# Client-Server Sessions in Linear Logic

Zesen Qian [1] **Alex Kavvos** [2] Lars Birkedal [1]

[1]Aarhus University

[2]University of Bristol

Proofs and Algorithms Seminar
Laboratoire d'informatique de l'École Polytechnique
23 June 2021

# Concurrency through Logic

**Objective**: to understand and reason about concurrency.

Many approaches:

- process calculus (e.g. CSP, CCS, $\pi$-calculus, ...)
- concurrent separation logic (e.g. Iris)
- Petri nets
- event structures

and so on.

In the early 1990s, Samson Abramsky asked:

$$\boxed{\text{Is there a Curry-Howard correspondence for concurrency?}}$$

The answer was meant to be

process calculus $\approx$ proofs in Girard's **Classical Linear Logic**

# Proofs as processes

Early attempts: Abramsky (circa 1991), Bellin and Scott (TCS 1994).

2010s: a breakthrough.

- ILL: Caires and Pfenning (CONCUR 2010, MSCS 2016)
- CLL: Wadler (ICFP 2012, JFP 2014)
- Hypersequent CLL: Kokke et al. (POPL 2019)

These term assignment systems are compelling process calculi.

**Benefits**:

- deadlock-freedom ($\approx$ progress)
- session fidelity/polite conversation ($\approx$ preservation)
- livelock-freedom ($\approx$ termination)

**Drawbacks**: one thing in common:

$\boxed{\text{the expressivity of these systems is remarkably poor}}$

**Programme**: increase expressivity, without losing good properties.

# Table of Contents


Classical Processes

Fixed points, exponentials, and coexponentials

Client-Server Linear Logic

Examples

Related and future work

# I. Classical Processes

# Classical Processes

CP: a typed $\pi$-calculus. Type system: Classical Linear Logic (CLL).
Formulas of CLL are *session types*.

$$
\begin{array}{rcll}
A, B, \ldots & ::= & \bot & \text{(receive end-of-session signal)} \\
& | & \mathbf{1} & \text{(send end-of-session signal)} \\
& | & A \parr B & \text{(input } A \text{ and continue as } B\text{)} \\
& | & A \otimes B & \text{(output } A \text{ and continue as } B\text{)} \\
& | & A \mathbin{\&} B & \text{(offer choice of } A \text{ or } B\text{)} \\
& | & A \oplus B & \text{(select one of } A \text{ or } B\text{)} \\
& | & ?A & \text{(???)} \\
& | & !A & \text{(???)}
\end{array}
$$

Duality:

$$(A \otimes B)^\perp \stackrel{\text{def}}{=} A^\perp \parr B^\perp \quad (A \oplus B)^\perp \stackrel{\text{def}}{=} A^\perp \mathbin{\&} B^\perp \quad (?A)^\perp \stackrel{\text{def}}{=} !A^\perp$$

We have $A^{\perp\perp} = A$.

# Type system: classical linear logic

$\boxed{P \vdash x : A}$ "$P$ will communicate along channel $x$ according to $A$."

$$\frac{P \vdash \Gamma, x : A \qquad Q \vdash \Delta, x : A^{\perp}}{\nu x.\,(P \mid Q) \vdash \Gamma, \Delta} \qquad \frac{}{x \leftrightarrow y \vdash x : A^{\perp}, y : A}$$

$$\frac{P \vdash \Gamma, x : A, y : B}{y(x).\,P \vdash \Gamma, y : A \parr B} \qquad \frac{P \vdash \Gamma, x : A \qquad Q \vdash \Delta, y : B}{y[x].\,(P \mid Q) \vdash \Gamma, \Delta, y : A \otimes B}$$

$$\frac{P \vdash \Gamma, x : A}{x[\mathrm{inl}].\,P \vdash \Gamma, x : A \oplus B} \qquad \frac{Q \vdash \Gamma, y : B}{y[\mathrm{inr}].\,Q \vdash \Gamma, y : A \oplus B}$$

$$\frac{P \vdash \Gamma, x : A \qquad Q \vdash \Gamma, x : B}{x.\mathrm{case}\{P; Q\} \vdash \Gamma, x : A \,\&\, B}$$

| | | |
|---|---|---|
| $A \parr B$ | input $A$ and continue as $B$ | **connected concurrency** |
| $A \otimes B$ | output $A$ and continue as $B$ | **disjoint concurrency** |

