Two-Level Hybrid: A System for Reasoning Using Higher-Order Abstract Syntax

Alberto Momigliano,[a] Alan J. Martin,[b] Amy P. Felty[c,b]

[a] *LFCS, School of Informatics, University of Edinburgh, U.K.& DSI, University of Milan, Italy*
Email: amomigl1@inf.ed.ac.uk

[b] *Department of Mathematics and Statistics, University of Ottawa, Canada*
Email: {amart045,afelty}@site.uottawa.ca

[c] *School of Information Technology and Engineering (SITE), University of Ottawa, Canada*

Abstract

Logical frameworks supporting *higher-order abstract syntax* (HOAS) allow a direct and concise specification of a wide variety of languages and deductive systems. Reasoning about such systems within the same framework is well-known to be problematic. We describe the new version of the Hybrid system, implemented on top of Isabelle/HOL (as well as Coq), in which a de Bruijn representation of λ-terms provides a definitional layer that allows the user to represent object languages in HOAS style, while offering tools for reasoning about them at the higher level. We briefly describe how to carry out two-level reasoning in the style of frameworks such as Linc, and briefly discuss our system's capabilities for reasoning using tactical theorem proving and principles of induction and coinduction.

Keywords: higher-order abstract syntax, interactive theorem proving, induction, variable binding, Isabelle/HOL

1 Introduction

We give a system presentation of Hybrid [4] (`http://hybrid.dsi.unimi.it/`), in coincidence with the release of its new and first official version, as well as with the porting to Isar and Coq [2,1]. Hybrid is a package that introduces a binding operator that (1) allows a direct expression of λ-abstraction in full higher-order abstract syntax (HOAS) style, and (2) is *defined* in such a way that expanding its definition results in the conversion of a term to its de Bruijn representation [8]. The latter makes Hybrid's specifications compatible with principles of (co)induction, available in standard proof assistants. The basic idea is inspired by the work of Gordon [10], where bound variables are presented to the user as strings. Instead of strings, we use a binding operator (LAM) defined using λ-abstraction at the meta-level. Hybrid provides a library of operations and lemmas to reason on the HOAS level, hiding the details of the de Bruijn representation. Hybrid originated as a (meta)language on top of Isabelle/HOL for reasoning over languages with bindings, aiming to enrich a traditional inductive setting with a form of HOAS. It soon became apparent

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: `www.elsevier.nl/locate/entcs`

that it could also provide definitional support for *two-level* reasoning as proposed in our previous work [9]; the latter aimed to endow Coq (in that case) with the style of reasoning of frameworks such as $FO\lambda^{\Delta\mathbb{N}}$ [12], Linc [17], and, to a lesser extent, Twelf [16]. Hybrid is defined as an Isabelle/HOL theory (approx. 150 lines of functional definitions and 400 lines of statements and proofs). In contrast to other approaches such as the Theory of Contexts [11], our Isabelle/HOL theory does not contain any axioms, which would require external justification.

Previous work has described Hybrid applied to a variety of object languages (OLs) and their (meta)properties, as detailed in Section 4. Here we concentrate on describing some new features of the system infrastructure, as well as recalling the two-level approach. We briefly mention some initial tactical support for reasoning in this setting, and describe how we state the adequacy of Hybrid's encodings.

The main improvement of this version is an overall reorganization of the infrastructure, based on the internalization of the set of *proper* terms: those are the subset of de Bruijn terms that are well-formed (in the sense that all indices representing bound variable occurrences in a term have a corresponding binder in the term), eliminating the need for adding well-formedness annotations in OL judgments. We recall that to represent syntax in the presence of binders, we use a predicate (abstr) that recognizes the *parametric* part of the function space between OL expressions. The crucial injectivity property of the Hybrid binder LAM:

$$\mathsf{abstr}\ f \Longrightarrow (\mathsf{LAM}\,x.\,f\,x = \mathsf{LAM}\,x.\,g\,x) = (f = g).$$

strengthens our previous version by requiring only one of f and g to satisfy this condition (instead of both), thus simplifying the elimination rules for inductively defined OL judgments.

