Lecture Notes on
Substructural Frameworks

15-836: Substructural Logics
Frank Pfenning

Lecture 21
November 30, 2023

1 Introduction

In the last lecture we introduced LF, although we only started to talk about the representation methodology. We continue this today, discuss some of the shortcomings, and explore how they might be addressed in a substructural logical framework. References to LF and its implementation in Twelf are provided in the previous lecture.

2 Representing Sequent Derivations

A more typical example for the use of a logical framework is the representation of a logic. Actually, we should be more precise: it is not a logic we represent but a specific inference system. So, for example, sequent calculus and the semi-axiomatic sequent calculus will have different representations.

First, we start with the representation of the propositions of (intuitionistic) logic. That's straightforward:

```
prop : type.
and : prop -> prop -> prop.
or : prop -> prop -> prop.
imp : prop -> prop -> prop.
```

Atomic propositions are either variables of type prop, or declared in addition to the logical connectives we already have.

For different constituents of an object logic like propositions or proofs, we have an (overloaded) representation function $\ulcorner - \urcorner$. For example, for propositions we have

$$P_1{:}\mathsf{prop}, \ldots, P_k{:}\mathsf{prop} \vdash \ulcorner A \urcorner : \mathsf{prop}$$

defined by

$$\begin{array}{lcl} \ulcorner P \urcorner & = & P \\ \ulcorner A \land B \urcorner & = & \mathsf{and}\ \ulcorner A \urcorner\ \ulcorner B \urcorner \\ \ulcorner A \lor B \urcorner & = & \mathsf{or}\ \ulcorner A \urcorner\ \ulcorner B \urcorner \\ \ulcorner A \supset B \urcorner & = & \mathsf{imp}\ \ulcorner A \urcorner\ \ulcorner B \urcorner \end{array}$$

Here, instead of $c\ (M_1\ ; \ldots ;\ M_n)$ from focusing we write $c\ M_1\ \ldots M_n$, which is the familiar source-level syntax from logical frameworks based on natural deduction.

When it comes to judgments, recall that basic judgments are represented as types. But what are the basic judgments here? It turns out the correct representation uses two: "A is an antecedent" and "A is a succedent", which we write as $\mathsf{ante}\ \ulcorner A \urcorner$ and $\mathsf{succ}\ \ulcorner A \urcorner$ respectively. A proof

$$\begin{array}{c} \mathcal{D} \\ A_1, \ldots, A_n \vdash C \end{array}$$

is then represented by the LF sequent

$$x_1 : \mathsf{ante}\ \ulcorner A_1 \urcorner, \ldots x_n : \mathsf{ante}\ \ulcorner A_n \urcorner \vdash_\Sigma \ulcorner \mathcal{D} \urcorner : \mathsf{succ}\ \ulcorner C \urcorner$$

where $\ulcorner \mathcal{D} \urcorner$ is an object of LF and the signature Σ contains the constructors for propositions and proofs. Actually, we have to modify this slightly if $A_1, \ldots, A_n, C$ contain propositional variables P_i, in which case it becomes

$$P_1 : \mathsf{prop}, \ldots, P_k : \mathsf{prop}, x_1 : \mathsf{ante}\ \ulcorner A_1 \urcorner, \ldots x_n : \mathsf{ante}\ \ulcorner A_n \urcorner \vdash_\Sigma \ulcorner \mathcal{D} \urcorner : \mathsf{succ}\ \ulcorner C \urcorner$$

Since atomic propositions may also be represented as constants in the signature, we'll ignore this detail and focus on the representation of proofs.

Let's start with the right rule for implication.

$$\mathcal{D} \quad = \quad \dfrac{\begin{array}{c}\mathcal{D}' \\ \Gamma, A \vdash B\end{array}}{\Gamma \vdash A \supset B} \supset\! R$$

Writing out the representation of the two sequents in LF:

$$\dfrac{\ulcorner \Gamma \urcorner, x : \mathsf{ante}\ \ulcorner A \urcorner \vdash \ulcorner \mathcal{D}' \urcorner : \mathsf{succ}\ \ulcorner B \urcorner}{\ulcorner \Gamma \urcorner \vdash \ulcorner \mathcal{D} \urcorner : \mathsf{succ}\ (\mathsf{imp}\ \ulcorner A \urcorner\ \ulcorner B \urcorner)}$$

