Asynchronous Session-Based Concurrency: Deadlock-freedom in Cyclic Process Networks

Bas van den Heuvel and **Jorge A. Pérez**

University of Groningen, The Netherlands

March 25, 2024

UNIFYING
C•RRECTNESS FOR
C•MMUNICATING
S•FTWARE

Formal verification for message-passing concurrency.

- Static approach, based on **protocols** expressed as **session types**.
- An important but elusive problem: **deadlock-freedom**.
  Ensuring that message-passing programs that never "get stuck".
- Deadlocks are central to many concurrency bugs in practice.

Formal verification for message-passing concurrency.

- Static approach, based on **protocols** expressed as **session types**.
- An important but elusive problem: **deadlock-freedom**.
  Ensuring that message-passing programs that never "get stuck".
- Deadlocks are central to many concurrency bugs in practice.

Case in point: Leesatapornwongsa et al.'s taxonomy of concurrency bugs in cloud-scale distributed systems (ASPLOS'16).

- *Bugs linger in concurrent executions of multiple protocols.*
  Many background protocols beyond user-facing foreground protocols.
- *Bugs triggered by an untimely message delivery that commits order violation or atomicity violation.*

Plan for Today

Here: A process calculi approach to correct, deadlock-free programs.

- Define a core language with concurrency, with a simple typing discipline;
- Compile programs into process calculi specifications; use this abstract level to enforce deadlock-freedom using advanced types;
- Transfer deadlock-freedom guarantees, based on strong connections between the core language and its process interpretation.

Plan for Today

Here: A process calculi approach to correct, deadlock-free programs.

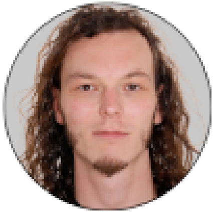
- Define a core language with concurrency, with a simple typing discipline;
- Compile programs into process calculi specifications; use this abstract level to enforce deadlock-freedom using advanced types;
- Transfer deadlock-freedom guarantees, based on strong connections between the core language and its process interpretation.

More concretely:

- $\mathtt{LAST}^n$: A core language with functions and asynchronous concurrency
- The expressivity of $\mathtt{LAST}^n$, by example
- A session type system for $\mathtt{LAST}^n$ (and its limitations)
- $\mathtt{APCP}$: A typed calculus of concurrency with deadlock-freedom by typing
- Transference of deadlock-freedom from $\mathtt{APCP}$ to $\mathtt{LAST}^n$

Origin of the results

- The PhD thesis of Bas van den Heuvel (currently a postdoc in Germany with Peter Thiemann and Martin Sulzmann).
- The thesis: "Correctly Communicating Software: Distributed, Asynchronous, and Beyond".
  Available online; to be publicly defended on April 2nd.
- Preliminary results on ICE'21, EXPRESS/SOS'22, and SCP'22.



Key Ideas

- A call-by-name variant of LAST (Linear Asynchronous Session Types) by Gay and Vasconcelos (JFP, 2010)
- Explicit substitutions neatly "delay" substitutions within a term (runtime syntax)
- Explicit closing of sessions with dedicated garbage collection of buffers
- Sequential terms can communicate when organized within configurations
- Types ensure protocol fidelity and communication safety but not deadlock-freedom

Syntax

The syntax of terms (M, N) combines standard functional constructs (call-by-name) with primitives for communication and concurrency:

$M, N ::=$	x	variable
$\mid$	$()$	unit value
$\mid$	$\lambda x.M$	abstraction
$\mid$	$M\ N$	application
$\mid$	(M, N)	construct pair
$\mid$	$\mathtt{let}\,(x, y) = M\,\mathtt{in}\,N$	deconstruct pair
$\mid$	$M\{\!\!\{N/x\}\!\!\}$	explicit substitution

Syntax

The syntax of terms (M, N) combines standard functional constructs (call-by-name) with primitives for communication and concurrency:

$$
\begin{array}{rcl|ll}
M, N & ::= & x & \mathtt{new} & \text{create new channel} \\
& \mid & () & \mathtt{spawn}\, M; N & \text{spawn } M \text{ in parallel to } N \\
& \mid & \lambda x.M & \mathtt{send}\, M\ N & \text{send } M \text{ along } N \\
& \mid & M\ \ N & \mathtt{recv}\, M & \text{receive along } M \\
& \mid & (M, N) & \mathtt{select}\, \ell\, M & \text{select label } \ell \text{ along } M \\
& \mid & \mathtt{let}\,(x, y) = M\,\mathtt{in}\,N & \mathtt{case}\, M\, \mathtt{of}\, \{i : M\}_{i \in I} & \text{offer labels in } I \text{ along } M \\
& \mid & M \{\!\!\{ N/x \}\!\!\} & \mathtt{close}\, M; N & \text{close } M
\end{array}
$$

Running Example: A Bookshop Scenario

A three-party protocol: a mother interacting with a bookshop to buy a book for her son.

- The shop receives a booktitle and then offers a choice between buying the book or freely accessing its blurb.
- If the client decides to buy, the shop receives credit card information and sends the book to the client. Otherwise, if the blurb is requested, the shop sends its text.
- Here the son delegates his session to her mother, who will complete the purchase from the shop.

$\texttt{LAST}^n$ Running Example: A Bookshop Scenario

A three-party protocol: a mother interacting with a bookshop to buy a book for her son.

- The shop receives a booktitle and then offers a choice between buying the book or freely accessing its blurb.
- If the client decides to buy, the shop receives credit card information and sends the book to the client. Otherwise, if the blurb is requested, the shop sends its text.
- Here the son delegates his session to her mother, who will complete the purchase from the shop.

We define two different sessions: one connects the son with the shop, another the mother with her son. Using a different term per participant, we have the configuration:

$$\begin{aligned}\mathsf{Sys} \triangleq \blacklozenge\ & \texttt{let}\,(s, s') = \texttt{new}\,\texttt{in}\,\texttt{spawn}\,\mathsf{Shop}_s;\\ & \texttt{let}\,(m, m') = \texttt{new}\,\texttt{in}\,\texttt{spawn}\,\mathsf{Mother}_m;\\ & \quad \mathsf{Son}_{s',m'}\end{aligned}$$

$\mathtt{LAST}^n$ Running Example: A Bookshop Scenario

The code for the son, which returns the result:

$$
\begin{aligned}
\mathsf{Son}_{s',m'} \triangleq\ & \mathtt{let}\, s'_1 = \mathtt{send}\ \text{“Dune”}\ s'\ \mathtt{in} \\
& \quad \mathtt{let}\, s'_2 = \mathtt{select}\, \mathsf{buy}\ s'_1\ \mathtt{in} \\
& \quad\quad \mathtt{let}\, m'_1 = \mathtt{send}\, s'_2\, m'\ \mathtt{in} \\
& \quad\quad\quad \mathtt{let}\, (\mathit{book}, m'_2) = \mathtt{recv}\, m'_1\ \mathtt{in} \\
& \quad\quad\quad\quad \mathtt{close}\, m'_2; \mathit{book}
\end{aligned}
$$

$\mathtt{LAST}^n$ Running Example: A Bookshop Scenario

The code for the son, which returns the result:

$$
\begin{aligned}
\mathsf{Son}_{s',m'} \triangleq\ & \mathtt{let}\ s'_1 = \mathtt{send}\ \text{“Dune”}\ s'\ \mathtt{in} \\
& \mathtt{let}\ s'_2 = \mathtt{select}\ \mathsf{buy}\ s'_1\ \mathtt{in} \\
& \mathtt{let}\ m'_1 = \mathtt{send}\ s'_2\ m'\ \mathtt{in} \\
& \mathtt{let}\ (\textit{book}, m'_2) = \mathtt{recv}\ m'_1\ \mathtt{in} \\
& \mathtt{close}\ m'_2; \textit{book}
\end{aligned}
$$

The code for the mother:

$$
\begin{aligned}
\mathsf{Mother}_{m} \triangleq\ & \mathtt{let}\ (x, m_1) = \mathtt{recv}\ m\ \mathtt{in} \\
& \mathtt{let}\ x_1 = \mathtt{send}\ \mathsf{visa}\ x\ \mathtt{in} \\
& \mathtt{let}\ (\textit{book}, x_2) = \mathtt{recv}\ x_1\ \mathtt{in} \\
& \mathtt{let}\ m_2 = \mathtt{send}\ \textit{book}\ m_1\ \mathtt{in} \\
& \mathtt{close}\ m_2; \mathtt{close}\ x_2; ()
\end{aligned}
$$

