Multi-Level Meta-Reasoning with Higher-Order Abstract Syntax

A. Momigliano (A.Momigliano@mcs.le.ac.uk) &
S. J. Ambler (S.Ambler@mcs.le.ac.uk)

Department of Mathematics and Computer Science, University of Leicester,
Leicester, LE1 7RH, U.K.

Abstract. Combining Higher Order Abstract Syntax (HOAS) and (co)-induction is well known to be problematic. In previous work [2] we have described the implementation of a tool called Hybrid, within Isabelle HOL, which allows object logics to be represented using HOAS, and reasoned about using tactical theorem proving and principles of (co)induction. Moreover, it is definitional, which guarantees consistency within a classical type theory. In this paper we describe how to use it in a multi-level reasoning fashion, similar in spirit to other meta-logics such $FO\lambda^{\Delta IN}$ and *Twelf*. By explicitly referencing provability, we solve the problem of reasoning by (co)induction in presence of non-stratifiable hypothetical judgments, which allow very elegant and succinct specifications. We demonstrate the method by formally verifying the correctness of a compiler for (a fragment) of Mini-ML, following [18]. To further exhibit the flexibility of our system, we modify the target language with a notion of non-well-founded closure, inspired by Milner & Tofte [27] and formally verify via coinduction a subject reduction theorem for this modified language.

1 Introduction

Higher Order Abstract Syntax (HOAS) is a representation technique, dating back to Church [4], where binding constructs in an object logic are encoded within the function space provided by a meta-language based on a λ-calculus. This specification is generic enough that sometimes HOAS has been identified [20] merely with a mechanism to delegate to the meta-language α-conversion and capture-avoiding substitutions. While this indeed important, it is by no means the whole story. As made clear by the LF [19] and λProlog experience [30], further benefits come when for an object logic type $\mathbf{i}$, this function space is taken to be $\mathbf{i} \Rightarrow \mathbf{i}$, or a subset of it. In this way, object-logic substitution can be rendered as meta-level β-conversion. A meta-language offering this facility is indeed a step up, but there is still room for improvement. Experiments such as the one reported in [29] suggests that the full benefits of HOAS can be enjoyed only when it is paired with hypothetical and parametric judgments. Indeed, judgments requiring to descend into a binding construct may contain *negative* occurrence of the predicate being defined. Set-theoretically, in fact, those judgments do not yield monotone set-theoretic operators and cannot be constructed as a least fix point [32]. Type-theoretically, they infringe the *strict positivity* requirement [31, 14] and break strong normalization. A standard example is the full HOAS encoding of type-inference which can do without any additional references to concrete data-structures and functions, (list of variable/type pairs, look-up). As much as it is appealing this begs the question on how we are going to reason on such encodings, in particular are there induction and case analysis principles available?

A solution that has emerged in the last five years is that *specification* and (inductive) *meta-reasoning* should be dealt with in a single system but at different *levels*. One such meta-logic is Miller & McDowell $FO\lambda^{\Delta IN}$ [23]; it is is based on higher-order intuitionistic logic augmented with definitional reflection [15] — to provide support for case analysis — and induction on natural numbers. Consistency is ensured by cut-elimination [22]. Inside this meta-language they develop a *specification logic* (SL) which in turn it is used to specify the *object-logic* (OL) under study. This partition solves the problem of meta-reasoning in the presence of negative occurrences, since hypothetical judgments are now encapsulated within the OL and therefore not required to be inductive. The price to pay is this additional layer where we explicitly reference provability and the necessity therefore of a sort of meta-interpreter (the SL logic) to access it.

Very recently, Felty [10] has suggested that, rather than implementing an interactive theorem prover for $FO\lambda^{\Delta IN}$ from scratch, the latter can be simulated within an existing system (Coq in that case); in particular, definitional reflection is mimicked by the elimination rules of inductive types. Nevertheless, this is not quite enough, as reasoning by inversion crucially depend on simplifying in the presence of constructors. Since these are by necessity non-inductive, Felty recurs to the assumption of a set of axioms stating the freeness and extensionality properties of constructors in the given signature. Under those conditions the author shows, in the example of the formalization of subject reduction for Mini-ML, how it is possible to replicate, in an well-understood and interactive setting, the style of proofs typical of $FO\lambda^{\Delta I\!N}$; namely the result is proven without "technical" lemmas foreign to the mathematics of the problem.

In previous work [2] we have described the implementation of a higher-order meta-language, called Hybrid, within Isabelle HOL, which provides a form of HOAS for the user to represent object logics. The user level is separated from the infrastructure, in which HOAS is implemented *definitionally* via a de Bruijn style encoding.

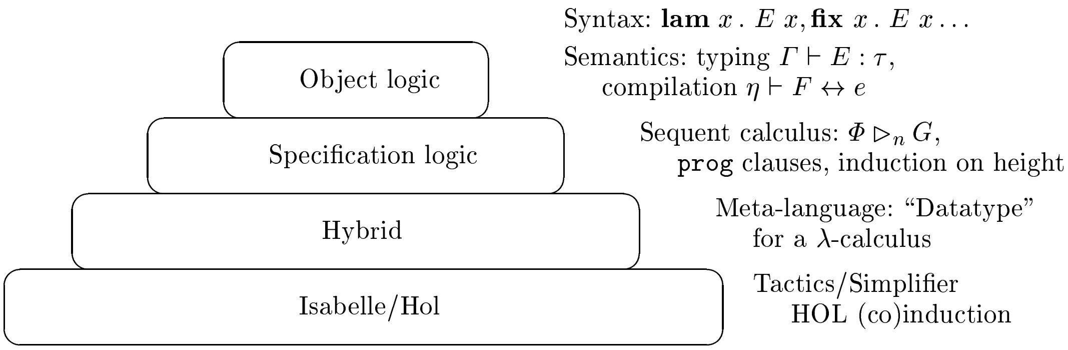

Fig. 1. 2-Level Architecture

In this paper we adopt most of Felty's architecture, with the notable difference of using Hybrid rather than Coq as the basic metalanguage. A graphical depiction of the proposed architecture is in Figure 1. Moreover, we take a further departure in design: we suggest to push at the OL only those judgments which would not be inductive, and to leave the rest at the Isabelle HOL level. We claim that this framework has several advantages:

– The system is more trustworthy: freeness of constructors and more importantly extensionality properties at higher types are not assumed, but proven via the related properties of the infrastructure, see Subsection 3.1.
– The mixing of meta-level and OL judgments makes proofs more easily mechanizable and allows us to use co-induction which is still unaccounted for in a logic such as $FO\lambda^{\Delta I\!N}$.
– More in general, there is a fruitful interaction between (co)-induction principles, Isabelle HOL datatypes, classical reasoning and hypothetical judgments, which tends to yield more automation than in a system such as Coq.

Our approach is also comparable with *Twelf* [35] (see Section 6), but is has a low mathematical overhead, as it simply consists of a package on top of Isabelle HOL. In a sense, we could look at Hybrid as a way to "compile" HOAS meta-proofs, such as those of Twelf's meta-logic $\mathcal{M}_\omega$ [38], into the well-understood setting of higher-order logic.

We demonstrate the method by formally verifying the correctness of (part of) a compiler for a fragment of Mini-ML, following [18]. To further exhibit the flexibility of our system, we modify the target language with a notion of non-well-founded closure, inspired by Milner & Tofte's paper [27] and formally verify, via coinduction, a subject reduction theorem for this new language.

The paper is organized as follows: in the next Section 2 we introduce at the informal level the syntax and semantics of our case study. Section 3 recalls some basics notion of Hybrid and its syntax representing techniques. In Section 4 we introduce the 2-level architecture, while Section 5 is the heart of the paper, detailing the formal verification of compiler correctness. We conclude with a few words on related and future work (Section 6 and 7). The Appendix completes the listing of all rekevant jugments and selected proof scrpits.

Notation 1. *We use a pretty-printed version of Isabelle HOL concrete syntax; a rule (a sequent) with conclusion C and premises $H_1 \dots H_n$ will be represented as $[\![H_1; \dots; H_n]\!] \Longrightarrow C$. A Isabelle HOL type declaration has the form $s\ :: [\, t_1, \dots t_n \,] \Rightarrow t$. Isabelle HOL connectives are represented via the usual logical notation. Free variables (upper-case) are implicitly universally quantified. The sign $\equiv$ (Isabelle meta-equality) is used for* equality by definition, *$\bigwedge$ for Isabelle universal meta-quantification. The keyword **MC-Theorem** denotes a machine-checked theorem, while* (Co)Inductive *introduces a (co)inductive relation in Isabelle HOL. We have tried to use the same notation for mathematical and formalized judgments. To facilitate the comparison with the Hannan & Pfenning's approach, we have used the same nomenclature and convention as [33].*

2 The Case Study

Compiler verification can be a daunting task, but it is a good candidate for mechanization, as it was recognized as early as [28]. Restricting to functional languages (see [1] for issue related to compilation of Java-like language), some first attempts were based on denotational semantics [5, 21]. Using instead operational semantics has the advantage that the meaning of both high and low level languages can be expressed in a common setting. In particular, both the operational semantics and the program translations can be represented using deductive systems. This approach was started in [7][1]. Other papers, such as [17, 16] have explored aspects of higher order encodings of abstract machines and compilation and [18] contains the first attempt to do formal verification of compiler correctness in LF. Only recently the notion of *regular world* introduced in [38, 39] provides a satisfying foundation for the theory of *mode, termination and coverage* checking [34, 37], which has been developed to justify using LF for meta-reasoning.

