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 \cite{HHP87, HHP93}, implemented in the Twelf system \cite{PS99} which is based on a minimal structural dependent type system \(\lambda^H\). 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 \cite{Sch00} as well as execution of some specifications as backward-chaining logic programs \cite{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 \(\lambda\)Prolog \cite{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 \cite{HM94, Mil94, Chi95, CP96, CP02} and ordered \cite{PP99b, PP99a, Pol01} logical frameworks, eventually culminating in the Concurrent Logical Framework (CLF) \cite{WCPW02, CPWW02, WCPW04} and its implementation in Celf \cite{SNS08}.

CLF is expressive and robust enough to allow logic programming \cite{LPPW05} but metatheoretic reasoning in the style of Twelf remains elusive (see \cite{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 exclusively 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 $A_U$.

\[
\begin{align*}
A_U^+ & ::= p_U^+ | \ldots \\
A_U^- & ::= A^+ \rightarrow B^- | \Pi x:A_U^+. B^- | A^- \& B^- | \uparrow A^+ \quad \text{(but not $p^-$)} \\
A^+ & ::= p^+ | A^+ \otimes B^+ | 1 | \exists x:A_U^+. B^+ \quad \text{(but not $\downarrow A^-$)}
\end{align*}
\]

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 ($\Pi x:A_U^+. B^+$) and existential ($\exists x:A_U^+. B^+$) 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 $\Pi$, 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
explicitly 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 $\Gamma$ for positive structural contexts, $\Delta$ for linear antecedents, and $\Omega^+$ for linear antecedents presented in an ordered fashion so that inversion is deterministic.

\[
\begin{align*}
\Gamma ; \Delta \vdash M : A^- & \quad \text{right inversion} \\
\Gamma ; \Delta ; \Omega^+ \vdash J & \quad \text{left inversion} \\
\Gamma ; \Delta, [R : A^-] \vdash E : C^+ & \quad \text{left focusing} \\
\Gamma ; \Delta \vdash [V : C^+] & \quad \text{right focusing} \\
\Gamma ; \Delta \vdash E : C^+ & \quad \text{stable sequent} \\
\Gamma \vdash [V_0 : A_0^+] & \quad \text{structural right focus}
\end{align*}
\]

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 $\Gamma ; \Delta \vdash M : A^-$. 

\[
\begin{align*}
\Gamma ; \Delta ; p : A^+ \vdash M : B^- & \quad \rightarrow R \\
\Gamma ; \Delta \vdash \lambda p. M : A^+ \rightarrow B^- & \quad \Pi R \\
\Gamma ; \Delta \vdash M : A^- & \quad \Gamma ; \Delta \vdash N : B^- & \quad \& R \\
\Gamma ; \Delta \vdash (M, N) : A^- \& B^- & \quad \Gamma ; \Delta \vdash E : A^+ & \quad \uparrow R
\end{align*}
\]

In the final rule we transition to the stable judgment, where all declarations in $\Delta$ are either $x : A^-$ or $x : p^+$. For $\Gamma$, we only consider $x_0^+ : p_0^+$, which is also stable.
Left Inversion. Left inversion operates on an ordered context \( \Omega \) with propositions \( p : A^+ \) where \( p \) is a \textit{pattern} (not an atomic type). When the context is empty and all inversion steps have been applied, we return to the judgment \( J \).

\[
\frac{\Gamma ; \Delta, x : p^+ ; \Omega \vdash J}{\Gamma ; \Delta, (x : p^+) \Omega \vdash J} \quad \text{atm}^+ \\
\frac{\Gamma ; \Delta ; (p : A^+) (q : B^+) \Omega \vdash J}{\Gamma ; \Delta ; ([p, q] : A^+ \otimes B^+) \Omega \vdash J} \quad \otimes L \\
\frac{\Gamma ; \Delta ; \Omega \vdash J}{\Gamma ; \Delta, ([] : 1) \Omega \vdash J} \quad 1L \\
\frac{\Gamma ; \Delta, ([x] : A) \Omega \vdash J}{\Gamma ; \Delta, ([x, q] : A_0^+ \otimes B^+) \Omega \vdash J} \quad \exists L \\
\frac{\Gamma ; \Delta \vdash J}{\Gamma ; \Delta ; \emptyset \vdash J} \quad \text{empty}
\]

