What Would A Polarized Logical Framework Be Good For?

Request For Logic (RFL) #2
Robert J. Simmons
October 9, 2009 (Revision 2)

In my note "Mismatch II," I noted that the Ordered Logical Framework could not encode the sequent calculus for ordered logic, and proposed one solution to the problem: a polarized logical framework. In this note I expand on this idea, giving a sketch of a (not fully dependent) polarized logical framework for linear logic and describing the adequacy of an encoding of propositional linear logic in this framework. Interestingly, we do not have, nor do we want, any of the concurrent equality so important to the (also polarized) logical framework CLF [KW].

1 Sketch of a Polarized Linear Logical Framework

In this section, we give the outlines of a polarized logical framework (where the logic is polarized in the sense of [NZ]) for linear logic. We'll ignore the syntactic category "m", which we treat as a canonical form from the simply typed lambda calculus (where simple types come from the syntactic category "a"), and we will treat the indices to types very informally; the treatment of terms is adequately considered elsewhere. In particular, there is a separate persistent context of individuals Ψ that never changes, and so we ignore it except in the rare instances where it matters, for instance in rules for $\forall$.

Notably missing from this framework is unrestricted implication and exponentials, merely because I don't happen to need them.

Canonical terms $m ::= \ldots$
Atomic types $Q^+, Q^- ::= \ldots$

Positive types $A^+ ::= Q^+ \mid {\downarrow}A^- \mid 0 \mid A^+ \oplus B^+ \mid 1 \mid A^+ \otimes B^+$
Negative types $A^- ::= Q^- \mid {\uparrow}A^+ \mid \forall x{:}a.A^- \mid A^+ \multimap B^- \mid \top \mid A \mathbin{\&} B$

Expressions $M ::= .T \mid x^- \cdot S \mid c \cdot S$
Positive terms $T ::= x^+ \mid .M \mid «» \mid «T_1, T_2» \mid \mathrm{inl}\ T \mid \mathrm{inr}\ T$
Negative terms $N ::= .R \mid \lambda x.N \mid \lambda P.N \mid \langle\,\rangle \mid \langle N_1, N_2 \rangle$

Spines $S ::= \circ \mid P \Rightarrow R \mid m, S \mid T; S \mid \pi_1; S \mid \pi_2;\ S$
Patterns $P ::= x^+ \mid .x^- \mid [\,] \mid [P \mid P] \mid «» \mid «P,\ P»$
Arms $R ::= [\,] \mid [R \mid R] \mid .M$

Let's look at rules. The ones for positive terms and negative terms, which correspond to right focus and inversion, aren't too bad, though the fact that we need two rules for where negative terms "bottom out" to arms is a little bit disappointing.

1.1 Positive Terms (Right focus)

$$\frac{}{x^{+}{:}Q^{+} \vdash x^{+} : [Q^{+}]}, \qquad \frac{\Gamma \vdash M : A^{-}}{\Gamma \vdash .M : [{\downarrow}A^{-}]}, \qquad \frac{}{\cdot \vdash «» : 1}$$

$$\frac{\Gamma \vdash T_1 : [A^{+}] \qquad \Gamma \vdash T_2 : [B^{+}]}{\Gamma, \Delta \vdash «T_1,T_2» : [A^{+} \otimes B^{+}]},$$

$$\frac{\Gamma \vdash T : [A^{+}]}{\Gamma \vdash \mathrm{inl}\ T : [A^{+} \oplus B^{+}]}, \qquad \frac{\Gamma \vdash T : [B^{+}]}{\Gamma \vdash \mathrm{inr}\ T : [A^{+} \oplus B^{+}]}$$

1.2 Negative Terms (Right inversion)

$$\frac{\Psi, x{:}a; \Gamma; \Omega \vdash N : A^{-}}{\Psi; \Gamma; \Omega \vdash \lambda x.N : \forall x{:}a.A^{-}} \qquad \frac{\Gamma; \Omega, P{:}A^{+} \vdash N : B^{-}}{\Gamma; \Omega \vdash \lambda P.N : A^{+} \multimap B^{-}}, \qquad \frac{}{\Gamma; \Omega \vdash \langle\,\rangle : \top},$$

$$\frac{\Gamma; \Omega \vdash N_1 : A^{-} \qquad \Gamma; \Omega \vdash N_2 : B^{-}}{\Gamma; \Omega \vdash \langle N_1,N_2 \rangle : A^{-}\ \&\ B^{-}}, \qquad \frac{\Gamma; \Omega \vdash R : Q^{-}}{\Gamma; \Omega \vdash .R : Q^{-}}, \qquad \frac{\Gamma; \Omega \vdash R : A^{+}}{\Gamma; \Omega \vdash .R : {\uparrow}A^{+}}$$

