Lecture Notes on
Cut Reduction as Computation

15-816: Linear Logic
Frank Pfenning

Lecture 4
January 30, 2012

In this lecture we will examine a computational interpretation of linear logic in its sequent formulation. We largely follow a recent paper by Caires et al. [CPT12], which contains additional details and further references. The basic idea is that propositions correspond to *session types* [Hon93], proofs to process expressions in the π-calculus [MPW92], and cut reduction to process reduction. We do not assume that you already know the π-calculus, although clearly it will be easier to follow the lecture if you do.

1 Interpreting Judgments

In a functional setting, the basic judgment is generally of the form $M : A$, meaning either that M is a proof of A, or M is a term of type A. In the setting of communicating processes, it is unclear what it would mean to say that "P *is a process of type* A". Processes communicate with their environment, so we write instead $P :: x : A$, meaning P is a process offering service A along channel x. The channel here is considered a variable with scope P, so we can rename it as $P\{y/x\} :: y : A$ if y is not already used in P. As is customary, we will perform such renamings silently as appropriate.

Processes are uninteresting unless they are placed in a context where they not only *offer* services, but also *use* services offered by other processes. We write a *sequent*

$$x_1{:}A_1, \ldots, x_n{:}A_n \vdash P :: x : A$$

to express that process P offers service A along x when composed with processes P_i providing A_i along x_i for $1 \leq i \leq n$. All the variables x_i

must be distinct. This just a decoration of a sequent $\Delta \vdash A$ that labels the conclusion as well as all resources in Δ uniquely. We continue to write Δ for the resources, now labeled by channels.

Offering and using services are counterparts, but they are not the same. Therefore, formally, the judgment $x{:}A$ on the right of the sequent and the judgment $x_i{:}A_i$ on the left should be considered different. Since we can always tell by position which one is meant, we use the same notation for both.

Processes evolve through interactions along channels. Interacting on a channel x_i therefore engenders a change of state, and the same channel cannot be used again with the same type. Therefore the turnstile symbol '$\vdash$' denotes a *linear hypothetical judgment* [CCP03] where each antecedent must be used exactly once. Therefore the context is not subject to weakening or contraction, but reordering is permitted since antecedents are identified by unique names.

Even without defining any particular kind of service, some principles should hold for the judgments in general. We discuss these first, because they are an important guide to the rest of the development.

2 Cut as Composition

When a process P *offers* service A along x, and another process Q *uses* a service A along x, the two can be composed so that they communicate along x.

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

The process expression for the composition puts P and Q together in a *parallel composition* $(P \mid Q)$, sharing x as a *private* channel, as indicated by the *name restriction* (νx). Note that this rule entails some implicit renaming, because the channel along which P offers A must be equated with the channel along which Q uses A. In the π-calculus, $(\nu x)P$ is a binder for the variable x with scope P.

Viewed from the purely logical perspective, this is simply the rule of *cut* in linear logic:

$$\frac{\Delta \vdash A \quad \Delta', A \vdash C}{\Delta, \Delta' \vdash C}\ \mathsf{cut}$$

It is a little unusual that we use an intuitionistic version of linear logic with a singleton right-hand side in a sequent formulation. This is not absolutely

essential (see [Abr93] for a related classical counterpart), but it streamlines the judgmental justification of the system. It also reflects an intrinsic asymmetry between offering and using a service, even though we will see they are strongly related. It will also be beneficial in developing a dependent type theory.

3 Identity as Forwarding

One way to fulfill the promise of A along x is to just use a channel y that in turn promises A. This just corresponds to the identity rule.

$$\frac{}{y{:}A \vdash [y \leftrightarrow x] :: x : A}\ \mathsf{id}$$

There is no standard notation for forwarding in the π-calculus, so we write $[y \leftrightarrow x]$ to forward between y and x. We can *implement* the forwarding behavior in the untyped π-calculus, but it is convenient to have it as a primitive for the session type discipline.

In accordance with the linear hypothetical judgment, the antecedent $y{:}A$ must be the only one. In purely logical form:

$$\frac{}{A \vdash A}\ \mathsf{id}$$

In summary, cut and identity connect offers and uses of services. Cut composes a process P that offers A along x with a process that uses A along x, making x a private channel $(\nu x)(P \mid Q)$. Identity uses a channel y supplying A to satisfy its own offer of x along A, $[y \leftrightarrow x]$. These general principles are not connected to any particular services, which will be associated with logical connectives.

