Lecture Notes on
A Mixed Linear/Nonlinear Logic

15-836: Substructural Logics
Frank Pfenning

Lecture 10
September 28, 2023

1 Introduction

As we have seen in the last lecture, introducing the judgment of validity and then internalizing it as the exponential modality $!A$ permits a compositional translation of structural (intuitionistic) logic into linear logic. At this point we could declare victory and see if similar techniques apply to other logics, e.g., if there is some embedding of linear logic into ordered logic.

However, there are some nagging issues, despite the elegance of the cut elimination proof.

1. The exponential $!A$ is neither positive nor negative in that it is invertible neither on the right nor on the left. On the right, in a judgment $\Gamma\ ;\ \Delta \vdash\ !A$ we may have to wait until Δ becomes empty before applying $!R$. On the left, we can move the linear antecedent $!A\ \mathit{true}$ to $A\ \mathit{valid}$, but we cannot necessarily move it back immediately to $A\ \mathit{true}$ because that renders the linear context nonempty (thereby possibly preventing $!R$).

2. If we have a particular interpretation of intuitionistic (structural) logic in mind, then translation to linear logic may mangle this interpretation. Among other concerns, a notion of observability may not be preserved, in which case the translation couldn't properly serve as a compiler.

So it is worth looking for a direct combination of logics, rather than translating one into the other. If structural logic represents functional programming, and linear logic message-passing, then we'd look for a way to combine functional and message-passing programming.

Today, we take a first step in this direction by developing a mixed linear/nonlinear logic (mostly) following Benton [1994]. Since ours is a minor variant, we also call it LNL. We will see that it repairs some of the noted issues. Moreover, it

presents a clear path towards further generalization in the form of *adjoint logic* [Reed, 2009, Pruiksma et al., 2018].

2 Shifting Between Logics

Given the desire to keep the native meaning of both structural and linear logic, we place them in two different strata and speculate that we may just go back and forth between them using two shift operators.

$$\begin{array}{lrcl} \text{Structural} & A_{\mathsf{S}} & ::= & P_{\mathsf{S}} \mid A_{\mathsf{S}} \supset B_{\mathsf{S}} \mid A_{\mathsf{S}} \wedge B_{\mathsf{S}} \mid \top \mid A_{\mathsf{S}} \vee B_{\mathsf{S}} \mid \bot \mid \uparrow A_{\mathsf{L}} \\ \text{Linear} & A_{\mathsf{L}} & ::= & P_{\mathsf{L}} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid A_{\mathsf{L}} \mathbin{\&} B_{\mathsf{L}} \mid \top \mid A_{\mathsf{L}} \oplus B_{\mathsf{L}} \mid \mathbf{0} \mid \downarrow A_{\mathsf{S}} \end{array}$$

We call S and L *modes*.

The key questions are: Which properties do we expect from the combination, and which laws should the two shifts satisfy so that these properties hold? At least, we the combination should satisfy cut and identity elimination. Beyond that, the combination should be *conservative* over each fragment in a strong sense: not only do we want purely structural or purely linear propositions to be true precisely if they are true in purely structural or purely linear logic, but we also want them to have essentially the same proofs.

A lesson from the study of validity is that we cannot allow validity (here: truth of a structural proposition) to depend on truth (here: truth of a linear proposition). This gives us two judgment forms for the mixed linear/nonlinear logic.

(1) $\Gamma_{\mathsf{S}} \,;\, \Delta_{\mathsf{L}} \vdash A_{\mathsf{L}}$

(2) $\Gamma_{\mathsf{S}} \vdash A_{\mathsf{S}}$

The significant differences to the judgments from dual intuitionistic linear logic is that (a) there no longer is an exponential $!A$, and (b) we can apply inference rules directly to structural propositions A_{S}, both in the antecedent and in the succedent.

3 Rules for Implication

Because we have two judgments and also two forms of implication, there are more rules concerning implication than one might at first suspect. This kind of redundancy is unfortunate, but, as we will see in the next lecture, it can be eliminated to obtain a quite streamlined system in which there are just a single right rule and a single left rule for implication.

