Pfenning Proof Theory Lecture 5: Linear Logic & Communication

June 23 2022

1 Introduction

> Our goal is to define logical foundations without assuming what already exists in mathematics. There is a strong connection between constructive proof and functional programming.

Questions for Today:

1. In ND (i.e. natural deduction), we check soundness by these reductions. Can we do a similar test on SEQ (i.e. sequent calculus)?

2. ND has a nice computational interpretation as functional programming. What about computational interpretation of SEQ?

We will address these questions by looking at intuitionistic *linear* logic, as that is an easier setting.

The key difference in the linear setting is that when you use an assumption, it's gone; you don't get to reuse them.

The correspondence between regular intuitionistic logic and intuitionistic linear logic is roughly:

Intuitionistic	Intuitionistic Linear
$\supset$	$\multimap$ (neg)
$\wedge$	$\&$ (neg) $\otimes$ (pos)
$\top$	$\top$ (neg) 1 (pos)
$\vee$	$\oplus$ (pos)
$\bot$	0 (pos)

2 Sequent Calculus for Intuitionistic Linear Logic

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B} \multimap R \qquad \frac{\Gamma \vdash A \qquad \Delta, B \vdash C}{\Gamma, \Delta, A \multimap B \vdash C} \multimap L$$

$$\frac{}{\vdash 1} 1R \qquad \frac{\Gamma \vdash C}{\Gamma, 1 \vdash C} 1L$$

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \& B} \& R \qquad \frac{\Gamma, A \vdash C}{\Gamma, A \& B \vdash C} \& L_1$$

$$\frac{\Gamma, B \vdash C}{\Gamma, A \& B \vdash C} \& L_2$$

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

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

$$\frac{\Gamma, A \vdash C \qquad \Gamma, B \vdash C}{\Gamma, A \oplus B \vdash C} \oplus L$$

$$\frac{}{A \vdash A} \text{ID}$$

$$\frac{\Gamma, A \qquad \Delta, A \vdash C}{\Gamma, \Delta \vdash C} \text{CUT}$$

$$\frac{}{\Gamma \vdash \top} \top R \qquad \frac{}{\Gamma, 0 \vdash C} 0L$$

Note: There is no T-L rule and no 0-R rule

Example Let's look at cut for &.

$$\dfrac{\dfrac{\Gamma \vdash A(\mathcal{D}_1) \qquad \Gamma \vdash B(\mathcal{D}_2)}{\Gamma \vdash A\&B}\,\&R \qquad \dfrac{\Delta, A \vdash C(\mathcal{E}_1)}{\Delta, A\&B \vdash C}\,\&L_1}{\Gamma, \Delta \vdash C}\,CUT_{A\&B}$$

reduces to

$$\dfrac{\Gamma \vdash A(\mathcal{D}_1) \qquad \Delta, A \vdash C(\mathcal{E}_1)}{\Gamma, \Delta \vdash C}\,CUT_A$$

Our goal was to provide a computational interpretation of a sequent calculus. It turns out that we can interpret these rules in terms of channels!

$$\dfrac{\dfrac{\Gamma \vdash A(\mathcal{D}_1) \qquad \Gamma \vdash B(\mathcal{D}_2)}{\Gamma \vdash x : A\&B}\,\&R \qquad \dfrac{\Delta, A \vdash C(\mathcal{E}_1)}{\Delta, x : A\&B \vdash C}\,\&L_1}{\Gamma, \Delta \vdash C}\,CUT_{A\&B}$$

View the derivation of the left premise of the cut (P) and the derivation of the right premise of the cut (Q) as processes interacting along a communication channel x. The right side is making a choice to ask for the first part (A) and sending that information along the channel, which is going to respond with an element of type A.