1.3 Expressions (Neutral sequent/Begin focus)

The rules for expressions just reveal how we can get started in the first place - one of those ways is by focusing on a negative proposition in the context, the other is focusing on a negative proposition in the signature.

$$\frac{\Gamma \vdash T : [A^{+}]}{\Gamma \vdash .T : A^{+}}, \qquad \frac{\Gamma[A^{-}] \vdash S : C}{\Gamma, x^{-}{:}A^{-} \vdash x^{-}{\cdot}S : C}, \qquad \frac{c{:}A^{-} \in \Sigma \qquad \Gamma[A^{-}] \vdash S : C}{\Gamma \vdash c{\cdot}S : C}$$

1.4 Spines (Left focus)

Spines are series of reductions - the negative formula "in focus" is being taken from a negative formula down to something else. If that something else is a positive formula, then we go to a pattern, but if that something else is a positive atomic proposition, then we have to finish.

$$\frac{\Psi \vdash m : a \qquad \Psi; \Gamma[A^{-}] \vdash S : C}{\Psi; \Gamma[\forall x{:}a.A^{-}] \vdash (m;\ S) : C}, \qquad \frac{\Gamma \vdash T : [A^{+}] \qquad \Delta[B^{-}] \vdash S : C}{\Gamma,\Delta[A^{+} \multimap B^{-}] \vdash (T;\ S) : C},$$

$$\frac{\Gamma[A^-] \vdash S : C}{\Gamma[A^- \mathbin{\&} B^-] \vdash \pi_1\ S : C}, \quad \frac{\Gamma[B^-] \vdash S : C}{\Gamma[A^- \mathbin{\&} B^-] \vdash \pi_1\ S : C}, \quad \frac{}{\Gamma[Q^-] \vdash \circ : Q^-}$$

$$\frac{\Gamma;\ P{:}A^+ \vdash R : C}{\Gamma[\uparrow A^+] \vdash P{=>}R : C}$$

1.5 Patterns and Arms (Left inversion)

Finally, we have left inversion, which is messy. This notation is closest to Krishnaswami's in [NK].

$$\frac{\Gamma,\ x^+{:}Q^+;\ \Omega \vdash R : C}{\Gamma;\ x^+{:}Q^+,\ \Omega \vdash R : C}, \quad \frac{\Gamma,\ x^-{:}A^-;\ \Omega \vdash R : C}{\Gamma;\ .x^-{:}\downarrow A^-,\ \Omega \vdash R : C}, \quad \frac{}{\Gamma;\ []{:}0,\ \Omega \vdash R : C}$$

$$\frac{\begin{array}{l}\Gamma;\ P_1{:}A^+,\ \Omega \vdash R_1 : C \\ \Gamma;\ P_2{:}B^+,\ \Omega \vdash R_2 : C\end{array}}{\Gamma;\ [P_1|P_2]{:}A^+ \oplus B^+,\ \Omega \vdash [R_1 \mid R_2] : C}, \quad \frac{\Gamma;\ \Omega \vdash R : C}{\Gamma;\ \text{«»}{:}1,\ \Omega \vdash R : C},$$

$$\frac{\Gamma;\ P_1{:}A^+,\ P_2{:}B^+,\ \Omega \vdash R : C}{\Gamma;\ \text{«}P_1,P_2\text{»}{:}A^+ \otimes B^+,\ \Omega \vdash R : C}, \quad \frac{\Gamma \vdash M : C}{\Gamma;\ \cdot \vdash .M : C}$$

1.6 Alpha Equivalence, But No Concurrent Equality

Note that, other than an appropriate notion of alpha-equivalence on variables and patterns, this is really what we want - whereas the example of CLF might lead us to expect that the following two (partial) proofs should be equated, that will not be the behavior here.

