A Nominal Logical Framework

Logic and Semantics Club
January 20, 2006

James Cheney

University of Edinburgh

Introduction

- A *logical framework* is a formal system for defining (and reasoning about) other formal systems.
- Typically, employs a *dependent type theory* to represent syntax and judgments of a logic or language
- Idea goes back to at least de Bruijn's AUTOMATH project
- (Edinburgh) LF, Calculus of Constructions, and variants have proven very expressive and powerful.
- Fertile area for interesting type theories (Linear LF, Ordered LF, Concurrent LF; Inductive Constructions)

Do we really need another LF?

- Existing LF family very expressive, but:
  1. Names are "second-class": difficult to encode judgments based on *name-inequality*
  2. Inductive (meta-)reasoning apparently must be performed externally; co-induction, logical relations not well understood
  3. Encoding generativity (e.g. allocation semantics) requires resorting to counters/linearity
- Eventual goal: handle all of these concerns; today focus on (1) and sketch (2), (3)

Another motivation

- *Nominal logic*: a recently proposed technique for "near first-order" reasoning about names & binding; basis for αProlog
- Related theories: Miller & Tiu's $FO\lambda^{\nabla}$, higher-order patterns, Pfenning, Pientka and Nanevski's *contextual modal type theory*
- Goal: Identify a common *core type theory* unifying the basic ideas underlying these systems
- thus providing a *constructive foundation* for nominal techniques
- and hopefully getting certain type theorists off my case...

Syntax

- Extends LF

$$
\begin{array}{rcl}
K & ::= & \mathsf{type} \mid \Pi x{:}A.K \mid \mathsf{name} \\
A, B & ::= & a \mid A\ M \mid \Pi x{:}A.B \mid \text{И}\mathfrak{a}{:}A.B \\
M, N & ::= & c \mid x[\rho] \mid \lambda x{:}A.M \mid M\ N \mid \mathfrak{a} \mid \langle \mathfrak{a} \rangle M \mid M \mathbin{@} \mathfrak{a} \\
\rho & ::= & \mathsf{id} \mid \rho, \mathfrak{a}/\mathfrak{b}
\end{array}
$$

- Note: *Names* $\mathfrak{a}, \mathfrak{b}, \ldots$ are a new syntactic class similar to (but disjoint from) variables
- ρ: *explicit renamings*
- $\text{И}\mathfrak{a}{:}A.B$, $\langle \mathfrak{a} : A \rangle M$, subject to α-equivalence
- α-equivalence & capture-avoiding substitution/renaming operations defined in traditional way for LF

Contexts

- Contexts: as LF, but with extra *name contexts* σ.
- Also, types in Γ are guarded by name-context.

$$\begin{array}{lcl} \Sigma & ::= & \cdot \mid c : A \mid a : K \\ \Gamma & ::= & \cdot \mid x : A[\sigma] \\ \sigma & ::= & \cdot \mid \mathfrak{a} : A \end{array}$$

- Well-formedness for contexts: as LF, but types in σ must be of kind name (and cannot depend on names).

$$\frac{\Gamma \vdash \sigma \ \mathsf{nctx} \quad \Gamma \vdash \cdot \triangleright A : \mathsf{name}}{\Gamma \vdash \sigma, \mathfrak{a} : A \ \mathsf{nctx}}$$

Judgments

- Judgments: as LF, but with extra name-context.

$\vdash \Sigma$ sig	Signature formation
$\vdash \Gamma$ ctx	Context formation
$\Gamma \vdash \sigma$ nctx	Name context formation
$\Gamma \vdash \sigma \triangleright K :$ kind	Kind formation
$\Gamma \vdash \sigma \triangleright A : K$	Type family formation
$\Gamma \vdash \sigma \triangleright M : A$	Object formation
$\Gamma \vdash \sigma \triangleright \rho : \sigma'$	Renaming formation
$\Gamma \vdash \sigma \triangleright K = K' :$ kind	Kind equality
$\Gamma \vdash \sigma \triangleright A = B : K$	Type family equality
$\Gamma \vdash \sigma \triangleright M = N : A$	Object equality
$\Gamma \vdash \sigma \triangleright \rho = \rho' : \sigma'$	Renaming equality

