Lecture Notes on Choice and Replication

15-816: Linear Logic
Frank Pfenning

Lecture 5
February 1, 2012

We continue to examine a computational interpretation of linear logic, where propositions are session types, and proof are π-calculus processes. The material is drawn from a recent paper by Caires et al. [CPT12], which contains additional details and further references.

1 Offering Output

We have already seen that $A \multimap B$, offering an input, also requires output in order to use the offer. But how do we offer an output? A natural candidate is $A \otimes B$, which is true if we can prove A and B separately, each using a portion of the available linear hypotheses.

$$\frac{\Delta \vdash A \quad \Delta' \vdash B}{\Delta, \Delta' \vdash A \otimes B} \otimes R$$

At first glance, it would appear process offer $A \otimes B$ along x should output two channels, one offering A and another offering B.

$$\frac{\Delta \vdash P :: y : A \quad \Delta' \vdash Q :: z : B}{\Delta, \Delta' \vdash P \; ? \; Q :: x : A \otimes B} \otimes R?$$

But we see that this doesn't quite work: P offers along y and Q offers along z, but neither y nor z are even mentioned in the final sequent. Looking at the pattern of $\multimap R$ we see that we need to exploit the linearity of the channel x along which $A \otimes B$ is offered. So we output only a single new channel of type A along x and then offer B, again along x. We could equally well output a new channel of type B and then behave as A, but it seems more natural if the continuation is on the right of the connective.

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

Again, this is a form of bound output, but this time as part of an offer.

Conversely, to communicate with a process offering $A \otimes B$ along x we have to input an A along x, after which we still have to communicate with x now of type B.

$$\frac{\Delta, y{:}A, x{:}B \vdash Q :: z : C}{\Delta, x{:}A \otimes B \vdash x(y).Q :: z : C} \otimes L$$

When we remove the process we recognize the usual left rule.

$$\frac{\Delta, A, B \vdash C}{\Delta, A \otimes B \vdash C} \otimes L$$

We should verify the cut reduction property and see which form of process reduction it suggests. First, on bare sequents:

$$\frac{\dfrac{\Delta_1 \vdash A \quad \Delta_2 \vdash B}{\Delta_1, \Delta_2 \vdash A \otimes B} \otimes R \quad \dfrac{\Delta, A, B \vdash C}{\Delta, A \otimes B \vdash C} \otimes L}{\Delta, \Delta_1, \Delta_2 \vdash C} \mathsf{cut}$$

$$\longrightarrow$$

$$\frac{\Delta_2 \vdash B \quad \dfrac{\Delta_1 \vdash A \quad \Delta, A, B \vdash C}{\Delta, \Delta_1, B \vdash C} \mathsf{cut}}{\Delta, \Delta_1, \Delta_2 \vdash C} \mathsf{cut}$$

We assign names and processes to the open premises:

$$\begin{array}{l} \Delta_1 \vdash P_1 :: y : A \\ \Delta_2 \vdash P_2 :: x : B \\ \Delta, w{:}A, x{:}B \vdash Q :: z : C \end{array}$$

With this process assignment, the reduction viewed on processes becomes:

$$\begin{array}{l} \Delta, \Delta_1, \Delta_2 \vdash (\nu x)((\nu y)\overline{x}\langle y \rangle.(P_1 \mid P_2) \mid x(w).Q) :: z : C \\ \longrightarrow \\ \Delta, \Delta_1, \Delta_2 \vdash (\nu x)(P_2 \mid (\nu y)(P_1 \mid Q\{y/w\})) :: z : C \end{array}$$

Again, this is a simple instance of the usual input/output reduction of the π-calculus, modulo some structural congruences.

2 Termination

The logical constant $\mathbf{1}$ means we have no antecedents. Consequently, the linear antecedent $\mathbf{1}$ is just replaced by the empty collection of antecedents, which means it is just erased.

$$\frac{}{\cdot \vdash \mathbf{1}}\ \mathbf{1}R \qquad \frac{\Delta \vdash C}{\Delta, \mathbf{1} \vdash C}\ \mathbf{1}L$$

A fruitful way to think of this is as the nullary version of $\otimes$, since it is in fact its unit. We would output nothing and have no continuation:

$$\frac{}{\cdot \vdash \overline{x}\langle\rangle.\mathbf{0} :: x : \mathbf{1}}\ \mathbf{1}R$$

In the π-calculus, the inactive or terminated process is represented by $\mathbf{0}$ which is the unit of parallel composition in that $P \mid \mathbf{0} \equiv P$.