So we can annotate the preceding derivations with the channel name x.

$$\dfrac{\dfrac{\Gamma \vdash A(\mathcal{D}_1)}{\Gamma \vdash x : A \oplus B}\,\oplus R_1 \qquad \dfrac{\Delta, A \vdash C(\mathcal{E}_1) \qquad \Delta, B \vdash C(\mathcal{E}_2)}{\Delta, x : A \oplus B \vdash C}\,\oplus L}{\Gamma, \Delta \vdash C}\,CUT_{A\oplus B}$$

reduces to

$$\dfrac{\Gamma \vdash A(\mathcal{D}_1) \qquad \Delta, A \vdash C(\mathcal{E}_1)}{\Gamma, \Delta \vdash C}\,CUT_A$$

If you're familiar with the pi calculus, this would be the program $(\nu.x)(P|Q)$, with Q sending either π_1 or π_2 to P.

Question Do we need to interpret these as processes? No, we don't *need* to, but it is natural.

Question Does information always flow from right tree to left tree? No, it depends on whether the type being cut on is positive or negative.

Going one step further, we can give CUT a proof term:

$$\frac{\Gamma \vdash P_x :: (x : A) \qquad \Delta, x : A \vdash Q_x :: (w : C)}{\Gamma, \Delta \vdash (x \leftarrow P(x); Q(x)) :: (w : C)} \, CUT_A$$

Here, x is a new, unidirectional communication channel along which exactly one message will be sent. The process P will write to it, and the process Q will read from it.

2.1 Communication direction and polarity of types

When we introduced the intuitionistic linear connectives, we mentioned that connectives are "positive" or "negative". Computationally, this corresponds to whether you have all the information (positive) or need to make a choice (negative). Positive connectives send information from the first premise to the second premise of a cut (left to right in the proof tree, but right rule to left rule). Negative connectives send information in the opposite direction, from the second premise to the first.

This distinguishes $A \oplus B$ and $A \& B$: to introduce both of them, we provide left and right derivations, but operationally, different processes choose left vs right. $A \oplus B$ is positive, and the producer (left derivation) decides whether it's an A or a B (like a regular intuitionistic $\vee$). $A \& B$ is negative, and is a lazy pair where the consumer (right derivation) will ask for an A or a B. $A \otimes B$ is positive, and is simply a pair of an A and a B which is passed to the consumer. $A \multimap B$ is negative, and is a function waiting for be applied (waiting for the consumer to send an argument of type A).

For the other connectives, $\top$ is negative, 0 is positive, 1 is positive.

2.2 Synchronous and Asynchronous Calculi

From before, what channel do they talk on after the initial reduction?

$$\frac{\dfrac{\Gamma \vdash y : A(\mathcal{D}_1)}{\Gamma \vdash x : A \oplus B} \oplus R_1 \qquad \dfrac{\Delta, y : A \vdash C(\mathcal{E}_1) \qquad \Delta, B \vdash C(\mathcal{E}_2)}{\Delta, x : A \oplus B \vdash C} \oplus L}{\Gamma, \Delta \vdash C} \, CUT_{A \oplus B}$$

reduces to

$$\frac{\Gamma \vdash y : A(\mathcal{D}_1) \qquad \Delta, y : A \vdash C(\mathcal{E}_1)}{\Gamma, \Delta \vdash C} \, CUT_A$$

by communicating not just π_1, but $\pi_1(y)$, including the continuation (y).

The computational interpretation of this sequent calculus is *synchronous* concurrency, with cut being the communication step. Synchronous concurrency is basically impossible to implement, so we'll now try to make things asynchronous.

For asynchronous communication, we're going to want the cut to disappear entirely, rather than just being pushed inward. We're going to "axiomatize" the right rule for $\oplus$ and the left rules for &. We will have that things of positive types will be messages and things of negative type will be processes.

