Cut Reduction in Linear Logic as Asynchronous Session-Typed Communication*

Henry DeYoung[1], Luís Caires[2], Frank Pfenning[1], and Bernardo Toninho[1,2]

1 **Computer Science Department, Carnegie Mellon University**
Pittsburgh, PA, USA
{hdeyoung, fp, btoninho}@cs.cmu.edu

2 **Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa**
Lisboa, Portugal
luis.caires@di.fct.unl.pt

Abstract

Prior work has shown that intuitionistic linear logic can be seen as a session-type discipline for the π-calculus, where cut reduction in the sequent calculus corresponds to synchronous process reductions. In this paper, we exhibit a new process assignment from the *asynchronous*, polyadic π-calculus to exactly the same proof rules. Proof-theoretically, the difference between these interpretations can be understood through permutations of inference rules that preserve observational equivalence of closed processes in the synchronous case. We also show that, under this new asynchronous interpretation, cut reductions correspond to a natural asynchronous buffered session semantics, where each session is allocated a separate communication buffer.

1998 ACM Subject Classification F.3.2 Semantics of Programming Languages

Keywords and phrases linear logic, cut reduction, asynchronous π-calculus, session types

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

Ever since linear logic was originally proposed, researchers have been discovering and exploring its deep and perhaps surprising connections with concurrency in a variety of ways. Girard himself first sketched a connection of linear logic with concurrency, by giving a high-level pattern of communication that manifested itself in proof nets [13]. Others expanded upon this with further models based on proof nets and related structures, e.g., [1, 6, 5, 16]. In a different vein, two of the present authors recently developed a Curry-Howard interpretation of intuitionistic linear logic [8], where propositions are interpreted as session types [15, 14], sequent calculus proofs are interpreted as π-calculus processes, and proof reduction during cut elimination is interpreted as synchronous communication.

A natural follow-up question to this work is whether a Curry-Howard correspondence between linear logic and an *asynchronous* process calculus can be established. An answer is relevant both to the concurrency theorist and the logician. For the concurrency theorist, asynchronous communication is a more realistic (and challenging) model for concurrency, and so being able to establish properties of asynchronous processes by static typing is of great interest. For the logician, asynchrony can be seen as eliminating some of the "bureaucracy of

* Support was provided by the Fundação para a Ciência e a Tecnologia through the Carnegie Mellon Portugal Program, under grants SFRH/BD/33763/2009 and INTERFACES NGN-44/2009, and CITI.

Conference title on which this volume is based on.
Editors: Billy Editor, Bill Editors; pp. 1–15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

syntax," so that the order in which certain proof rules are applied no longer imposes artificial sequentiality.

Our novel interpretation assigns processes from the asynchronous polyadic π-calculus[1] [7, 17] to a sequent calculus formulation of DILL [3, 10] (the same proof rules as [8]), and cut reductions to asynchronous communication, as we detail in Section 2. Moreover, in Section 3 we formally determine the relationship between the prior synchronous interpretation with our asynchronous one. We show that, proof-theoretically, the fundamental difference between the two is that a class of commuting conversions that in the synchronous interpretation corresponded to only observational equivalences, now map to natural *structural* equivalences in the asynchronous π-calculus. Finally, in Section 4, we relate our asynchronous interpretation to buffered communication.

2 Linear logic as asynchronous session-typed communication

2.1 Judgmental principles

Because processes offer services along designated channels, our basic session-typing judgment is $P :: x{:}A$, meaning "process P offers service A along channel x." However, to provide new services, most processes must themselves rely on services offered by other processes. Thus, more generally, we use the hypothetical judgment

$$x_1{:}A_1, \ldots, x_n{:}A_n \vdash P :: x{:}A\,,$$

meaning "Using services A_i that are assumed to be provided along channels x_i, process P offers service A along channel x." The channels x_i and x must all be distinct, and are binding occurrences with scope over the process P. (We use the metavariable Δ and its decorated variants to stand for an arbitrary context of services.)

When two processes interact, their state changes; one now offers, and the other uses, the continuation of the initial service. Due to this change of state, our hypothetical judgment can be seen as an annotation of the intuitionistic linear sequent $A_1, \ldots, A_n \vdash A$. The context of services satisfies neither weakening nor contraction. It does satisfy exchange, however, because antecedents are uniquely labeled. Our process interpretation also handles persistent antecedents, but we postpone their introduction until Section 2.6 to keep the overhead of initial exposition lower.

The sequent calculus cut and identity rules serve to clarify the relationship between offering and using a service.

Cut as composition. In the sequent calculus, the cut rule composes a proof of lemma A with its use in the proof of theorem C:

$$\frac{\Delta \vdash A \quad \Delta', A \vdash C}{\Delta, \Delta' \vdash C}\ \mathsf{cut}$$

Because proofs should correspond to processes, this reading suggests that the process interpretation of the cut rule should compose a process that offers service A with one that uses service A. Stated differently, an offer satisfies a use. Therefore, we annotate cut as

$$\frac{\Delta \vdash P :: x{:}A \quad \Delta', x{:}A \vdash Q :: z{:}C}{\Delta, \Delta' \vdash (\nu x)(P \mid Q) :: z{:}C}\ \mathsf{cut}$$

[1] Our interpretation in fact uses only niladic, monadic, and dyadic processes, not general polyadic communication; for brevity, however, we prefer to retain 'polyadic' as the collective term.

The process $(\nu x)(P \mid Q)$ allows to execute in parallel, as indicated by $P \mid Q$, and interact along the private channel x, as indicated by the name restriction (νx). (The occurrence of x in the name restriction (νx) is a binding occurrence, and is therefore subject to renaming.)

Identity as forwarding. The identity rule uses an antecedent to construct a proof directly:

$$\frac{}{A \vdash A}\ \mathsf{id}$$

This suggests an interpretation of id as a forwarding process, $[y \leftrightarrow x]$:

$$\frac{}{y{:}A \vdash [y \leftrightarrow x] :: x{:}A}\ \mathsf{id}$$

The process $[y \leftrightarrow x]$ forwards messages received along channel x further on to channel y, and vice versa, so that it offers A along x by directly using the service A that is available from y. Stated differently, a use is one way to fulfill an offer.

2.2 Implication as input

In our process interpretation, the linear logical connectives correspond to various basic forms of service. We adopt a verificationist perspective: the sequent calculus right rules will define what it means to offer a service, whereas the left rules show how to use a service.

Consider linear implication, written $A \multimap B$, and recall its right and left rules:

$$\frac{\Delta, A \vdash B}{\Delta \vdash A \multimap B}\ \multimap R \qquad \frac{\Delta_1' \vdash A \quad \Delta_2', B \vdash C}{\Delta_1', \Delta_2', A \multimap B \vdash C}\ \multimap L$$

The right rule says that $A \multimap B$ is provable if B is provable using a proof of A. Correspondingly, a process that offers service $A \multimap B$ should first input a channel offering service A and then continue the session by using this service to offer service B. Conversely, a process that uses service $A \multimap B$ should behave in a complementary way: the client must first output a new channel offering service A and then continue the session by using service B.

Based on this intuition, a first attempt at an asynchronous process assignment might be:

$$\frac{\Delta, y{:}A \vdash P :: x{:}B}{\Delta \vdash x(y).P :: x{:}A \multimap B}\ \multimap R? \qquad \frac{\Delta_1' \vdash Q_1 :: y{:}A \quad \Delta_2', x{:}B \vdash Q_2 :: z{:}C}{\Delta_1', \Delta_2', x{:}A \multimap B \vdash (\nu y)(\overline{x}\langle y\rangle \mid Q_1 \mid Q_2) :: z{:}C}\ \multimap L?$$

The syntax $x(y).P$ denotes a blocking input along channel x that guards process P; here y is a bound name and stands for the channel that will be received by the input. The syntax $\overline{x}\langle y\rangle$ denotes an asynchronous output of y along channel x; we often think of it as a message y somewhere in transit to x. Note that, in their premises, both typing rules reuse the session channel of type $A \multimap B$, namely x, as the channel for the session continuation at type B.

Unfortunately, there are two serious problems with this assignment. First, it leaves outputs along the channel x unordered, violating the session contract. As an example, consider the alleged typing derivation

$$\frac{\Delta_1 \vdash P_1 :: y_1{:}A_1 \quad \dfrac{\Delta_2 \vdash P_2 :: y_2{:}A_2 \quad \Delta', x{:}B \vdash Q :: z{:}C}{\Delta_2, \Delta', x{:}A_2 \multimap B \vdash (\nu y_2)(\overline{x}\langle y_2\rangle \mid P_2 \mid Q) :: z{:}C}\ \multimap L?}{\Delta_1, \Delta_2, \Delta', x{:}A_1 \multimap (A_2 \multimap B) \vdash (\nu y_1)\big(\overline{x}\langle y_1\rangle \mid P_1 \mid (\nu y_2)(\overline{x}\langle y_2\rangle \mid P_2 \mid Q)\big) :: z{:}C}\ \multimap L?$$

According to the contract imposed by the session type $x{:}A_1 \multimap (A_2 \multimap B)$, a process listening on channel x should expect to receive an A_1 and then, only later, an A_2. But, under the operational semantics of the asynchronous π-calculus, a process listening on x nondeterministically receives *either* $y_1{:}A_1$ or $y_2{:}A_2$ when composed with the above process.