$\mathtt{LAST}^n$ Running Example: A Bookshop Scenario

The code for the shop:

$$
\begin{aligned}
\mathsf{Shop}_s \triangleq\ & \mathtt{let}\,(\mathit{title}, s_1) = \mathtt{recv}\, s\ \mathtt{in} \\
& \mathtt{case}\, s_1\ \mathtt{of}\ \{\mathsf{buy} : \lambda s_2.\mathtt{let}\,(\mathit{card}, s_3) = \mathtt{recv}\, s_2\ \mathtt{in} \\
& \qquad\qquad\qquad\qquad \mathtt{let}\, s_4 = \mathtt{send}\,\mathsf{book}(\mathit{title})\, s_3\ \mathtt{in} \\
& \qquad\qquad\qquad\qquad \mathtt{close}\, s_4; (), \\
& \qquad\quad \mathsf{blurb} : \lambda s_2.\mathtt{let}\, s_3 = \mathtt{send}\,\mathsf{blurb}(\mathit{title})\, s_2\ \mathtt{in} \\
& \qquad\qquad\qquad\qquad \mathtt{close}\, s_3; ()\}
\end{aligned}
$$

Again, the code for the son:

$$
\begin{aligned}
\mathsf{Son}_{s',m'} \triangleq\ & \mathtt{let}\, s'_1 = \mathtt{send}\ \text{“Dune”}\ s'\ \mathtt{in} \\
& \mathtt{let}\, s'_2 = \mathtt{select}\,\mathsf{buy}\, s'_1\ \mathtt{in} \\
& \mathtt{let}\, m'_1 = \mathtt{send}\, s'_2\, m'\ \mathtt{in} \\
& \mathtt{let}\,(\mathit{book}, m'_2) = \mathtt{recv}\, m'_1\ \mathtt{in} \\
& \mathtt{close}\, m'_2; \mathit{book}
\end{aligned}
$$

Semantics

How to give semantics to our language? Our design is in two levels:

- Term reduction, noted $\longrightarrow_{\mathtt{M}}$, handles functional operations.
- Communicating terms are organized in configurations, equipped with a dedicated reduction relation, noted $\longrightarrow_{\mathtt{C}}$.
- Hence, parallel threads and asynchronous (i.e., buffered) communication are handled at the level of configurations.

LAST^n Semantics

- We define configurations (C, D, E) building upon terms, using markers (ϕ) and messages (m, n):

$$\phi ::= \blacklozenge \mid \lozenge$$

$$m, n ::= M \mid \ell$$

$$C, D, E ::= \phi\, M \mid C \parallel D \mid (\boldsymbol{\nu} x[\vec{m}\rangle y)C \mid C\{\!\!\{ M/x \}\!\!\}$$

- Reduction uses contexts for terms ($\mathcal{R}$), threads ($\mathcal{F}$), and configurations ($\mathcal{G}$):

$$\begin{aligned}
\mathcal{R} &::= [\cdot] \mid \mathcal{R}\ M \mid \texttt{send}\, M\, \mathcal{R} \mid \texttt{recv}\, \mathcal{R} \mid \texttt{let}\, (x, y) = \mathcal{R}\, \texttt{in}\, M \\
&\quad \mid \texttt{select}\, \ell\, \mathcal{R} \mid \texttt{case}\, \mathcal{R}\, \texttt{of}\, \{i : M\}_{i \in I} \mid \texttt{close}\, \mathcal{R}; M \mid \mathcal{R}\{\!\!\{ M/x \}\!\!\} \\
\mathcal{F} &::= \phi\, \mathcal{R} \\
\mathcal{G} &::= [\cdot] \mid \mathcal{G} \parallel C \mid (\boldsymbol{\nu} x[\vec{m}\rangle y)\mathcal{G} \mid \mathcal{G}\{\!\!\{ M/x \}\!\!\}
\end{aligned}$$

$\mathtt{LAST}^n$ Semantics

Rules for term reduction ($\longrightarrow_{\mathtt{M}}$) and structural congruence for terms ($\equiv_{\mathtt{M}}$):

$$\frac{\text{[RED-LAM]}}{(\lambda x.M)\ N \longrightarrow_{\mathtt{M}} M\{\!\![N/x]\!\!\}}$$

$$\frac{\text{[RED-PAIR]}}{\mathtt{let}\,(x,y) = (M_1, M_2)\,\mathtt{in}\,N \longrightarrow_{\mathtt{M}} N\{\!\![M_1/x, M_2/y]\!\!\}}$$

$$\frac{\text{[RED-NAME-SUB]}}{x\{\!\![M/x]\!\!\} \longrightarrow_{\mathtt{M}} M}$$

$$\text{[RED-LIFT]}\ \frac{M \longrightarrow_{\mathtt{M}} N}{\mathcal{R}[M] \longrightarrow_{\mathtt{M}} \mathcal{R}[N]}$$

$$\text{[SC-SUB-EXT]}\ \frac{x \notin \mathrm{fv}(\mathcal{R})}{(\mathcal{R}[M])\{\!\![N/x]\!\!\} \equiv_{\mathtt{M}} \mathcal{R}\,[M\{\!\![N/x]\!\!\}]}$$

$$\text{[RED-LIFT-SC]}\ \frac{M \equiv_{\mathtt{M}} M' \qquad M' \longrightarrow_{\mathtt{M}} N' \qquad N' \equiv_{\mathtt{M}} N}{M \longrightarrow_{\mathtt{M}} N}$$

Semantics

Some rules for configuration reduction ($\longrightarrow_{\mathtt{C}}$) use special thread contexts, denoted $\hat{\mathcal{F}}$, which do not affect variables bound by explicit substitutions:

$$\frac{[\text{RED-NEW}]}{\mathcal{F}[\mathtt{new}] \longrightarrow_{\mathtt{C}} (\boldsymbol{\nu} x[\varepsilon\rangle y)(\mathcal{F}[(x, y)])}$$

$$\frac{[\text{RED-SEND}]}{(\boldsymbol{\nu} x[\vec{m}\rangle y)(\hat{\mathcal{F}}[\mathtt{send}\, M\; x] \parallel C) \longrightarrow_{\mathtt{C}} (\boldsymbol{\nu} x[M, \vec{m}\rangle y)(\hat{\mathcal{F}}[x] \parallel C)}$$

$$\frac{[\text{RED-RECV}]}{(\boldsymbol{\nu} x[\vec{m}, M\rangle y)(\hat{\mathcal{F}}[\mathtt{recv}\, y] \parallel C) \longrightarrow_{\mathtt{C}} (\boldsymbol{\nu} x[\vec{m}\rangle y)(\hat{\mathcal{F}}[(M, y)] \parallel C)}$$

Additional rules for configuration reduction ($\longrightarrow_{\mathtt{C}}$):

$$\frac{[\textsc{Red-Select}]}{(\boldsymbol{\nu} x[\vec{m}\rangle y)(\mathcal{F}[\mathtt{select}\,\ell\,x] \parallel C) \longrightarrow_{\mathtt{C}} (\boldsymbol{\nu} x[\ell, \vec{m}\rangle y)(\mathcal{F}[x] \parallel C)}$$

$$[\textsc{Red-Case}]\quad \frac{j \in I}{(\boldsymbol{\nu} x[\vec{m}, j\rangle y)(\mathcal{F}[\mathtt{case}\, y\, \mathtt{of}\, \{i : M_i\}_{i\in I}] \parallel C) \longrightarrow_{\mathtt{C}} (\boldsymbol{\nu} x[\vec{m}\rangle y)(\mathcal{F}[M_j\ y] \parallel C)}$$

Rules for configuration reduction ($\longrightarrow_{\mathtt{C}}$) that enforce garbage-collection of closed sessions:

[RED-CLOSE]
$$\frac{}{(\boldsymbol{\nu} x[\vec{m}\rangle y)(\mathcal{F}[\mathtt{close}\,x; M] \parallel C) \longrightarrow_{\mathtt{C}} (\boldsymbol{\nu}\Box[\vec{m}\rangle y)(\mathcal{F}[M] \parallel C)}$$

[RED-RES-NIL]
$$\frac{}{(\boldsymbol{\nu}\Box[\epsilon\rangle\Box)C \longrightarrow_{\mathtt{C}} C}$$

[RED-PAR-NIL]
$$\frac{}{C \parallel \diamond\,() \longrightarrow_{\mathtt{C}} C}$$

