Encoding Substructural Logics in a Hybrid Framework
Request For Logic (RFL) #3

Robert J. Simmons

November 22, 2009

In this note, I take a page out of Reed and Pfenning's "constructive approach to the resource semantics of substructural logics" and make a simpler observation than the one they made; that a hybrid logical framework with flexible equations on worlds can be used to encode a large number of substructural logics.

1 Sketching the necessary hybrid logical framework

I must stress that this is a sketch; without the metatheory of this system worked out, it is likely that there would be substantial surprises, and even without surprises there are substantial implementation issues to consider. However, the logic I am considering here is quite similar both to Reed and Pfenning's FF [RP] and Reed's HLF [JR], so there is some hope that this is a reasonable approach.

It's also worth making clear that I have a vastly less ambitious goal than Reed and Pfenning's constructive resource semantics for substructural logics [RP]. We want to explore the properties of a hybrid logic where we can *encode* various substructural logics. Reed and Pfenning were *embedding* substructural logics in a hybrid logic, which is a substantially different enterprise and the topic of another note. Another way to explain the difference is that Reed encodes logics and deductive systems in the linear logical framework LLF and then uses HLF to reason about it; I'm just using a variant of HLF to encode some logics.

Types are similar to what we would expect in any first-order logical framework, except that atomic propositions are always annotated with a world, i.e. "Q @ p." Intuitively, we will use the world "p" to tell us about the structure of the context that the atomic type makes sense within. There are three sorts of things: types/propositions, individuals, and worlds; just as we have different sorts of individuals, we will allow different sorts of worlds when helpful. Individual variables are "$x_1, x_2, \ldots$" and world variables are "$\alpha, \beta, \ldots$"

$$A ::= \forall\alpha.A \mid \forall x.A \mid A \rightarrow A \mid Q@w$$

Worlds "$p, q, \ldots$" will be defined in each signature; if there are no constant worlds then the only way to create them is to assume them. In other words, in the signature containing only the type declaration "a : type," there are no closed atomic types, since there aren't any worlds that can be associated with them. However, there is a type "$\forall\alpha.\ a@\alpha \rightarrow a@\alpha$" whose sole inhabitant is "$\lambda\alpha.\lambda x.x$." The "non-hybrid" version of the logic should be recoverable by having a single constant "ε : w" defined in the signature and annotating every atomic type with "@ε".

1.1 Worlds: "tacking on" and "framing off"

I want to make the argument that there are two fundamental and complementary operations on worlds: "tacking on" and "framing off." The "tacking on" operation describes how a new world "α" can be combined with an existing world "p." In the linear logic formulation of Reed's thesis [JR], this was exclusively done with the associative, commutative operation "$*$" (with unit "ε"). For instance, the encoding of the implication-right rule is as follows:

$$\multimap R : (\forall\alpha.\ hyp(A)@\alpha \rightarrow conc(C)@(p*\alpha)) \rightarrow conc(A \multimap C)@p$$

The notion of “framing off” was done very explicitly in [RP], where explicit frames were introduced that, combined with axioms like “(f ⊛ p) ◃ q ≡ f ◃ (p * q)” ensured that the frames “f” acted like one of Huet’s zippers over the structure of the world [GH]. I’ll take the arguably less formal approach taken by [NB], where frames “p[-]” are just worlds p with a distinguished variable that come with an obvious notion of filling the hole “p[q]” that is implemented by substituting “q” for the distinguished variable in “p.”

This informal simplification is not without cost. In particular, we would seem to need to require that the equivalence relation be such that if “p ≡ q,” and “p” has a single distinguished variable, then “q” also has a single distinguished variable. This means that it’s generally going to be impossible to use equalities on worlds to describe weakening rules like “p⊑p*q” that Reed and Pfenning explored in their discussion of bunched logic.