Second, it is possible that the output will be misdirected to the session's continuation. Because the $\multimap L$ rule types the continuation process Q_2 with $\Delta_2', x{:}B \vdash Q_2 :: z{:}C$, the process Q_2 may contain an input along channel x. Because the asynchronous output $\overline{x}\langle y\rangle$ is in parallel with Q_2, it is possible that such an input might unintentionally receive $\overline{x}\langle y\rangle$.

Fortunately, there is a single, elegant fix for both problems: rather than using the same channel for the continuation, the session should continue along a fresh channel. Thus, the left rule, $\multimap L$, becomes

$$\frac{\Delta, y{:}A \vdash P :: x'{:}B}{\Delta \vdash x(y,x').P :: x{:}A \multimap B}\ \multimap R \qquad \frac{\Delta_1' \vdash Q_1 :: y{:}A \quad \Delta_2', x'{:}B \vdash Q_2 :: z{:}C}{\Delta_1', \Delta_2', x{:}A \multimap B \vdash (\nu y)(\nu x')(\overline{x}\langle y,x'\rangle \mid Q_1 \mid Q_2) :: z{:}C}\ \multimap L$$

It is crucial that the $\multimap L$ process sends both y and x', as represented by the dyadic output $\overline{x}\langle y,x'\rangle$. If the process sent only y, then its session partner would not know where to rendezvous for the session continuation. Accordingly, the right rule is a matching dyadic input. (The names y and x' are bound with scope over P in the input process $x(y,x').P$.)

By using a fresh channel for the continuation, both problems are resolved. First, outputs are now ordered. Since the $\multimap L$ rule types the continuation process as $\Delta_2', x'{:}B \vdash Q_2 :: z{:}C$, a subsequent output in Q_2 will occur along channel x', not x. Because of the name restriction $(\nu x')$, the channel x' is unknown outside of this process. Thus, no other process can be listening on x' until it receives the output $\overline{x}\langle y,x'\rangle$ and learns of the new channel x', thereby imposing an order on outputs within a given session.

Pictorially, we might represent this ordering of outputs within a session as the sequence $\overline{x}\langle y_1,x'\rangle$, $\overline{x'}\langle y_2,x''\rangle$, $\overline{x''}\langle y_3,x'''\rangle$, ... Because the typing discipline ensures that channels $x', x'', x''', \ldots$ are not used elsewhere, this suggests a reading of well-typed processes as using explicit communication buffers, such as the input buffer $x\langle y_1,y_2,y_3,\ldots\rangle$ at endpoint x. This intuition will be made precise in Section 4.

Second, the problem of misdirected outputs is also resolved by using a fresh channel for the session continuation. The $\multimap L$ rule does not allow the previous session channel, x, to appear in the continuation process Q_2. Therefore, Q_2 will not contain inputs along x, precluding it from ever mistakenly receiving the output $\overline{x}\langle y,x'\rangle$.

Cut reduction as communication. In the linear sequent calculus, the principal cut reduction for linear implication is a local check on the coherence of the $\multimap R$ and $\multimap L$ rules:

$$\frac{\dfrac{\Delta, A \vdash B}{\Delta \vdash A \multimap B}\ \multimap R \quad \dfrac{\Delta_1' \vdash A \quad \Delta_2', B \vdash C}{\Delta_1', \Delta_2', A \multimap B \vdash C}\ \multimap L}{\Delta, \Delta_1', \Delta_2' \vdash C}\ \mathsf{cut} \quad \longrightarrow \quad \frac{\dfrac{\Delta_1' \vdash A \quad \Delta, A \vdash B}{\Delta, \Delta_1' \vdash B}\ \mathsf{cut} \qquad \Delta_2', B \vdash C}{\Delta, \Delta_1', \Delta_2' \vdash C}\ \mathsf{cut}$$

Under the process interpretation, cut reduction is asynchronous session-typed communication. When annotated according to the process interpretation, the above principal cut reduction for linear implication yields the process reduction

$$(\nu x)\big(x(y,x').P \mid (\nu y)(\nu x')(\overline{x}\langle y,x'\rangle \mid Q_1 \mid Q_2)\big) \longrightarrow (\nu x')\big((\nu y)(Q_1 \mid P) \mid Q_2\big)\,.$$

Modulo the structural congruences of the π-calculus (including α-renaming of bound names), this is an instance of the standard asynchronous polyadic π-calculus process reduction that drives asynchronous communication: $x(w,z).P \mid \overline{x}\langle y,x'\rangle \longrightarrow P\{y/w,x'/z\}$. This justifies our claim that cut reduction is asynchronous session-typed communication.

It is also possible to give a computational interpretation of identity expansion, the act of reducing uses of the id rule at compound types to larger proofs that appeal to the id rule at smaller types. We do not pursue it in this paper because it is not germane to our study of cut reduction as communication and commuting conversions as the basis of asynchrony. For the details of identity expansion in the synchronous case, see [9].

2.3 Multiplicative conjunction as output

Even in the intuitionistic linear sequent calculus, there is a strong flavor of duality between linear implication and multiplicative conjunction. This duality should similarly extend to the process interpretation: just as a process of type $A \multimap B$ offers an input of an A and then behaves as B, a process of type $A \otimes B$ should offer an output of an A and then behave as B.

Thus, we arrive at the following process assignment for the usual right and left rules.

$$\frac{\Delta_1 \vdash P_1 :: y{:}A \quad \Delta_2 \vdash P_2 :: x'{:}B}{\Delta_1, \Delta_2 \vdash (\nu y)(\nu x')(\overline{x}\langle y, x'\rangle \mid P_1 \mid P_2) :: x{:}A \otimes B}\ \otimes R \qquad \frac{\Delta', y{:}A, x'{:}B \vdash Q :: z{:}C}{\Delta', x{:}A \otimes B \vdash x(y, x').Q :: z{:}C}\ \otimes L$$

Again, notice the use of a new channel, x', for the session continuation at type B. If we tried to reuse the original session channel, x, then we would again face the problems of unordered and misdirected outputs that plagued our first, failed process assignment for implication. We can verify that the $\otimes R$ and $\otimes L$ rules fit together by checking the principal cut reduction.

Cut reduction as communication. The principal cut reduction for $A \otimes B$ is

$$\frac{\dfrac{\Delta_1 \vdash A \quad \Delta_2 \vdash B}{\Delta_1, \Delta_2 \vdash A \otimes B}\ \otimes R \quad \dfrac{\Delta', A, B \vdash C}{\Delta', A \otimes B \vdash C}\ \otimes L}{\Delta_1, \Delta_2, \Delta' \vdash C}\ \mathsf{cut} \longrightarrow \frac{\Delta_2 \vdash B \quad \dfrac{\Delta_1 \vdash A \quad \Delta', A, B \vdash C}{\Delta_1, \Delta', B \vdash C}\ \mathsf{cut}}{\Delta_1, \Delta_2, \Delta' \vdash C}\ \mathsf{cut}$$

The same reduction can be carried out under the process interpretation, yielding

$$(\nu x)\big((\nu y)(\nu x')(\overline{x}\langle y, x'\rangle \mid P_1 \mid P_2) \mid x(y, x').Q\big) \longrightarrow (\nu x')\big(P_2 \mid (\nu y)(P_1 \mid Q)\big)\,.$$

Once again, modulo structural congruence, this is an instance of the standard asynchronous polyadic communication rule.

2.4 Multiplicative unit as termination

Because it is the unit of $\otimes$, it is often helpful to view the proposition $\mathbf{1}$ as the nullary form of $\otimes$. For instance, the inference rules for $\mathbf{1}$ are nullary versions of the rules for $A \otimes B$. We can extend this to our process interpretation:

$$\frac{}{\cdot \vdash \overline{x}\langle\rangle :: x{:}\mathbf{1}}\ \mathbf{1}R \qquad \frac{\Delta' \vdash Q :: z{:}C}{\Delta', x{:}\mathbf{1} \vdash x().Q :: z{:}C}\ \mathbf{1}L$$

The right rule outputs an empty message and has no continuation; the left rule inputs the empty message. Because the right rule has no continuation, the empty message serves as a session termination signal: the process will not offer any further service.

As the left rule is given, a process using a terminated session must block until it receives the termination signal, because the prefix $x()$ guards the continuation Q. We can enable more parallelism by modifying the left rule:

$$\frac{\Delta' \vdash Q :: z{:}C}{\Delta', x{:}\mathbf{1} \vdash x().\mathbf{0} \mid Q :: z{:}C}\ \mathbf{1}L$$

The left rule's continuation, Q, can now run in parallel while waiting to receive the termination signal. The $\mathbf{1}L$ rule is no longer an exact nullary version of the $\otimes L$ rule, but the process assignment is still in bijective correspondence with the proof rules.