We might conjecture we could achieve that if

$$\ulcorner \mathcal{D} \urcorner = \mathsf{impR}\ (\lambda x.\ \ulcorner \mathcal{D}' \urcorner)$$

so that

$$\begin{aligned}\mathsf{impR} : {} & (\mathsf{ante}\ \ulcorner A \urcorner \to \mathsf{succ}\ \ulcorner B \urcorner) \\ & \to \mathsf{succ}\ (\mathsf{imp}\ \ulcorner A \urcorner\ \ulcorner B \urcorner)\end{aligned}$$

but we also need to abstract over the propositions A and B:

$$\begin{aligned}\mathsf{impR} : {} & \Pi A : \mathsf{prop}.\, \Pi B : \mathsf{prop}. \\ & (\mathsf{ante}\ A \to \mathsf{succ}\ B) \\ & \to \mathsf{succ}\ (\mathsf{imp}\ A\ B)\end{aligned}$$

We play through what happens if we focus on this on the left, writing $?A$ and $?B$ for an as yet unknown object of type prop and omitting contexts $\ulcorner \Gamma \urcorner$.

$$\dfrac{\begin{matrix}\vdots \\ \vdash ?A : \mathsf{prop}\end{matrix} \qquad \dfrac{\begin{matrix}\vdots \\ \vdash ?B : \mathsf{prop}\end{matrix} \qquad \dfrac{\dfrac{\dfrac{\mathsf{ante}\ ?A \vdash \langle \mathsf{succ}\ ?B \rangle}{\mathsf{ante}\ ?A \vdash \mathsf{succ}\ ?B}}{\vdash \mathsf{ante}\ ?A \to \mathsf{succ}\ ?B}\ {\to}R \qquad \dfrac{\delta = \langle \mathsf{succ}\ (\mathsf{imp}\ ?A\ ?B) \rangle}{[\mathsf{succ}\ (\mathsf{imp}\ ?A\ ?B)] \vdash \delta}\ \mathsf{id}^-}{[(\mathsf{ante}\ ?A \to \mathsf{succ}\ ?B) \to \mathsf{succ}\ (\mathsf{imp}\ ?A\ ?B)] \vdash \delta}\ {\to}L}{[\Pi B{:}\mathsf{prop}.\ (\mathsf{ante}\ ?A \to \mathsf{succ}\ B) \to \mathsf{succ}\ (\mathsf{imp}\ ?A\ B)] \vdash \delta}\ \Pi L}{[\Pi A{:}\mathsf{prop}.\, \Pi B{:}\mathsf{prop}.\ (\mathsf{ante}\ A \to \mathsf{succ}\ B) \to \mathsf{succ}\ (\mathsf{imp}\ A\ B)] \vdash \delta}\ \Pi L$$

Due to the restriction on focusing with negative atoms, we see that focusing on impR can only succeed if the (metalevel) succedent has the form $\langle \mathsf{succ}\ (\mathsf{imp}\ \ulcorner A \urcorner\ \ulcorner B \urcorner) \rangle$ for some propositions A and B. Applying this throughout, we get the derived rule

$$\dfrac{\ulcorner \Gamma \urcorner, \mathsf{ante}\ \ulcorner A \urcorner \vdash \langle \mathsf{succ}\ \ulcorner B \urcorner \rangle}{\ulcorner \Gamma \urcorner \vdash \langle \mathsf{succ}\ (\mathsf{imp}\ \ulcorner A \urcorner\ \ulcorner B \urcorner) \rangle}$$

which is exactly what we were aiming for. If we add proof terms we get

$$\dfrac{\ulcorner \Gamma \urcorner, \mathsf{ante}\ \ulcorner A \urcorner \vdash M : \langle \mathsf{succ}\ \ulcorner B \urcorner \rangle}{\ulcorner \Gamma \urcorner \vdash (\mathsf{impR}\ \ulcorner A \urcorner\ \ulcorner B \urcorner\ (\lambda x.\, M(x))) : \langle \mathsf{succ}\ (\mathsf{imp}\ \ulcorner A \urcorner\ \ulcorner B \urcorner) \rangle}$$

Using focusing in this manner allows us to establish a bijection: If M is the representation of a proof $\mathcal{D}$, then $\mathsf{impR}\ \ulcorner A \urcorner\ \ulcorner B \urcorner\ (\lambda x.\, M(x))$ will also be one. Conversely, if $\ulcorner \Gamma \urcorner \vdash M : \langle \mathsf{succ}\ C \rangle$ then M must be the proof term for one of the inference rules encoded in the signature Σ. In each case, the same must be true for the unknown object in the premise(s), and we can systematically translate well-typed terms to sequent calculus proofs.