Typing: highlights

- The rules for variables and renamings:

$$\frac{\Gamma \vdash \sigma' \triangleright \rho : \sigma}{\Gamma, x{:}A[\sigma] \vdash \sigma' \triangleright x[\rho] : A[\rho]}$$

- Variables have to be instantiated with appropriate local names.

$$\frac{}{\Gamma \vdash \sigma \triangleright \mathsf{id} : \cdot}$$

$$\frac{\Gamma \vdash \sigma \triangleright \rho : \sigma'}{\Gamma \vdash \sigma, \mathfrak{b} : A \triangleright \rho, \mathfrak{a}/\mathfrak{b} : \sigma', \mathfrak{a} : A}$$

- Renamings must be one-to-one.

Typing: highlights

- The rules for names, abstractions, and concretions:

$$\frac{}{\Gamma \vdash \sigma, \mathfrak{a}{:}A \triangleright \mathfrak{a} : A}$$

$$\frac{\Gamma \vdash \sigma, \mathfrak{a}{:}A \triangleright M : B}{\Gamma \vdash \sigma \triangleright \langle \mathfrak{a}{:}A \rangle M : \text{И}\mathfrak{a}{:}A.B}$$

$$\frac{\Gamma \vdash \sigma \triangleright M : \text{И}\mathfrak{a}{:}A.B}{\Gamma \vdash \sigma, \mathfrak{a}{:}A \triangleright M @ \mathfrak{a} : B}$$

- Fresh name $\mathfrak{a}$ added to the context in $\langle \mathfrak{a} : A \rangle M$ rule
- Names *removed from* the context in $M @ \mathfrak{a}$ rule.
- Therefore, $\langle \mathfrak{a}{:}A \rangle \lambda x{:}\text{И}\mathfrak{a}{:}A.\text{И}\mathfrak{b}{:}A.x @ \mathfrak{a} @ \mathfrak{a}$ is not typable.
- However, $\lambda x{:}A \to A \to B.\langle \mathfrak{a}{:}A \rangle x\ \mathfrak{a}\ \mathfrak{a}$ is OK.

Equality: highlights

- β-reduction and η-expansion (extensionality) for Π-types/λ-terms as in LF
- For И-types, we have

$$\begin{array}{llcll} (\beta) & (\langle \mathfrak{a}{:}A\rangle M)@\mathfrak{b} & = & M[\mathfrak{b}/\mathfrak{a}] & (\mathfrak{b} \notin FV(M)) \\ (\eta) & \langle \mathfrak{a}{:}A\rangle (M\,@\,\mathfrak{a}) & = & M : И\mathfrak{a}{:}A.B & (\mathfrak{a} \notin FV(M)) \end{array}$$

Standard, but note that β-rule can never identify two different names.

Local reductions/expansions

- The following *local* soundness and completeness properties for И are important checks that the system is sensible:

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma \vdash \sigma, \mathfrak{a}{:}A \triangleright M : B\end{array}}{\Gamma \vdash \sigma \triangleright (\langle \mathfrak{a}{:}A \rangle M) : \text{И}\mathfrak{a}{:}A.B}}{\Gamma \vdash \sigma, \mathfrak{b}{:}A \triangleright (\langle \mathfrak{a}{:}A \rangle M) \mathbin{@} \mathfrak{b} : B[\mathfrak{b}/\mathfrak{a}]}$$

$$\Downarrow_\beta$$

$$\begin{array}{c}\mathcal{D}[\mathfrak{b}/\mathfrak{a}]\\ \Gamma \vdash \sigma, \mathfrak{b}{:}A \triangleright M[\mathfrak{b}/\mathfrak{a}] : B[\mathfrak{b}/\mathfrak{a}]\end{array}$$

