A Definitional Approach to Higher Order Abstract Syntax
or
How I spent my time in Leicester with Isabelle.

Alberto Momigliano
Department of Mathematics and Computer Science
University of Leicester

Joint work with Simon Ambler & Roy Crole.

Yale, May 20 2002

Overview of the Talk

- HOAS, theorem proving and (co)induction.

- Background on Isabelle/HOL and co-induction.

- Introducing Hybrid.

- Using Hybrid: encoding QPL.

- A significant case-study: Howe's technique.

- Two-level reasoning with Hybrid.

What Hybrid is

- A mechanized tool, Hybrid, coded within Isabelle/HOL.

- Provides a form of *logical framework* within which the syntax of an object level logic can be adequately represented by *higher order abstract syntax* (HOAS).

- Supports *tactical theorem proving* and principles of *induction and co-induction*.

- Is *definitional*, so consistent *within a classical type theory*.

- Bottom line: ‘Hybrid syntax’ provides the user with a form of HOAS, but it is syntactic sugar for an underlying de Bruijn representation.

The Predicament

- I'm hired to work on an EPSRC grant aimed to formal verification of operational equivalence of functional programs, towards compiler optimization transformations (e.g. Benton&Kennedy's MIL-lite).

- The plan was to use an interactive system such as Isabelle/HOL and intensively use **co-inductive** reasoning.

- Need a practical *scalable* methodology, based on existing robust technology. Yet, why not extend *Twelf* with co-induction? (Yes, why not?).

- Instead, we have designed a package on top of one of the most trusted system which tries to squeeze as much HOAS as possible in a (co)inductive and interactive setting.

Hybrid vs. HOAS

- *Full* HOAS (*Twelf*, $FO\lambda^{\Delta I\!N}$).

  1. Object-level variables as meta-level bound variables.
  2. Object level contexts as hypothetical judgments.
  3. Object-level substitution as meta-level β-conversion.
  4. Two-level reasoning approach to combine HOAS and induction.

- Hybrid: implements 1. and 3. ; point 2. only when hypothetical judgments can be stratified.

- Experiments with Hybrid so far used a one-level architecture, but see later on.

Related Work

Mechanized (meta)reasoning over languages with variable bindings. Classification according to the construct chosen to model **abstraction**:

- Name-carrying syntax: abstractions are pairs "(name, expression)" and the mechanization works directly on first-order parse trees, quotiented by α-conversion e.g. [McKinna&Pollack'98]. Can be refined so that named syntax is just a notation for De Bruijn terms, which is then used as a meta-logic whose equality is α-convertibility [A. Gordon'93] .

- Gabbay & Pitts: a first-order theory of α-conversion and binding founded on name *swapping* rather than renaming. Good for meta-programming. Not yet a logical framework (but wait for FM-HOL).

- Abstractions as functions *names* $\Rightarrow$ expressions: (Despeyroux et al. '95): a way to have binders as functions on inductive (positivized via coercion) data-types, while coping with *exotic* expressions stemming from an inductive characterization of the set of names.

- Υ framework [Honsell et al. '01]: A higher-order logic extended with a set of axioms, called the *Theory of Contexts* parametric to a HOAS signature, including the reification of properties of names such as *freshness* ($\forall t{:}expr.\exists n{:}name : n \not\in t$) and **assuming** higher-order induction and recursion schemata.

- Successful for case-studies such as the π-calculus, less where negative occurrence are crucial (λ-calculus).

Related Work

- Compatible with co-induction, but need to code substitution as a object-level inductive predicate makes some of the encodings (for example, of simulation) and the reasoning every awkward.

- Hypothetical judgments only partially supported (for example Howe's 'candidate' relation).

- Abstractions as functions *expressions* $\Rightarrow$ expressions (*Twelf*, $FO\lambda^{\Delta\mathbb{N}}$).

- Inductive reasoning is conducted at an additional meta-level; co-inductive definitions have to be rewritten in an inductive fashion, exploiting the co-continuity of the said notion, when possible.