We follow the stepwise approach to compilation in [18]; however, for reason of space, we limit ourselves to the verification of the correspondence between the big-step semantics of a higher-order functional language and the evaluation to closure of its translation into a target language of first-order expressions.

For interest, though, we add a coinductive twist due to Milner & Tofte [27] in the treatment of fix points, which is analogous to the semantics of *letrec* in the original formulation of Mini-ML [6].

The language we utilize here is a strict lambda calculus augmented with a fix point operator, although it could be easily generalized to the rendition of Mini-ML in [33]. This fragment is sufficient to illustrate the main ideas without cluttering the presentation with too many details.

The types and terms of the source language are given respectively by:

$$\begin{array}{l} \textit{Types}\ \tau ::= \mathbf{i} \mid \tau_1\ \mathsf{arrow}\ \tau_2 \\ \textit{Terms}\ e ::= x \mid \mathbf{lam}\ x\,.\,e \mid e\ @\ e' \mid \mathbf{fix}\ x\,.\,e \qquad \dagger \end{array}$$

The rules for call-by-value operational semantics ($e \Downarrow v$) and type inference ($\Gamma \vdash e : \tau$) are standard and are defined in Figure 2.

The usual subject reduction for this source language holds.

Theorem 1 (Subject Reduction). *If $e \Downarrow v$ and $\cdot \vdash e : \tau$, then $\cdot \vdash v : \tau$.*

Proof. By structural induction on evaluation and inversion on typing.

2.1 Compilation

We start the compilation process by translating the higher-order syntax into a first-order one. Terms are compiled into a simple calculus of explicit substitutions, where deBrujin indexes are represented by appropriate shifting on the 1 numeral:

$$\textit{dB Terms}\ F ::= 1 \mid F\uparrow \mid \mathbf{lam}'\ F \mid F\ @'\ F' \mid \mathbf{fix}'\ F$$

[1] A partial verification via first-order encoding can be found in [3].

$$\frac{e_1 \Downarrow \mathbf{lam}\ x\ .\ e_1' \qquad e_2 \Downarrow v_2 \qquad [v_2/x]e_1' \Downarrow v}{e_1 \ @\ e_2 \Downarrow v}\ \mathbf{ev\_app}$$

$$\frac{}{\mathbf{lam}\ x\ .\ e \Downarrow \mathbf{lam}\ x\ .\ e}\ \mathbf{ev\_lam} \qquad \frac{[\mathbf{fix}\ x\ .\ e/x]e \Downarrow v}{\mathbf{fix}\ x\ .\ e \Downarrow v}\ \mathbf{ev\_fix}$$

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

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

Fig. 2. Big step semantics and typing rules for the source language

Then we introduce environments and closures:

$$\begin{array}{c} \mathit{Environments} \quad \eta ::= \cdot \mid \eta; W \mid \eta + F \\ \mathit{Values}\ W ::= \{\eta, F\} \end{array}$$

In this setting the only possible values are *closures*, yet the presence of fix-points requires environments to possibly contain unevaluated expressions, via the environment constructor '+'. We will see in Subsection 2.2 how this can be refined using a notion of *non well-founded* closure.

The operational semantics of this language — judgment $\eta \vdash F \hookrightarrow W$ in Figure 3 — uses environments to represent mappings of free variables to values and closures for terms whose free variables have values in the environment. We remark that due to the presence of unrestricted fix points, the rule $\mathbf{fev\_1}^+$ in not just a look-up, but requires the evaluation of the body of the fix point.

$$\frac{}{\eta; W \vdash 1 \hookrightarrow W}\ \mathbf{fev\_1} \qquad \frac{\eta \vdash F \hookrightarrow W}{\eta; W' \vdash F\!\uparrow\hookrightarrow W}\ \mathbf{fev\_}\!\uparrow$$

$$\frac{\eta \vdash F \hookrightarrow W}{\eta + F \vdash 1 \hookrightarrow W}\ \mathbf{fev\_1}^+ \qquad \frac{\eta \vdash F \hookrightarrow W}{\eta + F' \vdash F\!\uparrow\hookrightarrow W}\ \mathbf{fev\_}\!\uparrow^+$$

$$\frac{}{\eta \vdash \mathbf{lam}'\ F \hookrightarrow \{\eta, \mathbf{lam}'\ F\}}\ \mathbf{fev\_lam}' \qquad \frac{\eta + \mathbf{fix}'\ \ F \vdash F \hookrightarrow W}{\eta \vdash \mathbf{fix}'\ \ F \hookrightarrow W}\ \mathbf{fev\_fix}'$$

$$\frac{\eta \vdash F_1 \hookrightarrow \{\eta', \mathbf{lam}'\ F_1'\} \qquad \eta \vdash F_2 \hookrightarrow W_2 \qquad \eta'; W_2 \vdash F_1' \hookrightarrow W}{\eta \vdash F_1\ @'\ F_2 \hookrightarrow W}\ \mathbf{fev\_@}'$$

Fig. 3. Evaluation to closures

The judgment $\eta \vdash F \leftrightarrow e$ in Figure 4 elegantly accomplishes the translation to the dB language using parametric and hypothetical judgments for the binding constructs: for the fix point, we assume we have extended the environment with a new *expression* parameter f; in the function case, the parameter w ranges over *values* and the judgment is mutually recursive with value translation, $W \Leftrightarrow v$. As remarked in [33], this translation is total, but can be non-deterministic, when η and e are given and F is computed.

The verification of compiler correctness includes

$$
\frac{\begin{array}{c}\overline{w \Leftrightarrow x}\,u \\ \vdots \\ \eta; w \vdash F \leftrightarrow e\end{array}}{\eta \vdash \mathbf{lam}'\ F \leftrightarrow \mathbf{lam}\ x\ .\ e}\ \mathbf{tr\_lam}^{w,x,u}
\qquad
\frac{\begin{array}{c}\overline{\eta \vdash f \leftrightarrow x}\,u \\ \vdots \\ \eta + f \vdash F \leftrightarrow e\end{array}}{\eta \vdash \mathbf{fix}'\ F \leftrightarrow \mathbf{fix}\ x\ .\ e}\ \mathbf{tr\_fix}^{f,x,u}
$$

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

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

$$
\frac{\eta \vdash F \leftrightarrow e}{\eta + F \vdash 1 \leftrightarrow e}\ \mathbf{tr\_1}^{+}
\qquad
\frac{\eta \vdash F \leftrightarrow e}{\eta + F' \vdash F\!\uparrow \leftrightarrow e}\ \mathbf{tr\_}\!\uparrow^{+}
$$

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

Fig. 4. Translation to modified dB terms and values

1. Soundness of compilation: for any $e \Downarrow v$ and $\eta \vdash F \leftrightarrow e$, there is W such that $\eta \vdash F \hookrightarrow W$ and $W \Leftrightarrow v$
2. Completeness of compilation: if $\eta \vdash F \hookrightarrow W$ and $\eta \vdash F \leftrightarrow e$, then there is v such that $e \Downarrow v$ and $W \Leftrightarrow v$

This can be graphically represented as:

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

We discuss the proof (Theorem 6) once we provide the mechanization in Section 5.

2.2 Compilation via non-well founded closures

In order to illustrate the flexibility of our approach we now depart from [33] and draw inspiration from [27] to give a different and simplified treatment of fix points. First, we take up the SML-like restriction that the body of a fix point is always a lambda. Then the idea is simply to allow a fix point to evaluate to a "circular" closure, where, intuitively, the free variable points to the closure itself. In our setting this means swapping the following rule for $\mathbf{fev\_fix}'$ and dropping rules $\mathbf{fev\_1}^{+}$ and $\mathbf{fev\_}\!\uparrow^{+}$. Note that the analogous rules in the translation judgment would also be dropped.

$$
\frac{cl = \{(\eta; cl), \mathbf{lam}'\ F\}}{\eta \vdash \mathbf{fix}'\ (\mathbf{lam}'\ F) \hookrightarrow cl}\ \mathbf{fev\_fix}'\ {}^{*}
$$

This amounts to a *non-well founded* notion of closure. Other versions are possible, viz. recursive environments, see [7] for a discussion.

To exemplify the style of co-inductive reasoning entailed, we adapt the proof of subject reduction for closures in [27]. First we introduce typing for dB terms $\eta \vdash F : \tau$ in Figure 5.

