Hybrid:A Definitional Two-Level Approach to Reasoning with Higher-Order Abstract Syntax*

Amy Felty
School of Information Technology and Engineering, University of Ottawa, Canada
afelty@site.uottawa.ca

Alberto Momigliano
Laboratory for the Foundations of Computer Science, University of Edinburgh, Scotland
amomigl1@inf.ed.ac.uk

September 19, 2008

Abstract

Combining higher-order abstract syntax and (co)-induction in a logical framework is well known to be problematic. Previous work [3] described the implementation of a tool called Hybrid, within Isabelle HOL, which aims to address many of these difficulties. It allows object logics to be represented using higher-order abstract syntax, and reasoned about using tactical theorem proving and principles of (co)induction. Moreover, it is definitional, which guarantees consistency within a classical type theory. The idea is to have a de Bruijn representation of λ-terms providing a definitional layer that allows the user to represent object languages using higher-order abstract syntax, while offering tools for reasoning about them at the higher level. In this paper we describe how to use it in a multi-level reasoning fashion, similar in spirit to other meta-logics such as *Linc* and *Twelf*. By explicitly referencing provability in a middle layer called a specification logic, we solve the problem of reasoning by (co)induction in the presence of non-stratifiable hypothetical judgments, which allow very elegant and succinct specifications of object logic inference rules. We first demonstrate the method on a simple example, formally proving type soundness (subject reduction) for a fragment of a pure functional language, using a minimal intuitionistic logic as the specification logic. We then prove an analogous result for a continuation-machine presentation of the operational semantics of the same language, encoded this time in an ordered linear logic that serves as the specification layer. This example demonstrates the ease with which we can incorporate new specification logics, and also illustrates a significantly more complex object logic whose encoding is elegantly expressed using features of the new specification logic.

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

1 Introduction

Logical frameworks provide general languages in which it is possible to represent a wide variety of logics, programming languages, and other formal systems. They are designed to capture uniformities of the deductive systems of these object logics and to provide support for implementing and reasoning about them. One application of particular interest of such frameworks is the specification of programming languages and the formalization of their semantics in view of formal reasoning about important properties of these languages, such as their soundness. Programming languages that enjoy such properties provide a solid basis for building software systems that avoid a variety of harmful defects, leading to software systems that are significantly more reliable and trustworthy.

*Felty was supported in part by the Natural Sciences and Engineering Research Council of Canada Discovery program. Momigliano was supported by EPSRC grant GR/M98555 and partially by the MRG project (IST-2001-33149), funded by the EC under the FET proactive initiative on Global Computing

The mechanism by which object-logics are represented in a logical framework has a paramount importance on the success of a formalization. A naive choice of representation can seriously endanger a project almost from the start, making it almost impossible to move beyond the very first step of the developments of a case study (see [64] which barely goes beyond encoding the syntax of the π-calculus).

Higher-Order Abstract Syntax (HOAS) is a representation technique used in some logical frameworks. Using HOAS, whose idea dates back to Church [21], binding constructs in an object logic are encoded within the function space provided by a meta-language based on a λ-calculus. For example, consider encoding a simple functional programming language such as Mini-ML [23] in a typed meta-language, where object-level programs are represented as meta-level terms of type *expr*. We can introduce a constant fun of type $(\underline{expr} \to expr) \to expr$ to represent functions of one argument. Using such a representation allows us to delegate to the meta-language α-conversion and capture-avoiding substitution. Further, object logic substitution can be rendered as meta-level β-conversion. However, experiments such as the one reported in [72] suggest that the full benefits of HOAS can be enjoyed only when the latter is paired with support for hypothetical and parametric judgments [47, 59, 84]. Such judgments are used, for example, in the well-known encoding of inference rules assigning simple types to Mini-ML programs. Both the encoding of programs and the encoding of the typing predicate typically contain *negative* occurrences of the type or predicate being defined (*e.g.*, the underlined occurrence of *expr* in the type of fun above). Set-theoretically, these definitions do not yield monotone operators and cannot be constructed as a least fixed point [45, 82]. Type-theoretically, they infringe on the *strict positivity* requirement [81] used to obtain strong normalization, and thus are not inductive in any ordinary sense. As much as HOAS sounds appealing, it raises the question(s): how are we going to reason about such encodings, in particular are there induction and case analysis principles available?

One solution that has emerged in the last decade is that *specification* and (inductive) *meta-reasoning* should be handled (ideally) within a single system but at different *levels*. The first example of such a meta-logic was $FO\lambda^{\Delta I\!N}$ [62], soon to be followed by its successor, *Linc* [105].[1] They are both based on intuitionistic logic augmented with introduction and elimination rules for *defined* atoms (partial inductive definitions, PIDs [46]), in particular *definitional reflection* (*defL*), which provides support for case analysis. While $FO\lambda^{\Delta I\!N}$ has only induction on natural numbers as the primitive form of inductive reasoning, the latter generalizes that to standard forms of induction and coinduction [75]; *Linc* also introduces the so-called "nabla" quantifier ∇ [69] to deal with parametric judgments.This quantifier accounts for the dual properties of eigenvariables, such as *freshness* (when viewed as constants introduced by the quantifier right rule) and *instantiability* as a consequence of an elimination rule and case analysis. How to incorporate such a quantifier in our architecture is a matter of future work. (See Sections 6.1 and 7.) Consistency and viability of proof search are ensured by cut-elimination [61, 105]. Inside the meta-language, a *specification logic* (SL) is developed that is in turn used to specify and (inductively) reason about the *object logic/language* (OL) under study. This partition solves the problem of meta-reasoning in the presence of negative occurrences in OL judgments, since hypothetical judgments are now encapsulated within the SL and therefore not required to be inductive. The price to pay is coping with this additional layer where we explicitly reference provability. Were we to work with only a bare proof-checker, this price could be indeed deemed too high; however, if we could rely on some form of automation such as tactical theorem proving, the picture would be significantly different.

The first author has proposed in [33] that, rather than implementing an interactive theorem prover for such meta-logics from scratch, they can be simulated within a modern proof assistant. (Coq [12] was used in that case.) The correspondence is roughly as follows: the ambient logic of the proof assistant in place of the basic (logical) inference rules of $FO\lambda^{\Delta I\!N}$, introduction and elimination (inversion) rules of inductive types (definitions) in place of the *defR* and *defL* rules of PIDs.[2] Both approaches introduce a minimal sequent calculus [55] as an SL, and a Prolog-like set of clauses for the OL. Nevertheless, in Coq this is not quite enough, as reasoning by inversion crucially depends on simplifying in the presence of constructors. When such constructors are non-inductive, which is typically the case with variable-binding operators, this presents a serious problem. The solution used in that work was essentially axiomatic;

[1] This is by no way the end of the story; on the contrary, the development of these ambient logics is very much a work in progress: Tiu [106] introduced the system LG^{Ω} to get rid of the local signatures required by ∇. Even more recently Gacek, Miller & Nadathur presented the logic $\mathscr{G}$ to ease reasoning on open terms and implemented it in the *Abella system* [38–40]. However, as this overdue report of our approach describes with an undeniable tardiness a system that was developed before the aforementioned new contributions, we will take the liberty to refer to *Linc* as the "canonical" two-level system. We will discuss the aforementioned new developments in more depth in Section 6.1.

[2] The *defL* rule for PIDs may use full higher-order unification, while inversion in an inductive proof assistant typically generates equations that may or may not be further simplified, especially at higher types.

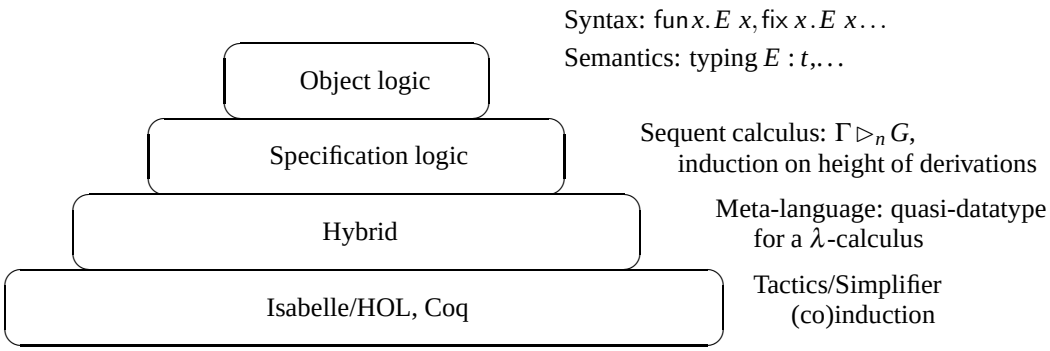

Figure 1: Architecture of the Hybrid system

introduce a set of constants for an HOAS signature in place of constructors of what could not be an inductive type, and add a set of axioms stating the freeness and extensionality properties of the constants. With the critical use of those axioms, it was shown that it was possible to replicate, in the well-understood and interactive setting of Coq, the style of proofs typical of $FO\lambda^{\Delta IN}$. In particular, subject reduction for Mini-ML is formalized in [33] following this style very closely; this means that the theorem is proved immediately without any "technical" lemmas required by the choice of encoding technique or results that may be trivial but are intrinsically foreign to the mathematics of the problem. Moreover, HOAS proofs of subject reduction typically do not require weakening or substitutions lemmas, as they are implicit in the higher-order nature of the encoding.

The Hybrid tool [3] was developed around the same time, and implements a higher-order meta-language within Isabelle/HOL [78] that provides a form of HOAS for the user to represent OLs. The user level is separated from the infrastructure, in which HOAS is implemented *definitionally* via a de Bruijn style encoding. Lemmas stating properties such as freeness and extensionality of constructors are *proved* and no additional axioms are required.

In this paper, we combine the HOAS meta-language provided by Hybrid with Miller & McDowell's two-level approach, modified for inductive proof assistants. We implement this combined architecture in both Isabelle/HOL and Coq, but we speculate that the approach also works for other tactic-based inductive proof assistants, such as PVS, LEGO *etc*. We describe mainly the Isabelle/HOL version here, though we compare it in some detail with the Coq implementation.[3] A graphical depiction of the architecture is shown in Figure 1. We often refer to the Hybrid and Isabelle/HOL levels together as the meta-logic. When we need to distinguish the Isabelle/HOL level on its own, we call it the meta-meta-logic. When we say *two-level* reasoning, we are referring to the object and specification levels, to emphasize that there are two separate reasoning levels in addition to the meta-level.

Moreover, we suggest a further departure in design (Section 4.4) from the original two-level approach [62]: when possible, *i.e.*, when the structural properties of the meta-logic are coherent with the style of encoding of the OL, we may reserve for the specification level only those judgments that cannot be *adequately* encoded inductively and leave the rest at the Isabelle/HOL level. We claim that this framework with or without this variation has several advantages:

- The system is more trustworthy: freeness of constructors and, more importantly, extensionality properties at higher types are not assumed, but proved via the related properties of the infrastructure, as we show in Section 3 (MC-Theorem 9).
- The mixing of meta-level and specification-level judgments makes proofs more easily mechanizable and more generally, there is a fruitful interaction between (co)-induction principles, meta-logic datatypes, classical reasoning, and hypothetical judgments, which lends itself to a good deal of automation.
- We are not committed to a single monolithic SL, but we may adopt different ones (linear, relevant, bunched...) according to the properties of the OL we are encoding. The only requirement is consistency, to be established with a formalized cut-elimination argument. We exemplify this methodology using non-commutative linear logic to reason about continuation machines (Section 5).

[3] We also compare it with a constructive version implemented in Coq [16], which we describe in Section 6.5.

Our architecture could also be seen as an approximation of *Twelf* [86], but it has a much lower mathematical overhead, simply consisting of a small set of theories (modules) on top of a proof assistant. In a sense, we could look at Hybrid as a way to "represent" Twelf's meta-proofs into the well-understood setting of higher-order logic as implemented by Isabelle/HOL (or the calculus of inductive constructions as implemented by Coq).