Local reductions/expansions

- The following *local* soundness and completeness properties for $\reflectbox{N}$ are important checks that the system is sensible:

$$\begin{array}{c} \mathcal{D} \\ \Gamma \vdash \sigma \triangleright M : \reflectbox{N}\mathfrak{a}{:}A.B \end{array}$$

$$\Downarrow_\eta$$

$$\dfrac{\dfrac{\begin{array}{c} \mathcal{D} \\ \Gamma \vdash \sigma \triangleright M : \reflectbox{N}\mathfrak{a}{:}A.B \end{array}}{\Gamma \vdash \sigma, \mathfrak{a}{:}A \triangleright M \mathbin{@} \mathfrak{a} : B}}{\Gamma \vdash \sigma \triangleright \langle \mathfrak{a}{:}A \rangle M \mathbin{@} \mathfrak{a} : \reflectbox{N}\mathfrak{a}{:}A.B}$$

Simple example

- As an example, consider lambda term typing encoded in NLF:

$$
\begin{array}{lcl}
wf & : & ctx \rightarrow exp \rightarrow ty \rightarrow type. \\
wf\_var & : & wf\ G\ (var\ V)\ T \leftarrow lookup\ G\ V\ T. \\
wf\_app & : & wf\ G\ (app\ E_1\ E_2)\ U \\
 & \leftarrow & wf\ G\ E_1\ (arr\ T\ U) \leftarrow wf\ G\ E_2\ T. \\
wf\_lam & : & wf\ G\ (lam\ M)\ (arr\ T\ U) \\
 & \leftarrow & \text{И}\mathfrak{a}.wf\ [G,(\mathfrak{a},T)]\ (M\ @\ \mathfrak{a})\ U.
\end{array}
$$

- Contexts are just lists.
- Note that we do not use implication for local hypotheses (and it would be incorrect to do so).

Encoding hypotheses

- A distinctive feature of LF is the higher-order encoding of hypothetical judgments:

$$\begin{array}{rcl} wf\_lam & : & wf\ (lam\ M)\ (arr\ T\ U) \\ & \leftarrow & \Pi x.(wf\ x\ T \to wf\ (M\ x)\ U). \end{array}$$

- This is nifty because (intuitionistic) object language contexts “disappear” into LF’s (intuitionistic) context.
- This *does not work* using И in nominal logic or NLF!

$$\begin{array}{rcl} wf\_lam & : & wf\ (lam\ M)\ (arr\ T\ U) \\ & \leftarrow & \text{И}\mathfrak{a}.(wf\ (var\ \mathfrak{a})\ T \to wf\ (M\ @\ \mathfrak{a})\ U). \end{array}$$

- Why?

Encoding hypotheses

- Why does this not work?
- Local hypotheses are "lifted" out of their name context.
- So the following is derivable:

$$\dfrac{\dfrac{\dfrac{x : wf\ (var\ \mathfrak{a})\ t[\mathfrak{a},\mathfrak{b}] \vdash \mathfrak{a},\mathfrak{b} \triangleright x[\mathfrak{b}/\mathfrak{a},\mathfrak{a}/\mathfrak{b}] : wf\ (var\ \mathfrak{b})\ t}{\vdash \mathfrak{a},\mathfrak{b} \triangleright \lambda x.x[\mathfrak{b}/\mathfrak{a},\mathfrak{a}/\mathfrak{b}] : \Pi x{:}wf\ (var\ \mathfrak{a})\ t.wf\ (var\ \mathfrak{b})\ t}}{\cdot \vdash \mathfrak{a} \triangleright \langle\mathfrak{b}\rangle\lambda x.x[\mathfrak{b}/\mathfrak{a},\mathfrak{a}/\mathfrak{b}] : \text{И}\mathfrak{b}.\Pi x{:}wf\ (var\ \mathfrak{a})\ t.wf\ (var\ \mathfrak{b})\ t}}{\cdot \vdash \cdot \triangleright \langle\mathfrak{a}\rangle\langle\mathfrak{b}\rangle\lambda x.x[\mathfrak{b}/\mathfrak{a},\mathfrak{a}/\mathfrak{b}] : \text{И}\mathfrak{a}.\text{И}\mathfrak{b}.\Pi x{:}wf\ (var\ \mathfrak{a})\ t.wf\ (var\ \mathfrak{b})}$$

- Bad!
- This problem is similar to the kind caused by *equivariance* in nominal logic.