Cut reduction as communication. The principal cut reduction at type $\mathbf{1}$ is

$$\frac{\dfrac{}{\cdot \vdash \mathbf{1}}\ \mathbf{1}R \quad \dfrac{\Delta' \vdash C}{\Delta', \mathbf{1} \vdash C}\ \mathbf{1}L}{\Delta' \vdash C}\ \mathsf{cut} \longrightarrow \Delta' \vdash C$$

When annotated according to the process interpretation, we can extract the process reduction $(\nu x)\big(\overline{x}\langle\rangle \mid (x().\mathbf{0} \mid Q)\big) \longrightarrow Q$, which, modulo structural congruences, is an instance of the asynchronous π-calculus reduction that matches a nullary output with a nullary input.

2.5 Additive conjunction and disjunction as choice

Processes should be able to offer the client a choice between two services, A and B. The client selects one of the services and then uses the selected service. This type of external choice corresponds to the additive conjunction $A \,\&\, B$. We can assign processes to the rules:

$$\frac{\Delta \vdash P_1 :: x_1'{:}A \quad \Delta \vdash P_2 :: x_2'{:}B}{\Delta \vdash x.\mathsf{case}((x_1').P_1, (x_2').P_2) :: x{:}A \,\&\, B}\ \&R$$

$$\frac{\Delta', x_1'{:}A \vdash Q :: z{:}C}{\Delta', x{:}A \,\&\, B \vdash (\nu x_1')(x.\mathsf{inl}\langle x_1'\rangle \mid Q) :: z{:}C}\ \&L_1 \qquad \frac{\Delta', x_2'{:}B \vdash Q :: z{:}C}{\Delta', x{:}A \,\&\, B \vdash (\nu x_2')(x.\mathsf{inr}\langle x_2'\rangle \mid Q) :: z{:}C}\ \&L_2$$

In the left rules, the client asynchronously sends his selection (either inl or inr) and a new channel at which the session should continue. In the right rule, the server must be prepared for either selection; it behaves like a case, waiting for a client's selection and then continuing accordingly.

The principal cut reductions and process reductions match: the process reductions are

$$(\nu x)\big(x.\mathsf{case}((x_1').P_1, (x_2').P_2) \mid (\nu x_1')(x.\mathsf{inl}\langle x_1'\rangle \mid Q)\big) \longrightarrow (\nu x_1')(P_1 \mid Q)$$
$$(\nu x)\big(x.\mathsf{case}((x_1').P_1, (x_2').P_2) \mid (\nu x_2')(x.\mathsf{inr}\langle x_2'\rangle \mid Q)\big) \longrightarrow (\nu x_2')(P_2 \mid Q)\,.$$

External choice is dual to internal choice, where a process offers one of two possible services with the choice at its own discretion. Because $A \& B$ is dual to the additive disjunction $A \oplus B$ in linear logic, $A \oplus B$ should be interpreted as internal choice and uses the same process constructs in a dual manner. Due to space constraints, we omit the details here.

2.6 Exponential as persistent service

Thus far, we have focused on the purely linear fragment of intuitionistic linear logic, but we can also give an asynchronous process interpretation of the 'of course!' exponential. In the judgmental formulation of intuitionistic linear logic, the reusable antecedents provided by the 'of course!' exponential are expressed with a new judgment, $A\ \mathsf{valid}$, that is subject to weakening, contraction, and exchange. To streamline notation, validity antecedents are usually written in a separate zone of the sequent, as in dual intuitionistic linear logic [3, 10]. With the process annotations added, the sequent becomes

$$u_1{:}B_1, \ldots, u_m{:}B_m; x_1{:}A_1, \ldots, x_n{:}A_n \vdash P :: x{:}A\,,$$

where $u_1{:}B_1, \ldots, u_m{:}B_m$ are the reusable, validity antecedents. We use the metavariable Γ to stand for an arbitrary context of validity antecedents. To match the new form for sequents, all of the previously presented inference rules are extended to include a context Γ in the conclusion and all premises.

2.6.1 Judgmental principles

A proposition A is valid if, and only if, A is true without linear antecedents. There are two new judgmental rules: a cut principle for validity and a rule relating validity to linear truth.

$$\frac{\Gamma; \cdot \vdash A \quad \Gamma, A; \Delta' \vdash C}{\Gamma; \Delta' \vdash C}\ \mathsf{cut!} \qquad \frac{\Gamma, A; \Delta', A \vdash C}{\Gamma, A; \Delta' \vdash C}\ \mathsf{copy}$$

What process interpretation should we give to validity and its cut! and copy rules? Because validity antecedents persist throughout a proof we will interpret $u{:}A$ as a server that persistently provides service A. Specifically:

$$\frac{\Gamma;\cdot \vdash P :: y{:}A \quad \Gamma, u{:}A; \Delta' \vdash Q :: z{:}C}{\Gamma; \Delta' \vdash (\nu u)(!u(y).P \mid Q) :: z{:}C}\ \mathsf{cut!} \qquad \frac{\Gamma, u{:}A; \Delta', x{:}A \vdash Q :: z{:}C}{\Gamma, u{:}A; \Delta' \vdash (\nu x)(\overline{u}\langle x\rangle \mid Q) :: z{:}C}\ \mathsf{copy}$$

The cut! rule shows that a persistent offer of service A is made by the replicated input $!u(y).P$. According to the copy rule, the client process, Q, can obtain service A by asynchronously sending the server, u, a new channel; the server spawns a copy of service A at that channel, and the server continues to be available for future requests.

Note that, in contrast with all previous rules, the copy rule's premise does *not* use a renamed persistent channel in the continuation. From an operational perspective, this is because persistent servers do not directly participate in long-lived sessions with clients. Instead, they just receive individual messages from various clients and spawn linear sessions to do the real work. Alternatively, from a proof-theoretic perspective, this is because there is a commuting conversion between *any* adjacent copy inferences.

Cut reduction as communication. The server's act of spawning a copy of service A is reflected in the process interpretation of the cut reduction that arises when cut! meets copy. When processes annotate the cut reduction

$$\dfrac{\Gamma;\cdot \vdash A \quad \dfrac{\dfrac{\Gamma, A; \Delta', A \vdash C}{\Gamma, A; \Delta' \vdash C}\ \mathsf{copy}}{\Gamma; \Delta' \vdash C}\ \mathsf{cut!}}{} \longrightarrow \dfrac{\Gamma;\cdot \vdash A \quad \dfrac{\Gamma;\cdot \vdash A \quad \Gamma, A; \Delta', A \vdash C}{\Gamma; \Delta', A \vdash C}\ \mathsf{cut!}}{\Gamma; \Delta' \vdash C}\ \mathsf{cut}$$

we obtain the following process reduction, which matches an output with a replicated input.

$$(\nu u)\big(!u(y).P \mid (\nu x)(\overline{u}\langle x\rangle \mid Q)\big) \longrightarrow (\nu x)\big(P\{x/y\} \mid (\nu u)(!u(y).P \mid Q)\big)$$

2.6.2 Right and left rules

In a judgmental formulation of the linear sequent calculus, the right and left rules for the 'of course!' connective, written $!A$, are:

$$\frac{\Gamma;\cdot \vdash A}{\Gamma;\cdot \vdash !A}\ !R \qquad \frac{\Gamma, A; \Delta' \vdash C}{\Gamma; \Delta', !A \vdash C}\ !L$$

How does $!A$ relate to validity as persistent service? Essentially, we will interpret a service $!A$ as one that creates a persistent server that offers A. The process assignment that we use is

$$\frac{\Gamma;\cdot \vdash P :: y{:}A}{\Gamma;\cdot \vdash (\nu u)(\overline{x}\langle u\rangle \mid !u(y).P) :: x{:}!A}\ !R \qquad \frac{\Gamma, u{:}A; \Delta' \vdash Q :: z{:}C}{\Gamma; \Delta', x{:}!A \vdash x(u).Q :: z{:}C}\ !L$$

The right rule says that a process offering $!A$ along x first chooses a new, persistent name, u, for itself and registers that channel by sending it to its session partner. The process then persistently provides service A by offering a replicated input at u. Conversely, the left rule says that a process that uses $!A$ must input a persistent channel, u, and may thereafter treat u as the name of a persistent server offering A.

Our process interpretation of the $!R$ and $!L$ rules departs significantly from the prior synchronous interpretations [8, 9], in ways that are orthogonal to asynchrony. Previously, the $!R$ rule was interpreted as a replicated input along a linear channel and the $!L$ rule was

interpreted as either an implicit [8] or explicit [9] substitution. We contend that our process assignment is proof-theoretically more pleasing: now, *all* of the non-invertible right and left rules [2] ($\multimap L$, $\otimes R$, $\& L_i$, $\oplus R_i$, and $!R$) have an output flavor, and *all* of the invertible right and left rules ($\multimap R$, $\otimes L$, $\& R$, $\oplus L$, and $!L$) have an input flavor.