Additionally, 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; in particular we can do meta-reasoning in the style of tactical theorem proving in a way compatible with induction. For example, as we will see in Section 5, when experimenting with a new type theory, we do not need to develop all the building blocks of a usable new framework, such as unification algorithms, type inference or proof search, but we rely on the ones provided by the proof assistant. The price to pay is, again, the additional layer where we explicitly reference provability, requiring a sort of meta-interpreter (the SL logic) to drive it. This indirectness can be alleviated, as we shall see, by defining appropriate tactics, but this is intrinsic to the design choice of relying on a general meta-meta-logic (here Isabelle/HOL or Coq, in [62, 105] some variation of *Linc*). This contrasts with the architecture proposed in [60], where the meta-meta-logic is itself sub-structural (linear in this case) and, as such, explicitly tailored to the automation of a specific framework.

We demonstrate the methodology by first formally verifying the subject reduction property for the standard simply-typed call-by-value λ-calculus, enriched with a recursion operator. While this property (and the calculus as well) has been cited as too trivial to be meaningful [5]—and, to a degree, we may agree with that—we feel that the familiarity of the set-up will ease the understanding of the several layers of our architecture. Secondly we tackle a more complex form of subject reduction, that of a continuation machine, whose operational semantics is encoded sub-structurally.

Outline The paper is organized as follows: Section 2 recalls some basic notions of Hybrid and its implementation in Isabelle/HOL and Coq. Section 3 shows how it can be used as a logical framework. In Section 4 we introduce a two-level architecture and present the first example SL and subject reduction proof, while Section 5 introduces a sub-structural SL and uses it for encoding continuation machines. We follow that up with an extensive review and comparison of related work in Section 6, and conclude in Section 7.

Notation 1 (Isabelle/HOL) We use a pretty-printed version of Isabelle/HOL concrete syntax. A type declaration has the form $s :: [t_1, \ldots t_n] \Rightarrow t$. We stick to the usual logical symbols for Isabelle/HOL connectives and quantifiers ($\neg$, $\wedge$, $\vee$, $\longrightarrow$, $\forall$, $\exists$). Free variables (upper-case) are implicitly universally quantified (from the outside) as in logic programming. The sign $==$ (Isabelle meta-equality) is used for *equality by definition*, $\bigwedge$ for Isabelle universal meta-quantification. A rule (a sequent) of the schematic form

$$\frac{H_1 \ldots H_n}{C}$$

is represented as $[\![H_1; \ldots; H_n]\!] \Longrightarrow C$. A rule with discharged assumptions such as conjunction elimination is represented as $[\![P \wedge Q; [\![P; Q]\!] \Longrightarrow R]\!] \Longrightarrow R$. The keyword **MC-Theorem (Lemma)** denotes a machine-checked theorem (lemma), while *Inductive* introduces an inductive relation in Isabelle/HOL, and *datatype* introduces a new datatype. We freely use infix notations, without explicit declarations. We have tried to use the same notation for mathematical and formalized judgments. The proof scripts underlying this paper are written in the so-called "Isabelle old style", *i.e.* they are exclusively in the by now deprecated tactical-style, *e.g.* sequences of commands. This was still fashionable and supported by Isabelle/HOL 2005, as opposed to the now required Isar [54] idioms of the new Isabelle/HOL versions. However, in the interest of time, intellectual honesty (and also consistency with Coq), we have decided to base the paper on the original code of the project, which had as a fundamental goal the *automation* of two-level reasoning. Naturally, some of the comments that we make about concrete features of the system, (as well as interactions with it) are by now relevant only to that version. When those happen to be obsolete, we will try to make this clear to the reader. We expect, however (and indeed we already are in the process, see Section 6.5) to carry over this work to the current version of Isabelle/HOL, possibly enhanced by the new features of the system.

Notation 2 (Coq) We keep Coq's notation similar to Isabelle/HOL's where possible. We use the same syntax for type declarations, though of course the allowable types are different in the two languages. We also use $==$ for equality by

definition and $=$ for equality. There is no distinction between a functional type arrow and logical implication in Coq, though we use both $\Rightarrow$ and $\Longrightarrow$ depending on the context. In Isabelle/HOL, there is a distinction between notation at the Isabelle meta-level and the HOL object-level, which we do not have in Coq. Whenever an Isabelle/HOL formula has the form $[\![H_1; \ldots; H_n]\!] \Longrightarrow C$, and we say that the Coq version is the same, we mean that the Coq version has the form $H_1 \Longrightarrow \cdots \Longrightarrow H_n \Longrightarrow C$, or equivalently $H_1 \Rightarrow \cdots \Rightarrow H_n \Rightarrow C$, where implication is right-associative as usual.

Source files for the Isabelle/HOL and Coq code can be found at `hybrid.dsi.unimi.it/journal-paper` [53].

2 Introducing Hybrid

The description of the Hybrid layer of our architecture is taken fairly directly from previous work, *viz.* [3]. Central to our approach is the introduction of a binding operator that (1) allows a direct expression of λ-abstraction, and (2) is *defined* in such a way that expanding its definition results in the conversion of a term to its de Bruijn representation. The basic idea is inspired by the work of Gordon [43], and also appears in collaborative work with Melham [44]. Gordon introduces a λ-calculus with constants where free and bound variables are named by *strings*; in particular, in a term of the form $(\mathsf{dLAM}\,\mathtt{v}\,t)$, $\mathtt{v}$ is a string representing a variable bound in t, and dLAM is a function of two arguments, which when applied, converts free occurrences of $\mathtt{v}$ in t to the appropriate de Bruijn indices and includes an outer de Bruijn abstraction operator. Not only does this approach provide a good mechanism through which one may work with *named* bound variables under α-renaming, but it can be used as a meta-logic by building it into an Isabelle/HOL type, say of *proper terms*, from which other binding signatures can be defined, as exemplified by Gillard's encoding of the object calculus [41]. As in the logical framework tradition, every OL binding operator is reduced to the λ-abstraction provided by the type of proper terms.

Our approach takes this a step further and exploits the built in HOAS which is available in systems such as Isabelle/HOL and Coq. Hybrid's LAM constructor is similar to Gordon's dLAM except that LAM is a *binding* operator. The syntax $(\mathsf{LAM}\,v.t)$ is actually notation for $(\mathsf{lambda}\ \lambda\, v.t)$, which makes explicit the use of bound variables in the meta-language to represent bound variables in the OL. Thus the v in $(\mathsf{LAM}\,v.t)$ is a meta-variable (and not a string as in Gordon's approach).

At the base level, we start with an inductive definition of de Bruijn expressions, as Gordon does.

$$\textit{datatype expr} \quad = \quad \mathsf{CON}\ \textit{con} \mid \mathsf{VAR}\ \textit{var} \mid \mathsf{BND}\ \textit{bnd} \mid \textit{expr}\ \$\ \textit{expr} \mid \mathsf{ABS}\ \textit{expr}$$

In our setting, *bnd* and *var* are defined to be the natural numbers, and *con* provides names for constants. The latter type is used to represent the constants of an OL, as each OL introduces its own set of constants.

To illustrate the central ideas, we start with the λ-calculus as an OL. To avoid confusion with the meta-language (i.e., λ-abstraction at the level of Isabelle/HOL or Coq), we use upper case letters for variables and a capital Λ for abstraction. For example, consider the object-level term $T_0 = \Lambda V_1.(\Lambda V_2.V_1V_2)V_1V_3$. The terms T_G and T_H below illustrate how this term is represented using Gordon's approach and Hybrid, respectively.

$$\begin{array}{rcl} T_G & = & \mathsf{dLAM}\ \mathtt{v1}\ (\mathsf{dAPP}\ (\mathsf{dAPP}\ (\mathsf{dLAM}\ \mathtt{v2}\ (\mathsf{dAPP}\ (\mathsf{dVAR}\ \mathtt{v1}) \\ & & \qquad (\mathsf{dVAR}\ \mathtt{v2})))\ (\mathsf{dVAR}\ \mathtt{v1}))\ (\mathsf{dVAR}\ \mathtt{v3})) \\ T_H & = & \mathsf{LAM}\ v_1.(((\mathsf{LAM}\ v_2.(v_1\ \$\ v_2))\ \$\ v_1)\ \$\ \mathsf{VAR}\ \ 3) \end{array}$$

In Hybrid we also choose to denote object-level free variables by terms of the form $(\mathsf{VAR}\ i)$, though this is not essential. In either case, the abstraction operator (dLAM or LAM) is defined, and expanding definitions in both T_G and T_H results in the same term, shown below using our de Bruijn notation.

$$\mathsf{ABS}\ (((\mathsf{ABS}\ (\underline{\mathsf{BND}\ 1}\ \$\ \mathsf{BND}\ 0))\ \$\ \underline{\mathsf{BND}\ 0})\ \$\ \mathsf{VAR}\ \ 3)$$

In the above term all the variable occurrences bound by the first ABS, which corresponds to the bound variable V_1 in the object-level term, are underlined. The lambda operator is central to this approach and its definition includes determining correct indices. We return to its definition in Section 2.1.

In summary, Hybrid provides a form of HOAS where object-level:

- free variables correspond to Hybrid expressions of the form $(\mathsf{VAR}\ i)$;
- bound variables correspond to (bound) meta-variables;
- abstractions $\Lambda V.E$ correspond to expressions $(\mathsf{LAM}\, v.\, e)$, defined as $(\mathsf{lambda}\ \lambda\, v.\, e)$;
- applications $E_1\ E_2$ correspond to expressions $(e_1\ \$\ e_2)$.

2.1 Definition of Hybrid in Isabelle/HOL

Hybrid consists of a small number of Isabelle/HOL theories (actually two for a total of about 130 lines of definitions and 80 lemmas and theorems), which introduce the basic definition for de Bruijn expressions (*expr*) given above and provide operations and lemmas on them, building up to those that hide the details of de Bruijn syntax and permit reasoning on HOAS representations of OLs. In this section we outline the remaining definitions, and give some examples. Note that our Isabelle/HOL theories do not contain any axioms which require external justification,[4] as in some other approaches such as the Theory of Contexts [51].

As mentioned, the operator $\mathsf{lambda} :: [\mathit{expr} \Rightarrow \mathit{expr}] \Rightarrow \mathit{expr}$ is central to our approach, and we begin by considering what is required to fill in its definition. Clearly $(\mathsf{lambda}\ e)$ must expand to a term with ABS at the head. Furthermore, we must define a function f such that $(\mathsf{lambda}\ e)$ is $(\mathsf{ABS}\ (f\ e))$ where f replaces occurrences of the bound variable in e with de Bruijn index 0, taking care to increment the index as it descends through inner abstractions. In particular, we will define a function lbind of two arguments such that formally:

$$\mathsf{lambda}\ e == \mathsf{ABS}\ (\mathsf{lbind}\ 0\ e)$$

and $(\mathsf{lbind}\ i\ e)$ replaces occurrences of the bound variable in e with de Bruijn index i, where recursive calls on inner abstractions will increase the index. As an example, consider the function $\lambda\, v.\, \mathsf{ABS}\ (\mathsf{BND}\ 0\ \$\ v)$. In this case, application of lbind with argument index 0 should result in a level 1 expression:

$$\mathsf{lbind}\ 0\ (\lambda\, v.\, \mathsf{ABS}\ (\mathsf{BND}\ 0\ \$\ v)) = \ldots = \mathsf{ABS}\ (\mathsf{BND}\ 0\ \$\ \mathsf{BND}\ 1)$$

and thus:

$$\mathsf{lambda}\ (\lambda\, v.\, \mathsf{ABS}\ (\mathsf{BND}\ 0\ \$\ v)) = \mathsf{ABS}\ (\mathsf{ABS}\ (\mathsf{BND}\ 0\ \$\ \mathsf{BND}\ 1)).$$

We define lbind as a total function operating on all functions of type $(\mathit{expr} \Rightarrow \mathit{expr})$, even *exotic* ones that do not encode λ-terms. For example, we could have $e = (\lambda x.\mathsf{count}\ x)$ where $(\mathsf{count}\ x)$ counts the total number of variables and constants occurring in x. Only functions that behave *uniformly* or *parametrically* on their arguments represent λ-terms. We refer the reader to the careful analysis of this phenomenon (in the context of Coq) given in [29] and to Section 6 for more background. We will return to this idea shortly and discuss how to rule out non-uniform functions in our setting. For now, we define lbind so that it maps non-uniform subterms to a default value. The subterms we target are those that do not satisfy the predicate $\mathsf{ordinary} :: [\mathit{expr} \Rightarrow \mathit{expr}] \Rightarrow \mathit{bool}$, defined as follows:

$$\begin{array}{lcl}
\mathsf{ordinary}\ e & == & (\exists a.\ e = (\lambda\, v.\, \mathsf{CON}\ a) \vee \\
& & e = (\lambda\, v.\, v) \vee \\
& & \exists n.\ e = (\lambda\, v.\, \mathsf{VAR}\ n) \vee \\
& & \exists j.\ e = (\lambda\, v.\, \mathsf{BND}\ j) \vee \\
& & \exists f\ g.\ e = (\lambda\, v.\ f\, v\ \$\ g\, v) \vee \\
& & \exists f.\ e = (\lambda\, v.\, \mathsf{ABS}\ (f\, v)))
\end{array}$$

(This definition is one of the points where the Isabelle/HOL and Coq implementations of Hybrid diverge. See Section 2.2.)