Let's look at the left rule for implication as one more example.

$$\dfrac{\Gamma \vdash A \quad \Gamma, B \vdash C}{\Gamma, A \supset B \vdash C}\ {\supset}L$$

We conjecture

$$\begin{array}{l}\mathsf{impL} : \Pi A : \mathsf{prop}.\, \Pi B : \mathsf{prop}.\, \Pi C : \mathsf{prop}. \\ \qquad\quad \mathsf{succ}\ A \to (\mathsf{ante}\ B \to \mathsf{succ}\ C) \\ \qquad\quad \to (\mathsf{ante}\ (\mathsf{imp}\ A\ B) \to \mathsf{succ}\ C)\end{array}$$

We won't go through the details of focusing, but what we arrive at is

$$\frac{\ulcorner\Gamma\urcorner \vdash M : \langle \mathsf{succ}\ \ulcorner A\urcorner\rangle \quad \ulcorner\Gamma\urcorner, y : \mathsf{ante}\ \ulcorner B\urcorner \vdash N(y) : \langle \mathsf{succ}\ \ulcorner C\urcorner\rangle}{\ulcorner\Gamma\urcorner, x : \mathsf{ante}\ \ulcorner A \supset B\urcorner \vdash \mathsf{impL}\ \ulcorner A\urcorner\ \ulcorner B\urcorner\ \ulcorner C\urcorner\ M\ (\lambda y.\, N(y))\ x : \langle \mathsf{succ}\ \ulcorner C\urcorner\rangle}$$

We see here that the encoding for a structural sequent calculus critically relies on the fact that the LF metalanguage is also structural. So there may be occurrences of x in M and N, just like the hypothesis $A \supset B$ can be used again in both premises of the sequent calculus rule.

You might also notice that modulo the explicit propositions and the argument order, this is exactly the proof term representation we used in Assignment 2 for ordered proofs. This is, of course, no coincidence. In practical implementations of logical frameworks such as LF, the arguments of type prop to the constructors can be omitted and will be determined by the implementation from context. In general, this problem is undecidable for LF and may require more information from the programmer, but in the vast majority of the cases the constraints on them are sufficient.

We can see how proof checking in an object logic (structural, for the moment) can be reduced to type-checking the proof representation in LF (also structural). Moreover, the representation is a bijection between proofs and the well-typed terms over a signature (the encoding of a proof system) and a particular context (the encoding of the antecedents).

Before moving on, let's look back at the type of impR.

$$\begin{array}{l}\mathsf{impR} : \Pi A : \mathsf{prop}.\, \Pi B : \mathsf{prop}. \\ \qquad\quad (\mathsf{ante}\ \ulcorner A\urcorner \to \mathsf{succ}\ \ulcorner B\urcorner) \\ \qquad\quad \to \mathsf{succ}\ (\mathsf{imp}\ \ulcorner A\urcorner\ \ulcorner B\urcorner)\end{array}$$

The adequacy of this encoding as mentioned in the preceding paragraph relies critically on the fact that the function spaces here are weak. For example, a function cannot examine the structure of its argument and base its result on it. Instead, an object of type $A \to B$ must be $\lambda x.\, M(x)$ which is parametric in x so that $M(N)$ is just the result of hereditary substitution of N for x in $M(x)$ without further computation.

3 A Linear Logical Framework

When encoding a substructural type system, the methodology we have exemplified in the preceding section no longer works for LF. That's because LF antecedents of the form $x : \mathsf{ante}\ \ulcorner A \urcorner$ are structural, so we cannot use them for the linear antecedents of linear logic. The most direct solution is to generalize the framework so it also admits linear antecedents. We cannot throw out the nonlinear functions, so the framework will be a mixed linear/nonlinear type theory call Linear LF (or LLF, for short) [Cervesato and Pfenning, 1996, 2002].