$$\begin{array}{l}
\qquad\qquad M \\
\qquad\qquad \vdots \\
\hline
x_1^+{:}Q_1^+,\ x_2^+{:}Q_2^+,\ x_3^+{:}Q_3^+,\ x_4^+{:}Q_4^+ \vdash C \\
\hline
\quad \text{... left focus on } x_{34}^- \text{ ...} \\
\hline
x_1^+{:}Q_1^+,\ x_2^+{:}Q_2^+,\ x_{34}^-{:}\uparrow(Q_3^+ \otimes Q_4^+) \vdash C \\
\hline
\quad \text{... left focus on } x_{12}^- \text{ ...} \\
\hline
x_{12}^-{:}\uparrow(Q_1^+ \otimes Q_2^+),\ x_{34}^-{:}\uparrow(Q_3^+ \otimes Q_4^+) \vdash C
\end{array}$$

Proof term: $(x_{12}^- \cdot (\text{«}x_1^+, x_2^+\text{»} => .(x_{34}^- \cdot (\text{«}x_3^+, x_4^+\text{»} => .M))))$

```
               M
               ⋮
==========================================
$x_1^+:Q_1^+$, $x_2^+:Q_2^+$, $x_3^+:Q_3^+$, $x_4^+:Q_4^+$ ⊢ C
==========================================
  ... left focus on $x_{34}^-$ ...
==========================================
$x_{12}^-:\uparrow(Q_1^+ \otimes Q_2^+)$, $x_3^+:Q_3^+$, $x_4^+:Q_4^+$ ⊢ C
==========================================
  ... left focus on $x_{12}^-$ ...
==========================================
$x_{12}^-:\uparrow(Q_1^+ \otimes Q_2^+)$, $x_{34}^-:\uparrow(Q_3^+ \otimes Q_4^+)$ ⊢ C

Proof term: ($x_{34}^-$·(«$x_3^+$,$x_4^+$» => .($x_{12}^-$·(«$x_1^+$,$x_2^+$» => .M))))
```

The proofs are distinct (even if we assume that the omitted rest of the proof, represented by M, is the same in both case).

2 Encoding (Unpolarized) Linear Logic in the Framework

We can give the following signature for a sequent calculus for linear logic. Note that for any proposition A, "hyp A" is a positive atomic proposition and "conc A" is a negative atomic proposition.

```
o : type.
atom : type.
hyp : o -> $type^+$.
conc : o -> $type^-$.

atm   : atom -> o.
one   : o.
and   : o -> o -> o.
top   : o.
with  : o -> o -> o.
zero  : o -> o -> o.
or    : o -> o -> o.
imp   : o -> o -> o.

init  : ∀Q:atom. hyp(atm Q) -o conc(atm Q).

oneR  : conc(one).
oneL  : hyp(one) -o ↑1.
andR  : ∀A:o.∀B:o. ↓conc(A) ⊗ ↓conc(B) -o conc(and A B).
andL  : ∀A:o.∀B:o. hyp(and A B) -o ↑(hyp(A) ⊗ hyp(B)).

topR  : conc(top) & ⊤.
%% no topL
withR : ∀A:o.∀B:o. ↓(conc(A) & conc(B)) -o conc(with A B).
withL : ∀A:o.∀B:o. hyp(with A B) -o ↑hyp(A) & ↑hyp(B).
```

```
%% no zeroR
zeroL : hyp(zero) -o ↑0.
orR   : ∀A:o.∀B:o. ↓conc(A) ⊕ ↓conc(B) -o conc(or A B).
orL   : ∀A:o.∀B:o. hyp(or A B) -o ↑(hyp A ⊗ hyp B).

impR  : ∀A:o.∀B:o. ↓(hyp(A) -o conc(B)) -o conc(imp A B).
impL  : ∀A:o.∀B:o. hyp(imp A B) ⊗ ↓conc(A) -o ↑hyp(B).
```

2.1 Stating Adequacy

The adequacy theorem goes in two directions. We assume an a priori proof of adequacy for the internal term language that gives a preexisting bijection between "A ⊗ B" and "⌞A ⊗ B⌟", i.e. "and ⌞A⌟ ⌞B⌟", where ⌞A⌟ is the function giving the adequate encoding of the linear logic proposition A into the simply typed lambda calculus. As mentioned above, ⌞A ⊗ B⌟ = and ⌞A⌟ ⌞B⌟, ⌞Q⌟ = atm Q, ⌞⊤⌟ = top, and so on; there is also an implicit relation set up between derivations and encodings of derivations that we're glossing over.