Conversely, we can expect nothing from an offer of $\mathbf{1}$ along x, except to terminate the connection.

$$\frac{\Delta \vdash Q :: z : C}{\Delta, x{:}\mathbf{1} \vdash x().Q :: z : C}\ \mathbf{1}L$$

The two match perfectly, with the following cut reduction.

$$\frac{\dfrac{}{\cdot \vdash \mathbf{1}}\ \mathbf{1}R \quad \dfrac{\Delta \vdash C}{\Delta, \mathbf{1} \vdash C}\ \mathbf{1}L}{\Delta \vdash C}\ \mathsf{cut}$$

$$\longrightarrow$$

$$\Delta \vdash C$$

On process expressions, labeling the open premise as $\Delta \vdash Q :: z : C$:

$$(\nu x)(\overline{x}\langle\rangle.\mathbf{0} \mid x().Q) \longrightarrow Q$$

This can be seen as an interaction in the polyadic π-calculus.

3 Uncovering More Parallelism

Even though the assignment above perfectly fulfills our goal of a Curry-Howard isomorphism, we have proposed some alternatives in order to achieve more parallelism in the composition of independent processes. In general, a process P that does not offer anything is typed as $\Delta \vdash P :: x : \mathbf{1}$. Once composed with appropriate processes as required by Δ, it should be a closed process that evolves by internal actions only. If we have a second such process, $\Delta' \vdash Q :: z : \mathbf{1}$ each should be able to evolve independently, without interaction. We could achieve this by adding $x{:}\mathbf{1}$ to Δ' and then using cut.

$$\dfrac{\Delta \vdash P :: x : \mathbf{1} \quad \dfrac{\Delta' \vdash Q :: z : \mathbf{1}}{\Delta', x{:}\mathbf{1} \vdash x().Q :: z : \mathbf{1}}\ \mathbf{1}L}{\Delta, \Delta' \vdash (\nu x)(P \mid x().Q) :: z : \mathbf{1}}\ \mathsf{cut}$$

However, because of the action prefix $x()$ guarding Q, this corresponds to a *sequential composition* (P before Q) rather than a parallel composition. If we cut the other way, we have Q before P. In other words, we can not compose noninteracting processes in a truly parallel manner.

There are several ways we can obtain more parallelism. One is to make the $\mathbf{1}L$ rule *silent* in that we have the same proof term in the premise and conclusion [CP10]. In that case multiple different proofs collapse to the same process, so we no longer have an isomorphism. We say that proofs are *contracted* to programs, a phenomenon which also occurs in numerous other applications of the idea underlying the Curry-Howard isomorphism. We could also allow process reduction under a prefix, which is, however, unusual for the π-calculus.

We briefly explore writing $(x().\mathbf{0}) \mid Q$ as the proof term for $\mathbf{1}L$, recovering parallelism.

$$\dfrac{}{\cdot \vdash \overline{x}\langle\rangle.\mathbf{0} :: x : \mathbf{1}}\ (\mathbf{1}R) \qquad \dfrac{\Delta \vdash (x().\mathbf{0}) \mid Q :: z : C}{\Delta, x{:}\mathbf{1} \vdash Q :: z : C}\ (\mathbf{1}L)$$

Again we give up a full isomorphism, because we cannot tell exactly at which point $\mathbf{1}L$ has been applied due to the convention that we type modulo structural congruences.

We can make corresponding improvements to other rules[1]. For exam-

[1] as suggested by Favonia in class; see also Exercise 1

ple, in the $\otimes R$ rule

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

we notice that P can depend only on y, but not x (since it is typed in Δ and offers A along y). We can therefore safely rewrite it as:

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

This allows P to reduce and even interact with its environment in Δ instead of waiting until the channel y has been sent along x. Since the only occurrence of y is the one sent along x, communication along y will have to wait, however, until y has been sent and the receiving process is ready to communicate along it.