Cut reduction as communication. Once again, the principal cut reduction corresponds to a process reduction. The principal cut reduction at type $!A$ is

$$\dfrac{\dfrac{\Gamma;\cdot \vdash A}{\Gamma;\cdot \vdash !A}\,!R \quad \dfrac{\Gamma, A; \Delta' \vdash C}{\Gamma; \Delta', !A \vdash C}\,!L}{\Gamma; \Delta' \vdash C}\,\mathsf{cut} \longrightarrow \dfrac{\Gamma;\cdot \vdash A \quad \Gamma, A; \Delta' \vdash C}{\Gamma; \Delta' \vdash C}\,\mathsf{cut!}$$

When this cut reduction is annotated with processes, we obtain the process reduction

$$(\nu x)\big((\nu u)(\overline{x}\langle u\rangle \mid !u(y).P) \mid x(u).Q\big) \longrightarrow (\nu u)(!u(y).P \mid Q)\,.$$

Thus, cut reduction at $!A$ corresponds to registering a server's persistent name with its client.

3 Relationship between synchronous and asynchronous process interpretations

Prior work has shown that the intuitionistic linear sequent calculus can be seen as a session-type discipline for the *synchronous* π-calculus [8, 9]. In the previous section, we presented a new process assignment from the *asynchronous*, polyadic π-calculus to exactly the same proof rules. This section serves to make precise the claim that, proof-theoretically, the difference between these interpretations lies in the commuting conversions that are permitted.

3.1 A synchronous, polyadic process interpretation

The first step in making our claim precise is to reconsider the synchronous process interpretation from [8, 9]. There, all outputs were represented as prefixes guarding a continuation process, as is standard in the synchronous π-calculus. For example, the assignment for the $\otimes R$ rule was a synchronous monadic output and the $\otimes L$ rule was a monadic input:

$$\dfrac{\Gamma; \Delta_1 \vdash P_1 :: y{:}A \quad \Gamma; \Delta_2 \vdash P_2 :: x{:}B}{\Gamma; \Delta_1, \Delta_2 \vdash (\nu y)\overline{x}\langle y\rangle.(P_1 \mid P_2) :: x{:}A \otimes B}\,\otimes R \qquad \dfrac{\Gamma; \Delta', y{:}A, x{:}B \vdash Q :: z{:}C}{\Gamma; \Delta', x{:}A \otimes B \vdash x(y).Q :: z{:}C}\,\otimes L$$

These processes reuse the session channel x as the channel for the session's continuation. There is no possibility of unordered or misdirected outputs here because P_1 and P_2 may not execute until the output guard, $\overline{x}\langle y\rangle$, synchronizes with another input process, $x(y).Q$.

Instead of relying on this implicit convention, we could modify the synchronous process assignment to be explicit about reusing the session channel. The $\otimes R$ and $\otimes L$ rules would thus become dyadic outputs and inputs, respectively, with the output explicitly transmitting x as a channel to be used for the session continuation. But, in fact, once dyadic outputs and inputs are used, there is no technical advantage to reusing the session channel, and we may as well use a fresh channel for the session continuation. Figure 1 presents a polyadic synchronous process interpretation in this style.

There is a very strong operational equivalence between this polyadic interpretation and the monadic synchronous interpretation from [9] (there is also a correspondence with [8], if we match their rules for $\mathbf{1}$). For example, consider the principal cut reduction at type $A \otimes B$. Under the monadic synchronous process assignment, it corresponds to

$$(\nu x)\big((\nu y)\overline{x}\langle y\rangle.(P_1 \mid P_2) \mid x(y).Q\big) \longrightarrow (\nu x)\big(P_2 \mid (\nu y)(P_1 \mid Q)\big)\,.$$

$$\frac{}{\Gamma; y{:}A \vdash [y \leftrightarrow x] :: x{:}A}\ \mathsf{id} \qquad \frac{\Gamma; \Delta \vdash P :: x{:}A \quad \Gamma; \Delta', x{:}A \vdash Q :: z{:}C}{\Gamma; \Delta, \Delta' \vdash (\nu x)(P \mid Q) :: z{:}C}\ \mathsf{cut}$$

$$\frac{\Gamma; \cdot \vdash P :: y{:}A \quad \Gamma, u{:}A; \Delta' \vdash Q :: z{:}C}{\Gamma; \Delta' \vdash (\nu u)(!u(y).P \mid Q) :: z{:}C}\ \mathsf{cut!} \qquad \frac{\Gamma, u{:}A; \Delta', x{:}A \vdash Q :: z{:}C}{\Gamma, u{:}A; \Delta' \vdash (\nu x)\overline{u}\langle x\rangle.Q :: z{:}C}\ \mathsf{copy}$$

$$\frac{\Gamma; \Delta, y{:}A \vdash P :: x'{:}B}{\Gamma; \Delta \vdash x(y, x').P :: x{:}A \multimap B}\ \multimap R \qquad \frac{\Gamma; \Delta_1' \vdash Q_1 :: y{:}A \quad \Gamma; \Delta_2', x'{:}B \vdash Q_2 :: z{:}C}{\Gamma; \Delta_1', \Delta_2', x{:}A \multimap B \vdash (\nu y)(\nu x')\overline{x}\langle y, x'\rangle.(Q_1 \mid Q_2) :: z{:}C}\ \multimap L$$

$$\frac{\Gamma; \Delta_1 \vdash P_1 :: y{:}A \quad \Gamma; \Delta_2 \vdash P_2 :: x'{:}B}{\Gamma; \Delta_1, \Delta_2 \vdash (\nu y)(\nu x')\overline{x}\langle y, x'\rangle.(P_1 \mid P_2) :: x{:}A \otimes B}\ \otimes R \qquad \frac{\Gamma; \Delta', y{:}A, x'{:}B \vdash Q :: z{:}C}{\Gamma; \Delta', x{:}A \otimes B \vdash x(y, x').Q :: z{:}C}\ \otimes L$$

$$\frac{}{\cdot \vdash \overline{x}\langle\rangle.\mathbf{0} :: x{:}\mathbf{1}}\ \mathbf{1}R \qquad \frac{\Gamma; \Delta' \vdash Q :: z{:}C}{\Gamma; \Delta', x{:}\mathbf{1} \vdash x().Q :: z{:}C}\ \mathbf{1}L$$

$$\frac{\Gamma; \Delta \vdash P_1 :: x_1'{:}A \quad \Gamma; \Delta \vdash P_2 :: x_2'{:}B}{\Gamma; \Delta \vdash x.\mathsf{case}((x_1').P_1, (x_2').P_2) :: x{:}A \,\&\, B}\ \&R$$

$$\frac{\Gamma; \Delta', x_1'{:}A \vdash Q :: z{:}C}{\Gamma; \Delta', x{:}A \,\&\, B \vdash x.\mathsf{inl}\langle x_1'\rangle; Q :: z{:}C}\ \&L_1 \qquad \frac{\Gamma; \Delta', x_2'{:}B \vdash Q :: z{:}C}{\Gamma; \Delta', x{:}A \,\&\, B \vdash x.\mathsf{inr}\langle x_2'\rangle; Q :: z{:}C}\ \&L_2$$

$$\frac{\Gamma; \Delta \vdash P :: x_1'{:}A}{\Gamma; \Delta \vdash x.\mathsf{inl}\langle x_1'\rangle; P :: x{:}A \oplus B}\ \oplus R_1 \qquad \frac{\Gamma; \Delta \vdash P :: x_2'{:}B}{\Gamma; \Delta \vdash x.\mathsf{inr}\langle x_2'\rangle; P :: x{:}A \oplus B}\ \oplus R_2$$

$$\frac{\Gamma; \Delta', x_1'{:}A \vdash Q_1 :: z{:}C \quad \Gamma; \Delta', x_2'{:}B \vdash Q_2 :: z{:}C}{\Gamma; \Delta', x{:}A \oplus B \vdash x.\mathsf{case}((x_1').Q_1, (x_2').Q_2) :: z{:}C}\ \oplus L$$

$$\frac{\Gamma; \cdot \vdash P :: y{:}A}{\Gamma; \cdot \vdash (\nu u)\overline{x}\langle u\rangle.!u(y).P :: x{:}!A}\ !R \qquad \frac{\Gamma, u{:}A; \Delta' \vdash Q :: z{:}C}{\Gamma; \Delta', x{:}!A \vdash x(u).Q :: z{:}C}\ !L$$

Figure 1 A polyadic synchronous process assignment that is equivalent to one from [9].

Under the polyadic synchronous process assignment, the same cut reduction corresponds to

$$\begin{aligned}&(\nu x)\big((\nu y)(\nu x')\overline{x}\langle y, x'\rangle.(P_1 \mid P_2\{x'/x\}) \mid x(y, x').Q\{x'/x\}\big)\\ &\quad\longrightarrow (\nu x')\big(P_2\{x'/x\} \mid (\nu y)(P_1 \mid Q\{x'/x\})\big)\,.\end{aligned}$$

Typing guarantees that x' is not free in P_1. Thus, by α-renaming x' to x, we obtain the same process after reduction as in the monadic synchronous assignment. In this way, there is a tight operational correspondence between the monadic and polyadic synchronous assignments.

3.2 Commuting conversions as process equivalences

We can now turn to relating the interpretation of commuting conversions under the asynchronous and synchronous process assignments. Having shown that the synchronous polyadic process assignment of Figure 1 is equivalent to the synchronous monadic assignment of [9], we can compare our asynchronous assignment from Section 2 with the synchronous polyadic assignment and know that the comparison extends to the synchronous monadic assignment.