- HOAS, theorem proving and (co)induction.
- **Background on Isabelle/HOL and co-induction.**
- Introducing Hybrid.
- Using Hybrid: encoding QPL.
- A significant case-study: Howe's technique.
- Two-level reasoning with Hybrid.

Isabelle/HOL: Basics 1/2

- Isabelle: a tactic-based **generic** interactive theorem provers, based on higher-order hereditary Harrop formulae, which mechanizes inference (unification + search) for a number of objects logics, such as FOL,LCF, ZF, CTT and

- Isabelle/HOL is Paulson and Nipkow's implementation of Church's a classical type theory (see also Mike Gordon&Melhams's HOL).

- Fair amount of automation (compared to, say, Coq): a sophisticated simplifier (rewriter), an almost decent classical tableau-based theorem prover, a lot of flexibility given by a rich set of tactic(al)s.

Isabelle/HOL: Induction 2/2

- A simply-typed set theory, where sets are predicates.
- Isabelle/HOL stresses a **definitional** approach: new types are introduced by definition as subsets of existing ones.
- (Co)inductive definitions and datatypes are built via the Knaster-Tarski construction, i.e. formalizing the required fragment of (set-theoretic) fixed points theory [Paulson'96].
- Incidentally, this can be done in a very weak (and constructive) subset of HOL [Harrison'98].
- Thus, inductive clauses satisfy the **positivity** condition, so that the resulting operator is monotone [Wenzel'98].

- A *rule set* $\mathcal{R}$ is set of pairs $\langle G, a \rangle$ (notation: $a \leftarrow G$) on an universe $\mathcal{U}$, such that $a \in \mathcal{U}, G \subseteq 2^{\mathcal{U}}$.

- $Id(\mathcal{R})$ set *inductively defined* by $\mathcal{R}$ is the intersection of all $\mathcal{R}$-closed subset of $\mathcal{U}$, i.e. sets A such that for every $a \leftarrow G \in \mathcal{R}$, $G \subseteq A$ entails $a \in A$.

- Set-theoretically to prove a property $P \subseteq Id(\mathcal{R})$ by (rule) induction means showing $Id(\mathcal{R}) \subseteq P$, which in turns amount to show that P is $\mathcal{R}$-closed.

- Alternatively for $\Phi_{\mathcal{R}} : 2^{\mathcal{U}} \rightarrow 2^{\mathcal{U}}$ define $\Phi_{\mathcal{R}}(A) = \{a \in \mathcal{U} \mid a \leftarrow G \in \mathcal{R}, G \subseteq A\}$. $Id(\mathcal{R})$ is the least (pre)-fix point of this operator.

- Dually, a co-inductive set is the greatest $\mathcal{R}$-dense set or the greatest post fixed point. As a proof rule:

$$\frac{S \subseteq \Phi_{\mathcal{R}}(S)}{S \subseteq gfp(\Phi_{\mathcal{R}})}CI \quad \text{or} \quad \frac{\exists S : a \in S \qquad S \subseteq \Phi_{\mathcal{R}}(S)}{a \in gfp(\Phi_{\mathcal{R}})}CI'$$

- Example: divergence in the λ-calculusas the **greatest** relation specified by the following rules:

$$\frac{e_1 \uparrow}{e_1 \; e_2 \uparrow}d_1 \qquad \frac{e_1 \downarrow \Lambda x \,.\, e \quad e[e_2/x] \uparrow}{e_1 \; e_2 \uparrow}d_2$$

- To show $\Omega \uparrow$, where $\Omega \stackrel{\text{def}}{=} (\Lambda x \,.\, x \; x) \; (\Lambda x \,.\, x \; x)$, show there is a set S such that $\Omega \in S \wedge S \subseteq \Phi_{\uparrow}(S)$, i.e.

$$\exists S : \Omega \in S \to \Omega = e_1 \; e_2 \wedge (e_1 \in S \vee (e_1 \downarrow \Lambda x \,.\, e \wedge e \; [e_2/x] \in S))$$