4 Example: File Indexing

We now illustrate the fragment of our system we have presented thus far through a simple example. We will further develop the example throughout the presentation, as we incrementally introduce the connectives that make up the full system.

Our example consists of a simple PDF indexing service. The high-level concept is a server that receives a PDF file from its client and then returns an indexed version of the file (a file containing a word index for searches). The system we have developed up to this point allows us to model such a server with the following type:

$$\mathsf{Idx} \triangleq \mathsf{file} \multimap (\mathsf{file} \otimes \mathbf{1})$$

We abstract away the details of what constitutes a proper PDF file with the type file. The type Idx describes the communication behavior that is expected of the server: it will first perform an input (of the file), followed by an output of a file, after which it terminates. A process that implements such a service along channel x is:

$$\mathsf{Srv} \triangleq x(f).(\nu y)\overline{x}\langle y \rangle.(P_y \mid \overline{x}\langle\rangle.\mathbf{0}) :: x : \mathsf{Idx}$$

We abstract away the details of the actual indexing of the file in the process P_y. The subscript here indicates that P will depend on y. A possible client for the server is given below (where the process Q represents the clients use of the indexed files):

$$\mathsf{Client} \triangleq (\nu p)\overline{x}\langle p\rangle.x(i).x().Q_{p,i}$$

and we can compose the server and the client as follows:

$$\frac{\cdot \vdash \mathsf{Srv} :: x : \mathsf{Idx} \quad x : \mathsf{Idx} \vdash \mathsf{Client} :: z : \mathbf{1}}{\cdot \vdash (\nu x)(\mathsf{Srv} \mid \mathsf{Client}) :: z : \mathbf{1}}\ \mathsf{cut}$$

5 Taking Stock, Part I

Let's look at the fragment of the logic so far. If we take the faithful interpretation of $\mathbf{1}$, we have on the process side:

$$\begin{array}{llcll}
\textit{Types} & A, B, C & ::= & A \multimap B & \text{input} \\
 & & \mid & A \otimes B & \text{output} \\
 & & \mid & \mathbf{1} & \text{termination} \\[1ex]
\textit{Processes} & P, Q & ::= & [x \leftrightarrow z] & \text{forwarding} \\
 & & \mid & (P \mid Q) & \text{parallel composition} \\
 & & \mid & (\nu x)P & \text{name restriction} \\
 & & \mid & x(y).P & \text{input} \\
 & & \mid & (\nu y)\overline{x}\langle y\rangle.P & \text{bound output} \\
 & & \mid & x().P & \text{wait} \\
 & & \mid & \overline{x}\langle\rangle.\mathbf{0} & \text{termination}
\end{array}$$

We also have the following reductions on well-typed processes, mimicking cut reductions. The typing requirement implies some straightforward conditions on the occurrences of bound variables which we do not detail separately.

$$\begin{array}{ll}
\textit{Structural} & (\nu x)([y \leftrightarrow x] \mid Q) \longrightarrow Q\{y/x\} \\
 & (\nu x)(P \mid [x \leftrightarrow z]) \longrightarrow P\{z/x\} \\
\textit{Interaction} & (\nu x)(x(y).P \mid (\nu w)\overline{x}\langle w\rangle.Q) \\
 & \qquad\qquad \longrightarrow (\nu x)(\nu w)(P\{w/y\} \mid Q) \\
\textit{Termination} & (\nu x)(\overline{x}\langle\rangle.\mathbf{0} \mid x().Q) \longrightarrow Q
\end{array}$$

and the following structural congruences, in addition to standard α-conversion:

$$\begin{array}{ll}
P \mid (Q \mid R) \equiv (P \mid Q) \mid R & \\
P \mid Q \equiv Q \mid P & \\
P \mid (\nu x)Q \equiv (\nu x)(P \mid Q) & \text{for } x \notin \mathsf{fn}(P) \\
(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P &
\end{array}$$

If we use the optimized form of $\mathbf{1}$, we replace $x().P$ by $x().\mathbf{0}$. We also need to add the congruences $P \mid \mathbf{0} \equiv P$ and $(\nu x)\mathbf{0} \equiv \mathbf{0}$.

