Lecture Notes on Of Course!

15-816: Substructural Logics
Frank Pfenning

Lecture 15
October 20, 2016

In this lecture we combine structural and substructural intuitionistic logics into a single system, using the previously discussed device of shift modalities to go between layers. This is the idea behind Benton's LNL [Ben94], who basis his constructions on the categorical concept of an adjunction.

As we have already seen, the idea is quite general. In this lecture we will restrict ourselves to the structural and linear fragments, leaving other considerations such as ordered, affine, or strict logics to a future lecture in order to reduce it to its essentials. From our approach the exponential modality $!A$ (read: *of course A!*) of Girard's linear logic [Gir87] will arise naturally as a composition of two shifts.

We will also resume our analysis of the operational interpretation of shifts, to see specifically what happens in the context of LNL.

1 Combining Linear and Structural Logic

Our starting point quite straightforwardly follows the approach laid out in Lecture 12. We have two separate layers of propositions, connected by two shifts. We use U (suggesting *unrestricted*) for the structural mode and L, as before, for the linear mode. We have, by definition $\mathsf{U} > \mathsf{L}$ since U admits exchange, weakening, and contraction while L admits only exchange.

$$\begin{array}{llcl} \text{Structural} & A_{\mathsf{U}} & ::= & \ldots \mid A_{\mathsf{U}} \rightarrow B_{\mathsf{U}} \mid {\uparrow}_{\mathsf{L}}^{\mathsf{U}} A_{\mathsf{L}} \\ \text{Linear} & A_{\mathsf{L}} & ::= & \ldots \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid {\downarrow}_{\mathsf{L}}^{\mathsf{U}} A_{\mathsf{U}} \end{array}$$

The independence principle states:

A structural succedent may not depend on a linear antecedent.

We can make this explicit by allowing only the following two judgment forms:

$$\Gamma_{\mathsf{U}} \vdash A_{\mathsf{U}}$$
$$\Gamma_{\mathsf{U}} \; ; \; \Delta_{\mathsf{L}} \vdash A_{\mathsf{L}}$$

It is possible to separate the antecedents into two zones since both modes admit exchange. From now on we will use Γ only for structural antecdents and Δ only for linear ones, so we can omit the subscript.

To begin with, we obtain the following rules of identity and cut. The phenomenon of obtain three cut rules should be familiar from Lecture 12. As we pointed out there, they can be unified into a single rule using *adjoint logic* [Ree09].

$$\frac{}{\Gamma, A_{\mathsf{U}} \vdash A_{\mathsf{U}}}\ \mathsf{id_U} \qquad \frac{}{\Gamma \; ; \; A_{\mathsf{L}} \vdash A_{\mathsf{L}}}\ \mathsf{id_L}$$

$$\frac{\Gamma \vdash A_{\mathsf{U}} \quad \Gamma, A_{\mathsf{U}} \vdash C_{\mathsf{U}}}{\Gamma \vdash C_{\mathsf{U}}}\ \mathsf{cut_{UU}} \qquad \frac{\Gamma \vdash A_{\mathsf{U}} \quad \Gamma, A_{\mathsf{U}} \; ; \; \Delta \vdash C_{\mathsf{L}}}{\Gamma \; ; \; \Delta \vdash C_{\mathsf{L}}}\ \mathsf{cut_{UL}}$$

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

As pointed out in the last lecture, the presence of weakening and contraction allows us to view structural antecedents as persistent, so they are propagated to all premises of each inference rule.

The next question are the rules for the shift modalities. They follow exactly the same pattern as before, with a small twist to incorporate the persistence of structural assumptions in the $\uparrow L$ rule. They are entirely based on the independence principle, which is built into the formulation of the judgments themselves.

$$\frac{\Gamma \; ; \; \cdot \vdash A_{\mathsf{L}}}{\Gamma \vdash {\uparrow}_{\mathsf{L}}^{\mathsf{U}} A_{\mathsf{L}}}\ {\uparrow}R \qquad \frac{\Gamma, {\uparrow}_{\mathsf{L}}^{\mathsf{U}} A_{\mathsf{L}} \; ; \; \Delta, A_{\mathsf{L}} \vdash C_{\mathsf{L}}}{\Gamma, {\uparrow}_{\mathsf{L}}^{\mathsf{U}} A_{\mathsf{L}} \; ; \; \Delta \vdash C_{\mathsf{L}}}\ {\uparrow}L$$

$$\frac{\Gamma \vdash A_{\mathsf{U}}}{\Gamma \; ; \; \cdot \vdash {\downarrow}_{\mathsf{L}}^{\mathsf{U}} A_{\mathsf{U}}}\ {\downarrow}R \qquad \frac{\Gamma, A_{\mathsf{U}} \; ; \; \Delta \vdash C_{\mathsf{L}}}{\Gamma \; ; \; \Delta, {\downarrow}_{\mathsf{L}}^{\mathsf{U}} A_{\mathsf{U}} \vdash C_{\mathsf{L}}}\ {\downarrow}L$$

