Names via Substructural and Dependent Types

Jason Reed

September 20, 2008

Binding and Names

- Various familiar ways of handling variable binding
- HOAS, Nominal Logic, deBruijn indices, etc.
- Nominal logic — easy reasoning about **disequality**, **apartness**: primitive apartness relation $a\#b$
- HOAS does not apparently make this as easy

Example: α-inequality of λ-terms (in Nominal Logic Programming)

[taken from Cheney, Urban '06]

$var : name \to term$

$lam : \langle name \rangle term \to term$

$aneq\ (lam\ \langle x \rangle E)\ (lam\ \langle x \rangle E')\ \text{:--}\ aneq\ E\ E'$

$aneq\ (var\ X)\ (var\ Y)\ \text{:--}\ X \# Y$

$\ldots$

Example: α-inequality of λ-terms (in HOAS)

$var : name \rightarrow term$

$lam : (name \rightarrow term) \rightarrow term$

$aneq\ (lam\ E)\ (lam\ E') \mathrel{:-} \Pi x{:}name.aneq\ (E\ x)\ (E'\ x)$

$aneq\ (var\ X)\ (var\ Y) \mathrel{:-}$?

Problem: last clause (apparently) can't help but match even when X and Y are equal.

Even worse with usual HOAS encoding of terms where variables are not specially distinguished!

Alternate HOAS Encoding

- Actually could tediously keep track of and pass around a list of names discovered so far each time a new name is introduced
- Effectively implement apartness maually by walking through this list
- Not terrifically satisfying

Another Idea

- Use concepts from linear logic, other substructural logics to get **simple encoding of apartness**
  - without introducing it as primitive as in nominal logic
  - without explicit list-passing or -crawling as in HOAS above
- Will need dependent types to interact properly, too
- (Falls naturally out of logical framework **HLF** designed originally for other reasons)
- Will just show the fragment required

Essential Claim

Apartness relation in nominal logic can be nicely encoded by the appropriate combination of **substructural** and **dependent** types.

Plan

- Sketch appropriate **logic** for encoding
- Show how **apartness** is encoded
- Examples of **use** of apartness relation

Foreshadowing

- **Declare** $X\#Y$ as a relation, with kind something like $\mathit{name} \to \mathit{name} \to \mathsf{type}$.
- **Define** $X\#Y$ with one clause something like $\Pi X{:}\mathit{name}.\Pi Y{:}\mathit{name}.X\#Y$.
- But we don't want **any** X and Y in this relation, just **different** ones
- So **consume** each argument linearly to enforce disjointness: think '$\mathit{name} \multimap \mathit{name} \multimap \cdots$'
- Want some kind of **linear Pi**, so we can say something like $\Pi X\hat{:}\mathit{name}.\Pi Y\hat{:}\mathit{name}.X\#Y$.
- **Affineness** will matter, but we can deal with it.

n-ary Linear Logic

- Generalize **linear Pi** (must use argument exactly once) to ***n*-ary Pi** $\Pi x{:}^n A.B$ (must use argument exactly n times)
- The cases $n = 0$ (!) and $n = 1$ will be the important ones for us.

Judgmental Setup

$(x :^{n} A)$ means: x gets used exactly n times

$$\Delta ::= x_1 :^{n_1} A_1, \ldots, x_K :^{n_K} A_K$$

$$\Gamma ::= x_1 : B_1, \ldots, x_K : B_K$$

Typing judgment:

$$\Delta; \Gamma \vdash M : C$$

n-linear dependent function types

$$\frac{\Gamma; \Delta, \boxed{x :^n A} \vdash M : B}{\Gamma; \Delta \vdash \hat{\lambda} x.M : \Pi x{:}^n A.B}$$

$$\frac{\Gamma; \Delta_1 \vdash M : \Pi x{:}^n A.B \qquad \Gamma; \Delta_2 \vdash N : A}{\Gamma; \boxed{\Delta_1 + n \cdot \Delta_2} \vdash M \,\hat{}\, N : [N/x]B}$$

$$(x :^n A) + (x :^m A) = (x :^{n+m} A)$$

$$n \cdot (x :^m A) = (x :^{nm} A)$$

Use of Variables

$$\frac{x : A \in \Gamma}{\Gamma;\ \boxed{0 \cdot \Delta} \vdash x : A} \qquad \frac{}{\Gamma;\ \boxed{(x :^1 A) + 0 \cdot \Delta} \vdash x : A}$$

Ordinary dependent function types

$$\frac{\Gamma, \boxed{x : A}\, ; \Delta \vdash M : B}{\Gamma; \Delta \vdash \lambda x.M : \Pi x{:}A.B}$$

$$\frac{\Gamma; \Delta \vdash M : \Pi x{:}A.B \qquad \Gamma; \boxed{0 \cdot \Delta} \vdash N : A}{\Gamma; \Delta \vdash M\,N : [N/x]B}$$