Because we can tell, by notation and by position, when antecedents are structural or linear propositions, we omit the subscription Γ_{S} and Δ_{L}.

First, the right rules. Since the mode of the succedent is uniquely determined by the judgment (or vice versa, depending on how you look at it), there are just two rules.

$$\frac{\Gamma, A_{\mathsf{S}} \vdash B_{\mathsf{S}}}{\Gamma \vdash A_{\mathsf{S}} \supset B_{\mathsf{S}}}\ \supset R \qquad \frac{\Gamma\ ;\ \Delta, A_{\mathsf{L}} \vdash B_{\mathsf{L}}}{\Gamma\ ;\ \Delta \vdash A_{\mathsf{L}} \multimap B_{\mathsf{L}}}\ \multimap R$$

There is only one left rule for $A_{\mathsf{L}} \multimap B_{\mathsf{L}}$ because it can appear in only one of the two judgments.

$$\frac{\Gamma\ ;\ \Delta_1 \vdash A_{\mathsf{L}} \quad \Gamma\ ;\ \Delta_2, B_{\mathsf{L}} \vdash C_{\mathsf{L}}}{\Gamma\ ;\ \Delta_1, \Delta_2, A_{\mathsf{L}} \multimap B_{\mathsf{L}} \vdash C_{\mathsf{L}}}\ \multimap L$$

On the other hand, structural implication can appear in both judgment forms, so there are two left rules for $A_{\mathsf{S}} \supset B_{\mathsf{S}}$.

$$\frac{\Gamma, A_{\mathsf{S}} \supset B_{\mathsf{S}} \vdash A_{\mathsf{S}} \quad \Gamma, A_{\mathsf{S}} \supset B_{\mathsf{S}}, B_{\mathsf{S}}\ ;\ \Delta \vdash C_{\mathsf{L}}}{\Gamma, A_{\mathsf{S}} \supset B_{\mathsf{S}}\ ;\ \Delta \vdash C_{\mathsf{L}}}\ \supset L_{\mathsf{SL}}$$

$$\frac{\Gamma, A_{\mathsf{S}} \supset B_{\mathsf{S}} \vdash A_{\mathsf{S}} \quad \Gamma, A_{\mathsf{S}} \supset B_{\mathsf{S}}, B_{\mathsf{S}} \vdash C_{\mathsf{S}}}{\Gamma, A_{\mathsf{S}} \supset B_{\mathsf{S}} \vdash C_{\mathsf{S}}}\ \supset L_{\mathsf{SS}}$$

Note that the first premises of the two rules are the same. That's because A_{S} is structural, and thereby determines the judgment form needed to prove it. It is also forced that we sort B_{S} into Γ, just from its mode.

4 Rules for Shifts

Besides identifying the right judgment forms, the rules for the shifts are a key to understanding the mixed linear/nonlinear system. As guidance in this process, let's recall the rules in the dyadic system from last lecture. We show the original form of $!R$ with an explicit, albeit forced, valid_R rule.

$$\frac{\Gamma\ ;\ \cdot \vdash A\ \mathit{true}}{\Gamma \vdash A\ \mathit{valid}}\ \mathsf{valid}_R \qquad \frac{\Gamma, A\ \mathit{valid}\ ;\ \Delta, A\ \mathit{true} \vdash C\ \mathit{true}}{\Gamma, A\ \mathit{valid}\ ;\ \Delta \vdash C\ \mathit{true}}\ \mathsf{valid}_L$$