Notation: An Isabelle/HOL type declaration has the form $s :: [\,t_1, \ldots t_n\,] \Rightarrow t$. Free variables are implicitly universally quantified. We use $\equiv$ and $\bigwedge$ for *equality by definition* and universal meta-quantification. We use the usual logical symbols for the connectives of Isabelle/HOL. A rule (a sequent) with premises $H_1 \ldots H_n$ and conclusion C will be represented as $[\![\, H_1; \ldots; H_n \,]\!] \Longrightarrow C$. The keyword *Inductive* introduces an inductive relation in Isabelle/HOL; similarly for *datatype*. We freely use infix notations, without explicit declarations.

2 Using Hybrid

The objective of Hybrid is to provide support for HOAS via an *approximation* of the following datatype definition, which is not well-formed in an inductive setting:

$$\textit{datatype}\ \alpha\ \textit{expr} = \mathsf{CON}\ \alpha \mid \mathsf{VAR}\ \textit{var} \mid \textit{expr}\ \$\ \textit{expr} \mid \mathsf{LAM}\ (\underline{\textit{expr}} \Rightarrow \textit{expr})$$

where *var* is a countably infinite set of free variables, and the type parameter α is used to supply object-language-specific constants. (It will henceforth be omitted, except where instantiated.) The problem is, of course, LAM, whose argument type involves a negative occurrence (underlined) of *expr*. Since it is possible to formalize Cantor's diagonal argument in Isabelle/HOL, such a function cannot be injective, and thus cannot be a constructor of a datatype.

However, for HOAS it is neither necessary nor desirable for LAM to be used with arbitrary Isabelle/HOL functions as arguments: only those functions that use their arguments *generically* are needed. These functions will be called *abstractions*; in the Hybrid system, they are recognized by a predicate $\mathsf{abstr} :: [\,\textit{expr} \Rightarrow \textit{expr}\,] \Rightarrow \textit{bool}$. Functions that are not abstractions are called *exotic*, for example $\lambda\, x.\, \mathsf{if}\ x = \mathsf{CON}\ a\ \mathsf{then}\ (x \$ x)\ \mathsf{else}\ x$ for some OL constant a. The possibility of introducing such functions would break the adequacy of any second-order encoding.

As described in Section 3, Hybrid defines a type *expr* together with functions of the appropriate types to replace the constructors of the problematic datatype definition above. Distinctness of the constructors and injectivity of CON, VAR, and \$ are proved. Two more properties would be needed for a datatype: injectivity of LAM and an induction principle. In Hybrid, the former is weakened by the addition of an abstr premise, while the latter uses a nonstandard LAM case to allow the induction hypothesis to have the type $\mathit{expr} \Rightarrow \mathit{bool}$ despite the presence of HOAS:

$$[\![\ \ldots ; \bigwedge v.\, P\,(e\,(\mathsf{VAR}\ v)) \Longrightarrow P\,(\mathsf{LAM}\ e)\ ]\!] \Longrightarrow P\,(u :: \mathit{expr})$$

Thus, the type *expr* is only a "quasi-datatype", and it is necessary to impose abstr conditions wherever LAM is used. Hybrid provides lemmas for proving the resulting abstr subgoals, so that basic reasoning about *expr* is similar to a true datatype. However, primitive recursion on *expr* is not (currently) available. Also, induction involves the use of explicit free variables, which HOAS techniques normally seek to avoid, and this complicates the reasoning; thus, induction on OL judgments is preferred.

From the user's point of view Hybrid provides a form of HOAS where: object level constants correspond to expressions of the form $\mathsf{CON}\ c$; bound object-level variables correspond to (bound) meta variables in expressions of the form $\mathsf{LAM}\, v.\, e$; subterms are combined with \$; and free object-level variables may be represented as $\mathsf{VAR}\ i$ (although typical examples will not use this feature).