In proof theory, commuting conversions describe structural equivalences among proofs. The following is an example of one commuting conversion between two adjacent cut inferences.

$$\frac{\dfrac{\Gamma; \Delta_1 \vdash B \quad \Gamma; \Delta_2, B \vdash A}{\Gamma; \Delta_1, \Delta_2 \vdash A}\ \mathsf{cut} \qquad \Gamma; \Delta', A \vdash C}{\Gamma; \Delta_1, \Delta_2, \Delta' \vdash C}\ \mathsf{cut} \quad\equiv\quad \frac{\Gamma; \Delta_1 \vdash B \qquad \dfrac{\Gamma; \Delta_2, B \vdash A \quad \Gamma; \Delta', A \vdash C}{\Gamma; \Delta_2, \Delta', B \vdash C}\ \mathsf{cut}}{\Gamma; \Delta_1, \Delta_2, \Delta' \vdash C}\ \mathsf{cut}$$

To what do such commuting conversions correspond under the synchronous and asynchronous process assignments? The commuting conversions can be sorted into three classes.

Class 1. Some commuting conversions correspond to structural equivalences under *both* the synchronous and asynchronous assignments. For example, both assignments interpret cut with the same process. Thus, if we annotate the above $\mathsf{cut}/\mathsf{cut}$ conversion accordingly, it can be read as the following structural equivalence on processes:

$$(\nu x)\big((\nu y)(P_1 \mid P_2) \mid Q\big) \equiv (\nu y)\big(P_1 \mid (\nu x)(P_2 \mid Q)\big)\,, \text{ if } x \notin \mathsf{fn}(P_1) \text{ and } y \notin \mathsf{fn}(Q).$$

This equivalence is derivable from more basic laws of the (synchronous and asynchronous) π-calculus structural congruence, such as associativity and commutativity of parallel composition and scope extrusion of name restrictions. (The side condition on the free names, denoted $\mathsf{fn}(-)$, is only necessary in the untyped π-calculus; here it is guaranteed by typing.)

Class 2. Most commuting conversions do *not* yield structural process equivalences under the synchronous interpretation. For example, one conversion between $\otimes R$ and $\multimap L$ is:

$$\dfrac{\Gamma;\Delta \vdash A_1 \quad \dfrac{\Gamma;\Delta_1' \vdash B_1 \quad \Gamma;\Delta_2', B_2 \vdash A_2}{\Gamma;\Delta_1', \Delta_2', B_1 \multimap B_2 \vdash A_2}\multimap L}{\Gamma;\Delta, \Delta_1', \Delta_2', B_1 \multimap B_2 \vdash A_1 \otimes A_2}\otimes R \quad\equiv\quad \dfrac{\Gamma;\Delta_1' \vdash B_1 \quad \dfrac{\Gamma;\Delta \vdash A_1 \quad \Gamma;\Delta_2', B_2 \vdash A_2}{\Gamma;\Delta, \Delta_2', B_2 \vdash A_1 \otimes A_2}\otimes R}{\Gamma;\Delta, \Delta_1', \Delta_2', B_1 \multimap B_2 \vdash A_1 \otimes A_2}\multimap L$$

Under the synchronous interpretation, this commuting conversion does not correspond to a structural process equivalence of the synchronous polyadic π-calculus because it reorders two blocking outputs:

$$\begin{aligned}&(\nu w)(\nu z')\overline{z}\langle w, z'\rangle.\big(P \mid (\nu y)(\nu x')\overline{x}\langle y, x'\rangle.(Q_1 \mid Q_2)\big)\\ &\quad\not\equiv (\nu y)(\nu x')\overline{x}\langle y, x'\rangle.\big(Q_1 \mid (\nu w)(\nu z')\overline{z}\langle w, z'\rangle.(P \mid Q_2)\big)\,.\end{aligned}$$

However, when composed with closed processes as required by the Γ and $\Delta, \Delta_1', \Delta_2', x{:}B_1 \multimap B_2$ contexts, these two processes are *observationally* equivalent, according to (a simple dyadic extension of) typed context bisimilarity as defined by Pérez et al. [20]. Essentially, the reason is this: When composed with the required processes, only the actions along $z{:}A_1 \otimes A_2$, namely $(\nu w)(\nu z')\overline{z}\langle w, z'\rangle$, are observable—all other interactions, such as $(\nu y)(\nu x')\overline{x}\langle y, x'\rangle$, are internal to the closed process—and so the reordering cannot be detected.

On the other hand, under our asynchronous process interpretation, this $\otimes R/\multimap L$ commuting conversion can be read as the structural process equivalence

$$\begin{aligned}&(\nu w)(\nu z')\big(\overline{z}\langle w, z'\rangle \mid P \mid (\nu y)(\nu x')(\overline{x}\langle y, x'\rangle \mid Q_1 \mid Q_2)\big)\\ &\equiv (\nu y)(\nu x')\big(\overline{x}\langle y, x'\rangle \mid Q_1 \mid (\nu w)(\nu z')(\overline{z}\langle w, z'\rangle \mid P \mid Q_2)\big),\ \text{if } w, z' \notin \mathsf{fn}(Q_1) \text{ and } y, x' \notin \mathsf{fn}(P).\end{aligned}$$

Once again, this equivalence is derivable from laws of structural congruence that are standard in the asynchronous π-calculus.

We contend that our asynchronous interpretation is therefore proof-theoretically more pleasing: it maps certain structural equivalences of proofs to standard *structural* equivalences of processes, whereas the synchronous interpretation only mapped these proof equivalences to strictly weaker observational equivalences on processes.

Class 3. Another class of commuting conversions are those that involve rules to which input-flavored processes are assigned. These conversions do not correspond to structural

process equivalences under *either* the synchronous or asynchronous assignments. For example, one such conversion is between $\otimes R$ and $\otimes L$:

$$\dfrac{\Gamma;\Delta_1 \vdash A_1 \quad \dfrac{\Gamma;\Delta_2, B_1, B_2 \vdash A_2}{\Gamma;\Delta_2, B_1 \otimes B_2 \vdash A_2}\otimes L}{\Gamma;\Delta_1,\Delta_2, B_1 \otimes B_2 \vdash A_1 \otimes A_2}\otimes R \quad\equiv\quad \dfrac{\dfrac{\Gamma;\Delta_1 \vdash A_1 \quad \Gamma;\Delta_2, B_1, B_2 \vdash A_2}{\Gamma;\Delta_1,\Delta_2, B_1, B_2 \vdash A_1 \otimes A_2}\otimes R}{\Gamma;\Delta_1,\Delta_2, B_1 \otimes B_2 \vdash A_1 \otimes A_2}\otimes L$$

Under the synchronous assignment, this conversion is only an observational equivalence:

$$(\nu w)(\nu z')\overline{z}\langle w, z'\rangle.\big(P \mid x(y,x').Q\big) \approx x(y,x').\big((\nu w)(\nu z')\overline{z}\langle w, z'\rangle.(P \mid Q)\big)\,.$$

The justification is similar to the previous one: When composed with closed processes required by Γ and $\Delta_1, \Delta_2, B_1 \otimes B_2$, only the actions along $z{:}A_1 \otimes A_2$, namely $(\nu w)(\nu z')\overline{z}\langle w, z'\rangle$, are observable because all other interactions, such as $x(y,x')$, are internal to the closed process.

Similarly, under the asynchronous process assignment, this commuting conversion does *not* yield a structural equivalence because permuting the input outward blocks actions in P:

$$(\nu w)(\nu z')\big(\overline{z}\langle w, z'\rangle \mid P \mid x(y,x').Q\big) \not\equiv x(y,x').(\nu w)(\nu z')(\overline{z}\langle w, z'\rangle \mid P \mid Q)\,.$$

This exposes a fundamental asymmetry between outputs, which can be interpreted asynchronously and give rise to very natural structural commutation laws, and inputs, which, in a process calculus, are inherently points of synchronization and cannot obey *structural* commutation laws since that would defeat the purpose of synchronization points.

With the above intuition for the three classes of conversions, we obtain the following.

▶ **Theorem 1.** *For each commuting conversion listed in Figure 2, its asynchronous process interpretation is a standard structural equivalence.*

4 Correspondence with an asynchronous buffered session semantics

In Section 2, we presented a novel Curry-Howard interpretation of intuitionistic linear logic as an asynchronous session-type system. Asynchronous outputs were represented abstractly as free-floating messages waiting to be received by an input process. However, in keeping with practical implementations of asynchronous communication, existing asynchronous session-type systems use explicitly buffered communication channels [12, 11, 18]. To relate our Curry-Howard interpretation to existing asynchronous session-type systems, we now show that the use of fresh channels for session continuations serves as an encoding of FIFO buffers. First, we must present a π-calculus with explicit two-sided FIFO buffers.

4.1 A π-calculus with explicit two-sided FIFO buffers

Syntax and structural congruence. Syntactically, this calculus extends the (synchronous) π-calculus with a FIFO buffer (i.e., queue) construct, $x[m_k, \ldots, m_1\rangle z$. It represents an input buffer at endpoint z that holds the sequence $m_1, \ldots, m_k$ of messages that have been sent by the peer endpoint x. A message m has one of several forms: a linear channel, y; a termination signal, fin; left and right selectors, inl and inr; or registration of a persistent channel, $!u$. We assume that a message sent by endpoint x immediately arrives at the tail of its peer endpoint's input buffer. It will also be useful to adopt $z\langle m_1, \ldots, m_k]x$ as alternate notation for the queue $x[m_k, \ldots, m_1\rangle z$.