# Exponentials

$$\frac{P \vdash \Gamma}{P \vdash \Gamma, x : ?A} \qquad \frac{P \vdash \Gamma, x : ?A, y : ?A}{P[x/y] \vdash \Gamma, x : ?A} \qquad \frac{P \vdash \Gamma, y : A}{?x[y].\, P \vdash \Gamma, x : ?A}$$

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

**Wadler's interpretation**: ! means server, ? means client

- weakening = no clients
- dereliction = one client
- contraction = many clients
- promotion = server
- $!A$ = replicable service obeying protocol $A$
- $?A$ = pool of clients, each of which obeys protocol $A$

# Exponentials: *not* server-client

$$\frac{P \vdash \Gamma}{P \vdash \Gamma, x : ?A}$$

"No clients in pool"?
But there is *at least* one process here!

$$\frac{P \vdash \Gamma, x : ?A, y : ?A}{P[x/y] \vdash \Gamma, x : ?A}$$

"More than one clients in the pool"?
But there is *only one* process here!

Suppose I have $\begin{cases} P \vdash z : A \\ Q \vdash w : A \end{cases}$. How to combine them into a pool?

$$\dfrac{\dfrac{\dfrac{P \vdash z : A}{?x[z].\, P \vdash x : ?A}\ ?d \qquad \dfrac{Q \vdash w : A}{?y[w].\, P \vdash y : ?A}\ ?d}{?x[z].\, P \mid ?y[w].\, Q \vdash x : ?A, y : ?A}\ \textsc{Mix}}{?x[z].\, P \mid ?x[w].\, Q \vdash x : ?A}\ ?c$$

# Mix and co.

To accommodate client pools, Wadler is forced to admit Mix.

$$\frac{P \vdash \Gamma \qquad Q \vdash \Delta}{P \mid Q \vdash \Gamma, \Delta}\ \textsc{Mix} \qquad\qquad \frac{}{\mathsf{stop} \vdash \cdot}\ \textsc{Mix0}$$

Cf. with tensor rule, the term for which is $x[y].\,(P \mid Q)$.

Mix0 looks like inconsistency. It allows for an empty client pool:

$$\frac{\dfrac{}{\mathsf{stop} \vdash \cdot}\ \textsc{Mix0}}{\mathsf{stop} \vdash x : ?A}\ ?w$$

Surprisingly, this allows some client/server-like behaviour.

# What's the deal with Mɪx?

Girard wavered on whether it should be included in CLL.

### Theorem

*The following are logically interderivable.*

1. $\bot \multimap \mathbf{1}$ *and Mɪx.*
2. $\mathbf{1} \multimap \bot$ *and Mɪx0.*
3. $A \otimes B \multimap A \parr B$ *and Mɪx.*

In terms of *propositions as sessions*:

- 1+2 conflate the units (OK)
- 3 states that output implies input (???)

But there is also another thing…

# A Slippery Slope

Suppose we have two clients

$$P \vdash z : A \qquad\qquad Q \vdash w : A$$

Using Mix: $P \mid Q \vdash z : A, w : A$. So $w(z).\,(P \mid Q) \vdash w : A \parr A$.

A corresponding server that we can cut with this must have type

$$S \vdash v : A^{\perp} \otimes A^{\perp}$$

$\therefore$ The two clients will be served by disjoint server components!

**Solution**: to write stateful server code we must also accept

$$A \parr B \multimap A \otimes B$$

Then $\otimes = \parr$, which is a *compact closed setting* $\Longrightarrow$ Deadlock

# II. Fixed points, exponentials, and coexponentials

# Exponentials as Fixed Points

We often think of $!A$ as an infinite supply of $A$'s.

