



An Infinitary Proof Theory of Linear Logic Ensuring Fair Termination in the Linear π -Calculus

Luca Ciccone  

University of Torino, Italy

Luca Padovani  

University of Torino, Italy

Abstract

Fair termination is the property of programs that may diverge “in principle” but that terminate “in practice”, *i.e.* under suitable fairness assumptions concerning the resolution of non-deterministic choices. We study a conservative extension of μMALL^∞ , the infinitary proof system of the multiplicative additive fragment of linear logic with least and greatest fixed points, such that cut elimination corresponds to fair termination. Proof terms are processes of πLIN , a variant of the linear π -calculus with (co)recursive types into which binary and (some) multiparty sessions can be encoded. As a result we obtain a behavioral type system for πLIN (and indirectly for session calculi through their encoding into πLIN) that ensures fair termination: although well-typed processes may engage in arbitrarily long interactions, they are *fairly* guaranteed to eventually perform all pending actions.

2012 ACM Subject Classification Theory of computation \rightarrow Linear logic; Theory of computation \rightarrow Process calculi; Theory of computation \rightarrow Program analysis

Keywords and phrases Linear π -calculus, Linear Logic, Fixed Points, Fair Termination

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.36

Related Version *Full Version:* <https://arxiv.org/abs/2207.03749> [10]

Acknowledgements We are grateful to Simona Ronchi Della Rocca and Francesco Dagnino for their comments on an early draft of this paper, to Stephanie Balzer for having provided pointers to related work and to the anonymous CONCUR reviewers for their questions and detailed feedback.

1 Introduction

The *linear π -calculus* [28] is a typed refinement of Milner’s π -calculus in which linear channels can be used only once, for a one-shot communication. As it turns out, the linear π -calculus is the fundamental model underlying a broad family of communicating processes. In particular, all *binary sessions* [22, 23, 25] and some *multiparty sessions* [24] can be encoded into the linear π -calculus [27, 5, 11, 8]. Sessions are private communication channels linking two or more processes and whose usage is disciplined by a *session type*, a type representing a structured communication protocol. In all session type systems, session endpoints are *linearized channels* that can be used repeatedly but in a sequential manner. The key insight for encoding sessions into the linear π -calculus is to encode linearized channels in a continuation-passing style: when some payload is transmitted over a linear channel, it can be paired with another linear channel (the continuation) on which the subsequent interaction takes place.

In this work we propose a type system for πLIN , a linear π -calculus with (co)recursive types, such that well-typed processes are *fairly terminating*. Fair termination [21, 17] is a liveness property stronger than *lock freedom* [26, 32] – *i.e.* the property that every pending action can be eventually performed – but weaker than *termination*. In particular, a fairly terminating program may diverge, but all of its infinite executions are considered “unfair” – read impossible or unrealistic – and so they can be ignored insofar termination is concerned. A simple example of fairly terminating program is that modeling a repeated interaction



© Luca Ciccone and Luca Padovani;

licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Slawomir Lasota, and Anca Muscholl; Article No. 36; pp. 36:1–36:18

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

between a buyer and a seller in which the buyer may either pay the seller and terminate or may add an item to the shopping cart and then repeat the same behavior. In principle, there is an execution of the program in which the buyer keeps adding items to the shopping cart and never pays. In practice, this behavior is considered unfair and the program terminates under the fairness assumption that the buyer eventually pays the seller.

Our type system is a conservative extension of μMALL^∞ [3, 16, 2], the infinitary proof system for the multiplicative additive fragment of linear logic with least and greatest fixed points. In fact, the modifications we make to μMALL^∞ are remarkably small: we add one (standard) rule to deal with *non-deterministic choices*, those performed autonomously by a process, and we relax the validity condition on μMALL^∞ proofs so that it only considers the “fair behaviors” of the program it represents. The fact that there is such a close correspondence between the typing rules of πLIN and the inference rules of μMALL^∞ is not entirely surprising. After all, there have been plenty of works investigating the relationship between π -calculus terms and linear logic proofs, from those of Abramsky [1], Bellin and Scott [4] to those on the interpretation of linear logic formulas as session types [15, 40, 6, 30, 37, 35]. Nonetheless, we think that the connection between πLIN and μMALL^∞ stands out for two reasons. First, πLIN is conceptually simpler and more general than the session-based calculi that can be encoded in it. In particular, all the session calculi based on linear logic rely on an asymmetric interpretation of the multiplicative connectives \otimes and \wp so that $\varphi \otimes \psi$ (respectively, $\varphi \wp \psi$) is the type of a session endpoint used for sending (respectively, receiving) a message of type φ and then used according to ψ . In our setting, the connectives \otimes and \wp retain their symmetry since we interpret $\varphi \otimes \psi$ and $\varphi \wp \psi$ formulas as the output/input of pairs, in the same spirit of the original encoding of linear logic proofs proposed by Bellin and Scott [4]. This interpretation gives πLIN the ability of modeling *bifurcating protocols* of which binary sessions are just a special case. The second reason why πLIN and μMALL^∞ get along has to do with the cut elimination result for μMALL^∞ . In finitary proof systems for linear logic, cut elimination may proceed by removing *topmost cuts*. In μMALL^∞ there is no such notion as a topmost cut since μMALL^∞ proofs may be infinite. As a consequence, the cut elimination result for μMALL^∞ is proved by eliminating *bottom-most cuts* [2]. This strategy fits perfectly with the reduction semantics of πLIN – and that of any other conventional process calculus, for that matter – whereby reduction rules act only on the exposed (*i.e.* unguarded) part of processes but not behind prefixes. As a result, the reduction semantics of πLIN is completely ordinary, unlike other logically-inspired process calculi that incorporate commuting conversions [40, 30], perform reductions behind prefixes [35] or swap prefixes [4].

In previous work [9] we have proposed a type system ensuring the fair termination of binary sessions. The present work achieves the same objective using a more basic process calculus and exploiting its strong logical connection with μMALL^∞ . In fact, the soundness proof of our type system piggybacks on the cut elimination property of μMALL^∞ . Other session typed calculi based on linear logic with fixed points have been studied by Lindley and Morris [30] and by Derakhshan and Pfenning [14, 13]. The type systems described in these works respectively guarantee termination and strong progress, whereas our type system guarantees fair termination which is somewhat in between these properties. Overall, our type system seems to hit a sweet spot: on the one hand, it is deeply rooted in linear logic and yet it can deal with common communication patterns (like the buyer/seller interaction described above) that admit potentially infinite executions and therefore are out of scope of other logic-inspired type systems; on the other hand, it guarantees lock freedom [26, 32], strong progress [14, Theorem 12.3] and also termination, under a suitable fairness assumption.

■ **Table 1** Syntax of πLIN .

$P, Q ::= x \leftrightarrow y$	link	$(x)(P \parallel Q)$	composition
$\text{case } x\{\}$	empty input	$P \oplus Q$	choice
$x().P$	unit input	$\bar{x}()$	unit output
$x(z, y).P$	pair input	$\bar{x}(z, y)(P \parallel Q)$	pair output
$\text{case } x(y)\{P, Q\}$	sum input	$\text{in}_i \bar{x}(y).P$	sum output $i \in \{1, 2\}$
$\text{corec } x(y).P$	corecursion	$\text{rec } \bar{x}(y).P$	recursion

The paper continues as follows. Section 2 presents πLIN and the fair termination property ensured by our type system. Section 3 describes πLIN types, which are suitably embellished μMALL^∞ formulas. Section 4 describes the inference rules of μMALL^∞ rephrased as typing rules for πLIN . Section 5 identifies the valid typing derivations and states the properties of well-typed processes. Section 6 discusses related work in more detail and Section 7 presents ideas for future developments. Supplementary material, additional examples and proofs can be found in the long version of the paper [10].

2 Syntax and Semantics of πLIN

In this section we define syntax and reduction semantics of πLIN , a variant of the linear π -calculus [28] in which all channels are meant to be used for *exactly* one communication. The calculus supports (co)recursive data types built using units, pairs and disjoint sums. These data types are known to be the essential ingredients for the encoding of sessions in the linear π -calculus [27, 11, 38].

We assume given an infinite set of *channels* ranged over by x, y and z . πLIN processes are coinductively generated by the productions of the grammar shown in Table 1 and their informal meaning is given below. A *link* $x \leftrightarrow y$ acts as a *linear forwarder* [18] that forwards a single message either from x to y or from y to x . The uncertainty in the direction of the message is resolved once the term is typed and the polarity of the types of x and y is fixed (Section 4). The term $\text{case } x\{\}$ represents a process that receives an empty message from x and then fails. This form is only useful in the metatheory: the type system guarantees that well-typed processes never fail, since it is not possible to send empty messages. The term $\bar{x}()$ models a process that sends the unit on x , effectively indicating that the interaction is terminated, whereas $x().P$ models a process that receives the unit from x and then continues as P . The term $\bar{x}(y, z)(P \parallel Q)$ models a process that creates two new channels y and z , sends them in a pair on channel x and then forks into two parallel processes P and Q . Dually, $x(y, z).P$ models a process that receives a pair containing two channels y and z from channel x and then continues as P . The term $\text{in}_i \bar{x}(y).P$ models a process that creates a new channel y and sends $\text{in}_i(y)$ (that is, the i -th injection of y in a disjoint sum) on x . Dually, $\text{case } x(y)\{P_1, P_2\}$ receives a disjoint sum from channel x and continues as either P_1 or P_2 depending on the tag in_i it has been built with. For clarity, in some examples we will use more descriptive labels such as **add** and **pay** instead of in_1 and in_2 . The terms $\text{rec } \bar{x}(y).P$ and $\text{corec } x(y).P$ model processes that respectively send and receive a new channel y and then continue as P . They do not contribute operationally to the interaction being modeled, but they indicate the points in a program where (co)recursive types are unfolded. A term $(x)(P \parallel Q)$ denotes the parallel composition of two processes P and Q that interact through the fresh channel x . Finally, the term $P \oplus Q$ models a non-deterministic choice between two behaviors P and Q .

