Lecture Notes on
The Concurrent Logical Framework

15-816: Substructural Logics
Frank Pfenning

Lecture 23
November 17, 2016

A *logical framework* is a metalanguage for the representation and analysis of deductive systems such as logics, type systems, specifications of operational semantics, etc. The goal is to distill the essence of deductive systems so that encodings are as direct and natural as possible. In many ways one can consider them *normative* in that they embody the judgmental principles upon which the design of logics and programming languages are (or ought to be) based on.

An early logical framework was LF [HHP87, HHP93], implemented in the Twelf system [PS99] which is based on a minimal structural dependent type system λ^{Π}. It elucidated and crystallized the notions of bound variable, capture-avoiding substitution, hypothetical judgment, and generic judgment. The high level nature of the encodings allowed automatic and programmatic theorem proving [Sch00] as well as execution of some specifications as backward-chaining logic programs [MN86, Pfe91].

It was recognized early on that substructural logics and related programming languages could not be represented as directly in LF and related frameworks such as λProlog [MN86] as one might hope. Essentially, early frameworks did not support linear hypothetical judgments directly, which hampered encodings. This was addressed in a line of research on substructural linear [HM94, Mil94, Chi95, CP96, CP02] and ordered [PP99b, PP99a, Pol01] logical frameworks, eventually culminating in the Concurrent Logical Framework (CLF) [WCPW02, CPWW02, WCPW04] and its implementation in Celf [SNS08].

CLF is expressive and robust enough to allow logic programming [LPPW05] but metatheoretic reasoning in the style of Twelf remains elusive (see [Ree09]

for one approach). In this lecture we focus on the *positive fragment* of CLF, applying a bit of hindsight to polarize the original presentation. This fragment is of particular interest since its forward-chaining operational semantics allows us to represent the deductive systems we have analyzed in this course in a high-level and executable manner.

1 The Positive Fragment of CLF

CLF arises from the polarized adjoint formulation of intuitionistic linear logic by admitting dependent types. We will largely downplay this aspect of the CLF, since it is rich enough to be the subject of its own course [Pfe92]. Instead we emphasize the substructural aspects of the framework. Before launching into its description, we should emphasize that we are interested almost exlusively in focused, cut-free proofs. It is terms representing these proofs that end up being in bijective correspondence with the objects we would like to represent.

By default, layers of the syntax are *linear*, so we will only annotate types that are structural as A_{U}.

$$
\begin{array}{lcll}
A_{\mathsf{U}}^+ & ::= & p_{\mathsf{U}}^+ \mid \ldots & \\
A^- & ::= & A^+ \multimap B^- \mid \Pi x{:}A_{\mathsf{U}}^+ . B^- \mid A^- \mathbin{\&} B^- \mid \uparrow A^+ & \text{(but not } p^-) \\
A^+ & ::= & p^+ \mid A^+ \otimes B^+ \mid \mathbf{1} \mid \exists x{:}A_{\mathsf{U}}^+ . B^+ & \text{(but not } \downarrow A^-)
\end{array}
$$

A few remarks on these types. We do not include negative atoms (p^-) or $\downarrow A^-$, which constitutes our restriction to the negative fragment. We omitted disjunction $A^+ \oplus B^+$ because we have not carried out the theory to understand what true concurrency would mean, something we discuss in the next lecture. We have left open what kinds of propositions we would have in the structural layer. Positive atoms p_{U}^+ are useful because they correspond to the persistent propositions we have used in various representations.

Note that universal ($\Pi x{:}A_{\mathsf{U}}^+ . B^+$) and existential ($\exists x{:}A_{\mathsf{U}}^+ . B^+$) are constructs of mixed mode, combining structural and linear types into a linear type. This appears to be necessary: while one can give formalistic constructions of a linear dependent function space, there is to date no fully satisfactory account of it. The reason lies in the question of what constitutes a "linear use" of x in a hypothetical linear Π, as compared to simply a "mention" of x in the type. In practice, we have developed a number of techniques to circumvent the need for linear dependent functions, mostly by splitting the name x (which is persistent) from a linear capability which

explictly marks uses x. The complications, by the way, are not specific to linear logic but appear in the literature of modal logic in various forms even just for first-order modal logic.