Additional rules for configuration reduction ($\longrightarrow_{\mathtt{C}}$):

$$\frac{}{\hat{\mathcal{F}}[\mathtt{spawn}\, M; N] \longrightarrow_{\mathtt{C}} \hat{\mathcal{F}}[N] \parallel \diamond\, M}\ [\textsc{red-spawn}]$$

$$\frac{C \longrightarrow_{\mathtt{C}} C'}{\mathcal{G}[C] \longrightarrow_{\mathtt{C}} \mathcal{G}[C']}\ [\textsc{red-lift-C}]$$

$$\frac{M \longrightarrow_{\mathtt{M}} M'}{\mathcal{F}[M] \longrightarrow_{\mathtt{C}} \mathcal{F}[M']}\ [\textsc{red-lift-M}]$$

$$\frac{C \equiv_{\mathtt{C}} C' \qquad C' \longrightarrow_{\mathtt{C}} D' \qquad D' \equiv_{\mathtt{C}} D}{C \longrightarrow_{\mathtt{C}} D}\ [\textsc{red-conf-lift-SC}]$$

$\mathtt{LAST}^n$ A Simple Example

$$
\begin{aligned}
(\lambda x.x\ (\lambda y.y))\ ((\lambda w.w)\ (\lambda z.z)) &\longrightarrow_{\mathtt{M}} (x\ (\lambda y.y))\{\!\!\{((\lambda w.w)\ (\lambda z.z))/x\}\!\!\} \\
&\equiv_{\mathtt{M}} (x\{\!\!\{((\lambda w.w)\ (\lambda z.z))/x\}\!\!\})\ (\lambda y.y) \\
&\longrightarrow_{\mathtt{M}} ((\lambda w.w)\ (\lambda z.z))\ (\lambda y.y) \\
&\longrightarrow_{\mathtt{M}} (w\{\!\!\{(\lambda z.z)/w\}\!\!\})\ (\lambda y.y) \\
&\longrightarrow_{\mathtt{M}} (\lambda z.z)\ (\lambda y.y) \\
&\longrightarrow_{\mathtt{M}} z\{\!\!\{(\lambda y.y)/z\}\!\!\} \\
&\longrightarrow_{\mathtt{M}} \lambda y.y
\end{aligned}
$$

Observe how β-reduction induces explicit substitutions, which are "pushed inside" reduction contexts.

The Bookshop Scenario, Revisited

The entire system:

$$\begin{aligned}\mathsf{Sys} \triangleq \blacklozenge\ & \texttt{let}\,(s, s') = \texttt{new}\,\texttt{in}\,\texttt{spawn}\,\mathsf{Shop}_s;\\ & \texttt{let}\,(m, m') = \texttt{new}\,\texttt{in}\,\texttt{spawn}\,\mathsf{Mother}_m;\\ & \quad \mathsf{Son}_{s',m'}\end{aligned}$$

The code for the shop:

$$\begin{aligned}\mathsf{Shop}_s \triangleq\ & \texttt{let}\,(\mathit{title}, s_1) = \texttt{recv}\,s\,\texttt{in}\\ & \texttt{case}\ s_1\ \texttt{of}\ \{\mathsf{buy} : \lambda s_2.\texttt{let}\,(\mathit{card}, s_3) = \texttt{recv}\,s_2\,\texttt{in}\\ & \qquad\qquad\qquad \texttt{let}\,s_4 = \texttt{send}\,\mathsf{book}(\mathit{title})\,s_3\,\texttt{in}\\ & \qquad\qquad\qquad \texttt{close}\,s_4; (),\\ & \qquad\quad \mathsf{blurb} : \lambda s_2.\texttt{let}\,s_3 = \texttt{send}\,\mathsf{blurb}(\mathit{title})\,s_2\,\texttt{in}\\ & \qquad\qquad\qquad \texttt{close}\,s_3; ()\}\end{aligned}$$

$$
\begin{aligned}
\mathsf{Sys} &= \blacklozenge\ \mathtt{let}\,(s,s') = \mathtt{new}\,\mathtt{in}\ldots \\
&\longrightarrow_{\mathsf{C}} (\boldsymbol{\nu} y[\varepsilon\rangle y')\blacklozenge\ \mathtt{let}\,(s,s') = (y,y')\,\mathtt{in}\ldots \\
&\longrightarrow_{\mathsf{C}} (\boldsymbol{\nu} y[\varepsilon\rangle y')\blacklozenge\ \mathtt{spawn}\,\mathsf{Shop}_s;\ldots\{\!\!\{y/s, y'/s'\}\!\!\} \\
&\equiv_{\mathsf{C}} (\boldsymbol{\nu} y[\varepsilon\rangle y')\big((\blacklozenge\ \mathtt{spawn}\,\mathsf{Shop}_s;\ldots)\{\!\!\{y/s, y'/s'\}\!\!\}\big) \\
&\longrightarrow_{\mathsf{C}} (\boldsymbol{\nu} y[\varepsilon\rangle y')\big((\blacklozenge\ \mathtt{let}\,(m,m') = \mathtt{new}\,\mathtt{in}\ldots \parallel \lozenge\,\mathsf{Shop}_s)\{\!\!\{y/s, y'/s'\}\!\!\}\big) \\
&\equiv_{\mathsf{C}} (\boldsymbol{\nu} y[\varepsilon\rangle y')\big((\blacklozenge\ \mathtt{let}\,(m,m') = \mathtt{new}\,\mathtt{in}\ldots \parallel \\
&\qquad\qquad\qquad \lozenge\ \mathtt{let}\,(\mathit{title}, s_1) = \mathtt{recv}\,(s\{\!\!\{y/s\}\!\!\})\,\mathtt{in}\ldots)\{\!\!\{y'/s'\}\!\!\}\big) \\
&\longrightarrow_{\mathsf{C}} (\boldsymbol{\nu} y[\varepsilon\rangle y')\big((\blacklozenge\ \mathtt{let}\,(m,m') = \mathtt{new}\,\mathtt{in}\ldots \parallel \lozenge\ \mathtt{let}\,(\mathit{title}, s_1) = \mathtt{recv}\,y\,\mathtt{in}\ldots)\{\!\!\{y'/s'\}\!\!\}\big) \\
&\longrightarrow^{2}_{\mathsf{C}} (\boldsymbol{\nu} y[\varepsilon\rangle y')\big((\boldsymbol{\nu} z[\varepsilon\rangle z')(\blacklozenge\ \mathtt{spawn}\,\mathsf{Mother}_m;\mathsf{Son}_{s',m'})\{\!\!\{z/m, z'/m', y'/s'\}\!\!\} \parallel \lozenge\,\mathsf{Shop}_y\big) \\
&\longrightarrow_{\mathsf{C}} (\boldsymbol{\nu} y[\varepsilon\rangle y')\big((\boldsymbol{\nu} z[\varepsilon\rangle z')\big(\blacklozenge\,\mathsf{Son}_{s',m'}\{\!\!\{z'/m', y'/s'\}\!\!\} \parallel \lozenge\,\mathsf{Mother}_m\{\!\!\{z/m\}\!\!\}\big) \parallel \lozenge\,\mathsf{Shop}_y\big) \\
&\equiv_{\mathsf{C}} (\boldsymbol{\nu} y[\varepsilon\rangle y')\big((\boldsymbol{\nu} z[\varepsilon\rangle z')\big(\blacklozenge\,\mathsf{Son}_{s',m'}\{\!\!\{z'/m', y'/s'\}\!\!\} \parallel \lozenge\ \mathtt{let}\,(x, m_1) = \mathtt{recv}\,(m\{\!\!\{z/m\}\!\!\})\,\mathtt{in}\ldots\big) \mid \\
&\longrightarrow_{\mathsf{C}} (\boldsymbol{\nu} y[\varepsilon\rangle y')\big((\boldsymbol{\nu} z[\varepsilon\rangle z')\big(\blacklozenge\,\mathsf{Son}_{s',m'}\{\!\!\{z'/m', y'/s'\}\!\!\} \parallel \\
&\qquad\qquad\qquad\qquad\qquad \lozenge\ \mathtt{let}\,(x, m_1) = \mathtt{recv}\,z\,\mathtt{in}\ldots\big) \parallel \lozenge\,\mathsf{Shop}_y\big) \\
&= (\boldsymbol{\nu} y[\varepsilon\rangle y')\big((\boldsymbol{\nu} z[\varepsilon\rangle z')\big(\blacklozenge\,\mathsf{Son}_{s',m'}\{\!\!\{z'/m', y'/s'\}\!\!\} \parallel \lozenge\,\mathsf{Mother}_z\big) \parallel \lozenge\,\mathsf{Shop}_y\big) =: \mathsf{Sys}^1
\end{aligned}
$$

Type System

Types include functional types (T, U) and session types for communication (S):

$$
\begin{array}{rcl r}
T, U & ::= & T \times U & \text{pair} \\
 & \mid & T \multimap U & \text{function} \\
 & \mid & \mathbf{1} & \text{unit} \\
 & \mid & S & \text{session}
\end{array}
\qquad
\begin{array}{rcl r}
S & ::= & !T.S & \text{send} \\
 & \mid & ?T.S & \text{receive} \\
 & \mid & \oplus\{i : T\}_{i \in I} & \text{select} \\
 & \mid & \&\{i : T\}_{i \in I} & \text{branch} \\
 & \mid & \mathsf{end} &
\end{array}
$$

Aligned with our semantics, we use '$\square$' to denote the session type for endpoints that have been already closed.