2.1 Example

As a small example (space limitations), we consider the encoding of the types of $F_{<:}$, the subtyping language of the PoplMark challenge [6] as an OL. Types have the form $\top$ (the maximum type), $\tau_1 \to \tau_2$ (type of functions), or $\forall x{<:}\tau_1.\tau_2$ (bound universal type). In the latter, x is a type variable possibly occurring in τ_2, which can be instantiated with subtypes of τ_1. To represent the OL types, we define:

$$\mathit{datatype}\ \mathit{con} = \mathit{cTOP} \mid \mathit{cARR} \mid \mathit{cUNI} \qquad \mathit{uexp} = \mathit{con}\ \mathit{expr}$$

$$\begin{aligned} \mathsf{top} &\equiv \mathsf{CON}\ \mathit{cTOP} \\ t_1\ \mathsf{arrow}\ t_2 &\equiv \mathsf{CON}\ \mathit{cARR} \$ t_1 \$ t_2 \\ \mathsf{univ}\, t_1\, (x.\, t_2\ x) &\equiv \mathsf{CON}\ \mathit{cUNI} \$ t_1 \$ \mathsf{LAM}\, x.\, t_2\ x. \end{aligned}$$

Note that *uexp* is introduced to abbreviate an instantiated version of the "quasi-datatype" *expr*, where α is replaced by the above type *con* that is introduced specifically for this OL.

To illustrate the representation of judgments of an OL, we consider rules for well-formed types, where Γ is a list of distinct type variables.

$$\frac{x \in \Gamma}{\Gamma \vdash x} \qquad \frac{}{\Gamma \vdash \top} \qquad \frac{\Gamma \vdash \tau_1 \qquad \Gamma \vdash \tau_2}{\Gamma \vdash \tau_1 \to \tau_2} \qquad \frac{\Gamma \vdash \tau_1 \qquad \Gamma, x \vdash \tau_2}{\Gamma \vdash \forall x{<:}\tau_1.\tau_2}$$

The standard Twelf-style encoding of the right premise of the last rule would be $\bigwedge x.\ \underline{\mathsf{isTy}\ x} \Longrightarrow \mathsf{isTy}\ (T_2\ x)$, where isTy is a meta-level predicate introduced to represent this OL judgment. Note the negative occurrence (underlined) of the predicate being defined (the same problem as before, but at the predicate level). *Two-level reasoning* is introduced to address this problem. In particular, a *specification logic* (SL) is defined inductively in Isabelle/HOL, which is in turn used to drive the encoding of the OL as an inductive set of Prolog-like clauses, avoiding any negative occurrences at the meta-level.

A Hybrid user can specify his/her own SL, but we envision a library of such logics that a user can choose from. Indeed, several such logics have been encoded to date. We can view our realization of the two-level approach as a way of "fast prototyping" HOAS logical frameworks. We can quickly implement and experiment with a potentially interesting SL, without the need to develop all the building blocks of a usable new framework, such as unification algorithms, type inference or proof search; instead we rely on the ones provided by Isabelle/HOL.

To illustrate, we choose a simple SL, a sequent formulation of a fragment of first-order minimal logic with backchaining, adapted from [12]. Its syntax can be encoded directly with an Isabelle/HOL datatype:

$$\mathit{datatype}\ \ \mathit{oo} = \mathsf{tt} \;\Big|\; \langle \mathit{atm} \rangle \;\Big|\; \mathit{oo}\ \ \mathsf{and}\ \ \mathit{oo} \;\Big|\; \mathit{atm}\ \ \mathsf{imp}\ \ \mathit{oo} \;\Big|\; \mathsf{all}\ (\mathit{uexp} \Rightarrow \mathit{oo})$$

where atm is a parameter used to represent atomic predicates of the OL and $\langle \_ \rangle$ coerces atoms into propositions. We use the symbol $\rhd$ for the sequent arrow of the SL, in this case decorated with natural numbers to allow reasoning by (complete) induction on the height of a proof. The inference rules of the SL are represented as the following Isabelle/HOL inductive definition:

$$\begin{array}{rl}
\multicolumn{2}{c}{\mathit{Inductive}\ \_ \rhd_{\_}\ \_\ ::\ [\,\mathit{atm\ set}, \mathit{nat}, \mathit{oo}\,] \Rightarrow \mathit{bool}} \\
& \Longrightarrow \Gamma \rhd_n \mathsf{tt} \\
[\![\, \Gamma \rhd_n G_1; \Gamma \rhd_n G_2 \,]\!] & \Longrightarrow \Gamma \rhd_{n+1} (G_1\ \mathsf{and}\ G_2) \\
[\![\, \forall x.\, \Gamma \rhd_n G\ x \,]\!] & \Longrightarrow \Gamma \rhd_{n+1} (\mathsf{all}\, x.\, G\ x) \\
[\![\, \{A\} \cup \Gamma \rhd_n G \,]\!] & \Longrightarrow \Gamma \rhd_{n+1} (A\ \mathsf{imp}\ G) \\
[\![\, A \in\ \Gamma \,]\!] & \Longrightarrow \Gamma \rhd_n \langle A \rangle \\
[\![\, A \longleftarrow G; \Gamma \rhd_n G \,]\!] & \Longrightarrow \Gamma \rhd_{n+1} \langle A \rangle
\end{array}$$

The backward arrow in $A \longleftarrow G$ in the last rule is used to encode OL judgments as logic programming style clauses. To reason about OLs, a small set of structural rules of the SL is proved once and for all, such as weakening and cut elimination. To complete our example OL, we define atm as a datatype with a single constructor $\mathsf{isTy}\ \mathit{uexp}$, thus representing the OL well-formedness judgment '$\vdash$' at the specification level, and encode the OL inference rules as the following definition of $(\_ \longleftarrow \_)$:

$$\begin{array}{rl}
\multicolumn{2}{l}{\mathit{Inductive}\ \_ \longleftarrow \_\ ::\ [\,\mathit{atm}, \mathit{oo}\,] \Rightarrow \mathit{bool}} \\
& \Longrightarrow \mathsf{isTy}\ \mathsf{top} \longleftarrow \mathsf{tt} \\
& \Longrightarrow \mathsf{isTy}\ (T_1\ \mathsf{arrow}\ T_2) \longleftarrow \langle \mathsf{isTy}\ T_1 \rangle\ \mathsf{and}\ \langle \mathsf{isTy}\ T_2 \rangle \\
[\![\, \mathsf{abstr}\ T_2 \,]\!] & \Longrightarrow \mathsf{isTy}\ (\mathsf{univ}\, T_1\, (x.\, T_2\ x)) \longleftarrow \\
& \quad \langle \mathsf{isTy}\ T_1 \rangle\ \mathsf{and}\ (\mathsf{all}\, x.\, (\mathsf{isTy}\ x)\ \mathsf{imp}\ \langle \mathsf{isTy}\ (T_2\ x) \rangle)
\end{array}$$

Note the negative occurrence of isTy now embedded in the SL. We remark that Hybrid is (currently) untyped in the sense that OL syntax is encoded as terms of type *uexp*; i.e., there is no new type introduced to represent well-formed types of $F_{<:}$. For this reason, the isTy predicate is necessary to identify the required subset of *uexp*.

For any OL represented in Hybrid, it is important to show that both terms and judgments are adequately encoded. For our example, this means showing that there is a bijection between object-level types of $F_{<:}$ and the subset of terms of type *uexp* formed from only variables, the constants top, arrow, and univ, and Isabelle/HOL λ-abstractions (the latter of which can only appear as the second argument of univ). We write ε_X for the *encoding* function from OL terms with free variables in X to terms in *uexp*, and δ_X for its inverse *decoding*. We also need to show that substitution commutes with the encoding. For OL judgments, the proof obligation is that whenever $x_1, \ldots, x_n \vdash t$ holds in the OL, then for some i, $(\mathsf{isTy}\ x_1, \ldots, \mathsf{isTy}\ x_n) \rhd_i \mathsf{isTy}\ (\varepsilon_{x_1,\ldots,x_n}(t))$ is provable. Conversely, whenever $(\mathsf{isTy}\ x_1, \ldots, \mathsf{isTy}\ x_n) \rhd_i \mathsf{isTy}\ T$ is provable, then T is in the domain of $\delta_{x_1,\ldots,x_n}$ and $x_1, \ldots, x_n \vdash \delta_{x_1,\ldots,x_n}(T)$ holds in the OL. It is also important to show the adequacy of SL encodings. For the SL presented here, we can adapt the proof from [12]. Since our version is defined inductively, the inversion properties of this definition play a central role in the proof.

