A Hybrid Encoding of Howe's Method for Establishing Congruence of Bisimilarity

Alberto Momigliano
Department of Mathematics and Computer Science
University of Leicester

Joint work with Simon Ambler & Roy Crole.

LFM Workshop, July 26 2002

Towards Multi-Level Reasoning in Hybrid

Caution: work in progress

Overview of the Talk

- Background on applicative simulation and Howe's proof.

- Introducing Hybrid.

- Using Hybrid:

  1. One-level architecture

  2. Multi-level reasoning with Hybrid.

- What's next.

- Formal verification of observational equivalence of functional programs, towards compiler optimization transformations (e.g. Benton&Kennedy's MIL-lite).

- Morris-style notion difficult to handle; better one is Abramski's **applicative (bi)simulation**: Let M, M', N, N', Q be closed terms in the λ-calculus:

$$M \leq N \quad \text{iff} \quad \forall M'. M \downarrow \lambda x\,.\,M' \rightarrow \exists N' N \downarrow \lambda x\,.\,N' \\ \wedge \forall Q.\ M'[Q/x] \leq N'[Q/x]$$

- Extended to every terms taking the λ-closure: if $x_1 \ldots x_n \vdash M_i$ is an open term, then

$$M_1 \leq^{\circ} M_2 \quad \text{iff} \quad \forall \overline{M_n} \text{ closed } M_1[\overline{M_n}/\overline{x_n}] \leq M_2[\overline{M_n}/\overline{x_n}]$$

Observational Equivalence

- Not an inductive, but a **coinductive** notion, inspired by Milner& Park's ones from concurrency.

- Leaving concurrency aside, valuable reasoning tool in the meta-theory of functional programming: see [Abramski'90, Milner & Tofte'91, Pitts&Stark'93, A. Gordon'95, Sangiorgi'96,Lassen'98].

- Example (Pitts'98): equivalence of programs over lazy data-types: if s is a stream, check: $filter\ p\ (map\ f\ s) \leq\ map\ f\ (filter\ (p\ \circ f)\ s)$

- This can get arduous: what we need is to establish simulation to be a **congruence** and then reason equationally,

Contextual Equivalence as Applicative Bisimulation

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

$$\frac{\Gamma, x \vdash x \leq^\circ m}{\Gamma, x \vdash x \leq^\bullet m}$$

$$\frac{\Gamma, x \vdash N \leq^\bullet N' \qquad \Gamma \vdash \lambda x . N' x \leq^\circ q}{\Gamma \vdash \lambda x . N x \leq^\bullet q}$$

$$\frac{\Gamma \vdash m_1 \leq^\bullet m_1' \qquad \Gamma \vdash m_2 \leq^\bullet m_2' \qquad \Gamma \vdash (m_1' \, m_2') \leq^\circ n}{\Gamma \vdash (m_1 \, m_2) \leq^\bullet n}$$

- 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 $\cdot \vdash p \leq^\bullet q$ and $p >> v$, then $\cdot \vdash v \leq^\bullet q$.

Implementation Requirements for Howe's-Style Proof

- Support for binding constructs.
- Support for proofs by structural induction over open terms (e.g. the candidate relation $M \leq^{\bullet} N$.
- Support for proof by co-induction, typically (bi)simulation.
- Ideally, this should be done with Higher-Order Abstract Syntax (HOAS).
- Sadly, no existing system offer all the above features in a satisfactory way.

The Four (?) Tenets of HOAS

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.

   (a) First-level to specify the object language

   (b) Second-level to reason (co)inductively over the latter

- Isabelle: a tactic-based **generic** interactive theorem provers, which mechanizes inference for a number of objects logics, such as Church's HOL.

- Fair amount of automation: simplifier, tableau-based theorem prover, flexibility via tactic(al)s.

- (Co)inductive definitions and data-types are built via the Knaster-Tarski construction [Paulson'96]. Thus, inductive clauses satisfy the **positivity** condition, so that the resulting operator is monotone [Wenzel'98].

