# Simple Compiler Verification in a Hybrid Logical Framework

Alberto Momigliano
Department of Mathematics and Computer Science
University of Leicester

Joint work with Simon Ambler

Nottingham, 06/12/2002

## Overview of the Talk

- The big issue: a logical framework which combines HOAS, interactive theorem proving and (co)induction.

- Introducing and using Hybrid: case study Mini-ML.

- Two-level reasoning with Hybrid.

  1. Type preservation

  2. Compilation

## Logical Frameworks

- **Logical Framework:** Meta-language for the specification, implementation and verification of the (meta)theory of deductive systems.

- For example, present the static and dynamic (natural) semantics of a functional programming language such as Mini-ML. There are two levels:

  1. Reasoning **within** a specified system: Implementation and execution of algorithms. For example judgments for evaluation ($e \Downarrow v$) and type inference ($\Gamma \vdash e : \tau$).

  2. Reasoning **about** a the meta-theoretic properties of a deductive system. For example the proof of subject reduction:

$$e \Downarrow v \text{ and } \vdash e : \tau \text{ then } \vdash v : \tau$$

## Brief History

- Automath [de Bruijn'68].

- Theory of Judgments [Martin-Löf'85].

- (Edinburgh) Logical Framework [Harper, Honsell & Plotkin'87], implemented as logic programming language in Elf [Pfenning'89,91] or as proof checker for PCC $LF_i$ [Necula'96]

- Calculus of Constructions, implemented in Lego [Pollack'94] or Coq [Dowek et al '99].

- Hereditary Harrop Formulas, [Nadathur, Miller, & Scedrov'87] [Paulson'87], realized in λProlog [Nadathur & Miller'88] and Isabelle [Paulson'89].

## How Abstract is your Encoding?

- The framework must be able to support encodings of informal (mathematical) entities as abstract as possible, do not introduce ad hoc low-level notions.
- Ideally a formalized proof of a math theorem should try to avoid technical lemmas that are foreign to the mathematics of the problems.
- Example: subject reduction for Mini-ML: the only lemmas the informal proof use is exchange and substitution for typing.
- Accordingly, the formal proof should be from first principles only (in particular, no lemmas about lists, say invariance of typing under permutation of assumptions).
- Maybe I'm suggesting something like an actual **metric** for encodings.

## Case Study

- EPSRC grant: formal validation of the effect-based transformation performed by the MLj compiler for SML into Java bytecode ([Benton & Kennedy '98,'99], as (contextual) equivalence preserving transformations.

- Compilation into a typed intermediate language with computational effects.

- Need to devise techniques which are truly **scalable** to such languages. The latter is a function of how *abstract* the encoding can be.

- We propose a new logical framework (Hybrid), which aims to marry HOAS and interactive theorem proving with a traditional HOL-like set-up.

- This case study: **compilation** of a (effect-free, for now) fragment of ML (Mini-ML). Prove the compiler sound and complete.

## Encoding Mini-ML

- The prototypical model of a (strict) applicative language [Kahn'86]. In this talk we are going to use this fragment:

$$\begin{array}{lcll} \textit{Types} & \tau & ::= & \mathbf{i} \mid \tau_1 \text{ arrow } \tau_2 \\ \textit{Terms} & e & ::= & x \mid \mathbf{fun}\, x\,.\, e \mid e \,@\, e' \mid \mathbf{fix}\, x\,.\, e \end{array}$$

- $\mathbf{fix}\, x\,.\, E$ (and $\mathbf{fun}\, x\,.$ ) is a **binder** (equivariant notion). It induces an equivalence class under $\alpha$-renaming, and requires capture avoiding substitutions: encoded how? This is the crucial choice, as we argue that this will affects the way judgments will be represented.

- Message of this talk: yes, $\alpha$-equivalence is easy, but also only the tip of the proverbial iceberg.

- Name-carrying syntax: here binders take pairs “(name, expression)” and we work directly on first-order parse trees, quotiented by $\alpha$-conversion: fix :: $var * exp \Rightarrow exp$. Incredibly, people do go thru this ([Ford & Mason’02], [Vestegaard ’01], while the essence of of is Gabbay and Pitts’ FM sets [ Gabbay and Pitts’99]

- De Bruijn syntax [DeBruijn72]: $\alpha$-equivalence is identity, but the handling of there pointers is quite a distraction.

- Toward higher-order syntax: fix is a function from *var* to expressions: [Despeyroux’95]. Let VAR be a coercion from vars to expressions. Then fix :: $(var \Rightarrow exp) \Rightarrow exp$. Example: $\ulcorner \mathbf{fix}\, x\,.\, x \urcorner = \mathrm{fix}(\lambda x. VAR\ x)$.