2.2 Tactical support

We chose to develop Hybrid as a package, rather than a standalone system mainly to exploit all the reasoning capabilities that a mature proof assistant can provide, in particular support for tactical theorem proving. Contrast this with a system such as Twelf, where proofs are coded as logic programs and post hoc checked for correctness. At the same time, our aim is to try to retain some of the conciseness of a language such as LF, which for us means hiding most of the administrative reasoning concerning variable binding and contexts. Because of the "hybrid" nature of our approach, this cannot be completely achieved, but some simple-minded tactics go a long way into mechanizing most of boilerplate scripting. While in the previous version of the system we employed specific tactics to recognize *proper* terms and *abstractions*, now this is completely delegated to Isabelle's simplification. Thus, we can concentrate on assisting two-level reasoning, which would otherwise be encumbered by the indirection in accessing OL specifications via the SL. Luckily, Twelf-like reasoning consists, at a high-level, of three basic steps: inversion, backchaining (filling, in Twelf's terminology) and recursion. This corresponds to highly stereotyped proof scripts that we have abstracted into:

(i) an *inversion* tactic `defL_tac`, which goes through the SL and applies as an elimination rule one of the OL clauses. This is complemented by the eager application of other *safe* elimination rules (viz. invertible SL rules such as conjunction elimination). This contributes to keeping the SL overhead to a minimum;

(ii) a dual *backchaining* tactic `defR_tac`; the latter is integrated into the tactic `2lprolog_tac`, which performs automatic depth first search (or other searches supported by Isabelle) on Prolog-like goals;

(iii) a *complete induction* tactic, to be fired when given the appropriate derivation height by the user.

These tactics have been tested most extensively on the minimal SL, while more human intervention is required when using sub-structural logics (such as *Olli* [15]), given the non-deterministic nature of their context management.

3 Definition of Hybrid in Isabelle/HOL

The Hybrid system defines the type *expr* in terms of an Isabelle/HOL datatype *dB* that uses de Bruijn indices to represent bound variables:

$$\mathit{datatype}\ \alpha\ dB = \mathsf{CON}'\,\alpha \mid \mathsf{VAR}'\,\mathit{var} \mid \mathsf{BND}'\,\mathit{bnd} \mid dB\ \$'\ dB \mid \mathsf{ABS}'\,dB \mid \mathsf{ERR}'$$

where the type *bnd* of de Bruijn indices is defined to be the natural numbers, and the type parameter α is the same as for *expr*.

Each occurrence of $\mathsf{BND}'\,i$ refers to the variable implicitly bound by the $(i+1)^{\text{th}}$ enclosing ABS' node. If there are not enough enclosing ABS' nodes, then it is called a *dangling index*. A term without dangling indices is called *proper*, and *expr* should consist of the proper terms of type *dB*; but since the subterms of a proper term are not always proper, a more general notion of *level* is needed. Thus, Hybrid defines a predicate $\mathsf{level} :: [\,\mathit{nat}, dB\,] \Rightarrow \mathit{bool}$ by primitive recursion; the meaning of $\mathsf{level}\,i\,s$ is that the term s would have no dangling indices if enclosed in at least i ABS' nodes.

Using Isabelle/HOL's *typedef* mechanism, the type *expr* is defined as a *bijective image* of the set $\{\, s :: dB \mid \mathsf{level}\,0\,s \,\}$, with inverse bijections $\mathsf{dB} :: \mathit{expr} \Rightarrow dB$ and $\mathsf{expr} :: dB \Rightarrow \mathit{expr}$. In effect, *typedef* makes *expr* a subtype of *dB*, but since Isabelle/HOL's type system does not have subtyping, the conversion function dB must be explicit. The notation $\ulcorner s \urcorner = \mathsf{dB}\ s$ and $\llcorner s \lrcorner = \mathsf{expr}\ s$ will be used below, although dB and expr will still be used when referring to them as functions[1]. At this point three of the four constructors of *expr* can be defined:

$$\mathsf{CON}\ a \equiv \llcorner \mathsf{CON}'\,a \lrcorner \qquad \mathsf{VAR}\ n \equiv \llcorner \mathsf{VAR}'\,n \lrcorner \qquad s\ \$\ t \equiv \llcorner \ulcorner s \urcorner\ \$'\ \ulcorner t \urcorner \lrcorner$$