The proof term assignment to this calculus turns out to be quite a bit different for call-by-push-value or for SILL, both of which were similarly polarized. Here, we are interested in representing only cut-free, focused proofs because these are used for representation. For starters, in call-by-push-value we had two forms of terms: computations (of negative type) and values (of positive type). Here we will have five different forms of terms, corresponding to right inversion and left focusing (negative types), left inversion and right focusing (positive types) and one for neutral sequents, before a focusing phase is started. We introduce the terms in stages, but first all five judgments. We use Γ for positive structural contexts, Δ for linear antecedents, and Ω^+ for linear antecedents presented in an ordered fashion so that inversion is deterministic.

$\Gamma\,;\,\Delta \vdash M : A^-$	right inversion
$\Gamma\,;\,\Delta\,;\,\Omega^+ \vdash J$	left inversion
$\Gamma\,;\,\Delta, [R : A^-] \vdash E : C^+$	left focusing
$\Gamma\,;\,\Delta \vdash [V : C^+]$	right focusing
$\Gamma\,;\,\Delta \vdash E : C^+$	stable sequent
$\Gamma \vdash [V_{\mathsf{U}} : A^+_{\mathsf{U}}]$	structural right focus

In left inversion, the judgment J on the right could be either $M : A^-$ or $E : C^+$.

Right Inversion. For right inversion, the assignment is straightforward, consistent with our call-by-push-value functional language, even though we are operating in a sequent calculus here. The judgment for right inversion is $\Gamma\,;\,\Delta \vdash M : A^-$.

$$\frac{\Gamma\,;\,\Delta\,;\,p : A^+ \vdash M : B^-}{\Gamma\,;\,\Delta \vdash \lambda p.\, M : A^+ \multimap B^-}\ \multimap R \qquad \frac{\Gamma, x{:}A^+_{\mathsf{U}}\,;\,\Delta \vdash M : B^-}{\Gamma\,;\,\Delta \vdash \lambda x.\, M : \Pi x{:}A.\, B^-}\ \Pi R$$

$$\frac{\Gamma\,;\,\Delta \vdash M : A^- \quad \Gamma\,;\,\Delta \vdash N : B^-}{\Gamma\,;\,\Delta \vdash \langle M, N\rangle : A^- \,\&\, B^-}\ \&R \qquad \frac{\Gamma\,;\,\Delta \vdash E : A^+}{\Gamma\,;\,\Delta \vdash \{E\} : \uparrow A^+}\ \uparrow R$$

In the final rule we transition to the stable judgment, where all declarations in Δ are either $x : A^-$ or $x : p^+$. For Γ, we only consider $x^+_{\mathsf{U}} : p^+_{\mathsf{U}}$, which is also stable.

Left Inversion. Left inversion operates on an ordered context Ω with propositions $p : A^+$ where p is a *pattern* (not an atomic type). When the context is empty and all inversion steps have been applied, we return to the judgment J.

$$\frac{\Gamma\,;\,\Delta, x{:}p^+\,;\,\Omega \vdash J}{\Gamma\,;\,\Delta\,;\,(x : p^+)\,\Omega \vdash J}\ \mathsf{atm}^+ \qquad \frac{\Gamma\,;\,\Delta\,;\,(p : A^+)\,(q : B^+)\,\Omega \vdash J}{\Gamma\,;\,\Delta\,;\,([p, q] : A^+ \otimes B^+)\,\Omega \vdash J}\ \otimes L$$

$$\frac{\Gamma\,;\,\Delta\,;\,\Omega \vdash J}{\Gamma\,;\,\Delta\,;\,([\,] : \mathbf{1})\,\Omega \vdash J}\ \mathbf{1}L \qquad \frac{\Gamma, x{:}A^+_{\mathsf{U}}\,;\,\Delta\,;\,(q : B^+)\,\Omega \vdash J}{\Gamma\,;\,\Delta\,;\,([x_{\mathsf{U}}, q] : \exists x_{\mathsf{U}} : A^+_{\mathsf{U}}.\, B^+)\,\Omega \vdash J}\ \exists L$$

$$\frac{\Gamma\,;\,\Delta \vdash J}{\Gamma\,;\,\Delta\,;\,\cdot \vdash J}\ \mathsf{empty}$$

Left Focus. When thinking about left focus, we have to think about the *signature* Σ which contains (in our case) constants $c : A^-_{\mathsf{L}}$, arbitrarily reusable. Strictly speaking, there should be a shift here, but we dispense with that due to the special case of the global signature.