π LIN binders are easily recognizable because they enclose channel names in round parentheses. Note that all outputs are in fact *bound outputs*. The output of free channels can be modeled by combining bound outputs with links [30]. For example, the output $\bar{x}(y, z)$ of a pair of free channels y and z can be modeled as the term $\bar{x}(y', z')(y \leftrightarrow y' \parallel z \leftrightarrow z')$. We identify processes modulo renaming of bound names, we write $\text{fn}(P)$ for the set of channel names occurring free in P and we write $\{y/x\}$ for the capture-avoiding substitution of y for the free occurrences of x . We impose a well-formedness condition on processes so that, in every sub-term of the form $\bar{x}(y, z)(P \parallel Q)$, we have $y \notin \text{fn}(Q)$ and $z \notin \text{fn}(P)$.

We omit any concrete syntax for representing infinite processes. Instead, we work directly with infinite trees obtained by corecursively unfolding contractive equations of the form $A(x_1, \dots, x_n) = P$. For each such equation, we assume that $\text{fn}(P) \subseteq \{x_1, \dots, x_n\}$ and we write $A(y_1, \dots, y_n)$ for its unfolding $P\{y_i/x_i\}_{1 \leq i \leq n}$. The technical report [10] describes the changes for supporting a more conventional (but slightly heavier) handling of infinite processes.

► **Notation 1.** To reduce clutter due to the systematic use of bound outputs, by convention we omit the continuation called y in Table 1 when its name is chosen to coincide with that of the channel x on which y is sent/received. For example, with this notation we have $x(z).P = x(z, x).P$ and $\text{in}_i \bar{x}.P = \text{in}_i \bar{x}(x).P$ and $\text{case } x\{P, Q\} = \text{case } x(x)\{P, Q\}$. ◻

A welcome side effect of adopting Notation 1 is that it gives the illusion of working with a session calculus in which the same channel x may be used repeatedly for multiple input/output operations, while in fact x is a linear channel used for exchanging a single message along with a fresh continuation that turns out to have the same name. If one takes this notation as native syntax for a session calculus, its linear π -calculus encoding [11] turns out to be precisely the π LIN term it denotes. Besides, the idea of rebinding the same name over and over is widespread in session-based functional languages [20, 34] as it provides a simple way of “updating the type” of a session endpoint after each use.

► **Example 2.** Below we model the interaction informally described in Section 1 between buyer and seller using the syntactic sugar defined in Notation 1:

$$(x)(\text{Buyer}\langle x \rangle \parallel \text{Seller}\langle x, y \rangle) \quad \text{where} \quad \begin{aligned} \text{Buyer}(x) &= \text{rec } \bar{x}.(\text{add } \bar{x}.\text{Buyer}\langle x \rangle \oplus \text{pay } \bar{x}.\bar{x}()) \\ \text{Seller}(x, y) &= \text{corec } x.\text{case } x\{\text{Seller}\langle x, y \rangle, x().\bar{y}()\} \end{aligned}$$

At each round of the interaction, the buyer decides whether to **add** an item to the shopping cart and repeat the same behavior (left branch of the choice) or to **pay** the seller and terminate (right branch of the choice). The seller reacts dually and signals its termination by sending a unit on the channel y . As we will see in Section 4, **rec** \bar{x} and **corec** x identify the points within processes where (co)recursive types are unfolded.

If we were to define *Buyer* using distinct bound names we would write an equation like

$$\text{Buyer}(x) = \text{rec } \bar{x}(y).(\text{add } \bar{y}(z).\text{Buyer}\langle z \rangle \oplus \text{pay } \bar{y}(z).\bar{z}())$$

and similarly for *Seller*. ◻

The operational semantics of the calculus is given in terms of the *structural precongruence* relation \preceq and the *reduction relation* \rightarrow defined in Table 2. As usual, structural precongruence relates processes that are syntactically different but semantically equivalent. In particular, [S-LINK] states that linking x with y is the same as linking y with x , whereas [S-COMM] and [S-ASSOC] state the expected commutativity and associativity laws for parallel composition. Concerning the latter, the side condition $x \in \text{fn}(Q)$ makes sure that Q (the process brought

■ **Table 2** Structural pre-congruence and reduction semantics of πLIN .

[S-LINK]	$x \leftrightarrow y \preceq y \leftrightarrow x$	
[S-COMM]	$(x)(P \parallel Q) \preceq (x)(Q \parallel P)$	
[S-ASSOC]	$(x)(P \parallel (y)(Q \parallel R)) \preceq (y)((x)(P \parallel Q) \parallel R)$	if $x \in \text{fn}(Q)$ and $y \notin \text{fn}(P)$
[R-LINK]	$(x)(x \leftrightarrow y \parallel P) \rightarrow P\{y/x\}$	
[R-UNIT]	$(x)(\bar{x}() \parallel x().P) \rightarrow P$	
[R-PAIR]	$(x)(\bar{x}(z, y)(P_1 \parallel P_2) \parallel x(z, y).Q) \rightarrow (z)(P_1 \parallel (y)(P_2 \parallel Q))$	
[R-SUM]	$(x)(\text{in}_i \bar{x}(y).P \parallel \text{case } x(y)\{P_1, P_2\}) \rightarrow (y)(P \parallel P_i)$	$i \in \{1, 2\}$
[R-REC]	$(x)(\text{rec } \bar{x}(y).P \parallel \text{corec } x(y).Q) \rightarrow (y)(P \parallel Q)$	
[R-CHOICE]	$P_1 \oplus P_2 \rightarrow P_i$	$i \in \{1, 2\}$
[R-CUT]	$(x)(P \parallel R) \rightarrow (x)(Q \parallel R)$	if $P \rightarrow Q$
[R-STRUCT]	$P \rightarrow Q$	if $P \preceq R \rightarrow Q$

closer to P when the relation is read from left to right) is indeed connected with P by means of the channel x . Note that [S-ASSOC] only states the right-to-left associativity of parallel composition and that the left-to-right associativity law $(x)((y)(P \parallel Q) \parallel R) \preceq (y)(P \parallel (x)(Q \parallel R))$ is derivable when $x \in \text{fn}(Q)$. The reduction relation is mostly unremarkable. Links are reduced with [R-LINK] by effectively merging the linked channels. All the reductions that involve the interaction between processes except [R-UNIT] create new continuation channels that connect the reducts. The rule [R-CHOICE] models the non-deterministic choice between two behaviors. Finally, [R-CUT] and [R-STRUCT] close reductions by cuts and structural pre-congruence. In the following we write \Rightarrow for the reflexive, transitive closure of \rightarrow and we say that P is *stuck* if there is no Q such that $P \rightarrow Q$.

We conclude this section by formalizing fair termination. To this aim, we introduce the notion of *run* as a maximal execution of a process. Hereafter ω stands for the lowest transfinite ordinal number and o ranges over the elements of $\omega + 1$.

► **Definition 3 (run).** A run of P is a sequence $(P_i)_{i \in o}$ where $o \in \omega + 1$ and $P_0 = P$ and $P_i \rightarrow P_{i+1}$ for all $i + 1 \in o$ and either the sequence is infinite or it ends with a stuck process.

Hereafter we use ρ to range over runs. Using runs we can define a range of termination properties for processes. In particular, P is *terminating* if all of its runs are finite and P is *weakly terminating* if it has a finite run. Fair termination is a termination property somewhat in between termination and weak termination in which only the “fair” runs of a process are taken into account insofar its termination is concerned. There exist several notions of fair run corresponding to different fairness assumptions [17, 29, 39]. The notion of fair run used here is an instance of the *fair reachability of predicates* by Queille and Sifakis [36].

► **Definition 4 (fair termination).** A run is fair if it contains finitely many weakly terminating processes. We say that P is fairly terminating if every fair run of P is finite.

To better understand our notion of fair run, it may be useful to think of the cases in which a run is *not* fair. An *unfair* run is necessarily infinite and describes the execution of a process that always has the chance to terminate but systematically avoids doing so. Think of the system modeled in Example 2: the (only) run in which the buyer adds items to the shopping cart forever and never pays the seller is unfair; any other run of the system is fair and finite. So, the system in Example 2 is not terminating (it admits an infinite execution) but it is fairly terminating (all the fair executions are finite).

The following result characterizes fair termination without using fair runs. Most importantly, it provides us with the key proof principle for the soundness of our type system.