- Benefits: object logic functional space is mapped to the function space of a meta-language to which $\alpha$-equivalence is delegated.

## Encoding Mini-ML



- Use a $\lambda$-calculus as a meta-language. It has a single binder $\lambda$, from which any other one is definable [Church'40]. A $\lambda$-theory has built-in $\alpha$-equivalence, well-behaved substitutions and more (actually too much):

- **Exotic terms**: is *var* an inductive set? It looks like: one needs an infinite supply of names and to tell them apart (freeness properties).

- The numerals are an obvious model, however:

- There exists legal terms in the meta-language, which do not represent any OL terms. For example: fix$(\lambda x. if\ x = 0\ then\ 0\ else\ 1)$ is well-typed, but its image is not a Mini-ML terms.

## Encoding Mini-ML



- Collapse of **Adequacy of Encodings**; one-to-one correspondence between meta-L and OL is lost.

- A "fix" [Honsell et al. '01]; do not take the set of names as inductive, but assume as **axioms** the relevant properties (freeness, enumerability etc. . . ).

- In my book, a bit of a cheat; moreover we're still waiting for consistency proof (complicated by Coq's extensionality issues).

- Moreover, even if consistent this falls short to what I believe is the real goal: identifying the above function space with $exp \Rightarrow exp$. That's what we call **Full HOAS**. To see why, let's move to encoding **judgments**, such as the *operational semantics* of this language.

- Big step (strict) evaluation $e \Downarrow v$:

$$\frac{e_1 \Downarrow \textbf{fun}\, x\,.\, e_1' \qquad e_2 \Downarrow v_2 \qquad [v_2/x]e_1' \Downarrow v}{e_1 \mathbin{@} e_2 \Downarrow v}\ \textbf{ev\_app}$$

$$\frac{}{\textbf{fun}\, x\,.\, e \Downarrow \textbf{fun}\, x\,.\, e}\ \textbf{ev\_lam} \qquad \frac{[\textbf{fix}\, x\,.\, e/x]e \Downarrow v}{\textbf{fix}\, x\,.\, e \Downarrow v}\ \textbf{ev\_fix}$$

- Key notion is **substitution** of expression for expression $\textbf{ev\_fix}, \textbf{ev\_app}$.

- If the body of a fix point is encoded as a function $E :: exp \Rightarrow exp$, then $E(\textbf{fix}\, x\,.\, E\ x)$ is well-typed at $exp$. Thus OL substitution = Meta-L β-conversion. One more notion that can be delegated to the meta-L.

- More savings in for verification: potentially, no need for a substitution lemmas.

## Operational Semantics



- If you want to execute such a judgment, you just need an interpreter for Higher-Order Horn (actually Hereditary Harrop) clauses ($\lambda$Prolog, Elf).

- But what about **meta-reasoning** over those specifications? For example, is evaluation deterministic?

- In textbooks, e.g. [Winskel], this relation, as well as the data-structures involved, are **inductive**, which depending on the environment can be formalized either set or type-theoretically.

- The catch: HOAS and induction are not best friends. Back to the basics (here (Isabelle HOL) set theory, but an analogous argument holds for inductive type theories).

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

- $Id(\mathcal{R})$, the 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 $a \longleftarrow G \in \mathcal{R}$, $G \subseteq A$ 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.

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

- This is the basis for the introduction rules, elimination rules (case analysis) and structural induction for a deductive system.

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

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

- Not everything is inductive. Check the negative occurrence of `exp` in this LF-like declaration:

```
expr : type.          %kind declaration
@   : expr -> expr -> expr.
fix : (expr -> expr) -> expr.
```

- This is **not** a datatype definition: `datatype` $expr = \mathbf{fix}\ (expr \Rightarrow expr) \mid \ldots$ because $expr$ occurs negatively, making the associate operator non-monotone. So you can run the following fragment as a logic program (note how substitution is realized via meta-level application):

```
eval : expr -> expr -> type.
ev_lam  : eval (lam E) (lam E).
ev_fix  : eval (fix E) V  <- eval (E (fix E)) V.
```

- But you cannot reason about it. Suppose you have the goal `eval (lam E) F ==> F = (lam E).Informally,` this is provable via an inversion step, but there is no immediate case analysis principle available.

- The notion of **partial** inductive definition [Halnass'91] allows for a proof-theoretical view of that, but induction is still unaccounted for.