Typing of closures (Figure 6) can be seen as taking the greatest fix point of the following rules, where we refer the reader to *ibid.* for motivation and examples.

$$\frac{}{\Delta;\tau \vdash 1 : \tau}\ \mathbf{ftp\_1} \qquad \frac{\Delta \vdash F : \tau}{\Delta;\tau' \vdash F\!\uparrow : \tau}\ \mathbf{ftp\_\!\uparrow}$$

$$\frac{\Delta;\tau \vdash F : \tau'}{\Delta \vdash \mathbf{lam}'\ F : \tau\ \mathsf{arrow}\ \tau'}\ \mathbf{ftp\_lam}' \qquad \frac{\Delta;\tau \vdash F : \tau}{\Delta \vdash \mathbf{fix}'\ F : \tau}\ \mathbf{ftp\_fix}'$$

$$\frac{\Delta \vdash F_1 : \tau'\ \mathsf{arrow}\ \tau \qquad \Delta \vdash F_2 : \tau'}{\Delta \vdash F_1\ @'\ F_2 : \tau}\ \mathbf{ftp\_@}'$$

Fig. 5. Type inference for dB

$$\frac{\Delta \vdash \mathbf{lam}'\ F : \tau \qquad \eta :' \Delta}{\{\eta, \mathbf{lam}'\ F\} : \tau}\ \mathbf{tpc} \qquad \frac{}{\cdot :' \cdot}\ \mathbf{etp\cdot} \qquad \frac{W : \tau \qquad \eta :' \Delta}{(\eta; W) :' (\Delta;\tau)}\ \mathbf{etp;}$$

Fig. 6. Typing for closures

We note the "circularity" of rule **tpc** and **etp;**, where the former requires well-typedness of value environment w.r.t. type environment, while the latter is just a recursion on the former.

Theorem 2 (Subject reduction for closures [27]). *Let $\eta :' \Delta$. If $\eta \vdash F \hookrightarrow W$ and $\Delta \vdash F : \tau$, then $W : \tau$.*

Proof. By structural induction on $\eta \vdash F \hookrightarrow W$: the only interesting case is the fix point one. By inversion $\Delta, \tau_1\ \mathsf{arrow}\ \tau_2, \tau_1 \vdash F : \tau_2$. To show $cl : \tau_1\ \mathsf{arrow}\ \tau_2$ by coinduction, we simply take $\{cl, \tau_1\ \mathsf{arrow}\ \tau_2\}$ as the dense set and set $\Delta' = \Delta, \tau_1\ \mathsf{arrow}\ \tau_2$ and $\eta' = \eta; cl$. Then $\Delta' \vdash \mathbf{lam}'\ F : \tau_1\ \mathsf{arrow}\ \tau_2$. Finally, $\eta' :' \Delta'$ and density holds.

We now move to a high-level description of the Hybrid system.

3 Hybrid Infrastructure

We recall that the theory Hybrid [2] provides support for a deep embedding of higher order abstract syntax within Isabelle HOL. In particular, it provides a model of the the untyped λ-calculus with constants. Let *con* denote a suitable type of constants. The model comprises a type *expr* together with functions

$$\begin{aligned} \mathsf{CON} &:: con \Rightarrow expr \\ \mathsf{VAR} &:: nat \Rightarrow expr \\ \$\$ &:: expr \Rightarrow expr \Rightarrow expr \\ \mathsf{lambda} &:: (expr \Rightarrow expr) \Rightarrow expr \end{aligned}$$

and two predicates $\mathsf{proper} :: expr \Rightarrow bool$ and $\mathsf{abstr} :: (expr \Rightarrow expr) \Rightarrow bool$. The elements of *expr* which satisfy proper are in one-one correspondence with the terms of the untyped λ-calculus modulo α-equivalence. The function CON is the inclusion of constants into terms, VAR is the enumeration of an infinite supply of free variables, and $$ is application. The function lambda is declared as a binder and we write $\mathsf{LAM}\, v.\, e$ for $\mathsf{lambda}\ (\lambda\, v.\, e)$. Note that Hybrid draws a sharp distinction between free variables, which are expressions of the form $\mathsf{VAR}\ i$, and bound variables which correspond to (bound) meta-variables (as in $\mathsf{LAM}\, v_1.\, v_1$ for instance).

For this data to faithfully represent the syntax of the un-typed λ-calculus, it must be that CON, VAR, $$ are injective on proper expressions, and furthermore, lambda is injective on some suitable subset of $expr \Rightarrow expr$. This cannot the whole of $expr \Rightarrow expr$ for cardinality reasons. In fact, we need only a small fragment of the set. The predicate abstr identifies those functions which are sufficiently parametric to be realized as the body of a λ-term, and lambda is injective on these.

There is a strong tradition in the HOL community of making extensions *by definition* wherever possible, that is, showing that additional constructs may be defined using those already in the logic rather than postulating them. This ensures the consistency of the logic relative to a small axiomatic core. Hybrid is implemented in a definitional style using a translation into de Bruijn notation. The type *expr* is defined by the grammar

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

The translation of terms is best explained by example. Let $T_O = \Lambda V_1.\, \Lambda V_2.\, V_1\ V_3$ be an expression in the concrete syntax of the λ-calculus[2]. This is rendered in Hybrid as $T_H = \mathsf{LAM}\, v_1.\, (\mathsf{LAM}\, v_2.\, (v_1\ \$\$\ \mathsf{VAR}\ 3))$ — note again the difference between the treatment of free variables and of bound variables. Expanding the binder, this expression is by definition

$$\mathsf{lambda}\ (\lambda\, v_1.\, (\mathsf{lambda}\ (\lambda\, v_2.\, (v_1\ \$\$\ \mathsf{VAR}\ 3))))$$

where λv_i is meta-abstraction. The function $\mathsf{lambda} :: (expr \Rightarrow expr) \Rightarrow expr$ is defined so as to map any function satisfying abstr to a corresponding proper de Bruijn expression. The expression T_H is reduced by higher order rewriting to the de Bruijn term $\mathsf{ABS}\ (\mathsf{ABS}\ (\mathsf{BND}\ 1\ \$\$\ \mathsf{VAR}\ 3))$.

Given these definitions, the essential properties of hybrid expressions can be proved as theorems from the properties of the underlying de Bruijn representation: for instance, the injectivity of lambda

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

and extensionality

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

There are several principles of induction over proper expressions which are also proved.

3.1 Coding the Syntax of OL System in Hybrid

We begin by showing how to represent (a fragment of) Mini-ML in Hybrid: in order to render the syntax † in HOAS format using Hybrid we need constants for abstraction and application, say *cAPP*, *cABS* and *cFIX*. Recall that in the meta-language, application is denoted by infix \$\$, and abstraction by LAM. Then † would correspond to the following grammar:

$$e ::= v \mid cABS\ \$\$\ (\mathsf{LAM}\, v.\, e\, v) \mid cAPP\ \$\$\ e_1\ \$\$\ e_2 \mid cFIX\ \$\$\ (\mathsf{LAM}\, v.\, e\, v)$$

Thus, we declare these constants to belong to *con* and then make the following definitions:

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

and similarly for the fix point, where lam (resp. fix) is indeed an Isabelle HOL binder. For example, the "real" underlying form of $\mathbf{fix}\ x\ .\ \mathbf{lam}\ y\ .\ x\ \mathbf{@}\ y$ is

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

It is now possible to *prove* the freeness properties of constructors.

MC-Theorem 1.

– *The constructors have distinct images. For example:*

$$\mathbf{lam}\ x\ .\ E\ x \neq \mathbf{fix}\ x\ .\ E'\ x$$

[2] We use a capital Λ and capital V to avoid confusion with meta-variables v and meta-abstraction λ in Isabelle HOL.

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

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

Proof. By Isabelle HOL's simplification, using property INJ in the binding cases.

Although dB expressions are strictly first-order, we still need to encode them as Hybrid expressions. In fact, we will use for compilation to dB expressions a judgment which is parametric in (higher-order) terms, dB expression and values. Therefore they all have to be interpreted via the SL universal quantification and consequently need to be synonymous to Hybrid (proper) terms, to make the universal quantification consistent[3]. The encoding is trivial and the details left to the on-line documentation.

The informal definition of environments and closure is by mutual recursion. Since our aim here is also to show how Hybrid expressions can coexist with regular Isabelle HOL ones, we will use Isabelle HOL mechanism for mutually inductive datatypes. This brings about the declaration of a datatype of polymorphic environments, intended to be instantiated with a value. Environments can be now mutually recursive with closures, where the type synonymous *exp′* is retained for forward compatibility with Subsection 5.1:

$$\begin{aligned} \mathit{datatype}\ (\alpha\ \mathit{env}) &::= \cdot \mid (\alpha\ \mathit{env})\ ;\ \alpha \mid (\alpha\ \mathit{env}) + \mathit{exp}' \\ \mathit{and}\quad \alpha\ \mathit{clos} &::= \mathsf{mk\_clo}\ (\alpha\ \mathit{env})\ \mathit{exp}' \end{aligned}$$

Then we introduce in *con* a constructor, say $cCLO(\mathit{val}\ \mathit{clos})$, which encapsulates a Isabelle HOL closure. Finally, we can *define* a Hybrid closure as a constant of type $[\,(\mathit{val}\ \mathit{env}), \mathit{exp}'\,] \Rightarrow \mathit{val}$, *defined* as

$$\{\eta, F\} == \mathsf{CON}\ (cCLO\ (\mathsf{mk\_clo}\ \eta\, F))$$

Thanks to this rather cumbersome encoding, we can establish freeness properties of closures as in Theorem 1.