Given what we know now regarding adjoint logic, we might have designed the framework differently. We start by just adding a linear function type, but leaving the remainder of the language the same; later on we'll need something one more type. We also just reuse λ-abstraction for linear functions and spines $(M \; ; \; S)$ for linear application since this ambiguity is not relevant here.

$$\begin{array}{llcl} \text{Negative types} & A, B & ::= & P \mid A \to B \mid \Pi x : A.\, B(x) \mid A \multimap B \\ \text{Stable antecedents} & \Delta & ::= & \cdot \mid \Delta, x_{\mathsf{S}} : A \mid \Delta, x_{\mathsf{L}} : A \end{array}$$

Stable antecedents may be structural $x_{\mathsf{S}} : A$ or linear $x_{\mathsf{L}} : A$. All declarations in the signature remain structural: even in a linear logic, inference rules can be used multiple times.

Focusing works similar to the way it worked before—we highlight only two rules for the key differences between the linear and nonlinear left rules, namely that the first premise of $\to L$ can only depend on structural antecedents.

$$\frac{\Gamma_{\mathsf{S}} \vdash M : [A] \quad \Gamma_{\mathsf{S}}, \Delta', [B] \vdash S : \delta}{\Gamma_{\mathsf{S}}, \Delta', [A \to B] \vdash (M \; ; \; S) : \delta} \to L$$

$$\frac{\Gamma_{\mathsf{S}}, \Delta \vdash M : [A] \quad \Gamma_{\mathsf{S}}, \Delta', [B] \vdash (M \; ; \; S) : \delta}{\Gamma_{\mathsf{S}}, \Delta, \Delta', [A \multimap B] \vdash (M \; ; \; S) : \delta} \multimap L$$

The focus on A in the first premise of both of these rules will be lost immediately since the framework consists entirely of negative types. The succedent δ will always be a suspended negative atom $\langle P \rangle$, as for LF.

As our example we use the (purely linear) semi-axiomatic sequent calculus (SAX). We represent a proof

$$\begin{array}{c} \mathcal{D} \\ A_1, \ldots, A_n \vdash C \end{array}$$

by

$$x_{\mathsf{L}}^1 : \mathsf{ante}\ \ulcorner A_1 \urcorner, \ldots, x_{\mathsf{L}}^n : \mathsf{ante}\ \ulcorner A_n \urcorner \vdash \ulcorner \mathcal{D} \urcorner : \langle \mathsf{succ}\ \ulcorner C \urcorner \rangle$$

The right rule for linear implication parallels what we have seen in the structural case, but the left rule is replace by the axiom

$$\frac{}{A, A \multimap B \vdash B} \multimap X$$

which simplifies the representation slightly.

$\mathsf{lolli} : \mathsf{prop} \to \mathsf{prop} \to \mathsf{prop}$

$\mathsf{lolliR} : \Pi A : \mathsf{prop}.\, \Pi B : \mathsf{prop}.$
$\quad (\mathsf{ante}\ A \multimap \mathsf{succ}\ B)$
$\quad \multimap \mathsf{succ}\ (\mathsf{lolli}\ A\ B)$

$\mathsf{lolliX} : \Pi A : \mathsf{prop}.\, \Pi B : \mathsf{prop}.$
$\quad \mathsf{ante}\ A \multimap \mathsf{ante}\ (\mathsf{lolli}\ A\ B) \multimap \mathsf{succ}\ B$

The identity is similar to the axiom.

$\mathsf{id} : \Pi A : \mathsf{prop}.\, \mathsf{ante}\ A \multimap \mathsf{succ}\ A$

he fram The additive conjunction represents a new challenge.

$$\frac{\Delta \vdash A \quad \Delta \vdash B}{\Delta \vdash A \mathbin{\&} B}\ \&R \qquad \frac{}{A \mathbin{\&} B \vdash A}\ \&X_1 \quad \frac{}{A \mathbin{\&} B \vdash B}\ \&X_2$$

The challenge here is how to "duplicate" the antecedents Δ to both premises of the $\&R$ rule. In the framework we have so far, there is not easy way to accomplish this. Fortunately, external choice is negative so we can just add it to the framework without disturbing much of its structure.