In addition to the usual basic laws of π-calculus structural congruence, we include the equivalence $x[\rangle z \equiv x\langle]z$. This expresses that an empty queue remains uncommitted to its direction—either endpoint may place a message onto the empty queue.

Class 2:

$(\mathsf{cut}/{\multimap}L/-), (-\!\circ L/-/\mathsf{cut}_1)$	$(\nu x)((\nu w, y')(\overline{y}\langle w, y'\rangle \mid P_1 \mid P_2) \mid Q)$ $\equiv (\nu w, y')(\overline{y}\langle w, y'\rangle \mid P_1 \mid (\nu x)(P_2 \mid Q))$
$(\mathsf{cut}/\mathbf{1}L/-), (\mathbf{1}L/\mathsf{cut}_1)$	$(\nu x)((y().\mathbf{0} \mid P) \mid Q) \equiv y().\mathbf{0} \mid (\nu x)(P \mid Q)$
$(\mathsf{cut}/\&L_i/-), (\&L_i/\mathsf{cut}_1)$	$(\nu x)((\nu y')(y.\mathsf{in}[\mathsf{l}/\mathsf{r}]\langle y'\rangle \mid P) \mid Q) \equiv (\nu y')(y.\mathsf{in}[\mathsf{l}/\mathsf{r}]\langle y'\rangle \mid (\nu x)(P \mid Q))$
$(\mathsf{cut}/\mathsf{copy}/-), (\mathsf{copy}/\mathsf{cut}_1)$	$(\nu x)((\nu y)(\overline{u}\langle y\rangle \mid P) \mid Q) \equiv (\nu y)(\overline{u}\langle y\rangle \mid (\nu x)(P \mid Q))$
$(\mathsf{cut}/-/{\multimap}L_1), (\mathsf{cut}/-/{\otimes}R_1),$ $(-\!\circ L/\mathsf{cut}/-), (\otimes R/\mathsf{cut}/-)$	$(\nu x)(P \mid (\nu w, y')(\overline{y}\langle w, y'\rangle \mid Q_1 \mid Q_2))$ $\equiv (\nu w, y')(\overline{y}\langle w, y'\rangle \mid (\nu x)(P \mid Q_1) \mid Q_2)$
$(\mathsf{cut}/-/{\multimap}L_2), (\mathsf{cut}/-/{\otimes}R_2),$ $(-\!\circ L/-/\mathsf{cut}_2), (\otimes R/-/\mathsf{cut})$	$(\nu x)(P \mid (\nu w, y')(\overline{y}\langle w, y'\rangle \mid Q_1 \mid Q_2))$ $\equiv (\nu w, y')(\overline{y}\langle w, y'\rangle \mid Q_1 \mid (\nu x)(P \mid Q_2))$
$(\mathsf{cut}/-/\mathbf{1}L), (\mathbf{1}L/\mathsf{cut}_2)$	$(\nu x)(P \mid (y().\mathbf{0} \mid Q)) \equiv y().\mathbf{0} \mid (\nu x)(P \mid Q)$
$(\mathsf{cut}/-/\&L_i), (\mathsf{cut}/-/\oplus R_i),$ $(\&L_i/\mathsf{cut}_2), (\oplus R_i/\mathsf{cut}_2)$	$(\nu x)(P \mid (\nu y')(y.\mathsf{in}[\mathsf{l}/\mathsf{r}]\langle y'\rangle \mid Q)) \equiv (\nu y')(y.\mathsf{in}[\mathsf{l}/\mathsf{r}]\langle y'\rangle \mid (\nu x)(P \mid Q))$
$(\mathsf{cut}/-/\mathsf{copy}), (\mathsf{copy}/\mathsf{cut}_2)$	$(\nu x)(P \mid (\nu y)(\overline{u}\langle y\rangle \mid Q)) \equiv (\nu y)(\overline{u}\langle y\rangle \mid (\nu x)(P \mid Q))$
$(-\!\circ L/\mathsf{cut}!/-), (\otimes R/\mathsf{cut}!/-)$	$(\nu w, z')(\overline{z}\langle w, z'\rangle \mid (\nu u)(!u(y).P_1 \mid P_2) \mid Q)$ $\equiv (\nu u)(!u(y).P_1 \mid (\nu w, z')(\overline{z}\langle w, z'\rangle \mid P_2 \mid Q))$
$(-\!\circ L/-/\mathsf{cut}!), (\otimes R/-/\mathsf{cut}!)$	$(\nu w, z')(\overline{z}\langle w, z'\rangle \mid P \mid (\nu u)(!u(y).Q_1 \mid Q_2))$ $\equiv (\nu u)(!u(y).Q_1 \mid (\nu w, z')(\overline{z}\langle w, z'\rangle \mid P \mid Q_2))$
$(\mathsf{cut}!/-/\mathbf{1}L), (\mathbf{1}L/\mathsf{cut}!)$	$(\nu u)(!u(y).P \mid (x().\mathbf{0} \mid Q)) \equiv x().\mathbf{0} \mid (\nu u)(!u(y).P \mid Q)$
$(\mathsf{cut}!/-/\&L_i), (\mathsf{cut}!/-/\oplus R_i),$ $(\&L_i/\mathsf{cut}!), (\oplus R_i/\mathsf{cut}!)$	$(\nu u)(!u(y).P \mid (\nu x')(x.\mathsf{in}[\mathsf{l}/\mathsf{r}]\langle x'\rangle \mid Q))$ $\equiv (\nu x')(x.\mathsf{in}[\mathsf{l}/\mathsf{r}]\langle x'\rangle \mid (\nu u)(!u(y).P \mid Q))$
$(\mathsf{cut}!/-/\mathsf{copy}), (\mathsf{copy}/\mathsf{cut}!)$	$(\nu u)(!u(y).P \mid (\nu x)(\overline{v}\langle x\rangle \mid Q)) \equiv (\nu x)(\overline{v}\langle x\rangle \mid (\nu u)(!u(y).P \mid Q))$
$(-\!\circ L/-\!\circ L/-), (-\!\circ L/-/-\!\circ L_1),$ $(-\!\circ L/-/{\otimes}R_1), (\otimes R/-\!\circ L/-)$	$(\nu y, x')(\overline{x}\langle y, x'\rangle \mid (\nu w, z')(\overline{z}\langle w, z'\rangle \mid P_1 \mid P_2) \mid Q)$ $\equiv (\nu w, z')(\overline{z}\langle w, z'\rangle \mid P_1 \mid (\nu y, x')(\overline{x}\langle y, x'\rangle \mid P_2 \mid Q))$
$(-\!\circ L/\mathbf{1}L/-), (\otimes R/\mathbf{1}L/-),$ $(\mathbf{1}L/-\!\circ L_1), (\mathbf{1}L/\otimes R_1)$	$(\nu y, x')(\overline{x}\langle y, x'\rangle \mid (z().\mathbf{0} \mid P) \mid Q) \equiv z().\mathbf{0} \mid (\nu y, x')(\overline{x}\langle y, x'\rangle \mid P \mid Q)$
$(-\!\circ L/\&L_i/-), (\otimes R/\&L_i/-),$ $(\&L_i/-\!\circ L_1), (\&L_i/\otimes R_1)$	$(\nu y, x')(\overline{x}\langle y, x'\rangle \mid (\nu z')(z.\mathsf{in}[\mathsf{l}/\mathsf{r}]\langle z'\rangle \mid P) \mid Q)$ $\equiv (\nu z')(z.\mathsf{in}[\mathsf{l}/\mathsf{r}]\langle z'\rangle \mid (\nu y, x')(\overline{x}\langle y, x'\rangle \mid P \mid Q))$
$(-\!\circ L/\mathsf{copy}/-), (\otimes R/\mathsf{copy}/-),$ $(\mathsf{copy}/-\!\circ L_1), (\mathsf{copy}/\otimes R_1)$	$(\nu y, x')(\overline{x}\langle y, x'\rangle \mid (\nu z)(\overline{u}\langle z\rangle \mid P) \mid Q)$ $\equiv (\nu z)(\overline{u}\langle z\rangle \mid (\nu y, x')(\overline{x}\langle y, x'\rangle \mid P \mid Q))$
$(-\!\circ L/-/-\!\circ L_2), (-\!\circ L/-/{\otimes}R_2),$ $(\otimes R/-/-\!\circ L)$	$(\nu w, z')(\overline{z}\langle w, z'\rangle \mid P \mid (\nu y, x')(\overline{x}\langle y, x'\rangle \mid Q_1 \mid Q_2))$ $\equiv (\nu y, x')(\overline{x}\langle y, x'\rangle \mid Q_1 \mid (\nu w, z')(\overline{z}\langle w, z'\rangle \mid P \mid Q_2))$
$(-\!\circ L/-/\mathbf{1}L), (\otimes R/-/\mathbf{1}L),$ $(\mathbf{1}L/-\!\circ L_2), (\mathbf{1}L/\otimes R_2)$	$(\nu y, x')(\overline{x}\langle y, x'\rangle \mid P \mid (z().\mathbf{0} \mid Q)) \equiv z().\mathbf{0} \mid (\nu y, x')(\overline{x}\langle y, x'\rangle \mid P \mid Q)$
$(-\!\circ L/-/\&L_i), (-\!\circ L/-/\oplus R_i),$ $(\otimes R/-/\&L_i), (\&L_i/-\!\circ L_2),$ $(\&L_i/\otimes R_2), (\oplus R_i/-\!\circ L)$	$(\nu y, x')(\overline{x}\langle y, x'\rangle \mid P \mid (\nu z')(z.\mathsf{in}[\mathsf{l}/\mathsf{r}]\langle z'\rangle \mid Q))$ $\equiv (\nu z')(z.\mathsf{in}[\mathsf{l}/\mathsf{r}]\langle z'\rangle \mid (\nu y, x')(\overline{x}\langle y, x'\rangle \mid P \mid Q))$
$(-\!\circ L/-/\mathsf{copy}), (\otimes R/-/\mathsf{copy}),$ $(\mathsf{copy}/-\!\circ L_2), (\mathsf{copy}/\otimes R_2)$	$(\nu y, x')(\overline{x}\langle y, x'\rangle \mid P \mid (\nu z)(\overline{u}\langle z\rangle \mid Q))$ $\equiv (\nu z)(\overline{u}\langle z\rangle \mid (\nu y, x')(\overline{x}\langle y, x'\rangle \mid P \mid Q))$
$(\mathbf{1}L/\mathbf{1}L)$	$x().\mathbf{0} \mid (z().\mathbf{0} \mid P) \equiv z().\mathbf{0} \mid (x().\mathbf{0} \mid P)$
$(\mathbf{1}L/\&L_i), (\mathbf{1}L/\oplus R_i),$ $(\&L_i/\mathbf{1}L), (\oplus R_i/\mathbf{1}L)$	$x().\mathbf{0} \mid (\nu z')(z.\mathsf{in}[\mathsf{l}/\mathsf{r}]\langle z'\rangle \mid P) \equiv (\nu z')(z.\mathsf{in}[\mathsf{l}/\mathsf{r}]\langle z'\rangle \mid (x().\mathbf{0} \mid P))$
$(\mathbf{1}L/\mathsf{copy}), (\mathsf{copy}/\mathbf{1}L)$	$x().\mathbf{0} \mid (\nu z)(\overline{u}\langle z\rangle \mid P) \equiv (\nu z)(\overline{u}\langle z\rangle \mid (x().\mathbf{0} \mid P))$
$(\&L_i/\&L_j), (\&L_i/\oplus R_j),$ $(\oplus R_i/\&L_j)$	$(\nu x')(x.\mathsf{in}[\mathsf{l}/\mathsf{r}]\langle x'\rangle \mid (\nu z')(z.\mathsf{in}[\mathsf{l}/\mathsf{r}]\langle z'\rangle \mid P))$ $\equiv (\nu z')(z.\mathsf{in}[\mathsf{l}/\mathsf{r}]\langle z'\rangle \mid (\nu x')(x.\mathsf{in}[\mathsf{l}/\mathsf{r}]\langle x'\rangle \mid P))$
$(\mathsf{copy}/\mathsf{copy})$	$(\nu x)(\overline{u}\langle x\rangle \mid (\nu z)(\overline{v}\langle z\rangle \mid P)) \equiv (\nu z)(\overline{v}\langle z\rangle \mid (\nu x)(\overline{u}\langle x\rangle \mid P))$