$$\frac{(c{:}A^- \in \Sigma) \quad \Gamma\,;\,\Delta, [c : A^-] \vdash E : C^+}{\Gamma\,;\,\Delta \vdash E : C^+}\ \mathsf{foc}^-_{\mathsf{U}} \qquad \frac{\Gamma\,;\,\Delta, [x : A^-] \vdash E : C^+}{\Gamma\,;\,\Delta, x{:}A^- \vdash E : C^+}\ \mathsf{foc}^-_{\mathsf{L}}$$

$$\frac{\Gamma\,;\,\Delta \vdash [V : A^+] \quad \Gamma\,;\,\Delta, [R\,V : B] \vdash E : C^+}{\Gamma\,;\,\Delta, \Delta', [R : A^+ \multimap B^-] \vdash E : C^+}\ \multimap L$$

$$\frac{\Gamma \vdash [V_{\mathsf{U}} : A^+_{\mathsf{U}}] \quad \Gamma\,;\,\Delta, [R\,V_{\mathsf{U}} : [V_{\mathsf{U}}/x]B^-] \vdash E : C^+}{\Gamma\,;\,\Delta, [R : \Pi x{:}A^+_{\mathsf{U}}.\, B^+] \vdash E : C^+}\ \Pi L$$

$$\frac{\Gamma\,;\,\Delta, [\pi_1\,R : A^-] \vdash E : C^+}{\Gamma\,;\,\Delta, [R : A^- \,\&\, B^-] \vdash E : C^+}\ \&L_1 \qquad \frac{\Gamma\,;\,\Delta, [\pi_2\,R : B^-] \vdash E : C^+}{\Gamma\,;\,\Delta, [R : A^- \,\&\, B^-] \vdash E : C^+}\ \&L_2$$

$$\frac{\Gamma\,;\,\Delta\,;\,p : A^+ \vdash E : C^+}{\Gamma\,;\,\Delta, [R : \uparrow A^+] \vdash \mathsf{let}\ \{p\} = R\ \mathsf{in}\ E : C^+}\ \uparrow L$$

The last rule here represents a transition to the left inversion judgment.

Right Focus. Finally, we come to right focus which, in the positive fragment, will always either succeed and finish the proof or fail. Since the pos-

itive fragment lacks $\downarrow A^-$ we cannot lose focus.

$$\frac{\Gamma\ ;\ \Delta \vdash [E : C^+]}{\Gamma\ ;\ \Delta \vdash E : C^+}\ \mathsf{foc}^+ \qquad \frac{}{\Gamma\ ;\ x{:}p^+ \vdash [x : p^+]}\ \mathsf{id}^+$$

$$\frac{\Gamma\ ;\ \Delta \vdash [V : A] \quad \Gamma\ ;\ \Delta \vdash [W : B]}{\Gamma\ ;\ \Delta, \Delta' \vdash [[V, W] : A \otimes B]}\ \otimes R \qquad \frac{}{\Gamma\ ;\ \cdot \vdash [[\,] : \mathbf{1}]}\ \mathbf{1}R$$

$$\frac{\Gamma \vdash [V_{\mathsf{U}} : A^+_{\mathsf{U}}] \quad \Gamma\ ;\ \Delta \vdash [W : [V_{\mathsf{U}}/x]B^+]}{\Gamma\ ;\ \Delta \vdash [[V_{\mathsf{U}}, W] : \exists x{:}A^+_{\mathsf{U}}.\, B^+]}\ \exists R$$

Structural Right Focus. Since in our language at the moment we only consider atomic structural propositions, we only have one rule.

$$\frac{}{\Gamma, x{:}p^+_{\mathsf{U}}\ ;\ \cdot \vdash [x : p^+_{\mathsf{U}}]}\ \mathsf{id}^+_{\mathsf{U}}$$

2 Summary of Proof Terms

We obtain the following language of terms, where we indicate in each line the corresponding proposition and the concrete Celf syntax for the con-

struct.

