Our sponsors make OPLSS possible.

Diamond Sponsor

https://www.janestreet.com/

Platinum Sponsors

https://www.nsf.gov/

https://www.sigplan.org/

Silver Sponsors

Session Types & Mailbox Types
Lecture 3: Session Types, Linear Logic and the Curry-Howard Correspondence

Simon Gay
School of Computing Science
University of Glasgow, UK

Stigler's law of eponymy

21 languages

Article Talk Read Edit View history

From Wikipedia, the free encyclopedia

Stigler's law of eponymy, proposed by University of Chicago statistics professor Stephen Stigler in 1980,[1] states that no scientific discovery is named after its original discoverer. Examples include Hubble's law, which was derived by Georges Lemaître two years before Edwin Hubble; the Pythagorean theorem, which was known to Babylonian and Indian mathematicians before Pythagoras; and Halley's Comet, which was observed by astronomers since at least 240 BC (although its official designation is due to the first ever mathematical prediction of such astronomical phenomenon in the sky, not to its discovery).

Stigler attributed the discovery of Stigler's law to sociologist Robert K. Merton, making "Stigler's law" an example of Stigler's law.

Curry-Howard / propositions as types

The foundation of typed functional programming is the *Curry-Howard isomorphism* or *propositions-as-types correspondence*:

propositions	$\leftrightarrow$	types
proofs	$\leftrightarrow$	programs
proof simplification	$\leftrightarrow$	computation

There's a clear presentation in an article by Phil Wadler:

> *Philip Wadler. Propositions as types. CACM, 58(12):75–84, 2015. https://doi.org/10.1145/2699407.*

Wadler on Propositions as Types

Powerful insights arise from linking two fields of study previously thought separate.

Examples include Descartes's coordinates, which links geometry to algebra, Quantum Mechanics, which links particles to waves, and Shannon's Information Theory, which links thermodynamics to communication.

Such a synthesis is offered by the principle of Propositions as Types, which links logic to computation. At first sight it appears to be a simple coincidence — almost a pun — but it turns out to be remarkably robust, inspiring the design of automated proof assistants and programming languages, and continuing to influence the forefronts of computing.

Curry-Howard / propositions as types

These rules are the typing rules for simply-typed λ-calculus and (reading only the blue parts) the axiom and inference rules for first-order intuitionistic propositional logic.

$$\Gamma, x : A \vdash x : A \qquad \frac{\Gamma, x : A \vdash N : B}{\Gamma \vdash \lambda x.N : A \to B} \qquad \frac{\Gamma \vdash L : A \to B \quad \Delta \vdash M : A}{\Gamma, \Delta \vdash LM : B}$$

The correspondence between proof simplification and computation is given by

$$\frac{\dfrac{\Gamma, x : A \vdash N : B}{\Gamma \vdash \lambda x.N : A \to B} \quad \Delta \vdash M : A}{\Gamma, \Delta \vdash (\lambda x.N)M : B} \rightsquigarrow \frac{\Gamma, x : A \vdash N : B \quad \Delta \vdash M : A}{\Gamma, \Delta \vdash N[M/x] : B}$$

Curry-Howard / propositions as types

There is also a connection with Cartesian closed categories, which form abstract models for intuitionistic logic and abstract denotational semantics for simply-typed λ-calculus.

Including the category-theoretic angle gives the Curry-Howard-Lambek correspondence.

Linear logic: towards Curry-Howard for concurrency?

It was natural to look for a Curry-Howard correspondence for concurrent programming, but not obvious how to construct it.

In the late 1980s Girard introduced linear logic and suggested that classical linear logic should be relevant to concurrency, but without giving a concrete proposal.

> *Linear logic is the first attempt to solve the problem of parallelism at the logical level, i.e. by making the success of the communication process only dependent on the fact that the programs can be viewed as proofs of something and are therefore sound.*

The $\parr$ connective of classical linear logic was suggestively named "par" for "parallel".

The cut rule as communication

When I started my PhD in 1991, with Samson Abramsky, he had recently produced this work (technical report in 1990):

> *Samson Abramsky. Computational interpretations of linear logic. TCS 111(1&2):3–57, 1993. https://doi.org/10.1016/0304-3975(93)90181-R*

He interpreted the *cut* rule as parallel composition of communicating concurrent processes:

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

and built a process calculus around the proof rules of classical linear logic.