We do not define lbind directly, but instead define a relation $\mathsf{lbnd} :: [\mathit{bnd}, \mathit{expr} \Rightarrow \mathit{expr}, \mathit{expr}] \Rightarrow \mathit{bool}$ and prove that this relation defines a function mapping the first two arguments to the third.

[4] We will keep emphasizing this point: the package is a definitional extension of Isabelle/HOL and could be brought back to HOL primitives, if one so wishes.

$$
\begin{array}{rcl}
\textit{Inductive}\ \mathsf{lbnd} & :: & [\mathit{bnd}, \mathit{expr} \Rightarrow \mathit{expr}, \mathit{expr}] \Rightarrow \mathit{bool} \\
 & \Longrightarrow & \mathsf{lbnd}\ i\ (\lambda\, v.\, \mathsf{CON}\ a)\ (\mathsf{CON}\ a) \\
 & \Longrightarrow & \mathsf{lbnd}\ i\ (\lambda\, v.\, v)\ (\mathsf{BND}\ i) \\
 & \Longrightarrow & \mathsf{lbnd}\ i\ (\lambda\, v.\, \mathsf{VAR}\ n)\ (\mathsf{VAR}\ n) \\
 & \Longrightarrow & \mathsf{lbnd}\ i\ (\lambda\, v.\, \mathsf{BND}\ j)\ (\mathsf{BND}\ j) \\
[\![\ \mathsf{lbnd}\ i\ f\ s; \mathsf{lbnd}\ i\ g\ t\ ]\!] & \Longrightarrow & \mathsf{lbnd}\ i\ (\lambda\, v.\, f\, v\ \$\ g\, v)\ (s\ \$\ t) \\
\mathsf{lbnd}\ (\mathsf{Suc}\ i)\ f\ s & \Longrightarrow & \mathsf{lbnd}\ i\ (\lambda\, v.\, \mathsf{ABS}\ (f\, v))\ (\mathsf{ABS}\ s) \\
\neg(\mathsf{ordinary}\ e) & \Longrightarrow & \mathsf{lbnd}\ i\ e\ (\mathsf{BND}\ 0)
\end{array}
$$

In showing that this relation is a function, uniqueness is an easy structural induction. Existence is proved using the following abstraction induction principle.

MC-Theorem 1 (abstraction_induct)

$$
\begin{array}{l}
[\![\ \bigwedge a.\, P\ (\lambda\, v.\, \mathsf{CON}\ a); P\ (\lambda\, v.\, v); \bigwedge n.\, P\ (\lambda\, v.\, \mathsf{VAR}\ n); \bigwedge j.\, P\ (\lambda\, v.\, \mathsf{BND}\ j); \\
\ \bigwedge f\ g.\, [\![\ P\ f; P\ g\ ]\!] \Longrightarrow P\ (\lambda\, v.\, f\, v\ \$\ g\, v); \\
\ \bigwedge f.\, [\![\ P\ f\ ]\!] \Longrightarrow P\ (\lambda\, v.\, \mathsf{ABS}\ (f\, v)); \\
\ \bigwedge f.\, [\![\ \neg\mathsf{ordinary}\ f\ ]\!] \Longrightarrow P\ f\ ]\!] \Longrightarrow P\ e
\end{array}
$$

The proof of this induction principle is by measure induction ($\bigwedge x.\, [\![\ \forall y.\, [\![\ f\ y < f\ x \longrightarrow P\ y\ ]\!] \Longrightarrow P\ x\ ]\!] \Longrightarrow P\ a$), where we instantiate f with rank and set $\mathsf{rank}\, e == \mathsf{size}\,(e\,(\mathsf{VAR}\ 0))$.

We now define $\mathsf{lbind} :: [\mathit{bnd}, \mathit{expr} \Rightarrow \mathit{expr}] \Rightarrow \mathit{expr}$ as follows, thus completing the definition of lambda:

$$\mathsf{lbind}\ i\ e == \mathit{THE}\, s.\, \mathsf{lbnd}\ i\ e\ s$$

where *THE* is the Isabelle's notation for the definite description operator ι. From these definitions, it is easy to prove a "rewrite rule" for every de Bruijn constructor. For example, the rule for ABS is:

MC-Lemma 2 (lbind_ABS)

$$\mathsf{lbind}\ i\ (\lambda\, v.\, \mathsf{ABS}\ (e\, v)) \quad = \quad \mathsf{ABS}\ (\mathsf{lbind}\ (\mathsf{Suc}\ i)\ e)$$

These rules are collected under the name lbind_simps, and thus can be used directly in simplification.

Ruling out non-uniform functions, which was mentioned before, will turn out to be important for a variety of reasons. For example, it is necessary for proving that our encoding adequately represents the λ-calculus. To prove adequacy, we identify a subset of the terms of type *expr* such that there is a bijection between this subset and the λ-terms that we are encoding. There are two aspects we must consider in defining a predicate to identify this subset. First, recall that BND i corresponds to a bound variable in the λ-calculus, and VAR i to a free variable; we refer to *bound* and *free indices* respectively. We call a bound index i *dangling* if i or less ABS labels occur between the index i and the root of the expression tree. We must rule out terms with dangling indices. Second, in the presence of the LAM constructor, we may have functions of type $(\mathit{expr} \Rightarrow \mathit{expr})$ that do not behave uniformly on their arguments. We must rule out such functions. We define a predicate proper, which rules out dangling indices from terms of type *expr*, and a predicate abstr, which rules out dangling indices and exotic terms in functions of type $(\mathit{expr} \Rightarrow \mathit{expr})$.

To define proper we first define level. Expression e is said to be at *level* $l \geq 1$, if enclosing e inside l ABS nodes ensures that the resulting expression has no dangling indices.

$$
\begin{array}{rcl}
\textit{Inductive}\ \mathsf{level} & :: & [\mathit{bnd}, \mathit{expr}] \Rightarrow \mathit{bool} \\
 & \Longrightarrow & \mathsf{level}\, i\, (\mathsf{CON}\ a) \\
 & \Longrightarrow & \mathsf{level}\, i\, (\mathsf{VAR}\ n) \\
j < i & \Longrightarrow & \mathsf{level}\, i\, (\mathsf{BND}\ j) \\
[\![\ \mathsf{level}\, i\, s; \mathsf{level}\, i\, t\ ]\!] & \Longrightarrow & \mathsf{level}\, i\, (s\ \$\ t) \\
\mathsf{level}\, (\mathsf{Suc}\ i)\, s & \Longrightarrow & \mathsf{level}\, i\, (\mathsf{ABS}\ s)
\end{array}
$$

Then, $\mathsf{proper} :: \mathit{expr} \Rightarrow \mathit{bool}$ is defined simply as:

$$\mathsf{proper}\ e == \mathsf{level}\,0\,e.$$

To define abstr, we first define $\mathsf{abst} :: [\mathit{bnd}, \mathit{expr} \Rightarrow \mathit{expr}] \Rightarrow \mathit{bool}$ as follows:

$$\begin{array}{rcl}
\mathit{Inductive}\ \mathsf{abst} & :: & [\mathit{bnd}, \mathit{expr} \Rightarrow \mathit{expr}] \Rightarrow \mathit{bool} \\
 & \Longrightarrow & \mathsf{abst}\ i\ (\lambda\, v.\, \mathsf{CON}\ a) \\
 & \Longrightarrow & \mathsf{abst}\ i\ (\lambda\, v.\, v) \\
 & \Longrightarrow & \mathsf{abst}\ i\ (\lambda\, v.\, \mathsf{VAR}\ n) \\
j < i & \Longrightarrow & \mathsf{abst}\ i\ (\lambda\, v.\, \mathsf{BND}\ j) \\
[\![\ \mathsf{abst}\ i\ f; \mathsf{abst}\ i\ g\ ]\!] & \Longrightarrow & \mathsf{abst}\ i\ (\lambda\, v.\, f\, v\ \$\ g\, v) \\
\mathsf{abst}\ (\mathsf{Suc}\ i)\ f & \Longrightarrow & \mathsf{abst}\ i\ (\lambda\, v.\, \mathsf{ABS}\ (f\, v))
\end{array}$$

Given $\mathsf{abstr} :: [\mathit{expr} \Rightarrow \mathit{expr}] \Rightarrow \mathit{bool}$, we set:

$$\mathsf{abstr}\ e == \mathsf{abst}\ 0\ e.$$

When an expression e of type $\mathit{expr} \Rightarrow \mathit{expr}$ satisfies this predicate, we say it is an *abstraction*.[5] In addition to being important for adequacy, the notion of an abstraction is central to the formulation of induction principles at the meta-level.[6]

It's easy to prove the analogue of abst introduction rules in terms of abstr, for example:

$$\mathsf{abst}\ (\mathsf{Suc}\ 0)\ f \Longrightarrow \mathsf{abstr}\ (\lambda\, v.\, \mathsf{ABS}\ (f\, v))$$

A simple, yet important lemma is:

MC-Lemma 3 (proper_abst)

$$\mathsf{proper}\ t \Longrightarrow \mathsf{abstr}\ (\lambda\, v.\, t)$$

So any function is a legal abstraction if it's body is a proper expression. This strongly suggests that were we to turn the predicate proper into a *type prpr*, then any function with source type $\mathit{prpr} \Rightarrow \mathit{prpr}$ would be de facto a legal abstraction[7].

It follows directly from the inductive definition of de Bruijn expressions that the functions CON, VAR, $\$$, and ABS are injective, with disjoint images. With the introduction of abstr, we can now also prove the following fundamental theorem:

MC-Theorem 4 (abstr_lam_simp)

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

which says that lambda is injective on the set of abstractions. This follows directly from an analogous property of lbind:

MC-Lemma 5 (abst_lbind_simp_lemma)

$$[\![\ \mathsf{abst}\ i\ e;\ \mathsf{abstr}\ i\, f\ ]\!] \Longrightarrow (\mathsf{lbind}\ i\ e = \mathsf{lbind}\ i\ f) = (e = f)$$

[5] This is akin to the valid and $\mathsf{valid1}$ predicates present in weak HOAS formalizations such as [29] (discussed further in Section 6.4), although this formalization has, in our notation, the "weaker" type $(\underline{\mathit{var}} \Rightarrow \mathit{expr}) \Rightarrow \mathit{bool}$.

[6] And so much more for the purpose of this paper: it allows inversion on inductive second-order predicates, simplification in presence of higher-order functions, and, roughly said, it ensures the consistency of higher-order relations with the ambient logic.

[7] This is indeed the case as we have shown in [73] and briefly comment on at the end of Section 6.

This is proved by structural induction on the abst predicate using simplification with lbind_simps.

Finally, it is possible to perform induction over the quasi-datatype of proper terms.

