

Proof Systems for Reasoning about Generic Judgments

Alwen Tiu

Australian National University and NICTA

Mathematical Theories of Abstraction, Substitution and Naming in
Computer Science
Edinburgh, May 27, 2007

Motivations

- Designing a framework for reasoning about higher-order abstract syntax.
- Challenges: inductive reasoning over α -equivalent classes of judgments, while retaining the benefits of higher-order abstract syntax specification.
- Relating some aspects of nominal techniques (i.e., equivariant reasoning) with a variant of HOAS-style approach to reasoning about binders.

Two-level approach to HOAS specifications

- We follow the two-level approach of McDowell and Miller: specify a computation system in an *object logic*, which is embedded in a *meta-logic*.
- The object logic is typically based on a version of first-order intuitionistic logic (e.g., the hereditary Harrop fragment). The meta-logic includes an induction principle.
- Reasoning about the computation system encoded in the object logic reduces to reasoning about the structure of proofs of the object logic.

Reasoning about binders

- Consider the following formula in an object logic HH

$$\hat{\forall}x.(p x r \Rightarrow \hat{\forall}y.(p y s \Rightarrow p x t))$$

- A meta-level observation: if the formula is provable then r and t must be syntactically equal.
- Encoding in $FO\lambda^{\Delta\nabla}$ (a subset of LINC):

$$\begin{aligned} L \vdash_{HH} P &\triangleq P \in L \\ L \vdash_{HH} P \Rightarrow Q &\triangleq (P :: L) \vdash_{HH} Q \\ L \vdash_{HH} \hat{\forall}x.P x &\triangleq \nabla x.L \vdash_{HH} P x \end{aligned}$$

- The following is provable in $FO\lambda^{\Delta\nabla}$:

$$\forall r \forall s \forall t. \emptyset \vdash_{HH} \hat{\forall}x.(p x r \Rightarrow \hat{\forall}y.(p y s \Rightarrow p x t)) \supset r = t$$

∇ and induction

- LINC allows case analyses on object logic proofs with binders, but some inductive properties are not provable, notably:

$$\forall L[(\nabla x.Lx \vdash_{HH} Px) \supset (\forall x.Lx \vdash_{HH} Px)]$$

- The inductive proof of this statement requires an induction hypothesis that hold for all possible variable context, i.e., hypothesis of the kind

$$\text{“for all” } \vec{x}, \quad \nabla \vec{x}(\nabla y.F \vec{x} y \supset \forall y.F \vec{x} y)$$

which requires some form of quantification over variable contexts.

Possible solutions

- Allow context quantification: possible, but too complicated.
- Strengthen the variable context: allow propositions to hold in arbitrary extensions of variable contexts. In effect, admit this axiom

$$B \supset \nabla x.B, \quad x \text{ not free in } B$$

Relation with Nominal Logic

∇ and the \mathcal{N} -quantifier of Nominal Logic share some similarity, in particular, they are both self-dual and distribute over all propositional connectives. However, the following are not theorems in LINC

$$B \equiv \nabla x.B, \quad \nabla x \nabla y.B \supset \nabla y \nabla x.B$$

$$\forall x.B \supset \nabla x.B \quad \nabla x.B \supset \exists x.B$$

but they are theorems of nominal logic, if ∇ is replaced with \mathcal{N} .

Extensions of LINC

- We consider extensions of LINC with the following axioms:

$$(A1) \quad B \supset \nabla x.B, \quad x \text{ not free in } B$$

$$(A2) \quad \nabla x.B \supset B, \quad x \text{ not free in } B$$

$$(A3) \quad \nabla x \nabla y.B \supset \nabla y \nabla x.B$$

- We examine two extensions: LINC + $\{A1, A3\}$ (call this LG_1) and LINC + $\{A1, A2, A3\}$ (call this LG_2).
- Cut-elimination for the other combinations of $(A1) - (A3)$ do not seem obvious.

Some observations about extensions of LINC

- The distinction between LG_1 and LG_2 is observable in intuitionistic logic, but not in classical logic. Axioms (A1) and (A2) are classically equivalent, since ∇ is self-dual.
- Axiom (A2) forces every type τ in ∇_τ to contain at least countably infinite number of elements:

$$\exists_\tau x_1 \cdots \exists_\tau x_n. x_1 \neq x_2 \neq \cdots \neq x_n$$

is provable for arbitrary n .

- By not accepting (A2), one can allow ∇ -quantification on finite types. This might be useful in encodings of object logic quantifiers over finite types.