4 2-level Architecture

In previous work [2, 29], we chose to work in a single level, implementing every judgment as a (co)inductive definition in Isabelle HOL, but exploiting the form of HOAS that our package supports. While the tool seemed successful in dealing with predicates over *closed* terms, say evaluation, we had to resort to a more traditional encoding, i.e. via explicit environments, with respect to judgments involving open ones such as typing. As we have mentioned earlier, a two-level approach solves this dilemma.

4.1 Encoding the Specification Logic

We introduce our specification logic, namely a fragment of second-order hereditary Harrop formulae, which is sufficient for the encoding of our case-study.

$$\begin{aligned} \mathit{Clauses}\ D &::= A \mid D_1\ \mathsf{and}\ D_2 \mid G\ \mathsf{imp}\ A \mid \mathsf{tt} \mid \mathsf{all}\, x.\, D \\ \mathit{Goals}\ G &::= A \mid G_1\ \mathsf{and}\ G_2 \mid A\ \mathsf{imp}\ G \mid \mathsf{tt} \mid \mathsf{all}\, x.\, G \end{aligned}$$

This syntax translates immediately in a Isabelle HOL datatype:

$$\mathit{datatype}\ \mathit{oo} ::= \mathsf{tt} \mid \langle \mathit{atm} \rangle \mid \mathit{oo}\ \ \mathsf{and}\ \ \mathit{oo} \mid \mathit{atm}\ \ \mathsf{imp}\ \ \mathit{oo} \mid \mathit{all}\ (\mathsf{prpr} \Rightarrow \mathit{oo})$$

where $\langle \_ \rangle$ coerces atoms into propositions. The universal quantifier is intended to range over all proper Hybrid terms. In analogy with logic programming, it will be left implicit in clauses.

This logic is so simple that its proof-system can be modeled with a logic programmer interpreter; in fact, for such a logic, *uniform provability* [26] (of a sequent-calculus) is complete. We give such a definition of provability in Figure 7.

Note the following:

[3] This issue arises from the fact that Hybrid, in its current formulation, provides only what amounts to an *untyped* meta-language. This is being addressed in the next version, where those sets will be Hybrid datatypes.

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

Fig. 7. Sequent calculus

- Only atomic antecedent are required in implications which therefore yield only atomic contexts.
- Atoms are provable either by assumption or via *backchaining* over a set of Prolog-like rules, which encode the properties of the object-logic in question. The suggestive notation $A \longleftarrow G$ corresponds to an inductive definition of a set `prog` of type $[\, atm, oo\,] \Rightarrow bool$, see Subsection 4.2. The sequent calculus is parametric in those clauses and so are its meta-theoretical properties.
- Sequents are decorated with natural numbers which represent the *height* of a proof; this measure allows reasoning by complete induction.
- For convenience we define $\Phi \rhd G$ iff there exist n such that $\Phi \rhd_n G$ and furthermore $\rhd G$ if $\cdot \rhd G$.
- The very fact that provability is inductive makes available *inversion principles* as elimination rules of the aforementioned definition. In Isabelle HOL (as well as in Coq) case analysis is particularly well-supported as part of the datatype/inductive package. For example the inversion theorem that analyses the shape of a derivation ending in an atom from the empty context is obtained simply with a call to the built-in `mk_cases` function, which specializes the elimination principle to the given constructors:

$$[\![\, \cdot \rhd \langle A \rangle;\ \bigwedge G\ i.\, [\![\, A \longleftarrow G;\ \cdot \rhd G \,]\!] \Longrightarrow P \,]\!] \Longrightarrow P$$

- The adequacy of encoding of the specification logic can be established adapting the proof in [23].

MC-Theorem 2 (Structural Rules). *The following rules are provable in Isabelle HOL:*

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:* $[\![\, \Phi, A \rhd G;\ \Phi \rhd \langle A \rangle \,]\!] \Longrightarrow \Phi \rhd G$

Proof.

1. The proof, by structural induction on sequents, consists of a one-line call to an automatic tactic using the elimination rule for successor (from the Isabelle HOL library) and the introduction rules for the sequent calculus. We mention these details in comparison with the proof of the same statement in Coq reported in [10], whose script instead consists of more than 100 lines.
2. By a similar fully automated induction on the structure of the sequent derivation:

```
be seq.induct 1;
by(auto_tac((claset() addIs seq.intrs), simpset()));
```

3. Cut is a corollary of the following lemma:

$$[\![\, \Phi' \rhd_i G;\ \Phi' = set\ (\Phi, A);\ \Phi \rhd_j \langle A \rangle \,]\!] \Longrightarrow \Phi \rhd_{i+j} G$$

easily proven by induction on the structure of the derivation of $\Phi' \rhd_i G$, using library facts relating set and list memberships. See the Appendix for more details.

4.2 Encoding the Object Logic

We introduce a datatype `atm` to encode the atomic formulae of the OL, which in this case study includes

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

We can now give the clauses for the OL deductive systems; we start with typing and evaluation, Figure 8, using the same notation of the informal ones.

$$\begin{array}{rl}
\textit{Inductive}\ _ \longleftarrow _ \ :: & [\,\textit{atm}, \textit{oo}\,] \Rightarrow \textit{bool} \\
\Longrightarrow & (E_1 \ @ \ E_2) : T \longleftarrow \langle E_1 : (T' \ \mathsf{arrow}\ T)\rangle \ \mathsf{and}\ \langle E_2 : T'\rangle \\
[\![\ \mathsf{abstr}\ E\ ]\!] \Longrightarrow & (\mathbf{lam}\ x \,.\, E\ x) : (T_1\ \mathsf{arrow}\ T_2) \longleftarrow \mathsf{all}\, x.\, (x : T_1)\ \mathsf{imp}\ \langle (E\ x) : T_2\rangle \\
[\![\ \mathsf{abstr}\ E\ ]\!] \Longrightarrow & (\mathbf{fix}\ x \,.\, E\ x) : T \longleftarrow \mathsf{all}\, x.\, (x : T)\ \mathsf{imp}\ \langle (E\ x) : T\rangle \\
\\
[\![\ \mathsf{abstr}\ E\ ]\!] \Longrightarrow & \mathbf{lam}\ x \,.\, E\ x \Downarrow \mathbf{lam}\ x \,.\, E\ x \longleftarrow \mathsf{tt} \\
[\![\ \mathsf{abstr}\ E_1'\ ]\!] \Longrightarrow & E_1 \ @ \ E_1 \Downarrow V \longleftarrow \\
& \langle E_1 \Downarrow \mathbf{lam}\ x \,.\, E_1'\ x\rangle \ \mathsf{and}\ \langle E_2 \Downarrow V_2\rangle \ \mathsf{and}\ \langle (E_1'\ V_2) \Downarrow V\rangle \\
[\![\ \mathsf{abstr}\ E\ ]\!] \Longrightarrow & \mathbf{fix}\ x \,.\, E\ x \Downarrow V \longleftarrow \langle E\ (\mathbf{fix}\ x \,.\, E\ x) \Downarrow V\rangle
\end{array}$$

Fig. 8. OL clauses:typing and evaluation

Note the presence of the *abstraction* annotations as Isabelle HOL premises in rules mentioning binding construct. This in turn allows to simulate definitional reflection via the built-in elimination rules of the `prog` inductive definition *without* the use of additional axioms. For example inversion on the abstraction typing rule is:

$$\begin{array}{l}
[\![\ (\mathbf{lam}\ x \,.\, E\ x : \tau) \longleftarrow G; \bigwedge F\ T_1\ T_2. [\![\ \mathsf{abstr}\ F; G = \mathsf{all}\, x.\, (x : T_1)\ \mathsf{imp}\ \langle F\ x : T_2\rangle); \\
\qquad\qquad \mathsf{lambda}\ E = \mathsf{lambda}\ F; \tau = (T_1 \to T_2)\ ]\!] \Longrightarrow P\ ]\!] \Longrightarrow P
\end{array}$$

Note also how the inversion principle has an explicit equation $\mathsf{lambda}\ E = \mathsf{lambda}\ F$ (whereas definitional reflection employs unification) and those are solvable under the assumption that the body of a lambda term is well-behaved, i.e. an abstraction.

Now we can address the meta-theory, starting, for example, with the subject reduction theorem:

MC-Theorem 3 (OL Subject Reduction).

$$\forall n.\, (\cdot \rhd_n E \Downarrow V) \Longrightarrow \forall T.\, (\rhd\ E : T) \to (\rhd\ V : T)$$

Proof. The proof is by complete induction on the height of the derivation of evaluation, analogously to [10] (except with an appeal to Theorem 1 rather than to the distinctness axioms).