6 External Choice

Branching, or *external choice*, means to offer both A and B along a channel and let the client decide which one to use. We add a binary guarded choice to the process calculus [SW01] and additive conjunction $A \mathbin{\&} B$ to our fragment of linear logic. Since only either A or B will be used (at the client's discretion), the context is propagated to both premises. Conversely, if $A \mathbin{\&} B$ is in the environment, we have to chose either A or B.

$$\frac{\Delta \vdash A \quad \Delta \vdash B}{\Delta \vdash A \mathbin{\&} B}\ \&R$$

$$\frac{\Delta, A \vdash C}{\Delta, A \mathbin{\&} B \vdash C}\ \&L_1 \qquad \frac{\Delta, B \vdash C}{\Delta, A \mathbin{\&} B \vdash C}\ \&L_2$$

When assigning process terms we again exploit linearity of channels. We indicate a choice (inl or inr) on a channel and then continue on the same channel with either A or B, respectively.

$$\frac{\Delta \vdash P :: x : A \quad \Delta \vdash Q :: x : B}{\Delta \vdash x.\mathsf{case}(P, Q) :: x : A \mathbin{\&} B}\ \&R$$

$$\frac{\Delta, x{:}A \vdash Q :: z : C}{\Delta, x{:}A \mathbin{\&} B \vdash x.\mathsf{inl}; Q :: z : C}\ \&L_1$$

$$\frac{\Delta, x{:}B \vdash Q :: z : C}{\Delta, x{:}A \mathbin{\&} B \vdash x.\mathsf{inr}; Q :: z : C}\ \&L_2$$

Reduction is also straightforward. In particular, process reduction and proof reduction match perfectly. We leave the details for the reader to reconstruct.

$$\begin{array}{l}(\nu x)(x.\mathsf{case}(P, Q) \mid x.\mathsf{inl}; R) \longrightarrow (\nu x)(P \mid R)\\ (\nu x)(x.\mathsf{case}(P, Q) \mid x.\mathsf{inr}; R) \longrightarrow (\nu x)(Q \mid R)\end{array}$$

7 Internal Choice

Choice, more precisely *internal choice*, means to offer either A or B along a channel x, the offering process is the one to decide. A party using x must therefore be prepared for both A and B. We do not need to extend the process language for this, having already added binary guarded choice, but we need $A \oplus B$ as an appropriate logical connective.

$$\frac{\Delta \vdash A}{\Delta \vdash A \oplus B}\ \oplus R_1 \qquad \frac{\Delta \vdash B}{\Delta \vdash A \oplus B}\ \oplus R_2$$

$$\frac{\Delta, A \vdash C \quad \Delta, B \vdash C}{\Delta, A \oplus B \vdash C}\ \oplus L$$

We present the process assignment with the guarded choice.

$$\frac{\Delta \vdash P :: x : A}{\Delta \vdash x.\mathsf{inl}; P :: x : A \oplus B}\ \oplus R_1$$

$$\frac{\Delta \vdash P :: x : B}{\Delta \vdash x.\mathsf{inr}; P :: x : A \oplus B}\ \oplus R_2$$

$$\frac{\Delta, x{:}A \vdash P :: z : C \quad \Delta, x{:}B \vdash Q :: z : C}{\Delta, x{:}A \oplus B \vdash x.\mathsf{case}(P, Q) :: z : C}\ \oplus L$$

The reduction was already discussed for branching.