► **Theorem 5** (proof principle for fair termination). *P is fairly terminating if and only if $P \Rightarrow Q$ implies that Q is weakly terminating.*

Theorem 5 says that any reasonable type system that ensures weak process termination also ensures fair process termination. By “reasonable” we mean a type system for which type preservation (also known as *subject reduction*) holds, which is usually the case. Indeed, if we consider a well-typed process P such that $P \Rightarrow Q$, we can deduce that Q is also well typed. Now, the soundness of the type system guarantees that Q is weakly terminating. By applying Theorem 5 from right to left, we conclude that every well-typed P is fairly terminating.

3 Formulas and Types

The types of π LIN are built using the multiplicative additive fragment of linear logic enriched with least and greatest fixed points. In this section we specify the syntax of types along with all the auxiliary notions that are needed to present the type system and prove its soundness.

The syntax of pre-formulas relies on an infinite set of *propositional variables* ranged over by X and Y and is defined by the grammar below:

Pre-formula $\varphi, \psi ::= \mathbf{0} \mid \top \mid \mathbf{1} \mid \perp \mid \varphi \oplus \psi \mid \varphi \& \psi \mid \varphi \otimes \psi \mid \varphi \wp \psi \mid \mu X.\varphi \mid \nu X.\varphi \mid X$

As usual, μ and ν are the binders of propositional variables and the notions of free and bound variables are defined accordingly. We assume that the body of fixed points extends as much as possible to the right of a pre-formula, so $\mu X.X \oplus \mathbf{1}$ means $\mu X.(X \oplus \mathbf{1})$ and not $(\mu X.X) \oplus \mathbf{1}$. We write $\{\varphi/X\}$ for the capture-avoiding substitution of all free occurrences of X with φ . We write φ^\perp for the *dual* of φ , which is the involution defined by the equations

$$\begin{aligned} \mathbf{0}^\perp &= \top & (\varphi \oplus \psi)^\perp &= \varphi^\perp \& \psi^\perp & (\mu X.\varphi)^\perp &= \nu X.\varphi^\perp \\ \mathbf{1}^\perp &= \perp & (\varphi \otimes \psi)^\perp &= \varphi^\perp \wp \psi^\perp & X^\perp &= X \end{aligned}$$

A *formula* is a closed pre-formula. In the context of π LIN, formulas describe how linear channels are used. Positive formulas (those built with the constants $\mathbf{0}$ and $\mathbf{1}$, the connectives \oplus and \otimes and the least fixed point) indicate output operations whereas negative formulas (the remaining forms) indicate input operations. The formulas $\varphi \oplus \psi$ and $\varphi \& \psi$ describe a linear channel used for sending/receiving a tagged channel of type φ or ψ . The tag (either in_1 or in_2) distinguishes between the two possibilities. The formulas $\varphi \otimes \psi$ and $\varphi \wp \psi$ describe a linear channel used for sending/receiving a pair of channels of type φ and ψ ; $\mu X.\varphi$ and $\nu X.\varphi$ describe a linear channel used for sending/receiving a channel of type $\varphi\{\mu X.\varphi/X\}$ or $\varphi\{\nu X.\varphi/X\}$ respectively. The constants $\mathbf{1}$ and \perp describe a linear channel used for sending/receiving the unit. Finally, the constants $\mathbf{0}$ and \top respectively describe channels on which nothing can be sent and from which nothing can be received.

► **Example 6.** Looking at the structure of *Buyer* and *Seller* in Example 2, we can make an educated guess on the type of the channel x they use. Concerning x , we see that it is used according to $\varphi \stackrel{\text{def}}{=} \mu X.X \oplus \mathbf{1}$ in *Buyer* and according to $\psi \stackrel{\text{def}}{=} \nu X.X \& \perp$ in *Seller*. Note that $\varphi = \psi^\perp$, suggesting that *Buyer* and *Seller* may interact correctly when connected. ◻

We write \preceq for the *subformula ordering*, that is the least partial order such that $\varphi \preceq \psi$ if φ is a subformula of ψ . For example, consider $\varphi \stackrel{\text{def}}{=} \mu X.\nu Y.X \oplus Y$ and $\psi \stackrel{\text{def}}{=} \nu Y.\varphi \oplus Y$. Then we have $\varphi \preceq \psi$ and $\psi \not\preceq \varphi$. When Φ is a set of formulas, we write $\min \Phi$ for its \preceq -minimum formula if it is defined. Occasionally we let \star stand for an arbitrary binary connective \oplus , \otimes , $\&$, or \wp and σ stand for an arbitrary fixed point operator μ or ν .

When two πLIN processes interact on some channel x , they may exchange other channels on which their interaction continues. We can think of these subsequent interactions stemming from a shared channel x as being part of the same conversation (the literature on *sessions* [22, 25] builds on this idea [27, 11]). The soundness proof of the type system is heavily based on the proof of the cut elimination property of μMALL^∞ , which relies on the ability to uniquely identify the types of the channels that belong to the same conversation and to trace conversations within typing derivations. Following the literature on μMALL^∞ [3, 16, 2], we annotate formulas with addresses. We assume an infinite set \mathcal{A} of *atomic addresses*, \mathcal{A}^\perp being the set of their duals such that $\mathcal{A} \cap \mathcal{A}^\perp = \emptyset$ and $\mathcal{A}^{\perp\perp} = \mathcal{A}$. We use a and b to range over elements of $\mathcal{A} \cup \mathcal{A}^\perp$. An *address* is a string aw where $w \in \{i, l, r\}^*$. The dual of an address is defined as $(aw)^\perp = a^\perp w$. We use α and β to range over addresses, we write \sqsubseteq for the prefix relation on addresses and we say that α and β are *disjoint* if $\alpha \not\sqsubseteq \beta$ and $\beta \not\sqsubseteq \alpha$.

A *type* is a formula φ paired with an address α written φ_α . We use S and T to range over types and we extend to types several operations defined on formulas: we use logical connectives to compose types so that $\varphi_{\alpha l} \star \psi_{\alpha r} \stackrel{\text{def}}{=} (\varphi \star \psi)_\alpha$ and $\sigma X.\varphi_{\alpha i} \stackrel{\text{def}}{=} (\sigma X.\varphi)_\alpha$; the dual of a type is obtained by dualizing both its formula and its address, that is $(\varphi_\alpha)^\perp \stackrel{\text{def}}{=} \varphi_{\alpha^\perp}^\perp$; type substitution preserves the address in the type within which the substitution occurs, but forgets the address of the type being substituted, that is $\varphi_\alpha\{\psi_\beta/X\} \stackrel{\text{def}}{=} \varphi\{\psi/X\}_\alpha$.

We often omit the address of constants (which represent terminated conversations) and we write \overline{S} for the formula obtained by forgetting the address of S . Finally, we write \rightsquigarrow for the least reflexive relation on types such that $S_1 \star S_2 \rightsquigarrow S_i$ and $\sigma X.S \rightsquigarrow S\{\sigma X.S/X\}$.

► **Example 7.** Consider once again the formula $\varphi \stackrel{\text{def}}{=} \mu X.X \oplus \mathbf{1}$ that describes the behavior of *Buyer* (Example 6) and let a be an arbitrary atomic address. We have

$$\varphi_a \rightsquigarrow (\varphi \oplus \mathbf{1})_{ai} \rightsquigarrow \varphi_{ail} \rightsquigarrow (\varphi \oplus \mathbf{1})_{aili} \rightsquigarrow \mathbf{1}_{ailir}$$

where the fact that the types in this sequence all share a common non-empty prefix “ a ” indicates that they belong to the same conversation. Note how the symbols i , l and r composing an address indicate the step taken in the syntax tree of types for making a move in this sequence: i means “inside”, when a fixed point operator is unfolded, whereas l and r mean “left” and “right”, when the corresponding branch of a connective is selected. \square

4 Type System

We now present the typing rules for πLIN . As usual we introduce typing contexts to track the type of the names occurring free in a process. A *typing context* is a finite map from names to types written $x_1 : S_1, \dots, x_n : S_n$. We use Γ and Δ to range over contexts, we write $\text{dom}(\Gamma)$ for the domain of Γ and Γ, Δ for the union of Γ and Δ when $\text{dom}(\Gamma) \cap \text{dom}(\Delta) = \emptyset$. Typing judgments have the form $P \vdash \Gamma$. We say that P is *quasi typed* in Γ if the judgment $P \vdash \Gamma$ is coinductively derivable using the rules shown in Table 3 and described below. For the time being we say “quasi typed” and not “well typed” because some infinite derivations using the rules in Table 3 are invalid. Well-typed processes are quasi-typed processes whose typing derivation satisfies some additional validity conditions that we detail in Section 5.