To define the predicate abstr and the remaining constructor LAM, it is helpful first to explicitly represent the structure of abstractions. Thus, Hybrid defines a polymorphic datatype *dB_fn*:

$$\begin{aligned}\mathit{datatype}\ (\beta,\alpha)\ dB\_fn = {} & \mathsf{ATOM}^*\,(\beta \Rightarrow \alpha\ dB) \mid \mathsf{CON}^*\,\alpha \mid \mathsf{VAR}^*\,\mathit{var} \mid \\ & \mathsf{BND}^*\,\mathit{bnd} \mid dB\_fn\ \$^*\ dB\_fn \mid \mathsf{ABS}^*\,dB\_fn \mid \mathsf{ERR}^*\end{aligned}$$

(where the type parameter α will once again be left implicit), together with a function $\text{-}_\star :: \beta\ dB\_fn \Rightarrow (\beta \Rightarrow dB)$ that maps the constructors of $\beta\ dB\_fn$ to corresponding constructors of *dB* applied pointwise, e.g., $(S\ \$^*\ T)_\star = \lambda\,x.\,(S_\star\,x)\ \$'\ (T_\star\,x)$, except that $(\mathsf{ATOM}^*\,f)_\star = f$.

A function is called *ordinary* if it is the image under $\text{-}_\star$ of a term whose root node is not ATOM^*, and a term of type $\beta\ dB\_fn$ is called *full* if in all of its occurrences of $\mathsf{ATOM}^*\,f$, the function f is not ordinary. Every function $f :: \beta \Rightarrow dB$ can be written uniquely in the form $T_\star$ for some full term $T :: \beta\ dB\_fn$: the non-ATOM^* constructors represent the common structure of $f\,x$ for all values of x, while the

[1] The function expr in the Isabelle/HOL theory is actually modified from the one provided by *typedef*, to produce a well-behaved result even when presented with a term of nonzero level. In particular, we will have $\ulcorner s\ \$\ t \urcorner = \ulcorner s \urcorner\ \$'\ \ulcorner t \urcorner$ without any level assumption.

ATOM^* constructors represent the places where $f\,x$ depends on $x$. That is, $\text{-}_\star$ is bijective on full terms; its inverse shall be denoted $\mathsf{dB\_fn} :: (\beta \Rightarrow dB) \Rightarrow \beta\ dB\_fn$.

Establishing this bijection is a significant part of the Isabelle/HOL theory, and it allows functions on $expr \Rightarrow dB$ to be defined by primitive recursion, and their properties proved by induction, on $expr\ dB\_fn$[2].

Now abstr is defined by $\mathsf{abstr}\ f \equiv \mathsf{Abstr}\ f^\star$, where $f^\star = \mathsf{dB\_fn}\,(\mathsf{dB} \circ f)$ and the auxiliary predicate Abstr is defined by primitive recursion on $expr\ dB\_fn$; the essential case is $\mathsf{Abstr}\,(\mathsf{ATOM}^*\,f) = (f = \mathsf{dB})$. Note that $\mathsf{ATOM}^*\,\mathsf{dB} = (\lambda\, x.\, x)^\star$ in $f^\star$ stands in for the bound metavariable in f.