- Pick $S = \{\Omega\}$: by RHS $(\Lambda x \,.\, x \; x) \downarrow (\Lambda x \,.\, x \; x)$ and $\Omega \in \{\Omega\}$.
- The ‘creative’ part of a co-inductive proof is in finding the right $\Phi_{\mathcal{R}}$ dense set; it may be as difficult as right strengthening of an inductive hypothesis.
- Valuable reasoning tool in the meta-theory of functional programming: see [Abramski’90, Milner & Tofte’91, Pitts&Stark’93, A. Gordon’95, Sangiorgi’95, Lassen’98].
- A (much nicer) proof-theoretic treatment is Coquand& **guarded induction** (circular proofs), which still needs to be applied to meta-logic for HOAS.

- HOAS, theorem proving and (co)induction.
- Background on Isabelle/HOL and co-induction.
- $\boxed{\textbf{Introducing Hybrid.}}$
- Using Hybrid: encoding QPL.
- A significant case-study: Howe's technique.
- Two-level reasoning with Hybrid.

Introducing Hybrid: λ-calculus as a datatype?

- We may think of defining a datatype for λ-calculus with constants over which we can deploy (co)induction principles, while representing variable binding through Isabelle's HOAS

$$\mathcal{E} ::= \mathsf{VAR}\ var \mid \mathcal{E}\ \$\$\ \mathcal{E} \mid \mathsf{lambda}\ (\mathcal{E} \Rightarrow \mathcal{E})$$

- There is no datatype of this form in HOL for cardinality reasons ($\mathsf{lambda} :: (\mathcal{E} \Rightarrow \mathcal{E}) \Rightarrow \mathcal{E}$ cannot be injective).

- The function space $\mathcal{E} \Rightarrow \mathcal{E}$ contains too many elements, giving rise to so-called *exotic* terms, e.g.

$$\mathsf{lambda}\ x.(\mathsf{if}\ x = \mathsf{VAR}\ 0\ \mathsf{then}\ \mathsf{VAR}\ 1\ \mathsf{else}\ \mathsf{lambda}\ z.z)$$

  The internal language of Isabelle/HOL is too rich and gives us too many ways of defining functions.

Hybrid's Infrastructure

- In the background, a Isabelle HOL datatype of de Bruijn expressions, where con provides names for constants

$$\boxed{expr ::= \mathsf{CON}\ con \mid \mathsf{VAR}\ var \mid \mathsf{BND}\ bnd \mid expr\ \$\$\ expr \mid \mathsf{ABS}\ expr}$$

- Consider $\Lambda\, V_1.\, \Lambda\, V_2.\, V_1\ V_3$ be a genuine syntax tree: In Hybrid it is rendered as $\mathsf{LAM}\, v_1.\, (\mathsf{LAM}\, v_2.\, (v_1\ \$\$\ \mathsf{VAR}\ 3))$

- This is simply a **definition for** the de Bruijn term ABS (ABS ((BND 1) $$ (VAR v3))), yet at the user level:

- bound variables correspond to (bound) meta variables; abstractions $\Lambda\, V.\, E$ correspond to expressions $\mathsf{LAM}\, v.\, e = \mathsf{lambda}\ (\lambda\, v.\, e)$;

Defining lambda

- First define $\mathsf{lbind} :: bnd \Rightarrow (expr \Rightarrow expr) \Rightarrow expr$ as the function that replaces each bound occurrence of a binding variable in the body of an abstraction with a bound index.

- The Hybrid binder $\mathsf{lambda} :: (expr \Rightarrow expr) \Rightarrow expr$ can now be **defined** as $\mathsf{lambda}\ e = \mathsf{ABS}\ (\mathsf{lbind}\ 0\ e)$

- The purpose of lbind is to transform an abstraction into the "corresponding" proper de Bruijn expression. For example $\mathsf{lambda}\ (\lambda\, v.\, \mathsf{ABS}\ (\mathsf{BND}\ 0\ \$\$\ v)) = \mathsf{ABS}\ (\mathsf{ABS}\ (\mathsf{BND}\ 0\ \$\$\ \mathsf{BND}\ 1))$

Proper Expressions and Abstractions

- We identify well-behaved subsets of $expr$ and $expr \Rightarrow expr$. The former consists of those expressions which are *proper*, i.e. the translation of legitimate λ-terms – no dangling references.