Rule [AX] states that a link $x \leftrightarrow y$ is quasi typed provided that x and y have dual types, but not necessarily dual addresses. Rule [CUT] states that a process composition $(x)(P \parallel Q)$ is quasi typed provided that P and Q use the linear channel x in complementary ways, one according to some type S and the other according to the dual type S^\perp . Note that the context Γ, Δ in the conclusion of the rule is defined provided that Γ and Δ have disjoint domains. This condition entails that P and Q do not share any channel other than x ensuring that the

■ **Table 3** Typing rules for π LIN.

$$\begin{array}{c}
\frac{}{x \leftrightarrow y \vdash x : \varphi_\alpha, y : \varphi_\beta^\perp} \text{[AX]} \qquad \frac{P \vdash \Gamma, x : S \quad Q \vdash \Delta, x : S^\perp}{(x)(P \parallel Q) \vdash \Gamma, \Delta} \text{[CUT]} \\
\\
\frac{}{\text{case } x\{\} \vdash \Gamma, x : \top} \text{[\top]} \qquad \frac{P \vdash \Gamma}{x().P \vdash \Gamma, x : \perp} \text{[\perp]} \qquad \frac{}{\bar{x}() \vdash x : \mathbf{1}} \text{[1]} \\
\\
\frac{P \vdash \Gamma, y : S, z : T}{x(y, z).P \vdash \Gamma, x : S \wp T} \text{[\wp]} \qquad \frac{P \vdash \Gamma, y : S \quad Q \vdash \Delta, z : T}{\bar{x}(y, z)(P \parallel Q) \vdash \Gamma, \Delta, x : S \otimes T} \text{[\otimes]} \\
\\
\frac{P \vdash \Gamma, y : S \quad Q \vdash \Gamma, y : T}{\text{case } x(y)\{P, Q\} \vdash \Gamma, x : S \& T} \text{[\&]} \qquad \frac{P \vdash \Gamma, y : S_i}{\text{in}_i \bar{x}(y).P \vdash \Gamma, x : S_1 \oplus S_2} \text{[\oplus]} \\
\\
\frac{P \vdash \Gamma, y : S\{\nu X.S/X\}}{\text{corec } x(y).P \vdash \Gamma, x : \nu X.S} \text{[\nu]} \qquad \frac{P \vdash \Gamma, y : S\{\mu X.S/X\}}{\text{rec } \bar{x}(y).P \vdash \Gamma, x : \mu X.S} \text{[\mu]} \qquad \frac{P \vdash \Gamma \quad Q \vdash \Gamma}{P \oplus Q \vdash \Gamma} \text{[CHOICE]}
\end{array}$$

interaction between P and Q may proceed without deadlocks. Rule $[\top]$ deals with a process that receives an empty message from channel x . Since this cannot happen, we allow the process to be quasi typed in any context. Rules $[1]$ and $[\perp]$ concern the exchange of units. The former rule states that $\bar{x}()$ is quasi typed in a context that contains a single association for the x channel with type $\mathbf{1}$, whereas the latter rule removes x from the context (hence from the set of usable channels), requiring the continuation process to be quasi typed in the remaining context. Rules $[\otimes]$ and $[\wp]$ concern the exchange of pairs. The former rule requires the two forked processes P and Q to be quasi typed in the respective contexts enriched with associations for the continuation channels y and z being created. The latter rule requires the continuation process to be quasi typed in a context enriched with the channels extracted from the received pair. Rules $[\oplus]$ and $[\&]$ deal with the exchange of disjoint sums in the expected way. Rules $[\mu]$ and $[\nu]$ deal with fixed point operators by unfolding the (co)recursive type of the channel x . As in μ MALL $^\infty$, the two rules have exactly the same structure despite the fact that the two fixed point operators being used are dual to each other. Clearly, the behavior of least and greatest fixed points must be distinguished by some other means, as we will see in Section 5 when discussing the validity of a typing derivation. Finally, $[\text{CHOICE}]$ deals with non-deterministic choices by requiring that each branch of a choice must be quasi typed in exactly the same typing context as the conclusion.

Besides the structural constraints imposed by the typing rules, we implicitly require that the types in the range of all typing contexts have pairwise disjoint addresses. This condition ensures that it is possible to uniquely trace a communication protocol in a typing derivation: if we have two channels x and y associated with two types φ_α and ψ_β such that $\alpha \sqsubseteq \beta$, then we know that y is a continuation resulting from a communication that started from x . In a sense, x and y represent different moments in the same conversation.

► **Example 8 (buyer and seller).** Let us show that the system described in Example 2 is quasi typed. To this aim, let $\varphi \stackrel{\text{def}}{=} \mu X.X \oplus \mathbf{1}$ and $\psi \stackrel{\text{def}}{=} \nu X.X \& \perp$ respectively be the formulas describing the behavior of *Buyer* and *Seller* on the channel x . Note that $\psi = \varphi^\perp$ and let a be an arbitrary atomic address. We derive

$$\frac{\frac{\vdots}{Buyer\langle x \rangle \vdash x : \varphi_{ai}} \text{add } \bar{x}.Buyer\langle x \rangle \vdash x : (\varphi \oplus \mathbf{1})_{ai} \text{ } [\oplus] \quad \frac{\frac{\vdots}{\bar{x}() \vdash x : \mathbf{1}} \text{ } [\mathbf{1}] \quad \text{pay } \bar{x}.\bar{x}() \vdash x : (\varphi \oplus \mathbf{1})_{ai} \text{ } [\oplus]}{\text{pay } \bar{x}.\bar{x}() \vdash x : (\varphi \oplus \mathbf{1})_{ai}} \text{ } [\text{CHOICE}]}{\text{add } \bar{x}.Buyer\langle x \rangle \oplus \text{pay } \bar{x}.\bar{x}() \vdash x : (\varphi \oplus \mathbf{1})_{ai}} \text{ } [\mu]}{Buyer\langle x \rangle \vdash x : \varphi_a} \text{ } [\mu]$$

and also

$$\frac{\frac{\vdots}{Seller\langle x, y \rangle \vdash x : \psi_{a^\perp i}, y : \mathbf{1}} \text{ } [\perp] \quad \frac{\frac{\vdots}{\bar{y}() \vdash y : \mathbf{1}} \text{ } [\mathbf{1}]}{x().\bar{y}() \vdash x : \perp, y : \mathbf{1}} \text{ } [\perp]}{\text{case } x\{Seller\langle x, y \rangle, x().\bar{y}()\} \vdash x : (\psi \& \perp)_{a^\perp i}, y : \mathbf{1}} \text{ } [\&]}{\text{Seller}\langle x, y \rangle \vdash x : \psi_{a^\perp}, y : \mathbf{1}} \text{ } [\nu]$$

showing that *Buyer* and *Seller* are quasi typed. Note that both derivations are infinite, but for dual reasons. In *Buyer* the infinite branch corresponds to the behavior in which *Buyer* chooses to add one more item to the shopping cart. This choice is made independently of the behavior of other processes in the system. In *Seller*, the infinite branch corresponds to the behavior in which *Seller* receives one more **add** message from *Buyer*. By combining these derivations we obtain

$$\frac{\frac{\vdots}{Buyer\langle x \rangle \vdash x : \varphi_a} \quad \frac{\vdots}{Seller\langle x, y \rangle \vdash x : \psi_{a^\perp}, y : \mathbf{1}}}{(x)(Buyer\langle x \rangle \parallel Seller\langle x, y \rangle) \vdash y : \mathbf{1}} \text{ } [\text{CUT}]$$

showing that the system as a whole is quasi typed. \lrcorner

As we have anticipated, there exist infinite typing derivations that are unsound from a logical standpoint, because they allow us to prove $\mathbf{0}$ or the empty sequent. For example, if we consider the non-terminating process $\Omega(x) = \Omega\langle x \rangle \oplus \Omega\langle x \rangle$ we obtain the infinite derivation

$$\frac{\frac{\vdots}{\Omega\langle x \rangle \vdash x : \mathbf{0}} \quad \frac{\vdots}{\Omega\langle x \rangle \vdash x : \mathbf{0}}}{\Omega\langle x \rangle \vdash x : \mathbf{0}} \text{ } [\text{CHOICE}] \tag{1}$$

showing that $\Omega\langle x \rangle$ is quasi typed. As illustrated by the next example, there exist non-terminating processes that are quasi typed also in logically sound contexts.

► **Example 9** (compulsive buyer). Consider the following variant of the *Buyer* process

$$Buyer(x, z) = \text{rec } \bar{x}.\text{add } \bar{x}.Buyer\langle x, z \rangle$$

that models a “compulsive buyer”, namely a buyer that adds infinitely many items to the shopping cart but never pays. Using $\varphi \stackrel{\text{def}}{=} \mu X.X \oplus \mathbf{1}$ and an arbitrary atomic address a we can build the following infinite derivation

$$\frac{\frac{\vdots}{Buyer\langle x \rangle \vdash x : \varphi_{ai}} \text{ } [\oplus]}{\text{add } \bar{x}.Buyer\langle x \rangle \vdash x : (\varphi \oplus \mathbf{1})_{ai}} \text{ } [\mu]}{Buyer\langle x \rangle \vdash x : \varphi_a} \text{ } [\mu]$$

showing that this process is quasi typed. By combining this derivation with the one for *Seller* in Example 8 we obtain a derivation establishing that $(x)(Buyer\langle x \rangle \parallel Seller\langle x, y \rangle)$ is quasi typed in the context $y : \mathbf{1}$, although this composition cannot terminate. \lrcorner

5 From Quasi-Typed to Well-Typed Processes

To rule out unsound derivations like those in Equation (1) and Example 9 it is necessary to impose a validity condition on derivations [3, 16]. Roughly speaking, μMALL^∞ 's validity condition requires every infinite branch of a derivation to be supported by the continuous unfolding of a greatest fixed point. In order to formalize this condition, we start by defining *threads*, which are sequences of types describing sequential interactions at the type level.