Type System

Given a session type S, its dual type $\overline{S}$ characterizes compatible behaviors.
In defining duality, only the continuations of send and receive types are dualized.

$$\overline{!T.S} = ?T.\overline{S} \qquad \overline{?T.S} = !T.\overline{S}$$

$$\overline{\oplus\{i : S_i\}_{i \in I}} = \&\{i : \overline{S_i}\}_{i \in I} \qquad \overline{\&\{i : S_i\}_{i \in I}} = \oplus\{i : \overline{S_i}\}_{i \in I} \qquad \overline{\mathsf{end}} = \mathsf{end}$$

Typing Judgments

The type system has three layers: typing for terms, for buffers, and for configurations.

- Judgments for terms:

$$\Gamma \vdash_{\mathtt{M}} M : T$$

  where the typing context Γ is a list of variable-type assignments $x : T$.

Typing Judgments

The type system has three layers: typing for terms, for buffers, and for configurations.

- Judgments for terms:

$$\Gamma \vdash_{\mathtt{M}} M : T$$

  where the typing context Γ is a list of variable-type assignments $x : T$.

- Judgments for buffered channels:

$$\Gamma \vdash_{\mathtt{B}} [\vec{m}\rangle : S' > S$$

  where S denotes a sequence of sends and selections corresponding to the values and labels in $\vec{m}$, after which the type continues as S'.

Typing Judgments

The type system has three layers: typing for terms, for buffers, and for configurations.

- Judgments for terms:

$$\Gamma \vdash_{\mathtt{M}} M : T$$

where the typing context Γ is a list of variable-type assignments $x : T$.

- Judgments for buffered channels:

$$\Gamma \vdash_{\mathtt{B}} [\vec{m}\rangle : S' > S$$

where S denotes a sequence of sends and selections corresponding to the values and labels in $\vec{m}$, after which the type continues as S'.

- Judgments for configurations:

$$\Gamma \vdash^{\phi}_{\mathtt{C}} C : T$$

where ϕ says whether C contains the main thread ($\phi = \blacklozenge$) or child threads ($\phi = \lozenge$).

$\mathtt{LAST}^n$ Typing Rules (1/4)

$$\frac{}{x : T \vdash_{\mathtt{M}} x : T}\ [\text{TYP-VAR}]$$

$$\frac{\Gamma, x : T \vdash_{\mathtt{M}} M : U}{\Gamma \vdash_{\mathtt{M}} \lambda x.M : T \multimap U}\ [\text{TYP-ABS}]$$

$$\frac{\Gamma \vdash_{\mathtt{M}} M : T \multimap U \qquad \Delta \vdash_{\mathtt{M}} N : T}{\Gamma, \Delta \vdash_{\mathtt{M}} M\ N : U}\ [\text{TYP-APP}]$$

$$\frac{\Gamma \vdash_{\mathtt{M}} M : T \qquad \Delta \vdash_{\mathtt{M}} N : U}{\Gamma, \Delta \vdash_{\mathtt{M}} (M, N) : T \times U}\ [\text{TYP-PAIR}]$$

$$\frac{\Gamma \vdash_{\mathtt{M}} M : T \times T' \qquad \Delta, x : T, y : T' \vdash_{\mathtt{M}} N : U}{\Gamma, \Delta \vdash_{\mathtt{M}} \mathtt{let}\ (x, y) = M\ \mathtt{in}\ N : U}\ [\text{TYP-SPLIT}]$$

$$\frac{}{\emptyset \vdash_{\mathtt{M}} () : \mathbf{1}}\ [\text{TYP-UNIT}]$$

$$\frac{\Gamma, x : T \vdash_{\mathtt{M}} M : U \qquad \Delta \vdash_{\mathtt{M}} N : T}{\Gamma, \Delta \vdash_{\mathtt{M}} M\{\!\!\{ N/x \}\!\!\} : U}\ [\text{TYP-SUB}]$$

Typing Rules (2/4)

[TYP-NEW]
$$\frac{}{\emptyset \vdash_{\mathtt{M}} \mathtt{new} : S \times \overline{S}}$$

[TYP-SEND]
$$\frac{\Gamma \vdash_{\mathtt{M}} M : T \qquad \Delta \vdash_{\mathtt{M}} N : !T.S}{\Gamma, \Delta \vdash_{\mathtt{M}} \mathtt{send}\, M\, N : S}$$

[TYP-SEL]
$$\frac{\Gamma \vdash_{\mathtt{M}} M : \oplus\{i : S_i\}_{i \in I} \qquad j \in I}{\Gamma \vdash_{\mathtt{M}} \mathtt{select}\, j\, M : S_j}$$

[TYP-RECV]
$$\frac{\Gamma \vdash_{\mathtt{M}} M : ?T.S}{\Gamma \vdash_{\mathtt{M}} \mathtt{recv}\, M : T \times S}$$

[TYP-CASE]
$$\frac{\Gamma \vdash_{\mathtt{M}} M : \&\{i : S_i\}_{i \in I} \qquad \forall i \in I.\ \Delta \vdash_{\mathtt{M}} N_i : S_i \multimap U}{\Gamma, \Delta \vdash_{\mathtt{M}} \mathtt{case}\, M\, \mathtt{of}\, \{i : N_i\}_{i \in I} : U}$$

LAST^n Typing Rules (3/4)

[TYP-CLOSE]
$$\frac{\Gamma \vdash_{\mathtt{M}} M : \mathsf{end} \qquad \Delta \vdash_{\mathtt{M}} N : T}{\Gamma, \Delta \vdash_{\mathtt{M}} \mathtt{close}\, M; N : T}$$

[TYP-SPAWN]
$$\frac{\Gamma \vdash_{\mathtt{M}} M : \mathbf{1} \qquad \Delta \vdash_{\mathtt{M}} N : T}{\Gamma, \Delta \vdash_{\mathtt{M}} \mathtt{spawn}\, M; N : T}$$

We need rules for buffers and "half-closed" sessions:

[TYP-BUF]
$$\frac{}{\emptyset \vdash_{\mathtt{B}} [\epsilon\rangle : S' > S'}$$

[TYP-BUF-SEND]
$$\frac{\Gamma \vdash_{\mathtt{M}} M : T \qquad \Delta \vdash_{\mathtt{B}} [\vec{m}\rangle : S' > S}{\Gamma, \Delta \vdash_{\mathtt{B}} [\vec{m}, M\rangle : S' > !T.S}$$

[TYP-BUF-SEL]
$$\frac{\Gamma \vdash_{\mathtt{B}} [\vec{m}\rangle : S' > S_j \qquad j \in I}{\Gamma \vdash_{\mathtt{B}} [\vec{m}, j\rangle : S' > \oplus\{i : S_i\}_{i \in I}}$$

[TYP-BUF-END-L]
$$\frac{}{\emptyset \vdash_{\mathtt{B}} [\varepsilon\rangle : \mathsf{end} > \square}$$

[TYP-BUF-END-R]
$$\frac{}{\emptyset \vdash_{\mathtt{B}} [\varepsilon\rangle : \square > \mathsf{end}}$$

$\mathtt{LAST}^n$ Typing Rules (4/4)