- We characterize the functions $e :: expr \Rightarrow expr$ which are *abstractions*, that is, the bodies of legitimate λ-terms.

$$
\begin{array}{rl}
 & \Longrightarrow \quad \mathsf{abst}\ i\ (\lambda\, v.\, v) \\
 & \Longrightarrow \quad \mathsf{abst}\ i\ (\lambda\, v.\, \mathsf{VAR}\ n) \\
j < i & \Longrightarrow \quad \mathsf{abst}\ i\ (\lambda\, v.\, \mathsf{BND}\ j) \ldots
\end{array}
$$

- An abstraction is produced by replacing each occurrence of a dangling index with a metavariable and then abstracting the meta variable.

Almost a Data-type

- We wish to be able to perform meta-reasoning over the Hybrid syntax, mainly case-analysis. For that, freeness properties of constructors are central.

- The set of proper expressions is *almost* a datatype.

- The images of $\mathsf{VAR}, \mathsf{BND}, \mathsf{CON}, \$\$$ and lambda are disjoint.

- $\mathsf{lambda} :: (\mathcal{E} \Rightarrow \mathcal{E}) \Rightarrow \mathcal{E}$ cannot be injective in general, but it is as injective as it needs to be:

$$[\![\, \mathsf{abstr}\ e;\ \mathsf{abstr}\ f \,]\!] \Longrightarrow (\mathsf{LAM}\, x.\, .\ e\ x = \mathsf{LAM}\, y.\, .\ f\ y) = (e = f)$$

Induction principles

$$\llbracket \, \mathsf{proper}\ u; \bigwedge n.\ P(\mathsf{VAR}\ n);$$

$$\bigwedge s\ t.\ \llbracket\ \mathsf{proper}\ s;\ \ P\ s;\ \ \mathsf{proper}\ t;\ \ P\ t\ \rrbracket \Longrightarrow P(s\ \$\$\ t);$$

$$\bigwedge e.\ \llbracket\ \mathsf{abstr}\ e;\ \ \forall t.\ \mathsf{proper}\ t \longrightarrow Pt \longrightarrow P(e\ t)\ \rrbracket \Longrightarrow P(\mathsf{LAM}\, x..\ e\ x)\ \rrbracket \Longrightarrow Pu$$

$$\llbracket\ \mathsf{abstr}\ e; \bigwedge j.P(\lambda x.\mathsf{VAR}\ j); P(\lambda x.\ x);$$

$$\bigwedge f g. \llbracket\ \mathsf{abstr}\ f;\ \ P\ f;\ \ \mathsf{abstr}\ g;\ \ P\ g\ \rrbracket \Longrightarrow P(\lambda x.f\ x\ \$\$\ g\ x);$$

$$\bigwedge f. \llbracket\ \mathsf{biAbstr}\ f; \forall n.P(\lambda x.\ f\ x\ (\mathsf{VAR}\ n)); \forall n.P(\lambda x.\ f\ (\mathsf{VAR}\ n)\ x)\ \rrbracket$$

$$\Longrightarrow P(\lambda x.\mathsf{LAM}\, y..\ f\ x\ y)\ \rrbracket \Longrightarrow P\ e$$

- Compare to the semantical justification of an analogous principle via tripos models in [Hofmann'99] and to the "context" induction *axiom* in [Honsell'01], again justified via a categorical construction.