The resulting system enjoys all the important properties such as admissibility of cut and identity and cut elimination. At the interface between the two judgment there is one instance of a cross-cut when $\uparrow R$ is matched against $\uparrow L$. We dispense with the details since there are no particular new ideas to be conveyed.

2 Operational Interpretation of Shifts, Revisited

Before we dive into the operational interpretation of the full structural component of this combined logic, we look only at the shifts. The $\uparrow R$ and $\downarrow L$ rules are invertible, that is, they can always be applied when the proposition $\uparrow A_{\mathsf{L}}$ appears on the right or $\downarrow A_{\mathsf{U}}$ appears on the left. This means these two rules will receive while $\uparrow L$ and $\downarrow R$ will send. What they send and receive is an indication to shift to a new mode of communication. Before, when we shifted, we re-used the same channel. This is no longer possible now because in the $\uparrow L$ rule the persistent channel remains. First, the proof term assignment.

$$\frac{\Gamma \,;\, \cdot \vdash P_y :: (y_{\mathsf{L}} : A_{\mathsf{L}})}{\Gamma \vdash y_{\mathsf{L}} \leftarrow (\mathsf{L})x_{\mathsf{U}} \,;\, P_y :: (x_{\mathsf{U}} : {\uparrow}^{\mathsf{U}}_{\mathsf{L}} A_{\mathsf{L}})}\ {\uparrow}R$$

$$\frac{\Gamma, (x_{\mathsf{U}} : {\uparrow}^{\mathsf{U}}_{\mathsf{L}} A_{\mathsf{L}}) \,;\, \Delta, (y_{\mathsf{L}} : A_{\mathsf{L}}) \vdash Q_y :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma, (x_{\mathsf{U}} : {\uparrow}^{\mathsf{U}}_{\mathsf{L}} A_{\mathsf{L}}) \,;\, \Delta \vdash y_{\mathsf{L}} \leftarrow (\mathsf{L})x_{\mathsf{U}} \,;\, Q_y :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ {\uparrow}L$$

In the synchronous communication rule we mark a process offering along a persistent channel itself as persistent. This is because this process may have multiple clients (inclding in Q_y!), so we cannot evolve it, but we can spawn a fresh linear copy offering along the new channel w_{L}.

$$\frac{\underline{\mathsf{proc}}(x_{\mathsf{U}}, y_{\mathsf{L}} \leftarrow (\mathsf{L})x_{\mathsf{U}} \,;\, P_y) \quad \mathsf{proc}(z_{\mathsf{L}}, y_{\mathsf{L}} \leftarrow (\mathsf{L})x_{\mathsf{U}} \,;\, Q_y)}{\mathsf{proc}(w_{\mathsf{L}}, P_w) \quad \mathsf{proc}(z_{\mathsf{L}}, Q_w)}\ {\uparrow}C^w$$

It is even more obvious in the synchronous version that the persistent process must receive, otherwise it could continuously spawn new messages!

$$\frac{\mathsf{proc}(z_{\mathsf{L}}, y_{\mathsf{L}} \leftarrow (\mathsf{L})x_{\mathsf{U}} \,;\, Q_y)}{\mathsf{msg}(w_{\mathsf{L}}, y_{\mathsf{L}} \leftarrow (\mathsf{L})x_{\mathsf{U}} \,;\, w_{\mathsf{L}} \leftarrow y_{\mathsf{L}}) \quad \mathsf{proc}(z_{\mathsf{L}}, Q_w)}\ {\uparrow}C^w\_\mathsf{send}$$

$$\frac{\underline{\mathsf{proc}}(x_{\mathsf{U}}, y_{\mathsf{L}} \leftarrow (\mathsf{L})x_{\mathsf{U}} \,;\, P_y) \quad \mathsf{msg}(w_{\mathsf{L}}, y_{\mathsf{L}} \leftarrow (\mathsf{L})x_{\mathsf{U}} \,;\, w_{\mathsf{L}} \leftarrow y_{\mathsf{L}})}{\mathsf{proc}(w_{\mathsf{L}}, P_w)}\ {\uparrow}C^w\_\mathsf{recv}$$