Can the exponential $!A$ be given as a fixed point?

$$!A \;\cong\; \mathbf{1} \,\&\, A \,\&\, (!A \otimes !A) \tag{$*$}$$

Certainly a logical equivalence (by the structural rules; circa 1987).
But can it be an *isomorphism* on the level of proofs?

The rules of $!A$ ensure that the exponential is *uniform*.

In other words: each dereliction to $A$ evaluates to same proof of $A$.
Cf. the embedding of intuitionistic logic into linear logic:

$$(A \to B)^\star \stackrel{\text{def}}{=} !A^\star \multimap B^\star$$

It would be unacceptable if the various uses of $!A^\star$ were not uniform.
…yet nothing in $(*)$ guarantees uniformity!

# Ignoring uniformity

That does not stop us from considering a non-uniform exponential!

Take Baelde's system for LL with fixed points (ACM ToCL 2012).

Specializing it to the LFP given by $?A \;\cong\; \bot \oplus A \oplus (?A \parr ?A)$:

$$\frac{\vdash \Gamma}{\vdash \Gamma, ?A} \qquad\qquad \frac{\Gamma, ?A, ?A}{\Gamma, ?A} \qquad\qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, ?A}$$

These are the usual rules.

Specializing it to the GFP given by $!A \;\cong\; \mathbf{1} \mathbin{\&} A \mathbin{\&} (!A \otimes !A)$:

$$\frac{\vdash \Gamma, B \qquad \vdash B^{\perp}, \mathbf{1} \qquad \vdash B^{\perp}, A \qquad \vdash B^{\perp}, B \otimes B}{\vdash \Gamma, !A}$$

**Not** promotion; looks like a comonoid instead. Coinductive!

# Upside down

$$!A \;\cong\; \mathbf{1} \,\&\, A \,\&\, (!A \otimes !A)$$

It is the **consumer's choice** ($\&$) to pick one of three:

- nothing, i.e. $\mathbf{1}$
- just an $A$
- recursively obtain two **disjoint components** of that type

$\therefore$ $!A$ cannot be a stateful server.

What if the components were not disjoint, but connected?

$$¡A \;\cong\; \bot \,\&\, A \,\&\, (¡A \parr ¡A)$$

It is the **consumer's choice** ($\&$) to pick one of three:

- nothing, i.e. $\bot$
- just an $A$
- recursively obtain two **connected components** of that type

# Coexponentials

Specializing Baelde's system to

$$¿A \;\cong\; \mathbf{1} \oplus A \oplus (¿A \otimes ¿A) \qquad ¡A \;\cong\; \bot \,\&\, A \,\&\, (¡A \parr ¡A)$$

we obtain the following rules.

$$\frac{}{\vdash ¿A}\,¿w \qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, ¿A}\,¿d \qquad \frac{\vdash \Gamma, ¿A \qquad \vdash \Delta, ¿A}{\vdash \Gamma, \Delta, ¿A}\,¿c$$

$$\frac{\vdash \Gamma, B \qquad \vdash B^{\bot}, \bot \qquad \vdash B^{\bot}, A \qquad \vdash B^{\bot}, B \parr B}{\vdash \Gamma, ¡A}\,¡$$

¿ means client
¡ means server

# III. Client-Server Linear Logic

## Clients and Servers

Seeking a “linear logic” that can model client/server interactions.

- A server with **state**, and
- a **pool of clients**,
- which **races** to **nondeterministically** connect to the server
- at a **unique endpoint**,
- holding **atomic** access to the server state.
- No Mix

We will use coexponentials, but with a twist: **lists instead of trees**.