Similarly, LAM is defined by $\mathsf{LAM}\ f \equiv \mathsf{Lambda}\ f^\star$, where

$$\mathsf{Lambda}\ S \equiv \mathsf{if}\ \mathsf{Abstr}\ S\ \mathsf{then}\ \llcorner \mathsf{ABS}'\,(\mathsf{Lbind}\ 0\ S)\lrcorner\ \mathsf{else}\ \llcorner \mathsf{ERR}' \lrcorner.$$

The conditional construction serves to distinguish LAM of an abstraction from LAM of an exotic function. The function $\mathsf{Lbind} :: [\,bnd, expr\ dB\_fn\,] \Rightarrow dB$ is defined by primitive recursion:

$$\begin{aligned} \mathsf{Lbind}\ i\ (\mathsf{ATOM}^*\,f) &= \mathsf{BND}'\,i \\ \mathsf{Lbind}\ i\ (S\ \$^*\ T) &= (\mathsf{Lbind}\ i\ S)\ \$'\ (\mathsf{Lbind}\ i\ T) \\ \mathsf{Lbind}\ i\ (\mathsf{ABS}^*\,S) &= \mathsf{ABS}'\big(\mathsf{Lbind}\ (\mathsf{Suc}\ i)\ S\big) \end{aligned}$$

where $\mathsf{Lbind}\ i\ S = S_\star\ \mathsf{arbitrary}$ in the remaining cases. Note that the Abstr condition ensures that all occurrences of $\mathsf{ATOM}^*\,f$ passed to Lbind have $f = \mathsf{dB}$.

With these definitions, statements such as the following are provable:

$$\begin{aligned} &\mathsf{LAM}\,x.\,\mathsf{LAM}\,y.\,\mathsf{CON}\ c\ \$\ x\ \$\ y\ \$\ \mathsf{VAR}\ 3 = \\ &\qquad \llcorner \mathsf{ABS}'\,(\mathsf{ABS}'\,(\mathsf{CON}'\,c\ \$'\ \mathsf{BND}'\,1\ \$'\ \mathsf{BND}'\,0\ \$'\ \mathsf{VAR}'\,3))\lrcorner \end{aligned}$$

Indeed, application of dB or expr triggers simplification rules that convert between HOAS and de Bruijn form.

4 Conclusion

Materials related to Hybrid, including source code, case studies and previous papers, can be found at `http://hybrid.dsi.unimi.it/`. Ready-to-use SLs are minimal and ordered linear logic. Several case studies have been carried out, only the first three being one-level:

- Encodings and proofs of simple properties of quantified propositional formulae (conversion to normal forms), and of the higher-order π-calculus (structural congruence and reaction rules) [4].
- A Howe-style proof that applicative bisimulation in the lazy λ-calculus is a congruence [13].
- A subject reduction theorem [5] for the intermediate language MIL-lite of the MLj compiler.
- The two-level approach with Coq as the meta-language is first introduced in [9];

[2] In previous versions of Hybrid [4], a related induction principle on $dB \Rightarrow dB$, called abstraction_induct, was used directly. A generalization of it is still used in establishing the bijection. Other instances of dB_fn may be useful in generalizing abstr to functions of more than one variable.

subject reduction and uniqueness of typing of Mini-ML are re-proved and compared to the proofs in McDowell's thesis.

- In [14] we verified the correctness of a compiler for (a fragment) of Mini-ML into an environment machine. To deal with recursion more succinctly, we enriched the language with Milner & Tofte's non-well-founded closures, and checked, via coinduction, a type preservation result.
- Properties of continuation machines are investigated in [15] with an ordered linear logic as SL, e.g. internalizing the instruction stack in the ordered context.

Future work will tackle the issue of formulating SLs capable of performing induction over *open* terms, required, for example, to complete the PoplMark challenge. We also plan to add primitive recursion principles for defining functions directly on the higher-order syntax, following on [5, 7]. On the practical side, we are looking into presenting Hybrid as a *"lightweight" HOAS package*, as opposed to Urban's nominal package [18], which is more concerned with a machine assisted reconstruction of the informal "Barendregt" style of mathematical reasoning in presence of binders. Our package would include some facilities that would automatically turn a user signature into appropriate Hybrid-based Isabelle/HOL theories, in the spirit of OTT [3]. The aim is to aid the user's focus on the problem at hand by further separating him from the machinery of defining an OL, such as CON instantiation, simplifier setup and customization of the tactics we have discussed earlier.