- HOAS, theorem proving and (co)induction.
- Background on Isabelle/HOL and co-induction.
- Introducing Hybrid.
- **Using Hybrid: encoding QPL.**
- A significant case-study: Howe's technique.
- Two-level reasoning with Hybrid.

$$Q \;::=\; V_i \mid Q \supset Q' \mid \forall V_i.\, Q \mid \ldots$$

- Constants (con) and type (oo) declarations:

$$\begin{array}{rcl} con & ::= & cIMP \mid cFORALL \mid \ldots \\ oo & == & con\ expr \\ q \ \mathsf{Imp}\ q' & == & \mathsf{CON}\ cIMP\ \$\$\ q\ \$\$\ q' \text{ where } \mathsf{Imp} :: [\, oo, oo\,] \Rightarrow oo \\ \mathsf{All}\, v.\, q\, v & == & \mathsf{CON}\ cFORALL\ \$\$\ \mathsf{LAM}\, v.\, q\, v \text{ where } \mathsf{All} :: (oo \Rightarrow oo) \Rightarrow oo \end{array}$$

- Example: $\forall V_1. \forall V_2. V_1 \supset V_2$ is represented by $\mathsf{All}\, v_1. \mathsf{All}\, v_2.\ v_1\ \mathsf{Imp}\ v_2$, although the "real" underlying form is

$$\mathsf{CON}\ cFORALL\ \$\$\ (\mathsf{LAM}\, v_1. \mathsf{CON}\ cFORALL\ \$\$\ \mathsf{LAM}\, v_2. (\mathsf{CON}\ cIMP\ \$\$\ v_1\ \$\$\ v_2)$$

QPL

$\mathcal{LF}$	$\mathsf{Imp}\ \ulcorner Q \urcorner\, \ulcorner Q' \urcorner$	$\mathsf{All}\ \ (\lambda\, v.\ulcorner Q \urcorner)$
$Hybrid$	$\mathsf{CON}\ cIMP\ \$\$\ \ulcorner Q \urcorner\ \$\$\ \ulcorner Q' \urcorner$	$\mathsf{CON}\ cFORALL\ \$\$\ \mathsf{LAM}\, v.\ulcorner Q \urcorner$
$Abbrv$	$\ulcorner Q \urcorner\ \mathsf{Imp}\ \ulcorner Q' \urcorner$	$\mathsf{All}\, v.\ulcorner Q \urcorner$

- The good news: object logics too are almost a data-type: the defined constants enjoy freeness properties, much as they would if they were datatype constructors, namely

- Distinction: for example $\mathsf{All}\, v.\, q\ v \neq\ q\ \mathsf{Imp}\ q'$

- Injectivity (of higher-order constructors): it holds conditionally on their bodies being abstractions (since there the LAM binder is injective):
  $[\![\ \mathsf{abstr}\ e; \mathsf{abstr}\ f; \mathsf{All}\, v.\, e\ v = \mathsf{All}\, v.\, f\ v\ ]\!] \Longrightarrow e = f.$

- Not so good news: the type oo of formulae is merely an abbreviation for Hybrid expressions; we need to introduce "well-formedness":

$$\text{isForm (VAR } i)$$

$$[\![\ \text{abstr } p; \forall i.\, \text{isForm } (p\ \ (\text{VAR } i))\ ]\!] \implies \text{isForm (All } v.\, p\ \ v)$$

- Adequacy of encoding: if Q is a formula, isForm ($\ulcorner Q \urcorner \theta$) holds, for $\theta(v_i) = \text{VAR } i$. Usual decoding function and compositionality of substitution.

- Keeping this proof obligation is tedious: internalize it as a subtype. Current work: user specifies a binding signature, system synthesize constant definitions, freeness properties and well-formed subtype and induction principle.