Negative types	A, B	$::=$	$P \mid A \to B \mid \Pi x : A.\, B(x) \mid A \multimap B \mid A \mathbin{\&} B$
Objects	M	$::=$	$c\ S \mid x\ S \mid \lambda x.\, M(x) \mid (M_1, M_2)$
Spines	S	$::=$	$M\ ;\ S \mid (\,) \mid \pi_1\ ;\ S \mid \pi_2\ ;\ S$
Stable antecedents	Δ	$::=$	$\cdot \mid \Delta, x_{\mathsf{S}} : A \mid \Delta, x_{\mathsf{L}} : A$

With this addition we can define

$\mathsf{with} : \mathsf{prop} \to \mathsf{prop} \to \mathsf{prop}$

$\mathsf{withR} : \Pi A : \mathsf{prop}.\, \Pi B : \mathsf{prop}.$
$\quad (\mathsf{succ}\ A \mathbin{\&} \mathsf{succ}\ B)$
$\quad \multimap \mathsf{succ}\ (\mathsf{with}\ A\ B)$

$\mathsf{withX_1} : \Pi A : \mathsf{prop}.\, \Pi B : \mathsf{prop}.$
$\quad \mathsf{ante}\ (\mathsf{with}\ A\ B) \multimap \mathsf{succ}\ A$

$\mathsf{withX_1} : \Pi A : \mathsf{prop}.\, \Pi B : \mathsf{prop}.$
$\quad \mathsf{ante}\ (\mathsf{with}\ A\ B) \multimap \mathsf{succ}\ B$

Because right inversion on $\mathsf{succ}\ A \mathbin{\&} \mathsf{succ}\ B$ will propagate all linear antecedents to both premises, the translation of

$$\frac{\begin{array}{c}\mathcal{D}\\ \Delta \vdash A\end{array} \quad \begin{array}{c}\mathcal{E}\\ \Delta \vdash B\end{array}}{\Delta \vdash A \mathbin{\&} B}\ \&R$$

as

$$\mathsf{withR}\ \ulcorner A \urcorner\ \ulcorner B \urcorner\ (\ulcorner \mathcal{D} \urcorner, \ulcorner \mathcal{E} \urcorner) : \langle \mathsf{succ}\ (\mathsf{with}\ \ulcorner A \urcorner\ \ulcorner B \urcorner)\rangle$$

is adequately typed.

If we also encode cut

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

as

$$\begin{aligned}\mathsf{cut} : \Pi A : \mathsf{prop}.\, \Pi B : \mathsf{prop}.\\ &\mathsf{succ}\ A \multimap (\mathsf{ante}\ A \multimap \mathsf{succ}\ C)\\ &\multimap \mathsf{succ}\ C\end{aligned}$$

then we can, for example, show that the usual sequent calculus left rules are derivable in SAX. The derivation

$$\frac{\dfrac{}{A \mathbin{\&} B \vdash A}\ \&X_1 \quad \Delta, A \vdash C}{\Delta, A \mathbin{\&} B \vdash C}\ \mathsf{cut}$$

becomes

$$\begin{aligned}\vdash (\lambda f.\, \lambda y.\, \mathsf{cut}\ \ulcorner A \urcorner\ \ulcorner C \urcorner\ (\mathsf{withX}_1\ \ulcorner A \urcorner\ \ulcorner B \urcorner\ y)\ (\lambda x. f x)) : (\mathsf{ante}\ \ulcorner A \urcorner \multimap \mathsf{succ}\ \ulcorner C \urcorner)\\ \multimap (\mathsf{ante}\ (\mathsf{with}\ \ulcorner A \urcorner\ \ulcorner B \urcorner) \multimap \mathsf{succ}\ \ulcorner C \urcorner)\end{aligned}$$

4 Metatheoretic Reasoning

One payoff for using high-level encodings is that they can enable elegant and concise metatheoretic reasoning [Schürmann, 2000]. For substructural logics, this is much less well understood and I believe Jason Reed's PhD Thesis 2009 using resource semantics is probably currently the high water mark, although more recent work with entirely different techniques is also promising [Sano et al., 2023, Crary, 2010].