Proofs as processes

Abramsky then proposed an assignment of pi calculus processes to classical linear logic proofs (again there was an earlier technical report):

> *Samson Abramsky. Proofs as processes. TCS 135(1):5–9, 1994. https://doi.org/10.1016/0304-3975(94)00103-0*

which was further developed by Bellin and Scott:

> *Gianluigi Bellin and Philip J. Scott. On the pi-calculus and linear logic. TCS 135(1):11–65, 1994. https://doi.org/10.1016/0304-3975(94)00104-9*

Proofs as processes

The rules for tensor and par are dual and correspond to complementary participants in a communication action:

$$\frac{P \vdash \Gamma, y : A \quad Q \vdash \Delta, z : B}{(\nu y)(\nu z)x\langle y, z\rangle.(P \mid Q) \vdash \Gamma, \Delta, x : A \otimes B} \qquad \frac{R \vdash \Gamma, y : A, z : B}{x(y, z).R \vdash \Gamma, x : A \parr B}$$

Communication is proof simplification:

$$\frac{\dfrac{P \vdash \Gamma, y : A \quad Q \vdash \Delta, z : B}{(\nu y)(\nu z)x\langle y, z\rangle.(P \mid Q) \vdash \Gamma, \Delta, x : A \otimes B} \quad \dfrac{R \vdash \Theta, y : A^{\perp}, z : B^{\perp}}{x(y, z).R \vdash \Theta, x : A^{\perp} \parr B^{\perp}}}{(\nu x)((\nu y)(\nu z)x\langle y, z\rangle.(P \mid Q) \mid x(y, z).R) \vdash \Gamma, \Delta, \Theta}$$

$$\rightsquigarrow$$

$$\frac{\dfrac{P \vdash \Gamma, y : A \quad R \vdash \Theta, y : A^{\perp}, z : B^{\perp}}{(\nu y)(P \mid R) \vdash \Gamma, \Theta, z : B^{\perp}} \quad Q \vdash \Delta, z : B}{(\nu z)((\nu y)(P \mid R) \mid Q) \vdash \Gamma, \Delta, \Theta}$$

Still no accepted Curry-Howard correspondence

The ideas of Abramsky/Bellin/Scott did not gain acceptance as a Curry-Howard correspondence for concurrency.

Possible reasons:

- Abramsky's first attempt proposed a new process calculus, in contrast to the original Curry-Howard correspondence which linked two pre-existing theories.
- Regarding proofs as processes, pi calculus itself was still fairly new and not fully established.

Interaction categories

Next, Abramsky proposed the theory of "interaction categories" as a semantic framework for concurrency.

It was based on $*$-autonomous categories, which are abstract models of classical linear logic, but it lacked the syntactic aspect.

Also, the semantic setting was limited in comparison with pi calculus and its support for mobility.

However, structures based on $*$-autonomous categories came back later as a foundation for quantum computing:

> *Samson Abramsky and Bob Coecke. A categorical semantics of quantum protocols. LICS 2004.*

Session types

At around the same time, Kohei Honda and his collaborators were developing session types:

> *Kohei Honda. Types for Dyadic Interaction. CONCUR 1993.*

> *Kaku Takeuchi, Kohei Honda, Makoto Kubo. An Interaction-Based Language and Its Typing System. PARLE 1994.*

Later it became clear that interaction categories contained semantic types corresponding to session types, but we didn't make the connection at the time.

Session types and linear logic

In 2010, Luís Caires and Frank Pfenning published a propositions-as-types correspondence between a form of pi calculus with session types and dual intuitionistic linear logic:

> *Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. CONCUR 2010. https://doi.org/10.1007/978-3-642-15375-4_16*

They developed the idea further in a series of papers with Bernardo Toninho and others.

This line of work has been recognised as a Curry-Howard correspondence for concurrency. Like the original correspondence, it has supported all sorts of extensions and variations.

Session types and linear logic

The easiest way to explain the key idea is to follow the work of Phil Wadler, who reformulated Caires and Pfenning's idea to use classical linear logic instead of dual intuitionistic linear logic.