2 Linear logic and queue logic

2.1 HLF-style linear logic

What are the fundamental “tacking on” and “framing off” operations of linear logic? Well, Reed’s thesis certainly provides one extremely reasonable answer. The “tacking on” operation is the resource combination operation “*.”

```
 p ::= ε | α | p * q
 ε * p ≡ p
 p * q ≡ q * p
 p * (q * r) ≡ (p * q) * r
```

Because of the associative and commutative nature of *, it is always the case that p[q] ≡ p[ε] * q, so we get to use * for both the “framing off” and the “tacking on” operations, a substantial simplification. Reed gives a sequent calculus definition in his thesis, but it is given as if it was written in LLF (he has license to do this because LLF is embedded in

HLF!) We can also write the sequent calculus presentation of linear logic directly into hybrid logic.

```
 lol-r : (∀α. hyp(A)@α → conc(C)@(p*α)) →
         conc(A ⊸ B)@p.
 lol-l : (∀β. hyp(B)@β → conc(C)@(p*β)) →
         (conc(A)@q) →
         (hyp(A ⊸ B)@r → conc(C)@(p*q*r)).

 wth-r : conc(A)@p → conc(B)@p →
         conc(A & B)@p.
 wth-l1: (∀α. hyp(A)@α → conc(C)@(p*α)) →
         (hyp(A & B)@q → conc(C)@(p*q)).
 wth-l2: (∀α. hyp(B)@β → conc(C)@(p*α)) →
         (hyp(A & B)@q → conc(C)@(p*q)).

 ten-r : conc(A)@p → conc(B)@q →
         conc(A ⊗ B)@(p*q).
 ten-l : (∀α.∀β. hyp(A)@α → hyp(B)@β → conc(C)@(p*α*β)) →
         (hyp(A ⊗ B)@q → conc(C)@(p*q)).
 or-r1 : conc(A)@p → conc(A ⊕ B)@p.
 or-r2 : conc(B)@p → conc(A ⊕ B)@p.
 or-l  : (∀α. hyp(A)@α → conc(C)@(p*α)) →
         (∀β. hyp(B)@β → conc(C)@(p*β)) →
         (hyp(A ⊕ B)@q → conc(C)@(p*r)).
```

To be explicit about the "framing off" concept, we can rewrite the above rules to use "p*q" when we are talking about "tacking on" the world q to the world p and using "p[q]" when we are talking about framing off the world q from the larger surrounding context p.

```
 lol-r : (∀α. hyp(A)@α → conc(C)@(p*α)) →
         conc(A ⊸ B)@p.
 lol-l : (∀β. hyp(B)@β → conc(C)@(p[β])) →
         (conc(A)@q) →
         (hyp(A ⊸ B)@r → conc(C)@(p[q*r])).

 wth-r : conc(A)@p → conc(B)@p →
         conc(A & B)@p.
 wth-l1: (∀α. hyp(A)@α → conc(C)@(p*α)) →
         (hyp(A & B)@q → conc(C)@(p[q])).
 wth-l2: (∀α. hyp(B)@β → conc(C)@(p[α])) →
         (hyp(A & B)@q → conc(C)@(p[q])).

 ten-r : conc(A)@p → conc(B)@q →
         conc(A ⊗ B)@(p*q).
 ten-l : (∀α.∀β. hyp(A)@α → hyp(B)@β → conc(C)@(p[α*β])) →
         (hyp(A ⊗ B)@q → conc(C)@(p[q])).

 or-r1 : conc(A)@p → conc(A ⊕ B)@p.
 or-r2 : conc(B)@p → conc(A ⊕ B)@p.
 or-l  : (∀α. hyp(A)@α → conc(C)@(p[α])) →
         (∀β. hyp(B)@β → conc(C)@(p[β])) →
         (hyp(A ⊕ B)@q → conc(C)@(p[r])).
```

In this rewriting, every left rule uses the framing-off operation in a fairly uniform manner. There are four rules that still use * in this formulation, corresponding to the two multiplicative connectives ⊸ and ⊗.

 lol-r: used in the premise to point out that we've tacked on a new thing
 lol-l: used in the conclusion to point out that the framed-off part has two different parts
 ten-r: used in the conclusion to point out that we need two distinct parts of the context
 ten-l: used in the premise to point out that the framed-off part is being replaced with two different parts.