We will take advantage of this and a presumed notion of subordination-based transport of adequacy as in [RH] to freely work between linear logic propositions and their encodings in the (informal) propositions below.

Proposition 1: Adequacy of the linear logic encoding

1) If there exists a term M in the given signature such that
 Q1:i,…,Qn:i; x1:hyp A1,…,xn:hyp Ak ⊢ M : conc C
then there exists a unique linear logic sequent calculus derivation of
 A1,…,Ak ⊢ C

2) If there exists a linear logic sequent calculus derivation of
 A1,…,Ak ⊢ C
that includes as subterms the atomic propositions Q1,…,Qn, then there exists a unique term M (up to alpha equality) in the given signature such that
 Q1:i,…,Qn:i; x1:hyp A1,…,xn:hyp Ak ⊢ M : conc C

Together, this proof establishes a bijection between derivations "on paper" and terms in the framework, which is the basis of adequacy.

2.2 Sketching Adequacy

To give just an example of how this proof would work, we'll give the case for impL. In the first direction we have this, because we have chosen to focus on impL in the signature:

 Q1:i,…,Qn:i; x1:hyp A1,…,xn:hyp Ak ⊢ impL·S : conc C

By repeated application of inversion we get the following derivation, where the Q1:i,…,Qn:i is omitted and Γ = ΔA ⋈ Δ ⋈ hyp(imp A B) = x1:hyp A1,…,xn:hyp Ak

```
                                                   ΔA ⊢ conc A
                                                 ===============
                                                 ΔA;  · ⊢ conc A    Δ,hyp(B) ⊢ conc C
                                                 ===============    =====================
                                                 ΔA;  · ⊢ conc A    Δ,hyp(B); · ⊢ conc C
 =============================                   ===============    =====================
 hyp(imp A B) ⊢ [hyp(imp A B)]                   ΔA ⊢ [↓conc A]     Δ; hyp(B) ⊢ conc C
 ================================================================   =====================
 ΔA, hyp(imp A B) ⊢ [hyp(imp A B) ⊗ ↓conc(A)]                       Δ[↑hyp(B)] ⊢ conc C
 =========================================================================================
 Γ [hyp(imp A B) ⊗ ↓conc(A) -o ↑hyp(B)] ⊢ conc C
 ======================================================
 Γ [∀B:o. hyp(imp A B) ⊗ ↓conc(A) -o ↑hyp(B)] ⊢ conc C
 ============================================================
 Γ [∀A:o.∀B:o. hyp(imp A B) ⊗ ↓conc(A) -o ↑hyp(B)] ⊢ conc C
 ============================================================
 Γ ⊢ conc C
```

The corresponding proof term is

```
 impL·(A; B; «xi,..MA»; x=>.MC)
```

Where we have for MA a unique corresponding proof in linear logic that ⌞ΔA⌟ ⊢ A, and for MC a unique corresponding proof that ⌞Δ,hyp(B)⌟ ⊢ C. Our translation should uniquely translate this proof term to ⌞Γ⌟ ⊢ C using the ⊸L rule.

In the other direction, we have the derivation

```
    D1          D2
 ΔA ⊢ A    Δ,B ⊢ C
 ================== ⊸L
   Δ,ΔA,B ⊢ C
```

By induction we get proof terms corresponding to D_1 (M_1) and D_2 (M_2), and show that our translation relation uniquely associates those terms with the proof term impL·(A; B; «xi,..M_1»; x=>.M_2).

3 Comparison with a CLF encoding.

We can give (most) of the same encoding that we gave above in the concurrent logical framework CLF. This is written for the Celf implementation, version 2.4 [AS].

```
 o : type.
 atom : type.
 hyp : o -> type.
 conc : o -> type.

 atm   : atom -> o.
 one   : o.
 and   : o -> o -> o.
 with  : o -> o -> o.
 imp   : o -> o -> o.

 init  : hyp(atm Q) -o conc(atm Q).

 oneR  : conc one.
 oneL  : hyp one -o {1}.
 andR  : {conc A} * {conc B} -o conc(and A B).
 andL  : hyp(and A B) -o {hyp A * hyp B}.

 withR : {conc A} & {conc B} -o conc(with A B).
 withL : hyp(with A B) -o {hyp A} & {hyp B}.

 impR  : (hyp A -o {conc B}) -o conc(imp A B).
 impL  : hyp(imp A B) * {conc A} -o {hyp B}.
```

First, we'll discuss what we had to (or chose to) leave out, then we will discuss how we changed the encoding, and finally we'll discuss the role of concurrent equality.