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

\[
\frac{\Gamma; \Delta, [c : A^−] \vdash E : C^+}{\Gamma; \Delta, c : A^− \vdash E : C^+} \quad \text{foc}_u^− \\
\frac{\Gamma; \Delta, [x : A^−] \vdash E : C^+}{\Gamma; \Delta, x : A^− \vdash E : C^+} \quad \text{foc}_u^− \\
\frac{\Gamma; \Delta \vdash [V : A^+]}{\Gamma; \Delta, \Delta', [R : A^+ \rightarrow B^-] \vdash E : C^+} \quad \text{-}\text{L} \\
\frac{\Gamma \vdash [V_0 : A_0^+]}{\Gamma; \Delta, [R V_0 : [V_0/x] B^-]] \vdash E : C^+} \quad \Pi L \\
\frac{\Gamma; \Delta, [\pi_1 R : A^-] \vdash E : C^+}{\Gamma; \Delta, [R : A^- \& B^-] \vdash E : C^+} \quad \& L_1 \\
\frac{\Gamma; \Delta, [\pi_2 R : B^-] \vdash E : C^+}{\Gamma; \Delta, [R : A^- \& B^-] \vdash E : C^+} \quad \& L_2 \\
\frac{\Gamma; \Delta, p : A^+ \vdash E : C^+}{\Gamma; \Delta, [R : \uparrow A^+] \vdash \text{let } \{p\} = R \text{ in } E : C^+} \quad \uparrow L
\]

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.

\[
\begin{align*}
\frac{}{\Gamma ; \Delta \vdash [E : C^+]} & \quad \text{foc}^+ \quad \frac{}{\Gamma ; x : p^+ \vdash [x : p^+] \text{id}^+} \\
\frac{\Gamma ; \Delta \vdash [V : A]}{\Gamma ; \Delta \vdash [W : B]} & \quad \frac{}{\Gamma ; \Delta, \Delta' \vdash [(V, W) : A \otimes B] \otimes R} \\
\frac{\Gamma \vdash [V_0 : A_0^+]}{\Gamma ; \Delta \vdash [W : [V_0/x]B^+]} & \quad \frac{}{\Gamma ; \Delta \vdash [(V_0, W) : \exists x : A_0^+ \cdot B^+] \exists R} \\
\end{align*}
\]

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

\[
\frac{}{\Gamma, x : p_0^+ ; \vdash [x : p_0^+] \text{id}^+}
\]