> *Philip Wadler. Propositions as sessions. ICFP 2012. https://doi.org/10.1145/2364527.2364568*

Abramsky-Bellin-Scott vs. Caires-Pfenning-Wadler

Abramsky-Bellin-Scott:

$$\frac{P \vdash \Gamma, y : A \quad Q \vdash \Delta, z : B}{(\nu yz)x\langle y, z\rangle.(P \mid Q) \vdash \Gamma, \Delta, x : A \otimes B} \qquad \frac{R \vdash \Gamma, y : A, z : B}{x(y, z).R \vdash \Gamma, x : A \parr B}$$

Wadler:

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

Caires-Pfenning:

$$\frac{\Gamma; \Delta \vdash P :: y : A \quad \Gamma; \Delta', x : B \vdash Q :: z : C}{\Gamma; \Delta, \Delta', x : A \multimap B \vdash (\nu y)x\langle y\rangle.(P \mid Q) :: z : C}$$

$$\frac{\Gamma; \Delta, y : A \vdash R :: x : B}{\Gamma; \Delta \vdash x(y).R :: x : A \multimap B}$$

And a little bit of Gay-Vasconcelos...

Having seen the key idea in Caires-Pfenning, it looks similar to the idea for typing send/receive that we saw in Lecture 2:

send $v\ c \to c$

send: $T \to_\omega\ !T.S \to_1 S$

receive $c \to (v, c)$

receive: $?T.S \to_\omega\ T \times_1 S$

but we weren't thinking about Curry-Howard at that time.

> *Simon Gay and Vasco Vasconcelos. Linear type theory for asynchronous session types. JFP 20(1):19–50, 2010. https://doi.org/10.1017/S0956796809990268*

The rest of this lecture

I will go into more detail about the Curry-Howard correspondence for session types, using Wadler's presentation. His language is called CP.

Although there is more literature in the Caires-Pfenning style, Wadler's version is closer to the pi calculus from Lecture 1.

I won't cover this, but eventually the $*$-autonomous categories came back in work by Bob Atkey:

> *Robert Atkey. Observed communication semantics for classical processes. ESOP 2017. https://doi.org/10.1007/978-3-662-54434-1_3*

Adjusting the pi calculus for CP

The *link* process $x \leftrightarrow y$ can interact with another process on either x or y, resulting in a substitution $[y/x]$ or $[x/y]$ respectively.

Close and *wait* are written $x[]$ and $x().P$, but in contrast to Lecture 1, only *wait* has a continuation.

Receive is written $x(y).P$ instead of $x?y.P$.

Send is written $x[y].(P \mid Q)$ instead of $x!y.P$. The message y is bound in P and does not occur in Q, while the channel x is used by the continuation process Q and does not occur in P.

There is no null process (0).

Adjusting the pi calculus for CP

Internal and external choice are binary with fixed labels inl and inr, so $x \lhd \mathsf{inl}.P$ and $x \lhd \mathsf{inr}.P$ have the usual syntax and $x \rhd \{P, Q\}$ corresponds to $x \rhd \{\mathsf{inl}\colon P, \mathsf{inr}\colon Q\}$.

There is also nullary external choice, $x \rhd \{\}$.

Parallel composition is combined with scope restriction into $(\nu x)(P \mid Q)$ (note: single binder).

Parallel composition $P \mid Q$ and scope restriction $(\nu xy)P$ from Lecture 1 are not available as separate process constructors.

Summary of the syntax changes

	Lecture 3	Lecture 1
inaction	—	0
link	$x \leftrightarrow y$	—
wait	$x().P$	$x?.P$
close	$x[]$	$x!.P$
nullary external choice	$x \triangleright \{\}$	—
receive	$x(y).P$	$x?y.P$
send	$x[y].(P \mid Q)$	$x!y.P$
external choice	$x \triangleright \{P, Q\}$	$x \triangleright \{l\colon P_l\}^{l \in L}$
internal choice	$x \triangleleft \mathsf{inr}.P$, $x \triangleleft \mathsf{inl}.P$	$x \triangleleft l.P$
parallel composition	—	$P \mid P$
scope restriction	—	$(\nu xy)P$
cut	$(\nu x)(P \mid Q)$	—

Types for CP

The types use the syntax of classical linear logic, not session types.