- Full benefits of HOAS can be enjoyed only when it is paired with hypothetical and parametric judgments. This is exemplified by the typing judgments:

$$\frac{\Gamma, x{:}\tau \vdash e : \tau'}{\Gamma \vdash \mathbf{fun}\ x\,.\ e : \tau \text{ arrow } \tau'}\ \mathbf{tp\_lam} \qquad \frac{\Gamma, x{:}\tau \vdash e : \tau}{\Gamma \vdash \mathbf{fix}\ x\,.\ e : \tau}\ \mathbf{tp\_fix}$$

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}\ \mathbf{tp\_var} \qquad \frac{\Gamma \vdash e_1 : \tau' \to \tau \qquad \Gamma \vdash e_2 : \tau'}{\Gamma \vdash e_1\ @\ e_2 : \tau}\ \mathbf{tp\_app}$$

- By using Schroeder-Heister's *higher-order* natural deduction, contexts ( $\vdash$) are **implicit**, i.e. their handling is deletegated to the meta-logic:

$$\dfrac{\begin{matrix} \overline{x : \tau}^{\,u} \\ \vdots \\ (E\ x) : \tau \end{matrix}}{\mathbf{fix}(\lambda x.\, E\ x) : \tau}\ \mathbf{tr\_fix}^{x,u} \qquad \text{no rule for } \mathbf{tp\_var} \qquad \dfrac{\begin{matrix} \overline{x : \tau_1}^{\,u} \\ \vdots \\ (E\ x) : \tau_2 \end{matrix}}{\mathbf{fun}(\lambda x.\, E\ x) : \tau_1 \text{ arrow } \tau_2}\ \mathbf{tr\_lam}^{x,u}$$

- In particular, *weakening* for the typing judgment is implicit by the assumption/discharge rules of natural deduction and does not need proving. In general, this applies to **all** judgment over terms in **context**.

- Furthermore, this obliterates the distinction between closed and open terms, where in the informal practice a notion is defined over *closed* terms and then extended canonically to open ones (example, applicative bisimulation).

- Do not underestimate the power of hypothetical-parametrical judgments in formal verification, see for example the fully automatic proof of the Church-Rosser and cut elimination theorem in *Twelf*.

- No free lunch: Meta-reasoning in their presence is been a long standing problems, only recently solved by Schürmann in his thesis [2000].

## The State of the Art

- *Elf*: specify a meta-theorem as a relation between judgments, induced by operational semantics of a logic programming-like interpretation. Finally, external checks (termination, moded-ness, totality) verify that the given relation is indeed a realizer for that theorem.

- *Twelf* devises an explicit (meta-)meta-logic for reasoning (inductively) about logical frameworks, in a fully automated way. $\mathcal{M}_\omega$ is a constructive first-order logic, whose quantifiers range over possibly open LF object over a signature.

- $FO\lambda^{\Delta I\!N}$ [Miller & McDowell '02]: higher-order intuitionistic logic augmented with definitional reflection to provide support for case analysis and induction on natural numbers. Inside this meta-language they develop a *specification logic* (SL) which in turn it is used to specify the *object-logic* (OL) under study.

- Unfortunately, we have to live with Isabelle HOL:

- 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), a simple-minded classical tableau-based theorem prover, a lot of flexibility given by a rich set of tactic(al)s.
- Still, no intuitionistic (conservative) fragment, no real notion of proof-terms, no interface to state-of-the art (resolution) theorem provers.

## Isabelle/HOL: Induction 

- 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].

- Thus, inductive clauses satisfy the **positivity** condition, so that the resulting operator is monotone [Wenzel'98].

- So how do we squeeze HOAS, hypothetical judgments and (co)induction in?

## Road-map

- The big issue: a logical framework which combines HOAS, interactive theorem proving and (co)induction.

- $\boxed{\textbf{Introducing and using Hybrid: case study Mini-ML}}$.

- Two-level reasoning with Hybrid:

  1. Type preservation

  2. Compilation

# Hybrid in short



- A mechanized tool, Hybrid, coded on top of Isabelle/HOL.

- A $\lambda$-calculus which 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 this is just syntactic sugar for an underlying de Bruijn representation.

- Hybrid implements a model of the the untyped $\lambda$-calculus with constants.

$$\begin{array}{rclcrcl} \mathsf{CON} & :: & con \Rightarrow expr & \quad & \mathsf{VAR} & :: & nat \Rightarrow expr \\ \$\$ & :: & expr \Rightarrow expr \Rightarrow expr & & \mathsf{LAM}\,. & :: & (expr \Rightarrow expr) \Rightarrow expr \end{array}$$

- It has a predicate proper $::\ expr \Rightarrow bool$, to restrict to terms one-one with those of the untyped $\lambda$-calculus modulo $\alpha$-conversion.