MC-Theorem 6 (proper_VAR_induct)

$$\begin{array}{l}[\![\ \mathsf{proper}\ u;\\ \bigwedge a.\ P\ (\mathsf{CON}\ a);\\ \bigwedge n.\ P\ (\mathsf{VAR}\ n);\\ \bigwedge s\ t.\ [\![\ \mathsf{proper}\ s; \mathsf{proper}\ t; P\ t\ ]\!] \Longrightarrow P\ (s\ \$\ t);\\ \bigwedge e.\ [\![\ \mathsf{abstr}\ e; \forall n.\ P\ (e\ (\mathsf{VAR}\ n))\ ]\!] \Longrightarrow P\ (\mathsf{LAM}\, x.\, e\ x)\ ]\!] \Longrightarrow P\ u\end{array}$$

The proof is by induction on the size of e, and follows from the following two lemmas.

MC-Lemma 7

1. $\mathsf{level}\,(\mathsf{Suc}\ i)\,e \Longrightarrow \exists f.\ (\mathsf{lbind}\ i\ f = e) \wedge \mathsf{abst}\ i\ f$ **(level_lbind_abst)**
2. $\mathsf{proper}\ (\mathsf{ABS}\ e) \Longrightarrow \exists f.\ (\mathsf{LAM}\, x.\ f\ x = \mathsf{ABS}\ e) \wedge \mathsf{abstr}\ f$ **(proper_lambda_abstr)**

MC-Lemma 8 (abstr_size_lbind)

$$\mathsf{abstr}\ e \Longrightarrow \mathsf{size}\,(\mathsf{lbind}\ i\ e) = \mathsf{size}\,(e\,(\mathsf{VAR}\ n))$$

Note that MC-Theorem 6 does not play any active role in the two-level architecture, as induction will be performed on the derivability of judgments.

2.2 Remarks on Hybrid in Coq

In this section we comment briefly on the differences between the Isabelle/HOL and Coq implementations of Hybrid, which arise mainly from the differences in the meta-languages. Isabelle/HOL implements something like a polymorphic version of Church's higher-order (classical) logic plus facilities for axiomatic classes and local reasoning in the form of *locales* [7]. Coq implements a constructive higher-order type theory, but includes libraries for reasoning classically, which we used in order to keep the implementations as similar as possible.

Note that the definition of lbind uses Isabelle/HOL's definite description operator, which is not available in Coq. The use of this operator is the main reason for the differences in the two libraries. In Coq, we instead use the description axiom available in Coq's classical libraries:[8]

$$\begin{array}{l}\forall A\ B :: \mathit{Type}.\ \forall R :: [A,B] \Rightarrow \mathit{Prop}.\\ \quad (\forall x.\ \exists y.\ (R\ x\ y \wedge \forall y'.\ R\ x\ y' \Longrightarrow y = y')) \Longrightarrow \exists f.\ \forall x.\ R\ x\ (f\ x)\end{array}$$

with lbnd as relation R. The Coq version of Hybrid is larger than the Isabelle/HOL version, mainly due to showing uniqueness for the lbnd relation. We then eliminate the existential quantifier in the description theorem to get a function that serves as the Coq version of lbind.[9]

In more detail, if we consider the Isabelle/HOL theory just described, the operations and predicates ordinary, lbnd, level, proper, abst, and abstr are defined nearly the same as in the Isabelle/HOL version. For predicates such as level, we have a choice that we did not have in Isabelle/HOL. In Coq, *Prop* is the type of logical propositions, whereas *Set* is the type of datatypes. *Prop* and *Set* allow us to distinguish *logical* aspects from *computational* ones w.r.t. our libraries. The datatype *bool* for example, distinct from *Prop*, is defined inductively in the Coq standard library as a member of *Set*. One option in defining level is to define it as a function with target type *bool*, which evaluates via conversion to $true$ or $false$. The other is to define it as an inductive predicate (in *Prop*), and then we will need to provide proofs of level subgoals instead of reducing them to $true$. We chose the latter option, using *Prop* in the definition of level and all other predicates. This allowed us to define inductive predicates in Coq that have the same structure as the

[8] In the Coq libraries, a dependent-type version of this axiom is stated, from which the version here follows directly.

[9] Although this elimination is not always justified, it is in our case since we define the type *expr* to be a Coq *Set*.

Isabelle/HOL definitions, keeping the two versions as close as possible. For our purposes, however, the other option should have worked equally well.

For predicates ordinary, lbnd, abst, and abstr, which each have an argument of functional type, there is one further difference in the Coq definitions. Equality in Isabelle/HOL is extensional, while in Coq, it is not. Thus, it was necessary to define extensional equality on type $(expr \Rightarrow expr)$ *explicitly* and use that equality whenever it is expressed on this type, *viz.*

$$=_{ext} \quad :: \quad [expr \Rightarrow expr, expr \Rightarrow expr] \Rightarrow Prop$$

Formally, $(f =_{ext} g) == \forall x.(fx = gx)$. For example, this new equality appears in the definition of abst. In the Coq version, we first define an auxiliary predicate abst_aux defined exactly as abst in Isabelle/HOL, and then define abst as:

$$\mathsf{abst}\ i\ e == \exists e'.\ e' =_{ext} e \wedge \mathsf{abst_aux}\ i\ e'.$$

The predicate abstr has the same definition as in Isabelle/HOL, via this new version of abst. The definition of lbnd parallels the one for abst, in this case using lbnd_aux. For the ordinary predicate, we obtain the Coq version from the Isabelle/HOL definition simply by replacing $=$ with $=_{ext}$.

The proof that lbnd is a total relation is by induction on rank and the induction case uses a proof by cases on whether or not a term of type $(expr \Rightarrow expr)$ is ordinary. Note that the ordinary property is not decidable, and thus this proof requires classical reasoning, which is a second reason for using Coq's classical libraries.

Coq provides a module which helps to automate proofs using user-defined equalities that are declared as *setoids*. A setoid is a pair consisting of a type and an equivalence relation on that type. To use this module, we first show that $=_{ext}$ is reflexive, symmetric, and transitive. We then declare certain predicates as morphisms. A morphism is a predicate in which it is allowable to replace an argument by one that is equivalent according to the user-defined equality. Such replacement is possible as long as the corresponding compatibility lemma is proved. For example, we declare ordinary, lbnd, abst, and abstr as morphisms. In particular, the lemma for lbnd proves that if $(\mathsf{lbnd}\ i\ e\ t)$, then for all terms e' that are extensionally equal to e, we also have $(\mathsf{lbnd}\ i\ e'\ t)$. Setoid rewriting then allows us to replace the second argument of lbnd by extensionally equal terms, and is especially useful in the proof that every e is related to a unique t by lbnd.

As stated above, we obtain lbind by eliminating the existential quantifier in the description theorem. Once we have this function, we can define lambda as in Isabelle/HOL and prove the Coq version of the *abstr_lam_simp* theorem (MC-Theorem 4):

$$\mathsf{abstr}\ e \Longrightarrow \mathsf{abstr}\ f \Longrightarrow [(\mathsf{LAM}\, x.\, e\, x = \mathsf{LAM}\, y.\, f\, y) \longleftrightarrow (e =_{ext} f)]$$

Note the use of logical equivalence ($\longleftrightarrow$) between elements of $Prop$. Extensional equality is used between elements of type $(expr \Rightarrow expr)$ and Coq equality is used between other terms whose types are in Set. Similarly, extensional equality replaces equality in other theorems involving expressions of type $(expr \Rightarrow expr)$. For example *abstraction_induct* (MC-Theorem 1) is stated as follows:

$$\begin{array}{l}
[\![\ \forall e\ a.\ [\![\ e =_{ext} (\lambda\, v.\, \mathsf{CON}\ a)\ ]\!] \Longrightarrow P\ e; \\
\quad \forall e\ [\![\ e =_{ext} (\lambda\, v.\, v)\ ]\!] \Longrightarrow P\ e; \\
\quad \forall e\ n.\ [\![\ e =_{ext} (\lambda\, v.\, \mathsf{VAR}\ n)\ ]\!] \Longrightarrow P\ e; \\
\quad \forall e\ j.\ [\![\ e =_{ext} (\lambda\, v.\, \mathsf{BND}\ j)\ ]\!] \Longrightarrow P\ e; \\
\quad \forall e\ f\ g.\ [\![\ e =_{ext} (\lambda\, v.\, f\, v\ \$\ g\, v); P\ f; P\ g\ ]\!] \Longrightarrow P\ e; \\
\quad \forall e\ f.\ [\![\ e =_{ext} (\lambda\, v.\, \mathsf{ABS}\ (f\, v)); P\ f\ ]\!] \Longrightarrow P\ e; \\
\quad \forall e.\ [\![\ \neg \mathsf{ordinary}\ e\ ]\!] \Longrightarrow P\ e\ ]\!] \Longrightarrow P\ e
\end{array}$$

3 Hybrid as a Logical Framework

In this section we show how to use Hybrid as a logical framework, first by introducing our first OL (Section 3.1), discussing the adequacy of the encoding of its syntax (Section 3.2), and then representing object-level judgments (Section 3.3).

The system at this level provides:

- A suite of theorems: roughly three or four dozens propositions, most of which are only intermediate lemmas leading to the few that are relevant to our present purpose: namely, injectivity and distinctness properties of Hybrid constants.
- Definitions proper and abstr, which are important for Hybrid's adequate representation of OLs.
- A very small number of automatic tactics: for example proper_tac (resp. abstr_tac) automatically recognizes whether a given term is indeed proper (resp. an abstraction).

We report here the (slightly simplified) code for abstr_tac, to give an idea of how lightweight such tactics are:

```
fun abstr_tac defs =
       simp_tac (simpset()
                addsimps defs @ [abstr_def,lambda_def] @ lbind_simps)
        THEN'
        fast_tac(claset()
                addDs [abst_level_lbind]
                addIs abstSet.intrs
                addEs [abstr_abst, proper_abst]);
```

This is an outermost basic tactic: first the goal is simplified (`simp_tac`) using the definition of abstr, lambda, other user-provided lemmas (`defs`), and more importantly the lbind "rewrite rules" (lbind_simps). At this point, it's merely a question of resolution with the introduction rules for abst (`abstSet.intrs`) and a few key lemmas, such as MC-Lemma 3, possibly as elimination rules. In Isabelle/HOL 2005, a tactic, even a user defined one, could also be "packaged" into a *solver*. In this way, it can be combined with the other automatic tools, such as the simplifier or user defined tactics, *viz.* 2lprolog_tac. (See Section 4.3.)

3.1 Coding the Syntax of an OL in Hybrid

The OL we consider here is a fragment of a pure functional language known as Mini-ML. As mentioned, we concentrate on a λ-calculus augmented with a fixed point operator, although it could be easily generalized as in [85]. 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:

$$\begin{array}{rrcl} \textit{Types} & \tau & ::= & i \mid \tau \to \tau' \\ \textit{Terms} & e & ::= & x \mid \mathbf{fun}\, x.\, e \mid e \bullet e' \mid \mathbf{fix}\, x.\, e \end{array}$$

We begin by showing how to represent the syntax in HOAS format using Hybrid. Since types for this language have no bindings, they are represented with a standard datatype, named *tp* and defined in the obvious way; more interestingly, as far as terms are concerned, we need constants for abstraction, application and fixed point, say *cABS*, *cAPP*, and *cFIX*. Recall that in the meta-language, application is denoted by infix \$, and abstraction by LAM.