- This rules out data type for λ-terms, as well as judgments such as $M \leq^{\bullet} N$. Hence need to use a first-order (un)named syntax. Similarly:

State of the Art

- PVS has support for co-algebraic reasoning, but using it in this setting seems foolish (apologies to Ford & Mason).

- Coq requires positive types: on the bright side, it implements coinductive reasoning via Coquand's guarded induction.

- Better: Υ framework [Honsell et al. '01], still substitution coded at object-level inductive type and no support for non stratified hypothetical judgments.

- Twelf: the right one, but co-inductive definitions have to be rewritten in an inductive fashion, exploiting the co-continuity of the said notion, when possible. Theory needs extending, both at LF and $\mathcal{M}^{\omega}$ level.

Road-map

- Background on applicative simulation and Howe's proof.

- $\boxed{\textbf{Introducing Hybrid.}}$

- Using Hybrid:

  1. One-level architecture

  2. Multi-level reasoning with Hybrid.

- What's next.

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.

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 $\mathsf{ABS}\ (\mathsf{ABS}\ ((\mathsf{BND}\ 1)\ \$\$\ (\mathsf{VAR}\ \mathtt{v3})))$, 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 then make it into a subtype.

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

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

- 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:

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

Road-map

- Background on applicative simulation and Howe's proof.

- Introducing Hybrid.

- $\boxed{\textbf{Using Hybrid:}}$

  1. One-level architecture

  2. Multi-level reasoning with Hybrid.

- What's next.

$$e \quad ::= \quad x \mid e\, e' \mid \Lambda x.\, e$$

- Constants (con) and type ($lexp$) declarations:

$$\begin{array}{rcl}
con & ::= & cAPP \mid cABS \\
lexp & == & con\, expr \\
e \mathbin{@} e' & == & \mathsf{CON}\ cAPP \ \$\$\ e\ \$\$\ e' \text{ where } \mathbin{@} :: [\,lexp, lexp\,] \Rightarrow lexp \\
\mathsf{Fun}\, v.\, e\, v & == & \mathsf{CON}\ cABS \ \$\$\ \mathsf{LAM}\, v.\, e\, v \text{ where } \mathsf{Fun} :: (lexp \Rightarrow lexp) \Rightarrow lexp
\end{array}$$

- Example: $\Lambda v_1 \Lambda v_2 . v_1\ v_2$ is represented by $\mathsf{Fun}\, v_1.\, \mathsf{Fun}\, v_2.\, v_1 \mathbin{@} v_2$, although the “real” underlying form is

$$\mathsf{CON}\ cABS\ \$\$\ (\mathsf{LAM}\, v_1.\, \mathsf{CON}\ cABS\ \$\$\ \mathsf{LAM}\, v_2.\, (\mathsf{CON}\ cAPP\ \$\$\ v_1\ \$\$\ v_2)$$

Encoding the Syntax of the λ-calculus

- The encoding has freeness properties, much as they would if they were datatype constructors:

  Distinction It holds $\mathsf{Fun}\, v.\, q\ v \neq q\ @\ q'$

  Injectivity (of higher-order constructors): it holds conditionally on their bodies being abstractions:
  $[\![\ \mathsf{abstr}\ e; \mathsf{abstr}\ f; \mathsf{Fun}\, v.\, e\ v = \mathsf{Fun}\, v.\, f\ v\ ]\!] \Longrightarrow e = f.$

- the type *lexp* of formulae is merely an abbreviation for Hybrid expressions; we need to introduce “well-formedness”to ensure adequacy.