4 Composing Cuts

Multiple parallel compositions should be required to synchronize only to the extent that their interactions require it. Proof-theoretically, this means that the order of consecutive cuts should be insignificant. The corresponding laws have no inherent orientation as rewrite rules, so we think of them as structural proof equivalences. We leave it to the reader to write out

the simple proof figures. On the process calculus we obtain corresponding *structural congruences*.

$$(\nu x)((\nu y)(P \mid Q) \mid R) \equiv (\nu y)(P \mid (\nu x)(Q \mid R)) \quad \text{provided } x \notin \mathsf{fn}(P),\ y \notin \mathsf{fn}(R)$$
$$(\nu x)(P \mid (\nu y)(Q \mid R)) \equiv (\nu y)(Q \mid (\nu x)(P \mid R)) \quad \text{provided } x \notin \mathsf{fn}(Q),\ y \notin \mathsf{fn}(P)$$

These can be derived from more fundamental structural equivalences of associativity of parallel composition and scope extrusion.

$$\begin{array}{lcll} (P \mid Q) \mid R & \equiv & P \mid (Q \mid R) & \text{associativity} \\ P \mid Q & \equiv & Q \mid P & \text{commutativity} \\ P \mid (\nu x)Q & \equiv & (\nu x)(P \mid Q) & \text{scope extrusion} \\ & & & \text{provided } x \notin \mathsf{fn}(P) \end{array}$$

The last rule comes with a side condition, namely that the variable x is not among the free names in P. Since (νx) is a binder, this just makes sure that two different variables with the same name are not confused. We can always (silently) rename the bound variable to allow the scope extrusion to happen in case there is a conflict.

In general, we identify processes up to structural congruence. Other approaches are possible, but this is convenient for our purposes here. A disadvantage of this approach is that some care is needed in implementing an algorithm for type-checking processes, because we may need to rearrange expressions by structural congruences first, before the typing rule becomes applicable.

5 Input

We approach the individual logical connectives by thinking of their meaning as defined by their *right rules* in the sequent calculus. In their process interpretation, we have to analyze what it means to *offer* a particular service along a channel. Linear implication $A \multimap B$ is true if B is true under the (linear) assumption A.

$$\frac{\Delta, A \vdash B}{\Delta \vdash A \multimap B}\ \multimap R$$

Reading the premise under the process interpretation

$$\Delta, y{:}A \vdash P :: x : B$$

it says that P offers B along x if provided with the opportunity to use A along y. So a process offering $A \multimap B$ must *input* an A and then offer B.

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

Note that we use the same channel name x for the service $A \multimap B$ and then the service B. This is possible because channels are *linear*. Once we input an A along x, we can not input another along x. Instead, the channel x changes state. This is the origin of the term *session types* (or, more broadly, *behavioral types*) for this kind of system. The type $A \multimap B$ describes the type of a session that inputs an A and then behaves like B.

Under this definition, how can we *use* a channel x of type $A \multimap B$? We have to *output* an A along x. In return, we assume that x now offers B. But what precisely does it mean to "*output an* A"? In the π-calculus we do not pass processes, we pass *names* that provide access to processes. So we must have a process P offering A along y and then output that y along x.

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

Note that each channel in the context must be used either in P or in Q, but not both. This is essential to maintain the linearity in the use of channels. Also, the channel y along which P must offer A must be *bound* in the conclusion, so that there cannot be confusion with any other channel. In the π-calculus this is called a *bound output*, which always outputs a fresh name. Note also that P and Q do not share any names, so they do not communicate with each other directly.

If we strip the process expressions, we obtain just the usual left rule for linear implication in the linear sequent calculus.

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

6 Reduction

We can define right and left rules in the sequent calculus at will, but to form a coherent logical system they must match appropriately. As a global theorem about a logic, the theorems witnessing such coherence are cut elimination and identity. These decompose into two local properties, separately

for each connective, namely *cut reduction* and *identity expansion*. Cut reduction corresponds to process reduction, while identity expansion shows how to reduce channel forwarding at complex types to forwarding at simpler types.