Below, $\hat{T}$ denotes a non-session type:

$$[\text{TYP-MAIN}]\ \frac{\Gamma \vdash_{\mathtt{M}} M : \hat{T}}{\Gamma \vdash_{\mathtt{C}}^{\blacklozenge} \blacklozenge M : \hat{T}}$$

$$[\text{TYP-PAR}]\ \frac{\Gamma \vdash_{\mathtt{C}}^{\phi_1} C : T_1 \qquad \Delta \vdash_{\mathtt{C}}^{\phi_2} D : T_2}{\Gamma, \Delta \vdash_{\mathtt{C}}^{\phi_1 + \phi_2} C \parallel D : T_1 + T_2}$$

$$[\text{TYP-CHILD}]\ \frac{\Gamma \vdash_{\mathtt{M}} M : \mathbf{1}}{\Gamma \vdash_{\mathtt{C}}^{\lozenge} \lozenge M : \mathbf{1}}$$

$$[\text{TYP-RES}]\ \frac{\Gamma \vdash_{\mathtt{B}} [\vec{m}\rangle : S' > S \qquad \Delta, x : S' \vdash_{\mathtt{C}}^{\phi} C : T \qquad \Gamma', y : \overline{S} = \Gamma, \Delta}{\Gamma' \vdash_{\mathtt{C}}^{\phi} (\boldsymbol{\nu} x[\vec{m}\rangle y)C : T}$$

$$[\text{TYP-CONF-SUB}]\ \frac{\Gamma, x : T \vdash_{\mathtt{C}}^{\phi} C : U \qquad \Delta \vdash_{\mathtt{M}} M : T}{\Gamma, \Delta \vdash_{\mathtt{C}}^{\phi} C\{\!\!\{M/x\}\!\!\} : U}$$

The Booking Scenario

Consider the system where all session interactions have taken place, and all three threads are ready to close their sessions:

$$
\begin{aligned}
\mathsf{Sys} &\longrightarrow^*_{\mathtt{C}} (\boldsymbol{\nu} y[\varepsilon\rangle y')\big((\boldsymbol{\nu} z'[\varepsilon\rangle z)(\blacklozenge\ \mathtt{close}\, z'; \mathsf{book}(\text{“Dune”}) \parallel \lozenge\ \mathtt{close}\, z; \mathtt{close}\, y') \parallel \lozenge\, \mathtt{close}\, y\big)\\
&\longrightarrow_{\mathtt{C}} \blacklozenge\, \mathsf{book}(\text{“Dune”}) \parallel (\boldsymbol{\nu} y[\varepsilon\rangle y')\big((\boldsymbol{\nu} \square[\varepsilon\rangle z)\lozenge\ \mathtt{close}\, z; \mathtt{close}\, y' \parallel \lozenge\, \mathtt{close}\, y\big)\\
&\equiv_{\mathtt{C}} \blacklozenge\, \mathsf{book}(\text{“Dune”}) \parallel (\boldsymbol{\nu} y[\varepsilon\rangle y')\big((\boldsymbol{\nu} z[\varepsilon\rangle \square)\lozenge\ \mathtt{close}\, z; \mathtt{close}\, y' \parallel \lozenge\, \mathtt{close}\, y\big)\\
&\longrightarrow_{\mathtt{C}} \blacklozenge\, \mathsf{book}(\text{“Dune”}) \parallel (\boldsymbol{\nu} y[\varepsilon\rangle y')\big((\boldsymbol{\nu} \square[\varepsilon\rangle \square)\lozenge\ \mathtt{close}\, y' \parallel \lozenge\, \mathtt{close}\, y\big)\\
&\longrightarrow_{\mathtt{C}} \blacklozenge\, \mathsf{book}(\text{“Dune”}) \parallel (\boldsymbol{\nu} y[\varepsilon\rangle y')\big(\lozenge\ \mathtt{close}\, y' \parallel \lozenge\, \mathtt{close}\, y\big)\\
&\equiv_{\mathtt{C}} \blacklozenge\, \mathsf{book}(\text{“Dune”}) \parallel (\boldsymbol{\nu} y'[\varepsilon\rangle y)\big(\lozenge\ \mathtt{close}\, y' \parallel \lozenge\, \mathtt{close}\, y\big)\\
&\longrightarrow_{\mathtt{C}} \blacklozenge\, \mathsf{book}(\text{“Dune”}) \parallel \lozenge\, () \parallel (\boldsymbol{\nu} \square[\varepsilon\rangle y)\lozenge\, \mathtt{close}\, y\\
&\longrightarrow_{\mathtt{C}} \blacklozenge\, \mathsf{book}(\text{“Dune”}) \parallel (\boldsymbol{\nu} \square[\varepsilon\rangle y)\lozenge\, \mathtt{close}\, y\\
&\equiv_{\mathtt{C}} \blacklozenge\, \mathsf{book}(\text{“Dune”}) \parallel (\boldsymbol{\nu} y[\varepsilon\rangle \square)\lozenge\, \mathtt{close}\, y \qquad (*)\\
&\longrightarrow_{\mathtt{C}} \blacklozenge\, \mathsf{book}(\text{“Dune”}) \parallel (\boldsymbol{\nu} \square[\varepsilon\rangle \square)\lozenge() \longrightarrow_{\mathtt{C}} \blacklozenge\, \mathsf{book}(\text{“Dune”}) \parallel \lozenge() \longrightarrow_{\mathtt{C}} \blacklozenge\, \mathsf{book}(\text{“Dune”})
\end{aligned}
$$

Example of Typing

We illustrate the typing of half-closed sessions on the configuration

$$\blacklozenge\,\mathsf{book}(\text{“Dune”}) \parallel (\boldsymbol{\nu} y[\varepsilon\rangle\square)\lozenge\,\mathtt{close}\,y$$

We write B (book) to denote a primitive non-linear type that can be weakened/contracted at will and is self-dual. We have:

$$\dfrac{\dfrac{\emptyset \vdash_{\mathtt{M}} \mathsf{book}(\text{“Dune”}) : \mathsf{B}}{\emptyset \vdash^{\blacklozenge}_{\mathtt{C}} \blacklozenge\,\mathsf{book}(\text{“Dune”}) : \mathsf{B}}\,[\text{T-MAIN}] \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{y : \mathsf{end} \vdash_{\mathtt{M}} y : \mathsf{end}}\,[\text{T-VAR}] \quad \dfrac{}{\emptyset \vdash_{\mathtt{M}} () : \mathbf{1}}\,[\text{T-UNIT}]}{y : \mathsf{end} \vdash_{\mathtt{M}} \mathtt{close}\,y; () : \mathbf{1}}\,[\text{T-CLOSE}]}{y : \mathsf{end} \vdash^{\lozenge}_{\mathtt{C}} \lozenge\,\mathtt{close}\,y; () : \mathbf{1}}\,[\text{T-CHILD}] \quad \dfrac{}{\emptyset \vdash_{\mathtt{B}} [\varepsilon\rangle : \mathsf{end} > \square}\,[\text{T-BL}]}{\emptyset \vdash^{\lozenge}_{\mathtt{C}} (\boldsymbol{\nu} y[\varepsilon\rangle\square)\lozenge\,\mathtt{close}\,y; () : \mathbf{1}}\,[\text{T-RES}]}{\emptyset \vdash^{\blacklozenge}_{\mathtt{C}} \blacklozenge\,\mathsf{book}(\text{“Dune”}) \parallel (\boldsymbol{\nu} y[\varepsilon\rangle\square)\lozenge\,\mathtt{close}\,y; () : \mathsf{B}}\,[\text{T-PAR}]$$

Guarantees Derived From Typing

Theorem (Type Preservation for $\texttt{LAST}^n$)

Given $\Gamma \vdash^{\phi}_{\texttt{C}} C : T$, *if* $C \equiv_{\texttt{C}} D$ *or* $C \longrightarrow_{\texttt{C}} D$, *then* $\Gamma \vdash^{\phi}_{\texttt{C}} D : T$.

This theorem entails protocol fidelity and communication safety, but not deadlock-freedom.

$\mathtt{LAST}^n$ Typing Does Not Exclude Deadlocks

- Consider the term:

$$\begin{aligned} M_{a,b} := \mathtt{let}\, a_1 &= \mathtt{send}\,()\, a\, \mathtt{in} \\ \mathtt{let}\,(v, b_1) &= \mathtt{recv}\, b\, \mathtt{in} \\ &\mathtt{close}\, a_1; \mathtt{close}\, b_1; v \end{aligned}$$

- $M_{a,b}$ sends on a, receives on b, and then closes both sessions.