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

Here is a thought experiment: what if the structural layer was only occupied by $\uparrow A_{\mathsf{L}}$? Then for any proposition $\downarrow A_{\mathsf{S}}$ the proposition A_{S} must have the form $\uparrow A_{\mathsf{L}}$. Then we can try to gain insight from the decomposition $!A_L \triangleq \downarrow\uparrow A_{\mathsf{L}}$. Here, both A_{L} and $\downarrow\uparrow A_{\mathsf{L}}$ are at mode L. Therefore, $A_{\mathsf{S}}\ \mathit{valid}$ corresponds to $\uparrow A_{\mathsf{L}}\ \mathit{true}$ because this is the only possibility for A_{S}. From this we get the following rules (showing the prior

rule on the left, the LNL rule on the right):

$$\frac{\Gamma\ ;\cdot \vdash A\ \mathit{true}}{\Gamma \vdash A\ \mathit{valid}}\ \mathsf{valid}_R \qquad \frac{\Gamma\ ;\cdot \vdash A_{\mathsf{L}}}{\Gamma \vdash \uparrow A_{\mathsf{L}}}\ \uparrow R$$

$$\frac{\Gamma, A\ \mathit{valid}\ ;\ \Delta, A\ \mathit{true} \vdash C\ \mathit{true}}{\Gamma, A\ \mathit{valid}\ ;\ \Delta \vdash C\ \mathit{true}}\ \mathsf{valid}_L \qquad \frac{\Gamma, \uparrow A_{\mathsf{L}}\ ;\ \Delta, A_{\mathsf{L}} \vdash C_{\mathsf{L}}}{\Gamma, \uparrow A_{\mathsf{L}}\ ;\ \Delta \vdash C_{\mathsf{L}}}\ \uparrow L$$

$$\frac{\Gamma \vdash A\ \mathit{valid}}{\Gamma\ ;\cdot \vdash\ !A\ \mathit{true}}\ !R \qquad \frac{\Gamma \vdash A_{\mathsf{S}}}{\Gamma\ ;\cdot \vdash \downarrow A_{\mathsf{S}}}\ \downarrow R$$

$$\frac{\Gamma, A\ \mathit{valid}\ ;\ \Delta \vdash C\ \mathit{true}}{\Gamma\ ;\ \Delta, !A \vdash C\ \mathit{true}}\ !L \qquad \frac{\Gamma, A_{\mathsf{S}}\ ;\ \Delta \vdash C_{\mathsf{L}}}{\Gamma\ ;\ \Delta, \downarrow A_{\mathsf{S}} \vdash C_{\mathsf{L}}}\ \downarrow L$$

Some of the anomalies have disappeared. For example, every rule now concerns a particular logical connective rather than a judgment.

Furthermore, we have an explanation why $!A$ was neither positive nor negative. From writing out the proofs of identities we can see that $\uparrow$ is negative (invertible on the right) and $\downarrow$ is positive (invertible on the left). So $!A_{\mathsf{L}} = \downarrow\uparrow A_{\mathsf{L}}$ is neither positive nor negative because there is a shift in polarity between the two shifts. It is sometimes said that ! is "*positive on the outside and negative on the inside*", which reflects this decomposition precisely.

5 Identity and Cut

In LNL we have two forms of identity, and several forms of cut. This is because we can apply rules directly to structural propositions.

$$\frac{}{\Gamma, A_{\mathsf{S}} \vdash A_{\mathsf{S}}}\ \mathsf{id}_{\mathsf{S}} \qquad \frac{}{\Gamma\ ;\ A_{\mathsf{L}} \vdash A_{\mathsf{L}}}\ \mathsf{id}_{\mathsf{L}}$$

For the same reason we have multiple left rules, we also get multiple versions of cut (three, to be precise).

$$\frac{\Gamma \vdash A_{\mathsf{S}} \quad \Gamma, A_{\mathsf{S}} \vdash C_{\mathsf{S}}}{\Gamma \vdash C_{\mathsf{S}}}\ \mathsf{cut}_{\mathsf{SS}} \qquad \frac{\Gamma \vdash A_{\mathsf{S}} \quad \Gamma, A_{\mathsf{S}}\ ;\ \Delta' \vdash C_{\mathsf{L}}}{\Gamma\ ;\ \Delta' \vdash C_{\mathsf{L}}}\ \mathsf{cut}_{\mathsf{SL}}$$

$$\frac{\Gamma\ ;\ \Delta \vdash A_{\mathsf{L}} \quad \Gamma\ ;\ \Delta', A_{\mathsf{L}} \vdash C_{\mathsf{L}}}{\Gamma\ ;\ \Delta, \Delta' \vdash C_{\mathsf{L}}}\ \mathsf{cut}_{\mathsf{LL}}$$

The admissibility of cut and identity is not essentially different from before, and perhaps proof are even a bit simpler since we do not need to order different forms of cut among each other. But we need to prove it simultaneously for the two forms of identity and three three forms of cut since shifts will move us between these judgments.