Using QPL

- Encoding of an algorithm for negation normal form as an inductive relation, where Φ abbreviates $\mathsf{abstr}\ p; \mathsf{abstr}\ q$:

$$
\begin{array}{rcl}
 & & \mathsf{nnf}\ (\mathsf{VAR}\ i)\ (\mathsf{VAR}\ i) \\
 & & \mathsf{nnf}\ (\mathsf{Not}\ (\mathsf{VAR}\ i))\ (\mathsf{Not}\ (\mathsf{VAR}\ i)) \\
[\![\ \Phi;\ \forall i.\, \mathsf{nnf}\ (\mathsf{Not}\ (p\ (\mathsf{VAR}\ i)))(q\ (\mathsf{VAR}\ i))\ ]\!] & \Longrightarrow & \mathsf{nnf}\ (\mathsf{Not}\ (\mathsf{Exists}\, v.\, p\ v))\ (\mathsf{All}\, v.\, q\ v) \\
[\![\ \Phi;\ \forall i.\, \mathsf{nnf}\ (\mathsf{Not}\ (p\ (\mathsf{VAR}\ i)))(q\ (\mathsf{VAR}\ i))\ ]\!] & \Longrightarrow & \mathsf{nnf}\ (\mathsf{Not}\ (\mathsf{All}\, v.\, p\ v))\ (\mathsf{Exists}\, v.\, q\ v)\ \ldots
\end{array}
$$

- The injectivity of All via abstraction annotation crucial in deriving the appropriate inversion rules.

- Sample theorems: the algorithm yields negation normal forms and it is a functional relation.

```
Goal "nnf A B ==> !C. nnf A C -->  B=C";
be nnfSet.induct 1;
by(ALLGOALS(fast_tac(HOL_cs addEs nnf_E)));
```

Road-map

- HOAS, theorem proving and (co)induction.

- Background on Isabelle/HOL and co-induction.

- Introducing Hybrid.

- Using Hybrid: encoding QPL.

- **A significant case-study: Howe's technique.**

- Two-level reasoning with Hybrid.

Moving on: The Lazy λ-Calculus

- Up to this point, we haven't gained much w.r.t. *Twelf*, apart interactive theorem-proving at some cost.

- Coinductive predicates over the (lazy) λ-calculus:

$$
\begin{array}{rcl}
@ & :: & [\,lexp, lexp\,] \Rightarrow lexp \\
e_1 \,@\, e_2 & == & \mathsf{CON}\; cAPP \;\$\$\; e_1 \;\$\$\; e_2
\end{array}
\qquad
\begin{array}{rcl}
\mathsf{Fun}\,. & :: & (lexp \Rightarrow lexp) \Rightarrow lexp \\
\mathsf{Fun}\, x.\, e\ x & == & \mathsf{CON}\; cABS \;\$\$\; \mathsf{LAM}\, x.\, e\ x
\end{array}
$$

- *Divergence* as a co-inductively predicate, given encodings of evaluation ($\gg$) and closed expression:

$$
\begin{array}{rcl}
[\![\, \mathsf{cloExp}\ e1;\ \mathsf{cloExp}\ e2;\ \mathsf{divrg}\ e1\ ]\!] & \Longrightarrow & \mathsf{divrg}\ (e1 \,@\, e2) \\
[\![\ e1 \gg\ \mathsf{Fun}\, x.\, e\ x;\ \mathsf{abstr}\ e;\ \mathsf{cloExp}\ e2;\ \mathsf{divrg}\ (e\ \ e2)\ ]\!] & \Longrightarrow & \mathsf{divrg}\ (e1 \,@\, e2)
\end{array}
$$

- $\Omega \stackrel{\mathsf{def}}{=} (\mathsf{Fun}\, x.\, x \;@\; x) \;@\; (\mathsf{Fun}\, x.\, x \;@\; x)$ diverges:

```
Goalw [omega_def] "divrg omega";
br divrg.coinduct 1;by(Fast_tac 1);br disjI2 1;
by(fast_tac(claset() addIs eval.intrs@isexpSet.intrs
                     addss (simpset() addSolver abstr_solver unlam_defs))1);
qed "omega_diverges";
```