A ::=	$\bot$	wait
	1	close
	$\top$	nullary external choice
	0	nullary internal choice
	$A \parr A$	input
	$A \otimes A$	output
	$A \,\&\, A$	external choice
	$A \oplus A$	internal choice
	$!A$	server accept
	$?A$	client request

Duality

$$\bot^\bot = 1 \qquad 1^\bot = \bot$$
$$0^\bot = \top \qquad \top^\bot = 0$$

$$(A \parr B)^\bot = A^\bot \otimes B^\bot \qquad (A \otimes B)^\bot = A^\bot \parr B^\bot$$
$$(A \mathbin{\&} B)^\bot = A^\bot \oplus B^\bot \qquad (A \oplus B)^\bot = A^\bot \mathbin{\&} B^\bot$$

$$(!A)^\bot = ?(A^\bot) \qquad (?A)^\bot = !(A^\bot)$$

CP typing rules (1)

$$x \leftrightarrow y \vdash x : A, y : A^{\perp} \qquad \text{(T-Ax)}$$

$$x[] \vdash x : 1 \qquad \text{(T-1)}$$

$$\frac{P \vdash \Gamma}{x().P \vdash \Gamma, x : \bot} \qquad \text{(T-}\bot\text{)}$$

$$x \rhd \{\} \vdash \Gamma, x : \top \qquad \text{(T-}\top\text{)}$$

(no rule for 0)

CP typing rules (2)

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

$$\frac{P \vdash \Gamma, y : A, x : B}{x(y).P \vdash \Gamma, x : A \parr B} \quad (\text{T-}\parr)$$

$$\frac{P \vdash \Gamma, x : A \qquad Q \vdash \Delta, x : A^{\perp}}{(\nu x)(P \mid Q) \vdash \Gamma, \Delta} \quad (\text{T-Cut})$$

CP typing rules (3)

$$\frac{P \vdash \Gamma, x : A}{x \lhd \mathsf{inl}.P \vdash \Gamma, x : A \oplus B} \quad (\text{T-}\oplus\text{L})$$

$$\frac{P \vdash \Gamma, x : B}{x \lhd \mathsf{inr}.P \vdash \Gamma, x : A \oplus B} \quad (\text{T-}\oplus\text{R})$$

$$\frac{P \vdash \Gamma, x : A \qquad Q \vdash \Gamma, x : B}{x \rhd \{P, Q\} \vdash \Gamma, x : A \,\&\, B} \quad (\text{T-}\&)$$

The axiom link

The new process $x \leftrightarrow y$, together with T-Cut, can extend any part of a process's typing context to a new variable.

$$\dfrac{P \vdash \Gamma, x : A \qquad \dfrac{}{x \leftrightarrow y \vdash x : A^{\perp}, y : A}\ \text{T-Ax}}{(\nu x)(P \mid x \leftrightarrow y) \vdash \Gamma, y : A}\ \text{T-Cut}$$

This configuration reduces to a substitution:

$$P[y/x] \vdash \Gamma, y : A$$

CP structural congruence

$$x \leftrightarrow y \equiv y \leftrightarrow x \qquad \text{(C-Link)}$$

$$(\nu x)(P \mid Q) \equiv (\nu x)(Q \mid P) \qquad \text{(C-CommCut)}$$

$$(\nu x)((\nu y)(P \mid Q) \mid M) \equiv (\nu y)(P \mid (\nu x)(Q \mid M)) \qquad \text{(C-AssocCut)}$$

