A New Linear Logic for Deadlock-Free Session-Typed Processes

Ornela Dardha
(joint work with Simon Gay)

University of Leicester
23 November 2018
The deep correspondence between **types** and **logic**:

*foundation of functional programming.*

\[
\text{types} \approx \text{propositions} \\
\text{programs} \approx \text{proofs} \\
\text{evaluation} \approx \text{proof normalisation} \\
\quad \text{(cut elimination)}
\]

**Example:**
a function of type \(A \rightarrow B\) corresponds to a proof of \(A\) **implies** \(B\); computationally, it constructs a proof of \(B\) (the result) from a proof of \(A\) (the parameter).
In 1987, Girard speculated that linear logic could form the basis of a Curry-Howard correspondence for concurrent computation.

Connections between linear logic and the pi-calculus were developed [Abramsky 1990, 1994; Bellin & Scott 1994], but did not become foundation of concurrent programming.
Session types were introduced by Honda et al. [1993, 1994, 1998] as type-theoretic specifications of communication protocols.

?A . B receive a message of type A, then continue protocol B.
! A . B send a message of type A, then continue protocol B.

Duality: A and A⊥ are complementary views of a protocol.

During the subsequent 20+ years, session types developed into a large and active research area.

Kohei Honda
Caires and Pfenning [2010] discovered a correspondence between session types for pi-calculus and dual intuitionistic linear logic.

The logical approach to session types has been extended: dependent types, failures, sharing and races,…

Proof normalisation (cut elimination) corresponds to communication.

Further work by Caires et al. [2010 -]; Wadler [2012 -] …
session types $\simeq$ propositions

pi-calculus processes $\simeq$ proofs

communication $\simeq$ proof normalisation
  (cut elimination)

- $?A . B$ corresponds to $A \& B$
- $!A . B$ corresponds to $A \otimes B$
- Branch $\&\{l_i : A_i\}_{i \in I}$ and
- Select $\oplus\{l_i : A_i\}_{i \in I}$ are the same...
Input corresponds to \textit{par}

\[
R \vdash \emptyset, y : A, x : B \\
\frac{}{x(y).R \vdash \emptyset, x : A \otimes B}
\]

Output corresponds to \textit{tensor}

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

Notice the threading of the continuation channel through the rules.
Cut Elimination as Communication

Session Types and Classical Linear Logic

Wadler 2012; Caires 2014 (@Luca Cardelli Fest)

\( P \xrightarrow{A \otimes B} P \)
\[ P, Q ::= \begin{array}{ll}
  x[y].P & \text{(output)} \\
  x(y).P & \text{(input)} \\
  x \lhd l_j.P & \text{(selection)} \\
  x \triangleright \{l_i : P_i\}_{i \in I} & \text{(branching)} \\
  0 & \text{(inaction)} \\
  P | Q & \text{(composition)} \\
  (\nu x^A y)P & \text{(session restriction)} \\
\end{array} \]
Deadlock arises from **cyclic** dependency between communication operations when two processes share at least two channels.

\[
(\nu x_1 y_1)(\nu x_2 y_2)[x_1(z).x_2(w).0 \mid y_1[42].y_2[\text{true}].0] \quad \text{OK}
\]

\[
(\nu x_1 y_1)(\nu x_2 y_2)[x_1(z).x_2(w).0 \mid y_2[\text{true}].y_1[42].0] \quad \text{STUCK}
\]
Deadlock arises from **cyclic** dependency between communication operations when two processes share at least two channels.

\[(\nu x_1 y_1)(\nu x_2 y_2)[x_1(z).x_2(w).0 | y_1[42].y_2[\text{true}].0] \quad \text{OK}\]

\[(\nu x_1 y_1)(\nu x_2 y_2)[x_1(z).x_2(w).0 | y_2[\text{true}].y_1[42].0] \quad \text{STUCK}\]

The linear logic type system guarantees deadlock-freedom because *two processes can only be connected by a single channel*.

\[
\begin{align*}
\Gamma & \quad x & \quad \Delta \\
\hline 
\hline 
\text{(cut)} \\
\hline 
\text{cut:} & \\
\hline 
P \vdash \Gamma, x:A & Q \vdash \Delta, y:A^{\perp} & (\nu x^A y)(P \mid Q) \vdash \Gamma, \Delta
\end{align*}
\]
Deadlock arises from **cyclic** dependency between communication operations when two processes share at least two channels.

\[
(\nu x_1 y_1)(\nu x_2 y_2)[x_1(z).x_2(w).0 \mid y_1[42].y_2[\text{true}].0]
\]

**OK**

\[
(\nu x_1 y_1)(\nu x_2 y_2)[x_1(z).x_2(w).0 \mid y_2[\text{true}].y_1[42].0]
\]

**STUCK**

The linear logic type system **rejects** both the processes above because they are connected by two channels.