- And a predicate $\mathsf{abstr} ::\ (expr \Rightarrow expr) \Rightarrow bool$ identifying those functions which are sufficiently parametric to realize the body of a (non-exotic) $\lambda$-term. The function $\mathsf{LAM}\,.$ is just a **definition** for such a function.

- Example: a terms such as $\lambda V_1 . \lambda V_2 . V_1\ V_2$ is rendered in Hybrid as $\mathsf{LAM}\, v_1 . (\mathsf{LAM}\, v_2 . (v_1 \ \$\$\ v_2))$ and reduced by rewriting to the de Bruijn term $\mathsf{ABS}\ (\mathsf{ABS}\ (\mathsf{BND}\ 1\ \$\$\ \mathsf{BND}\ 0))$.

- Freeness and extensionality properties of hybrid expressions **proved** from the properties of the underlying de Bruijn representation:

$$[\![\ \mathsf{abstr}\ E;\ \mathsf{abstr}\ F\ ]\!] \Longrightarrow (\mathsf{LAM}\, v.\, E\ v = \mathsf{LAM}\, v.\, F\ v) = (E = F)$$

$$[\![\ \mathsf{abstr}\ E;\ \mathsf{abstr}\ F;\ \forall i.\, E\ (\mathsf{VAR}\ i) = F\ (\mathsf{VAR}\ i)\ ]\!] \Longrightarrow E = F$$

- Several induction principles over proper terms also proved. In summary:

- Object-level variables as meta-level bound variables.

- Object-level substitution as meta-level $\beta$-conversion.

- How do we encode judgments?

## Mini-ML in Hybrid

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

- Underlying form of $\mathbf{fix}\; x\,.\; \mathbf{fun}\, y.\, x \;@\; y$ is

$$\mathsf{CON}\; cFIX \;\$\$\; (\mathsf{LAM}\, x.\, \mathsf{CON}\; cABS \;\$\$\; \mathsf{LAM}\, y.\, (\mathsf{CON}\; cAPP \;\$\$\; x \;\$\$\; y)$$

- The constructors have distinct images. For example:

$$\mathbf{fun}\; x\,.\; E\; x \neq \mathbf{fix}\; x\,.\; E'\; x$$

- Every *binding* constructor is injective on abstractions. For example:

$$[\![\, \mathsf{abstr}\; E;\; \mathsf{abstr}\; E' \,]\!] \Longrightarrow (\mathbf{fix}\; x\,.\; E\; x = \mathbf{fix}\; x\,.\; E'\; x) = (E = E')$$

- Now it is immediate to encode evaluation as an *inductive* relation, where *abstraction* annotations as Isabelle HOL premises when dealing with binders are crucial to derive appropriate inversion principles:

$$\begin{array}{rcl} \mathit{Inductive}\ \_ \Downarrow \_ & :: & [\mathit{exp}, \mathit{exp}] \Rightarrow \mathit{bool} \\ [\![\, E_1 \Downarrow \mathbf{fun}\, x. E'\ x;\ \mathsf{abstr}\ E'; E_2 \Downarrow V_2;\ (E'\ V_2) \Downarrow V \,]\!] & \Longrightarrow & (E_1 \mathbin{@} E_2) \Downarrow V \\ [\![\, \mathsf{abstr}\ E;\ E\ (\mathbf{fix}\, x\,.\, E\ x) \Downarrow V \,]\!] & \Longrightarrow & \mathbf{fix}\, x\,.\, E\ x \Downarrow V \end{array}$$

- Yet typing is not inductive, as a clause such the following fails the positivity condition:

$$[\![\, \mathsf{abstr}\ E; \forall x.(x : T) \ \rightarrow\ (E\ x : T) \,]\!] \ \Longrightarrow\ (\mathbf{fix}\, x\,.\, E\ x) : T$$

- **Two-level reasoning** to combine HOAS, induction and hypothetical judgments so that OL contexts are implicit but consistent with Isabelle HOL.

# Road-map

- The big issue: a logical framework which combines HOAS, interactive theorem proving and (co)induction.

- Introducing and using Hybrid: case study Mini-ML.