As we remarked, the main reason to reference provability was the intrinsic incompatibility of induction with hypothetical (non-stratifiable) judgments. On the other hand, since the definition of evaluation makes no use of hypothetical judgments, it is perfectly acceptable at the meta-level, that is, we can directly give an inductive definition for evaluation:

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

Moreover, it is easy to show (formally) the two encodings equivalent:

MC-Theorem 4. $E \Downarrow V$ *iff* $\cdot \rhd_n E \Downarrow V$.

Proof. Left-to right holds by straightforward structural induction. The converse by complete induction.

This suggest we delegate to the OL level *only* those judgments, such as typing, which would not be inductive at the meta-level. This has the benefit of limiting the indirect-ness of using an explicit SL. Moreover, it has the further advantage of replacing complete with structural induction, which is better behaved from a proof-search point of view. Complete induction, in fact, places an additional burden to the user by requiring him/her to provide the correct instantiation for the height of the derivation in question. Thus subject reduction at the meta-level has the form:

MC-Theorem 5 (Meta-level Subject Reduction).

$$E \Downarrow V \Longrightarrow \forall T. (\rhd E : T) \to (\rhd V : T)$$

Proof. By structural induction on evaluation, using only inversion principles on provability and OL typing.

5 Formal Verification of Compilation

Similarly, we implement evaluation to closures $\eta \vdash F \hookrightarrow W$ at the meta-level as a straightforward (i.e. entirely first-order) inductive definition, whose rules we omit are reported in the Appendix. Compilation to dB expressions and values is instead represented at the OL level and depicted in Figure 9.

$$
\begin{array}{rl}
 & \Longrightarrow \eta \vdash (F_1 \ @' \ F_2) \leftrightarrow (E_1 \ @ \ E_2) \longleftarrow \langle \eta \vdash F_1 \leftrightarrow E_1 \rangle \ \mathsf{and} \ \langle \eta \vdash F_2 \leftrightarrow E_2 \rangle \\
[\![\ \mathsf{abstr} \ E \ ]\!] & \Longrightarrow \eta \vdash \mathbf{lam}' \ F \leftrightarrow (\mathbf{lam} \ x \, . \ E \ x) \longleftarrow \\
 & \qquad\qquad\qquad \mathsf{all}\, w.\, \mathsf{all}\, x.\, w \Leftrightarrow x \ \mathsf{imp} \ \langle (\eta; w) \vdash F \leftrightarrow (E \ x) \rangle \\
[\![\ \mathsf{abstr} \ E \ ]\!] & \Longrightarrow \eta \vdash \mathbf{fix}' \ F \leftrightarrow (\mathbf{fix} \ x \, . \ E \ x) \longleftarrow \\
 & \qquad\qquad\qquad \mathsf{all}\, f.\, \mathsf{all}\, x.\, \eta \vdash f \leftrightarrow x \ \mathsf{imp} \ \langle (\eta + f) \vdash F \leftrightarrow (E \ x) \rangle \\
 & \Longrightarrow (\eta; W) \vdash 1 \leftrightarrow E \longleftarrow \langle W \Leftrightarrow E \rangle \\
 & \Longrightarrow (\eta; W) \vdash F {\uparrow} {\leftrightarrow} \ E \longleftarrow \langle \eta \vdash F \leftrightarrow E \rangle \\
[\![\ \mathsf{abstr} \ E \ ]\!] & \Longrightarrow \{\eta, \mathbf{lam}' \ F\} \Leftrightarrow \mathbf{lam} \ x \, . \ E \ x \longleftarrow \langle \eta \vdash \mathbf{lam}' \ F \leftrightarrow \mathbf{lam} \ x \, . \ E \ x \rangle
\end{array}
$$

Fig. 9. OL clauses: compilation to dB expressions and values

We can now tackle the verification of compiler correctness:

MC-Theorem 6 (Soundness of Compilation).

$$E \Downarrow V \Longrightarrow \forall n \ \eta \ F. (\cdot \rhd_n \eta \vdash F \leftrightarrow e) \to \exists W. \eta \vdash F \hookrightarrow W \land (\rhd W \Leftrightarrow V)$$

Proof. The informal proof proceeds by lexicographic induction on the pair consisting of the derivation $\mathcal{D}$ of $E \Downarrow V$ and $\mathcal{C}$ of $\eta \vdash F \leftrightarrow e$. We examine one case, both formally and informally to demonstrate the use of hypothetical judgments and its realization in the system. Let $\mathcal{D}$ end in **ev_app** and assume $\eta \vdash F \leftrightarrow e_1 \ @ \ e_2$. Inversion on the latter yields three cases. Let us consider only the last one, where $F = F_1 \ @' \ F_2$ such that $\eta \vdash F_1 \leftrightarrow e_1$ and $\eta \vdash F_2 \leftrightarrow e_2$. By IH, $\eta \vdash F_2 \hookrightarrow W_2$ and $W_2 \Leftrightarrow v_2$. Moreover, $\eta \vdash F_1 \hookrightarrow W_1$ and $W_1 \Leftrightarrow \mathbf{lam} \ x \, . \ e'_1$. By inversion, $W_1 = \{\eta_1, \mathbf{lam}' \ F'_1\}$ and $\eta_1 \vdash \mathbf{lam}' \ F'_1 \leftrightarrow \mathbf{lam} \ x \, . \ e'_1$. By a further inversion we have a proof of $\eta_1 ; w \vdash F'_1 \leftrightarrow e'_1$ under the parametric assumption $w \Leftrightarrow x$. Now substitute W_2 for w and v_2 for x: since $W_2 \Leftrightarrow v_2$ holds, we infer $\eta_1 ; W_2 \vdash F'_1 \leftrightarrow [v_2/x]e'_1$. We can now apply the IH once more yielding $\eta_1 ; W_2 \vdash F'_1 \hookrightarrow W_3$ and $W_3 \Leftrightarrow v$. The case is finished with an appeal to rule **feval_app**.

In the formal development we proceed by a nested induction, the outermost on the structure of $E \Downarrow V$ and the innermost by complete induction on n, representing the height of the SL's proof of $\eta \vdash F \leftrightarrow e$. More details in the Appendix. Let $IH(E,V) = \forall n\ \eta\ F. \cdot \rhd_n \langle \eta \vdash F \leftrightarrow E\rangle \rightarrow \exists W. \eta \vdash F \hookrightarrow W \wedge \rhd \langle W \Leftrightarrow V\rangle$ and $G = \exists W. \eta \vdash F_1\ @'\ F_2 \hookrightarrow W \wedge \rhd \langle W \Leftrightarrow V\rangle$. Then the case for application from the outermost induction is, omitting the innermost IH which is not used here:

$$\begin{array}{c}[\![\ E_1 \Downarrow \mathbf{lam}\ x\ .\ E'\ x;\ IH(E_1, \mathbf{lam}\ x\ .\ E'_1\ x);\ \mathsf{abstr}\ E';\ E_2 \Downarrow V_2;\ IH(E_2, V_2);\\ (E'\ V_2) \Downarrow V;\ IH((E'\ V_2), V);\ \cdot \rhd_n \langle \eta \vdash F_1 \leftrightarrow E_1\rangle\ \mathsf{and}\ \langle \eta \vdash F_2 \leftrightarrow E_2\rangle\ ]\!] \Longrightarrow G\end{array}$$

Eliminating the SL conjunction "and" and applying twice the IH, we get:

$$[\![\ \ldots \exists W. \eta \vdash F_1 \hookrightarrow W \wedge \rhd \langle W \Leftrightarrow \mathbf{lam}\ x\ .\ E'\ x\rangle;\ \exists W. \eta \vdash F_2 \hookrightarrow W \wedge \rhd \langle W \Leftrightarrow V_2\rangle\ ]\!] \Longrightarrow G$$

Now we perform inversion on $W \Leftrightarrow \mathbf{lam}\ x\ .\ E'\ x$ and simplification:

$$[\![\ \ldots \rhd\ \langle \eta \vdash \mathbf{lam}'\ F'_1 \leftrightarrow \mathbf{lam}\ x\ .\ E'_1\ x\rangle \ldots\ ]\!] \Longrightarrow G$$

More inversion on $\eta_1 \vdash \mathbf{lam}'\ F'_1 \leftrightarrow \mathbf{lam}\ x\ .\ E'_1\ x$, plus Hybrid injectivity to solve $\mathsf{lambda}\ E = \mathsf{lambda}\ E'_1$:

$$\begin{array}{r}[\![\ \ldots \eta \vdash F_1 \hookrightarrow \{\eta_1, \mathbf{lam}'\ F'_1\};\ \eta \vdash F_2 \hookrightarrow W_2;\ \rhd \langle W_2 \Leftrightarrow V_2\rangle;\\ \rhd\ \mathsf{all}\, w.\, \mathsf{all}\, x.\, (w \Leftrightarrow x\ \mathsf{imp}\ \langle \eta_1;\ w \vdash F'_1 \leftrightarrow (E'_1\ x)\rangle)\ ]\!] \Longrightarrow G\end{array}$$

We now invert on the (proof of the) parametric and hypothetical judgment and then instantiate w with W_2 and x with V_2:

$$\begin{array}{r}[\![\ \ldots \eta \vdash F_1 \hookrightarrow \{\eta_1, \mathbf{lam}'\ F'_1\};\ \eta \vdash F_2 \hookrightarrow W_2;\ \rhd \langle W_2 \Leftrightarrow V_2\rangle;\\ W_2 \Leftrightarrow V_2 \rhd \langle \eta_1; W_2 \vdash F'_1 \leftrightarrow (E'_1\ V_2)\rangle\ ]\!] \Longrightarrow G\end{array}$$

After a cut we can apply the IH to $\eta_1; W_2 \vdash F'_1 \leftrightarrow (E'_1\ V_2)$ yielding $\eta_1; W_2 \vdash F'_1 \hookrightarrow W$. Then the proof is concluded with the introduction rule for application.