Theorem 1 (Admissibility of Identity) *The rules of identity* id_S *and* id_L *are admissible in the system where they are restricted to atomic propositions.*

Proof: By simultaneous structural induction on A_S and A_L. □

Theorem 2 (Admissibility of Cut) *The three rules of cut are admissible in the system without cut.*

Proof: By simultaneous nested induction on all three forms of cut, first on the cut proposition A_L and A_S, and second on the first and second given derivation.

To account for the requirement that the structural context Γ be the same among the premises of the cut we may have to apply the admissibility of weakening for structural antecedents, that is, we can add a new antecedent B_S to every sequent in a proof. □

Once we have cut elimination, the conservative extension properties are almost immediate.

Theorem 3 (Conservative Extension)

(i) If Γ *and* A_S *do not contain any upshift* $\uparrow$ *then* $\Gamma \vdash A_\mathsf{S}$ *if and only if* $\Gamma \vdash A_\mathsf{S}$ *in structural (intuitionistic) logic.*

(ii) If Δ *and* A_L *do not contain any downshift* $\downarrow$ *then* $\cdot \,;\, \Delta \vdash A_\mathsf{L}$ *if and only if* $\Delta \vdash A_\mathsf{L}$ *in purely linear logic.*

Proof: All the rules except cut have the subformula property, and the rules can be read as rules from the purely structural (part (i)) or purely linear (part (ii)) sequent calculus. Therefore conservative extension follows directly from cut elimination. □

6 Examples

We can now experiment with some properties. For example, we see that $\uparrow$ distributes over implication, we just have to be careful to pick the *correct* kind of implication. We elide some structural antecedents when they are no longer needed.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{- \,;\, A_\mathsf{L} \vdash A_\mathsf{L}}\,\mathsf{id} \quad \dfrac{}{- \,;\, B_\mathsf{L} \vdash B_\mathsf{L}}\,\mathsf{id}}{- \,;\, A_\mathsf{L} \multimap B_\mathsf{L}, A_\mathsf{L} \vdash B_\mathsf{L}}\,{\multimap}L}{\uparrow(A_\mathsf{L} \multimap B_\mathsf{L}), \uparrow A_\mathsf{L} \,;\, \cdot \vdash B_\mathsf{L}}\,\uparrow L \times 2}{\uparrow(A_\mathsf{L} \multimap B_\mathsf{L}), \uparrow A_\mathsf{L} \vdash \uparrow B_\mathsf{L}}\,\uparrow R}{\cdot \vdash \uparrow(A_\mathsf{L} \multimap B_\mathsf{L}) \supset (\uparrow A_\mathsf{L} \supset \uparrow B_\mathsf{L})}\,{\supset}R \times 2$$

The downshift also distributes over implication.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{-, A_\mathsf{s} \vdash A_\mathsf{s}}\,\mathsf{id}_\mathsf{s} \quad \dfrac{}{-, B_\mathsf{s} \vdash B_\mathsf{s}}\,\mathsf{id}_\mathsf{s}}{A_\mathsf{s} \supset B_\mathsf{s}, A_\mathsf{s} \vdash B_\mathsf{s}}\,{\supset}L}{A_\mathsf{s} \supset B_\mathsf{s}, A_\mathsf{s} \,;\, \cdot \vdash {\downarrow}B_\mathsf{s}}\,{\downarrow}R}{\cdot \,;\, {\downarrow}(A_\mathsf{s} \supset B_\mathsf{s}), {\downarrow}A_\mathsf{s} \vdash {\downarrow}B_\mathsf{s}}\,{\downarrow}L \times 2}{\cdot \,;\, \cdot \vdash {\downarrow}(A_\mathsf{s} \supset B_\mathsf{s}) \multimap ({\downarrow}A_\mathsf{s} \multimap {\downarrow}B_\mathsf{s})}\,{\supset}R \times 2$$

The properties we proved last time using $!A$ carry over under the definition as $\downarrow\uparrow A$. The proofs remain essentially the same, with just a couple of additional administrative steps and different naming of the rules.