Absorbing axioms into rules

$$(A1) : \frac{\Sigma; \bar{x}z\bar{y} \triangleright B, \Gamma \vdash C}{\Sigma; \bar{x}\bar{y} \triangleright B, \Gamma \vdash C} \text{ssl} \qquad \frac{\Sigma; \Gamma \vdash \bar{x}\bar{y} \triangleright B}{\Sigma; \Gamma \vdash \bar{x}z\bar{y} \triangleright B} \text{swr}$$
$$(A2) : \frac{\Sigma; \bar{x}z\bar{y} \triangleright B, \Gamma \vdash C}{\Sigma; \bar{x}z\bar{y} \triangleright B, \Gamma \vdash C} \text{swl} \qquad \frac{\Sigma; \Gamma \vdash \bar{x}z\bar{y} \triangleright B}{\Sigma; \Gamma \vdash \bar{x}\bar{y} \triangleright B} \text{ssr}$$
$$(A3) : \frac{\Sigma; \bar{x}v\bar{u}\bar{y} \triangleright B, \Gamma \vdash C}{\Sigma; \bar{x}u\bar{v}\bar{y} \triangleright B, \Gamma \vdash C} \text{sel} \qquad \frac{\Sigma; \Gamma \vdash \bar{x}v\bar{u}\bar{y} \triangleright B}{\Sigma; \Gamma \vdash \bar{x}u\bar{v}\bar{y} \triangleright B} \text{ser}$$

Rules for quantifiers

Same as in LINC:

$$\frac{\Sigma, \sigma \vdash t : \gamma \quad \Sigma; \sigma \triangleright B[t/x], \Gamma \vdash \mathcal{C}}{\Sigma; \sigma \triangleright \forall_{\gamma} x. B, \Gamma \vdash \mathcal{C}} \forall \mathcal{L} \qquad \frac{\Sigma, h; \Gamma \vdash \sigma \triangleright B[(h \sigma)/x]}{\Sigma; \Gamma \vdash \sigma \triangleright \forall x. B} \forall \mathcal{R}$$

Use *raising* to encode dependency between eigenvariables and local contexts.

$$\frac{\Sigma; (\sigma, y) \triangleright B[y/x], \Gamma \vdash \mathcal{C}}{\Sigma; \sigma \triangleright \nabla x B, \Gamma \vdash \mathcal{C}} \nabla \mathcal{L} \qquad \frac{\Sigma; \Gamma \vdash (\sigma, y) \triangleright B[y/x]}{\Sigma; \Gamma \vdash \sigma \triangleright \nabla x B} \nabla \mathcal{R}$$

Cut elimination for LG_1 and LG_2

Cut elimination holds for both LG_1 and LG_2 . Central to the cut-elimination proof is the *context weakening* lemma: if

$$\Sigma; \sigma_1 \triangleright B_1, \dots, \sigma_n \triangleright B_n \vdash \sigma \triangleright C$$

is provable then

$$\Sigma; x\sigma_1 \triangleright B_1, \dots, x\sigma_n \triangleright B_n \vdash x\sigma \triangleright C$$

where x is a fresh variable, is provable with smaller or equal proof length, but *without the context weakening rules* (i.e., *swl* or *swr*).

An alternative formulation of LG_2

- Since weakening, strengthening and exchange rules are applicable to local signatures, we can prove in LG_2

$$\sigma \triangleright B \vdash \sigma' \triangleright B'$$

where B is obtained from B' by an injective renaming of the variables in σ .

- It turns out that we can simplify the sequent system of LG_2 if we remove the local signatures and adopt the *equivariant principles*.

Equivariant predicates

- Assume an infinite set of special *name constants*. Name constants can be of arbitrary types, depending on applications. Call these types *nominal types*.
- Propositions are considered equivalent under permutations of name constants.

$$\frac{\pi.B = \pi'.B'}{B \vdash B'} \text{ id}$$

for some type-preserving permutations π and π' .

- Example: instead of proving

$$abc \triangleright P a \multimap de \triangleright P e$$

by a series of strengthening of local signatures and renaming, we can simply regard a, b, c, d, e as name constants and conclude

$$\frac{(a e).P a = (a e).P e}{P a \multimap P e}$$

Permutations and Support

- The *support* of a term is the set of name constants appearing in it.
- Application of permutation on arbitrary terms is defined as

$$\begin{aligned}\pi.a &= \pi(a), \quad a \text{ is a name constant} \\ \pi.c &= c, \quad c \text{ is an ordinary constant.} \\ \pi.x &= x \quad \pi.(M N) = (\pi.M) (\pi.N) \quad \pi.(\lambda x.M) = \lambda x.(\pi.M)\end{aligned}$$

- Similar to Nominal Logic, except that: variables have empty supports, substitutions cannot mention name constants.
- Freshness constraints and permutations (or swapping) are not part of the syntax of the logic.