2.2 An alternate formulation of linear logic

In this section, we will make an odd proposal for linear logic based on the fact that linear implication ⊸, not conjunction ⊗, is the fundamental way of "tacking things on" to linear logic contexts.

This suggests that, instead of having a symmetric conjunction, we can define worlds like this:

```
 p ::= ε | α::p
 α::β::p ≡ β::α::p
```

Now, instead of worlds being a "tree-structured" syntactic object that is made into a multiset soup by the equalities imposed on it, it is a "list-structured" syntactic object that is made a multiset soup by the equalities imposed on it. The point I wanted to make, though, is that it now exactly captures context formation as suggested both by the inference rule and the encoded inference rule for linear implication:

```
 A,Δ ⊢ B
 ----------
 Δ ⊢ A ⊸ B

 lol-r : (∀α. hyp(A)@α → conc(B)@(α::p)) →
         conc(A ⊸ B)@p.
```

The left rule, on the other hand, uses the implicit "framing off" operation. Remember, this rule has to "frame off" a world p, and α is no longer a world p, though the singleton list α::ε is.

```
 lol-l : (∀β. hyp(B)@β → conc(C)@(p[β::ε])) →
         (conc(A)@q) →
         (hyp(A ⊸ B)@h → conc(C)@(p[h::q])).
```

Since "h" is attached to a hypothesis, it actually belongs to the same syntactic class as α and β, which is why "h::q" makes sense (we used "h" instead of "r" in the previous examples to emphasize this just a bit). So, reading from bottom to top:

- A context "h::q" is framed off
- The head "h," associated with the proposition A ⊸ B, is consumed
- The tail "q" is used to prove the premise A.
- The hole is filled in by the singleton context associated (via the freshly-introduced world variable β) with our newly-derived fact B.

The right rule for tensor is awkwardly asymmetric - we really want to split the context into two parts, and it happens that we can do that by framing off one part of the context. This won't scale to ordered logic, but it still works here so I'll mention it.

```
 ten-r : conc(A)@p[ε] → conc(B)@q →
         conc(A ⊗ B)@p[q].
 ten-l : (∀α. hyp(A)@α → ∀β. hyp(B)@β → conc(C)@p[α::β::ε])
         (hyp(A ⊗ B)@h → conc(C)@p[h::ε]).
```

2.3 Queue logic

The payoff of this unusual formulation of linear logic is that, if we remove all equations on worlds, we end up with the queue logic proposed by Reed [JRQ]. Queue logic as presented by Reed has an ordered context, a linear implication that allows propositions to be tacked on to the right-hand side of the context but only decomposed when on the left-hand side of the context. Our presentation is the "other way around" to match the way a list works - conceptually, we are used to tacking things on to the left-hand side of a list, and the way we use "framing off p[q] naturally forces us to decompose only the hypothesis on the right-hand side of the list.

```
init : hyp(A)@h → conc(A)@(h::ε).

one-r : conc(1)@ε.
one-l : conc(C)@p[ε] →
        (hyp(1)@h → conc(C)@p[h :: ε]).

imp-r : (∀α. hyp(A)@α → conc(B)@(α::p)) →
        conc(A ↣ B)@p.
imp-l : (∀β. hyp(B)@β → conc(C)@(p[β::ε])) →
        (conc(A)@q[ε]) →
        (hyp(A ↣ B)@h → conc(C)@(p[q[h :: ε]])).


and-r : conc(A)@p[ε] → conc(B)@q →
        conc(A • B)@p[q].
and-l : (∀α. hyp(A)@α → ∀β. hyp(B)@β → conc(C)@p[α :: β :: ε]) →
        (hyp(A • B)@h → conc(C)@p[h :: ε]).
```