We'll also throw in some atoms to our signature in order to make life a little easier for ourselves:

```
 a : atom. qA : o = atm a.
 b : atom. qB : o = atm b.
 c : atom. qC : o = atm c.
 d : atom. qD : o = atm d.
```

We'll also use throughout a judgment to check that two proofs are equal, which will come up frequently in our discussion of equality:

```
 id : conc C -> conc C -> type.
 refl : id D D.
```

3.1 Omissions

The CLF encoding itself does not have support for 0 (impossibility) or ⊕ (disjunction), and this is carried over from the fact that CLF itself has neither 0 nor ⊕. There doesn't seem to be any way to remedy this without changing the framework.

⊤, the unit of additive conjunction, is also missing, just as ⊤ itself was left out of the redesign of CLF due to complications that arose in the implementation due to its presence. There is a non-adequate way to encode

top in our setting, just create a proposition "faketop" that allows us to
assume anything we want.

```
 top : o.
 faketop : type.
 faketop/z : faketop.
 faketop/s : faketop o- hyp A * faketop.
 topR  : faketop -o conc(top).
```

However, this is not an adequate, because now there will be six proofs of
the encoded linear logic sequent A, B, C ⊢ ⊤ where there is only supposed
to be one.

```
 pfABC : conc (imp qA (imp qB (imp qC top))) =
   impR \xA.{(impR \xB. {(impR \xC. {(
    topR (faketop/s (faketop/s (faketop/s faketop/z xA) xB) xC)
   )})})}.

 pfACB : conc (imp qA (imp qB (imp qC top))) =
   impR \xA.{(impR \xB. {(impR \xC. {(
    topR (faketop/s (faketop/s (faketop/s faketop/z xA) xC) xB)
   )})})}.

 pfBAC : conc (imp qA (imp qB (imp qC top))) =
   impR \xA.{(impR \xB. {(impR \xC. {(
    topR (faketop/s (faketop/s (faketop/s faketop/z xB) xA) xC)
   )})})}.
```

Maybe this is a problem, maybe it isn't; as we'll see below the encoding
without ⊤ is already not adequate, but for a different reason - there are
two few distinct CLF terms, as opposed to the problem here where there are
too many.

However, for now, we can use the concurrent equality that we will later
see as a problem by giving a different notion of top that allows us to
gather hypotheses out of the context in any arbitrary order.

```
 top : o.
 faketop : type.
 faketop/get : faketop -> hyp A -o {1}.
 topR  : (faketop -> {1}) -o conc(top).
```

Two of the six proofs of A,B,C ⊢ ⊤ which look different are below:

```
 pfABC : conc (imp qA (imp qB (imp qC top))) =
   impR \xA.{(impR \xB. {(impR \xC. {(
    topR (\!ft. {(
      let {1} = faketop/get ft xA in
      let {1} = faketop/get ft xB in
      let {1} = faketop/get ft xC in
      1)})
   )})})}.
 pfCAB : conc (imp qA (imp qB (imp qC top))) =
   impR \xA.{(impR \xB. {(impR \xC. {(
    topR (\!ft. {(
      let {1} = faketop/get ft xC in
      let {1} = faketop/get ft xA in
      let {1} = faketop/get ft xB in
      1)})
   )})})}.
```

However, while the lets are in a different order, the concurrent equality of CLF declares that we will treat as indistinguishable the two proofs because we can permute one to the other without difficulty.

```
 proofs_are_equal : id pfABC pfCAB = refl.
```

3.2 How The Monad Lets Us Replace Polarity

CLF only has negative atomic propositions, so that we are unable to use the technique that we used to great effect in the previous encoding, which was making atomic propositions "hyp A" positive, so that a premise "hyp A" could only be satisfied when the proposition was immediately available in the context.