► **Definition 10 (thread).** A thread of S is a sequence of types $(S_i)_{i \in o}$ for some $o \in \omega + 1$ such that $S_0 = S$ and $S_i \rightsquigarrow S_{i+1}$ whenever $i + 1 \in o$.

Hereafter we use t to range over threads. For example, if we consider $\varphi \stackrel{\text{def}}{=} \mu X.X \oplus \mathbf{1}$ from Example 2 we have that $t \stackrel{\text{def}}{=} (\varphi_a, (\varphi \oplus \mathbf{1})_{ai}, \varphi_{ail}, \dots)$ is an infinite thread of φ_a . A thread is *stationary* if it has an infinite suffix of equal types. The above thread t is not stationary.

Among all threads, we are interested in finding those in which a ν -formula is unfolded infinitely often. These threads, called ν -threads, are precisely defined thus:

► **Definition 11 (ν -thread).** Let $t = (S_i)_{i \in \omega}$ be an infinite thread, let \bar{t} be the corresponding sequence $(\bar{S}_i)_{i \in \omega}$ of formulas and let $\text{inf}(t)$ be the set of elements of \bar{t} that occur infinitely often in \bar{t} . We say that t is a ν -thread if $\min \text{inf}(t)$ is defined and is a ν -formula.

If we consider the infinite thread t above, we have $\text{inf}(t) = \{\varphi, \varphi \oplus \mathbf{1}\}$ and $\min \text{inf}(t) = \varphi$, so t is not a ν -thread because φ is not a ν -formula. Consider instead $\varphi \stackrel{\text{def}}{=} \nu X.\mu Y.X \oplus Y$ and $\psi \stackrel{\text{def}}{=} \mu Y.\varphi \oplus Y$ and observe that ψ is the “unfolding” of φ . Now $t_1 \stackrel{\text{def}}{=} (\varphi_a, \psi_{ai}, (\varphi \oplus \psi)_{aai}, \varphi_{aail}, \dots)$ is a thread of φ_a such that $\text{inf}(t_1) = \{\varphi, \psi, \varphi \oplus \psi\}$ and we have $\min \text{inf}(t_1) = \varphi$ because $\varphi \preceq \psi$, so t_1 is a ν -thread. If, on the other hand, we consider the thread $t_2 \stackrel{\text{def}}{=} (\varphi_a, \psi_{ai}, (\varphi \oplus \psi)_{aai}, \psi_{aair}, (\varphi \oplus \psi)_{aairi}, \dots)$ such that $\text{inf}(t_2) = \{\psi, \varphi \oplus \psi\}$ we have $\min \text{inf}(t_2) = \psi$ because $\psi \preceq \varphi \oplus \psi$, so t_2 is not a ν -thread. Intuitively, the \preceq -minimum formula among those that occur infinitely often in a thread is the outermost fixed point operator that is being unfolded infinitely often. It is possible to show that this minimum formula is always well defined [16]. If such minimum formula is a greatest fixed point operator, then the thread is a ν -thread.

Now we proceed by identifying threads along branches of typing derivations. To this aim, we provide a precise definition of *branch*.

► **Definition 12 (branch).** A branch of a typing derivation is a sequence $(P_i \vdash \Gamma_i)_{i \in o}$ of judgments for some $o \in \omega + 1$ such that $P_0 \vdash \Gamma_0$ occurs somewhere in the derivation and $P_{i+1} \vdash \Gamma_{i+1}$ is a premise of the rule application that derives $P_i \vdash \Gamma_i$ whenever $i + 1 \in o$.

An infinite branch is valid if supported by a ν -thread that originates somewhere therein.

► **Definition 13 (valid branch).** Let $\gamma = (P_i \vdash \Gamma_i)_{i \in \omega}$ be an infinite branch in a derivation. We say that γ is valid if there exists $j \in \omega$ such that $(S_k)_{k \geq j}$ is a non-stationary ν -thread and S_k is in the range of Γ_k for every $k \geq j$.

For example, the infinite branch in the typing derivation for *Seller* of Example 2 is valid since it is supported by the ν -thread $(\psi_{a^+}, (\psi \& \perp)_{a^+i}, \psi_{a^+il}, \dots)$ where $\psi \stackrel{\text{def}}{=} \nu X.X \& \perp$ happens to be the \preceq -minimum formula that is unfolded infinitely often. On the other hand, the infinite branch in the typing derivation for *Buyer* of Example 9 is invalid, because the only infinite thread in it is $(\varphi_a, (\varphi \oplus \mathbf{1})_{ai}, \varphi_{ail}, \dots)$ which is not a ν -thread.

A μMALL^∞ derivation is valid if so is every infinite branch in it [3, 16]. For the purpose of ensuring fair termination, this condition is too strong because some infinite branches in a typing derivation may correspond to unfair executions that, by definition, we neglect insofar its termination is concerned. For example, the infinite branch in the derivation for *Buyer* of Example 8 corresponds to an unfair run in which the buyer insists on adding items to the shopping cart, despite it periodically has a chance of paying the seller and terminate the interaction. That typing derivation for *Buyer* would be considered an invalid proof in μMALL^∞ because the infinite branch is not supported by a ν -thread (in fact, there is a μ -formula that is unfolded infinitely many times along that branch, as in Example 9).

It is generally difficult to understand if a branch corresponds to a fair or unfair run because the branch describes the evolution of an incomplete process whose behavior is affected by the interactions it has with processes found in other branches of the derivation. However, we can detect (some) unfair branches by looking at the non-deterministic choices they traverse, since choices are made autonomously by processes. To this aim, we introduce the notion of *rank* to estimate the least number of choices a process can possibly make during its lifetime.

► **Definition 14 (rank).** *Let r and s range over the elements of $\mathbb{N}^\infty \stackrel{\text{def}}{=} \mathbb{N} \cup \{\infty\}$ equipped with the expected total order \leq and operation $+$ such that $r + \infty = \infty + r = \infty$. The rank of a process P , written $|P|$, is the least element of \mathbb{N}^∞ such that*

$$\begin{array}{lll} |x \leftrightarrow y| = 0 & |x(y, z).P| = |P| & |\text{case } x(y)\{P, Q\}| = \max\{|P|, |Q|\} \\ |\text{case } x\{| & |\text{in}_i \bar{x}(y).P| = |P| & |P \oplus Q| = 1 + \min\{|P|, |Q|\} \\ |\bar{x}()| = 0 & |\text{rec } \bar{x}(y).P| = |P| & |(x)(P \parallel Q)| = |P| + |Q| \\ |x().P| = |P| & |\text{corec } x(y).P| = |P| & |\bar{x}(y, z)(P \parallel Q)| = |P| + |Q| \end{array}$$

Roughly, the rank of terminated processes is 0, that of processes with a single continuation P coincides with the rank of P , and that of processes spawning two continuations P and Q is the sum of the ranks of P and Q . Then, the rank of a sum input with continuations P and Q is conservatively estimated as the maximum of the ranks of P and Q , since we do not know which one will be taken, whereas the rank of a choice with continuations P and Q is 1 plus the minimum of the ranks of P and Q . If we take *Buyer* and *Seller* from Example 2 we have $|\text{Buyer}\langle x \rangle| = 1$ and $|\text{Seller}\langle x, y \rangle| = 0$. We also have $|\Omega\langle x \rangle| = \infty$. Note that $|P|$ only depends on the structure of P but not on the actual names occurring in P , so it is well and uniquely defined as the least solution of a finite system of equations [10].

► **Definition 15.** *A branch is fair if it traverses finitely many, finitely-ranked choices.*

A finitely-ranked choice is at finite distance from a region of the process in which there are no more choices. An *unfair* branch gets close to such region infinitely often, but systematically avoids entering it. Note that every finite branch is also fair, but there are fair branches that are infinite. For instance, all the infinite branches of the derivation in Equation (1) and the only infinite branch in the derivation for *Seller* $\langle x, y \rangle$ of Example 8 are fair since they do not traverse any finitely-ranked choice. On the contrary, the only infinite branch in the derivation for *Buyer* $\langle x \rangle$ of the Example 8 is unfair since it traverses infinitely many finitely-ranked choices. All fair branches in the same derivation for *Buyer* are finite.

At last we can define our notion of well-typed process.

► **Definition 16 (well-typed process).** *We say that P is well typed in Γ , written $P \Vdash \Gamma$, if the judgment $P \vdash \Gamma$ is derivable and each fair, infinite branch in its derivation is valid.*

Note that Ω is ill typed since the fair, infinite branches in Equation (1) are all invalid. We can now formulate the key properties of well-typed processes, starting from subject reduction.