Typing Does Not Exclude Deadlocks

- Consider the term:

$$\begin{aligned} M_{a,b} := \mathtt{let}\, a_1 &= \mathtt{send}\, ()\, a\, \mathtt{in} \\ \mathtt{let}\, (v, b_1) &= \mathtt{recv}\, b\, \mathtt{in} \\ &\mathtt{close}\, a_1; \mathtt{close}\, b_1; v \end{aligned}$$

- $M_{a,b}$ sends on a, receives on b, and then closes both sessions. Now consider C:

$$\begin{aligned} C := \blacklozenge\, \mathtt{let}\, (x, x') &= \mathtt{new}\, \mathtt{in} \\ \mathtt{let}\, (y, y') &= \mathtt{new}\, \mathtt{in} \\ &\mathtt{spawn}\, M_{x,y}; M_{y',x'} \end{aligned}$$

- Intuitively, we would like the two threads to communicate. However, they get stuck:

$$M_{x,y} \longrightarrow_{\mathtt{M}} \big(\mathtt{let}\,(v, y_1) = \mathtt{recv}\, y\, \mathtt{in} \ldots\big)\{\!\![\mathtt{send}\,()\,x/x_1]\!\!\} =: M'_{x,y} \not\longrightarrow_{\mathtt{M}}$$

$$M_{y',x'} \longrightarrow_{\mathtt{M}} \big(\mathtt{let}\,(v', x'_1) = \mathtt{recv}\, x'\, \mathtt{in} \ldots\big)\{\!\![\mathtt{send}\,()\,y'/y'_1]\!\!\} =: M'_{y',x'} \not\longrightarrow_{\mathtt{M}}$$

$$C \longrightarrow^9_{\mathtt{C}} (\boldsymbol{\nu} s[\varepsilon\rangle s')(\boldsymbol{\nu} t[\varepsilon\rangle t')\big(\lozenge\, M'_{x,t}\{\!\![s/x]\!\!\} \parallel \blacklozenge\, M'_{y',s'}\{\!\![t'/y']\!\!\}\big) \not\longrightarrow_{\mathtt{C}}$$

Typing Does Not Exclude Deadlocks

Clearly, there are deadlock-free alternatives to $M_{a,b}$. For instance:

$$
\begin{aligned}
N_{a,b} := \mathtt{let}\, a_1 = \mathtt{send}\, ()\, a\, \mathtt{in}\\
\mathtt{close}\, a_1;\\
\mathtt{let}\, (v, b_1) = \mathtt{recv}\, b\, \mathtt{in}\\
\mathtt{close}\, b_1; v
\end{aligned}
$$

We would like a general technique that excludes deadlocked configurations such as C. We could either

1. Strengthen the type system of $\mathtt{LAST}^n$ so as to exclude deadlocks
2. Transfer the deadlock-freedom guarantee from an external type system

APCP Asynchronous Priority-based Classical Processes

- In prior work, we developed APCP: a session type system for π-calculus processes.
- Key features: *cyclic process networks*, *asynchronous communication*, and *recursion*.
- Extends the Curry-Howard correspondences between linear logic and session types.
- Priorities on types are used to rule out **circular dependencies** in processes (Kobayashi, 2006; Padovani, 2014; Dardha and Gay, 2018).
- Key properties: *session fidelity*, *communication safety*, and *deadlock-freedom*.
- APCP is expressive enough for a decentralized analysis of Multiparty Session Types (cf. our journal paper at SCP'22).

Syntax

Process syntax:

$P, Q ::= x[a,b]$	send	$\mid$	$x(y,z);P$	receive
$\mid\ x[b] \triangleleft \ell$	selection	$\mid$	$x(z) \triangleright \{i : P\}_{i \in I}$	branch
$\mid\ (\boldsymbol{\nu} xy)P$	restriction	$\mid$	$P \mid Q$	parallel
$\mid\ \mathbf{0}$	inaction	$\mid$	$[x \leftrightarrow y]$	forwarder
$\mid\ \mu X(\tilde{z});P$	recursive definition	$\mid$	$X\langle \tilde{z} \rangle$	recursive call

Derivable constructs We use the following syntactic sugar:

$$\overline{x}[y] \cdot P := (\boldsymbol{\nu} ya)(\boldsymbol{\nu} zb)(x[a,b] \mid P\{z/x\}) \qquad \overline{x} \triangleleft \ell \cdot P := (\boldsymbol{\nu} zb)(x[b] \triangleleft \ell \mid P\{z/x\})$$

$$x(y);P := x(y,z);P\{z/x\} \qquad x \triangleright \{i : P_i\}_{i \in I} := x(z) \triangleright \{i : P_i\{z/x\}\}_{i \in I}$$

Reduction Semantics

[RED-SEND-RECV]

$$\frac{}{(\boldsymbol{\nu} xy)(x[a,b] \mid y(z,y');Q) \longrightarrow Q\{a/z, b/y'\}}$$

[RED-SEL-BRA]

$$\frac{j \in I}{(\boldsymbol{\nu} xy)(x[b] \triangleleft j \mid y(y') \triangleright \{i : Q_i\}_{i \in I}) \longrightarrow Q_j\{b/y'\}}$$

[RED-FWD]

$$\frac{y \neq z}{(\boldsymbol{\nu} xy)([x \leftrightarrow z] \mid P) \longrightarrow P\{z/y\}}$$

[RED-CONG]

$$\frac{P \equiv P' \qquad P' \longrightarrow Q' \qquad Q' \equiv Q}{P \longrightarrow Q}$$

[RED-RES]

$$\frac{P \longrightarrow Q}{(\boldsymbol{\nu} xy)P \longrightarrow (\boldsymbol{\nu} xy)Q}$$

[RED-PAR]

$$\frac{P \longrightarrow Q}{P \mid R \longrightarrow Q \mid R}$$

Reduction Semantics

Consider process P:

$$P \triangleq (\boldsymbol{\nu} zu)\big((\boldsymbol{\nu} xy)\big((\boldsymbol{\nu} ax')(x[v_1, a] \mid x'[v_2, b]) \\ \mid (\boldsymbol{\nu} cz')(z[v_3, c] \mid y(w_1, y'); y'(w_2, y''); Q)\big) \\ \mid u(w_3, u'); R\big)$$

Or, using the sugared syntax:

$$P = (\boldsymbol{\nu} zu)((\boldsymbol{\nu} xy)(\overline{x}[v_1] \cdot \overline{x}[v_2] \cdot \mathbf{0} \mid \overline{z}[v_3] \cdot y(w_1); y(w_2); Q') \mid u(w_3); R')$$

where $Q' \triangleq Q\{y/y''\}$ and $R' \triangleq R\{u/u'\}$.

Reduction Semantics

Consider process P:

$$P \triangleq (\boldsymbol{\nu} zu)\big((\boldsymbol{\nu} xy)\big((\boldsymbol{\nu} ax')(x[v_1, a] \mid x'[v_2, b]) \\ \mid (\boldsymbol{\nu} cz')(z[v_3, c] \mid y(w_1, y'); y'(w_2, y''); Q)\big) \\ \mid u(w_3, u'); R\big)$$

Or, using the sugared syntax:

$$P = (\boldsymbol{\nu} zu)((\boldsymbol{\nu} xy)(\overline{x}[v_1] \cdot \overline{x}[v_2] \cdot \mathbf{0} \mid \overline{z}[v_3] \cdot y(w_1); y(w_2); Q') \mid u(w_3); R')$$

where $Q' \triangleq Q\{y/y''\}$ and $R' \triangleq R\{u/u'\}$.

We have:

$$P \longrightarrow (\boldsymbol{\nu} zu)((\boldsymbol{\nu} xy)(\overline{x}[v_2] \cdot \mathbf{0} \mid \overline{z}[v_3] \cdot y(w_2); Q'\{v_1/w_1\}) \mid u(w_3); R')$$

$$P \longrightarrow (\boldsymbol{\nu} xy)(\overline{x}[v_1] \cdot \overline{x}[v_2] \cdot \mathbf{0} \mid y(w_1); y(w_2); Q') \mid R'\{v_3/w_3\}$$

Note: There is no reduction involving from P the send on x', since x' is connected to the continuation name of the send on x and is thus not (yet) paired with a dual receive.

APCP Type System

APCP types processes by assigning binary session types to names.