- $\boxed{\textbf{Two-level reasoning with Hybrid}}$:

  1. Type preservation

  2. Compilation

## Two-Levels Reasoning in Hybrid

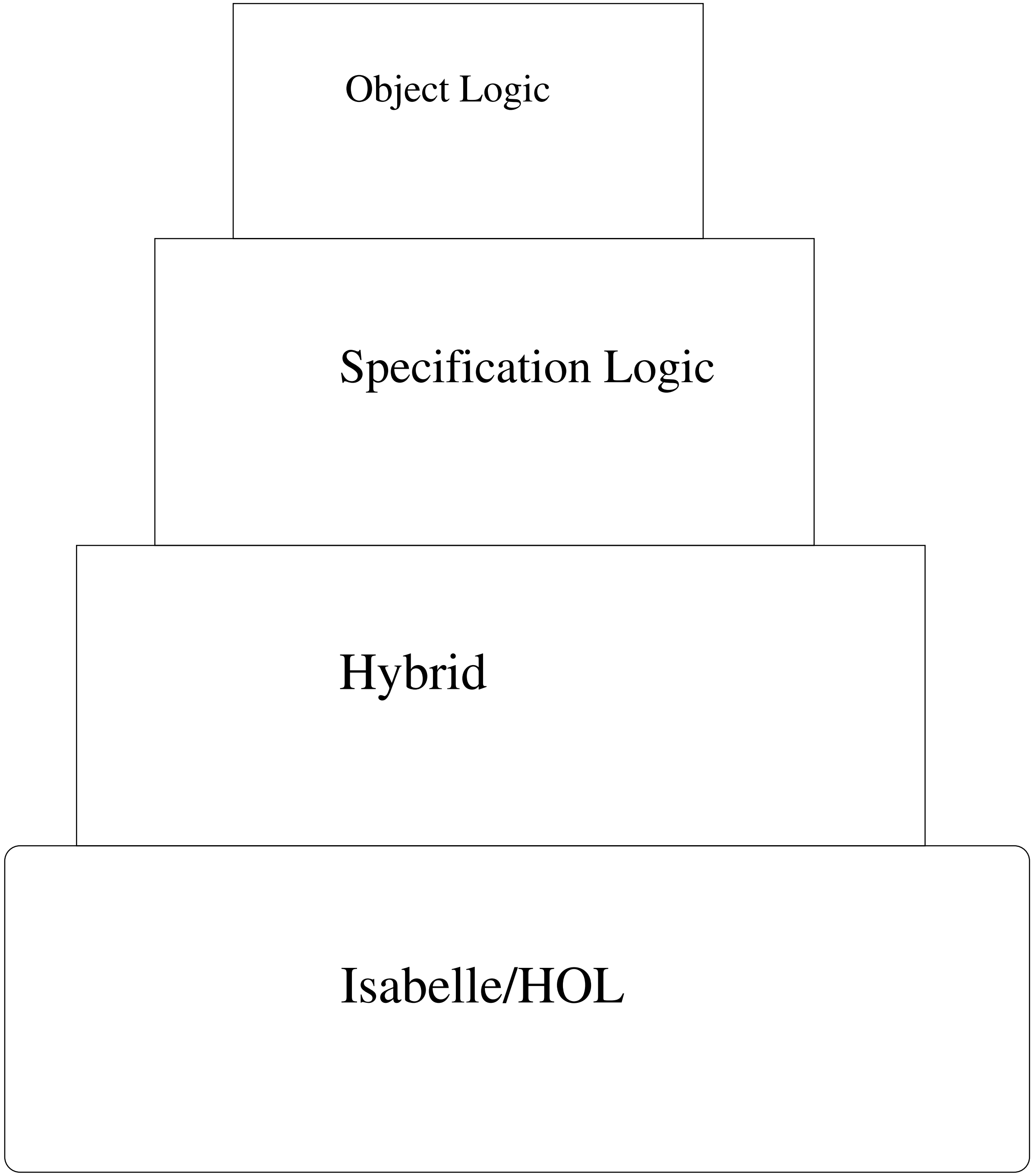


- Moral: HOAS without hypothetical judgments falls short. Idea: introduce an additional layer to mediate between hypothetical judgments and induction, the so-called two-levels architecture of $FO\lambda^{\Delta\mathbb{N}}$, adapted to Hybrid in analogy (but *definitionally)* with [Felty'02].

- A specification logic (a sequent calculus) where meta-reasoning is performed (as a inductive definition). This distinction solves the problem of negative occurrences, since hypothetical judgments need not be inductive.

- **Definitional** reflection [Halnass'91] simulated by inversion on inductive definitions aided by Hybrid frenees of constructors and extensionality properties at higher types; not assumed and in dire need of a semantics.

- $FO\lambda^{\Delta\mathbb{N}}$ number induction replaced (sometimes) by HOL's structural induction on stratifiable judgments (viz. evaluation).




Syntax: M @ N, fun x . E x
Semantics: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

Tactics/Simplifier
induction principles for
stratified jugments

## Two-Levels Reasoning in Hybrid 

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

- The mixing of meta-level (evaluation) and OL (typing) judgments makes proofs more easily mechanizable.

- This allows us to use co-induction which is still unaccounted for in a logic such as $FO\lambda^{\Delta\mathbb{N}}$. Better: enrich the sequent calculus with *guarded* induction (work in progress with Dale Miller).