Encoding Judgments: the One-Level View

- Inductive definition of evaluation:

$$\mathsf{abstr}\ E \Longrightarrow \mathsf{Fun}\, x.\, E\ x \gg \mathsf{Fun}\, x.\, E\ x$$

$$[\![\ p_1 \gg \mathsf{Fun}\, x.\, E\ x;\ \mathsf{abstr}\ E;\ (E\ p_2) \gg v\ ]\!] \Longrightarrow p_1\ @\ p_2 \gg v$$

- The `abstr` annotation allows the system to derive the right inversion principle:

```
["[| Fun ?E >>> ?v;
  !!E. [| abstr E; lambda ?E = lambda E; ?v = Fun E |] ==> ?P |]
      ==> ?P",
"[| ?e @ ?e1.0 >>> ?v;
   !!N. [| ?e >>> Fun N; abstr N; N ?e1.0 >>> ?v |] ==> ?P |] ==> ?P"]
```

- We get the expected structural induction principle.

Encoding Judgments: the One-Level View

- Remember the lambda clause for Howe's:

$$\frac{\Gamma, x \vdash N \leq^{\bullet} N' \qquad \Gamma \vdash \lambda x\,.\,N'\,x \leq^{\circ} q}{\Gamma \vdash \lambda x\,.\,N\,x \leq^{\bullet} q}$$

- What we would like to write is the following, but we can't, because of negative occurrences:

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

- Then we need to introduce explicit environments:

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

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

Encoding judgments: the One-Level View

- Cascade effect: the neat formulation of simulation:

$$\llbracket\ \mathsf{cloExp}\ r;\ \mathsf{cloExp}\ s;\ \forall T.\ r >> \ \mathsf{Fun}\, x.\, T\ x \longrightarrow (\mathsf{cloAbstr}\ T \longrightarrow$$
$$(\exists U.\ s >> \ \mathsf{Fun}\, x.\, U\ x \wedge \mathsf{cloAbstr}\ U \wedge (\forall p.\ \mathsf{cloExp}\ p \rightarrow\ (T\ p) \leq\ (U\ p))))\ \rrbracket \quad \Longrightarrow \quad r \leq\ s$$

- Needs extension to open terms, where `subst` is free Hybrid's VAR (here it starts to get ugly):

```
s <= t ==> [] |- s <opensim> t
[|isExp s; isExp t; ~(n mem env);
   ! p.  cloExp p --> env |- subst s n p <opensim> subst t n p|]
        ==>  n # env |- s <opensim> t
```

- We need to prove that open simulation is substitutive.

- Only then we can prove the same for Howe's, but it's a pain (and very low-level).