- We write $\circ, \pi, \rho, \ldots$ to denote priorities.
- Also, we use ω to denote the ultimate priority that is greater than all other priorities and cannot be increased further. That is, $\forall \circ \in \mathbb{N}.\ \omega > \circ$ and $\forall \circ \in \mathbb{N}.\ \omega + \circ = \omega$.
- Session types (linear logic propositions) include priorities:

$$A, B ::= A \otimes^\circ B \mid A \parr^\circ B \mid \oplus^\circ \{i : A\}_{i \in I} \mid \&^\circ \{i : A\}_{i \in I} \mid \bullet \mid \mu X.A \mid X$$

  where $\bullet$ denotes the self-dual type for 'end'.
- The *dual* of session type A, denoted $\overline{A}$, is defined inductively as follows:

$$\overline{A \otimes^\circ B} \triangleq \overline{A} \parr^\circ \overline{B} \qquad \overline{\oplus^\circ \{i : A_i\}_{i \in I}} \triangleq \&^\circ \{i : \overline{A_i}\}_{i \in I} \qquad \overline{\bullet} \triangleq \bullet \qquad \overline{\mu X.A} \triangleq \mu X.\overline{A}$$

$$\overline{A \parr^\circ B} \triangleq \overline{A} \otimes^\circ \overline{B} \qquad \overline{\&^\circ \{i : A_i\}_{i \in I}} \triangleq \oplus^\circ \{i : \overline{A_i}\}_{i \in I} \qquad \overline{X} \triangleq X$$

APCP Type System

The typing rules ensure that prefixes with lower priority are not blocked by prefixes with higher priority.

Essential laws:

1. Sends and selections with priority $\circ$ must have continuations/payloads with priority strictly larger than $\circ$;
2. A prefix with priority $\circ$ must be prefixed only by receives and branches with priority strictly smaller than $\circ$;
3. Dual prefixes leading to a synchronization must have equal priorities.

Type System

The typing rules ensure that prefixes with lower priority are not blocked by prefixes with higher priority.

Essential laws:

1. Sends and selections with priority $\circ$ must have continuations/payloads with priority strictly larger than $\circ$;
2. A prefix with priority $\circ$ must be prefixed only by receives and branches with priority strictly smaller than $\circ$;
3. Dual prefixes leading to a synchronization must have equal priorities.

Judgments are of the form $\Omega \vdash P :: \Gamma$, where:

- P is a process;
- Γ is a context that assigns types to channels $(x : A)$;
- Ω is a context that assigns tuples of types to recursion variables $(X : (A, B, \ldots))$.

Typing Rules (Selected)

[TYP-SEND]
$$\frac{\circ < \mathrm{pr}(A), \mathrm{pr}(B)}{\Omega \vdash x[y,z] :: x : A \otimes^{\circ} B, y : \overline{A}, z : \overline{B}}$$

[TYP-RECV]
$$\frac{\Omega \vdash P :: \Gamma, y : A, z : B \qquad \circ < \mathrm{pr}(\Gamma)}{\Omega \vdash x(y,z); P :: \Gamma, x : A \parr^{\circ} B}$$

[TYP-END]
$$\frac{\Omega \vdash P :: \Gamma}{\Omega \vdash P :: \Gamma, x : \bullet}$$

[TYP-PAR]
$$\frac{\Omega \vdash P :: \Gamma \qquad \Omega \vdash Q :: \Delta}{\Omega \vdash P \mid Q :: \Gamma, \Delta}$$

[TYP-RES]
$$\frac{\Omega \vdash P :: \Gamma, x : A, y : \overline{A}}{\Omega \vdash (\boldsymbol{\nu} xy) P :: \Gamma}$$

[TYP-SEND⋆]
$$\frac{\Omega \vdash P :: \Gamma, y : A, x : B \qquad \circ < \mathrm{pr}(A), \mathrm{pr}(B)}{\Omega \vdash \overline{x}[y] \cdot P :: \Gamma, x : A \otimes^{\circ} B}$$

Typing by Example

We give the typing of the two consecutive sends on x (omitting the context Ω):

$$\dfrac{\dfrac{\dfrac{\circ < \mathrm{pr}(A_1), \pi}{\vdash x[v_1, a] :: x : A_1 \otimes^{\circ} A_2 \otimes^{\pi} B,\ v_1 : \overline{A_1}, a : \overline{A_2 \otimes^{\pi} B}}\ [\text{TYP-SEND}] \qquad \dfrac{\pi < \mathrm{pr}(A_2), \mathrm{pr}(B)}{\vdash x'[v_2, b] :: x' : A_2 \otimes^{\pi} B,\ v_2 : \overline{A_2}, b : \overline{B}}\ [\text{TYP-SEND}]}{\vdash x[v_1, a] \mid x'[v_2, b] :: v_1 : \overline{A_1}, v_2 : \overline{A_2}, b : \overline{B}, x : A_1 \otimes^{\circ} A_2 \otimes^{\pi} B,\ a : \overline{A_2 \otimes^{\pi} B}, x' : A_2 \otimes^{\pi} B}\ [\text{TYP-PAR}]}{\vdash (\boldsymbol{\nu} a x')(x[v_1, a] \mid x'[v_2, b]) :: v_1 : \overline{A_1}, v_2 : \overline{A_2}, b : \overline{B}, {\color{red} x : A_1 \otimes^{\circ} A_2 \otimes^{\pi} B}}\ [\text{TYP-RES}]$$

Typing by Example

Let us type the consecutive inputs on y, i.e., the subprocess $y(w_1, y'); y'(w_2, y''); Q$.

Because x and y are dual names in P, the type of y should be dual to the type of x:

$$\dfrac{\dfrac{\vdash Q :: \Gamma, w_1 : \overline{A_1}, w_2 : \overline{A_2}, y'' : \overline{B} \qquad \pi < \mathrm{pr}(\Gamma, w_1 : \overline{A_1})}{\vdash y'(w_2, y''); Q :: \Gamma, w_1 : \overline{A_1}, y' : \overline{A_2} \parr^{\pi} \overline{B}}\,[\textsc{typ-recv}] \qquad \circ < \mathrm{pr}(\Gamma)}{\vdash y(w_1, y'); y'(w_2, y''); Q :: \Gamma, y : \overline{A_1} \parr^{\circ} \overline{A_2} \parr^{\pi} \overline{B}}\,[\textsc{typ-recv}]$$

These two derivations tell us that

$$\circ < \pi < \mathrm{pr}(A_1), \mathrm{pr}(A_2), \mathrm{pr}(B), \mathrm{pr}(\Gamma)$$

Properties Derived From Typing

In APCP, type preservation corresponds to the elimination of (top-level) applications of Rule [TYPE-RES].

Theorem (Subject Reduction, Simplified)

If $\Omega \vdash P :: \Gamma$ *and* $P \longrightarrow Q$*, then there exists* Γ' *such that* $\Omega \vdash Q :: \Gamma'$*.*

APCP Properties Derived From Typing

- We say a process is **deadlocked** if it is not the inactive process and cannot reduce.
- Following Dardha and Gay, we target the elimination of [TYPE-RES].
- In APCP, Rule [TYPE-RES] is key in our sugared notation to bind asynchronous sends/selections and their continuations.
  These occurrences of [TYPE-RES] cannot be eliminated via reduction.

APCP Properties Derived From Typing

- We say a process is **deadlocked** if it is not the inactive process and cannot reduce.
- Following Dardha and Gay, we target the elimination of [TYPE-RES].
- In APCP, Rule [TYPE-RES] is key in our sugared notation to bind asynchronous sends/selections and their continuations.
  These occurrences of [TYPE-RES] cannot be eliminated via reduction.

To formulate deadlock-freedom, we use two auxiliary notions:

- The **active names** of P, denoted $\mathrm{an}(P)$:
  the set of (free) names that are used for non-blocked communications (send, receive, selection, branch)
- **Evaluation contexts**, denoted $\mathcal{E}$.

Properties Derived From Typing

Definition (Live Process)

A process P is *live*, denoted $\text{live}(P)$, if

1. there are names x, y and process P' such that $P \equiv (\boldsymbol{\nu} xy)P'$ with $x, y \in \text{an}(P')$, or
2. there are names x, y, z and process P' such that $P \equiv \mathcal{E}\big[(\boldsymbol{\nu} yz)([x \leftrightarrow y] \mid P')\big]$ and $z \neq x$ (i.e., the forwarder is independent).

Lemma

If $\emptyset \vdash P :: \emptyset$ and P is not live, then P must be $\mathbf{0}$.

Theorem (Progress)

If $\emptyset \vdash P :: \Gamma$ and $\text{live}(P)$, then there is a process Q such that $P \longrightarrow Q$.

Theorem (Deadlock-freedom)

If $\emptyset \vdash P :: \emptyset$, then either $P \equiv \mathbf{0}$ or $P \longrightarrow Q$ for some Q.