8 Example: File Indexing Revisited

We now extend our PDF indexer to incorporate branching and choice, embodied in the types $\&$ and $\oplus$. The server so far only implements a single service: indexing. Conceivably, a server implements multiple services. For instance, we might want the server to check whether an indexed PDF is indeed indexed correctly. This verification service can be modelled through the following type:

$$\mathsf{Verif} \triangleq \mathsf{file} \multimap (\mathbf{1} \oplus \mathbf{1})$$

The verification service inputs the pdf file that is to be verified and then will either send inl or inr, indicating whether the file is valid or not. This form of choice that is internal to the server is modelled by the use of $\oplus$. A process implementing the verification service is (along channel x):

$$\cdot \Rightarrow x(f).\mathsf{Check} :: x : \mathsf{Verif}$$

with $\mathsf{Check} :: x : \mathbf{1} \oplus \mathbf{1}$ depending on the validity of the received file. To include this service in our original example, we make use of $\&$ to obtain:

$$\mathsf{Srv} \triangleq \mathsf{Idx} \mathbin{\&} \mathsf{Verif}$$

And the process offering the two services becomes

$$\cdot \Rightarrow x.\mathsf{case}(\mathsf{Srv}, x(f).\mathsf{Check}) :: x : \mathsf{Srv}$$

A client that wishes to use the verification service must then choose accordingly (and branch on the two possibilities, abstracted here by the processes P_{ok} and P_{nok}):

$$\mathsf{Client} \triangleq x.\mathsf{inr}; (\nu p)\overline{x}\langle p\rangle.x.\mathsf{case}(P_{\mathsf{ok}}, P_{\mathsf{nok}})$$

9 Replication

So far, the logic we have considered is purely linear. In order to capture process replication, we will need the proposition $!A$ as a type. A process offering a service $!A$ along x is offering the service A *persistently*: clients may use A as many times as they would like, including not at all. The judgmentally sound way to capture this logically uses persistent resources.

$$\underbrace{B_1, \ldots, B_k}_{\substack{\Gamma \\ \text{persistent}}} \; ; \; \underbrace{A_1, \ldots A_n}_{\substack{\Delta \\ \text{ephemeral}}} \vdash C$$

When we label the sequents with channels and processes, not much changes. We use a different kind of letter, u, to stand for *shared* channels. They may occur arbitrarily often in the process P, unlike the linear channels.

$$u_1{:}B_1, \ldots, u_k{:}B_k \; ; \; x_1{:}A_1, \ldots, x_n{:}A_n \vdash P :: z : C$$

All the rules we have shown so far are extended to allow a context Γ of persistent antecedents in the conclusion and all premises. This context satisfies weakening, contraction, and exchange, according to the persistent nature of the assumptions in them. We will use these properties silently.

10 Cut as Composition Revisited

Persistent truth is defined as truth not depending on ephemeral assumptions. Therefore we have a new rule of cut to eliminate a persistent antecedent A from a sequent, where one premise is a proof of *A with no linear antecedents*. This allows us to use this premise each time the persistent A is used, without violating any linearity constraints.

$$\frac{\Gamma\,;\cdot\vdash A \quad \Gamma, A\,;\Delta\vdash C}{\Gamma\,;\Delta\vdash C}\ \mathsf{cut!}$$

It is straightforward to annotate the premises with channels and processes, but what about the conclusion?

$$\frac{\Gamma\,;\cdot\vdash P :: x : A \quad \Gamma, u{:}A\,;\Delta\vdash Q :: z : C}{\Gamma\,;\Delta\vdash\ ? :: z : C}\ \mathsf{cut!}$$

The idea is for a shared channel u to represent the offer of a *replicated input*. What u receives is a *channel x of type A* along which a single session of type A can proceed. Meanwhile, u remains available to receive another channel of type A. Each use of A is satisfied by a different copy of P, as we will see. So:

$$\frac{\Gamma\,;\cdot\vdash P :: x : A \quad \Gamma, u{:}A\,;\Delta\vdash Q :: z : C}{\Gamma\,;\Delta\vdash (\nu u)(!u(x).P \mid Q) :: z : C}\ \mathsf{cut!}$$

Here, $!u(x).P$ is the π-calculus notation for a replicated input that persists even as it inputs a channel y for x.

