Session Types and Mailbox Types — Simon Gay

Lecture 3 - *July 3, 2026*

1 From Propositions as Types to Processes as Proofs

The so-called Curry–Howard correspondence, or propositions-as-types, has been established among different calculi and logics. The point is not only the following dictionary.

$$\begin{array}{ccc} \text{propositions} & \leftrightarrow & \text{types} \\ \text{proofs} & \leftrightarrow & \text{programs} \\ \text{proof simplification} & \leftrightarrow & \text{computation,} \end{array}$$

This dictionary also connects two bodies of ideas that were not originally designed for one another. Wadler's discussion of propositions as types [8] presents it as a unifying idea on the same level as other large scientific identifications, such as the relation between geometry and algebra via co-ordinates. This is a useful way for computing science to explain its own intellectual content to non-specialists: not merely as technical infrastructure, but as a source of conceptual syntheses.

In the simply typed lambda calculus, the correspondence can be read directly from the typing rules. If one ignores the proof terms, the rules are rules of intuitionistic propositional logic; if one reads the proof terms, they are the typing rules of the lambda calculus. The computational step is the following.

$$(\lambda x.N)M \longrightarrow N[M/x]$$

is simultaneously beta-reduction and a proof simplification. Logically, one has introduced an implication $A \to B$ from a derivation of B under assumption A, and then immediately eliminated that implication using a proof of A. The introduction was unnecessary: the proof of A could have been plugged directly into the derivation that needed it.

There is also a categorical side to the same story. Cartesian closed categories are abstract models both of intuitionistic logic and of the simply typed lambda calculus. With that third leg included, the correspondence is often called Curry–Howard–Lambek. The lecture did not develop the category theory, but this observation is useful later because attempts to understand concurrency also repeatedly pass through categorical models of linear logic.

Compiled By:
Billy Bai, Henry Olson, Xusheng Zhi

2 Why Linear Logic Looked Like the Right Place

The original Curry–Howard correspondence is a foundation for sequential, functional computation. It was therefore natural to ask whether there is an analogous correspondence for concurrent computation. The hard part was not the desire for such a correspondence, but finding the right logical and computational structures to put into correspondence.

Girard's linear logic [5], introduced in 1987, provided a strong clue. In intuitionistic linear logic, ordinary implication can be decomposed into a linear implication together with a modality, for example by reading $A \Rightarrow B$ through the shape $!A \multimap B$. This is a typical kind of conceptual progress: something formerly treated as primitive is explained in terms of more fine-grained structure. The second part of Girard's paper, classical linear logic, was even more suggestive for concurrency. Girard explicitly suggested that linear logic might solve the problem of parallelism "at the logical level", by making successful communication depend on the fact that the communicating programs are proofs of something. He did not, however, give a concrete process calculus interpretation.

Classical linear logic also introduced the connective $\parr$, pronounced "par", whose name already hints at parallel computation. But the name alone is not a good enough explanation. The logical rule for $\parr$ is not literally a rule that puts processes in parallel. What eventually matters is how $\parr$ and its dual connective $\otimes$ participate in proof reduction, and how those proof reductions can be read as communication.

3 Abramsky, Bellin, and Scott: Cut as Communication

Samson Abramsky's early work on computational interpretations of linear logic [1] made a first concrete proposal, followed by Gianluigi Bellin and Philip J. Scott [2]. The central idea was to interpret the cut rule of classical linear logic as the composition of two communicating processes.

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

In the logic, A and $A^{\perp}$ cancel. In the process interpretation, the two ports named x are plugged together, and the rest of the interaction points remain visible. The terminology is slightly misleading: a rule called *cut* is interpreted operationally as putting processes together.

Abramsky first built a new process calculus tightly around the proof rules of classical linear logic. He then moved toward pi calculus processes as proof terms, an idea developed in the "proofs as processes" line by Abramsky, Bellin, and Scott. The important tensor/par rules are dual. In that setting, the tensor rule takes the following two separately derived pieces.

$$P \vdash \Gamma, y\colon A \qquad Q \vdash \Delta, z\colon B,$$

and packages the two interaction points y and z into a new point $x\colon A \otimes B$. The par rule takes a single process using both y and z and packages them into a single point $x\colon A \parr B$. In proof

reduction, a tensor proof cut against a par proof communicates: the artificial packaging around y and z disappears, and the processes that were already compatible on those channels are connected directly.