36:12 Fair Termination in the Linear π -Calculus

► **Theorem 17** (subject reduction). *If $P \Vdash \Gamma$ and $P \rightarrow Q$ then $Q \Vdash \Gamma$.*

All reductions in Table 2 except those for non-deterministic choices correspond to cut-elimination steps in a quasi typing derivation. As an illustration, below is a fragment of derivation tree for two processes exchanging a pair of y and z on channel x .

$$\frac{\frac{\frac{\vdots}{P \vdash \Gamma, y : S} \quad \frac{\vdots}{Q \vdash \Delta, z : T}}{\bar{x}(y, z)(P \parallel Q) \vdash \Gamma, \Delta, x : S \otimes T} [\otimes] \quad \frac{\frac{\vdots}{R \vdash \Gamma', y : S^\perp, z : T^\perp}}{x(y, z).R \vdash \Gamma', x : S^\perp \wp T^\perp} [\wp]}{\frac{\bar{x}(y, z)(P \parallel Q) \parallel x(y, z).R \vdash \Gamma, \Delta, \Gamma'}{(x)(\bar{x}(y, z)(P \parallel Q) \parallel x(y, z).R) \vdash \Gamma, \Delta, \Gamma'} [\text{CUT}]}$$

As the process reduces, the quasi typing derivation is rearranged so that the cut on x is replaced by two cuts on y and z . The resulting quasi typing derivation is shown below.

$$\frac{\frac{\frac{\vdots}{P \vdash \Gamma, y : S} \quad \frac{\frac{\vdots}{Q \vdash \Delta, z : T} \quad \frac{\vdots}{R \vdash \Gamma', y : S^\perp, z : T^\perp}}{(z)(Q \parallel R) \vdash \Delta, \Gamma', y : S^\perp} [\text{CUT}]}{(y)(P \parallel (z)(Q \parallel R)) \vdash \Gamma, \Delta, \Gamma'} [\text{CUT}]}$$

It is also interesting to observe that, when $P \rightarrow Q$, the reduct Q is well typed in the same context as P but its rank may be different. In particular, the rank of Q can be *greater* than the rank of P . Recalling that the rank of a process estimates the number of choices that the process must perform to terminate, the fact that the rank of Q increases means that Q *moves away* from termination instead of getting closer to it (we will see an instance where this phenomenon occurs in Example 23). What really matters is that a well-typed process is weakly terminating. This is the second key property ensured by our type system.

► **Lemma 18** (weak termination). *If $P \Vdash x : \mathbf{1}$ then $P \Rightarrow \bar{x}()$.*

The proof of Lemma 18 is a refinement of the cut elimination property of μMALL^∞ . Essentially, the only new case we have to handle is when a choice $P_1 \oplus P_2$ “emerges” towards the bottom of the typing derivation, meaning that it is no longer guarded by any action. In this case, we reduce the choice to the P_i with smaller rank, which is guaranteed to lay on a fair branch of the derivation. An auxiliary result used in the proof of Lemma 18 is that our type system is a conservative extension of μMALL^∞ .

► **Lemma 19.** *If $P \Vdash x_1 : S_1, \dots, x_n : S_n$ then $\vdash S_1, \dots, S_n$ is derivable in μMALL^∞ .*

The property that well-typed processes can always successfully terminate is a simple consequence of Theorem 17 and Lemma 18.

► **Theorem 20** (soundness). *If $P \Vdash x : \mathbf{1}$ and $P \Rightarrow Q$ then $Q \Rightarrow \bar{x}()$.*

Theorem 20 entails all the good properties we expect from well-typed processes: *failure freedom* (no unguarded sub-process `case $y\{\}$` ever appears), *deadlock freedom* (if the process stops it is terminated), *lock freedom* [26, 32] (every pending action can be completed in finite time) and *junk freedom* (every channel can be depleted). The combination of Theorems 5 and 20 also guarantees the termination of every fair run of the process.

► **Corollary 21** (fair termination). *If $P \Vdash x : \mathbf{1}$ then P is fairly terminating.*

Observe that zero-ranked process do not contain any non-deterministic choice. In that case, every infinite branch in their typing derivation is fair and our validity condition coincides with that of μMALL^∞ . As a consequence, we obtain the following strengthening of Corollary 21:

► **Proposition 22.** *If $P \Vdash x : \mathbf{1}$ and $|P| = 0$ then P is terminating.*

For regular processes (those consisting of finitely many distinct sub-trees, up to renaming of bound names) it is possible to easily adapt the algorithm that decides the validity of a μMALL^∞ proof so that it decides the validity of a πLIN typing derivation. The algorithm is sketched in more detail in the technical report [10, Appendix D].

► **Example 23** (parallel programming). In this example we see a πLIN modeling of a *parallel programming pattern* whereby a *Work* process creates an unbounded number of workers each one dedicated to an independent task and a *Gather* process collects and combines the partial results from the workers. The processes *Work* and *Gather* are defined as follows:

$$\begin{aligned} \text{Work}(x) &= \text{rec } \bar{x}.(\text{complex } \bar{x}.\bar{x}(y)(\bar{y}() \parallel \text{Work}\langle x \rangle) \oplus \text{simple } \bar{x}.\bar{x}()) \\ \text{Gather}(x, z) &= \text{corec } x.\text{case } x\{x(y).y().\text{Gather}\langle x, z \rangle, x().\bar{z}()\} \end{aligned}$$

At each iteration, the *Work* process non-deterministically decides whether the task is **complex** (left hand side of the choice) or **simple** (right hand side of the choice). In the first case, it bifurcates into a new worker, which in the example simply sends a unit on y , and another instance of itself. In the second case it terminates. The *Gather* process joins the results from all the workers before signalling its own termination by sending a unit on z . Note that the number of actions *Gather* has to perform before terminating is unbounded, as it depends on the non-deterministic choices made by *Work*.

Below is a typing derivation for *Work* where $\varphi \stackrel{\text{def}}{=} \mu X.(\mathbf{1} \otimes X) \oplus \mathbf{1}$ and a is an arbitrary atomic address:

$$\frac{\frac{\frac{\frac{\vdots}{\bar{y}() \vdash y : \mathbf{1}} [\mathbf{1}]}{\bar{x}(y)(\bar{y}() \parallel \text{Work}\langle x \rangle) \vdash x : (\mathbf{1} \otimes \varphi)_{ai}} [\otimes]}{\text{complex } \bar{x} \dots \vdash x : ((\mathbf{1} \otimes \varphi) \oplus \mathbf{1})_{ai}} [\oplus]}{\text{simple } \bar{x}.\bar{x}() \vdash x : ((\mathbf{1} \otimes \varphi) \oplus \mathbf{1})_{ai}} [\mathbf{1}, [\oplus]]}{\text{complex } \bar{x}.\bar{x}(y)(\bar{y}() \parallel \text{Work}\langle x \rangle) \oplus \text{simple } \bar{x}.\bar{x}() \vdash x : ((\mathbf{1} \otimes \varphi) \oplus \mathbf{1})_{ai}} [\text{CHOICE}]}{\text{Work}\langle x \rangle \vdash x : \varphi_a} [\mu]$$

Note that the only infinite branch in this derivation is unfair because it traverses infinitely many choices with rank $1 = |\text{Work}\langle x \rangle|$. So, *Work* is well typed.

Concerning *Gather*, we obtain the following typing derivation where $\psi \stackrel{\text{def}}{=} \nu X.(\perp \wp X) \& \perp$:

$$\frac{\frac{\frac{\vdots}{\text{Gather}\langle x, z \rangle \vdash x : \psi_{a^+ilr}, z : \mathbf{1}} [\perp]}{y().\text{Gather}\langle x, z \rangle \vdash x : \psi_{a^+ilr}, y : \perp, z : \mathbf{1}} [\perp]}{x(y).y().\text{Gather}\langle x, z \rangle \vdash x : (\perp \wp \psi)_{a^+il}, z : \mathbf{1}} [\wp]}{\text{case } x\{x(y).y().\text{Gather}\langle x, z \rangle, x().\bar{z}()\} \vdash x : ((\perp \wp \psi) \& \perp)_{a^+i}, z : \mathbf{1}} [\&]}{\text{Gather}\langle x, z \rangle \vdash x : \psi_{a^+}, z : \mathbf{1}} [\nu]$$

Here too there is just one infinite branch, which is fair and supported by the ν -thread $t = (\psi_{a^+}, ((\perp \wp \psi) \& \perp)_{a^+i}, (\perp \wp \psi)_{a^+il}, \psi_{a^+ilr}, \dots)$. Indeed, all the formulas in \bar{t} occur infinitely often and $\min \text{inf}(t) = \psi$ which is a ν -formula. Hence, *Gather* is well typed and so is the composition $(x)(\text{Work}\langle x \rangle \parallel \text{Gather}\langle x, z \rangle)$ in the context $z : \mathbf{1}$. We conclude that the program is fairly terminating, despite the fact that the composition of *Work* and *Gather* may grow arbitrarily large because *Work* may spawn an unbounded number of workers. \square