Based on this intuition we have to send u a new channel $y{:}A$ in order to use u. This is realized in the copy rule. Logically, it is justified by saying that persistent truth entails ephemeral truth.

$$\frac{\Gamma, u{:}A\,;\Delta, y{:}A\vdash P :: z : C}{\Gamma, u{:}A\,;\Delta\vdash (\nu y)\overline{u}\langle y\rangle.P :: z : C}\ \mathsf{copy}$$

What happens if a cut! meets a copy? Logically, this is straightforward: we spawn two new cuts.

$$\dfrac{\Gamma ; \cdot \vdash A \qquad \dfrac{\Gamma, A ; \Delta, A \vdash C}{\Gamma, A ; \Delta \vdash C}\,\textsf{copy}}{\Gamma ; \Delta \vdash C}\,\textsf{cut!}$$

$$\longrightarrow$$

$$\dfrac{\Gamma ; \cdot \vdash A \qquad \dfrac{\Gamma ; \cdot \vdash A \quad \Gamma, A ; \Delta, A \vdash C}{\Gamma ; \Delta, A \vdash C}\,\textsf{cut!}}{\Gamma ; \Delta \vdash C}\,\textsf{cut}$$

The first one is considered smaller because the right subproof is shorter; the second one takes place on a smaller judgment (A linearly true vs. A persistently true). If we annotate these with processes, we have the open premises

$$\begin{array}{l} \Gamma ; \cdot \vdash P :: x : A \\ \Gamma, u{:}A ; \Delta, y{:}A \vdash Q :: z : C \end{array}$$

Replaying the proof reduction on the assigned processes yields:

$$\begin{array}{ll} & (\nu u)(!u(x).P \mid (\nu y)\overline{u}\langle y\rangle.Q) \\ \longrightarrow & (\nu y)(P\{y/x\} \mid (\nu u)(!u(x).P \mid Q)) \end{array}$$

With some structural congruences we obtain a familiar reduction for replicated input, in this case interacting with a bound output.

We also have to consider some additional interactions of the new judgmental rules with the old ones, specifically cut and identity. These are commuting conversions or structural equivalences between proofs, exchanging different forms of cut; on the π-calculus side they are behavioral equivalences called *sharpened replication theorems* [SW01]. We omit the details which can be found in [CP10].

So far, all the rules and reductions arose entirely from considerations at the level of judgments—we did not even introduce the $!A$ connective yet!

11 Sharing

After preparing the grounds by analyzing the judgment of persistent truth associated with shared channels, we can now internalize the judgment with the corresponding proposition $!A$. The $!R$ rule just requires the proof of the premise not to depend on linear assumptions. The $!L$ rule just promotes the linear antecedent $!A$ to the persistent antecedent A.

$$\frac{\Gamma\ ;\ \cdot \vdash A}{\Gamma\ ;\ \cdot \vdash !A}\ !R \qquad \frac{\Gamma, A\ ;\ \Delta \vdash C}{\Gamma\ ;\ \Delta, !A \vdash C}\ !L$$

From the process perspective, $!R$ introduces a replicated input, while $!L$ just corresponds to promoting a channel from linear to shared status.

$$\frac{\Gamma\ ;\ \cdot \vdash P :: y : A}{\Gamma\ ;\ \cdot \vdash !x(y).P :: x : !A}\ !R$$

$$\frac{\Gamma, u{:}A\ ;\ \Delta \vdash Q :: z : C}{\Gamma\ ;\ \Delta, x{:}!A \vdash x/u.Q :: z : C}\ !L$$

In order to retain an isomorphism between proofs and processes, and also respect the linearity of channels x, we have an explicit renaming construct $x/u.Q$ which binds u with scope Q. This is just an explicit substitution, for typing purposes, mapping into $Q\{x/u\}$.

We have to check that these rule match up. Fortunately, a cut at type $!A$ turns into a cut! at type A.

$$\frac{\dfrac{\Gamma\ ;\ \cdot \vdash A}{\Gamma\ ;\ \cdot \vdash !A}\ !R \quad \dfrac{\Gamma, A\ ;\ \Delta \vdash C}{\Gamma\ ;\ \Delta, !A \vdash C}\ !L}{\Gamma\ ;\ \Delta \vdash C}\ \mathsf{cut}$$

$$\longrightarrow$$

$$\frac{\Gamma\ ;\ \cdot \vdash A \quad \Gamma, A\ ;\ \Delta \vdash C}{\Gamma\ ;\ \Delta \vdash C}\ \mathsf{cut!}$$

If we annotate these with process expressions, we see that it just renames a bound variable and removes an explicit renaming construct.

$$(\nu x)(!x(y).P \mid x/u.Q) \longrightarrow (\nu u)(!u(y).P \mid Q)$$

We will have additional examples in the next lecture.