- Applicative simulation $\leq\ ::[\,\mathit{lexp}, \mathit{lexp}\,] \Rightarrow \mathit{bool}$:

$$\begin{array}{c} [\![\ \mathsf{cloExp}\ r;\ \mathsf{cloExp}\ s;\ \forall T.\ r \gg\ \mathsf{Fun}\, x.\, T\ \ x \longrightarrow (\mathsf{cloAbstr}\ T \longrightarrow \\ (\exists U.\ s \gg\ \mathsf{Fun}\, x.\, U\ \ x \wedge \mathsf{cloAbstr}\ U \wedge (\forall p.\ \mathsf{cloExp}\ p \to\ (T\ \ p) \leq\ (U\ \ p))))\ ]\!] \implies\ r \leq\ s \end{array}$$

```
Goal " closed s ==> s <sim> s";
br sim.coinduct 1;
by (res_inst_tac [("P","% (u,v).  u = v & closed u & closed v ")] CollectI 1);
auto(); by(fast_tac (HOL_cs addDs [clo_sub]) 1);
qed "sim_refl";
```

Contextual Equivalence as Applicative Bisimulation

- Howe's-style proof: to show simulation to be a pre-congruence, introduce a *candidate* relation $\Gamma \vdash s \leq^{\bullet} t$, which is easily shown to be pre-congruent and prove that it coincides with similarity.

- What we would like to write is:

$$\frac{\forall y.(\forall m.\, y \leq m \longrightarrow y \leq^{\bullet} m) \longrightarrow N\, y \leq^{\bullet} N'\, y \qquad \mathsf{Fun}\, v.\, N'\, v \leq q}{\mathsf{Fun}\, v.\, N\, v \leq^{\bullet} q}$$

- But we can't. Thus, we need to define *open similarity* $\Gamma \vdash s \leq^{\circ} t$ on open expressions, via explicit environments. Two open terms are related if their λ-closures are similar.

- Then proves that open similarity is a pre-congruence (duplication of work and back to first-order encodings).

Howe's-style Proof

$$\Gamma \vdash \ \mathsf{VAR}\, x \leq^\circ \ m \quad \Longrightarrow \quad \Gamma \vdash \ \mathsf{VAR}\, x \leq^\bullet \ m$$

$$[\![\ \forall y.\ \Gamma, y \vdash \ N\,(\mathsf{VAR}\ y) \leq^\bullet \ N'\,(\mathsf{VAR}\ y);$$
$$\mathsf{abstr}\ N; \mathsf{abstr}\ N';\ \Gamma \vdash \ \mathsf{Fun}\, v.\, N'\, v \leq^\circ \ q \ ]\!] \quad \Longrightarrow \quad \Gamma \vdash \ \mathsf{Fun}\, v.\, N\, v \leq^\bullet \ q \ldots$$

- Properties: $(\leq^\bullet; \leq^\circ) \subseteq \leq^\bullet$ $\qquad \leq^\bullet \subseteq Id$ $\qquad \leq^\circ \subseteq \leq^\bullet$ and

$$\frac{\Gamma, y \vdash \ s \leq^\bullet \ s' \qquad \Gamma \vdash \ t \leq^\bullet \ t'}{\Gamma \vdash \ s[t/y] \leq^\bullet \ s'[t'/y]} Sub$$

- If $[] \vdash \ \mathsf{Fun}\, x.\, E\, x \leq^\bullet \ p$, then $p \gg \ \mathsf{Fun}\, y.\, F\, y$ and for every closed p $[] \vdash \ E\, p' \leq^\bullet \ F\, p'$

- If $[] \vdash \ p \leq^\bullet \ q$ and $p \gg \ v$, then $[] \vdash \ v \leq^\bullet \ q$.

- $[] \vdash \ p \leq^\bullet \ q \Longrightarrow [] \vdash \ p \leq^\circ \ q$

Road-map

- HOAS, theorem proving and (co)induction.
- Background on Isabelle/HOL and co-induction.
- Introducing Hybrid.
- Using Hybrid: encoding QPL.
- A significant case-study: Howe's technique.
- $\boxed{\textbf{Two-level reasoning with Hybrid.}}$

- Moral: HOAS without hypothetical judgments falls short. Idea: adapt to Hybrid $FO\lambda^{\Delta\mathbb{N}}$ two-levels architecture, in analogy (but *definitionallly*) with [Felty'02] + reflection [Weyrauch'80].