CP reduction rules (1)

$$\frac{P \;\to\; P'}{(\nu x)(P \mid Q) \;\to\; (\nu x)(P' \mid Q)} \qquad \text{(R-Scope)}$$

$$\frac{P \equiv P' \qquad P' \to Q' \qquad Q' \equiv Q}{P \to Q} \qquad \text{(R-Struct)}$$

$$(\nu x)(y \leftrightarrow x \mid P) \;\to\; P[y/x] \qquad \text{(R-AxCut)}$$

CP reduction rules (2)

$$(\nu x)(x[] \mid x().P) \rightarrow P \qquad \text{(R-Close)}$$

$$(\nu x)(x[y].(P \mid Q) \mid x(y).M) \rightarrow (\nu y)(P \mid (\nu x)(Q \mid M)) \qquad \text{(R-Com)}$$

$$(\nu x)(x \lhd \mathsf{inl}.P \mid x \rhd \{Q, M\}) \rightarrow (\nu x)(P \mid Q) \qquad \text{(R-ChoiceL)}$$

$$(\nu x)(x \lhd \mathsf{inr}.P \mid x \rhd \{Q, M\}) \rightarrow (\nu x)(P \mid M) \qquad \text{(R-ChoiceR)}$$

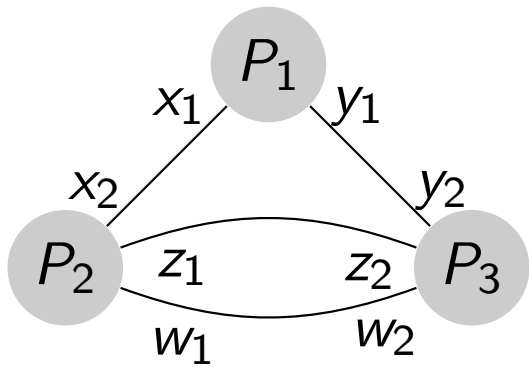
This diagram illustrates R-Com:

Deadlock-freedom and (lack of) cycles

In Lecture 2 we could connect processes in arbitrary ways, limited only by the requirement for dual types when making connections.

For example:

$$(\nu x_1 x_2)(\nu y_1 y_2)(P_1 \mid (\nu z_1 z_2)(\nu w_1 w_2)(P_2 \mid P_3))$$



Cyclic configurations can introduce deadlocks, if there are also cyclic dependencies between communications on different channels.

Deadlock-freedom and (lack of) cycles

In CP, cyclic structures cannot be formed, because T-Cut only allows a connection between two separate processes. This is a very conservative approach to guaranteeing deadlock-freeedom.

Adding this rule:

$$\frac{P \vdash \Gamma, x : A, y : A^{\perp}}{(\nu xy)P \vdash \Gamma} \qquad \text{(T-Cycle)}$$

increases expressivity, breaks the logical connection, and allows deadlocks.

Other techniques can be added to eliminate deadlocks again:

> *Naoki Kobayashi. A partially deadlock-free typed process calculus. TOPLAS 20(2):436–482, 1998.*
>
> *Ornela Dardha and Simon Gay. A new linear logic for deadlock-free session-typed processes. FOSSACS 2018.*

Example reduction sequence

$$
\begin{aligned}
(\underline{\nu x})(\underline{x[y]}.(y[] \mid x[]) \mid \underline{x(z)}.x().z().w[]) = & \quad \text{(Variable convention)} \\
(\underline{\nu x})(\underline{x[y]}.(y[] \mid x[]) \mid \underline{x(y)}.x().y().w[]) \rightarrow & \quad \text{(R-Com)} \\
(\nu y)(y[] \mid (\underline{\nu x})(\underline{x[]} \mid \underline{x()}.y().w[])) \rightarrow & \\
& \quad \text{(R-Close inside R-Scope)} \\
(\underline{\nu y})(\underline{y[]} \mid \underline{y()}.w[]) \rightarrow & \quad \text{(R-Close)} \\
w[] & \quad (\checkmark)
\end{aligned}
$$

The limitation of not allowing sharing

Suppose we represent Boolean values by defining

$$\mathsf{bool} = 1 \oplus 1$$

with left for true and right for false. The type

$$\mathsf{bool}^{\perp} \parr (\mathsf{bool}^{\perp} \parr (\mathsf{bool} \otimes 1))$$

is the protocol for a two-input Boolean operator, and we can define

$$\mathtt{and} = y \triangleright \{y \triangleright \{y \triangleleft \mathsf{inl}.y[], y \triangleleft \mathsf{inl}.y[]\}, y \triangleright \{y \triangleleft \mathsf{inl}.y[], y \triangleleft \mathsf{inr}.y[]\}\}$$

with

$$\mathtt{and} \vdash y : \mathsf{bool}^{\perp} \parr \mathsf{bool}^{\perp} \parr \mathsf{bool} \otimes 1.$$

The `and` process can only be used once, and a system that needs to compute several conjunctions would have to repeat the `and` process exactly as many times as necessary.