12 Taking Stock, Part II

The extension of the calculus by internal and external choice and replication does not create any new difficulties. Much of the computational content for replication is associated with the judgmental rules, rather than the proposition $!A$ itself.

We extend the syntax and the reductions.

Types	A, B, C	$::=$	$\dots$	
		$\mid$	$A \mathbin{\&} B$	external choice
		$\mid$	$A \oplus B$	internal choice
		$\mid$	$!A$	replication
Processes	P, Q	$::=$	$\dots$	
		$\mid$	$x.\mathsf{inl}; P \mid x.\mathsf{inr}; P$	selection
		$\mid$	$x.\mathsf{case}(P, Q)$	branching
		$\mid$	$!u(x).P$	replicating input
		$\mid$	$x/u.P$	promotion

We also have the following new reductions

(Structural) $(\nu x)(!x(y).P \mid x/u.Q) \longrightarrow (\nu u)(!u(y).P \mid Q)$

Choice $(\nu x)(x.\mathsf{case}(P, Q) \mid x.\mathsf{inl}; R) \longrightarrow (\nu x)(P \mid R)$
$(\nu x)(x.\mathsf{case}(P, Q) \mid x.\mathsf{inr}; R) \longrightarrow (\nu x)(Q \mid R)$

Replication $(\nu u)(!u(x).P \mid (\nu y)\overline{u}\langle y\rangle.Q)$
$\longrightarrow (\nu y)(P\{y/x\} \mid (\nu u)(!u(x).P \mid Q))$

Exercises

Exercise 1 Explore if the technique of uncovering additional parallelism in $\mathbf{1}L$ and $\otimes R$ from Section 3 also applies to $\multimap L$. If so, present the modified rule and an example where it could make a difference.

Exercise 2 *Explore the process interpretations of* $\mathbf{0}$ *and* $\top$*, the units of* $\oplus$ *and* $\&$*.*

Exercise 3 We have the following interaction laws for !

(i) $!!A \dashv\vdash !A$

(ii) $!(A \& B) \dashv\vdash (!A) \otimes (!B)$

In each case, give the sequent proofs in both directions. Then exhibit the process interpretation of your four proofs by giving a P such that $x{:}L \vdash P :: z : R$, where L is left-hand side and R is the right-hand side of the sequent.

Give an intuitive reading of each direction under the process interpretation.

Exercise 4 The identity expansions of the linear sequent calculus have no direct computational interpretation via reduction, but they do capture some process equivalences. Explain this informally using the example of the identity expansions for $A \multimap B$ and $!A$. In other words, explain how the process assignment for the identity rule at these two types is related to the process assigned to its identity expansion.

Exercise 5 We explore a variation of the coin exchange example from Section 7 of Lecture 4. Return to original formulation, using a *persistent*

$$u : \mathsf{d} \otimes \mathsf{d} \otimes \mathsf{n} \multimap \mathsf{q}$$

and give the process assignment that demonstrates how two dimes and a nickel can be exchanged for a quarter.

Then add a *persistent* $v : \mathsf{d} \multimap \mathsf{n} \otimes \mathsf{n}$ and exhibit the process that exchanges three dimes x_1, x_2 and x_3 to offer a quarter and a nickel ($\mathsf{q} \otimes \mathsf{n}$) along z.

You may use the channel output abbreviation introduced in Lecture 4 as appropriate to simplify the process expression.

References

[CP10] Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In *Proceedings of the 21st International Conference on Concurrency Theory (CONCUR 2010)*, pages 222–236, Paris, France, August 2010. Springer LNCS 6269.

[CPT12] Luís Caires, Frank Pfenning, and Bernardo Toninho. Towards concurrent type theory. In B. Pierce, editor, *Proceedings of the Workshop for Types in Language Design and Implementation*, TLDI'12, pages 1–12, Philadelphia, Pennsylvania, January 2012. ACM. Notes for an invited talk.

[SW01] Davide Sangiorgi and David Walker. *The π-Calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.