Let's first consider the structural case, and let's assume we have formalized the sequent calculus *without the cut rule*. Then, at some informal level, the (constructive!) admissibility proof for cut would correspond to a function from a proof of A and a proof of C from A to a proof of C. In LF, this might be written as

$$\begin{aligned}\mathsf{cutadmit} : \Pi A : \mathsf{prop}.\, \Pi C : \mathsf{prop}.\\ &\mathsf{succ}\ A \to (\mathsf{ante}\ A \to \mathsf{succ}\ C) \to \mathsf{succ}\ C\end{aligned}$$

Unfortunately, such a function is not representable in LF because it would have to distinguish all the different cases for the proofs of $\mathsf{succ}\ A$ and the hypothetical proof of $\mathsf{ante}\ A \to \mathsf{succ}\ C$. Allowing such case distinction would destroy the adequacy of the encoding, although there are systems such as $\mathcal{M}_2^+$ [Schürmann, 2000] and

Beluga [Pientka and Cave, 2015] that support multiple different kinds of function spaces. In Twelf [Pfenning and Schürmann, 1999], the solution is to represent the metatheoretic proof instead as a *relation*.

$$\begin{aligned}\mathsf{cutadmit} : \Pi A : \mathsf{prop}.\, \Pi C : \mathsf{prop}.\\ &\mathsf{succ}\ A \to (\mathsf{ante}\ A \to \mathsf{succ}\ C) \to \mathsf{succ}\ C\\ &\to \mathbf{type}\end{aligned}$$

We can then check properties of this relation to verify our theorem. Specifically, it should be total in A, C and the two given derivations. You can read more about this in the following two papers [Pfenning, 1995, 2000].

It turns out this relational method is quite general, and there are many examples and case studies in the Twelf distribution and on the website.[1]

References

Iliano Cervesato and Frank Pfenning. A linear logical framework. In E. Clarke, editor, *Proceedings of the Eleventh Annual Symposium on Logic in Computer Science*, pages 264–275, New Brunswick, New Jersey, July 1996. IEEE Computer Society Press.

Iliano Cervesato and Frank Pfenning. A linear logical framework. *Information & Computation*, 179(1):19–75, November 2002. Revised and expanded version of an extended abstract, LICS 1996, pp. 264-275.

Karl Crary. Higher-order representation of substructural logics. In P.Hudak and S.Weirich, editors, *Proceedings of the 15th International Conference on Functional Programming (ICFP 2010)*, pages 131–142, Baltimore, Maryland, September 2010. ACM.

Frank Pfenning. Structural cut elimination. In D. Kozen, editor, *Proceedings of the Tenth Annual Symposium on Logic in Computer Science*, pages 156–166, San Diego, California, June 1995. IEEE Computer Society Press.

Frank Pfenning. Structural cut elimination I. Intuitionistic and classical logic. *Information and Computation*, 157(1/2):84–141, March 2000.

Frank Pfenning and Carsten Schürmann. System description: Twelf — a meta-logical framework for deductive systems. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction (CADE-16)*, pages 202–206, Trento, Italy, July 1999. Springer-Verlag LNAI 1632.

[1] `http://twelf.org/`

Brigitte Pientka and Andrew Cave. Inductive Beluga: Programming proofs. In A. Felty and A. Middeldorp, editors, *25th International Conference on Automated Deduction (CADE 2015)*, pages 272–281, Berlin, Germany, August 2015. Springer LNCS 9195.

Jason C. Reed. *A Hybrid Logical Framework*. PhD thesis, Carnegie Mellon University, September 2009. Available as Technical Report CMU-CS-09-155.

Chuta Sano, Ryan Kavanagh, and Brigitte Pientka. Mechanizing session-types using a structural view: Enforcing linearity without linearity. In *Proceedings of the ACM on Programming Languages*, volume 7 (OOPSLA2), pages 374–399. ACM, 2023. Extended version available at `https://arxiv.org/abs/2309.12466`.

Carsten Schürmann. *Automating the Meta Theory of Deductive Systems*. PhD thesis, Department of Computer Science, Carnegie Mellon University, August 2000. Available as Technical Report CMU-CS-00-146.