# The server rule

Original tree-shaped rule:

$$\frac{\vdash \Gamma, B \qquad \vdash B^{\perp}, \perp \qquad \vdash B^{\perp}, A \qquad \vdash B^{\perp}, B \parr B}{\vdash \Gamma, ¡A}$$

Replace with a list-shaped rule:

$$\frac{\vdash \Gamma, B \qquad \vdash B^{\perp}, \perp \qquad \vdash B^{\perp}, A \parr B}{\vdash \Gamma, ¡A}$$

Simplify, and replace $\perp$ with an arbitrary $\Delta$ (logically equivalent):

$$\frac{\vdash \Gamma, B \qquad \vdash B^{\perp}, \Delta \qquad \vdash B^{\perp}, A, B}{\vdash \Gamma, \Delta, ¡A}$$

Reasons for the list-shape:

- there is a single global 'logical' server state
- servers don't (always) fork children to serve clients

# The client rules

Original tree-shaped rules:

$$\frac{}{\vdash ¿A}\ ¿w \qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, ¿A}\ ¿d \qquad \frac{\vdash \Gamma, ¿A \qquad \vdash \Delta, ¿A}{\vdash \Gamma, \Delta, ¿A}\ ¿c$$

To correspond with the list-like ¡ rule we must use

$$\frac{}{\vdash ¿A} \qquad \frac{\vdash \Gamma, ¿A \qquad \vdash \Delta, A}{\vdash \Gamma, \Delta, ¿A}$$

To **generate nondeterminism** we quotient the permutations away:

$$\frac{\dfrac{\vdash \Gamma, ¿A \qquad \vdash \Delta, A}{\vdash \Gamma, \Delta, ¿A} \qquad \vdash \Sigma, A}{\vdash \Gamma, \Delta, \Sigma, ¿A} \equiv \frac{\dfrac{\vdash \Gamma, ¿A \qquad \vdash \Sigma, A}{\vdash \Gamma, \Sigma, ¿A} \qquad \vdash \Delta, A}{\vdash \Gamma, \Delta, \Sigma, ¿A}$$

This last is included as a **structural equivalence**.

# Clients and Servers

$$\frac{}{\vdash ¿A} \qquad\qquad \frac{\vdash \Gamma, ¿A \qquad \vdash \Delta, A}{\vdash \Gamma, \Delta, ¿A}$$

$$\frac{\vdash \Gamma, B \qquad \vdash B^{\perp}, \Delta \qquad \vdash B^{\perp}, A, B}{\vdash \Gamma, \Delta, ¡A}$$

Cutting a (non-empty) $¿A$ with a $¡A$ causes the following events:

- The "state' $B$ spawns an $A$ session and a 'next state' $B$.
- The spawned $A$ nondeterministically connects to some client.
- The server is re-initialised with the next $B$ state…
- …and re-connected to the remaining pool (minus one client).

When there are no clients left $\vdash B^{\perp}, \Delta$ is used to 'terminate.'

# Client-Server Linear Logic

CSLL is based on Kokke, Montesi and Peressoti's HCP (POPL 2019).
It uses **hyperenvironments** to decompose the terms of CP.

$$\frac{P \vdash \mathcal{G} \mid \Gamma, x : A \mid \Delta, y : A^{\perp}}{\nu xy.\, P \vdash \mathcal{G} \mid \Gamma, \Delta} \textsc{Cut}$$

$$\frac{P \vdash \Gamma, x : A \mid \Delta, y : B}{y[x].\, P \vdash \Gamma, \Delta, y : A \otimes B} \textsc{Tensor} \qquad \frac{P \vdash \mathcal{G}}{¿x[].\, P \vdash \mathcal{G} \mid x : ¿A} \textsc{QueW}$$

$$\frac{P \vdash \mathcal{G} \mid \Gamma, x : ¿A \mid \Delta, y : A}{¿x[y].\, P \vdash \mathcal{G} \mid \Gamma, \Delta, x : ¿A} \textsc{QueA}$$

$$\frac{P \vdash \mathcal{G} \mid \Gamma, i : B \mid \Delta, f : B^{\perp} \qquad Q \vdash z : B^{\perp}, z' : B, y' : A}{¡y\{z, z', y'.\, Q\}(i, f).\, P \vdash \mathcal{G} \mid \Gamma, \Delta, y : ¡A} \textsc{Claro}$$

These come with a **reaction relation** $P \longrightarrow Q$.
**Theorem**: $\longrightarrow$ satisfies progress and preservation.