Some rules of LG_2 (with implicit local signatures)

- The propositional rules are standard rules of intuitionistic logic.
- Quantifiers:

$$\frac{\Sigma, \mathcal{K}, \mathcal{C}_{\mathcal{N}} \vdash t : \tau \quad \Sigma; \Gamma, B[t/x] \vdash C}{\Sigma; \Gamma, \forall_{\tau} x. B \vdash C} \forall \mathcal{L} \qquad \frac{\Sigma, h; \Gamma \vdash B[h\vec{c}/x]}{\Sigma; \Gamma \vdash \forall x. B} \forall \mathcal{R}$$

where $h \notin \Sigma$ and $\text{supp}(B) = \{\vec{c}\}$. Notice the use of *raising* to encode dependency on nominal constants (since substitutions cannot mention nominal constants).

$$\frac{\Sigma; \Gamma, B[a/x] \vdash C}{\Sigma; \Gamma, \nabla x. B \vdash C} \nabla \mathcal{L} \qquad \frac{\Sigma; \Gamma \vdash B[a/x]}{\Sigma; \Gamma \vdash \nabla x. B} \nabla \mathcal{R}$$

where $a \notin \text{supp}(B)$.

Adding fixed points and equality

Associate an atomic formula with a defining formula:

$$p\vec{x} \triangleq B\vec{x}.$$

Introduction rules for defined predicates:

$$\frac{\Sigma; \Gamma, B[\vec{t}/\vec{x}] \vdash C}{\Sigma; \Gamma, p\vec{t} \vdash C} \text{def}\mathcal{L}, p\vec{x} \triangleq B \qquad \frac{\Sigma; \Gamma \vdash B[\vec{t}/\vec{x}]}{\Sigma; \Gamma \vdash p\vec{t}} \text{def}\mathcal{R}, p\vec{x} \triangleq B$$

Equality rules:

$$\frac{\{\Sigma\theta; \Gamma\theta \vdash C\theta \mid (\lambda\vec{c}.t)\theta =_{\beta\eta} (\lambda\vec{c}.s)\theta\}}{\Sigma; \Gamma, s = t \vdash C} \text{eq}\mathcal{L} \qquad \frac{}{\Sigma; \Gamma \vdash t = t} \text{eq}\mathcal{R}$$

where $\text{supp}(s = t) = \{\vec{c}\}$

Adding natural number induction

We use similar rules as the ones in $FO\lambda^{\Delta\mathbb{N}}$ [McDowell & Miller]

$$\frac{\vdash Dz \quad j; Dj \vdash D(sj) \quad \Sigma; \Gamma, DI \vdash C}{\Sigma; \Gamma, nat\ I \vdash C} \text{ nat}\mathcal{L}$$

$$\frac{}{\Sigma; \Gamma \vdash nat\ z} \text{ nat}\mathcal{R} \quad \frac{\Sigma; \Gamma \vdash nat\ I}{\Sigma; \Gamma \vdash nat\ (s\ I)} \text{ nat}\mathcal{R}$$

Encoding an object logic

- Consider the hereditary harrop fragment of intuitionistic logic, given by

$$\begin{aligned} D & ::= A \mid G \Rightarrow A \mid \bigwedge_{\tau} x.D \\ G & ::= A \mid tt \mid G \& G \mid A \Rightarrow G \mid \bigwedge_{\iota} x.G \end{aligned}$$

- The object logic formulas are encoded using the (ordinary) constants:

$$\begin{aligned} \langle \rangle & : atm \rightarrow prp & tt & : prp \\ \& & : prp \rightarrow prp \rightarrow prp & \Rightarrow : atm \rightarrow prp \rightarrow prp \\ \bigwedge_{\iota} & : (\tau \rightarrow prp) \rightarrow prp & \bigvee_{\iota} & : (\tau \rightarrow prp) \rightarrow prp \end{aligned}$$

- The domain of object logic quantifier is a nominal type.