Priorities \( o, o' \ldots \) are natural numbers and annotate types.

Priorities must obey the following laws:
(i) an action (input/output) of priority \( o \) must be prefixed only by actions of priorities strictly smaller than \( o \).
(ii) communication requires equal priorities of dual actions.

Priority-based type systems type more processes than linear logic, as they allow processes to share more than a single channel.
Exercise: are the following processes typable?

\[
(\nu x_1 y_1)(\nu x_2 y_2)[x_1(z).x_2(w).0 | y_1[42].y_2[\text{true}].0]
\]

(i) \(\text{pr}(x_1) < \text{pr}(x_2)\)  \(\text{pr}(y_1) < \text{pr}(y_2)\)
(ii) \(\text{pr}(x_1) = \text{pr}(y_1)\)  \(\text{pr}(x_2) = \text{pr}(y_2)\)

\[
(\nu x_1 y_1)(\nu x_2 y_2)[x_1(z).x_2(w).0 | y_2[\text{true}].y_1[42].0]
\]

(i) \(\text{pr}(x_1) < \text{pr}(x_2)\)  \(\text{pr}(y_2) < \text{pr}(y_1)\)
(ii) \(\text{pr}(x_1) = \text{pr}(y_1)\)  \(\text{pr}(x_2) = \text{pr}(y_2)\)
We combine classical linear logic with priorities for a more expressive session type system for deadlock-free processes.

Replace the cut rule with mix and cycle.

\[
\begin{array}{c}
\text{(mix)} \\
P \vdash \Gamma \quad Q \vdash \Delta \\
\hline
P \mid Q \vdash \Gamma, \Delta
\end{array}
\quad \begin{array}{c}
\text{(cycle)} \\
P \vdash \Gamma, x : A, y : A^\perp \\
(\nu x^A y)P \vdash \Gamma
\end{array}
\]

With these rules, multicut is derivable.

\[
\begin{array}{c}
\text{(multicut)} \\
P \vdash \Gamma, x_1 : A_1, \ldots, x_n : A_n \quad Q \vdash \Delta, y_1 : A_1^\perp, \ldots, y_n : A_n^\perp \\
\hline
(\nu x_1^{A_1} y_1 \ldots x_n^{A_n} y_n)(P \mid Q) \vdash \Gamma, \Delta
\end{array}
\]

Our Goal: Priority-based Linear Logic Typing
Propositions are annotated with priorities.

\[ A, B ::= \bot^\circ | 1^\circ | A \otimes^\circ B | A \otimes^\circ B \mid \oplus^\circ \{l_i : A_i\}_{i \in I} \mid \&^\circ \{l_i : A_i\}_{i \in I} \mid ?^\circ A \mid !^\circ A \]

Proofs must obey laws (i) and (ii) on priorities.
\( \circ < \text{pr}(\Gamma) \) satisfies law (i) on strictly smaller.

\[
\begin{align*}
P \vdash \Gamma, y : A, x : B & \quad \circ < \text{pr}(\Gamma) \\
x(y).P \vdash \Gamma, x : A \otimes^\circ B \quad \text{(\otimes)} \\
P \vdash ?\Gamma, y : A & \quad \circ < \text{pr}(\Gamma) \\
!x(y).P \vdash ?\Gamma, x : !^\circ A \quad \text{(!)}
\end{align*}
\]
When forming a cycle, the priorities of the connected types must be equal, to satisfy (ii):

\[
\frac{P \vdash \Gamma, \, x : A, \, y : A^\perp}{(\nu x^A y) P \vdash \Gamma}
\]

Meaning: eventually \(x\) and \(y\) will be ready to communicate at the same time/step, allowing a reduction step and a proof rewrite.