We can get a similar effect from CLF by only allowing conclusions of the form "hyp A" under a monad. Then, if we have a premise "hyp A," because that premise is negative and non-monadic, we are only allowed to left focus if that left focus allows us to prove that "hyp A" is true. But the property that we only allow "hyp A" to be a lax conclusion means this is impossible.

Roughly, we need a lax monad in the CLF encoding whenever we needed a polarity shift in the first encoding. The only difference is when we have a downshift ↓ in a premise, such as the premise to impR, ↓(hyp A -o conc B). The focusing behavior of this premise is to enter right inversion, add hyp A to the context, and then try to prove conc B from a neutral sequent that allows left or right focus. In our setting, the ability to prove conc C from a neutral sequent that allows left or right focus requires us to be proving a lax goal - therefore, the premise translates to (hyp A -o {conc B}) in the CLF encoding.

3.3 Concurrent Equality Causes Problems

The biggest issue with this CLF encoding is that our encoding of left rules is not adequate. In particular, the example from 1.6 of two proofs that we wanted to be different is precisely what is violated. Let's look at two proofs of $((A \otimes B) \multimap (C \otimes D) \multimap ((A \otimes C) \otimes (B \otimes D))$ that seemingly correspond to two different ways of proving this sequent.

```
    ⋮
A, B, C, D ⊢ (A ⊗ C) ⊗ (B ⊗ D)
============================================= ⊗L
A, B, C ⊗ D ⊢ (A ⊗ C) ⊗ (B ⊗ D)
============================================= ⊗L
A ⊗ B, C ⊗ D ⊢ (A ⊗ C) ⊗ (B ⊗ D)
============================================= ⊸R
A ⊗ B ⊢ (C ⊗ D) ⊸ ((A ⊗ C) ⊗ (B ⊗ D))
============================================= ⊸R
⊢ (A ⊗ B) ⊸ (C ⊗ D) ⊸ ((A ⊗ C) ⊗ (B ⊗ D))

pf1 :
  conc (imp (and qA qB)
        (imp (and qC qD)
         (and (and qA qC) (and qB qD)))) =
   impR \xAB.{(impR (\xCD. {
   let {[xA,xB]} = andL xAB in
   let {[xC,xD]} = andL xCD in
   (andR [{(andR [{(init xA)},{(init xC)}])},
          {(andR [{(init xB)},{(init xD)}])}])}
   ))}.

    ⋮
A, B, C, D ⊢ (A ⊗ C) ⊗ (B ⊗ D)
============================================= ⊗L
A ⊗ B, C, D ⊢ (A ⊗ C) ⊗ (B ⊗ D)
============================================= ⊗L
A ⊗ B, C ⊗ D ⊢ (A ⊗ C) ⊗ (B ⊗ D)
============================================= ⊸R
A ⊗ B ⊢ (C ⊗ D) ⊸ ((A ⊗ C) ⊗ (B ⊗ D))
============================================= ⊸R
⊢ (A ⊗ B) ⊸ (C ⊗ D) ⊸ ((A ⊗ C) ⊗ (B ⊗ D))

pf2 :
  conc (imp (and qA qB)
        (imp (and qC qD)
         (and (and qA qC) (and qB qD)))) =
   impR \xAB.{(impR (\xCD. {
   let {[xC,xD]} = andL xCD in
   let {[xA,xB]} = andL xAB in
   (andR [{(andR [{(init xA)},{(init xC)}])},
          {(andR [{(init xB)},{(init xD)}])}])}
   ))}.
```

Because the left rule acting on A ⊗ B and the left rule acting on C ⊗ D act on independent parts of the context, the concurrent equality of CLF declares that these two proofs, which in our "on paper" encoding are different, are really the same.

```
proofs_are_equal_12 : id pf1 pf2 = refl.
```