The exponentials: syntax and types

$$
\begin{array}{lll}
P ::= \ldots & & \\
\quad & !x(y).P & \text{server accept} \\
\quad & ?x[y].P & \text{client request} \\
\quad & ?x[].P & \text{client weaken} \\
\quad & ?x[y,z].P & \text{client contract}
\end{array}
$$

$$
\begin{array}{lll}
A ::= \ldots & & \\
\quad & !A & \text{server accept} \\
\quad & ?A & \text{client request}
\end{array}
$$

(Come back to these later, if there is time.)

Preservation Theorem

Theorem (Preservation): If $P \vdash \Gamma$ and $P \rightarrow Q$ then $Q \vdash \Gamma$.

Proof: By rule induction on $P \rightarrow Q$, in each case analysing the derivation of $P \vdash \Gamma$ and using the components to build a derivation of $Q \vdash \Gamma$.

Progress

Theorem (Progress): If $P \vdash \Gamma$ then either

1. $P \rightarrow Q$, for some process Q, or
2. P offers a prefix, or
3. $P = x_1 \leftrightarrow y_1 \parallel \cdots \parallel x_n \leftrightarrow y_n$ for some x_i and y_i.

Example of case (2):

$$P = (\nu x)(x[] \mid y().x().z[]) \vdash y : \bot$$

does not reduce, but offers a prefix on y.

Session-typed functional programming with Curry-Howard

Wadler also defined a session-typed functional language called GV.

It is similar to the language from Lecture 2, but with a few differences to help it fit into the logical view.

There is a translation of GV into CP, preserving typing.

Lindley and Morris refined GV and defined an operational semantics.

> *Sam Lindley and Garrett Morris. A semantics of propositions as sessions. ESOP 2015.*

The Caires-Pfenning approach

This uses Dual Intuitionistic Linear Logic (Barber & Plotkin 1997).

Typing judgements have the form $\Gamma; \Delta \vdash P :: x : A$

$\parr$ is packaged as $\multimap$ (linear implication) via $A \multimap B = A^{\perp} \parr B$. So $\multimap$ is input (and $\otimes$ is output as before).

Each connective has an introduction rule and an elimination rule.

$$\frac{\Gamma; \Delta, y : A \vdash R :: x : B}{\Gamma; \Delta \vdash x(y).R :: x : A \multimap B}$$

$$\frac{\Gamma; \Delta \vdash P :: y : A \qquad \Gamma; \Delta', x : B \vdash Q :: z : C}{\Gamma; \Delta, \Delta', x : A \multimap B \vdash (\nu y) x\langle y \rangle.(P \mid Q) :: z : C}$$

The Caires-Pfenning approach

Each syntactic construct (e.g. input or output) appears twice, once in an introduction rule and once in an elimination rule.

$$\frac{\Gamma; \Delta, y : A, x : B \vdash R :: z : C}{\Gamma; \Delta, x : A \otimes B \vdash x(y).R :: z : C}$$

$$\frac{\Gamma; \Delta \vdash P :: y : A \qquad \Gamma; \Delta' \vdash Q :: z : B}{\Gamma; \Delta, \Delta' \vdash (\nu y)x\langle y\rangle.(P \mid Q) :: z : A \otimes B}$$

The cut rule matches types on the left and right.

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

Our sponsors make OPLSS possible.

Diamond Sponsor

https://www.janestreet.com/

Platinum Sponsors

https://www.nsf.gov/

https://www.sigplan.org/

Silver Sponsors

The relationship with Classical Linear Logic

The system CP assigns process calculus expressions as proof terms to proofs in CLL.

A key property of CLL is *cut elimination*.

The proof reductions needed for cut elimination form three groups.