7 A Programming Example

One advantage of LNL is more direct expression of mixed programs. As an example that also illustrates parallelism, we show *mapreduce* over trees that have data only at the leaves. *mapreduce* is a fold operation over such trees, so the type tree_A is replaced by a fresh type variable B.

$$\mathsf{tree}_A = \oplus\{\mathsf{leaf} : A, \mathsf{node} : \mathsf{tree}_A \otimes \mathsf{tree}_A\}$$
$$\mathit{mapreduce}_{A\,B}\,(r : B)\;(h : A \multimap B)\;(f : B \multimap (B \multimap B))\;(t : \mathsf{tree}_A) = \ldots \;?$$

In the type of *mapreduce*, r stands for the result, that is, the channel to deliver the result at. We have curried the type of the function f for ease of programming.

A problem is that we will not be able to write a recursive *mapreduce* with this type since h and f are channels of linear type. h will be used for every leaf, and f will be used for every node, so both will be used multiple times. We could try to rewrite *mapreduce* as an iterator—here we want to write it directly. This means that both h and f should be structural: we need upshifts.

$$\mathit{mapreduce}_{A\,B}\,(r : B)\;(h : {\uparrow}(A \multimap B))\;(f : {\uparrow}(B \multimap (B \multimap B)))\;(t : \mathsf{tree}_A) = \ldots$$

At this point it would be relatively straightforward to write the code, if we only had the *dynamics* of the shifts. Let's develop this and then come back to the example.

8 Dynamics of the Shifts

We start with $\uparrow A$. Since it is negative it will receive, but what? Let's look at the rule and annotate with channels.

$$\dfrac{\Gamma \,;\, \cdot \vdash A_\mathsf{L}}{\Gamma \vdash {\uparrow}A_\mathsf{L}}\,{\uparrow}R \qquad \dfrac{\Gamma \,;\, \cdot \vdash (y_\mathsf{L} : A_\mathsf{L})}{\Gamma \vdash (x_\mathsf{S} : {\uparrow}A_\mathsf{L})}\,{\uparrow}R$$

We need to transition from a structural channel x_S to a linear channel y_L, so that's what we need to receive! We write $\langle y \rangle$ for receiving a channel of a different mode (linear or structural). Then we have:

$$\frac{\Gamma \,;\, \cdot \vdash P(y_\mathsf{L}) :: (y_\mathsf{L} : A_\mathsf{L})}{\Gamma \vdash \mathbf{recv}\ x_\mathsf{S}\ (\langle y_\mathsf{L} \rangle \Rightarrow P(y_\mathsf{L})) :: (x_\mathsf{S} : \uparrow A_\mathsf{L})}\ \uparrow R$$

Next, the left rule $\uparrow L$. To match $\uparrow R$, we clearly need to send a channel but which one? We annotate the rule and then think of a process notation.

$$\frac{\Gamma, \uparrow A_\mathsf{L} \,;\, \Delta, A_\mathsf{L} \vdash C_\mathsf{L}}{\Gamma, \uparrow A_\mathsf{L} \,;\, \Delta \vdash C_\mathsf{L}}\ \uparrow L \qquad \frac{\Gamma, x_\mathsf{S} : \uparrow A_\mathsf{L} \,;\, \Delta, y_\mathsf{L} : A_\mathsf{L} \vdash (z : C_\mathsf{L})}{\Gamma, x_\mathsf{S} : \uparrow A_\mathsf{L} \,;\, \Delta \vdash (z : C_\mathsf{L})}\ \uparrow L$$

We see that what we have to send is a fresh linear channel y_L. This is a new phenomenon since there is no cut involved, and so far only cut would create a fresh channel. So we have to make up some new form of syntax.