This is a genuine correspondence between communication and proof simplification. Nevertheless, it did not become the accepted Curry–Howard correspondence for concurrency. Prof. Simon Gay suggested two reasons. First, the original Curry–Howard correspondence was persuasive because it unified two mature, independently existing structures. Abramsky's first proposal linked a new logic to a process calculus invented for the purpose. Second, even the pi-calculus version used a computational model that was itself still relatively new. The pi calculus was powerful, especially because of *mobility*: the ability to send a channel as a message on another channel, thereby moving the ability to communicate with a process. But at that time it had not yet acquired the same established status as lambda calculus.

Abramsky also developed interaction categories, based on $*$-autonomous categories, which are abstract models of classical linear logic in the way Cartesian closed categories model intuitionistic logic. Interaction categories were an attempt to reach the semantic side of a concurrent Curry–Howard correspondence, but they lacked a satisfying syntactic process calculus side and did not account well for pi-calculus mobility. The same categorical structures later reappeared fruitfully in categorical quantum mechanics, for example in the work of Abramsky and Coecke.

4 The Session-Type Reorientation

Session types were being developed at roughly the same time by Honda and collaborators [6, 7]. In retrospect, there were semantic types inside interaction categories that corresponded closely to session types, but the connection was not made at the time. The decisive step came much later, in Caires and Pfenning's 2010 paper on session types as intuitionistic linear propositions [3]. That work gave a propositions-as-types correspondence between a session-typed form of pi calculus and a form of linear logic, and it generated a substantial line of later work with Toninho and others.

The lecture presents the key idea through Wadler's reformulation, *Propositions as Sessions* [8], which uses classical linear logic and a process language called CP. CP stands both for Caires–Pfenning and for classical processes. Wadler also introduced a corresponding functional language, GV, standing both for Gay–Vasconcelos and "good variation".

The essential difference between the older Abramsky–Bellin–Scott interpretation and the Caires–Pfenning–Wadler interpretation is visible in a small change of names. In the older tensor rule, two channels y and z are combined and replaced by a new channel x. In Wadler's rule, the two channels above the line are y and x, and the conclusion still contains x.

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

The channel x is not a fresh package replacing both old names. It is the same channel whose type changes from B to $A \otimes B$ as we move through the typing derivation. This is exactly the phenomenon seen in ordinary session typing: after communicating once on a channel, the same endpoint continues with the continuation type.

Thus $A \otimes B$ is read as "output an A, then behave as B", and $A \parr B$ is read as "input an A, then behave as B". Once this shift is made, the linear-logic notation replaces the earlier session-type notation: there is no need for separate syntactic constructors like $!A.B$ and $?A.B$. Tensor is output; par is input. This is considered the whole key idea of the correspondence.

5 CP: Syntax and Types

CP is a deliberately adjusted pi calculus. It contains a link process $x \leftrightarrow y$, which behaves like a wire: interacting on one endpoint turns into a substitution for the other. It has close and wait, written $x[]$ and $x().P$, but only wait has a continuation. Receive is written $x(y).P$ and binds y in P.

Send is the most significant syntactic change.

$$x[y].(P \mid Q).$$

The sent channel y is bound, and there are two subprocesses. The process P is responsible for the message y, while Q is responsible for the continuing behaviour on x. This mirrors the tensor rule, where the two premises are separate. In contrast with the earlier pi calculus, there is no null process. The closest terminal behaviour is closing a channel.

Choice is binary, with fixed labels inl and inr. Internal choice is written using left and right selection, $x \lhd \mathsf{inl}.P$ and $x \lhd \mathsf{inr}.P$, while external choice is $x \rhd \{P, Q\}$. CP also contains a nullary external choice $x \rhd \{\}$, included because the corresponding unit exists in the logic.

Most importantly, parallel composition and restriction are not primitive separate constructs. They are combined in cut.

$$(\nu x)(P \mid Q).$$

This connects two separate processes along a single channel name x. This single-binder formulation is possible because the typing discipline already tracks duality. The language is therefore more restrictive than the pi calculus from Lecture 1, and that restriction will matter for deadlock freedom.

The types are the connectives of classical linear logic.

$$\begin{array}{rcll} A & ::= & \bot & \text{wait} \\ & | & \mathbf{1} & \text{close} \\ & | & \top & \text{nullary external choice} \\ & | & \mathbf{0} & \text{nullary internal choice} \\ & | & A \parr B & \text{input} \\ & | & A \otimes B & \text{output} \\ & | & A \,\&\, B & \text{external choice} \\ & | & A \oplus B & \text{internal choice} \\ & | & !A & \text{server} \\ & | & ?A & \text{client.} \end{array}$$

The lecture mostly concentrated on the purely linear fragment, leaving the exponentials for a brief discussion. Duality exchanges the paired constants $\bot$ and $\mathbf{1}$, $\top$ and $\mathbf{0}$, tensor and par, with and plus, and server and client.