The above grammar is coded in Hybrid verbatim, provided that we declare these constants to belong to the enumerated datatype *con*

$$\textit{datatype con} \quad = \quad \textit{cABS} \mid \textit{cAPP} \mid \textit{cFIX}$$

add the type abbreviation

$$\textit{uexp} \quad == \quad \textit{con expr}$$

and the following *definitions*:

$$\begin{array}{rcl} @ & :: & [\textit{uexp}, \textit{uexp}] \Rightarrow \textit{uexp} \\ \mathsf{fun} & :: & [\textit{uexp} \Rightarrow \textit{uexp}] \Rightarrow \textit{uexp} \\ \mathsf{fix} & :: & [\textit{uexp} \Rightarrow \textit{uexp}] \Rightarrow \textit{uexp} \\ E_1 \,@\, E_2 & == & \mathsf{CON}\ \textit{cAPP}\ \$\ E_1\ \$\ E_2 \\ \mathsf{fun}\, x.\, E\ x & == & \mathsf{CON}\ \textit{cABS}\ \$\ \mathsf{LAM}\, x.\, E\ x \\ \mathsf{fix}\, x.\, E\ x & == & \mathsf{CON}\ \textit{cFIX}\ \$\ \mathsf{LAM}\, x.\, E\ x \end{array}$$

where fun (resp. fix) is indeed an Isabelle/HOL binder, *e.g.*, $(\mathsf{fix}\, x\,.\, E\ x)$ is a syntax translation for $(\mathsf{fix}(\lambda x.\ E\ x))$. For example, the "real" underlying form of $(\mathsf{fix}\, x\,.\, \mathsf{fun}\, y.\, x\ @\ y)$ is

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

Note again that the above are only *definitions* and by themselves would not inherit any of the properties of the constructors of a datatype. However, thanks to the thin infra-structural layer that we have interposed between the λ-calculus natively offered by Isabelle and the rich logical structure provided by the axioms of Isabelle/HOL, it is now possible to *prove* the freeness properties of those definitions as if they were the constructors of what Isabelle/HOL would ordinarily consider an "impossible" datatype as discussed earlier. More formally:

MC-Theorem 9 ("Freeness" properties of constructors) *Consider the constructors*[10] $\mathsf{fun}, \mathsf{fix}, @$*:*

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

$$\mathsf{fun}\, x.\, E\ x \quad \neq \quad (E_1\ @\ E_2) \qquad\qquad \textbf{(FA_clash)}$$

- *Every non binding constructor is* injective **(App_inj)**
- *Every* binding *constructor is* injective *on* abstractions*. For example:*

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

Proof: By a call to Isabelle/HOL's standard simplification, augmented with the left-to-right direction of the crucial property *abstr_lam_simp* (MC-Theorem 4). □

This result will hold for any signature containing at most second-order constructors, provided they are encoded as we have exhibited. These "quasi-freeness" properties—meaning freeness conditionally on whether the function in a binding construct is indeed an abstraction—are added to Isabelle/HOL's standard simplifier, so that they will be automatically applied in all reasoning contexts that concern the constructors. In particular, clash theorems are best encoded in the guise of *elimination* rules, already incorporating the "ex falso quodlibet" theorem. For example, *FA_clash* of MC-Theorem 9 is equivalent to:

$$[\![\ \mathsf{fun}\, x.\, E\ x = (E_1\ @\ E_2)\ ]\!] \quad \Longrightarrow \quad P \qquad\qquad \textbf{(clash_App_Fun)}$$

3.2 Adequacy of the Encoding

It is a customary proof obligation (at least) w.r.t. higher-order encoding to show that the syntax (and later the judgments) of an OL such as Mini-ML are *adequately* represented in the framework. While this is quite well-understood in a framework such as LF, the "atypical" nature of Hybrid requires a discussion and some additional work. We take for granted (as suggested in [3], then painstakingly detailed in [25]) that Hybrid provides an adequate representation of the λ-calculus. Yet, it would not be possible to provide a "complete" proof of the adequacy of Hybrid as a theory running on a complex tool such as Isabelle/HOL. Here we take a more narrow approach, by working with a convenient fiction, *i.e.*, a *model* of Hybrid as a simply-typed λ-calculus presented as a logical framework. We can use this as our framework to represent OLs; further this model is what we consider when we state meta-theoretical properties of OL encodings and prove them adequate. We will return to this and add a further proviso before Lemma 11, when we talk about the issue of derivability in the ambient logic.

We follow quite closely Pfenning's account in the *Handbook of Automated Reasoning* [84]. By adequacy of a language representation we mean that there is an *encoding* function $\varepsilon_\Gamma(\cdot)$ from OL terms with free variables in Γ to the canonical forms of the framework in an appropriate signature, as well as its inverse $\delta_\Gamma(\cdot)$ such that:

1. validity: for every mathematical object t with free variables in Γ, $\varepsilon_\Gamma(t)$ is a canonical (and thus unique, modulo α-conversion) representation in the framework. Note that we use Γ both for the Hybrid and the OL's context;

[10] By abuse of language, we call *constructors* what are more precisely Isabelle/HOL constant definitions.

2. completeness: for every canonical term E over Γ underlying the encoding, applying the decoding $\delta_\Gamma(E)$, results in a unique OL term t; furthermore $\varepsilon_\Gamma(\delta_\Gamma(E)) = E$ and $\delta_\Gamma(\varepsilon_\Gamma(t)) = t$.

3. compositionality: this bijection commutes with substitution; formally $\varepsilon_\Gamma([t_1/x]t_2) = [\varepsilon_\Gamma(t_1)/x]\ \varepsilon_\Gamma(t_2)$ and $\delta_\Gamma([E_1/x]E_2) = [\delta_\Gamma(E_1)/x]\ \delta_\Gamma(E_2)$.

Clearly the first requirement seems easier to satisfy, while the second one tends to be more problematic.[11] In general, there are two major obstacles when trying to represent an OL's signature with some form of HOAS in a logical framework, both related to the existence of "undesirable" canonical terms in the framework, *i.e.*, honest-to-goodness terms that are *not* in the image of the desired encoding:

1. If the framework is *uni-typed*, we need predicates to express the well-formedness of the encoding of expressions of the OL. Such well-formedness properties must now be *proved*, differently from settings such as LF, where such properties are handled by type-checking. In particular, Hybrid constants are not part of a datatype, so they do not enjoy the usual closure condition. Moreover there are proper Hybrid terms such as $\mathsf{LAM}\, x.\ x\ \$\ (\mathsf{VAR}\ 0)$ that are not in the image of the encoding, but are still canonical forms of type *expr*.

2. If the framework is strong enough, in particular if its type system supports at least a primitive recursive function space, we have the problem of *exotic* terms as discussed earlier, *i.e.*, terms containing irreducible functions that are not parametric on their arguments, *e.g.*, $\mathsf{fix}\, x\,.\, \mathsf{fun}\, y.\, \mathsf{if}\ x = y\ \mathsf{then}\ x\ \mathsf{else}\ y$.

As far as the second issue is concerned, we use `abstr` annotations to get rid of such "non-parametric functions". One way to get around the first issue, although quite unsatisfactory in our view, is to weaken the completeness statement and make it *conditional*: informally this would mean that *provided* a canonical term $\Gamma \vdash E :: \mathit{uexp}$ *is* in the image of the encoding, *then* completeness holds. Instead, as prescribed by [84] and standard practice (*e.g.*, the `Vclosed` predicate in the "locally nameless" representation of [63]), we will bite the bullet and introduce well-formedness predicates (as inductive definitions in Isabelle/HOL) to represent OL types.

To make clear the correspondence between the OL and its encoding, we re-formulate the BNF grammar for Mini-ML terms as a well-formedness judgment:

$$\frac{}{\Gamma, x \vdash x} \qquad \frac{\Gamma \vdash t_1 \qquad \Gamma \vdash t_2}{\Gamma \vdash t_1 \bullet t_2} \qquad \frac{\Gamma, x \vdash t}{\Gamma \vdash \mathbf{fun}\, x.\, t} \qquad \frac{\Gamma, x \vdash t}{\Gamma \vdash \mathbf{fix}\, x.\, t}$$

Based on this formulation, the definition of encoding of a Mini-ML term into Hybrid and its decoding is unsurprising [85]. Notation-wise, we overload the comma so that Γ, x means $\Gamma \cup \{x\}$; we also use Γ for both the context of OL variables and of Hybrid variables of type *uexp*:

$$\begin{array}{rcl@{\qquad}rcl} \varepsilon_{\Gamma,x}(x) & = & x & \varepsilon_\Gamma(t_1 \bullet t_2) & = & \varepsilon_\Gamma(t_1)\ @\ \varepsilon_\Gamma(t_2) \\ \varepsilon_\Gamma(\mathbf{fun}\, x.\, t) & = & \mathsf{fun}\, x.\, \varepsilon_{\Gamma,x}(t) & \varepsilon_\Gamma(\mathbf{fix}\, x.\, t) & = & \mathsf{fix}\, x\,.\, \varepsilon_{\Gamma,x}(t) \end{array}$$

$$\begin{array}{rcl@{\qquad}rcl} \delta_{\Gamma,x}(x) & = & x & \delta_\Gamma(E_1\ @\ E_2) & = & \delta_\Gamma(E_1) \bullet \delta_\Gamma(E_2) \\ \delta_\Gamma(\mathsf{fun}\, x.\, E) & = & \mathbf{fun}\, x.\, \delta_{\Gamma,x}(E) & \delta_\Gamma(\mathsf{fix}\, x\,.\, E) & = & \mathbf{fix}\, x.\, \delta_{\Gamma,x}(E) \end{array}$$

We then introduce an inductive predicate $_ \Vdash \mathsf{isterm}\ _$ of type $[\mathit{uexp\ set}, \mathit{uexp}] \Rightarrow \mathit{bool}$, which pragmatically addresses at the same time the two aforementioned issues. It identifies the subset of *uexp* that corresponds to the open terms of Mini-ML over a set of (free) variables.

$$\begin{array}{rcl} \textit{Inductive}\ _ \Vdash \mathsf{isterm}\ _ & :: & [\mathit{uexp\ set}, \mathit{uexp}] \Rightarrow \mathit{bool} \\ [\![\, x \in \Gamma \,]\!] & \Longrightarrow & \Gamma \Vdash \mathsf{isterm}\ (x) \\ [\![\, \Gamma \Vdash \mathsf{isterm}\ E_1;\ \Gamma \Vdash \mathsf{isterm}\ E_2 \,]\!] & \Longrightarrow & \Gamma \Vdash \mathsf{isterm}\ (E_1\ @\ E_2) \\ [\![\, \forall x.\ \mathsf{proper}\ x \longrightarrow \Gamma, x \Vdash \mathsf{isterm}\ (E\ x);\ \mathsf{abstr}\ \mathsf{E} \,]\!] & \Longrightarrow & \Gamma \Vdash \mathsf{isterm}\ (\mathsf{fun}\, x.\, E\ x) \\ [\![\, \forall x.\ \mathsf{proper}\ x \longrightarrow \Gamma, x \Vdash \mathsf{isterm}\ (E\ x);\ \mathsf{abstr}\ \mathsf{E} \,]\!] & \Longrightarrow & \Gamma \Vdash \mathsf{isterm}\ (\mathsf{fix}\, x\,.\, E\ x) \end{array}$$

[11] Incidentally, some *first-order* encodings, which are traditionally assumed not to be troublesome, may fail to satisfy the second requirement in the most spectacular way. Case in point are encodings used by the Boyer-Moore theorem prover, to name some names, case studies concerning the properties of the Java Virtual Machine [58]. Since the framework's language consists of S-expressions, a decoding function doesn't really exist, but it is only informally understood how to connect a list of pairs of S-exp to an informal function in, say, the operational semantics of the JVM, assuming that the code maintains the invariants of association lists. Within Hybrid we can do better, although we will fall short of LF's standards.

We can now proceed to show validity of the encoding in the sense that $\Gamma \vdash t$ entails that $\Gamma \mid\!\models \mathsf{isterm}\ \varepsilon_\Gamma(t)$ is provable in Isabelle/HOL. However, there is an additional difficulty: the obvious inductive proof requires, in the binding case, the derivability of the following:

$$\mathsf{proper}\ x \Longrightarrow \mathsf{abstr}(\lambda x.\ \varepsilon_{\Gamma,x}(t)) \tag{1}$$

A proof by induction on the structure of t immediately leads to the problematic case:

$$\ldots \Longrightarrow \mathsf{abstr}(\lambda x.\ \mathsf{LAM}\, y.\, \varepsilon_{\Gamma,x,y}(t))$$

which has been shown to hold once $\lambda xy.\, \varepsilon_{\Gamma,x,y}(t)$ is a *biabstraction* (Lemma `biAbstr_abstr_LAM1` in theory `hybrid.dsi.unimi.it/journal-paper/BiAbstr.thy` [53]). Biabstractions are the generalization of abstractions to functions of type $(expr \Rightarrow expr \Rightarrow expr) \Rightarrow expr$. While the theory has been formally verified in Isabelle/HOL, we feel it would render the present paper too heavy, especially since the emphasis is, as we shall see, on two-level reasoning. We therefore resort to assuming (1) as a Fact and we leave a self-contained proof to future work.