Abbreviations

Can generalize Linear Logical Framework LLF [Cervesato, Pfenning] if we set

$$A \multimap B \equiv \Pi x{:}^1 A.B$$

And moreover say for convenience

$$A \to B \equiv \Pi x{:}A.B$$

$$A \not\multimap B \equiv \Pi x{:}^0 A.B$$

Digression on Substructural Dependent Types

- Can be hard, so usually we only have $A \multimap B$: consider

  $o : \mathsf{type}$

  $\mathit{fam} : o \multimap \mathsf{type}$

$$\frac{x \mathrel{\hat{:}} o \vdash x : o \qquad y \mathrel{\hat{:}} \mathit{fam} \,\hat{}\, x \multimap o \vdash y : \mathit{fam} \,\hat{}\, x \multimap o}{x \mathrel{\hat{:}} o,\ y \mathrel{\hat{:}} \mathit{fam} \,\hat{}\, x \multimap o \vdash y \,\hat{}\, x : o}$$

  Context splitting strands y away from x!

- Works better with '0-linear' type family:

  $\mathit{fam} : o \not\multimap \mathsf{type}$

$$\frac{x :^0 o,\ y :^1 \mathit{fam} \,\hat{}\, x \multimap o \vdash y : \mathit{fam} \,\hat{}\, x \multimap o \qquad x :^1 o \vdash x : o}{x :^1 o,\ y :^1 \mathit{fam} \,\hat{}\, x \multimap o \vdash y \,\hat{}\, x : o}$$

Well-Formedness of Dependent Types

$$\frac{\Gamma; \Delta, \boxed{x :^0 A} \vdash B : \mathsf{type}}{\Gamma; \Delta \vdash \Pi x{:}^n A.B : \mathsf{type}} \qquad \frac{\Gamma, \boxed{x : A}; \Delta \vdash B : \mathsf{type}}{\Gamma; \Delta \vdash \Pi x{:}A.B : \mathsf{type}}$$

- Argument of a (n-)linear Π is required to "be used **zero** times" in the body of the type.
- Safe generalization of requiring it **not** to occur ($\multimap$)
- Strict generalization because other constants used in B may have types like $C \not\multimap D$, which promise that they use their substructural argument zero times.

Encoding Apartness

$name : \mathsf{type}$.

$\# : name \not\multimap name \not\multimap \mathsf{type}$

$irrefl : \Pi X{:}^1 name.\Pi Y{:}^1 name.\ (X\#Y \ \circ\!- \top\ )$

That's it!

Encoding Apartness

$\mathit{name} : \mathsf{type}\,.$

$\# : \mathit{name} \mathrel{\not\multimap} \mathit{name} \mathrel{\not\multimap} \mathsf{type}$

$\mathit{irrefl} : \Pi X{:}^1\mathit{name}.\Pi Y{:}^1\mathit{name}.\ (X\#Y \ \multimapinv \top\ )$

Note that:

- $X\#Y$ short for $\#\,\hat{}\,X\,\hat{}\,Y$
- $\multimapinv \top$ because other names besides X and Y may be present $\left(\text{The intro rule for } \top \text{ is just } \dfrac{}{\Gamma; \Delta \vdash \langle\rangle : \top}\right)$
- Linear hypotheses of names consumed in **derivation** of apartness and not in **formation** of the apartness relation

Encoding α-inequality

$var : name \multimap\!\!\!\!/\; term$

$lam : (name \multimap\!\!\!\!/\; term) \to term$

$\_ : aneq\ (lam\ E)\ (lam\ E') \circ\!\!- (\Pi x{:}^{1} name.aneq\ (E \mathbin{\hat{}} x)\ (E' \mathbin{\hat{}} x))$

$\_ : aneq\ (var\ X)\ (var\ Y) \circ\!\!- X \# Y$

$\cdots$ (more cases, just as in nominal logic program)

Encoding α-inequality

$var : name \not\multimap term$

$lam : (name \not\multimap term) \rightarrow term$

$\_ : aneq\ (lam\ E)\ (lam\ E') \multimapinv (\Pi x{:}^1 name.aneq\ (E \mathbin{\hat{}} x)\ (E' \mathbin{\hat{}} x))$

$\_ : aneq\ (var\ X)\ (var\ Y) \multimapinv X \# Y$

- Functions over names are 0-linear dependent functions.

Encoding α-inequality

$var : name \not\multimap term$

$lam : (name \not\multimap term) \rightarrow term$

$\_ : aneq\ (lam\ E)\ (lam\ E') \multimapinv (\Pi x{:}^{1} name.aneq\ (E \hat{}\ x)\ (E' \hat{}\ x))$

$\_ : aneq\ (var\ X)\ (var\ Y) \multimapinv X \# Y$

- Functions over names are 0-linear dependent functions.
- Linear functions automatically propagate the set of names.