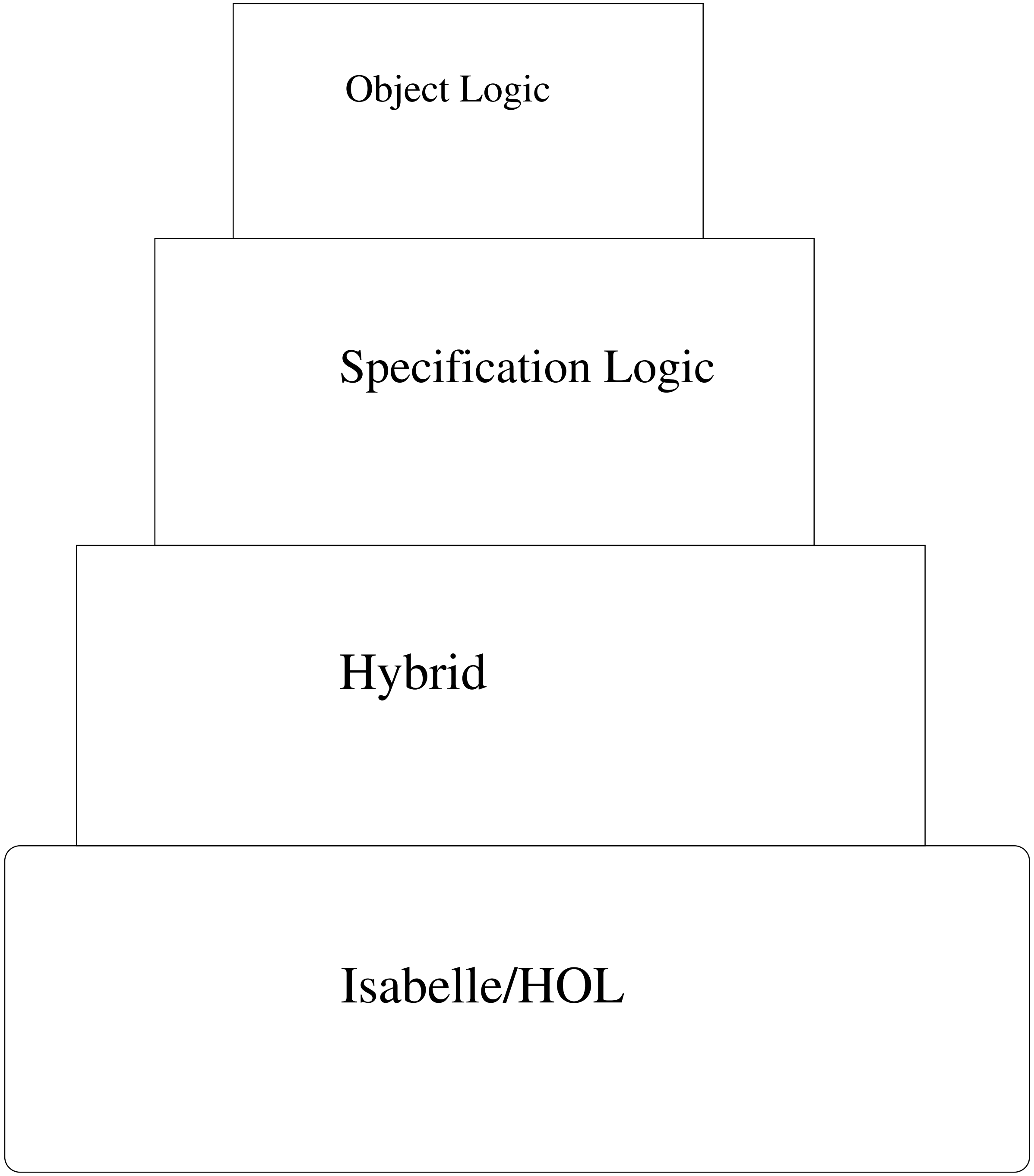
This Really Stinks

- Compare with the Twelf proof, which goes thru in a sec:

```
%theorem hsubst : forallG  (pi {x:exp}{u:{M:exp} sim x M -> howe x M})
                  forall* {M: exp -> exp}{M': exp -> exp}
                          {N: exp}{N':exp}
forall  {D1: {x:exp} ({M:exp} sim x M -> howe x M) -> howe (M x) (M' x)}
                          {D2: howe N N'}
                  exists  {E:  howe (M N) (M' N')}
                  true.
%prove 4 D1 (hsubst D1 _ _).
```

- Moral: HOAS without hypothetical judgments falls short.

- Idea: adapt to Hybrid $FO\lambda^{\Delta I\!N}$ two-levels architecture, in analogy with [Felty'02], where **definitional** reflection ($\mathcal{D}-L$) is simulated by inversion on inductive definitions.



Syntax: M @ N, Fun x . E x
Semantics: evaluation/typing
via prog clauses: A <== G

Sequent Calculus: Gs |– B : i
complete induction for
hypothetical judgments

"Datatype" for the l–calculus
primitives: $$, LAM, VAR n
freeness of constructors

Tactics/Simplifier
induction principles for
stratified judgments

Two-Levels Reasoning in Hybrid

- Inherit all the automation (simplifier,reasoner) of Isabelle.

- Hybrid frenees of constructors and extensionality properties at higher types; not justified semantically.

- Non stratifiable judgments of the object logic are implemented via `prog` clauses, i.e. $p \leq^{\bullet} q$, isExp s...

- Stratified judgments (example evaluation) can be pushed at the Hybrid leve, so that $FO\lambda^{\Delta I\!N}$ number induction is replaced by HOL's structural induction.

- Code coinductive ones directly in Hybrid. Use *reflection* to inject coinductive theorems at the `prog` level.