Another example: Closure conversion

- Important FP compilation phase
- Idea: Make all functions *closed*
- Translate functions to (closed function, environment) pair
- Environment shape depends on context:

$$\Gamma = x_n, \ldots, x_1 \mapsto env = \langle v_n, \langle v_{n-1}, \ldots, v_1 \rangle \cdots \rangle$$

- Need ability to test equality and inequality of names.

Closure conversion, informally

- A typical "paper" presentation

$$
\begin{array}{rcl}
C[\![\Gamma, x \vdash x]\!]e & = & \pi_1(e) \\
C[\![\Gamma, x \vdash y]\!]e & = & C[\![\Gamma \vdash y]\!]\pi_2(e) \quad (x \neq y) \\
C[\![\Gamma \vdash e_1\ e_2]\!]e & = & \mathbf{let}\ c = C[\![\Gamma \vdash e_1]\!] \\
 & & \mathbf{in}\ (\pi_1(c))\ (C[\![\Gamma \vdash e_2]\!]e, \pi_2(e)) \\
C[\![\Gamma \vdash \lambda x.e_0]\!]e & = & \langle \lambda y.C[\![\Gamma, x \vdash e_0]\!]y, e\rangle \quad (y \notin FV(\Gamma, x, e, e_0))
\end{array}
$$

- Inequality side conditions: non-obvious how to encode in LF.

Closure conversion in NLF

- This is no problem in NLF.
- Use И-quantifier; # defined in terms of И.

$$\begin{array}{lcl}
cconv & : & list\ id \rightarrow exp \rightarrow exp \rightarrow exp \rightarrow type. \\
cconv_var2 & : & cconv\ [G, X]\ (var\ Y)\ Env\ E \\
 & \leftarrow & X \# Y \\
 & \leftarrow & cconv\ G\ (var\ Y)\ (pi2\ Env)\ E. \\
 & & \\
cconv_lam & : & cconv\ G\ (lam\ F_1)\ Env \\
 & & (pair\ (lam\ F_2)\ Env) \\
 & \leftarrow & \text{И}\mathfrak{x}.\text{И}\mathfrak{y}.cconv\ [G, \mathfrak{x}]\ (F_1 @ \mathfrak{x})\ (var\ \mathfrak{y})\ (F_2 @ \mathfrak{y}).
\end{array}$$

Closure conversion in LF

- This is the best I can do (there may be a better way...)
- Idea: Maintain a list L of “bound” variables; ensure that hypotheses $neq\ X\ Y$ are derivable whenever X, Y are distinct elements of list.

$$
\begin{array}{lcl}
cconv & : & list\ id \rightarrow list\ id \rightarrow tm \rightarrow tm \rightarrow tm \rightarrow type. \\
\\
cconv\_var2 & : & cconv\ L\ [G, X]\ (var\ Y)\ Env\ E \\
 & \leftarrow & neq\ X\ Y \\
 & \leftarrow & cconv\ L\ G\ (var\ Y)\ (pi2\ Env)\ E.
\end{array}
$$

Closure conversion in LF

- Tricky part: Use "distinctness" predicate to encode $2|L|$ inequalities compactly.

$$
\begin{array}{lcl}
distinct & : & id \longrightarrow list\ id \longrightarrow \mathsf{type}. \\
neq\_1 & : & neq\ X\ Y \leftarrow distinct\ X\ L \leftarrow member\ Y\ L \\
neq\_2 & : & neq\ X\ Y \leftarrow distinct\ Y\ L \leftarrow member\ X\ L \\
cconv\_lam & : & cconv\ L\ G\ (lam\ F_1)\ Env\ (pair\ (lam\ F_2)\ Env) \\
 & \leftarrow & \Pi x.\Pi y.distinct\ x\ L \longrightarrow \\
 & & \quad cconv\ [L,x]\ [G,x]\ (F_1\ x)\ (var\ y)\ (F_2\ y)
\end{array}
$$

Formal properties

- Weakening, substitution: standard
- Injective Renaming (but not general renaming) of name-contexts:

  Lemma 1. *If* $\Gamma \vdash \sigma \triangleright \rho : \sigma'$ *and* $\Gamma \vdash \sigma \triangleright M : A$ *then* $\Gamma \vdash \sigma' \triangleright M[\rho] : A[\rho]$.
- Subject Reduction
- Church-Rosser, Strong Normalization: standard, reduction to LF
- Decidability of typechecking: Extensions of proofs by [Goguen 05] or [Harper and Pfenning 05]
- Warning: Currently revising system, still need to check details.