Translating $\texttt{LAST}^n$ into $\texttt{APCP}$ Key Ideas

To translate $\texttt{LAST}^n$ into $\texttt{APCP}$, we follow Milner's translation of the lazy λ-calculus.

- In $\texttt{LAST}^n$, variables are (i) placeholders for future substitutions and (ii) access points to buffered channels.
- Accordingly, we translate variables as $\texttt{APCP}$ endpoints that (i) enable the translation of explicit substitutions and (ii) enable interaction with the translation of buffers

Translating $\mathtt{LAST}^n$ into APCP Key Ideas

To translate $\mathtt{LAST}^n$ into APCP, we follow Milner's translation of the lazy λ-calculus.

- In $\mathtt{LAST}^n$, variables are (i) placeholders for future substitutions and (ii) access points to buffered channels.
- Accordingly, we translate variables as APCP endpoints that (i) enable the translation of explicit substitutions and (ii) enable interaction with the translation of buffers

Given a configuration C, we define an APCP process $[\![C]\!]z$, where z is a fresh name. We also define translations of types and buffers.

Translating $\texttt{LAST}^n$ into $\texttt{APCP}$ Key Ideas

To translate $\texttt{LAST}^n$ into $\texttt{APCP}$, we follow Milner's translation of the lazy λ-calculus.

- In $\texttt{LAST}^n$, variables are (i) placeholders for future substitutions and (ii) access points to buffered channels.
- Accordingly, we translate variables as $\texttt{APCP}$ endpoints that (i) enable the translation of explicit substitutions and (ii) enable interaction with the translation of buffers

Given a configuration C, we define an $\texttt{APCP}$ process $[\![C]\!]z$, where z is a fresh name. We also define translations of types and buffers.

We establish correctness for our translation following Gorla's correctness criteria:

Completeness Given $\Gamma \vdash^{\phi}_{\texttt{C}} C : T$, if $C \longrightarrow_{\texttt{C}} D$, then $[\![C]\!]z \longrightarrow^* [\![D]\!]z$.

Soundness Given $\Gamma \vdash^{\phi}_{\texttt{C}} C : T$, if $[\![C]\!]z \longrightarrow^* Q$, then there exists D such that $C \longrightarrow^*_{\texttt{C}} D$ and $Q \longrightarrow^* [\![D]\!]z$.

Soundness is critical to transfer deadlock-freedom from $\texttt{APCP}$ to $\texttt{LAST}^n$

Translating Types

Our typed translation takes a typed term $\Gamma \vdash_{\mathtt{M}} M : T$ and returns a typed process

$$\vdash^* [\![M]\!]z :: (\!|\Gamma|\!), z : [\![T]\!].$$

where $\vdash^*$ indicates typability in APCP ignoring priorities and priority checks.
Translation of types:

$$(\!|T|\!) \triangleq \bullet \otimes \overline{[\![T]\!]} \quad (\text{if } T \neq \square)$$

$$[\![T \times U]\!] \triangleq \overline{(\!|T|\!)} \otimes \overline{(\!|U|\!)} \qquad [\![T \multimap U]\!] \triangleq (\!|T|\!) ⅋ [\![U]\!] \qquad [\![\mathbf{1}]\!] \triangleq \bullet$$

$$[\![!T.S]\!] \triangleq \bullet \otimes (\!|T|\!) ⅋ \overline{(\!|S|\!)} \qquad [\![\oplus\{i : S_i\}_{i \in I}]\!] \triangleq \bullet \otimes \&\{i : \overline{(\!|S_i|\!)}\}_{i \in I} \qquad [\![\mathsf{end}]\!] \triangleq \bullet \otimes \bullet$$

$$[\![?T.S]\!] \triangleq \overline{(\!|T|\!)} \otimes \overline{(\!|S|\!)} \qquad [\![\&\{i : S_i\}_{i \in I}]\!] \triangleq \oplus\{i : \overline{(\!|S_i|\!)}\}_{i \in I} \qquad [\![\square]\!] \triangleq (\!|\square|\!) \triangleq \bullet$$

Intuitively, session types such as '$\bullet \otimes \ldots$' codify the enabling of an interaction (with an explicit substitution or with a buffer). A kind of "announcement" for interacting parties.

Translating Terms (Selection)

Below, we write '_' to denote a fresh name of type $\bullet$; when sending names denoted '_', we omit binders '$(\boldsymbol{\nu}\_\_)$'.

[TYP-VAR]	$[\![x]\!]z \triangleq x[\_, z]$	
[TYP-ABS]	$[\![\lambda x.M]\!]z \triangleq z(x, a); [\![M]\!]a$	receive x, then run body
[TYP-APP]	$[\![M\ N]\!]z \triangleq (\boldsymbol{\nu}ab)(\boldsymbol{\nu}cd)([\![M]\!]a$	run abstraction
	$\mid b[c, z]$	trigger function body
	$\mid d(\_, e); [\![N]\!]e)$	parameter as future substitution
[TYP-SUB]	$[\![M\{\!\{N/x\}\!\}]\!]z \triangleq (\boldsymbol{\nu}xa)([\![M]\!]z$	run body
	$a(\_, b); [\![N]\!]b)$	block until body is variable

Translating $\texttt{LAST}^n$ into $\texttt{APCP}$ — Translating Terms (Selection)

[TYP-NEW]	$[\![\texttt{new}]\!]z \triangleq (\boldsymbol{\nu} ab)(a[\_, z]$	activate buffer
	$\mid b(\_, c); (\boldsymbol{\nu} dx)(\boldsymbol{\nu} ey)($	block until activated
	$[\![[\varepsilon\rangle]\!]d\rangle e$	prepare buffer
	$\mid [\![(x, y)]\!]c))$	return pair of endpoints
[TYP-SEND]	$[\![\texttt{send}\, M\, N]\!]z \triangleq (\boldsymbol{\nu} ab)(\boldsymbol{\nu} cd)(a(\_, e); [\![M]\!]e$	block payload until received
	$\mid [\![N]\!]c$	run channel term to activate buffer
	$\mid d(\_, f); (\boldsymbol{\nu} gh)($	wait for buffer to activate
	$f[b, g]$	send to buffer
	$\mid h[\_, z]))$	prepare returned endpoint variable
[TYP-RECV]	$[\![\texttt{recv}\, M]\!]z \triangleq (\boldsymbol{\nu} ab)([\![M]\!]a$	run channel term to activate buffer
	$\mid b(c, d);$	receive from buffer
	$(\boldsymbol{\nu} ef)(z[c, e] \mid f(\_, g); d[\_, g]))$	returned pair

- Well-typed $\mathtt{APCP}$ processes that are typable under empty contexts ($\vdash P :: \emptyset$) are deadlock-free.
- We transfer this result to $\mathtt{LAST}^n$ configurations by appealing to the **operational correctness** of our translation (completeness and soundness properties).

Each deadlock-free configuration in $\mathtt{LAST}^n$ thus obtained satisfies two requirements:

- The configuration is typable $\emptyset \vdash^{\blacklozenge}_{\mathtt{C}} C : \mathbf{1}$, i.e., it needs no external resources and has no external behavior.
- The typed translation of the configuration satisfies priority requirements in $\mathtt{APCP}$.

Theorem (Deadlock-freedom for $\mathtt{LAST}^n$)

Given $\emptyset \vdash^{\blacklozenge}_{\mathtt{C}} C : \mathbf{1}$, *if* $\vdash [\![C]\!]z :: \Gamma$ *for some* Γ, *then* $C \equiv \blacklozenge\,()$ *or* $C \longrightarrow_{\mathtt{C}} D$ *for some* D.

Conclusion

Summary:

- Two different formal systems ($\texttt{LAST}^n$ and $\texttt{APCP}$) that express asynchronous message-passing concurrency
- They are defined at different levels of abstraction, and are connected via a correct translation
- The design of $\texttt{LAST}^n$ builds upon the best features of $\texttt{APCP}$
- Transference of deadlock-freedom allows us to exploit already developed machinery (for $\texttt{APCP}$) and also keep the formulation of $\texttt{LAST}^n$ within "familiar territory"

Future work:

- Recursive types in $\texttt{LAST}^n$
- Behavioral theory for $\texttt{LAST}^n$ (by leveraging $\texttt{APCP}$)

Asynchronous Session-Based Concurrency: Deadlock-freedom in Cyclic Process Networks

Bas van den Heuvel and **Jorge A. Pérez**

University of Groningen, The Netherlands

March 25, 2024

UNIFYING C•RRECTNESS FOR C•MMUNICATING S•FTWARE