Two-Levels Reasoning in Hybrid


```
datatype prp    = at atm             ("/< (_) />" [40] 40)
                | rat ratm           ("/<< (_) />>" [40] 40)
                | AND  prp prp       (infixr "&&" 35)
                | forall "(proper => prp)"      (binder "all " 190)
        ...

inductive seq intrs
[| A <-- Body; Gamma |- Body ::: i|] ==> Gamma |-  < A >  ::: Suc i
[| < A >  mem Gamma|]                ==> Gamma |-  < A>  ::: i
[| reflect A MA; MA |]               ==> Gamma |-  << A >>  ::: i
[| A # Gamma |- B ::: i |]           ==> Gamma |- A -> B ::: Suc i
[| !x.  Gamma |- B x ::: i |]        ==> Gamma |- all x. B x ::: Suc i
...
inductive Meval intrs
[|abstr E|]  ==> Fun x . E x >>> Fun x.  E x
[|p1 >>> Fun x . N x; abstr N;  N p2 >>> v|] ==> p1 @ p2 >>> v

inductive reflect intrs  (* ::  "[ratm,bool] => bool" *)
 reflect (M sim N) ( M <MMsim> N)

coinductive MMsim intrs
[|! T.    R >>> Fun x . T x   --> ( abstr T  -->
       (? U.   S >>> Fun x . U x  & abstr U  &
       (! p.   (T  p) <MMsim> (U  p)  )))|] ==> R <MMsim> S
```

```
prog :: [atm,prp] => bool    ("((_) /<-- (_) )" [20,20] 20)

inductive prog intrs
[| abstr E |] ==> Fun E >> Fun E <-- tt
[| abstr R |] ==> M @ N >> V
                  <-- < M >> Fun R > && < R N >> V >

[| abstr E |] ==> Fun E ::: ar T1 T2
                  <-- all x .  < x ::: T1 > -> <  E x ::: T2 >
                   M @ N ::: T
                  <-- < M ::: ar U T > && <  N ::: U >

(* equivalence between meta-level and prog level evaluation *)

Goal " P >>> V ==> [] |--  < P >> V >";
be Meval.induct 1;
by(ALLGOALS(fast_tac(HOL_cs addIs sseqIs@prog.intrs)));
qed "Meval_eval";

Goal  "ALL i P V .  [] |- < P >> V > ::: i -->   P >>> V";
br allI 1;
by (rtac nat_less_induct 1);
...
```

Two-Levels Reasoning in Hybrid


```
(* subject reduction *)

Goal "ALL i P V. [] |- < P >> V > ::: i -->
            ALL T. [] |-- < P ::: T > --> [] |-- < V ::: T > ";
br allI 1;
by (rtac nat_less_induct 1);
...
Goal "P >>> V  ==>
           ALL T. [] |-- < P ::: T > --> [] |-- < V ::: T > ";
be Meval.induct 1;
...

(* reflexivity of simulation at meta-level *)
Goal " S <MMsim> S";
br MMsim.coinduct 1;
by (res_inst_tac ["P","% (u,v). u = v "] CollectI 1);
auto();
qed "MMsim_refl";

(* reflection at the object level *)
Goalw[exseq_def] " [] |-- << M sim M >> ";
by(fast_tac(HOL_cs addIs [ MMsim_refl,seq.srf,reflect.rfl_ind]) 1);
qed "sim_refl";
```

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.
- $FO\lambda^{\Delta I\!N}$-in-Hybrid is the way forward.

What's Next

- Current work:

  1. M&M's explicit eigenvariable encoding for induction over open terms required to do Howe's proof the right way.

  2. At the infrastructire level, generalization to n-ary abstraction and definitions by primitive recursion.

- Future work:

  1. Different specification logics, say linear and/or with the "New" quantifier.

  2. Co-induction via circular proofs: take as co-inductive the very definition of sequents and move simulation at `prog` level.

More Info

- `www.mcs.le.ac.uk/amomigliano/isabelle/hybrid2.html`:
  - TPHOLs 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`:
  - Code for one-level Howe's proof.
  - Code for two-level architecture and examples.