## 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.

<table>
<thead>
<tr>
<th>Abstract Syntax</th>
<th>Type</th>
<th>Concrete Syntax</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Term</td>
<td>Term</td>
<td>Term</td>
<td>Type</td>
</tr>
<tr>
<td>$M ::= \lambda p. M$</td>
<td>$A^+ \rightarrow B^-$</td>
<td>$\lambda p. M$</td>
<td>A $\rightarrow$ B</td>
</tr>
<tr>
<td>$\lambda x. M$</td>
<td>$B$ $: A$ $\rightarrow \exists x. M$</td>
<td>$x$ $: A$ $\rightarrow B$</td>
<td></td>
</tr>
<tr>
<td>$\langle M, N \rangle$</td>
<td>$A \land B$</td>
<td>$\langle M, N \rangle$</td>
<td>A $\land$ B</td>
</tr>
<tr>
<td>${E}$</td>
<td>$\uparrow A^+$</td>
<td>${E}$</td>
<td>A</td>
</tr>
<tr>
<td>$p ::= x$</td>
<td>$p^+$</td>
<td>$x$</td>
<td></td>
</tr>
<tr>
<td>$[p, q]$</td>
<td>$A^+ \otimes B^+$</td>
<td>$[p, q]$</td>
<td>A $\times$ B</td>
</tr>
<tr>
<td>$[]$</td>
<td>$1$</td>
<td>$1$</td>
<td></td>
</tr>
<tr>
<td>$[x, p]$</td>
<td>$\exists x: A^+ B^+$</td>
<td>$[x, p]$</td>
<td>Exists $x: A$. B</td>
</tr>
<tr>
<td>$R ::= c$</td>
<td>$c: A^- \in \Sigma$</td>
<td>$c$</td>
<td></td>
</tr>
<tr>
<td>$x$</td>
<td>$x: A^- \in \Delta$</td>
<td>$x$</td>
<td></td>
</tr>
<tr>
<td>$RV$</td>
<td>$B^+ \rightarrow A^-$</td>
<td>$RV$</td>
<td>A $\rightarrow$ B</td>
</tr>
<tr>
<td>$\pi_1 R</td>
<td>\pi_2 R$</td>
<td>$A \land B$</td>
<td>$#1 R</td>
</tr>
<tr>
<td>$RV_0$</td>
<td>$B$</td>
<td>$RV_0$</td>
<td>!V</td>
</tr>
<tr>
<td>$V ::= x$</td>
<td>$p^+$</td>
<td>$x$</td>
<td></td>
</tr>
<tr>
<td>$[V, W]$</td>
<td>$A^+ \otimes B^+$</td>
<td>$[V, W]$</td>
<td>A $\times$ B</td>
</tr>
<tr>
<td>$[]$</td>
<td>$1$</td>
<td>$1$</td>
<td></td>
</tr>
<tr>
<td>$[V_0, W]$</td>
<td>$\exists x: A^+ B^+$</td>
<td>$[V_0, W]$</td>
<td>Exists $x{:} A$. B</td>
</tr>
<tr>
<td>$E ::= \text{let } {p} = R \text{ in } E$</td>
<td>left focus</td>
<td>let ${p} = R \text{ in } E$</td>
<td>A</td>
</tr>
<tr>
<td>$V$</td>
<td>right focus</td>
<td>V</td>
<td></td>
</tr>
</tbody>
</table>

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 $\times$ d $\times$ n $\rightarrow$ { q }.
q2d : q $\rightarrow$ { d $\times$ d $\times$ n }.

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

\[
\text{example1} : n \times n \times n \rightarrow o \{d \times n\} = \\
\{ [n1, [n2, n3]]. \{ \quad \% n1:n, n2:n, n3:n \mid - _ : d \times n \\
\quad \text{let} \{d1\} = n2d [n1, n3] \text{ in} \quad \% d1:n, n2:n \mid - _ : d \times n \\
\quad \quad \{d1, n2\} \quad \% d1:n, n2:n \mid - _ : d \times n \\
\}.
\]

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.

\[
\text{Query} (11, 1, *, 1) (n \times (n \times n)) \rightarrow o \{d \times n\}. \\
\text{Solution:} \quad \{ [X1, [X2, X3]]. \{ \quad \% n1:n, X2:n, X3:n \mid - _ : d \times n \\
\quad \text{let} \{X4\} = n2d [X1, X3] \text{ in} \quad \% d1:n, X4:n \mid - _ : d \times n \\
\quad \text{let} \{X5, X6\} = d2n X4 \text{ in} \\
\quad \text{let} \{X7\} = n2d [X2, X5] \text{ in} \\
\quad \text{let} \{X8, X9\} = d2n X7 \text{ in} \\
\quad \text{let} \{X10\} = n2d [X8, X9] \text{ in} \\
\quad \text{let} \{X11, X12\} = d2n X10 \text{ in} \\
\quad \text{let} \{X13\} = n2d [X6, X11] \text{ in} \\
\quad \text{let} \{X14, X15\} = d2n X13 \text{ in} \\
\quad \text{let} \{X16\} = n2d [X12, X14] \text{ in} \\
\quad \text{let} \{X17, X18\} = d2n X16 \text{ in} \\
\quad \text{let} \{X19\} = n2d [X15, X17] \text{ in} [X19, X18] \}.
\]

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.

```plaintext
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 \texttt{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.
The Concurrent Logical Framework

References