We get rid of $\& L_1$ and $\& L_2$ and we add axioms:

$$A \& B \vdash A \; \& A_1$$

$$A \& B \vdash B \; \& A_2$$

So now we might build the derivation:

$$\dfrac{\dfrac{\Gamma \vdash A(\mathcal{D}_1) \qquad \Gamma \vdash B(\mathcal{D}_2)}{\Gamma \vdash A \& B}\,\& R \qquad \dfrac{\pi_1}{A \& B \vdash A}\,\& A_1}{\Gamma \vdash A}\, CUT_{A\&B}$$

reducing to

$$\Gamma \vdash A(\mathcal{D}_1)$$

The axioms represent messages: π_1 is a message saying "give me the first part". The rest of the derivations are processes. So this reduction is a "message receive".

We can do this for the other connectives as well. For $\oplus$:

$$\dfrac{\pi_1}{A \vdash A \oplus B}\,\oplus A_1$$

$$\dfrac{\pi_2}{B \vdash A \oplus B}\,\oplus A_2$$

(Reusing the name π between $\oplus$ and & is fine because they are distinguished by the types.)

We can give proof terms to these rules:

$$\dfrac{}{x : A \vdash \mathrm{msg}\, y(\pi_1 x) :: (y : A \oplus B)} \qquad \dfrac{}{x : B \vdash \mathrm{msg}\, y(\pi_2 x) :: (y : A \oplus B)}$$

$$\dfrac{}{x : A \& B \vdash \mathrm{msg}\, y(\pi_1 x) :: (y : A)} \qquad \dfrac{}{x : A \& B \vdash \mathrm{msg}\, y(\pi_2 x) :: (y : B)}$$

$$\dfrac{\Gamma_1, x : A \vdash P :: (w : C) \qquad \Gamma_1, x : B \vdash Q :: (w : C)}{\Gamma, y : A \oplus B \vdash \mathrm{recv}\, y(\pi_1 x \Rightarrow P | \pi_2 x \Rightarrow Q)(w : C)}$$

We can use this to write a simple program

$$\mathrm{msg}\, y(\pi_2 a) |\, \mathrm{recv}\, y(\pi_1 x \Rightarrow P | \pi_2 x \Rightarrow Q)$$

which steps to

$$P[x \coloneqq a]$$

Instead of the message-passing interpretation, we can also interpret these as a kind of shared memory: "recv" is like reading from a location, "msg" is like writing to it, and :: $(y : A)$ specifies the location Y where a process's return value should be written.

2.3 Doing computation

Finally, let's see how we can actually write programs in such a language.

Consider the (recursive) type of binary sequences:

$$\mathrm{bin} = (b_0 : \mathrm{bin}) \oplus (b_1 : \mathrm{bin}) \oplus (e : 1)$$

Suppose we wish to write a function that flips sequences. We might try something like:

$$\begin{aligned} x : \mathrm{bin} \vdash \mathrm{flip} :: (y : \mathrm{bin}) = \mathrm{recv}\, x(& \\ & b0(x') \Rightarrow \\ & b1(x') \Rightarrow \\ & be(x') \Rightarrow \\ ) & \end{aligned}$$

But this won't work because we don't have a continuation after sending a message in our asynchronous language. So instead we annotate flip with x, y:

$$\begin{aligned} x : \mathrm{bin} \vdash \mathrm{flip}_{x,y} :: (y : \mathrm{bin}) = \mathrm{recv}\, x(& \\ & b0(x') \Rightarrow y' \leftarrow \mathrm{flip}(x', y'); \mathrm{msg}\, y(b1(y')) \\ & b1(x') \Rightarrow y' \leftarrow \dots \\ & e(x') \Rightarrow y'\, \mathrm{msg}\, y(e(x')) \\ ) & \end{aligned}$$

2.4 Parting thoughts on linear logic

Two ways of recovering general computation:

- Introduce $!A$ (read "of course A") as a type constructor, which recovers nonlinearity.
- Introduce recursive types and recursive definitions.

After starting with the first option, Frank has found the second option to be preferable.

For more on linear logic, you can check out Frank's lecture notes on linear logic (`http://www.cs.cmu.edu/~fp/courses/15816-s12/`), plus notes from last fall that provide a somewhat more computational interpretation (`http://www.cs.cmu.edu/~fp/courses/15814-f21/schedule.html`).