The converse does not contribute any new ideas and we leave it to the on-line documentation:

MC-Theorem 7 (Completeness of Compilation).

$$\eta \vdash F \hookrightarrow W \Longrightarrow \forall n\ E.\, (\cdot \rhd_n \eta \vdash F \leftrightarrow e) \rightarrow \exists V. E \Downarrow V \wedge \rhd\ W \Leftrightarrow V$$

Proof. By structural induction on $\eta \vdash F \hookrightarrow W$ and inversion on $\cdot \rhd_n \eta \vdash F \leftrightarrow e$.

5.1 Implementation of Subject Reduction for Closures

We now turn to the coinductive part. Ideally, we would implement closures and environment as a *co-datatype*; indeed, this is possible in Isabelle/ZF, but not at the moment in Isabelle HOL. We then resort to a definitional approach where we introduce a Hybrid constant $\{\_,\_\}^\infty :: [\,(val\ \ env), exp'\,] \Rightarrow val$ and we prove it to be injective as usual. There are alternatives; for example we have implemented on operational semantics with *recursive* closures, but we present here the above to be faithful to [27]. On the other hand, as remarked in Subsection 2.2, since we will not need the + environment constructor anymore, nothing prevents us from encoding here dB expressions with a Isabelle HOL datatype:

$$\mathit{datatype}\ exp' ::= 1 \mid exp'\!\uparrow\mid \mathbf{lam}'\ exp' \mid exp'\ @'\ exp' \mid \mathbf{fix}'\ exp'$$

We only mention here the new "circular" rule — a complete listing in the Appendix:

$$\begin{array}{c}\mathit{Inductive}\ \_ \vdash \_ \hookrightarrow \_\ ::\ [\,(val\ \ env), exp', val\,] \Rightarrow bool\\ [\![\ cl = \{(\eta; cl), \mathbf{lam}'\ F\}^\infty\ ]\!] \Longrightarrow \eta \vdash \mathbf{fix}'\ (\mathbf{lam}'\ F) \hookrightarrow cl\end{array}$$

After declaring a standard Isabelle HOL datatype for types environments, we encode the judgment $\Delta \vdash F : \tau$ with an inductive definitions of type $[\![\ tenv, exp', tp\ ]\!] \Rightarrow bool$, whose rules are reported in the Appendix.

More interestingly, we introduce typing of closures and type consistency of value and type environments as a *mutually co-inductive* definition:

$$
\begin{aligned}
\mathit{Conductive}\ \_ : \_ \ &:: \ [\,\mathit{val}, \mathit{tp}\,] \Rightarrow \mathit{bool} \\
\_ :' \_ \ &:: \ [\,(\mathit{val}\ \mathit{env}), \mathit{tenv}\,] \Rightarrow \mathit{bool} \\
[\![\, \Delta \vdash \mathbf{lam}'\ F : \tau;\ \eta :' \Delta \,]\!] &\Longrightarrow \{\eta, \mathbf{lam}'\ F\}^{\infty} : \tau \\
&\Longrightarrow \cdot :' \cdot \\
[\![\, W : \tau;\ \eta :' \Delta \,]\!] &\Longrightarrow (\eta; W) :' (\Delta; \tau)
\end{aligned}
$$

MC-Theorem 8.

$$\eta \vdash F \hookrightarrow W \Longrightarrow (\forall T.\, \eta :' \Delta \to (\Delta \vdash F : \tau \to W : \tau))$$

Proof. By structural induction on evaluation, where each case is proven with an appeal to an automatic tactic, which uses appropriate elimination rules. The only delicate case is the fix point, where we need to prove:

$$[\![\, cl = \{(\eta; cl), \mathbf{lam}'\ F\}^{\infty};\ \eta :' \Delta;\ \Delta \vdash \mathbf{fix}'\ (\mathbf{lam}'\ F) : \tau \,]\!] \Longrightarrow cl : \tau$$

In Isabelle HOL (co)induction is realized set theoretically via the Knaster-Tarski's construction and the user provides the right set to be checked for *density* w.r.t. the rule set. Since our definition is by mutual coinduction, the greatest fix point is constructed as a disjoint sum. Thus, the right set turns out to be $\{\mathsf{Inr}(\eta; cl), \mathsf{Inl}(\Delta; \tau)\}$. More details in the Appendix

6 Related Work

We have briefly reviewed the relevant literature regarding the automation of compiler correctness in the functional setting in Section 2. Here we turn to logical frameworks in general. We have so far concentrated on $FO\lambda^{\Delta I\!N}$, but the other major contender in the field is the *Twelf* project [35]. Meta-reasoning can be carried over in two complementary styles. In an interactive one [34], LF is used to specify a meta-theorem as a relation (type family) between judgments, a logic programming-like interpretation provides an operational semantics to those relations. Finally, external checks (termination, moded-ness, totality) verify that the given relation is indeed a realizer for that theorem [37]. The second approach [38] is built on the idea of devising 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. In the meta-logic it is possible to express and inductively prove meta-logical properties of an object logic. By the adequacy of the encoding, the proof of the existence of the appropriate LF object(s) guarantees the proof of the corresponding object-level property. It must be remarked that *Twelf* is not programmable by tactics.

Other architectures are essentially one level. For more information, we refer to the review in [2], but we just mention the orthogonal attempt to try to allow (primitive) recursive definitions on functions of higher type while preserving adequacy of representations; this has been realized for the simply-typed case in [9] and more recently for the dependently-typed case in [8]. The idea is to separate at the type-theoretic level, via an S4 modal operator, the *primitive* recursive space (which encompasses functions defined via case analysis and iteration) from the *parametric* function space (whose members are those convertible to expressions built only via the constructors). This approach has not been tested on the field yet.

Honsell et al.'s framework [20] embraces an *axiomatic* approach to meta-reasoning with HOAS. It consists of higher-order logic extended with a set of axioms parametric to a HOAS signature, including the reification of key properties of names akin to *freshness*. Further higher-order induction and recursion schemata on expressions are also assumed. To date, the consistency w.r.t. a categorical semantic has been investigated for the simply typed case. Since this approach is meant to co-exists with an inductive setting the positivity condition for hypothetical judgments and datatype must be preserved.

An original view of the notion of "freshness" is present in the FM approach [11, 36], which aims to be a foundation of programming and reasoning with *names*, in a one-level architecture. Recently Gabbay ha stared to implement nominal logic on top of Isabelle HOL [12, 13]. For our purposes it would be interesting to look at using a version of the "New" quantifier in the specification logic, especially for those applications where the behavior of the object-logic binder is not faithfully mirrored by a traditional universal quantification at the SL-level, for example the π-calculus. Sometimes we may have 'all fresh y', 'some fresh y' or, in this case, even 'some y'. In a one level world, each of these gives rise to a different principle of induction for expressions. They can be proved equivalent using techniques presented by McKinna and Pollack [24], but to do so is a great deal of work and apparently has to be repeated for each judgment which is defined inductively. Having localized this at the SL level, we may be able to prove the equivalence of 'for all fresh' and 'for some fresh' once-and-for-all using the McKinna and Pollack technique.

7 Conclusions and Future Work

We have presented a multi-level architecture to allow (co)inductive reasoning about objects defined via HOAS in a well-known environment such as Isabelle HOL. Similarly to [10] this has several benefits:

– It is possible to replicate in an well-understood and interactive setting the style of proof of $FO\lambda^{\Delta I\!N}$, so all results are proven without "technical" lemmas foreign to the mathematics of the problem.
– Results about the SL, such as cut elimination, are proven once and for all, if we are happy with that logic. Otherwise, different logics (say linear) can be exchanged and used, without changing infrastructure. This would allow, for example, the utilization of the most elegant encodings of the meta-theory of functional programming with references proposed, for example, in [23].

Differently to [10], our architecture is based not directly on a standard proof-assistant, but on a package which builds a HOAS meta-language on top of such a system. This allows us to do without of any axiomatic assumptions, such as freeness of HOAS constructors and extensionality properties at higher types, which are now theorems. Another difference is the mixing of meta-level and OL specifications, which we have shown makes proofs more easily mechanizable and allows us to use co-induction which is still unaccounted for in $FO\lambda^{\Delta I\!N}$. Finally, by the simple reason that the Hybrid system sits on top of Isabelle HOL, we benefit of the higher degree of automation of the latter.