The rules for ${\downarrow}^{\mathsf{U}}_{\mathsf{L}} A$ reverse the roles.

$$\frac{\Gamma \vdash P_y :: (y_{\mathsf{U}} : A_{\mathsf{U}})}{\Gamma \,;\, \cdot \vdash y_{\mathsf{U}} \leftarrow (\mathsf{U})x_{\mathsf{L}} \,;\, P_y :: (x_{\mathsf{L}} : {\downarrow}^{\mathsf{U}}_{\mathsf{L}} A_{\mathsf{U}})}\ {\downarrow}R$$

$$\frac{\Gamma, (y_{\mathsf{U}} : A_{\mathsf{U}}) \,;\, \Delta \vdash Q_y :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma \,;\, \Delta, (x_{\mathsf{L}} : {\downarrow}^{\mathsf{U}}_{\mathsf{L}} A_{\mathsf{U}}) \vdash y_{\mathsf{U}} \leftarrow (\mathsf{U})x_{\mathsf{L}} \,;\, Q_y :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ {\downarrow}L$$

In the operational semantics we *create* a persistent process.

$$\frac{\mathsf{proc}(x_{\mathsf{L}}, y_{\mathsf{U}} \leftarrow (\mathsf{U})x_{\mathsf{L}}\ ;\ P_y) \quad \mathsf{proc}(z_{\mathsf{L}}, y_{\mathsf{U}} \leftarrow (\mathsf{U})x_{\mathsf{L}}\ ;\ Q_y)}{\underline{\mathsf{proc}}(w_{\mathsf{U}}, P_w) \quad \mathsf{proc}(z_{\mathsf{L}}, Q_w)}\ {\downarrow}C^w$$

Here, the first process sends the persistent channel, as we can see from the asynchronous version of the semantics.

$$\frac{\mathsf{proc}(x_{\mathsf{L}}, y_{\mathsf{U}} \leftarrow (\mathsf{U})x_{\mathsf{L}}\ ;\ P_y) \quad \mathsf{proc}(z_{\mathsf{L}}, y_{\mathsf{U}} \leftarrow (\mathsf{U})x_{\mathsf{L}}\ ;\ Q_y)}{\underline{\mathsf{proc}}(w_{\mathsf{U}}, P_w) \quad \mathsf{msg}(x_{\mathsf{L}}, y_{\mathsf{U}} \leftarrow (\mathsf{U})x_{\mathsf{L}}\ ;\ y_{\mathsf{U}} \leftarrow w_{\mathsf{U}})}\ {\downarrow}C^w\_\mathsf{send}$$

$$\frac{\mathsf{msg}(x_{\mathsf{L}}, y_{\mathsf{U}} \leftarrow (\mathsf{U})x_{\mathsf{L}}\ ;\ y_{\mathsf{U}} \leftarrow w_{\mathsf{U}}) \quad \mathsf{proc}(z_{\mathsf{L}}, y_{\mathsf{U}} \leftarrow (\mathsf{U})x_{\mathsf{L}}\ ;\ Q_y)}{\mathsf{proc}(z_{\mathsf{L}}, Q_w)}\ {\downarrow}C^w\_\mathsf{recv}$$

3 Example: Map

As we have seen in Exercise 10.2 of Lecture 10, it is possible to define concurrent versions of map and fold using recursive types. Another natural approach is to allow the transformer that is mapped over a list to be persistent. We abbreviate ${\uparrow}_{\mathsf{L}}^{\mathsf{U}}A$ as ${\uparrow}A$ and similarly for ${\downarrow}_{\mathsf{L}}^{\mathsf{U}}A$ since in this lecture we only consider structural and linear modes. We leave linear channels undecorated and write x_{U} for unrestricted channels which can be used arbitrarily often.