Encoding α-inequality

$var : name \not\multimap term$

$lam : (name \not\multimap term) \rightarrow term$

$\_ : aneq\ (lam\ E)\ (lam\ E') \multimapinv (\ \Pi x{:}^1 name\ .aneq\ (E \hat{}\ x)\ (E' \hat{}\ x))$

$\_ : aneq\ (var\ X)\ (var\ Y) \multimapinv X\#Y$

- Functions over names are 0-linear dependent functions.
- Linear functions automatically propagate the set of names.
- 1-linear dependent function abstracts over new name.

The Encoding In Action

(abbreviate *name* as n)

$$\dfrac{\dfrac{x_1 :^1 n, x_3 :^1 n \vdash \top \qquad x_2 :^1 n \vdash x_2 : n \qquad x_4 :^1 n \vdash x_4 : n}{x_1 :^1 n, x_2 :^1 n, x_3 :^1 n, x_4 :^1 n \vdash x_4 \# x_2}}{x_1 :^1 n, x_2 :^1 n, x_3 :^1 n, x_4 :^1 n \vdash aneq\ (var\ x_4)\ (var\ x_2)}$$

Recall: $irrefl : \Pi X{:}^1 name.\Pi Y{:}^1 name.\ (X \# Y \multimapinv \top)$

$$\dfrac{\dfrac{x_1 :^1 n, x_3 :^1 n \vdash \top \qquad x_2 :^{X} n \vdash x_2 : n \qquad x_2 :^{X} n \vdash x_2 : n}{x_1 :^1 n, x_2 :^1 n, x_3 :^1 n, x_4 :^1 n \vdash x_2 \# x_2}}{x_1 :^1 n, x_2 :^1 n, x_3 :^1 n, x_4 :^1 n \vdash aneq\ (var\ x_2)\ (var\ x_2)}$$

Problem: no $X \in \mathbb{N}$ s.t. $X + X = 1$

Encoding a Programming Language with Store

$eval : store \to exp \to result \to \mathsf{type}$

$letref : val \to (val \to exp) \to exp$ $\%\ \mathbf{let}\, x = \mathbf{ref}\, v\, \mathbf{in}\, e$

$let! : val \to (val \to exp) \to exp$ $\%\ \mathbf{let}\, x = (!v)\, \mathbf{in}\, e$

$loc : name \not\multimap val$

$\_ : eval\ S\ (letref\ V\ E)\ R \multimapinv \Pi\ell{:}^1 n.\ eval\ ((\ell, V) :: S)\ (E\ (loc \,\hat{}\, \ell))\ R$

$\_ : eval\ S\ (let!\ (loc \,\hat{}\, L)\ E)\ R \multimapinv (lookup\ S \,\hat{}\, L\ V\ \&\ eval\ S\ (E\ V)\ R)$

$lookup : store \to name \not\multimap val \to \mathsf{type}$

$\_ : lookup\ ((N, V) :: S) \,\hat{}\, N\ V \multimapinv \top$

$\_ : lookup\ ((N', \_) :: S) \,\hat{}\, N\ V \multimapinv (N \# N'\ \&\ lookup\ S \,\hat{}\, N\ V)$

Reasoning in a Programming Language with Store

$wfstore : store \to \mathsf{type}$

$notin : name \not\multimap store \to \mathsf{type}$

$\_ : wfstore\ nil \circ\!\!- \top$

$\_ : wfstore\ ((\ N\ , \_) :: S) \circ\!\!- (notin \,\hat{}\ N\ \ S \,\&\, wfstore\ S)$

$\_ : notin \,\hat{}\ N\ \ nil \circ\!\!- \top$

$\_ : notin \,\hat{}\ N\ \ ((\ N'\ , \_) :: S) \circ\!\!- (notin \,\hat{}\ N\ \ S \,\&\ \ N \# N'\ )$

Or: could use substructural features directly, for shorter or more expressive encoding

$wfstore' : store \to \mathsf{type}$

$\_ : wfstore'\ nil \circ\!\!- \top$ (or just $\_ : wfstore'\ nil$)

$\_ :\ \Pi x{:}^{1} name\ .(wfstore'\ S \multimap wfstore'\ ((\ x\ , \_) :: S))$

Related Work

- n-ary use functions [Wright, Momigliano]
- 0-ary use ("irrelevant") functions [Pfenning, Ley-Wild]
- RLF [Ishtiaq, Pym]
- HLF
  - Designed for statement of metatheorems for Linear LF.
  - Does n-linear Πs above, and more (e.g. some of BI)
  - Prototype implementation

Conclusion

- Substructural dependent types can imitate nominal logic programming techniques
- Practical?
- In what ways does it do even better?

Thanks