However, the final remaining proof of the formula will be treated as a distinct proof by CLF.

```
     ⋮
 A, B, C, D ⊢ (A ⊗ C) ⊗ (B ⊗ D)
 ============================================ ⊗L
 A, B, C ⊗ D ⊢ (A ⊗ C) ⊗ (B ⊗ D)
 ============================================ ⊸R
 A, B ⊢ (C ⊗ D) ⊸ ((A ⊗ C) ⊗ (B ⊗ D))
 ============================================ ⊗L
 A ⊗ B ⊢ (C ⊗ D) ⊸ ((A ⊗ C) ⊗ (B ⊗ D))
 ============================================ ⊸R
 ⊢ (A ⊗ B) ⊸ (C ⊗ D) ⊸ ((A ⊗ C) ⊗ (B ⊗ D))
```

```
 pf3 : conc (imp (and qA qB)
         (imp (and qC qD)
          (and (and qA qC) (and qB qD)))) =
    impR \xAB.{
    let {[xA,xB]} = andL xAB in
    (impR (\xCD. {
    let {[xC,xD]} = andL xCD in
    (andR [{(andR [{(init xA)},{(init xC)}])},
           {(andR [{(init xB)},{(init xD)}])}])}))}.
```

The equating of the seemingly distinct first two proofs is problematic, because it means that our encoding is not adequate - there is not a bijection between CLF terms and on-paper proofs. Exactly the property that saved us from having too many proofs when we were dealing with the encoding of ⊤ now means that we have too few.

3.4 Potential Remedies

There are a few remedies, some speculative and some more-or-less standard, which we will now consider as we conclude.

One solution is that we can modify the encoding to pass around a linear token "tok", which is a previously recognized way of forcing a concurrent computation to act sequential.

```
 tok : type.
 oneR  : tok -o conc one.
 oneL  : tok * hyp one -o {tok}.
 andR  : tok * (tok -o {conc A}) * (tok -o {conc B}) -o conc(and A B).
 andL  : tok * hyp(and A B) -o {tok * hyp A * hyp B}.

 withR : tok * (tok -o ({conc A} & {conc B})) -o conc(with A B).
 withL : tok * hyp(with A B) -o {tok * hyp A} & {tok * hyp B}.

 impR  : tok * (tok * hyp A -o {conc B}) -o conc(imp A B).
 impL  : tok * hyp(imp A B) * (tok -o {conc A}) -o {tok * hyp B}.
```

Then, instead of trying to prove (conc C), we will try to prove (tok -o conc C). The repeated consumption and re-introduction of the token prevents any application of concurrent equality, because no two rule applications can ever be independent if they are competing for a single token.

This is a relatively ugly solution, though - as you can see, it has made the framework much less pleasing.

Another solution would be to remove concurrent equality from CLF, but this would take away many of the interesting benefits that drew us to CLF in the first place. In our example, we can see that it would prevent us from making an adequate encoding for ⊤, for instance.

A third approach would be to add the polarized connectives and atomic propositions to CLF without giving them any notion of concurrent equality.

A final approach would be two modify the object language by introducing the same idea of concurrent equality that was imposed by CLF to linear logic. This would mean that we would say that there were, in fact, only two proofs of $(A \otimes B) \multimap (C \otimes D) \multimap ((A \otimes C) \otimes (B \otimes D))$, the first of which corresponds to both pf1 and pf2 (as those two proofs are equivalent), and the second of which corresponds to pf3. The correspondence between this suggestion and Miller et al.'s work on multi-focusing may deserve investigation [KC].

4 References

[AS] A. Schack-Nielsen and C. Schürmann, "Celf – a logical framework for deductive and concurrent systems (system description)," 2008, pp. 320-326. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-71070-7_28

[KC] K. Chaudhuri, D. Miller, and A. Saurin, "Canonical sequent proofs via multi-focusing," 2008, pp. 383-396. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-09680-3_26

[KW] K. Watkins, I. Cervesato, F. Pfenning, and D. Walker, "A concurrent logical framework: The propositional fragment," 2004, pp. 355-377. [Online]. Available: http://www.springerlink.com/content/fwlrpm102djtlaeb

[NK] N. R. Krishnaswami, "Focusing on pattern matching," in POPL '09: Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages. New York, NY, USA: ACM, 2009, pp. 366-378. [Online]. Available: http://dx.doi.org/10.1145/1480881.1480927

[NZ] N. Zeilberger, "The logical basis of evaluation order and pattern-matching," Ph.D. dissertation, Carnegie Mellon University, April 2009. [Online]. Available: http://reports-archive.adm.cs.cmu.edu/anon/2009/abstracts/09-122.html

[RH] R. Harper and D. R. Licata, "Mechanizing metatheory in a logical framework," Journal of Functional Programming, vol. 17, no. 4-5, pp. 613-673, 2007. [Online]. Available: http://dx.doi.org/10.1017/S0956796807006430