As far as future work is concerned, we plan to further pursue our case study by finally compiling our target language into a CAM-like abstract machine, as in [18]. We shall also investigate co-inductive issues in compilation, starting with verifying the equivalence between the standard operational semantics and the one with non-well founded closures. Another property of interest that would benefit of a co-inductive slant is machine *safety* [25].

Note that in this case study we only needed to induct — either directly, or on the structure of derivations — over *closed* terms, although we extensively reasoned in presence of hypothetical judgments. Inducting over open terms is a major challenge [38] and can be accomplished in this framework by switching to a more expressive SL, based on a eigenvariable encoding, as proposed in [23]. While it is already simple enough to implement such a logic, the new theory of *N-ary abstractions* which underlines the next version of the Hybrid infrastucture will directly support this syntax, as well as a form of Isabelle HOL-like datatypes over HOAS signatures. With that in place, we will be able, for example, to revisit in a full HOAS style the material in [29].

Source files for the Isabelle HOL code can be found at

`http://www.mcs.le.ac.uk/mechsem/Hybrid/2level/Compilation`.

Acknowledgments This paper has benefited from discussions with Dale Miller, Frank Pfenning, Carsten Schürmann and Amy Felty, who kindly made available to us the Coq proof script relative to [10].

References

1. J. Alves-Foss, editor. *Formal syntax and semantics of Java*, volume 1523 of *Lecture Notes in Computer Science*. Springer-Verlag Inc., New York, NY, USA, 1999.

2. S. Ambler, R. Crole, and A. Momigliano. Combining higher order abstract syntax with tactical theorem proving and (co)induction. In V. A. Carreño, editor, *Proceedings of the 15th International Conference on Theorem Proving in Higher Order Logics, Hampton, VA, 1-3 August 2002*, volume 2342 of *Lecture Notes in Computer Scienc*. Springer Verlag, 2002.
3. S. Boutin. Proving correctness of the translation from mini-ML to the CAM with the Coq proof development system. Technical Report RR-2536, Inria, Institut National de Recherche en Informatique et en Automatique, 1995.
4. A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.
5. B. Ciesielski and M. Wand. Using the theorem prover Isabelle-91 to verify a simple proof of compiler correctness. Technical Report NU-CCS-91-20, College of Computer Science, Northeastern University, Dec. 1991.
6. D. Clement, J. Despeyroux, T. Despeyroux, and G. Kahn. A simple applicative language: Mini-ML. In *Proceedings of the 1986 ACM Conference on Lisp and Functional Programming*, pages 13–27. ACM, ACM, Aug. 1986.
7. J. Despeyroux. Proof of translation in natural semantics. In *Proceedings of LICS'86*, pages 193–205, Cambridge, MA, 1986. IEEE Computer Society Press.
8. J. Despeyroux and P. Leleu. Metatheoretic results for a modal λ-calculus. *Journal of Functional and Logic Programming*, 2000(1), 2000.
9. J. Despeyroux, F. Pfenning, and C. Schürmann. Primitive recursion for higher-order abstract syntax. In R. Hindley, editor, *Proceedings of the Third International Conference on Typed Lambda Calculus and Applications (TLCA'97)*, pages 147–163, Nancy, France, Apr. 1997. Springer-Verlag LNCS.
10. A. Felty. Two-level meta-reasoning in Coq. In V. A. Carreño, editor, *Proceedings of the 15th International Conference on Theorem Proving in Higher Order Logics, Hampton, VA, 1-3 August 2002*, volume 2342 of *Lecture Notes in Computer Scienc*. Springer Verlag, 2002.
11. M. Gabbay and A. Pitts. A new approach to abstract syntax involving binders. In G. Longo, editor, *Proceedings of the 14th Annual Symposium on Logic in Computer Science (LICS'99)*, pages 214–224, Trento, Italy, July 1999. IEEE Computer Society Press.
12. M. J. Gabbay. Automating Fraenkel-Mostowski syntax. In *TPHOLs, 15th International Conference on Theorem Proving in Higher Order Logics*, August 2002. Work-in-progress submission, to be published as a NASA technical report.
13. M. J. Gabbay. FM-HOL, a higher-order theory of names. In F. Kamareddine, editor, *35 Years of Automath*, `http://www.cee.hw.ac.uk/~fairouz/automath2002/informal-proceedings/`, April 2002. Heriot-Watt University, Edinburgh, Scotland.
14. E. Gimenez. A tutorial on recursive types in coq. Technical Report RT-0221, Inria, Institut National de Recherche en Informatique et en Automatique, 1998.
15. L. Hallnas. Partial inductive definitions. *Theoretical Computer Science*, 87(1):115–147, July 1991.
16. J. Hannan. Staging transformations for abstract machines. In P. Hudak and N. Jones, editors, *Proceedings of the ACM SIGPLAN Symposium on Partial Evaluation and Semantics Based Program Manipulation*, pages 130–141, New Haven, Connecticut, June 1991.
17. J. Hannan and D. Miller. From operational semantics to abstract machines. *Mathematical Structures in Computer Science*, 2(4):415–459, 1992.
18. J. Hannan and F. Pfenning. Compiler verification in LF. In A. Scedrov, editor, *Seventh Annual IEEE Symposium on Logic in Computer Science*, pages 407–418, Santa Cruz, California, June 1992.
19. R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, Jan. 1993.
20. F. Honsell, M. Miculan, and I. Scagnetto. An axiomatic approach to metareasoning on systems in higher-order abstract syntax. In *Proc. ICALP'01*, number 2076 in LNCS, pages 963–978. Springer-Verlag, 2001.
21. D. Lester and S. Mintchev. Towards machine–checked compiler correctness for higher–order pure functional languages. In L. Pacholski and J. Tiuryn, editors, *Proceedings of the 1994 Annual Conference of the European Association for Computer Science Logic*, pages 369–381. Springer-Verlag LNCS 933, 1995.
22. R. McDowell. *Reasoning in a Logic with Definitions and Induction*. PhD thesis, University of Pennsylvania, 1997.
23. R. McDowell and D. Miller. Reasoning with higher-order abstract syntax in a logical framework. *ACM Transactions on Computational Logic*, 3(1):80–136, January 2002.
24. J. McKinna and R. Pollack. Some lambda calculus and type theory formalized. *JAR*, 1998.
25. N. G. Michael and A. A. Appel. Machine instruction, syntax and semantics in higher-order logic. In D. McAllister, editor, *CADE-17*, volume 1831 of *LNCS*, pages 7–24. Springer, 2000.
26. D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.
27. R. Milner and M. Tofte. Co-induction in relational semantics. *Theoretical Computer Science*, 87:209–220, 1991.
28. R. Milner and R. Weyrauch. Proving compiler correctness in a mechanized logic. In B. Meltzer and D. Michie, editors, *Machine Intelligence 7*, pages 51–72. Edinburgh University Press, 1972.

29. A. Momigliano, S. Ambler, and R. Crole. A hybrid encoding of howe's method for establishing congruence of bisimilarity. *ENTCS*, 70(2), 2002.
30. G. Nadathur and D. Miller. An overview of λProlog. In K. A. Bowen and R. A. Kowalski, editors, *Fifth International Logic Programming Conference*, pages 810–827, Seattle, Washington, Aug. 1988. MIT Press.
31. C. Paulin-Mohring. Inductive definitions in the system Coq: Rules and properties. In M. Bezem and J. Groote, editors, *Proceedings of the International Conference on Typed Lambda Calculi and Applications*, pages 328–345, Utrecht, The Netherlands, Mar. 1993. Springer-Verlag LNCS 664.
32. L. C. Paulson. A fixedpoint approach to implementing (co)inductive definitions. In A. Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, pages 148–161, Nancy, France, June 1994. Springer-Verlag LNAI 814.
33. F. Pfenning. *Computation and Deduction*. Cambridge University Press, 2000. In preparation. Draft from April 1997 available electronically.
34. F. Pfenning and E. Rohwedder. Implementing the meta-theory of deductive systems. In D. Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction*, pages 537–551, Saratoga Springs, New York, June 1992. Springer-Verlag LNAI 607.
35. F. Pfenning and C. Schürmann. System description: Twelf — a meta-logical framework for deductive systems. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction (CADE-16)*, pages 202–206, Trento, Italy, July 1999. Springer-Verlag LNAI 1632.
36. A. M. Pitts. Nominal logic: A first order theory of names and binding. In N. Kobayashi and B. C. Pierce, editors, *Theoretical Aspects of Computer Software, 4th International Symposium, TACS 2001, Sendai, Japan, October 29-31, 2001, Proceedings*, volume 2215 of *Lecture Notes in Computer Science*, pages 219–242. Springer-Verlag, Berlin, 2001.
37. E. Rohwedder and F. Pfenning. Mode and termination checking for higher-order logic programs. In H. R. Nielson, editor, *Proceedings of the European Symposium on Programming*, pages 296–310, Linköping, Sweden, Apr. 1996. Springer-Verlag LNCS 1058.
38. C. Schürmann. *Automating the Meta-Theory of Deductive Systems*. PhD thesis, Carnegie-Mellon University, 2000. CMU-CS-00-146.
39. C. Schürmann. A type-theoretic approach to induction with higher-order encodings. In *Proceedings of Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2001)*, volume 2142 of *Lecture Notes in Computer Science*, pages 266–281, 2001.