1 A specification logic (a sequent calculus) where (co)inductive reasoning is performed (implemented as a Isabelle/HOL inductive definition).

2 Separated from the object logic (here the lazy -calculus) as a Hybrid object-logic (in other terms a semi-datatype).

- What about the judgments over the object logic? Some (those non-monotonic) are implemented via `prog` clauses, for example typing, isExp s. . .

Two-Levels Reasoning in Hybrid

- Coinductive ones directly as in Hybrid, i.e. simulation $s \leq t$.
- Mixed ones (Howe's relation) via **reflection**.
- As in Felty, **definitional** reflection simulated by inversion on (co)inductive definitions aided by Hybrid frenees of constructors and extensionality properties at higher types; not assumed (Hoffmann/Honsell), but proven via Hybrid infrastructure.
- $FO\lambda^{\Delta I\!N}$ number induction replaced by HOL's structural induction on sequents.
- Inherit all the automation (simplifier, classical reasoner) of Isabelle.

```
(* prop: HOL datatype for first-order logic 'a la Weak HOAS *)
prop ::= and prop prop | <> atm | All 'a => prop | \/ 'a => prop ...

prog :: atm => prop ==> bool                              infix  '<=='
reflect :: atm ==> set ==> bool
seq  :: nat => prp list => prp => bool
Gamma |- p == ? i . seq Gamma i p.

inductive seq intrs
sbc  ''[| a <== G; seq i Gamma G|] ==> seq (i+1) Gammma < a >''
srf  ''[| reflect a a'; a'|] ==> seq (i+1) Gammma < a >''
sall ''[| ! y.  seq i Gamma (p y)|] ==> seq (i+1) Gammma (All x . p x)''
...

(* Hybrid semi-datatype for the lazy lambda calculus as before *)

sim :: uexp => uexp => bool                               infix 'sim'
```

Two-Levels Reasoning in Hybrid: tentative

```
coinductive sim intrs
sf '' [|!t.( [] |- r >> Fun x . t x) --> (abstr t -->
            (? u. ([] |- s >> Fun x . u x) & abstr u &
                       (!p. (t p) sim (u p))))|] ==> r sim s''
inductive reflect
ref_sim     ''reflect (p Rsim q) ((p,q) : sim)''

datatype atm = howe uexp * uexp | Rsim uexp * uexp | isexp uexp

inductive prog intrs
hap ''(M1 @ M2 howe N) <==   <M1 howe M1'> and
                             <M2 howe M2'> and
                             <M1' @ M2' Rsim N>''
hf  ''(Fun x . M x howe N) <==
              <Fun x. M' x Rsim N> and
              (All x. (All Q.  (x Rsim Q) imp (x howe Q))
                                imp <(M x) howe (M' x)>)''
(* Sample goal *) Goal ''[] |- <isexp M>  ==>  [] |- <M howe M>''
```

Conclusions

- A logical framework giving a definitional account of HOAS, compatible with tactic based TP and co-induction.

- Crucial properties of expressions are proved via the de Bruijn representation, rather than postulated as axioms.

- One-level architecture sufficient when hypothetical judgments are rare.

- $FO\lambda^{\Delta\mathbb{N}}$-in-Hybrid is the way forward, though direct usage of co-induction relies on the still untested amalgamation of predicates at different levels.

- Examples, examples, examples.

More Info

- `www.mcs.le.ac.uk/amomigliano/isabelle/hybrid2.html`:
  - TPHOLs 2002 paper and extended technical report
  - Proof scripts for the infrastructure
  - Examples suite: QPL, λ-calculus, higher-order π-calculus.

- `www.mcs.le.ac.uk/amomigliano/isabelle/Howe`:
  - LFM 2002 paper and code for Howe's proof.