Figure 2 Asynchronous structural equivalences that arise from commuting conversions.

Syntax and structural congruence:

$$m, n ::= y \mid \mathsf{fin} \mid \mathsf{inl} \mid \mathsf{inr} \mid !u$$
$$P, Q ::= (\nu x)P \mid (P \mid Q) \mid \mathbf{0} \mid \overline{x}\langle y\rangle.P \mid x(y).P \mid \overline{x}\langle\rangle.\mathbf{0} \mid x().P \mid x.\mathsf{inl}; P \mid x.\mathsf{inr}; P \mid x.\mathsf{case}(P, Q)$$
$$\mid \overline{x}\langle u\rangle.P \mid x(u).P \mid \overline{u}\langle x\rangle \mid !u(y).P \mid x[\vec{m}\rangle z$$

All π-calculus laws of structural congruence, plus $x[\rangle z \equiv x\langle]z$.

Untyped reductions:

$$\overline{x}\langle y\rangle.P \mid x[\vec{m}\rangle z \longrightarrow P \mid x[y, \vec{m}\rangle z \quad \text{(S-CH)}$$
$$\overline{x}\langle\rangle.\mathbf{0} \mid x[\vec{m}\rangle z \longrightarrow x[\mathsf{fin}, \vec{m}\rangle z \quad \text{(S-FIN)}$$
$$x.\mathsf{inl}; P \mid x[\vec{m}\rangle z \longrightarrow P \mid x[\mathsf{inl}, \vec{m}\rangle z \quad \text{(S-INL)}$$
$$x.\mathsf{inr}; P \mid x[\vec{m}\rangle z \longrightarrow P \mid x[\mathsf{inr}, \vec{m}\rangle z \quad \text{(S-INR)}$$
$$\overline{x}\langle u\rangle.P \mid x[\vec{m}\rangle z \longrightarrow P \mid x[!u, \vec{m}\rangle z \quad \text{(S-!CH)}$$
$$x[\vec{m}, y\rangle z \mid z(w).Q \longrightarrow x[\vec{m}\rangle z \mid Q\{y/w\} \quad \text{(R-CH)}$$
$$x[\mathsf{fin}\rangle z \mid z().Q \longrightarrow Q \quad \text{(R-FIN)}$$
$$x[\vec{m}, \mathsf{inl}\rangle z \mid z.\mathsf{case}(Q_1, Q_2) \longrightarrow x[\vec{m}\rangle z \mid Q_1 \quad \text{(R-INL)}$$
$$x[\vec{m}, \mathsf{inr}\rangle z \mid z.\mathsf{case}(Q_1, Q_2) \longrightarrow x[\vec{m}\rangle z \mid Q_2 \quad \text{(R-INR)}$$
$$x[!u\rangle z \mid z(v).Q \longrightarrow Q\{u/v\} \quad \text{(R-!CH)}$$
$$!u(y).P \mid \overline{u}\langle x\rangle.Q \longrightarrow !u(y).P \mid P\{x/y\} \mid Q \quad \text{(REP)}$$

Notational definitions:

$$(\nu z)(P \mid z[\rangle x) \triangleq P\{x/z\}$$
$$(\nu z)\big(P_2 \mid (\nu y)(P_1 \mid z[\vec{m}, y\rangle x)\big)$$
$$\triangleq (\nu y)(\nu x')\big(\overline{x}\langle y, x'\rangle \mid P_1 \mid (\nu z)(P_2 \mid z[\vec{m}\rangle x')\big)$$
$$(\nu z)(z[\mathsf{fin}\rangle x) \triangleq \overline{x}\langle\rangle$$
$$(\nu z)(\nu u)(P \mid z[!u\rangle x) \triangleq (\nu u)(\overline{x}\langle u\rangle \mid P)$$
$$(\nu z)(P \mid z[\vec{m}, \mathsf{inl}\rangle x) \triangleq (\nu x')\big(x.\mathsf{inl}\langle x'\rangle \mid (\nu z)(P \mid z[\vec{m}\rangle x')\big)$$
$$(\nu z)(P \mid z[\vec{m}, \mathsf{inr}\rangle x) \triangleq (\nu x')\big(x.\mathsf{inr}\langle x'\rangle \mid (\nu z)(P \mid z[\vec{m}\rangle x')\big)$$

$$\overline{x}\langle y\rangle.P \triangleq (\nu x')(\overline{x}\langle y, x'\rangle \mid P\{x'/x\})$$
$$x(y).P \triangleq x(y, x').P\{x'/x\}$$
$$\overline{x}\langle\rangle.\mathbf{0} \triangleq \overline{x}\langle\rangle$$
$$\overline{x}\langle u\rangle.P \triangleq \overline{x}\langle u\rangle \mid P$$
$$x.\mathsf{inl}; P \triangleq (\nu x')(x.\mathsf{inl}\langle x'\rangle \mid P\{x'/x\})$$
$$x.\mathsf{inr}; P \triangleq (\nu x')(x.\mathsf{inr}\langle x'\rangle \mid P\{x'/x\})$$
$$x.\mathsf{case}(P_1, P_2) \triangleq x.\mathsf{case}((x_1').P_1\{x_1'/x\}, (x_2').P_2\{x_2'/x\})$$

Figure 3 A π-calculus with explicit two-sided FIFO buffers.

Reduction semantics. The reductions are given in Figure 3. Reductions S-CH, S-FIN, S-INL, S-INR, and S-!CH show that an output from endpoint x can always be placed at the tail of its peer endpoint's input buffer. Thus, outputs are non-blocking. Conversely, reductions R-CH, R-FIN, R-INL, R-INR, and R-!CH show how the peer endpoint z responds to these messages using inputs and cases. Note that receipt of a fin termination message (R-FIN) causes the buffer to be deallocated. Similarly, receipt of a persistent channel (R-!CH) deallocates the buffer because persistent channels spawn linear sessions rather than establishing a persistent pattern of communication in their own right.