We use the following notation: If $\Gamma = \{x_1, \ldots, x_n\}$, we write $\mathsf{proper}\,\Gamma$ to denote the Isabelle/HOL context $[\![\ \mathsf{proper}\ x_1; \ldots; \mathsf{proper}\ x_n\ ]\!]$.

Lemma 10 (Validity of Representation) *If $\Gamma \vdash t$, then $(\mathsf{proper}\ \Gamma \Longrightarrow \Gamma \mid\!\models \mathsf{isterm}\ \varepsilon_\Gamma(t))$ is provable in Isabelle/HOL.*

Proof: By the standard induction on the derivation of $\Gamma \vdash t$, using Fact (1) in the binding cases. □

Note that the next two lemmas do not depend on the above Fact. As far as the converse of Lemma 10 goes, we need a (hopefully) final proviso. As opposed to intentionally weak frameworks [28], Isabelle/HOL has considerable expressive power; various features of the underlying logic, such as classical reasoning, can be used to construct proofs about an OL that don't correspond to the informal constructive proofs we aim to formalize. We also need to keep in mind that a judgment J of an OL is encoded in LF as a type family and in *Linc*-like languages as a PID that expresses *rules* for introducing and eliminating the atomic object of J. In Isabelle/HOL, J is instead encoded as an inductive definition, which in turn is merely a *definitional extension* of higher-order logic, via the well-known impredicative construction through fixed points. To be able to state the completeness part of an adequacy proof, we will consider a kernel of the logic underlying Isabelle/HOL consisting of the following:

- a first-order λ-calculus (*i.e.*, where *bool* can only occur as the target of a legal arrow type) as our term language;
- a second-order minimal logic [55] as a sequent calculus;
- *monotonic* introduction and elimination rules (structural induction included) for atoms;
- simplification on the Hybrid level and other decidable theories like linear arithmetic.

We call such derivations *minimal*. Since Isabelle/HOL does have a form of proof terms [10], this notion is in principle (easily) checkable.[12]

Lemma 11 (Completeness of Representation) *Let Γ be the set $\{x_1 : uexp, \ldots, x_n : uexp\}$; if $(\mathsf{proper}\ \Gamma \Longrightarrow \Gamma \mid\!\models \mathsf{isterm}\ E)$ has a* minimal *derivation in Isabelle/HOL, then $\delta_\Gamma(E)$ is defined and yields a Mini-ML expression t such that $\Gamma \vdash t$ and $\varepsilon_\Gamma(\delta_\Gamma(E)) = E$. Furthermore, $\delta_\Gamma(\varepsilon_\Gamma(t)) = t$.*

Proof: The last equation follows by a straightforward induction on t. The main statement goes by induction on the minimal derivation of $\mathsf{proper}\ \Gamma \Longrightarrow \Gamma \mid\!\models \mathsf{isterm}\ E$; we sketch one case: assume $\mathsf{proper}\ \Gamma \Longrightarrow \Gamma \mid\!\models \mathsf{isterm}\ (\mathsf{fix}\, x\,.\, E\ x)$; by inversion, $\Gamma, x \mid\!\models \mathsf{isterm}\ (E\ x)$ holds for a parameter x under the assumption $\mathsf{proper}\ (\Gamma, x)$. By definition $\delta_\Gamma(\mathsf{fix}\, x\,.\,(E\ x)) = \mathbf{fix}\, x.\, \delta_{\Gamma,x}(E\ x)$. By the I.H. the term $\delta_{\Gamma,x}(E\ x)$ is defined and there is a t s.t. $t = \delta_{\Gamma,x}(E\ x)$ and $\Gamma, x \vdash t$. By the BNF rule for **fix**, $\Gamma \vdash \mathbf{fix}\, x.\, t$ and again by the I.H. and definition, $\varepsilon_\Gamma(\delta_\Gamma(\mathsf{fix}\, x\,.\, E\ x)) = \mathsf{fix}\, x\,.\, E\ x$. □

[12] When full automation is sought, Isabelle/HOL provides a basic intuitionistic prover iprover, and it could, in principle, be connected to an external more efficient intuitionistic prover.

$$\frac{e_1 \Downarrow \textbf{fun x}.\ e_1' \qquad e_2 \Downarrow v_2 \qquad [v_2/x]e_1' \Downarrow v}{e_1 \bullet e_2 \Downarrow v}\ \textbf{ev_app}$$

$$\frac{}{\textbf{fun x}.\ e \Downarrow \textbf{fun x}.\ e}\ \textbf{ev_fun} \qquad \frac{[\textbf{fix}\, x.\, e/x]e \Downarrow v}{\textbf{fix}\, x.\, e \Downarrow v}\ \textbf{ev_fix}$$

$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$

$$\frac{\Gamma, x{:}\tau \vdash e : \tau'}{\Gamma \vdash \textbf{fun x}.\ e : \tau \to \tau'}\ \textbf{tp_fun} \qquad \frac{\Gamma, x{:}\tau \vdash e : \tau}{\Gamma \vdash \textbf{fix}\, x.\, e : \tau}\ \textbf{tp_fix}$$

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}\ \textbf{tp_var} \qquad \frac{\Gamma \vdash e_1 : \tau' \to \tau \qquad \Gamma \vdash e_2 : \tau'}{\Gamma \vdash e_1 \bullet e_2 : \tau}\ \textbf{tp_app}$$

Figure 2: Big step semantics and typing rules for a fragment of Mini-ML.

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

Figure 3: Encoding of big step evaluation in Mini-ML.

Lemma 12 (Compositionality) *1.* $\varepsilon_\Gamma([t_1/x]t_2) = [\varepsilon_\Gamma(t_1)/x]\ \varepsilon_\Gamma(t_2)$, *where x may occur in* Γ.

2. If $\delta_\Gamma(E_1)$ *and* $\delta_\Gamma(E_2)$ *are defined, then* $\delta_\Gamma([E_1/x]E_2) = [\delta_\Gamma(E_1)/x]\ \delta_\Gamma(E_2)$.

Proof: The first result may be proved by induction on t_2 as in Lemma 3.5 of [85], since the encoding function is the same, or we can appeal to the compositionality property of Hybrid, proved as Theorem 4.3 of [25], by unfolding the Hybrid definition of the constructors. The proof of the second part is a similar induction on E_2. □

3.3 Encoding Object-Level Judgments

We now turn to the encoding of object-level *judgments*. In this and the next section, we will consider the standard judgments for big-step call-by-value operational semantics ($e \Downarrow v$) and type inference ($\Gamma \vdash e : \tau$), depicted in Figure 2. Evaluation can be directly expressed as an *inductive* relation (Figure 3) in full HOAS style. Note that substitution is encoded via meta-level β-conversion in clauses for **ev_app** and **ev_fix**.

This definition is an honest to goodness inductive relation that can be used as any other one in an HOL-like setting: for example, queried in the style of Prolog, as in $\exists t.\ \mathsf{fix}\ x.\, \mathsf{fun}\, y.\, x\ @\ y \Downarrow t$, by using only its introduction rules and abstraction solving. Further this kind of relations can be reasoned about using standard induction and case analysis. In fact, the very fact that evaluation is recognized by Isabelle/HOL as inductive yields *inversion principles* in the form of elimination rules. This would correspond, in meta-logics such as *Linc*, to applications of *definitional reflection*. In Isabelle/HOL (as well as in Coq) case analysis is particularly well-supported as part of the datatype/inductive package.

Each predicate p has a general inversion principle p.elim, which can be specialized to a given instance $(\mathtt{p}\ \vec{\mathtt{t}})$ by an ML built-in function p.mk_cases that operates on the current simplification set; specific to our architecture, note again the *abstraction* annotations as meta-logical premises in rules mentioning binding constructs. To take this into account, we have re-implemented this ML function so that it is aware of the quasi-freeness properties of Hybrid constructors and makes the appropriate discrimination. For example the value of eval.mkH_cases $(\mathsf{fun}\,x.\,E \Downarrow V)$ is:

$$\textbf{(meval_fun_E)} \qquad [\![\ \mathsf{fun}\,x.\,E\ x \Downarrow V;\ \bigwedge F\ [\![\ \varnothing \mid\!\models \mathsf{isterm}\ (\mathsf{fun}\,x.\,F\ x);\ \mathsf{abstr}\ F \\ \mathsf{lambda}\ E = \mathsf{lambda}\ F;\ V = \mathsf{fun}\,x.\,F\ x\ ]\!] \Longrightarrow P\ ]\!] \Longrightarrow P$$

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

Finally, using such elimination rules, and more importantly the structural induction principle provided by Isabelle/HOL's inductive package, we can prove standard meta-theorems, for instance uniqueness of evaluation.

MC-Theorem 13 (eval_unique) $E \Downarrow F \Longrightarrow \forall G.\ E \Downarrow G \longrightarrow F = G$.

Proof: By induction on the structure of the derivation of $E \Downarrow F$ and inversion on $E \Downarrow G$. □

The mechanized proof does not appeal, as expected, to the functionality of substitution, as the latter is inherited by the meta-logic, contrary to first-order and "weak" HOAS encodings (see Section 6.4). Compare this also with the standard paper and pencil proof, which usually ignores this property.

We can also prove some "hygiene" results, showing that the encoding of evaluation preserves properness and well-formedness of terms:

MC-Lemma 14 (eval_proper, eval_isterm)

1. $E \Downarrow V \Longrightarrow \mathsf{proper}\ E \wedge \mathsf{proper}\ V$
2. $E \Downarrow V \Longrightarrow \varnothing \mid\!\models \mathsf{isterm}\ E \wedge \varnothing \mid\!\models \mathsf{isterm}\ V$

Note the absence in Figure 3 of any proper assumptions at all while only the isterm assumption in the application case is needed. We have included just enough assumptions to prove the above results.

With respect to the adequacy of object-level judgments, we can establish first the usual statements, for example soundness and completeness of the representation; for the sake of clarity as well as brevity in the statement and proof of the lemma we drop the infix syntax in the Isabelle/HOL definition of evaluation, and omit the obvious definition of the encoding of said judgment:

Lemma 15 (Soundness of the encoding of evaluation) *Let e and v be* closed *Mini-ML expressions such that $e \Downarrow v$; then we can prove in Isabelle/HOL* $(\mathsf{eval}\ \varepsilon_{\varnothing}(e)\ \varepsilon_{\varnothing}(v))$.

Proof: By induction on the derivation of $e \Downarrow v$. Consider the **ev_fun** case: by definition of the encoding on expressions and its soundness (Lemma 10) we have that $\varnothing \mid\!\models \mathsf{isterm}\ \varepsilon_{\varnothing}(\mathbf{fun}\,x.\,e)$ is provable in Isabelle/HOL; by definition and inversion $\varnothing \mid\!\models \mathsf{isterm}\ (\mathsf{fun}\,x.\,\varepsilon_x(e))$ and $\mathsf{abstr}\ (\lambda x.\ \varepsilon_x(e))$ holds, hence by the introduction rules of the inductive definition of evaluation $(\mathsf{eval}\ \mathsf{fun}\,x.\,\varepsilon_x(e)\ \mathsf{fun}\,x.\,\varepsilon_x(e))$ is provable, that is, by definition, $(\mathsf{eval}\ \varepsilon_{\varnothing}(\mathbf{fun}\,x.\,e)\ \varepsilon_{\varnothing}(\mathbf{fun}\,x.\,e))$. The other two cases also use compositionality (Lemma 12) and the induction hypothesis. □

Lemma 16 (Completeness of the encoding of evaluation) *If* $(\mathsf{eval}\ E\ V)$ *has a minimal derivation in Isabelle/HOL, then $\delta_{\varnothing}(E)$ and $\delta_{\varnothing}(V)$ are defined and yield Mini-ML expressions e and v such that $e \Downarrow v$.*

Proof: It follows from MC-Lemma 14 that $\varnothing \mid\!\models \mathsf{isterm}\ E$ and $\varnothing \mid\!\models \mathsf{isterm}\ V$, and thus from Lemma 11 that $\delta_{\varnothing}(E)$ and $\delta_{\varnothing}(V)$ are defined. The proof of $e \Downarrow v$ follows directly by induction on the minimal derivation of $(\mathsf{eval}\ E\ V)$, using compositionality (Lemma 12). □

Now that we have achieved this, does that mean that all the advantages of HOAS are now available in a well-understood system such as Isabelle/HOL? The answer is, unfortunately, a qualified "no". Recall the three "tenets" of HOAS:

1. α-renaming for free, inherited from the ambient λ-calculus identifying meta-level and object-level bound variables;

2. object-level substitution as meta-level β-reduction;

3. object-level contexts as meta-level assumptions.

