedal (2021): *Client-server sessions in linear logic*. *Proc. ACM Program. Lang.* 5(ICFP), pp. 1–31, doi:10.1145/3473567. Available at https://doi.org/10.1145/3473567.

[21] Pedro Rocha & Luís Caires (2021): *Propositions-as-types and shared state*. *Proc. ACM Program. Lang.* 5(ICFP), pp. 1–30, doi:10.1145/3473584. Available at https://doi.org/10.1145/3473584.

[22] Pedro Rocha & Luís Caires (2023): *Safe Session-Based Concurrency with Shared Linear State*. In Thomas Wies, editor: *Programming Languages and Systems - 32nd European Symposium on Programming, ESOP 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings*, *Lecture Notes in Computer Science* 13990, Springer, pp. 421–450, doi:10.1007/978-3-031-30044-8_16. Available at https://doi.org/10.1007/978-3-031-30044-8_16.

[23] Philip Wadler (2012): *Propositions as sessions*. In Peter Thiemann & Robby Bruce Findler, editors: *ACM SIGPLAN International Conference on Functional Programming, ICFP'12, Copenhagen, Denmark, September 9-15, 2012*, ACM, pp. 273–286, doi:10.1145/2364527.2364568. Available at https://doi.org/10.1145/2364527.2364568.