1. Principal cut reductions, which correspond to communication steps in pi calculus.
2. Commuting conversions (next slide), which allow prefixes to be moved outwards.
3. Congruence rules such as

$$\frac{P \rightarrow Q}{x(y).P \rightarrow x(y).Q}$$

Commuting conversions

$$(\nu z)(x[y].(P \mid Q) \mid R) \to x[y].((\nu z)(P \mid R) \mid Q) \quad \text{if } z \in \mathtt{fv}(P) \qquad (\kappa_{\otimes 1})$$

$$(\nu z)(x[y].(P \mid Q) \mid R) \to x[y].(P \mid (\nu z)(Q \mid R)) \quad \text{if } z \in \mathtt{fv}(Q) \qquad (\kappa_{\otimes 2})$$

$$(\nu z)(x(y).P \mid Q) \to x(y).(\nu z)(P \mid Q) \qquad (\kappa_{\parr})$$

$$(\nu z)(x \lhd \mathsf{inl}.P \mid Q) \to x \lhd \mathsf{inl}.(\nu z)(P \mid Q) \qquad (\kappa_{\oplus l})$$

$$(\nu z)(x \lhd \mathsf{inr}.P \mid Q) \to x \lhd \mathsf{inr}.(\nu z)(P \mid Q) \qquad (\kappa_{\oplus r})$$

$$(\nu z)(x \rhd \{P, Q\} \mid R) \to x \rhd \{(\nu z)(P \mid R), (\nu z)(Q \mid R)\} \qquad (\kappa_{\&})$$

$$(\nu z)(!x(y).P \mid Q) \to !x(y).(\nu z)(P \mid Q) \quad \text{if } Q = !z(w).Q' \qquad (\kappa_{!})$$

$$(\nu z)(?x[y].P \mid Q) \to ?x[y].(\nu z)(P \mid Q) \qquad (\kappa_{?})$$

$$(\nu z)(?x[].P \mid Q) \to ?x[].(\nu z)(P \mid Q) \qquad (\kappa_{W})$$

$$(\nu z)(?x[u, v].P \mid Q) \to ?x[u, v].(\nu z)(P \mid Q) \qquad (\kappa_{C})$$

$$(\nu z)(x().P \mid Q) \to x().(\nu z)(P \mid Q) \qquad (\kappa_{\bot})$$

Cut elimination and its consequences

Theorem: If $P \vdash \Gamma$ then there exists Q such that $P \rightarrow^* Q$ and Q does not contain cuts.

Lemma: If $P \vdash \Gamma$ and the typing derivation does not contain T-Cut then Γ is non-empty.

Corollary: If $P \vdash \Gamma$ then Γ is non-empty.

Theorem: If $P \vdash \Gamma$ then there exists Q such that $P \rightarrow^* Q$ and Q does not reduce.

Corollary: Removing the commuting conversions and the congruence rules from the semantics gives termination for typed pi calculus processes in CP.

The exponentials: typing rules

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

$$\frac{P \vdash \Gamma, y : A}{?x[y].P \vdash \Gamma, x : ?A} \quad \text{(T-?)}$$

$$\frac{P \vdash \Gamma}{?x[].P \vdash \Gamma, x : ?A} \quad \text{(T-Weak)}$$

$$\frac{P \vdash \Gamma, y : ?A, z : ?A}{?x[y, z].P \vdash \Gamma, x : ?A} \quad \text{(T-Contr)}$$

The exponentials: reduction rules

$$(\nu x)(!x(y).P \mid ?x[y].Q) \rightarrow (\nu y)(P \mid Q) \qquad \text{(R-ServerClient)}$$

$$(\nu x)(!x(y).P \mid ?x[].Q) \rightarrow ?z_1[].\cdots?z_n[].Q \qquad \text{(R-Weaken)}$$

$$(\nu x)(!x(y).P \mid ?x[x', x''].Q) \rightarrow \qquad \text{(R-Contract)}$$
$$?z_1[z_1', z_1''].\cdots?z_n[z_n', z_n''].(\nu x')(!x'(y').P' \mid (\nu x'')(!x''(y'').P'' \mid Q))$$

where

$$\mathtt{fv}(P) = \{y, z_1, \ldots, z_n\}$$

$$P' = P[y'/y][z_1'/z_1]\cdots[z_n'/z_n]$$

$$P'' = P[y''/y][z_1''/z_1]\cdots[z_n''/z_n]$$

A flexible server

Consider a minimal server

$$Q = !x(y).y[]$$

which accepts a request on x, establishing channel y, which it immediately closes. The typing derivation for Q is

$$\dfrac{\dfrac{}{y[] \vdash y : 1}\ \text{T-1}}{Q \vdash x : !1}\ \text{T-!}$$

in which the condition for rule T-!, that any context beyond $y : 1$ must be of the form $?\Gamma$, is satisfied vacuously.