6 Typing and Communication in CP

The axiom rule is represented operationally by a link.

$$x \leftrightarrow y \vdash x\colon A, y\colon A^{\bot}.$$

Cutting a process against a link merely renames an interaction point. If $P \vdash \Gamma, x\colon A$, then cutting P with $x \leftrightarrow y$ gives a process typed with $y\colon A$, and the operational rule reduces it to $P[y/x]$.

The unit rules say that a channel of type $\mathbf{1}$ may be closed.

$$x[] \vdash x\colon \mathbf{1},$$

A channel of type $\bot$ may be waited on before continuing.

$$\frac{P \vdash \Gamma}{x().P \vdash \Gamma, x\colon \bot}.$$

The rule for $\top$ gives the strange-looking nullary case process $x \rhd \{\}$; there is no rule for $\mathbf{0}$.

The main communication rules match the session reading of the connectives. Tensor is output.

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

Par is input.

$$\frac{P \vdash \Gamma, y\colon A, x\colon B}{x(y).P \vdash \Gamma, x\colon A \parr B}.$$

Cut connects dual behaviours.

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

The reduction for send/receive is the heart of the operational reading.

$$(\nu x)(x[y].(P \mid Q) \mid x(y).M) \;\to\; (\nu y)(P \mid (\nu x)(Q \mid M)).$$

The notation deliberately uses the same bound name y on the sending and receiving sides, an "anti-Barendregt convention", so the rule can be written without an explicit substitution. After the communication, P is connected to M on the message channel y, and Q is connected to M on the continuation channel x. This exactly expresses that output sends a channel and then continues along the original session.

The close/wait and choice reductions have the same principal-cut flavour.

$$(\nu x)(x[] \mid x().P) \;\to\; P,$$

$$(\nu x)(x \lhd \mathsf{inl}.P \mid x \rhd \{Q, M\}) \;\to\; (\nu x)(P \mid Q), \qquad (\nu x)(x \lhd \mathsf{inr}.P \mid x \rhd \{Q, M\}) \;\to\; (\nu x)(P \mid M).$$

Structural congruence is correspondingly small: links are symmetric, cut is symmetric, and nested cuts can be reassociated.

7 Deadlock Freedom by Forbidding Cycles

The pi calculus from the earlier lectures allowed processes to be connected in arbitrary topologies, subject only to duality at each connection. Such topologies may contain cycles. Cycles are not automatically deadlocks, but they make deadlock possible: two processes connected by multiple channels can each insist on a different first communication.

CP rules out cyclic connection structures at the level of syntax and typing. The cut rule connects two separate processes along one channel, and after that one cannot create another independent connection between the same components. The resulting connection graph is essentially tree-shaped. This is a very conservative way of obtaining deadlock freedom: instead of allowing cycles and checking whether their communication dependencies are harmless, CP prevents the cycles from being built.

One could add a more liberal rule.

$$\frac{P \vdash \Gamma, x\colon A, y\colon A^{\perp}}{(\nu xy)P \vdash \Gamma},$$

which connects dual endpoints already present inside the same process. This increases expressivity, but it breaks the direct logical interpretation and admits deadlocks. Other systems regain deadlock freedom by more refined static techniques, such as ordering communication dependencies, as in work by Kobayashi and later by Dardha and Gay. The tradeoff is clear: CP gets a clean logical correspondence and strong global guarantees by accepting a substantial restriction on process topology.

8 Sharing, Exponentials, and Mix

The purely linear fragment cannot express reusable services conveniently. For example, one can encode booleans as follows.

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

and write a process implementing a two-input Boolean conjunction protocol, but that process is linear: it can be used only once. If a program needs many conjunctions, it must contain that many copies of the process.

Linear logic's exponentials address this. In Wadler's interpretation, $!A$ is a reusable server and $?A$ is a client capability. There is one introduction rule for the server side, while the client side has rules corresponding to use, weakening, and contraction. Operationally, a client request interacts with a server to create a fresh session.

$$(\nu x)(!x(y).P \mid ?x[y].Q) \rightarrow (\nu y)(P \mid Q).$$

Weakening discards an unused server, and contraction duplicates a server so that two client capabilities can use two renamed copies. The lecture is not able to fully expand on this extension, but intuitively, the conceptual role of then is: the exponentials are the controlled way to reintroduce copying and sharing into a linear world.