$$\frac{\Gamma, x_\mathsf{S} : \uparrow A_\mathsf{L} \,;\, \Delta, y_\mathsf{L} : A_\mathsf{L} \vdash Q(y_\mathsf{L}) :: (z : C_\mathsf{L})}{\Gamma, x_\mathsf{S} : \uparrow A_\mathsf{L} \,;\, \Delta \vdash \mathbf{send}\ x_\mathsf{S}\ (\langle y_\mathsf{L} \rangle \Rightarrow Q(y_\mathsf{L})) :: (z : C_\mathsf{L})}\ \uparrow L$$

What happens operationally? The client sends a fresh channel and the provider continues with the fresh channel. So the first approximation would be

$$\begin{array}{lll} & \mathsf{proc}(\mathbf{recv}\ a_\mathsf{S}\ (\langle y_\mathsf{L} \rangle \Rightarrow P(y_\mathsf{L}))), & \mathsf{proc}(\mathbf{send}\ a_\mathsf{S}\ (\langle y_\mathsf{L} \rangle \Rightarrow Q(y_\mathsf{L}))) \\ \longrightarrow & \mathsf{proc}(P(b_\mathsf{L})), & \mathsf{proc}(Q(b_\mathsf{L})) \qquad b_\mathsf{L}\ \text{fresh} \end{array}$$

This does not quite work, however, since structural channels may have multiple clients. In the rule above the provider of a_S disappears after interacting with one client, so the other clients will be left dangling without a provider.

A similar practically occurring scenario is a web server. When a client sends an HTTP request, the server spawns a fresh one-to-one connection and meanwhile continues to serve other requests. What this means here is that the provider of a structural channel spawns a fresh linear process *but remains in the configuration* to receive further requests.

We model this with a new feature of multiset rewriting: we combine linear and structural inference. The state consists of a set (the persistent propositions) and a multiset (the ephemeral propositions). We just *underline* the persistent proposition, because the alternative notation $!A$ may be misleading. Then the rule reads:

$$\begin{array}{lll} & \underline{\mathsf{proc}}(\mathbf{recv}\ a_\mathsf{S}\ (\langle y_\mathsf{L} \rangle \Rightarrow P(y_\mathsf{L}))), & \mathsf{proc}(\mathbf{send}\ a_\mathsf{S}\ (\langle y_\mathsf{L} \rangle \Rightarrow Q(y_\mathsf{L}))) \\ \longrightarrow & \mathsf{proc}(P(b_\mathsf{L})), & \mathsf{proc}(Q(b_\mathsf{L})) \qquad b_\mathsf{L}\ \text{fresh} \end{array}$$

The persistent semantic objects originate in the $\mathsf{cut}_{\mathsf{SL}}$ rule (we omit the $\mathsf{cut}_{\mathsf{SS}}$ rule). With process terms:

$$\frac{\Gamma \vdash P(x_\mathsf{S}) :: (x_\mathsf{S} : A_\mathsf{S}) \quad \Gamma, x_\mathsf{S} : A_\mathsf{S} \,;\, \Delta' \vdash Q(x_\mathsf{S}) :: (z_\mathsf{L} : C_\mathsf{L})}{\Gamma \,;\, \Delta' \vdash x_\mathsf{S} \leftarrow P(x_\mathsf{S}) \,;\, Q(x_\mathsf{S}) :: (z_\mathsf{L} : C_\mathsf{L})}\ \mathsf{cut}_{\mathsf{SL}}$$

And the dynamics:

$$\mathsf{proc}(x_\mathsf{s} \leftarrow P(x_\mathsf{s}) \,;\, Q(x_\mathsf{s})) \quad \longrightarrow \quad \underline{\mathsf{proc}}(P(a_\mathsf{s})), \mathsf{proc}(Q_\mathsf{s}(a_\mathsf{s})) \quad (a_\mathsf{s} \text{ fresh})$$

Because it is not needed for the example, for the downshifts we jump directly to the annotated versions and dynamics. As with other symmetric pairs of connectives ($\multimap$ / $\otimes$ and $\&$ / $\oplus$) we'd like to reuse the syntax for a streamlined language, just swapping provider and client roles.