Extensions: induction/coinduction

- Judgments can be defined without recourse to recursion through negative type occurrences
- Hence, supporting "internal" induction/coinduction using standard type theoretic methods ought to be straightforward.

$$\frac{\Gamma \vdash \sigma \triangleright M : A[\mu X.A[X]]}{\Gamma \vdash \sigma \triangleright \mathsf{fold}\ M : \mu X.A[X]} \qquad \frac{\Gamma \vdash \sigma \triangleright M : \mu X.A[X]}{\Gamma \vdash \sigma \triangleright \mathsf{unfold}\ M : A[\mu X.A[X]]}$$

- However, this departs significantly from the "traditional" LF methodology...
- [Momigliano and Tiu 2003] approach can probably be used

Extensions: generativity

- The ability to reason about “fresh for world” name generation is a key aspect of nominal logic.
- Core NLF doesn’t support it: if $\mathsf{a} \notin FN(B)$, then $NL \vdash \reflectbox{N}\mathsf{a}.B \supset B$ but $NLF \not\vdash M : \reflectbox{N}\mathsf{a}.B \to B$.
- Why? The following derivation attempt is stuck:

$$\frac{x{:}\reflectbox{N}\mathsf{a}.B \vdash \cdot \triangleright ?? : B}{\vdash \cdot \triangleright \lambda x{:}(\reflectbox{N}\mathsf{a}{:}A).?? : \reflectbox{N}\mathsf{a}{:}A.B \to B}$$

- A possible solution: Add a “fresh name choice” proof term $\nu\mathsf{a}{:}A.M$

$$\frac{\Gamma \vdash \sigma, \mathsf{a} : A \triangleright M : B}{\Gamma \vdash \sigma \triangleright \nu\mathsf{a}{:}A.M : B}$$

- Equational theory, formal properties seem challenging

What if we vary the allowed renamings?

- Existing name-contexts: allow *weakening*, *exchange*, *injective renaming* but not *contraction* or general substitution
- If we *limit* renamings so that reordering and weakening are forbidden, we get something like core $FO\lambda^{\nabla}$.
- If we *relax* renamings so that contraction/arbitrary substitutions are allowed, then we get something like Binding Algebras.
- Both variations might be interesting!

Semantics

- From a type/proof-theoretic point of view, proving that proofs have normal forms and typechecking is decidable is generally enough.
- From this follows consistency, adequacy, other key properties.
- But it would be nice to work out the semantics of NLF, relate to semantics of NL, $FO\lambda^{\nabla}$, etc.
- Probably easier for the λ-free fragment...

NLF in context

- Nominal techniques rely on bijective renamings; NLF's injective renamings can always be extended to bijective ones (and seem to be more compatible with type theoretic setting).
- Rules for И type are *self-dual*, as in NL
- They also correspond to natural deduction forms of the ∇-quantifier rules from $FO\lambda^{\nabla}$:

$$\frac{\Sigma : \Gamma, (\sigma, x) \triangleright A \Rightarrow C}{\Sigma : \Gamma, \sigma \triangleright \nabla x.A \Rightarrow C} \quad \frac{\Sigma : \Gamma \Rightarrow (\sigma, x) \triangleright A}{\Sigma : \Gamma \Rightarrow \sigma \triangleright \nabla x.A}$$

- Well-formedness restrictions on concretion terms correspond to *higher-order pattern restriction* familiar in higher-order unification

Future work

- Integrating full induction/coinduction (following Momigliano and Tiu?)
- Generativity?
- Implementation/translation to basic LF?
- Semantics?!

Conclusions

- Higher-order techniques underlying traditional LF powerful, but have some limitations.
- Using ideas drawn from a number of sources, we've seen how "first-class" name-inequality can be supported in a *nominal extension* of LF
- Next steps: induction, generativity
- Still lots to do.