		Abstract Syntax		Concrete Syntax	
		Term	Type	Term	Type
M	$::=$	$\lambda p.\, M$	$A^+ \multimap B^-$	`\p. M`	`A -o B`
	$\mid$	$\lambda x_{\mathsf{U}}.\, M$	$\Pi x_{\mathsf{U}}{:}A^-_{\mathsf{U}}.\, B^-$	`\!x. M`	`Pi x:A. B`
	$\mid$	$\langle M, N\rangle$	$A^- \,\&\, B^-$	`< M , N >`	`A & B`
	$\mid$	$\{E\}$	$\uparrow A^+$	`{ E }`	`{ A }`
p	$::=$	x	p^+	`x`	
	$\mid$	$[p, q]$	$A^+ \otimes B^+$	`[p,q]`	`A * B`
	$\mid$	$[\,]$	$\mathbf{1}$	`1`	`1`
	$\mid$	$[x_{\mathsf{U}}, p]$	$\exists x_{\mathsf{U}}{:}A^+_{\mathsf{U}}.\, B^+$	`[!x,p]`	`Exists x:A. B`
R	$::=$	c	$c{:}A^- \in \Sigma$	`c`	
	$\mid$	x	$x{:}A^- \in \Delta$	`x`	
	$\mid$	$R\, V$	$A^+ \multimap B^-$	`R V`	`A -o B`
	$\mid$	$\pi_1\, R \mid \pi_2\, R$	$A \,\&\, B$	`#1 R  #2 R`	`A & B`
	$\mid$	$R\, V_{\mathsf{U}}$	$\Pi x_{\mathsf{U}}{:}A^+_{\mathsf{U}}.\, B^-$	`R !V`	`Pi x:A. B`
V	$::=$	x	p^+	`x`	
	$\mid$	$[V, W]$	$A^+ \otimes B^+$	`[V,W]`	`A * B`
	$\mid$	$[\,]$	$\mathbf{1}$	`1`	`1`
	$\mid$	$[V_{\mathsf{U}}, W]$	$\exists x_{\mathsf{U}}{:}A^+_{\mathsf{U}}.\, B^+$	`[!V,W]`	`Exists x{:}A. B`
E	$::=$	let $\{p\} = R$ in E	left focus	`let {p} = R in E`	`{ A }`
	$\mid$	V	right focus	`V`	

3 Example: Coin Exchange

We have already seen a significant transcription of inference rules into Celf in Lecture 22 on call-by-value and call-by-name.

Let's see CLF in action on a simpler example: the old coin exchange.[1]

```
q : type.
d : type.
n : type.

d2q : d * d * n -o { q }.
q2d : q -o { d * d * n }.
```

[1] Source at http://www.cs.cmu.edu/~fp/courses/15816-f16/misc/exchange.clf

```
n2d : n * n -o { d }.
d2n : d -o { n * n }.
```

We can now perform type-checking by using the form `c : A = M.` which verifies that term M has type A. Moreover, c stands for M in the remainder of the file. The first example is just one step, where we convert three nickels to a dime and a nickel.

```
example1 : n * n * n -o {d * n} =
  \[n1, [n2, n3]]. {             % n1:n, n2:n, n3:n |- _ : d * n
    let {d1} = n2d [n1, n3] in   % d1:n, n2:n       |- _ : d * n
     [d1, n2]                    % d1:n, n2:n       |- _ : d * n
  }.
```

We used, rather arbitrarily, the first and the third nickel to convert to a dime, leaving the last one in our possession. We showed, after each line, the antecedent and the succedent, omitting the proof terms.

We can also see if our forwarding chaining engine would find this proof. Actually it does not, because our forward chaining engine applies rules until quiescence. But since we can exchange coins back and forth, this specification (when viewed as a program) will never terminate. Once we put a bound on the number of steps to take, it depends on luck. In this case, with a bound of 11, it happens to succeed.

```
Query (11, 1, *, 1) (n * (n * n)) -o {d * n}.
Solution: \[X1, [X2, X3]]. {
    let {X4} = n2d [X1, X3] in
    let {[X5, X6]} = d2n X4 in
    let {X7} = n2d [X2, X5] in
    let {[X8, X9]} = d2n X7 in
    let {X10} = n2d [X8, X9] in
    let {[X11, X12]} = d2n X10 in
    let {X13} = n2d [X6, X11] in
    let {[X14, X15]} = d2n X13 in
    let {X16} = n2d [X12, X14] in
    let {[X17, X18]} = d2n X16 in
    let {X19} = n2d [X15, X17] in [X19, X18]}
Query ok.
```

We can clearly see in the proof that it displays, that it changed back-and-forth between two nickels and a time and stops forward chaining to

examine the goal after 11 iterations. It so happens that we do have one dime and one nickel at that point. Here is one more example, this time using the more reliable type-checking.

```
example2 : n * n * n * n * n -o { q } =
  \[n1, [n2, [n3, [n4, n5]]]]. {
    let {d1} = n2d [n1, n2] in
    let {d2} = n2d [n3, n4] in
    let {q0} = d2q [d1, [d2, n5]] in
    q0
  }.
```

Exercises

Exercise 1 Implement your choice of a finite state transducer like binary increment or compressing runs of b's as a forward-chaining concurrent logic program. You should use the technique of destinations to represent the ordered context linearly so that, for example, the character `a` might be represented as `msg L a R` where `L` and `R` represent destinations that tie this character to its left and right neighbors of the predicate representing the state of a transducer.