- Fruitful interaction between (co)-induction principles, Isabelle HOL datatypes, classical reasoning and hypothetical judgments, hence more automation than in Coq.

A fragment of second-order hereditary Harrop formulae in a Isabelle HOL datatype:

$$
\begin{array}{rrcl}
\textit{Clauses} & D & ::= & A \mid D_1 \text{ and } D_2 \mid G \text{ imp } A \mid \text{tt} \mid \text{all}\, x.\, D \\
\textit{Goals} & G & ::= & A \mid G_1 \text{ and } G_2 \mid A \text{ imp } G \mid \text{tt} \mid \text{all}\, x.\, G \\
\textit{datatype} & oo & ::= & \text{tt} \mid \langle atm \rangle \mid oo \;\text{ and }\; oo \mid atm \;\text{ imp }\; oo \mid all\ (\text{prpr} \Rightarrow oo)
\end{array}
$$

$$
\begin{array}{rcl}
\textit{Inductive}\ \_ \rhd_{\_}\, \_ & :: & [nat, atm\ list, oo] \Rightarrow bool \\
 & \Longrightarrow & \Phi \rhd_n \text{tt} \\
[\![\, \Phi \rhd_n G_1; \Phi \rhd_n G_2 \,]\!] & \Longrightarrow & \Phi \rhd_{n+1} (G_1 \text{ and } G_2) \\
[\![\, \forall x.\, \Phi \rhd_n (G\ x) \,]\!] & \Longrightarrow & \Phi \rhd_{n+1} (\text{all}\, x.\, G\ x) \\
[\![\, \Phi, A \rhd_n G \,]\!] & \Longrightarrow & \Phi \rhd_{n+1} (A \text{ imp } G) \\
[\![\, A \in \Phi \,]\!] & \Longrightarrow & \Phi \rhd_n \langle A \rangle \\
[\![\, A \longleftarrow G; \Phi \rhd_n G \,]\!] & \Longrightarrow & \Phi \rhd_{n+1} \langle A \rangle
\end{array}
$$

- Atoms are provable by assumption or via *backchaining* over a set of Prolog-like rules, $A \longleftarrow G$ corresponds to an inductive definition of a set `prog` of type $[atm, oo] \Rightarrow bool$ which encode the properties of the object-logic in question

- Sequents are decorated with natural numbers which represent the *height* of a proof; this measure allows reasoning by complete induction.

- Develop a bit of meta-theory; $\Phi \rhd G$ iff $\exists n$ such that q $\Phi \rhd_n G$ and $\rhd\, G$ if $\cdot \rhd G$.

  1. Weakening for numerical bounds: $[\![\, \Phi \rhd_n G;\ n < m \,]\!] \Longrightarrow \Phi \rhd_m G$
  2. Context weakening: $[\![\, \Phi \rhd G;\ \Phi \subseteq \Phi' \,]\!] \Longrightarrow \Phi' \rhd G$
  3. (Atomic) cut: $[\![\, \Phi, A \rhd G;\ \Phi \rhd \langle A \rangle \,]\!] \Longrightarrow \Phi \rhd G$

## The Object Logic Level

- Non inductive judgments are encapsulated as Prolog-like clauses and executed in the SL:

$$\textit{datatype} \quad atm \quad ::= \quad exp : tp \mid (val\ env) \vdash exp' \leftrightarrow exp \mid val \Leftrightarrow exp$$

$$\begin{array}{rcl}
\textit{Inductive}\ _ \longleftarrow _ & :: & [atm, oo] \Rightarrow bool \\
& \Longrightarrow & (E_1\ @\ E_2) : T \longleftarrow \langle E_1 : (T'\ \mathsf{arrow}\ T)\rangle\ \mathsf{and}\ \langle E_2 : T'\rangle \\
[\![\ \mathsf{abstr}\ E\ ]\!] & \Longrightarrow & (\mathbf{fun}\, x.\, E\ x) : (T_1\ \mathsf{arrow}\ T_2) \longleftarrow \mathsf{all}\, x.\, (x : T_1)\ \mathsf{imp}\ \langle (E\ x) : T_2\rangle \\
[\![\ \mathsf{abstr}\ E\ ]\!] & \Longrightarrow & (\mathbf{fix}\ x\, .\ E\ x) : T \longleftarrow \mathsf{all}\, x.\, (x : T)\ \mathsf{imp}\ \langle (E\ x) : T\rangle
\end{array}$$

- Now we can address the meta-theory, starting, for example, with the subject reduction theorem, proven by HOL structural induction on evaluation, using only inversion principles on provability and OL typing

$$E \Downarrow V \Longrightarrow \forall T.\, (\triangleright\ E : T) \rightarrow (\triangleright\ V : T)$$

## Road-map

- The big issue: a logical framework which combines HOAS, interactive theorem proving and (co)induction.