As of now, we have achieved only the first two. However, while accomplishing in a consistent and relatively painless way the first two points above is no little feat,[13] the second one, in particular, being in every sense novel, no HOAS system can really be worth its name without an accounting and exploiting of reasoning in the presence of *hypothetical and parametric judgments*. We consider the standard example of encoding type inference (Figure 2) in a language such as Twelf. Using Isabelle/HOL-like syntax (where we use *bool* for Twelf's *type*, the **tp_app**, **tp_fun**, and **tp_fix** rules would be represented as follows:

$$\begin{array}{rcl} _ : _ & :: & [uexp, tp] \Rightarrow bool \\ [\![\, E_1 : (T' \to T);\ E_2 : T' \,]\!] & \Longrightarrow & (E_1 \ @\ E_2) : T \\ [\![\, \forall x\ (\underline{x : T \longrightarrow} (E\ x) : T') \,]\!] & \Longrightarrow & (\mathsf{fun}\, x.\, E\ x) : (T \to T') \\ [\![\, \forall x\ (\underline{x : T \longrightarrow} (E\ x) : T) \,]\!] & \Longrightarrow & (\mathsf{fix}\, x.\, E\ x) : T \end{array}$$

Each typing judgment $x : \tau$ in an object-level context (Γ in Figure 2) is represented as a logical assumption of the form $x : T$. In the spirit of higher-order encoding, there is no explicit representation of contexts and no need to encode the **tp_var** rule. However, because of the underlined negative recursive occurrences in the above formulas, there is simply no way to encode this directly in an inductive setting, short of betraying its higher-order nature by introducing ad-hoc datatypes (in this case lists for environments) and, what's worse, all the theory they require. The latter may be trivial on paper, but it is time-consuming and has little to do with the mathematics of the problem.[14]

Moreover, at the level of the meta-theory, it is only the coupling of items 2 and 3 above that makes HOAS encodings—and thus proofs—so elegant and concise; while it is nice not to have to encode substitution for every new signature, it is certainly much nicer not to have to *prove* the related substitution lemmas. This is precisely what the pervasive use of hypothetical and parametric judgments makes possible—one of the many lessons by Martin-Löf.

Even when hypothetical judgments are stratified and therefore inductive, using Hybrid directly within Isabelle/HOL (i.e., at a single level as will become clear shortly) has been only successful in dealing with predicates over *closed* terms (such as simulation). However, it is necessary to resort to a more traditional encoding, i.e. via explicit environments, when dealing with judgments involving *open* objects. These issues became particularly clear in the case-study reported in [72], where the Hybrid syntax allowed the following elegant encoding of *closed* applicative (bi)simulation [1]:

$$\begin{array}{l} [\![\, \forall T.\ R \Downarrow \mathsf{fun}\, x.\, T\ x \longrightarrow (\mathsf{abstr}\ T \longrightarrow \\ \exists U.\ S \Downarrow \mathsf{fun}\, x.\, U\ x \wedge \mathsf{abstr}\ U \wedge \forall p.\ \ (T\ p) \preccurlyeq (U\ p)) \,]\!] \\ \qquad\qquad \Longrightarrow R \preccurlyeq S \end{array}$$

together with easy proofs of its basic properties (for example, being a pre-order). Yet, dealing with *open* (bi)simulation required the duplication of analogous work in a much less elegant way.

This does not mean that results of some interest cannot be proved working at one level. For example, the aforementioned paper (painfully) succeeded in checking non-trivial results such as Howe-style's proof of congruence of applicative (bi)simulation [52].[15] Another example [2] is the quite intricate verification of subject reduction of MIL-LITE [8], the intermediate language of the SML into JVM compiler. Achieving that result, however, was complicated by the (dubious in hindsight) choice of treating ML-like references with the eigenvariable encoding proposed in [62].

In those experiments, HOAS in Isabelle/HOL seemed only a nice interlude, soon to be overwhelmed by tedious and non-trivial (at least mechanically) proofs of list-based properties of open judgments and by a number of substitutions lemmas that we had hoped to have eliminated for good. These are the kinds of issues we address with the two-level architecture, discussed next.

[13] Compare this to other methods of obtaining α-conversion by constructing equivalence classes [34, 109] in a proof assistant.

[14] A compromise is the "weak" HOAS view mentioned earlier and discussed in Section 6.4.

[15] However, it would take a significant investment in man-months to extend the result from the lazy λ-calculus to more interesting calculi such as [56].

4 A Two-Level Architecture

The *specification* level mentioned earlier (see Figure 1) is introduced to solve the problems discussed in the previous section of reasoning in the presence of negative occurrences of OL judgments and reasoning about open terms. A *specification logic* (SL) is defined inductively, and used to encode OL judgments. Since hypothetical judgments are encapsulated within the SL, they are not required to be inductive themselves. In addition, SL contexts can encode assumptions about OL variables, which allows reasoning about open terms of the OL. We introduce our first example SL in Section 4.1. Then, in Section 4.2, we continue the discussion of the sample OL introduced in Section 3, this time illustrating the encoding of judgments at the SL level.

4.1 Encoding the Specification Logic

We introduce our first SL, namely a fragment of second-order hereditary Harrop formulas [68]. This is sufficient for the encoding of our first case-study: subject reduction for the sub-language of Mini-ML that we have introduced before (Figure 2). The SL language is defined as follows, where τ is a ground type.

$$\begin{array}{rcll} \textit{Clauses} & D & ::= & \top \mid A \mid D_1 \wedge D_2 \mid G \rightarrow A \mid \forall^{\tau} x.\, D \mid \forall^{\tau \rightarrow \tau} x.\, D \\ \textit{Goals} & G & ::= & \top \mid A \mid G_1 \wedge G_2 \mid A \rightarrow G \mid \forall^{\tau} x.\, G \\ \textit{Context} & \Gamma & ::= & \varnothing \mid A, \Gamma \end{array}$$

The τ in the grammar for goals is instantiated with *expr* in this case. Thus, quantification is over a ground type whose exact elements depend on the instantiation of *con*, which, as discussed in Section 4.2, is defined at the OL level. Quantification in clauses includes second-order variables. We will use it, for instance, to encode variables E of type $\textit{expr} \Rightarrow \textit{expr}$ that appear in terms such as $\mathsf{fun}\, x.\, E\ x$. Quantification in clauses may also be over first-order variables of type *expr*, as well as over variables of other ground types such as *tp*. In this logic, we view contexts as *sets*, where we overload the comma to denote adjoining an element to a set. Not only does this representation make mechanical proofs of the standard proof-theoretic properties easier compared to using lists, but it is also appropriate for a sequent calculus that enjoys contraction and exchange, and designed so that weakening is an admissible property. This approach will also better motivate the use of lists in sub-structural logics in the next section. Further, in our setting, contexts are particularly simple, namely sets of *atoms*, since only atoms are legal antecedents in implications in goals.

The syntax of *goal* formulas can be directly rendered with an Isabelle/HOL datatype:

$$\textit{datatype} \quad o \quad = \quad \mathsf{tt} \mid \langle \textit{atm} \rangle \mid o \ \mathsf{and}\ o \mid \textit{atm}\ \mathsf{imp}\ o \mid \mathsf{all}\ (\textit{expr} \Rightarrow o)$$

We write *atm* to represent the type of atoms; $\langle _ \rangle$ coerces atoms into propositions. The definition of *atm* is left as an implicit parameter at this stage, because various instantiations will yield the signature of different OLs, specifically predicates used to encode their judgments.

This language is so simple that its sequent calculus is analogous to a logic programming interpreter. All clauses allowed by the above grammar can be normalized to (a set of) clauses of the form:

$$\textit{Clauses} \quad D \quad ::= \quad \forall^{\sigma_1} x_1 \ldots \forall^{\sigma_n} x_n\ (G \rightarrow A)$$

where $n \geq 0$, and for $i = 1, \ldots, n$, σ_i is either a ground type, or has the form $\tau_1 \rightarrow \tau_2$ where τ_1 and τ_2 are ground types. In analogy with logic programming, when writing clauses, outermost universal quantifiers will be omitted, as those variables are implicitly quantified by the meta-logic; implication will be written in the reverse direction, *i.e.*, we write simply $A \longleftarrow G$,[16] or when we need to be explicit about the quantified variables, we write $\forall \Sigma (A \longleftarrow G)$ where $\Sigma = \{x_1, \ldots, x_n\}$. This notation yields a more proof-search oriented notion of clauses. In fact, we can write inference rules so that the only left rule is similar to Prolog's backchaining. Sequents have the form $\Sigma; \Gamma \longrightarrow_{\Pi} G$, where Σ is the current signature of eigenvariables and we distinguish clauses belonging to a static database, written Π, from atoms introduced via the right implication rule, written Γ. The rules for this logic are given in Figure 4. In the **bc** rule, $[\Pi]$ is the set of all possible instances of clauses in Π obtained by instantiating outermost universal quantifiers with all closed terms of appropriate types.

[16] This is also why we can dispose of the mutual definition of clauses and goals and avoid using a mutually inductive datatype, which, in the lack of some form of subtyping, would make the encoding redundant.

$$\frac{}{\Sigma;(\Gamma,A) \longrightarrow_\Pi A}\,\textbf{init} \qquad \frac{\Sigma;(\Gamma,A) \longrightarrow_\Pi G}{\Sigma;\Gamma \longrightarrow_\Pi A \to G}\,\to_R$$

$$\frac{\Sigma;\Gamma \longrightarrow_\Pi G_1 \qquad \Sigma;\Gamma \longrightarrow_\Pi G_2}{\Sigma;\Gamma \longrightarrow_\Pi G_1 \wedge G_2}\,\wedge_R$$

$$\frac{}{\Sigma;\Gamma \longrightarrow_\Pi \top}\,\top_R \qquad \frac{(\Sigma,a:\tau);\Gamma \longrightarrow_\Pi G[a/x]}{\Sigma;\Gamma \longrightarrow_\Pi \forall^\tau x.\, G}\,\forall_R$$

$$\frac{\Sigma;\Gamma \longrightarrow_\Pi G \qquad A \longleftarrow G \in [\Pi]}{\Sigma;\Gamma \longrightarrow_\Pi A}\,\textbf{bc}$$

Figure 4: A minimal sequent calculus with backchaining

This inference system is equivalent to the standard presentation of minimal logic [55], where the right rules are the same and the left rules (given below) for conjunction, implication and universal quantification replace the **bc** rule.

$$\frac{\Sigma;\Gamma,D_1,D_2 \longrightarrow_\Pi G}{\Sigma;\Gamma,D_1 \wedge D_2 \longrightarrow_\Pi G}\,\wedge_{\mathbf{L}} \qquad \frac{\Sigma;\Gamma,D[t/x] \longrightarrow_\Pi G}{\Sigma;\Gamma,\forall x.\, D \longrightarrow_\Pi G}\,\forall_{\mathbf{L}}$$

$$\frac{\Sigma;\Gamma \longrightarrow_\Pi G \qquad \Sigma;\Gamma,B \longrightarrow_\Pi A}{\Sigma;\Gamma,(G \to B) \longrightarrow_\Pi A}\,\to_{\mathbf{L}}$$

In fact, the **bc** rule is derivable by eliminating the universal quantifiers until the head of a clause matches the atom on the right and then applying $\to_{\mathbf{L}}$. The reader should remember that we are working in an ambient logic modulo some equational theory (in the case of Isabelle $=_{\alpha\beta\eta}$) and that both atomic rules (**init** and **bc**) are applicable in the case when an atom on the right appears as an assumption and unifies with the head of a definite clause in the program Π. Thus, we can inherit the completeness of *uniform provability* [68] w.r.t. an ordinary sequent calculus, which holds for a much more expressive conservative extension of our SL, namely higher-order Harrop formulas.