$$\frac{\Gamma \vdash P(y_\mathsf{s}) :: (y_\mathsf{s} : A_\mathsf{s})}{\Gamma \,;\, \cdot \vdash \mathbf{send}\ x_\mathsf{L}\ (\langle y_\mathsf{s}\rangle \Rightarrow P(y_\mathsf{s})) :: (x_\mathsf{L} : {\downarrow}A_\mathsf{s})}\ {\downarrow}R$$

$$\frac{\Gamma, y_\mathsf{s} : A_\mathsf{s} \,;\, \Delta \vdash Q(y_\mathsf{s}) :: (z_\mathsf{L} :: C_\mathsf{L})}{\Gamma \,;\, \Delta, x_\mathsf{L} : {\downarrow}A_\mathsf{s} \vdash \mathbf{recv}\ x_\mathsf{L}\ (\langle y_\mathsf{s}\rangle \Rightarrow P(y_\mathsf{s})) :: (z_\mathsf{L} : C_\mathsf{L})}\ {\downarrow}L$$

The dynamics is similar to the one for ${\uparrow}A_\mathsf{L}$, except that different processes are persistent and channels go from linear to structural instead of vice versa.

$$\begin{array}{lll} & \mathsf{proc}(\mathbf{send}\ a_\mathsf{L}\ (\langle y_\mathsf{s}\rangle \Rightarrow P(y_\mathsf{s}))), & \mathsf{proc}(\mathbf{recv}\ a_\mathsf{L}\ (\langle y_\mathsf{s}\rangle \Rightarrow Q(y_\mathsf{s}))) \\ \longrightarrow & \underline{\mathsf{proc}}(P(b_\mathsf{s})), & \mathsf{proc}(Q(b_\mathsf{s})) \qquad\qquad b_\mathsf{s} \text{ fresh} \end{array}$$

9 Example Continued

Returning to the example, we can now write the process for *mapreduce*. Both h and f are structural channels; we omit the subscript on the linear channels.

$$\begin{array}{l} \mathsf{tree}_A = \oplus\{\mathsf{leaf} : A, \mathsf{node} : \mathsf{tree}_A \otimes \mathsf{tree}_A\} \\ \mathit{mapreduce}_{AB}\ (r : B)\ (h_\mathsf{s} : {\uparrow}(A \multimap B))\ (f_\mathsf{s} : {\uparrow}(B \multimap (B \multimap B)))\ (t : \mathsf{tree}_A) = \\ \quad \mathbf{recv}\ t\ (\ \mathsf{leaf} \Rightarrow \mathbf{send}\ h_\mathsf{s}\ (\langle h'\rangle \Rightarrow \mathbf{send}\ h'\ t\ ;\ \mathbf{fwd}\ r\ h') \\ \qquad\quad \mid \mathsf{node} \Rightarrow \mathbf{recv}\ t\ (s \Rightarrow \\ \qquad\qquad\qquad x \leftarrow \mathbf{call}\ \mathit{mapreduce}_{AB}\ x\ h_\mathsf{s}\ f_\mathsf{s}\ s\ ; \\ \qquad\qquad\qquad y \leftarrow \mathbf{call}\ \mathit{mapreduce}_{AB}\ y\ h_\mathsf{s}\ f_\mathsf{s}\ t\ ; \\ \qquad\qquad\qquad \mathbf{send}\ f_\mathsf{s}\ (\langle f'\rangle \Rightarrow \\ \qquad\qquad\qquad \mathbf{send}\ f'\ x\ ; \\ \qquad\qquad\qquad \mathbf{send}\ f'\ y\ ; \\ \qquad\qquad\qquad \mathbf{fwd}\ r\ f')\,)\,) \end{array}$$

The structural nature of h_s and f_s is crucial here because they are indeed used multiple or zero times in the two branches.