$\mathsf{list}_A = \oplus\{\mathsf{cons} : A \otimes \mathsf{list}_A, \mathsf{nil} : \mathbf{1}\}$
$f_{\mathsf{U}} : {\uparrow}(A \multimap B)\ ;\ k{:}\mathsf{list}_A \vdash \mathit{map} :: (l : \mathsf{list}_B)$

```
$l \leftarrow \mathit{map} \leftarrow k\ f_{\mathsf{U}} =$
  $\mathsf{case}\ k\ \ (\mathsf{nil} \Rightarrow \mathsf{wait}\ k\ ;\ l.\mathsf{nil}\ ;\ \mathsf{close}\ l$
            $\mid \mathsf{cons} \Rightarrow \ldots)$
```

We already took advantage here of weakening: f_{U} is not used in the nil branch of *map*. We will need it twice in the cons branch: once to apply to the element, and then pass it on to be mapped over the rest of the list.

$\mathsf{list}_A = \oplus\{\mathsf{cons} : A \otimes \mathsf{list}_A, \mathsf{nil} : \mathbf{1}\}$
$f_{\mathsf{U}} : {\uparrow}(A \multimap B)\ ;\ k{:}\mathsf{list}_A \vdash \mathit{map} :: (l : \mathsf{list}_B)$

```
$l \leftarrow \mathit{map} \leftarrow k\ f_{\mathsf{U}} =$
   case $k$  (nil $\Rightarrow$ wait $k$ ; $l$.nil ; close $l$
            | cons $\Rightarrow$ $x \leftarrow$ recv $k$ ;
                      $y \leftarrow (\mathsf{L}) f_{\mathsf{U}}$ ;               % $y{:}A \multimap B$
                      send $y\ x$ ;                  % $y{:}B$
                      $l$.cons ; send $l\ y$ ;
                      $l \leftarrow \mathit{map} \leftarrow k\ f_{\mathsf{U}}$)
```

Below is a slight different style of expressing this computation proposed in lecture. It uses library processes for *nil* and *cons*. Here, the call to *map* is not a tail call. This means, without any further optimizations, it will spawn a new process and therefore it is likely less efficient. We don't particularly care at this point about low-level efficiency or even how many processes are spawned, but it is still worth noting this difference.

$\mathsf{list}_A = \oplus\{\mathsf{cons} : A \otimes \mathsf{list}_A, \mathsf{nil} : \mathbf{1}\}$
$\cdot \vdash \mathit{nil} :: (l : \mathsf{list}_A)$
$x{:}A, k{:}\mathsf{list}_A \vdash \mathit{cons} :: (l : \mathsf{list}_A)$

$f_{\mathsf{U}} : {\uparrow}(A \multimap B)\ ;\ k{:}\mathsf{list}_A \vdash \mathit{map} :: (l : \mathsf{list}_B)$

```
$l \leftarrow \mathit{map} \leftarrow k\ f_{\mathsf{U}} =$
   case $k$  (nil $\Rightarrow$ wait $k$ ; $l \leftarrow \mathit{nil}$
            | cons $\Rightarrow$ $x \leftarrow$ recv $k$ ;
                      $y \leftarrow (\mathsf{L}) f_{\mathsf{U}}$ ;               % $y{:}A \multimap B$
                      send $y\ x$ ;                  % $y{:}B$
                      $l' \leftarrow \mathit{map} \leftarrow k\ f_{\mathsf{U}}$ ;
                      $l \leftarrow \mathit{cons} \leftarrow l'\ y$)
```

As a sample mapping function we write one that turns an element into a singleton list.

```
$x{:}A \vdash \mathit{singleton} :: (l{:}\mathsf{list}_A)$
$l \leftarrow \mathit{singleton} \leftarrow x =$
   $n \leftarrow \mathit{nil}$ ;
   $l \leftarrow \mathit{cons} \leftarrow x\ n$
```

$\cdot \vdash \mathit{map\_singleton} :: (f_u : {\uparrow}(A \multimap \mathsf{list}_A))$

```
$f_u \leftarrow \mathit{map\_singleton} =$
   $y \leftarrow (\mathsf{L}) f_u$ ;
```

```
$x \leftarrow$ recv $y$ ;
$y \leftarrow$ singleton $\leftarrow x$
```

4 Of Course!

In Girard's linear logic [Gir87] there is no explicit structural layer, but we have a so-called exponential modality $!A$ (pronounced "*of course* A" or sometimes "*bang* A" to allow an A to be used multiple times. Briefly, Girard started from the idea that the intuitionistic function space $A \to B$ could be decomposed into a modality and a linear function space $(!A) \multimap B$. This idea of a fine-grained analysis of computation while retaining the means to express all the prior functions also pervades this course.