- Introducing and using Hybrid: case study Mini-ML.

- Two-level reasoning with Hybrid.

1. $\boxed{\textbf{Type preservation}}$ (Board)

2. Compilation

# Formal Proof of Subject Reduction 1/2

Legend: `P >> V` for $P \Downarrow V$, `[] |-- < P : T >` for $\rhd \langle P : T \rangle$, `all, ->, &` for the SL connectives.

```
P >> V ==> ALL T. [] |-- < P : T > --> [] |-- < V : T >
(* by structural induction, app case *)

[| E1 >> fun x. E1' x; abstr E1'; E2 >> V2;
   (E1' V2) >> V;[] |-- < E1 @ E2 : T > |] ==> [] |-- < V : T >
(* inversion on atomic provability of @ *)

[|... E1 @ E2 : T <-- Body ; [] |-- Body |] ==> [] |-- < V : T >
(* inversion on typing *)

[|... [] |-- < E1 : T' arrow T > & < E2 : T' > |] ==> [] |-- < V : T >
```

```
(* by and elimination and ind. hyp. on E1 >> fun x. E1’ x *)

[|... [] |-- < fun x. E1’ x : ?T> |]  ==> [] |-- < V : T >
(* inversion on fun *)

[|... [] |-- all x. (x : T1 -> < E x : T2 > |] ==> [] |-- < V : T2 >
(* inverting and instantiating the SL quantifier x:= V2, cf. subst. lemma *)

[|... [] |-- (V2 : T1 -> < (E V2) : T2 > |] ==> [] |-- < V : T2 >
(* by ind. hyp on E2 >> V2 *)

[|...[] |-- (V2 : T1); [] |-- (V2 : T1 -> < (E V2) : T2 > |] ==> [] |-- < V

(* by inversion on implication and cut *)

[|... (E V2) >> V; [] |-- < (E V2) : T2 > |] ==> [] |-- < V : T2 >

(* the result follows by ind hyp *)
```

## Noteworthy

- Evaluation is encoded at the Hybrid level, typing at the OL level.

- The encoding of typing relies on **implicit** contexts, that is the mathematical statement $x_1 : \tau_1 \ldots x_n : \tau_n \vdash e : \tau$ is represented at the SL level as the **sequent** $[X_1 : \tau_1 \ldots X_n : \tau_n] \rhd \langle e : \tau \rangle$, where the $X_i : \tau_i$'s are SL atoms and $X_i$'s are Isabelle's logic variables.

- Structural rules (i.e. weakening and exchange) are proven **once and for all** at the SL level and do not need repeating for every judgment over open terms.

- The call to the substitution lemma is **avoided** altogether by the combination of hypothetical judgments and (atomic) cut.

- This is extremely elegant and effective in formal development, as we show next.

## Road-map

- The big issue: a logical framework which combines HOAS, interactive theorem proving and (co)induction.

- Introducing and using Hybrid: case study Mini-ML.

- Two-level reasoning with Hybrid.

1. Type preservation

2. **Compilation**

## Compiler Verification



- Based on operational (rather than denotational) semantics, so that meaning of both high and low level languages are expressed uniformly.

- Both the operational semantics and the program translations are represented using deductive systems [Despeyroux'86], mechanized in [Pfenning&Hannah'92].

- Stepwise approach: form big-step to closure evaluation to stack machine.

- Add a co-inductive twist due to [Milner & Tofte'91] in the treatment of fix points which evaluate to a "circular" closure, where, intuitively, the free variable points to the closure itself.

- Source language a strict lambda calculus augmented with a fix point operator compiled into a simple calculus explicit substitutions, where deBrujin indexes are represented by appropriate shift:

$$\begin{array}{rccl} \textit{dB Terms} & F & ::= & 1 \mid F\uparrow \mid \mathbf{fun}'\, F \mid F\ @'\ F' \mid \mathbf{fix}'\, F \\ \textit{Environments} & \eta & ::= & \cdot \mid \eta;W \mid \eta + F \\ \textit{Values} & W & ::= & \{\eta, F\} \end{array}$$

- Standard operational semantics $\eta \vdash F \hookrightarrow W$) uses environments to associate variables to values and closures for terms with free variables.