A definition of object logic

$seq_I L tt$	\triangleq	$\top.$
$seq_I L \langle A \rangle$	\triangleq	$elem A L.$
$seq_{(sI)} L (A \& B)$	\triangleq	$seq_I L A \wedge seq_I L B.$
$seq_{(sI)} L (A \Rightarrow B)$	\triangleq	$seq_I (A :: L) B.$
$seq_{(sI)} L (\bigwedge x. Gx)$	\triangleq	$\forall x. seq_I L Gx.$
$seq_{(sI)} L \langle A \rangle$	\triangleq	$\exists B. prog A B \wedge seq_I L B.$

Example: behaviors of the object logic eigenvariables

In the object logic, $p X \Rightarrow \bigwedge y. p y$ is not provable.

$$\frac{\frac{\frac{X, l_2; seq_{l_2} [p X] \langle p a \rangle \vdash \perp}{X, l_2; \nabla y. seq_{l_2} [p X] \langle p y \rangle \vdash \perp} \nabla \mathcal{L}}{X, l_1; seq_{l_1} [p X] (\bigwedge y. \langle p y \rangle) \vdash \perp} def\mathcal{L}}{X, l; seq_l nil (p X \Rightarrow \bigwedge y. \langle p y \rangle) \vdash \perp} def\mathcal{L}}{\vdash \forall X \forall l. (seq_l nil (p X \Rightarrow \bigwedge y. \langle p y \rangle) \supset \perp)} \forall \mathcal{R}; \supset \mathcal{R}$$

The leaf sequent is provable since unification fails on $\lambda x. X = \lambda a. a$.

Properties of the object logic

Theorem

The following formulas are provable in LG_2 with the definition of the object logic HH :

① *Structural rules:*

$$\forall L \forall L' \forall G \forall A \forall i. \text{nat } i \supset \text{list } L \supset \text{list } L' \\ (\forall A. \text{elem } A L \supset \text{elem } A L') \supset \text{seq}_i L G \supset \text{seq}_i L' G.$$

② *Atomic cut:* $\forall L \forall G \forall A. \text{list } L \supset \exists i. (\text{nat } i \wedge \text{seq}_i L (A \Rightarrow G)) \supset$
 $\exists i. (\text{nat } i \wedge \text{seq}_i L \langle A \rangle) \supset \exists i. \text{nat } i \wedge \text{seq}_i L G.$

③ *Specialization:*

$$\forall L \forall G \forall i. \text{nat } i \supset \text{list } L \supset \text{seq}_{(si)} L (\bigwedge G) \supset \forall x. \text{seq}_i L (Gx).$$

Encoding simply typed λ -calculus

Encode the terms and the types using:

$$arr : ty \rightarrow ty \rightarrow ty.$$

$$app : tm \rightarrow tm \rightarrow tm \quad abs : ty \rightarrow (tm \rightarrow tm) \rightarrow tm$$

The type tm is a nominal type, and ty is an ordinary type.
To encode the operational semantics, use the two-level approach as in [McDowell & Miller, TOCL 2002]

$$\begin{aligned} eval (abs T M) (abs T M) &\Leftarrow tt. \\ \bigwedge P.[eval (app M N) V &\Leftarrow eval M P \& eval (P N) V]. \\ typeof (abs T M) (ar T T') &\Leftarrow \bigwedge x.typeof x T \Rightarrow typeof (Mx) T'. \\ \bigwedge T'.[typeof (app M N) T &\Leftarrow typeof M (ar T' T) \& typeof N T']. \end{aligned}$$

Subject reduction

$L \triangleright G$ denotes the formula $\exists i.nat\ i \wedge seq_i\ L\ G$.

Theorem

Subject reduction. *The following formula is provable*

$$\forall M \forall V \forall T. \triangleright \langle eval\ M\ V \rangle \wedge \triangleright \langle typeof\ M\ T \rangle \supset \triangleright \langle typeof\ V\ T \rangle.$$

Uniqueness of typing

Let $var\ X$ denote the formula

$$\forall M \forall N. (X = app\ M\ N \supset \perp) \wedge \forall M. (X = (abs\ M) \supset \perp).$$

Let $ctx\ L$ denote the well-formedness of a context L , defined as:

$$\begin{aligned} & (\forall X \forall T. elem\ (typeof\ X\ T)\ L \supset var\ X) \wedge \\ & (\forall X \forall T_1 \forall T_2. elem\ (typeof\ X\ T_1)\ L \supset elem\ (typeof\ X\ T_2)\ L \supset T_1 = T_2.) \end{aligned}$$

Theorem

The following formula is provable:

$$\forall L \forall X \forall T_1 \forall T_2. list\ L \supset ctx\ L \supset L \triangleright \langle typeof\ X\ T_1 \rangle \supset L \triangleright \langle typeof\ X\ T_2 \rangle \supset T_1 = T_2.$$

Future work

- Extensions of LINC without the axiom $\nabla x \nabla y . B \supset \nabla y \nabla x . B$.
- Implementation: can it be done in existing proof assistants? e.g., Isabelle/Nominal package? Or implementation from scratch is required?
- Bigger case studies, e.g., the POPLmark challenge.
- Semantics of *LG*: Categorical semantics? Supports models for nominal logic?