We first consider cut reduction in the sequent calculus. We want to show that the right and left rules match. This means that if we have a cut where the cut formula was just introduced by its left and right rules, then we can reduce it to cuts on subformulas. For linear implication we have

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

$$\longrightarrow$$

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

To obtain a process interpretation of this reduction, we first assign names and processes to the three premises:

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

The annotated conclusions before and after the reduction then are:

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

In order to read this more easily, we can apply the structural congruences of the π-calculus, extruding the bindings on x and w and exchanging P_1 and P_2:

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

We see that in the presence of the structural rules, the cut reduction is mirrored by a process reduction, matching an input with a corresponding output. In general, it is an instance of the reduction

$$(x(y).P \mid \overline{x}\langle w\rangle.Q) \longrightarrow (P\{w/y\} \mid Q)$$

7 Example: Coin Exchange

In this simple example we see which process expression corresponds to exchanging two dimes and a nickel for a quarter. Recall that we write the rule as $\mathsf{d} \otimes \mathsf{d} \otimes \mathsf{n} \multimap \mathsf{q}$. Unfortunately, we have not yet discussed the process interpretation of simultaneous conjunction ($\otimes$), but there is an isomorphic formulation without it:

$$\mathsf{d} \multimap (\mathsf{d} \multimap (\mathsf{n} \multimap \mathsf{q}))$$

The relationship between the two is called Currying, honoring the logician Curry who used this transformation (even if he did not invent it). Let's prove that with two dimes and a nickel, we can obtain a quarter if we have a single copy of the above.

$$\dfrac{\dfrac{\dfrac{}{\mathsf{d} \vdash \mathsf{d}}\,\mathsf{id} \quad \dfrac{\dfrac{\dfrac{}{\mathsf{n} \vdash \mathsf{n}}\,\mathsf{id} \quad \dfrac{}{\mathsf{q} \vdash \mathsf{q}}\,\mathsf{id}}{\mathsf{n}, \mathsf{n} \multimap \mathsf{q} \vdash \mathsf{q}} \multimap L}{\mathsf{d}, \mathsf{n}, \mathsf{d} \multimap (\mathsf{n} \multimap \mathsf{q}) \vdash \mathsf{q}} \multimap L}{\mathsf{d}, \mathsf{d}, \mathsf{n}, \mathsf{d} \multimap (\mathsf{d} \multimap (\mathsf{n} \multimap \mathsf{q})) \vdash \mathsf{q}} \multimap L$$

We now assign channel names, and then processes. Let's start at the bottom. We label our resources x_i, which we interpret as names for the coins we own. The coin exchange service is labeled x, and the name of the quarter we want to obtain is z.

$$x_1{:}\mathsf{d}, x_2{:}\mathsf{d}, x_3{:}\mathsf{n}, x{:}\mathsf{d} \multimap (\mathsf{d} \multimap (\mathsf{n} \multimap \mathsf{q})) \vdash P :: z : \mathsf{q}$$

We have written P as a placeholder for the process we still want to construct. Let's take one step in the bottom-up construction of this proof, and also fill in the identity.

$$\dfrac{\dfrac{}{x_1{:}\mathsf{d} \vdash [x_1 \leftrightarrow y_1] :: y_1 : \mathsf{d}}\,\mathsf{id} \qquad x_2{:}\mathsf{d}, x_3{:}\mathsf{n}, x{:}\mathsf{d} \multimap (\mathsf{n} \multimap \mathsf{q}) \vdash P' :: z : \mathsf{q}}{x_1{:}\mathsf{d}, x_2{:}\mathsf{d}, x_3{:}\mathsf{n}, x{:}\mathsf{d} \multimap (\mathsf{d} \multimap (\mathsf{n} \multimap \mathsf{q})) \vdash (\nu y_1)\overline{x}\langle y_1\rangle.([x_1 \leftrightarrow y_1] \mid P') :: z : \mathsf{q}} \multimap L$$

Notice that this determines the shape of P, although P' remains as an unknown, to be determined by further proof construction. Of course, the next step is essentially the same as the previous one:

$$\dfrac{\dfrac{}{x_2{:}\mathsf{d} \vdash [x_2 \leftrightarrow y_2] :: y_2 : \mathsf{d}}\,\mathsf{id} \qquad x_3{:}\mathsf{n}, x{:}\mathsf{n} \multimap \mathsf{q} \vdash P'' :: z : \mathsf{q}}{x_2{:}\mathsf{d}, x_3{:}\mathsf{n}, x{:}\mathsf{d} \multimap (\mathsf{n} \multimap \mathsf{q}) \vdash (\nu y_2)\overline{x}\langle y_2\rangle.([x_2 \leftrightarrow y_2] \mid P'') :: z : \mathsf{q}}\,\multimap L$$