We arrived at this idea following a different path. Rather than decomposing existing languages, we have started from a substructural point of view and added various liberties. Starting from this direction we notice that $!A \simeq {\downarrow}^{\mathsf{U}}_{\mathsf{L}}{\uparrow}^{\mathsf{U}}_{\mathsf{L}}A$L. In the (intuitionistic) version of Girard's logic, we have the following inference rules pertaining to the exponential modality.

$$\frac{\Delta \vdash C}{\Delta, !A \vdash C}\ \mathsf{weaken} \qquad \frac{\Delta, !A, !A \vdash C}{\Delta, !A \vdash C}\ \mathsf{contract}$$

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

Here we use $!\Delta$ is stand for a collection of antecedents all of whose propositions have the form $!A$. While there are many interesting semantic approaches to understand this and its classical counterpart, the proof theory and the cut elimination proofs are not nearly as elegant as for the substructural approach we have followed here. Some notes on prior work can be found in [Pfe94, CCP03].

Using the expansion of $!A_{\mathsf{L}} = {\downarrow}{\uparrow}A_{\mathsf{L}}$ we can validate all of the rules above by showing that they are admissible. For example:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{{\uparrow}A \vdash {\uparrow}A}\ \mathsf{id_U}}{{\uparrow}A \,;\, \cdot \vdash !A}\ {\downarrow}R \quad \dfrac{\dfrac{}{{\uparrow}A \vdash {\uparrow}A}\ \mathsf{id_U}}{{\uparrow}A \,;\, \cdot \vdash !A}\ {\downarrow}R}{{\uparrow}A \,;\, \cdot \vdash !A \otimes !A}\ \otimes R}{\cdot \,;\, !A \vdash !A \otimes !A}\ {\downarrow}L \quad \dfrac{\cdot \,;\, \Delta, !A, !A \vdash C}{\cdot \,;\, \Delta, !A \otimes !A \vdash C}\ \otimes L}{\Delta, !A \vdash C}\ \mathsf{cut}$$

Because of several standard embeddings of structural intuitionistic logic in linear logic [TCP12], this means that in a certain sense we do not need the full structural layer in what we have discussed. It is sufficient to just have

$$\begin{array}{llll} \text{Structural} & A_{\mathsf{U}} & ::= & {\uparrow}_{\mathsf{L}}^{\mathsf{U}} A_{\mathsf{L}} \\ \text{Linear} & A_{\mathsf{L}} & ::= & \ldots \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid {\downarrow}_{\mathsf{L}}^{\mathsf{U}} A_{\mathsf{U}} \end{array}$$

and the rest is a question of *pragmatics*: how easy or difficult is it to program certain algorithms in the resulting language as compared to coding them when the structural layer is more complete.

Exercises

Exercise 1 Implement the operation of fold from Exercise 10.3, but with fold a persistent process analogous to map in Section 3.

Exercise 2 Show that all the rules for intuitionistic linear logic using $!A$ are derivable or admissible in the combined structural/linear logic using the definition of $!A$ as ${\downarrow}^{\mathsf{U}}_{\mathsf{L}}{\uparrow}^{\mathsf{U}}_{\mathsf{L}}A_{\mathsf{L}}$. Give a derivation of the rule where possible.

Exercise 3 Prove that if $\vdash A_{\mathsf{L}}$ and A_{L} uses ${\downarrow}^{\mathsf{U}}_{\mathsf{L}}{\uparrow}^{\mathsf{U}}_{\mathsf{L}}$ as its only modality, then it is provable in linear logic under the rules in Section 4.

References

[Ben94] 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 (CLS'94)*, Kazimierz, Poland, September 1994. Springer LNCS 933. An extended version appears as Technical Report UCAM-CL-TR-352, University of Cambridge.

[CCP03] Bor-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.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Pfe94] Frank Pfenning. Structural cut elimination in linear logic. Technical Report CMU-CS-94-222, Department of Computer Science, Carnegie Mellon University, December 1994.

[Ree09] Jason Reed. A judgmental deconstruction of modal logic. Unpublished manuscript, 2009.

[TCP12] Bernardo Toninho, Luís Caires, and Frank Pfenning. Functions as session-typed processes. In L. Birkedal, editor, *15th International Conference on Foundations of Software Science and Computation Structures*, FoSSaCS'12, pages 346–360, Tallinn, Estonia, March 2012. Springer LNCS.