Exercise 2 In the style of Exercise 1, implement a pushdown automaton that recognizes a string of properly matched parentheses.

References

[Chi95] Jawahar Lal Chirimar. *Proof Theoretic Approach to Specification Languages*. PhD thesis, University of Pennsylvania, May 1995.

[CP96] 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.

[CP02] 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.

[CPWW02] Iliano Cervesato, Frank Pfenning, David Walker, and Kevin Watkins. A concurrent logical framework II: Examples and applications. Technical Report CMU-CS-02-102, Department of Computer Science, Carnegie Mellon University, 2002. Revised May 2003.

[HHP87] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. In *Symposium on Logic in Computer Science*, pages 194–204. IEEE Computer Society Press, June 1987.

[HHP93] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993.

[HM94] Joshua Hodas and Dale Miller. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation*, 110(2):327–365, 1994. A preliminary version appeared in the Proceedings of the Sixth Annual IEEE Symposium on Logic in Computer Science, pages 32–42, Amsterdam, The Netherlands, July 1991.

[LPPW05] Pablo López, Frank Pfenning, Jeff Polakow, and Kevin Watkins. Monadic concurrent linear logic programming. In A.Felty, editor, *Proceedings of the 7th International Symposium on Principles and Practice of Declarative Programming (PPDP'05)*, pages 35–46, Lisbon, Portugal, July 2005. ACM Press.

[Mil94] Dale Miller. A multiple-conclusion meta-logic. In S. Abramsky, editor, *Ninth Annual Symposium on Logic in Computer Science*, pages 272–281, Paris, France, July 1994. IEEE Computer Society Press.

[MN86] Dale Miller and Gopalan Nadathur. Higher-order logic programming. In Ehud Shapiro, editor, *Proceedings of the Third International Logic Programming Conference*, pages 448–462, London, June 1986.

[Pfe91] Frank Pfenning. Logic programming in the LF logical framework. In Gérard Huet and Gordon Plotkin, editors, *Logical Frameworks*, pages 149–181. Cambridge University Press, 1991.

[Pfe92] Frank Pfenning. Computation and deduction. Unpublished lecture notes, 277 pp. Revised May 1994, April 1996, May 1992.

[Pol01] Jeff Polakow. *Ordered Linear Logic and Applications*. PhD thesis, Department of Computer Science, Carnegie Mellon University, August 2001.

[PP99a] Jeff Polakow and Frank Pfenning. Natural deduction for intuitionistic non-commutative linear logic. In J.-Y. Girard, editor, *Proceedings of the 4th International Conference on Typed Lambda Calculi and Applications (TLCA'99)*, pages 295–309, L'Aquila, Italy, April 1999. Springer-Verlag LNCS 1581.

[PP99b] Jeff Polakow and Frank Pfenning. Relating natural deduction and sequent calculus for intuitionistic non-commutative linear logic. In Andre Scedrov and Achim Jung, editors, *Proceedings of the 15th Conference on Mathematical Foundations of Programming Semantics*, pages 449–466, New Orleans, Louisiana, April 1999. Electronic Notes in Theoretical Computer Science, Volume 20.

[PS99] 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.

[Ree09] Jason C. Reed. *A Hybrid Logical Framework*. PhD thesis, Carnegie Mellon University, September 2009. Available as Technical Report CMU-CS-09-155.

[Sch00] 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.

[SNS08] Anders Schack-Nielsen and Carsten Schürmann. Celf - a logical framework for deductive and concurrent systems. In A. Armando, P. Baumgartner, and G. Dowek, editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning (IJCAR'08)*, pages 320–326, Sydney, Australia, August 2008. Springer LNCS 5195.

[WCPW02] Kevin Watkins, Iliano Cervesato, Frank Pfenning, and David Walker. A concurrent logical framework I: Judgments and properties. Technical Report CMU-CS-02-101, Department of Computer Science, Carnegie Mellon University, 2002. Revised May 2003.

[WCPW04] Kevin Watkins, Iliano Cervesato, Frank Pfenning, and David Walker. A concurrent logical framework: The propositional fragment. In S. Berardi, M. Coppo, and F. Damiani, editors, *Types for Proofs and Programs*, pages 355–377. Springer-Verlag LNCS 3085, 2004. Revised selected papers from the *Third International Workshop on Types for Proofs and Programs*, Torino, Italy, April 2003.