4.2 Typing and well-typed reductions for buffered processes

In our asynchronous process assignment, the use of fresh channels for session continuations orders outputs within a session into a queue: the sequence $\overline{x}\langle y_1, x'\rangle$, $\overline{x'}\langle y_2, x''\rangle$, $\overline{x''}\langle y_3, x'''\rangle$, ... can be read as a queue, $x\langle y_1, y_2, y_3, \ldots]$. This intuition allows us to treat buffered processes as notational definitions for polyadic asynchronous processes, as shown in Figure 3.

To type processes in the calculus with explicit buffers, we expand the definitions and type the resulting process according the polyadic asynchronous process assignment. For instance, to type the process $(\nu z)\big(\overline{z}\langle\rangle.\mathbf{0} \mid (\nu y)(P_1 \mid z[y\rangle x)\big)$, we would expand the definitions, typing $(\nu y)(\nu x')\big(\overline{x}\langle y, x'\rangle \mid P_1 \mid \overline{x'}\langle\rangle\big)$ instead. The reductions with explicit buffers also correspond to reductions in the polyadic asynchronous process assignment:

▶ **Theorem 2.** *Well-typed reductions respect the definitions from Figure 3.*

Proof. As a representative example, consider the well-typed reductions derived from (S-CH) and (R-CH). The well-typed reduction corresponding to (R-CH) is

$$(\nu x)\big((\nu z)(P_2 \mid (\nu y)(P_1 \mid z[\vec{m}, y\rangle x)) \mid x(y).Q\big) \longrightarrow (\nu x)\big((\nu z)(P_2 \mid z[\vec{m}\rangle x) \mid (\nu y)(P_1 \mid Q)\big)\,.$$

By expanding according to the notational definitions from Figure 3, the reducing process is $(\nu x)\big((\nu y, x')(\overline{x}\langle y, x'\rangle \mid P_1 \mid (\nu z)(P_2 \mid z[\vec{m}\rangle x')) \mid x(y, x').Q\{x'/x\}\big)$. By the principal cut at $\otimes$ type, this reduces to $(\nu x')\big((\nu z)(P_2 \mid z[\vec{m}\rangle x') \mid (\nu y)(P_1 \mid Q\{x'/x\})\big)$, which, modulo α-conversion (since x' is not free in P_1 or P_2), is the same as the direct result.

The well-typed reduction corresponding to (S-CH) is

$$(\nu x)\big((\nu z)((\nu y)\overline{z}\langle y\rangle.(P_1 \mid P_2) \mid z[\vec{m}\rangle x) \mid Q\big) \longrightarrow (\nu x)\big((\nu z)(P_2 \mid (\nu y)(P_1 \mid z[y, \vec{m}\rangle x)) \mid Q\big)\,.$$

We can show by induction on the length of $\vec{m}$ that these processes are structurally equivalent when the definitions from Figure 3 are applied. The inductive case is straightforward. In the base case, it suffices to show that $(\nu z)((\nu y)\overline{z}\langle y\rangle.(P_1 \mid P_2) \mid z[\rangle x)$ and $(\nu z)(P_2 \mid (\nu y)(P_1 \mid z[y\rangle x))$ are structurally equivalent when the definitions are expanded. The former expands to $(\nu y, x')(\overline{x}\langle y, x'\rangle \mid P_1 \mid P_2\{x'/z\})$ and so does the latter. ◀

This result shows that our asynchronous polyadic process assignment does indeed faithfully represent buffered asynchronous session types.

5 Related Work

The connections between linear logic and concurrency have been studied by both the logic and concurrency theory communities. Abramsky gave a process algebraic interpretation of classical linear logic proofs [1]. Since then, some work has taken a "propositions as types" approach. Two of the present authors proposed an interpretation of linear logic in which *synchronous* π-calculus session-typed processes are intuitionistic linear logic proofs [8], giving rise to several interesting extensions and applications [9].

The connections between *asynchronous* process algebras and linear logic are also not new. Honda and Laurent [16] show a correspondence between polarized proof nets and typings for the asynchronous polyadic π-calculus. In contrast to our work, they consider the much simpler IO-type system, rather than session types. Moreover, they use classical proof nets, whereas we capture asynchrony while remaining in a sequent calculus.

Bellin and Scott [6] also give a process interpretation of classical linear logic using proof nets. They modify the synchronous π-calculus by adding structural laws that allow for arbitrary prefix commutations. This greatly simplifies the match with the proof theory but also makes the development somewhat artificial from the process calculus perspective. In contrast, we compromise between the two worlds in an arguably more satisfying way.

Beauxis et al. [4] showed that, in an untyped setting, the asynchronous π-calculus corresponds to using bags for buffered communication. Buffers for session-typed asynchrony have been considered for binary session types in a functional language [12], an object-oriented language [11], and for multiparty sessions [18]. The systems of [12, 11] are similar to ours but lack a clean logical interpretation which we get from our operational correspondence results.

6 Conclusion

In this paper, we have exhibited a novel process assignment from the asynchronous, polyadic π-calculus to the proof rules of intuitionistic linear logic (Section 2). By allowing non-blocking

outputs, our asynchronous interpretation exposes additional parallelism inherent in linear logic that remained hidden in the prior synchronous interpretation [8, 9]. Proof-theoretically, this arises from a better match between proof equivalences and process equivalences (Section 3).

As future work, we would like to further study the behavioral theory of the asynchronous process assignment. With this understanding, it should then be possible to relate the synchronous and the asynchronous assignments by developing a form of delayed bisimulation for synchronous processes. We would also like to extend the asynchronous assignment to *multiparty* session types [18]; we conjecture that hybrid logic [19] might prove useful.

References

1 S. Abramsky. Computational interpretations of linear logic. *Theoret. Comput. Sci.*, 111(1–2):3–57, 1993.

2 J-M. Andreoli. Logic programming with focusing proofs in linear logic. *J. Logic Comput.*, 2(3):197–347, 1992.

3 A. Barber. Dual intuitionistic linear logic. Technical Report LFCS-96-347, Univ. of Edinburgh, 1996.

4 R. Beauxis, C. Palamidessi, and F. D. Valencia. On the asynchronous nature of the asynchronous π-calculus. In *Concurrency, Graphs and Models*, pages 473–492, 2008.

5 E. Beffara. A concurrent model for linear logic. In *21st Ann. Conf. Math. Found. Program. Semantics*, pages 147–168, 2006.

6 G. Bellin and P. Scott. On the π-calculus and linear logic. *Theoret. Comput. Sci.*, 135(1):11–65, 1994.

7 G. Boudol. Asynchrony and the π-calculus. Rapport de recherche RR-1702, INRIA, 1992.

8 L. Caires and F. Pfenning. Session types as intuitionistic linear propositions. In *21st Int. Conf. Concur. Theory*, pages 222–236. LNCS 6269, 2010.

9 L. Caires, F. Pfenning, and B. Toninho. Towards concurrent type theory. In *8th ACM SIGPLAN Workshop on Types in Language Design and Implementation*, pages 1–12, 2012.

10 B-Y. E. Chang, K. Chaudhuri, and F. Pfenning. A judgmental analysis of linear logic. Technical Report CMU-CS-03-131R, Carnegie Mellon Univ., 2003.

11 M. Dezani-Ciancaglini, S. Drossopoulou, D. Mostrous, and N. Yoshida. Objects and session types. *Inform. and Comput.*, 207(5):595–641, 2009.

12 S. Gay and V. T. Vasconcelos. Linear type theory for asynchronous session types. *J. Funct. Programming*, 20(1):19–50, 2010.

13 J.-Y. Girard. Linear logic. *Theoret. Comput. Sci.*, 50(1):1–102, 1987.

14 M. Giunti and V. T. Vasconcelos. A linear account of session types in the π-calculus. In *21st Int. Conf. Concur. Theory*, pages 432–446. LNCS 6269, 2010.

15 K. Honda. Types for dyadic interaction. In *4th Int. Conf. Concur. Theory*, pages 509–523. LNCS 715, 1993.

16 K. Honda and O. Laurent. An exact correspondence between a typed π-calculus and polarised proof-nets. *Theoret. Comput. Sci.*, 411(22–24):2223–2238, 2010.

17 K. Honda and M. Tokoro. An object calculus for asynchronous communication. In *5th Eur. Conf. Object-Oriented Programming*, pages 133–147. LNCS 512, 1991.

18 K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *35th ACM SIGPLAN-SIGACT Symp. Prin. Program. Lang.*, pages 273–284, 2008.

19 T. Murphy, VII. *Modal Types for Mobile Code*. PhD thesis, Carnegie Mellon Univ., January 2008. Available as technical report CMU-CS-08-126.

20 J. A. Pérez, L. Caires, F. Pfenning, and B. Toninho. Linear logical relations for session-based concurrency. In *22nd Eur. Symp. Program.*, pages 539–558. LNCS 7211, 2012.