Client one: discards the server

$$M_0 = ?x[].a[]$$

$$\dfrac{\dfrac{}{a[] \vdash a : 1}\ \text{T-1}}{M_0 \vdash a : 1, x : ?\bot}\ \text{T-Weak}$$

Using T-Cut we can derive

$$(\nu x)(Q \mid M_0) \vdash a : 1.$$

By rule R-Weaken we have

$$(\nu x)(Q \mid M_0) \rightarrow a[]$$

in which Q disappears.

Client two: uses the server once

$$M_1 = ?x[y].y().a[]$$

$$\dfrac{\dfrac{}{a[] \vdash a : 1}\ \text{T-Ax}}{y().a[] \vdash a : 1, y : \bot}\ \text{T-}\bot \qquad \text{T-?}$$
$$M_1 \vdash a : 1, x : ?\bot$$

Using T-Cut to combine the client and server as before, R-ServerClient gives

$$(\nu x)(Q \mid M_1) \rightarrow y[] \mid y().a[]$$

in which the body of the server, $y[]$, is ready to interact with the body of the client, $y().a[]$.

Client three: uses the server twice

$$M_2 = ?x[x', x''].?x'[y].y().?x''[y].y().a[]$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{a[] \vdash a : 1}\,\text{T-Ax}}{y().a[] \vdash y : \bot, a : 1}\,\text{T-}\bot}{?x''[y].y().a[] \vdash x'' : ?\bot, a : 1}\,\text{T-?}}{y().?x''[y].y().a[] \vdash x'' : ?\bot, y : \bot, a : 1}\,\text{T-}\bot}{?x'[y].y().?x''[y].y().a[] \vdash x' : ?\bot, x'' : ?\bot, a : 1}\,\text{T-?}}{M_2 \vdash x : ?\bot, a : 1}\,\text{T-Contr}$$

Using T-Cut to combine the client and server, R-Contract gives

$$(\nu x)(Q \mid M_2) \rightarrow (\nu x')(Q' \mid (\nu x'')(Q'' \mid ?x'[y].y().?x''[y].y().a[]))$$

Q has been duplicated: $Q' = !x'(y).y[]$, $Q'' = !x''(y).y[]$.

Clients in parallel

$$M_1 = ?x'[y].y().a[] \qquad\qquad M_1' = ?x''[y].y().b[]$$

in parallel with the server, prefixed by a contraction:

$$(\nu x)(Q \mid ?x[x', x''].(M_1 \parallel M_1')).$$

$$\dfrac{\dfrac{\vdots}{Q \vdash x : !1} \qquad \dfrac{\dfrac{\dfrac{\vdots}{M_1 \vdash a : 1, x' : ?\bot} \qquad \dfrac{\vdots}{M_1' \vdash b : 1, x'' : ?\bot}}{M_1 \parallel M_1' \vdash a : 1, b : 1, x' : ?\bot, x'' : ?\bot}\ \text{T-Mix}}{?x[x', x''].(M_1 \parallel M_1') \vdash a : 1, b : 1, x : ?\bot}\ \text{T-Contr}}{(\nu x)(Q \mid ?x[x', x''].(M_1 \parallel M_1')) \vdash a : 1, b : 1}\ \text{T-Cut}$$

This example has some new syntax, $\parallel$, and a new typing rule, T-Mix.

The Mix rule

$$P ::= \ldots$$
$$P \parallel P \qquad \text{parallel composition}$$

$$\frac{P \vdash \Gamma \qquad Q \vdash \Delta}{P \parallel Q \vdash \Gamma, \Delta} \qquad \text{(T-Mix)}$$

$$\frac{P \to P'}{P \parallel Q \to P' \parallel Q} \qquad \text{(R-Par)}$$