Focusing in again on the second premise, we can now complete the proof.

$$\dfrac{\dfrac{}{x_3{:}\mathsf{d} \vdash [x_3 \leftrightarrow y_3] :: y_3 : \mathsf{d}}\,\mathsf{id} \quad \dfrac{}{x{:}\mathsf{q} \vdash [x \leftrightarrow z] :: z : \mathsf{q}}\,\mathsf{id}}{x_3{:}\mathsf{n}, x{:}\mathsf{n} \multimap \mathsf{q} \vdash (\nu y_3)\overline{x}\langle y_3\rangle.([x_3 \leftrightarrow y_3] \mid [x \leftrightarrow z]) :: z : \mathsf{q}}\,\multimap L$$

The overall process expression is now:

$$\begin{array}{l}(\nu y_1)\overline{x}\langle y_1\rangle.([x_1 \leftrightarrow y_1] \\ \quad \mid (\nu y_2)\overline{x}\langle y_2\rangle.([x_2 \leftrightarrow y_2] \\ \qquad \mid (\nu y_3)\overline{x}\langle y_3\rangle.([x_3 \leftrightarrow y_3] \\ \qquad\quad \mid [x \leftrightarrow z])))\end{array}$$

We can abbreviate $(\nu y)\overline{x}\langle y\rangle.([x' \leftrightarrow y] \mid P) = \overline{x}\langle x'\rangle.P$, since in the above inference pattern x' nor y can appear in P. Essentially, the use of the identity turns the bound output into a regular output. Then in the judgment

$$x_1{:}\mathsf{d}, x_2{:}\mathsf{d}, x_3{:}\mathsf{n}, x{:}\mathsf{d} \multimap (\mathsf{d} \multimap (\mathsf{n} \multimap \mathsf{q})) \vdash P :: z : \mathsf{q}$$

the process term P can be written as

$$\overline{x}\langle x_1\rangle.\overline{x}\langle x_2\rangle.\overline{x}\langle x_3\rangle.[x \leftrightarrow z]$$

Reading it out, it means: "*send x_1, then x_2, then x_3, all along x, and then forward x to z*". Since x_1 and x_2 are dimes, and x_3 is a nickel, while x and z are names for quarters, this makes sense.

Exercises

Exercise 1 Write out the ways in which two consecutive cuts on unrelated propositions A and B can be permuted without changing the conclusion. Which structural congruences on associated processes does this entail?

Exercise 2 The cut reduction given for $\multimap R$ against $\multimap L$ is just one of two possibilities.

(i) Show the other possible cut reduction.

(ii) Show the process assignment before and after the alternative cut reduction.

(iii) Discuss its relationship to the one shown in Section 6.

Exercise 3 In Section 7 we introduced the abbreviation $(\nu y)\overline{x}\langle y\rangle.([x' \leftrightarrow y] \mid P) = \overline{x}\langle x'\rangle.P$ under certain conditions. This can be seen as a process term assignment for a derived rule of inference. Determine this derived rule of inference, and read off the conditions under which the abbreviation above makes sense.

Exercise 4 We expand on the example in Section 7. Assume thats besides $x{:}\mathsf{d} \multimap (\mathsf{d} \multimap (\mathsf{n} \multimap \mathsf{q}))$ we also have $w{:}\mathsf{n} \multimap (\mathsf{n} \multimap \mathsf{d})$ for changing two nickels into a dime. Show a process term for changing five nickels $x_i{:}\mathsf{n}$ for $1 \leq i \leq 5$ into a quarter $z : \mathsf{q}$. You may use the abbreviated notation for output that is not bound, if appropriate.

References

[Abr93] Samson Abramsky. Computational interpretations of linear logic. *Theoretical Computer Science*, 111:3–57, 1993.

[CCP03] Boy-Yuh Evan Chang, Kaustuv Chaudhuri, and Frank Pfenning. A judgmental analysis of linear logic. Technical Report CMU-CS-03-131R, Carnegie Mellon University, Department of Computer Science, December 2003.

[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.

[Hon93] Kohei Honda. Types for dyadic interaction. In *4th International Conference on Concurrency Theory*, CONCUR'93, pages 509–523. Springer LNCS 715, 1993.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes. *Information and Computation*, 100(1):1–77, September 1992. Parts I and II.