► **Example 24** (forwarder). In this example we illustrate a deterministic, well-typed process that unfolds a least fixed point infinitely many times. In particular, we consider once again the formulas $\varphi \stackrel{\text{def}}{=} \mu X.X \oplus \mathbf{1}$ and $\psi \stackrel{\text{def}}{=} \nu X.X \& \perp$ and the process Fwd defined by the equation

$$Fwd(x, y) = \text{corec } x.\text{rec } \bar{y}.\text{case } x\{\text{in}_1 \bar{y}.Fwd(x, y), \text{in}_2 \bar{y}.x().\bar{y}()\}$$

which forwards the sequence of messages received from channel x to channel y . We derive

$$\frac{\frac{\frac{\vdots}{Fwd(x, y) \vdash x : \psi_{ail}, y : \varphi_{bil}}{\text{in}_1 \bar{y}.Fwd(x, y) \vdash x : \psi_{ail}, y : (\varphi \oplus \mathbf{1})_{bi}} [\oplus]}{\text{case } x\{\text{in}_1 \bar{y}.Fwd(x, y), \text{in}_2 \bar{y}.x().\bar{y}()\} \vdash x : (\psi \& \perp)_{ai}, y : (\varphi \oplus \mathbf{1})_{bi}} [\&]}{\frac{\frac{\frac{\vdots}{x().\bar{y}() \vdash x : \perp, y : \mathbf{1}}{\text{in}_2 \bar{y}.x().\bar{y}() \vdash x : \perp, y : (\varphi \oplus \mathbf{1})_{bi}} [\oplus]}{\text{case } x\{\text{in}_1 \bar{y}.Fwd(x, y), \text{in}_2 \bar{y}.x().\bar{y}()\} \vdash x : (\psi \& \perp)_{ai}, y : (\varphi \oplus \mathbf{1})_{bi}} [\&]}{Fwd(x, y) \vdash x : \psi_a, y : \varphi_b} [\nu], [\mu]}$$

and observe that $|Fwd(x, y)| = 0$. This typing derivation is valid because the only infinite branch is fair and supported by the ν -thread of ψ_a . Note that $\varphi = \psi^\perp$ and that the derivation proves an instance of [AX]. In general, the axiom is admissible in μMALL^∞ [3]. \lrcorner

► **Example 25** (slot machine). Rank finiteness is not a necessary condition for well typedness. As an example, consider the system $(x)(\text{Player}(x) \parallel \text{Machine}(x, y))$ where

$$\begin{aligned} \text{Player}(x) &= \text{rec } \bar{x}.\text{play } \bar{x}.\text{case } x\{\text{Player}(x), \text{rec } \bar{x}.\text{quit } \bar{x}.\bar{x}()\} \oplus \text{quit } \bar{x}.\bar{x}() \\ \text{Machine}(x, y) &= \text{corec } x.\text{case } x\{\text{win } \bar{x}.\text{Machine}(x, y) \oplus \text{lose } \bar{x}.\text{Machine}(x, y), x().\bar{y}()\} \end{aligned}$$

which models a game between a player and a slot machine. At each round, the player decides whether to **play** or to **quit**. In the first case, the slot machine answers with either **win** or **lose**. If the player **wins**, it also **quits**. Otherwise, it repeats the same behavior. It is possible to show that $\text{Player}(x) \vdash x : \varphi_a$ and $\text{Machine}(x, y) \vdash x : \psi_{a^\perp}, y : \mathbf{1}$ are derivable where $\varphi \stackrel{\text{def}}{=} \mu X.(X \& X) \oplus \mathbf{1}$ and $\psi \stackrel{\text{def}}{=} \nu X.(X \oplus X) \& \perp$ [10]. The only infinite branch in the derivation for Player is unfair since $|\text{Player}(x)| = 1$, so Player is well typed. There are infinitely many branches in the derivation for Machine accounting for all the sequences of **win** and **lose** choices that can be made. Since $|\text{Machine}(x, y)| = \infty$, all these branches are fair but also valid. So, the system as a whole is well typed. \lrcorner

6 Related Work

On account of the known encodings of sessions into the linear π -calculus [27, 11, 38], πLIN belongs to the family of process calculi providing logical foundations to sessions and session types. Some representatives of this family are πDILL [6] and its variant equipped with a circular proof theory [14, 13], CP [40] and μCP [30], among others. There are two main aspects distinguishing πLIN from these calculi. The first one is that these calculi take sessions as a native feature. This fact can be appreciated both at the level of processes, where session endpoints are *linearized* resources that can be used *multiple times* albeit in a sequential way, and also at the level of types, where the interpretation of the $\varphi \otimes \psi$ and $\varphi \wp \psi$ formulas is skewed so as to distinguish the type φ of the message being sent/received on a channel from the type ψ of the channel after the exchange has taken place. In contrast, πLIN adopts a more fundamental communication model based on *linear* channels, and is thus closer to the spirit of the encoding of linear logic proofs into the π -calculus proposed by Bellin and Scott [4] while retaining the same expressiveness of the aforementioned calculi. To some extent, πLIN is also more general than those calculi, since a formula $\varphi \otimes \psi$ may be

interpreted as a protocol that bifurcates into two independent sub-protocols φ and ψ (we have seen an instance of this pattern in Example 23). So, πLIN is natively equipped with the capability of modeling some multiparty interactions, in addition to all of the binary ones. A session-based communication model identical to πLIN , but whose type system is based on intuitionistic rather than classical linear logic, has been presented by DeYoung et al. [15]. In that work, the authors advocate the use of explicit continuations with the purpose of modeling an asynchronous communication semantics and they prove the equivalence between such model and a buffered session semantics. However, they do not draw a connection between the proposed calculus and the linear π -calculus [28] through the encoding of binary sessions [27, 11] and, in the type system, the multiplicative connectives are still interpreted asymmetrically. The second aspect that distinguishes πLIN from the other calculi is its type system, which is still deeply rooted into linear logic and yet it ensures fair termination instead of progress [6, 40, 35], termination [30] or strong progress [14, 13]. Fair termination entails progress, strong progress and lock freedom [26, 32], but at the same time it does not always rule out processes admitting infinite executions. Simply, infinite executions are deemed unrealistic because they are unfair.

Another difference between πLIN and other calculi based on linear logic is that its operational semantics is completely ordinary, in the sense that it does not include commuting conversions, reductions under prefixes, or the swapping of communication prefixes. The cut elimination result of μMALL^∞ , on which the proof of Theorem 20 is based, works by reducing cuts from the bottom of the derivation instead of from the top [3, 16, 2]. As a consequence, it is not necessary to reduce cuts guarded by prefixes or to push cuts deep into the derivation tree to enable key reductions in πLIN processes.

Some previous works [9, 7] have studied type systems ensuring the fair termination of binary and multiparty sessions. Although the enforced property is the same and the communication models of these works are closely related to the one we consider here, the used techniques are quite different and the families of well-typed processes induced by the two type systems are not comparable. One aspect that is shared among all of these works, including the present one, is the use of a *rank* annotation or function that estimates how far a process is from termination. In previous works for session-based communications [9, 7], ranks account for the number of sessions that processes create and the number of times they use subtyping before they terminate. Since ranks are required to be finite, this means that deterministic processes can only create a finite number of sessions and can only use subtyping a finite number of times. As a consequence, the forwarder in Example 24 is ill typed according to those typing disciplines because it applies subtyping (in an implicit way, whenever it sends a tag) an unbounded number of times. At the same time, a “compulsive” variant of the player in Example 25 that keeps playing until it wins is well typed in previous type systems [9, 7] but ill typed in the present one. This difference stems, at least in part, from the fact that the previous type systems support processes defined using general recursion, whereas πLIN ’s type system relies on the duality between recursion and corecursion. A deeper understanding of these differences requires further investigation.

The extension of calculi based on linear logic with non-deterministic features has recently received quite a lot of attention. Rocha and Caires [37] have proposed a session calculus with shared cells and non-deterministic choices that can model mutable state. Their typing rule for non-deterministic choices is the same as our own, but in their calculus choices do not reduce. Rather, they keep track of every possible evolution of a process to be able to prove a confluence result. Qian et al. [35] introduce *coexponentials*, a new pair of modalities that enable the modeling of concurrent clients that compete in order to interact with a shared

server that processes requests sequentially. In this setting, non-determinism arises from the unpredictable order in which different clients are served. Interestingly, the coexponentials are derived by resorting to their semantics in terms of least and greatest fixed points. For this reason, the cut elimination result of μMALL^∞ might be useful to attack the termination proof in their setting.

7 Concluding Remarks

We have studied a conservative extension of μMALL^∞ [3, 16, 2], an infinitary proof system of multiplicative additive linear logic with fixed points, that serves as type system for πLIN , a linear π -calculus with (co)recursive types. Well-typed processes fairly terminate.

One drawback of the proposed type system is that establishing whether a quasi-typed process is also well typed requires a global check on the whole typing derivation. The need for a global check seems to arise commonly in infinitary proof systems [12, 16, 3, 2], so an obvious aspect to investigate is whether the analysis can be localized. A possible source of inspiration for devising a local type system for πLIN might come from the work of Derakhshan and Pfenning [14]. They propose a compositional technique for dealing with infinitary typing derivations in a session calculus, although their type system is limited to the additive fragment of linear logic.

Fair subtyping [31, 33] is a refinement of the standard subtyping relation for session types [19] that preserves fair termination and that plays a key role in our previous type system ensuring fair termination for binary and multiparty sessions [9, 7]. Given the rich literature exploring the connections between linear logic and session types, πLIN and its type system might provide the right framework for investigating the logical foundations of fair subtyping.