The reserve slides also mention a separate parallel composition construct $P \parallel Q$, governed by the logical mix rule. This is used, for example, when two clients run in parallel against duplicated server capabilities. It is not part of the minimal CP presentation in the same way as cut, but it is a standard extension when one wants independent parallel components.

9 Meta-Theory and the Logical Source of Reduction

The standard lemmas for CP look very much like the lemmas for the earlier session-typed pi calculus. If $P \vdash \Gamma$, then the free variables of P are exactly the variables in Γ. Typability is preserved by structural congruence. Substitution preserves typing when a channel name is renamed consistently. These are proved by straightforward inductions over typing or congruence derivations.

Preservation has the expected statement.

$$\text{if } P \vdash \Gamma \text{ and } P \rightarrow Q, \text{ then } Q \vdash \Gamma.$$

The proof proceeds by induction on the reduction derivation, inspecting the typing derivation of the source and rebuilding a typing derivation for the target. In the principal communication cases, the proof is essentially the same rearrangement as cut elimination: the derivations that introduced dual connectives are replaced by derivations for their continuations.

Progress is stated carefully because a well-typed open process may be waiting to interact with its environment. A process may reduce; or it may offer a prefix on some free variable; or, with mix, it may be only a parallel composition of links. For instance, a process typed with a free $y\colon \bot$ may be stuck internally but still offer a wait prefix on y, which is exactly what an environment using the dual endpoint may complete.

The deepest connection is with cut elimination in classical linear logic. The principal cut reductions are the communication rules: close/wait, send/receive, and select/case. Full cut elimination also needs commuting conversions, which move prefixes outward through cuts, and congruence rules that permit reductions under prefixes. These latter rules are not ordinary pi-calculus operational rules: reducing underneath an input prefix, for example, is unlike the usual blocking interpretation of input. But if all the logical reductions are included, cut elimination gives normalization for CP. If one removes the commuting conversions and prefix-congruence rules to obtain a more standard process semantics, termination still follows as a corollary because the operational system has fewer reductions.

10 Relation to GV and to the Caires–Pfenning Style

Wadler also defined GV, a session-typed functional language close in spirit to the Gay–Vasconcelos language [4], together with a translation from GV into CP preserving typing. Later, Lindley and Morris refined GV and gave it a direct operational semantics. The lecture noted that when current work says it is based on GV, it often means this Lindley–Morris refinement rather than exactly Wadler's original presentation.

The original Caires–Pfenning approach uses dual intuitionistic linear logic, with judgments of the following form.

$$\Gamma; \Delta \vdash P :: x\colon A.$$

Here Γ contains copyable assumptions and Δ contains linear assumptions, while the right side has a distinguished offered channel $x\colon A$. The connective $\multimap$ packages par as follows.

$$A \multimap B = A^{\bot} \parr B,$$

so input is written using linear implication, while tensor remains output.

Because the logic is presented with two-sided sequents, each connective has both an introduction rule and an elimination rule. This means a process construct can appear in two different-looking typing rules: for example, linear implication has an introduction rule corresponding directly to input, but its elimination rule involves output, because a process that is allowed to *use* an input-capable service must interact with it by sending. The cut rule connects a process offering $x\colon A$ on the right with a process using $x\colon A$ on the left.

Thus the Caires–Pfenning presentation and Wadler's CP presentation organize the same central idea differently. CP is closer to the one-sided classical linear logic notation and to the pi calculus used

earlier in the course. Caires–Pfenning is the style in which much of the subsequent session-types literature was developed. In both cases, the crucial insight is the same: session continuation is the proof-theoretic continuation of a linear-logical connective, and communication is cut elimination.

References

[1] Samson Abramsky. Proofs as processes. *Theoretical Computer Science*, 135(1):5–9, 1994.

[2] G. Bellin and P.J. Scott. On the π-calculus and linear logic. *Theoretical Computer Science*, 135(1):11–65, 1994.

[3] Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In Paul Gastin and François Laroussinie, editors, *CONCUR 2010 - Concurrency Theory*, pages 222–236, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[4] SIMON J. GAY and VASCO T. VASCONCELOS. Linear type theory for asynchronous session types. *Journal of Functional Programming*, 20(1):19–50, 2010.

[5] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987.

[6] Kohei Honda. Types for dyadic interaction. In *International Conference on Concurrency Theory*, pages 509–523. Springer, 1993.

[7] Kaku Takeuchi, Kohei Honda, and Makoto Kubo. An interaction-based language and its typing system. In *International Conference on Parallel Architectures and Languages Europe*, pages 398–413. Springer, 1994.

[8] Philip Wadler. Propositions as types. *Commun. ACM*, 58(12):75–84, November 2015.