Equality of priorities is captured by the duality definition:

\[
\begin{align*}
(A \otimes^\circ B)^\perp &= A^\perp \otimes^\circ B^\perp \\
(A \otimes^\circ B)^\perp &= A^\perp \otimes^\circ B^\perp \\
(\&^\circ \{ l_i : A_i \} \}_{i \in I})^\perp &= \oplus^\circ \{ l_i : A_i^\perp \} \}_{i \in I} \\
(\oplus^\circ \{ l_i : A_i \} \}_{i \in I})^\perp &= \&^\circ \{ l_i : A_i^\perp \} \}_{i \in I}
\end{align*}
\]

\[
\begin{align*}
(\perp^\circ)^\perp &= 1^\circ \\
(1^\circ)^\perp &= \perp^\circ \\
?^\circ A^\perp &= !^\circ A^\perp \\
!^\circ A^\perp &= ?^\circ A^\perp
\end{align*}
\]
Beta-reduction for tensor/output and par/input: derivation

\[
\begin{array}{c}
o < \text{pr}(\Gamma) \\
P \vdash \Gamma, v : A, x : B \\
x[v].P \vdash \Gamma, x : A \otimes B \\
\hline
(P) \\
x[v].P \mid y(w).Q \vdash \Gamma, \Delta, x : A \otimes B, y : A^\perp \otimes B^\perp \\
(\nu x^A \otimes y^B y)(x[v]).P \mid y(w).Q \vdash \Gamma, \Delta \\
\hline
(\otimes) \\
\vdash \quad \text{(mix)} \\
\text{(cycle)} \\
P \vdash \Gamma, v : A, x : B \\
Q \vdash \Delta, w : A^\perp, y : B^\perp \\
\hline
(\otimes) \\
\vdash \quad \text{(mix)} \\
\text{(cycle)} \\
P \mid Q \vdash \Gamma, \Delta, v : A, x : B, w : A^\perp, y : B^\perp \\
(\nu x^B y)(P \mid Q) \vdash \Gamma, \Delta, v : A, w : A^\perp \\
(\nu v^A w)(\nu x^B y)(P \mid Q) \vdash \Gamma, \Delta
\end{array}
\]

Beta-reduction for other connectives: summary

\[(\nu y^A z)(x \rightarrow y^A \mid P) \vdash \Gamma, x : A^\perp \quad \Rightarrow \quad P[x/z] \vdash \Gamma, x : A^\perp\]

\[(\nu x^A y)(x[\cdot].0 \mid y().P) \vdash \Gamma \quad \Rightarrow \quad P \vdash \Gamma\]

\[(\nu x^{\otimes \{l_i : B_i\}_{i \in I}} y)(x \triangleleft l_j.P \mid y \triangleright \{l_i : Q_i\}_{i \in I}) \vdash \Gamma, \Delta \quad \Rightarrow \quad (\nu x^{B_j} y)(P \mid Q_j) \vdash \Gamma, \Delta\]

\[(\nu x^{\bowtie A} y)(!x(v).P \mid ?y[w].Q) \vdash \Gamma, \Delta \quad \Rightarrow \quad (\nu v^A w)(P \mid Q) \vdash \Gamma, \Delta\]
Theorem (cycle elimination). Given a proof of a sequent, we can construct a cycle-free proof for it.

Proof: (following cut elimination proof) a cycle is eliminated by either:

i) replacing it with another cycle on smaller propositions;
ii) pushing it further up the proof tree.

Since we are allowing cyclic structures, how do we make sure we capture only the good ones?
A concurrent system is a collection of parallel processes, each with a top-level input or output action (prefix).

Pick a top-level prefix with \textit{smallest} priority $o$: $x(z)$, say.

Somewhere there is the co-action $y[42]$ with \textit{equal} priority $o$.

$y[42]$ must be in a \textbf{different parallel component}, otherwise it would be guarded by $x(z)$, requiring $o < o$.

$y[42]$ must be a \textbf{top-level prefix}, otherwise it is guarded by a prefix with priority $o' < o$, contradicting the dominance of $o$.

Communication on endpoints $x$ and $y$ is possible immediately.
Cycle elimination corresponds to communication.

**Theorem (subject reduction).** Well-typed processes reduce to well-typed processes.

**Proof:** beta-reductions and commuting conversions.

**Theorem (top-level deadlock freedom).** If process $P$ is well typed and it is a cycle, then there is some $Q$, such that $P$ reduces to $Q$ and $Q$ is not a cycle.

**Proof:** follows from cycle elimination.
Conclusion and Future Work

- Presented a new priority-based linear logic combining mix and cycle rules with Kobayashi’s priorities.
- Used it as a basis for a Curry-Howard isomorphism with session typed pi-calculus, allowing “good” cyclic processes.
- We prove the cycle elimination theorem, obtaining as a result deadlock freedom for session typed processes.

Future work:

i) develop a type system for a functional language, GV with cycle and translate it to our system.

ii) extend our priority-based logic to allow recursion and sharing.
Thank you!

Questions?