Acknowledgement

Felty and Martin acknowledge the support of the Natural Sciences and Engineering Research Council of Canada and the University of Ottawa. Momigliano is partially supported by the European Project Mobius within the frame of IST 6th Framework.

References

[1] *The Coq proof assistant, v.8.0*, `http://coq.inria.fr/`.

[2] *Isabelle/Isar 2005*, `http://isabelle.in.tum.de/Isar`.

[3] *Ott*, `http://www.cl.cam.ac.uk/~pes20/ott/`.

[4] Ambler, S., R. Crole and A. Momigliano, *Combining higher order abstract syntax with tactical theorem proving and (co)induction*, in: *Fifteenth International Conference on Theorem Proving in Higher-Order Logics*, Springer-Verlag LNCS **2342**, 2002, pp. 13–30.

[5] Ambler, S. J., R. L. Crole and A. Momigliano, *A definitional approach to primitive recursion over higher order abstract syntax*, in: *ACM SIGPLAN Workshop on Mechanized Reasoning about Languages with Variable Binding* (2003), pp. 1–11.

[6] Aydemir, B. E. et al., *Mechanized metatheory for the masses: the* PoplMark *challenge*, in: *Eighteenth International Conference on Theorem Proving in Higher-Order Logics*, Springer-Verlag LNCS **3605**, 2005, pp. 50–65.

[7] Capretta, V. and A. Felty, *Combining de Bruijn indices and higher-order abstract syntax in Coq*, in: *Proceedings of TYPES 2006*, Springer-Verlag LNCS **4502**, 2007, pp. 63–77.

[8] de Bruijn, N. G., *Lambda-calculus notation with nameless dummies: a tool for automatic formula manipulation with application to the Church-Rosser theorem*, Indagationes Mathematicæ **34** (1972), pp. 381–392.

[9] Felty, A., *Two-level meta-reasoning in Coq*, in: *Fifteenth International Conference on Theorem Proving in Higher-Order Logics*, Springer-Verlag LNCS **2342**, 2002, pp. 198–213.

[10] Gordon, A., *A mechanisation of name-carrying syntax up to α-conversion*, in: *Higher Order Logic Theorem Proving and its Applications*, Springer-Verlag LNCS **780**, 1993, pp. 414–426.

[11] Honsell, F., M. Miculan and I. Scagnetto, *An axiomatic approach to metareasoning on nominal algebras in HOAS*, in: *28th International Colloquium on Automata, Languages and Programming*, Springer-Verlag LNCS **2076**, 2001, pp. 963–978.

[12] McDowell, R. and D. Miller, *Reasoning with higher-order abstract syntax in a logical framework*, ACM Transactions on Computational Logic **3** (2002), pp. 80–136.

[13] Momigliano, A., S. Ambler and R. Crole, *A Hybrid encoding of Howe's method for establishing congruence of bisimilarity*, Electronic Notes in Theoretical Computer Science **70** (2002), pp. 60–75.

[14] Momigliano, A. and S. J. Ambler, *Multi-level meta-reasoning with higher order abstract syntax*, in: *Sixth International Conference on Foundations of Software Science and Computational Structures*, Springer-Verlag LNCS **2620**, 2003, pp. 375–391.

[15] Momigliano, A. and J. Polakow, *A formalization of an ordered logical framework in Hybrid with applications to continuation machines*, in: *ACM SIGPLAN Workshop on Mechanized Reasoning about Languages with Variable Binding* (2003), pp. 1–9.

[16] Pfenning, F. and C. Schürmann, *System description: Twelf — a meta-logical framework for deductive systems*, in: *Sixteenth International Conference on Automated Deduction*, Springer-Verlag LNCS **1632**, 1999, pp. 202–206.

[17] Tiu, A., "A Logical Framework for Reasoning about Logical Specifications," Ph.D. thesis, Pennsylvania State University (2004).

[18] Urban, C. and C. Tasson, *Nominal techniques in Isabelle/HOL*, in: *Twentieth International Conference on Automated Deduction*, Springer-Verlag LNCS **3632**, 2005, pp. 38–53.