# IV. Examples

# Compare-and-Set (CAS)

A very powerful concurrent primitive!

### Definition

A register implements **compare-and-set** if:

- There is an instruction $\textsc{Cas}(e, d)$. ($e$ = expected, $d$ = desirable)
- When a thread runs $\textsc{Cas}(e, d)$:
  - If the register equals $e$: it is set to $d$, and the inst. returns $\textsc{True}$.
  - If not: register stays put, the function returns $\textsc{False}$.
- Threads **race** to perform $\textsc{Cas}(e_i, d_i)$ **atomically**.

Server = register. Client pool = threads racing to perform a $\textsc{Cas}$.
Letting $\mathbf{2} \stackrel{\text{def}}{=} \mathbf{1} \oplus \mathbf{1}$, the client protocol is

$$¿(\mathbf{2} \otimes \mathbf{2} \otimes \mathbf{2}^{\perp} \parr \mathbf{1})$$

(The server protocol is the dual of this.)
Access to CAS register is not atomic, but **causally atomic**.

# CSGV = linear FP + session types + clients and servers

CSLL is a low-level language. Need higher-level notation.

$$T, \ldots \ ::= \quad T \multimap T \mid T \to T \mid T + T \mid T \otimes T \mid \mathsf{Unit} \mid T_S$$

$$\begin{array}{lll} T_S, \ldots \ ::= & !T.T_S & \text{(output value of type } T\text{, then behave as } T_S\text{)} \\ & \mid\ ?T.T_S & \text{(input value of type } T\text{, then behave as } T_S\text{)} \\ & \mid\ T_S \oplus T_S \mid\ T_S \,\&\, T_s & \text{(select from options, offer choice)} \\ & \mid\ \mathsf{end}_? \mid \mathsf{end}_! & \text{(end-of-session)} \\ & \mid\ ¿T_S \mid\ ¡T_s & \text{(request or serve } T_S \text{ session)} \end{array}$$

We can write programs that

- control access to a **shared functional data structure**
- implement **nondeterministic choice**
- implement **fork-join parallelism**
- implement **Keynes' beauty contest**

$\Longrightarrow$ **sharing** and **nondeterminism**, without deadlock!

# V. Related and future work

# Client/Server interaction

**All** previous CLL-based approaches use Mɪx.
(Wadler, JFP 2014; Atkey et al, WadlerFest 2016; Caires and Pérez, ESOP 2017; Kokke et al, POPL 2019)

**Outlier**: Kokke, Morris and Wadler (LMCS 2020) use **bounded linear logic** (# of clients in pool bounded).

Two totally different approaches:

- **multiparty session types**
- Kobayashi's (priority-based) **type systems for the $\pi$-calculus**

Some deep connections to **differential linear logic**; add rules

$$\frac{\vdash \Gamma, !A \qquad \vdash \Delta, !A}{\vdash \Gamma, \Delta, !A} \qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, !A} \qquad \frac{}{\vdash !A} \qquad \frac{\vdash \Gamma \qquad \ldots \qquad \vdash \Gamma}{\vdash \Gamma}$$

DILL can embed finitary $\pi$-calculus (Ehrhard and Laurent, IC 2010)
…but criticized by Mazza (MSCS 2018) for being **confusion-free**.

# Future work

- Weakening the coexponential rule to 'match' promotion gives

$$\frac{\vdash \bigotimes ¿\Gamma, A}{\vdash \bigotimes ¿\Gamma, ¡A}\ ¡$$

  How to **eliminate cuts**?
- Develop a **logical relations** toolkit for CLL and CSLL, with a view to reasoning about programs (and proving termination)

Thank you!