A Appendix

Summary of judgments: object logic

- Call-by-value operational semantics: $e \Downarrow v$ and type inference $\Gamma \vdash e : \tau$, for the source language, Figure 2.
- Compilation from modified dB to closures: $\eta \vdash F \hookrightarrow W$, Figure 3.
- Compilation to modified dB: $\eta \vdash F \leftrightarrow e$ and values: $W \Leftrightarrow v$ Figure 4.
- Typing of modified dB: $\eta \vdash F : \tau$ Figure 5.
- (Coinductive) Typing of closures: $W : \tau$ Figure 6.

Specification level:

- Sequent calculus: $\Phi \rhd_n G$, Figure 7.
- `Prog` clauses: $A \longleftarrow G$, Figure 8 and 9.

Encoding of typing and evaluation for closures, in both flavors:

```
inductive (_ |- _ >>> _) :: [Val env, dexp, Val] => bool
fev_1 "K ; W |- 1 >> W"
fev_1p "K |- F >> W ==> K +++ F |- 1 >> W"
fev_s "K |- F >> W ==> K ; W' |- F ^ >> W"
fev_sp "K |- F >> W ==> K +++ F' |- F ^ >> W"
fev_lam "K |- lam' F >> clo K (lam' F)"
fev_app "[| K |- F1 >> clo K' (lam' F1'); K |- F2 >> W2; \
\         K';W2 |- F1' >> W|] ==> K |- F1 $' F2 >> W"
fev_fix " K +++ (fix' F) |- F >> W ==>  K |- fix' F >> W"
```

```
(* ‘‘coinductive’’ version *)
fev’_1 "K ; W |- 1 >> W"
fev’_s "K  |- F >> W ==> K ; W’ |- ( F ^ )  >> W"
fev’_lam "K |- lamd F >> clo K (lam’ F)"
fev’_app "[| K |- F1 >> clo K’ (lam’ F1’); K |- F2 >> W2; \
\          K’;W2 |- F1’ >> W|] ==>  K |- F1 $’ F2 >> W"
fev’_fix " cl = <|( K;cl) , lam’ F |> ==> \
                \   K |- fix’ (lam’ F) >> cl"

inductive (_ |- _ : _) :: [tenv, dexp, tp] => bool
hs_1 "TE ;; A |- 1 : A"
hs_s "TE |- F : A ==> TE ;; A’ |- F ^ : A"
hs_lam "TE ;; A1 |- F : A2 ==> TE |- lam’ F : ar A1 A2"
hs_fix "TE ;; A |- F : A ==> TE |- fix’ F : A"
hs_app "[| TE |- F1 : ar A2 A; TE |- F2 : A2|] ==> TE |- F1 $’ F2 : A"
```

Sample Code: we refer to the on-line documentation for unexplained lemmas.
Cut-elimination:

```
Goal "Delta |- B ::: i ==> !A j Gamma . set Delta = set (A # Gamma) --> \
                        \ (Gamma |- ( < A >) ::: j --> Gamma |- B ::: i + j)";
be seq.induct 1;
by(Safe_tac );

(* backchaining *)
by(force_tac((claset() addIs seq.intrs), simpset()) 1);

(* axiom *)
by(case_tac "A= Aa" 1);auto();
by(case_tac "i = 0 " 1); force 1;
br bound_w 1;auto();        (* weakening for bounds *)
br sax 1;
by(asm_full_simp_tac(simpset() addsimps [set_mem_eq]) 1);

(* tt *)
brs seq.intrs 1;

(* and *)
by(force_tac((claset() addIs seq.intrs), simpset()) 1);

(* forall *)
by(force_tac((claset() addIs seq.intrs), simpset()) 2);

(* imp *)
by(Simp_tac 1);
brs seq.intrs 1;
by(eres_inst_tac [("x"," Aa")] allE 1);
by(eres_inst_tac [("x"," j")] allE 1);
by(eres_inst_tac [("x"," A # Gammaa")] allE 1);
bd mp 1; force 1;
bd seq_weak 1;back(); force 1;  (* context weakening *)
qed_spec_mp "seq_cut_aux" ;
```

Co-inductive subject reduction:

```
Goal "eta |- F >> W ==> \
\  (! t te. eta hastyenv te --> te |- F : t --> W hasty t)";
by (etac feval.induct 1);
by(Safe_tac);

(* 1 *)
by(etacs has_one_elim 1);
be hastyenv_E 1; ba 1;

(* shift *)
by(etacs has_sh_elim 1);
be hastyenv_E 1; force 1;

(* lam *)
by(fast_tac(HOL_cs addSIs [ hasty_rel_hastyenv_rel.hasty_clos]) 1);

(* app *)
by(blast_tac(HOL_cs addSEs[has_app_elim,has_fn_elim]
                    addDs[hasty_elim_clos, hasty_rel_hastyenv_rel.hr]) 1);

(* fix *)

val [_,defhasty,_] = hasty_rel_hastyenv_rel.defs;
by(simp_tac(simpset () addsimps [defhasty]) 1);
be has_fix_elim 1;
by(res_inst_tac [("X"," {Inr((K ; cl),(te ;; t)), Inl (cl,t)}")]
                hasty_rel_hastyenv_rel.coinduct 1);
auto();
qed_spec_mp "consistency";
```

Soundness of compilation:

```
Goal " E >>> V ==> !n K F . [] |- (< trans K F E >) ::: n --> \
 \           (? W . K |- F >> W & [] |-- (<vtrans W V>))";
be meval.induct 1;

(* eval induction lam case *)
(* nested induction *)
br allI 1;
by (rtac nat_less_induct 1);
by(Clarify_tac 1);

(* inversion on trans *)
by(bcf_tac trans_lam'_E );

(* F is 1 *)
by(force_tac((claset() addSEs [] addSIs []@feval.intrs),
                       (simpset() addsimps[exseq_def]))1);

(* F is  Fa^ *)
by(eres_inst_tac [("x"," j:: nat")] allE 1);
by(blast_tac(claset() addSIs feval.intrs ) 1);
```

```
(* F is 1, K = Ka +++ Fa *)
by(eres_inst_tac [("x"," j:: nat")] allE 1);
by(blast_tac(claset() addSIs feval.intrs ) 1);

(* F is Fa^, K = Ka +++ Fa *)
by(eres_inst_tac [("x"," j:: nat")] allE 1);
by(blast_tac(claset() addSIs feval.intrs ) 1);

(* F is lam' Fa *)
by(hysimp [] 1);
by(force_tac((claset() addSEs [trans_lam'_E] addSIs [sbc,vtr_lam]@feval.intrs),
                       (simpset() addsimps[exseq_def]))1);

(* eval induction case, app case *)
(* nested induction *)

br allI 1;
by (rtac nat_less_induct 1);
by(Clarify_tac 1);

(* inversion and exclusion *)
be seq_atom_nil_inv 1;
by(etacs trans_app_E 1);

(* F = 1, impossible *)

by(fast_tac(HOL_cs addSEs [seq_atom_nil_inv,vtransEs] addss (simpset() )) 1);

(* F = Fa^ *)
by(hyp_subst_tac 1);
by(eres_inst_tac [("x"," i:: nat")] allE 1);
back();back();back();
by(blast_tac(claset() addSIs feval.intrs ) 1);

(* F = 1, +++ case *)
by(hyp_subst_tac 1);
by(eres_inst_tac [("x"," i:: nat")] allE 1);
back();back();back();
by(blast_tac(claset() addSIs feval.intrs ) 1);

(* F = Fa^, +++ case *)
by(hyp_subst_tac 1);
by(eres_inst_tac [("x"," i:: nat")] allE 1);
back();back();back();
by(blast_tac(claset() addSIs feval.intrs ) 1);

(* app *)
(* as described in Section *)
...
(* eval induction case fix case *)
(* nested induction *)
...
(* other cases as above *)
```

```
(* F = fix' Fa *)
by(hysimp [] 1);
by(etacs seq_all_inv 1);
by(eres_inst_tac [("x"," fix' Fa")] allE 1);
by(etacs seq_all_inv 1);
by(eres_inst_tac [("x"," Fix Ea :: uexp")] allE 1);
by(etacs seq_imp_inv 1);
by (forw_inst_tac [("i","ib"),("j"," Suc (Suc (Suc (Suc ib)))")] bound_w 1);
force 1;
by(fast_tac(claset() addSDs [seq_cut_ex] addSIs feval.intrs
                     addss (simpset() addsimps [exseq_def])) 1);

qed "map_eval1";
```