References

- 1 Samson Abramsky. Proofs as processes. *Theor. Comput. Sci.*, 135(1):5–9, 1994. doi:10.1016/0304-3975(94)00103-0.
- 2 David Baelde, Amina Doumane, Denis Kuperberg, and Alexis Saurin. Bouncing threads for circular and non-wellfounded proofs. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS, 2022*, 2022. arXiv:2005.08257.
- 3 David Baelde, Amina Doumane, and Alexis Saurin. Infinitary proof theory: the multiplicative additive case. In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic, CSL 2016, August 29 – September 1, 2016, Marseille, France*, volume 62 of *LIPICs*, pages 42:1–42:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.CSL.2016.42.
- 4 Gianluigi Bellin and Philip J. Scott. On the pi-calculus and linear logic. *Theor. Comput. Sci.*, 135(1):11–65, 1994. doi:10.1016/0304-3975(94)00104-9.
- 5 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. doi:10.1007/978-3-319-39570-8_6.
- 6 Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logic propositions as session types. *Math. Struct. Comput. Sci.*, 26(3):367–423, 2016. doi:10.1017/S0960129514000218.
- 7 Luca Ciccone, Francesco Dagnino, and Luca Padovani. Fair termination of multiparty sessions. In Karim Ali and Jan Vitek, editors, *36th European Conference on Object-Oriented Programming, ECOOP 2022, June 6-10, 2022, Berlin, Germany*, volume 222 of *LIPICs*, pages 26:1–26:26. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ECOOP.2022.26.

- 8 Luca Ciccone and Luca Padovani. A dependently typed linear π -calculus in agda. In *PPDP '20: 22nd International Symposium on Principles and Practice of Declarative Programming, Bologna, Italy, 9-10 September, 2020*, pages 8:1–8:14. ACM, 2020. doi:10.1145/3414080.3414109.
- 9 Luca Ciccone and Luca Padovani. Fair termination of binary sessions. *Proc. ACM Program. Lang.*, 6(POPL):1–30, 2022. doi:10.1145/3498666.
- 10 Luca Ciccone and Luca Padovani. An infinitary proof theory of linear logic ensuring fair termination in the linear π -calculus. Technical report, Università di Torino, 2022. URL: <http://www.di.unito.it/~padovani/>.
- 11 Ornela Dardha, Elena Giachino, and Davide Sangiorgi. Session types revisited. *Inf. Comput.*, 256:253–286, 2017. doi:10.1016/j.ic.2017.06.002.
- 12 Christian Dax, Martin Hofmann, and Martin Lange. A proof system for the linear time μ -calculus. In S. Arun-Kumar and Naveen Garg, editors, *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science, 26th International Conference, Kolkata, India, December 13-15, 2006, Proceedings*, volume 4337 of *Lecture Notes in Computer Science*, pages 273–284. Springer, 2006. doi:10.1007/11944836_26.
- 13 Farzaneh Derakhshan. *Session-Typed Recursive Processes and Circular Proofs*. PhD thesis, Carnegie Mellon University, 2021. URL: https://www.andrew.cmu.edu/user/fderakhs/publications/Dissertation_Farzaneh.pdf.
- 14 Farzaneh Derakhshan and Frank Pfenning. Circular proofs as session-typed processes: A local validity condition. *CoRR*, abs/1908.01909, 2019. arXiv:1908.01909.
- 15 Henry DeYoung, Luís Caires, Frank Pfenning, and Bernardo Toninho. Cut reduction in linear logic as asynchronous session-typed communication. In Patrick Cégielski and Arnaud Durand, editors, *Computer Science Logic (CSL'12) – 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, September 3-6, 2012, Fontainebleau, France*, volume 16 of *LIPICs*, pages 228–242. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPICs.CSL.2012.228.
- 16 Amina Doumane. *On the infinitary proof theory of logics with fixed points. (Théorie de la démonstration infinitaire pour les logiques à points fixes)*. PhD thesis, Paris Diderot University, France, 2017. URL: <https://tel.archives-ouvertes.fr/tel-01676953>.
- 17 Nissim Francez. *Fairness*. Texts and Monographs in Computer Science. Springer, 1986. doi:10.1007/978-1-4612-4886-6.
- 18 Philippa Gardner, Cosimo Laneve, and Lucian Wischik. Linear forwarders. *Inf. Comput.*, 205(10):1526–1550, 2007. doi:10.1016/j.ic.2007.01.006.
- 19 Simon J. Gay and Malcolm Hole. Subtyping for session types in the pi calculus. *Acta Informatica*, 42(2-3):191–225, 2005. doi:10.1007/s00236-005-0177-z.
- 20 Simon J. Gay and Vasco Thudichum Vasconcelos. Linear type theory for asynchronous session types. *J. Funct. Program.*, 20(1):19–50, 2010. doi:10.1017/S0956796809990268.
- 21 Orna Grumberg, Nissim Francez, and Shmuel Katz. Fair termination of communicating processes. In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, PODC '84, pages 254–265, New York, NY, USA, 1984. Association for Computing Machinery. doi:10.1145/800222.806752.
- 22 Kohei Honda. Types for dyadic interaction. In Eike Best, editor, *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*, volume 715 of *Lecture Notes in Computer Science*, pages 509–523. Springer, 1993. doi:10.1007/3-540-57208-2_35.
- 23 Kohei Honda, Vasco Thudichum Vasconcelos, and Makoto Kubo. Language primitives and type discipline for structured communication-based programming. In Chris Hankin, editor, *Programming Languages and Systems – ESOP'98, 7th European Symposium on Programming, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'98, Lisbon, Portugal, March 28 – April 4, 1998, Proceedings*, volume 1381 of *Lecture Notes in Computer Science*, pages 122–138. Springer, 1998. doi:10.1007/BFb0053567.

- 24 Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. *J. ACM*, 63(1):9:1–9:67, 2016. doi:10.1145/2827695.
- 25 Hans Hüttel, Ivan Lanese, Vasco T. Vasconcelos, Luís Caires, Marco Carbone, Pierre-Malo Deniérou, Dimitris Mostros, Luca Padovani, António Ravara, Emilio Tuosto, Hugo Torres Vieira, and Gianluigi Zavattaro. Foundations of session types and behavioural contracts. *ACM Comput. Surv.*, 49(1):3:1–3:36, 2016. doi:10.1145/2873052.
- 26 Naoki Kobayashi. A type system for lock-free processes. *Inf. Comput.*, 177(2):122–159, 2002. doi:10.1006/inco.2002.3171.
- 27 Naoki Kobayashi. Type systems for concurrent programs. In *10th Anniversary Colloquium of UNU/IIST*, LNCS 2757, pages 439–453. Springer, 2002. Extended version at <http://www.kb.ecei.tohoku.ac.jp/~koba/papers/tutorial-type-extended.pdf>. doi:10.1007/978-3-540-40007-3_26.
- 28 Naoki Kobayashi, Benjamin C. Pierce, and David N. Turner. Linearity and the pi-calculus. *ACM Trans. Program. Lang. Syst.*, 21(5):914–947, 1999. doi:10.1145/330249.330251.
- 29 M.Z. Kwiatkowska. Survey of fairness notions. *Information and Software Technology*, 31(7):371–386, 1989. doi:10.1016/0950-5849(89)90159-6.
- 30 Sam Lindley and J. Garrett Morris. Talking bananas: structural recursion for session types. In Jacques Garrigue, Gabriele Keller, and Eijiro Sumii, editors, *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016, Nara, Japan, September 18-22, 2016*, pages 434–447. ACM, 2016. doi:10.1145/2951913.2951921.
- 31 Luca Padovani. Fair subtyping for open session types. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming – 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II*, volume 7966 of *Lecture Notes in Computer Science*, pages 373–384. Springer, 2013. doi:10.1007/978-3-642-39212-2_34.
- 32 Luca Padovani. Deadlock and lock freedom in the linear π -calculus. In Thomas A. Henzinger and Dale Miller, editors, *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, Vienna, Austria, July 14–18, 2014*, pages 72:1–72:10. ACM, 2014. doi:10.1145/2603088.2603116.
- 33 Luca Padovani. Fair subtyping for multi-party session types. *Math. Struct. Comput. Sci.*, 26(3):424–464, 2016. doi:10.1017/S096012951400022X.
- 34 Luca Padovani. A simple library implementation of binary sessions. *J. Funct. Program.*, 27:e4, 2017. doi:10.1017/S0956796816000289.
- 35 Zesen Qian, G. A. Kavvos, and Lars Birkedal. Client-server sessions in linear logic. *Proc. ACM Program. Lang.*, 5(ICFP):1–31, 2021. doi:10.1145/3473567.
- 36 Jean-Pierre Queille and Joseph Sifakis. Fairness and related properties in transition systems – A temporal logic to deal with fairness. *Acta Informatica*, 19:195–220, 1983. doi:10.1007/BF00265555.
- 37 Pedro Rocha and Luís Caires. Propositions-as-types and shared state. *Proc. ACM Program. Lang.*, 5(ICFP):1–30, 2021. doi:10.1145/3473584.
- 38 Alceste Scalas, Ornela Dardha, Raymond Hu, and Nobuko Yoshida. A linear decomposition of multiparty sessions for safe distributed programming (artifact). *Dagstuhl Artifacts Ser.*, 3(2):03:1–03:2, 2017. doi:10.4230/DARTS.3.2.3.
- 39 Rob van Glabbeek and Peter Höfner. Progress, justness, and fairness. *ACM Comput. Surv.*, 52(4):69:1–69:38, 2019. doi:10.1145/3329125.
- 40 Philip Wadler. Propositions as sessions. *J. Funct. Program.*, 24(2-3):384–418, 2014. doi:10.1017/S095679681400001X.