- Translation ($\eta \vdash F \leftrightarrow e$) to the dB language uses parametric and hypothetical judgments for the binding construct.

$$\dfrac{\dfrac{}{w \Leftrightarrow x}\,u \quad \vdots \quad \eta; w \vdash F \leftrightarrow e}{\eta \vdash \mathbf{fun}'\ F \leftrightarrow \mathbf{fun}\ x\,.\ e}\,\mathbf{tr\_lam}^{w,x,u} \qquad \dfrac{\dfrac{}{\eta \vdash f \leftrightarrow x}\,u \quad \vdots \quad \eta + f \vdash F \leftrightarrow e}{\eta \vdash \mathbf{fix}'\ F \leftrightarrow \mathbf{fix}\ x\,.\ e}\,\mathbf{tr\_fix}^{f,x,u}$$

$$\dfrac{\eta \vdash F_1 \leftrightarrow e_1 \qquad \eta \vdash F_2 \leftrightarrow e_2}{\eta \vdash F_1\ @'\ F_2 \leftrightarrow e_1 @\ e_2}\,\mathbf{tr\_app}$$

$$\dfrac{W \Leftrightarrow e}{\eta; W \vdash 1 \leftrightarrow e}\,\mathbf{tr\_1} \qquad \dfrac{\eta \vdash F \leftrightarrow e}{\eta; W \vdash F\uparrow \leftrightarrow e}\,\mathbf{tr\_}\uparrow$$

$$\dfrac{\eta \vdash \mathbf{fun}'\ F \leftrightarrow \mathbf{fun}\ x\,.\ e}{\{\eta, \mathbf{fun}'\ F\} \Leftrightarrow \mathbf{fun}\ x\,.\ e}\,\mathbf{vtr\_lam}$$

# Compiler Verification



$$\begin{tikzcd}
e \arrow[rr, "e \hookrightarrow v"] \arrow[d, "\eta \vdash F \leftrightarrow e"'] & & v \\
F \arrow[rr, "\eta \vdash F \hookrightarrow W"'] \arrow[urr, phantom, "\equiv"] & & W \arrow[u, "W \Leftrightarrow v"']
\end{tikzcd}$$

- Soundness of compilation:
  $E \Downarrow V \Longrightarrow \forall n\ \eta\ F. (\cdot \rhd_n \eta \vdash F \leftrightarrow e) \to \exists W. \eta \vdash F \hookrightarrow W \wedge (\rhd\ W \Leftrightarrow V)$, by nested induction, outermost on the structure of $E \Downarrow V$ innermost by complete induction on $n$, i.e. the height of the SL's proof of $\eta \vdash F \leftrightarrow e$.

- Completeness of compilation:
  $\eta \vdash F \hookrightarrow W \Longrightarrow \forall n\ E. (\cdot \rhd_n \eta \vdash F \leftrightarrow e) \to \exists V. E \Downarrow V \wedge \rhd\ W \Leftrightarrow V$

- Proofs are from first principles, without any technical lemma.

- Unrestricted fix-points requires environments to possibly contain unevaluated expressions and evaluation of variables not just a look-up, but requires the evaluation of the body of the fix point.

- With the SML restriction that body of a fix point is always a lambda. fix point can evaluate to non-well founded closures:

$$[\![\, cl = \{(\eta; cl), \mathbf{fun}'\, F\}^{\infty} \,]\!] \implies \eta \vdash \mathbf{fix}'\, (\mathbf{fun}'\, F) \hookrightarrow cl$$

- This brings about co-inductive reasoning, for example in the proof of subject reduction for closures, where typing of closures and type consistency of value and type environments are *mutually co-inductive* definitions [Milner &Tofte'91].

## Conclusions

- A logical framework giving a definitional account of HOAS, compatible with tactic based theorem proving 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.
- Multi-level architecture allows to simulate $FO\lambda^{\Delta I\!N}$ and Twelf in the interactive and well-known setting of higher-order logic.
- Results about the SL, (such as cut elimination) are proven once and for all. Different logics (for example, linear well suited to imperative languages) can be selected, without changing infrastructure.

## Current and Future Work

- Current work:

  1. Miller & McDowell's explicit eigenvariable encoding for induction over open terms required to allow induction over generic judgments (i.e. proof of type uniqueness).

  2. At the infrastructure 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 co-inductive judgments at `prog` level.

## More Info

- `http://www.mcs.le.ac.uk/mechsem/Hybrid`
  - TPHOLs '02 paper with the theory of Hybrid, scripts for the current infrastructure and example suite: quantified propositional logic, higher-order $\pi$-calculus.

- `http://www.mcs.le.ac.uk/mechsem/Hybrid/Howe`
  - LFM'02 paper and code for Howe's congruence proof.

- `http://www.mcs.le.ac.uk/mechsem/Hybrid/2level/Compilation`
  - Submitted paper with the 2-level architecture and the compilation example.