3 Ordered logic

We can pull the same trick with ordered logic that we did for linear logic, either making an implication-looking thing the primary way of manipulating the context or making a conjunction-looking thing the primary way of manipulating the context. Chris Martens' undergraduate thesis works through much of the process of putting an ordered hybrid framework together in the conjunction-ey style:

```
p ::= α | ε | p · q
ε · a ≡ a ≡ a · ε
a · (b · c) ≡ (a · b) · c

limp-r : (∀α. hyp(A)@α → conc(B)@(α·p)) →
        conc(A ↣ B)@p.
limp-l : (∀β. hyp(B)@β → conc(C)@(p·β·r)) →
         (conc(A)@q) →
         (hyp(A ↣ B)@h → conc(C)@(p·(q·h)·r)).

rimp-r : (∀α. hyp(A)@α → conc(B)@(p·α)) →
        conc(A ↠ B)@p.
rimp-l : (∀β. hyp(B)@β → conc(C)@(p·β·r)) →
         (conc(A)@q) →
         (hyp(A ↠ B)@h → conc(C)@(p·(h·q)·r)).

fuse-r : conc(A)@p → conc(B)@r → conc(A • B)@(p·r).
fuse-l : (∀α. hyp(A)@α → ∀β. hyp(B)@β → conc(C)@(p·(α·β)·r)) →
         (hyp(A • B)@h → conc(C)@(p·h·r)).
```

Here, as in linear logic, we can just as easily give the left rules using the "framed-off" style. This is kind of nice, instead of saying "h exists somewhere in the middle of the context" as "p·h·r" we can write it as "p[h]."

```
limp-r : (∀α. hyp(A)@α → conc(B)@(α·p)) →
        conc(A ↣ B)@p.
limp-l : (∀β. hyp(B)@β → conc(C)@(p[β])) →
         (conc(A)@q) →
         (hyp(A ↣ B)@h → conc(C)@(p[q·h])).
```

```
 rimp-r : (∀α. hyp(A)@α → conc(B)@(p·α)) →
        conc(A ↠ B)@p.
 rimp-l : (∀β. hyp(B)@β → conc(C)@(p[β])) →
          (conc(A)@q) →
          (hyp(A ↠ B)@h → conc(C)@(p[h·q])).

 fuse-r : conc(A)@p → conc(B)@r → conc(A • B)@(p·r).
 fuse-l : (∀α. hyp(A)@α → ∀β. hyp(B)@β → conc(C)@(p[α·β])) →
          (hyp(A • B)@h → conc(C)@(p[h])).
```

As a final point, we can encode the negative fragment of ordered logic in an implication-centric style - there are two "tacking on" operations corresponding to left and right implication.

```
 p ::= ε | α ▹ p | p ◃ α

 ε ◃ α ≡ α ▹ ε
 α ▹ (p ◃ β) ≡ (α ▹ p) ◃ β
```

Let's see how we can re-associate a context from "right" to "left" using these rules:

```
 α ▹ (β ▹ (γ ▹ ε))
   ≡ α ▹ (β ▹ (ε ◃ γ))
   ≡ α ▹ ((β ▹ ε) ◃ γ)
   ≡ α ▹ ((β ▹ ε) ◃ γ)
   ≡ (α ▹ (β ▹ ε)) ◃ γ
   ≡ (α ▹ (ε ◃ β)) ◃ γ
   ≡ ((α ▹ ε) ◃ β) ◃ γ
   ≡ ((ε ◃ α) ◃ β) ◃ γ
```

The biggest issue is that we cannot encode fuse-r in this setting. In ordered logic, p[q] means (roughly) "a string p with a q somewhere in the middle." But fuse-r above needs to be able to express p·q, a string composed of p concatenated with string q. Some different kind of notation (or a different understanding of our current notation) would be needed.

That said, we're often interested in the fragment of a language without conjunction, so it's still worth describing what the rules for the two forms of implication look like.

```
 limp-r : (∀α. hyp(A)@α → conc(B)@(α ▹ p)) →
        conc(A ↣ B)@p.
 limp-l : (∀β. hyp(B)@β → conc(C)@(p[ε ◃ β])) →
          (conc(A)@q) →
          (hyp(A ↣ B)@h → conc(C)@(p[q ◃ h])).

 rimp-r : (∀α. hyp(A)@α → conc(B)@(p ◃ α)) →
        conc(A ↠ B)@p.
 rimp-l : (∀β. hyp(B)@β → conc(C)@(p[β ▹ ε])) →
          (conc(A)@q) →
          (hyp(A ↠ B)@h → conc(C)@(p[h ▹ q])).
```

It's curious that limp-r uses ▹ and limp-l uses ◃, isn't it? It would also be interesting to explore what kind of logic happens when one or both of the equalities described above is removed. Is it even still a sensible logic?