This may also be a good time to think about parallelism. We see that the two recursive calls to *mapreduce* proceed entirely independently. But the parallelism even goes further: the computation of f' on x and y can proceed while *mapreduce* is still computing. So x and y are shared between the providers (the recursive calls

to $mapreduce$) and the client (f', a linear instance of f_s), synchronization between these two processes only takes place during input or output on those channels. This phenomenon of *pipelining* can improve the asymptotic complexity of some parallel algorithms, as observed by Blelloch and Reid-Miller [1999].

Since it came up during lecture: if we change the type of f_s to take a pair of channels, we have to create this pair in the code which is minimally more complicated. The result is below.

$$
\begin{array}{l}
\mathsf{tree}_A = \oplus\{\mathsf{leaf} : A, \mathsf{node} : \mathsf{tree}_A \otimes \mathsf{tree}_A\} \\
mapreduce_{AB}\,(r : B)\;(h_\mathsf{s} : \uparrow(A \multimap B))\;(f_\mathsf{s} : \uparrow((B \otimes B) \multimap B))\;(t : \mathsf{tree}_A) = \\
\quad \mathbf{recv}\ t\ (\,\mathsf{leaf} \Rightarrow \mathbf{send}\ h_\mathsf{s}\ (\langle h' \rangle \Rightarrow \mathbf{send}\ h'\ t\ ;\ \mathbf{fwd}\ r\ h') \\
\qquad\quad \mid \mathsf{node} \Rightarrow \mathbf{recv}\ t\ (s \Rightarrow \\
\qquad\qquad\qquad x \leftarrow \mathbf{call}\ mapreduce_{AB}\ x\ h_\mathsf{s}\ f_\mathsf{s}\ s\ ; \\
\qquad\qquad\qquad y \leftarrow \mathbf{call}\ mapreduce_{AB}\ y\ h_\mathsf{s}\ f_\mathsf{s}\ t\ ; \\
\qquad\qquad\qquad \mathbf{send}\ f_\mathsf{s}\ (\langle f' \rangle \Rightarrow \\
\qquad\qquad\qquad p_{B\otimes B} \leftarrow (\mathbf{send}\ p\ x\ ;\ \mathbf{fwd}\ p\ y)\ ; \\
\qquad\qquad\qquad \mathbf{send}\ f'\ p\ ; \\
\qquad\qquad\qquad \mathbf{fwd}\ r\ f')\,)\,)
\end{array}
$$

10 Summary

We have presented LNL [Benton, 1994], a mixed linear/nonlinear logic that arises from the logic of validity from the last lecture by decomposing $!A$ into two shifts: one ($\uparrow$) from linear to structural and one ($\downarrow$) from structural to linear. Benton makes the point that there is an adjunction between the two shifts, which results in their composition $\downarrow\uparrow$ being a comonad and the opposite composition $\uparrow\downarrow$ being a monad.

This approach solves some small issues with the dyadic formulation from last lecture. For example, $\uparrow$ is negative and $\downarrow$ is positive, which means their composition $!$ is neither.

Since we view LNL mainly as a stepping stone to full adjoint logic[1] we do not summarize the rules here.

References

Nick Benton. A mixed linear and non-linear logic: Proofs, terms and models. In Leszek Pacholski and Jerzy Tiuryn, editors, *Selected Papers from the 8th International Workshop on Computer Science Logic (CSL'94)*, pages 121–135, Kazimierz, Poland, September 1994. Springer LNCS 933. An extended version appears as Technical Report UCAM-CL-TR-352, University of Cambridge.

[1] briefly previewed in this lecture, but covered in the notes for the next lecture

G. E. Blelloch and M. Reid-Miller. Pipeling with futures. *Theory of Computing Systems*, 32:213–239, 1999.

Klaas Pruiksma, William Chargin, Frank Pfenning, and Jason Reed. Adjoint logic. Unpublished manuscript, April 2018. URL http://www.cs.cmu.edu/~fp/papers/adjoint18b.pdf.

Jason Reed. A judgmental deconstruction of modal logic. Unpublished manuscript, May 2009. URL http://www.cs.cmu.edu/~jcreed/papers/jdml2.pdf.