4 Bunched logic

Bunched logic is where we most need our concept of a separate “framing off” operator. Bunched contexts are inherently tree-structured, so we cannot use the conjunctive tacking-on operator to simulate framing off as we could in ordered and linear logic. The worlds for bunched logic are formed from two commutative monoids that have their own units but no distributivity or cancellation properties.

```
p ::= α | εa | p ⊚ q | εm | p ⊛ q |

 εm ⊛ p ≡ p ≡ p ⊛ εm
 p ⊛ (q ⊛ r) ≡ (p ⊛ q) ⊛ r
 p ⊛ q ≡ q ⊛ p

 εa ⊚ p ≡ p ≡ εa ⊚ p
 p ⊚ (q ⊚ r) ≡ (p ⊚ q) ⊚ r
 p ⊚ q ≡ q ⊚ p
```

We can’t directly represent weakening and contraction as equalities, but we can represent them as rules.

```
 weakening   : conc(A)@p[q] → conc(B)@p[q⊚r].
 contraction : conc(A)@p[q⊚q] → conc(A)@p[q].
```

The “context equality” rule:

```
 Δ ≡ Γ  Γ ⊢ A
 ------------- E
   Δ ⊢ A
```

is inherent in the equations above. The rest of the rules follow naturally, using the framing off operation in a now-familiar style.

```
 init        : hyp(A)@h → conc(A)@h

 mwand-r : (∀α. hyp(A)@α → hyp(B)@(p⊛α)) →
           conc(A -* B)@p.
 mwand-l : (∀β. hyp(B)@β → conc(C)@p[β])
           conc(A)@q
           (hyp(A -* B)@h → conc(C)@p[h⊛q]).

 impl-r :  (∀α. hyp(A)@α → hyp(B)@(p⊚α)) →
           conc(A ⇒ B)@p.
 impl-l :  (∀β. hyp(B)@β → conc(C)@p[β])
           conc(A)@q
           (hyp(A ⇒ B)@h → conc(C)@p[h⊚q]).

 mconj-r : conc(A)@p → conc(B)@q → conc(A * B)@(p⊛q)  .
 mconj-l : (∀α. hyp(A)@α → ∀β. hyp(B)@β → conc(C)@p[α⊛β]) →
           (hyp(A * B)@h → conc(C)@p[h]).

 aconj-r : conc(A)@p → conc(B)@q → conc(A ∧ B)@(p⊚q)  .
 aconj-l : ∀α. hyp(A)@α → ∀β. hyp(B)@β → conc(C)@p[α⊚β] →
           (hyp(A ∧ B)@h → conc(C)@p[h]).
```

5 References

[GH] G. G. Huet, “The zipper,” Journal of Functional Programming, vol. 7, no. 05, pp. 549-554, September 1997. [Online]. Available: http://dx.doi.org/10.1017/S0956796897002864

[JRQ] J. Reed, “Queue logic: An undisplayable logic?” April 2009. [Online]. Available: http://www.cs.cmu.edu/~jcreed/papers/queuelogic.pdf

[JR] J. Reed, “A hybrid logical framework,” Ph.D. dissertation, Carnegie Mellon University, July 2009. [Online]. Available: http://reports-archive.adm.cs.cmu.edu/anon/2009/abstracts/09-155.html

[NB] N. Berregeb, A. Bouhoula, and M. Rusinowitch, “Observational proofs with critical contexts,” 1998, pp. 38+. [Online]. Available: http://www.springerlink.com/content/79j071j4hk36w82x

[RP] J. Reed and F. Pfenning, “A constructive approach to the resource semantics of substructural logics.”