We encode this SL in Figure 5. We use the symbol $\rhd$ for the sequent arrow, in this case decorated with natural numbers that represent the *height* of a proof; this measure allows us to reason by complete induction.[17] For convenience we write $\Gamma \rhd G$ if there exists an n such that $\Gamma \rhd_n G$, and furthermore we simply write $\rhd G$ when $\varnothing \rhd G$. The first four clauses of the definition directly encode the introduction (R) rules of the figure. In the encoding of the $\forall_R$ rule, when we introduce new eigenvariables of type *expr*, we need to assume that they are proper. This assumption might be required for proving subgoals of the form $(\mathsf{abstr}\, E)$ for subterms $E :: expr \Rightarrow expr$ that appear in the goal as arguments to binding constructors; see MC-Lemma 3 (*proper_abst)*.

We remark that the only dependence on Hybrid in this layer is on the definition of proper. This will also be true of the SL we consider in Section 5. Although we do not discuss it here, we could use SLs with (different) kinds of quantifiers that could not be implemented via a datatype but only with Hybrid constants; for example universal quantification in higher-order logic. In this case, the specification layer would have a much greater dependence on Hybrid. On the other hand, if we take the alternative solution to proper terms mentioned earlier (when discussing MC-Lemma 3) and replace *expr* with a type *prpr* containing exactly the terms that satisfy proper, and consider only

[17] Proven in the Isabelle/HOL's library in the form $(\bigwedge n.\ \forall m < n.\ P\ m \Longrightarrow P\ n) \Longrightarrow P\ x$.

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

Figure 5: Encoding of a minimal specification logic

the SLs presented in this paper, then these SLs can be parameterized by the type of terms used in quantification, and can be instantiated with types other than *prpr*.

In the last two rules in Figure 5, atoms are provable either by assumption or via *backchaining* over a set of Prolog-like rules, which encode the properties of the OL in question as an inductive definition of the predicate prog of type $[\textit{atm}, \textit{oo}] \Rightarrow \textit{bool}$, which will be instantiated in Section 4.2. The sequent calculus is parametric in those clauses and so are its meta-theoretical properties. Because prog is static it will be mentioned explicitly only in adequacy proofs. The notation $A \longleftarrow G$ in Figure 5 represents an instance of one of the clauses of the inductive definition of prog.

As a matter of fact our encoding of the judgment $\Gamma \rhd_n G$ can be seen as a simple extension of the so-called "vanilla" Prolog meta-interpreter, often known as `demo` [49]; similarly, the **bc** rule would correspond to the following clause, using `prog` in place of Prolog's built-in `clause`:

```
demo(Gamma,s(N),A):- prog(A,G), demo(Gamma,N,G).
```

Existential quantification could be added to the grammar of goals, as follows:

$$[\![\, \exists x. \Gamma \rhd_n (G\ x) \,]\!] \quad \Longrightarrow \quad \Gamma \rhd_{n+1} (\mathsf{ex}\ x.\ G\ x)$$

but this yields no real increase in expressivity, as existentials in the body of goals can be safely transformed to outermost universal quantifiers, while (continuing the logic programming analogy) the above rule simply delegates the witness choice to the ambient logic unification algorithm.

As before, the fact that provability is inductive yields inversion principles as elimination rules. For example the inversion theorem that analyzes the shape of a derivation ending in an atom from the *empty* context is obtained simply with a call to the standard `mk_cases` function, namely `mk_cases`"$\rhd_j \langle A \rangle$" is:

$$[\![\, \rhd_j \langle A \rangle;\ \bigwedge G\ i. [\![\, A \longleftarrow G;\ \rhd_i G;\ j = \textit{Suc}\ i \,]\!] \Longrightarrow P \,]\!] \Longrightarrow P$$

The adequacy of the encoding of the SL can be established adapting the analogous proof in [62]. To do so, we overload the decoding function in several ways. First, we need to decode terms of types other than *expr*. For example, decoding terms of type $\textit{expr} \Rightarrow \textit{expr}$ is required for most OLs. For Mini-ML, we also need to decode terms of type *tp*. The decoding is extended in the obvious way. For example, for decoding second-order terms, we define $\delta_\Sigma(\lambda\, x. E) = \lambda\, x.\, \delta_{\Sigma,x}(E)$. Second, to decode both goals and clauses, we extend Σ to allow both first- and second-order variables. We can then extend the decoding so that if G is a term of type *oo* with free variables in Σ, then $\delta_\Sigma(G)$ is its translation to a formula of minimal logic, and if Γ is a set of terms of type *atm set*, then $\delta_\Sigma(\Gamma)$ is its translation to a set of atomic formulas of minimal logic. In addition, we restrict the form of the definition of prog so that every clause of the inductive definition is a closed formula of the form:

$$\bigwedge \Sigma \Big([\![\, \mathsf{abstr}\ E_1; \ldots; \mathsf{abstr}\ E_n \,]\!] \Longrightarrow (A \longleftarrow G) \Big)$$

where Σ is a set of variables including at least $E_1, \ldots, E_n$, each of type *expr* $\Rightarrow$ *expr*, with $n \geq 0$. To obtain a theory in minimal logic that corresponds to the definition of prog, we decode each clause to a formula of minimal logic of the form $\forall\Sigma(\delta_\Sigma(G) \rightarrow \delta_\Sigma(A))$. For SL adequacy, we also need to introduce two conditions, which become additional proof obligations when establishing OL adequacy. They are:

1. It is only ever possible to instantiate universal quantifiers in prog clauses with terms for which the decoding is defined.

2. For every term E :: *expr* $\Rightarrow$ *expr* used to instantiate universal quantifiers in prog clauses, $(\mathsf{abstr}\ E)$ holds.

The latter will follow from the former and the fact that for *all* terms E :: *expr* $\Rightarrow$ *expr* for which the decoding is defined, $(\mathsf{abstr}\ E)$ holds.

Lemma 17 (Soundness and completeness of the encoding of the specification logic) *Let* prog *be an inductive definition of the restricted form described above, and let* Π *be the corresponding theory in minimal logic. Let* G *be a formula of type* o *and let* Γ *be a set of atoms. Let* Σ *be a set of variables of type expr that contains all the free variables in* Γ *and* G*. Then the sequent* $\mathsf{proper}\ \Sigma \Longrightarrow \Gamma \rhd G$ *has a minimal derivation in Isabelle/HOL (satisfying conditions 1 and 2 above) if and only if there is a derivation of* $\Sigma; \delta_\Sigma(\Gamma) \longrightarrow_\Pi \delta_\Sigma(G)$ *according to the rules of Figure 4.*

Proof: The proof of the forward direction follows directly by induction on the minimal derivation of $\mathsf{proper}\ \Sigma \Longrightarrow \Gamma \rhd G$. Compositionality (Lemma 12) is needed for the case when $\Gamma \rhd G$ is proved by the last clause of Figure 5. The proof of the backward direction is by direct induction on the derivation of $\Sigma; \delta_\Sigma(\Gamma) \longrightarrow_\Pi \delta_\Sigma(G)$. Compositionality (Lemma 12) and conditions 1 and 2 are needed for the **bc** case. □

MC-Theorem 18 (Structural Rules) *The following rules are admissible:*

1. *Height weakening:* $[\![\, \Gamma \rhd_n G;\ n < m \,]\!] \Longrightarrow \Gamma \rhd_m G$.[18]

2. *Context weakening:* $[\![\, \Gamma \rhd G;\ \Gamma \subseteq \Gamma' \,]\!] \Longrightarrow \Gamma' \rhd G$.

3. *Atomic cut:* $[\![\, A, \Gamma \rhd G;\ \Gamma \rhd \langle A \rangle \,]\!] \Longrightarrow \Gamma \rhd G$.

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.

2. By a similar fully automated induction on the structure of the sequent derivation, combining resolution on the sequent introduction rules with simplification in order to to discharge some easy set-theoretic subgoals.

3. Atomic cut is a corollary of the following lemma:

$$[\![\, A, \Gamma \rhd_i G;\ \Gamma \rhd_j \langle A \rangle \,]\!] \quad \Longrightarrow \quad \Gamma \rhd_{i+j} G$$

easily proved by complete induction on the height of the derivation of $A, \Gamma \rhd_i G$. The whole proof consists of two dozen instructions, with very little ingenuity required from the human collaborator. We show the implication case, which uses context weakening (the other cases are simpler):

$$[\![\, IH;\ A, \Gamma \rhd_i (B \ \mathsf{imp}\ C);\ \Gamma \rhd_j \langle A \rangle \,]\!] \Longrightarrow \Gamma \rhd_{i+j} (B \ \mathsf{imp}\ C)$$

where IH is the induction hypothesis:

$$\forall m, n, A, G, \Gamma'. [\![\, m < i;\ A, \Gamma' \rhd_m G;\ \Gamma' \rhd_n \langle A \rangle \,]\!] \Longrightarrow \Gamma' \rhd_{m+n} G.$$

[18] This lemma turns out to be fairly useful, as it lets you manipulate as appropriate the height of two sub-derivations, such as in the $\wedge_R$ rule.

From $A,\Gamma \rhd_i (B \,\mathsf{imp}\, C)$, we can conclude by the inversion principle for the fourth clause of the inductive definition of the SL (Figure 5), that there is an i_1 such that $i = i_1 + 1$ and $B,A,\Gamma \rhd_{i_1} C$. Simple arithmetic and replacement reduces the goal to:

$$[\![\, IH;\ B,A,\Gamma \rhd_{i_1} C;\ \Gamma \rhd_j \langle A \rangle \,]\!] \Longrightarrow \Gamma \rhd_{i_1+j+1} (B \,\mathsf{imp}\, C)$$

and applying the fourth clause of the SL to the right side further reduces it to:

$$[\![\, IH;\ B,A,\Gamma \rhd_{i_1} C;\ \Gamma \rhd_j \langle A \rangle \,]\!] \Longrightarrow B,\Gamma \rhd_{i_1+j} C.$$

At this point, we can apply the induction hypothesis instantiating universally quantified variables m,n,Γ',G in IH with $i_1, j, (B,\Gamma), C$, respectively, leaving three subgoals:

$$\begin{array}{lcl} [\![\, IH;\ B,A,\Gamma \rhd_{i_1} C;\ \Gamma \rhd_j \langle A \rangle \,]\!] & \Longrightarrow & i_1 < i \\ [\![\, IH;\ B,A,\Gamma \rhd_{i_1} C;\ \Gamma \rhd_j \langle A \rangle \,]\!] & \Longrightarrow & A,B,\Gamma \rhd_{i_1} C \\ [\![\, IH;\ B,A,\Gamma \rhd_{i_1} C;\ \Gamma \rhd_j \langle A \rangle \,]\!] & \Longrightarrow & B,\Gamma \rhd_j \langle A \rangle \end{array}$$

which are easily provable, the third one by weakening.

□

4.2 The Object Logic

Recall the rules for call-by-value operational semantics ($e \Downarrow v$) and type inference ($\Gamma \vdash e : \tau$) given in Figure 2. The subject reduction for this source language is stated as usual.

Theorem 19 (Subject Reduction) *If $e \Downarrow v$ and $\vdash e : \tau$, then $\vdash v : \tau$.*

Proof: By structural induction on evaluation and inversion on typing, using weakening and a substitution lemma in the **ev_app** and **ev_fix** cases. □

We now return to the encoding of the OL, this time using the SL to encode judgments. The encoding of OL syntax is unchanged. (See Section 3.) Recall that it involved introducing a specific type for *con*. Here, we will also instantiate type *atm* and predicate prog. In this section and the next, we now also make full use of the definitions and theorems in both Hybrid and the SL layers.

Type *atm* is instantiated as expected, defining the atomic formulas of the OL.

$$\textit{datatype} \quad \textit{atm} \quad = \quad \mathsf{isterm}\ \textit{uexp} \mid \textit{uexp} \Downarrow \textit{uexp} \mid \textit{uexp} : \textit{tp}$$

The clauses for the OL deductive systems are given as rules of the prog (recall the notation $_ \longleftarrow _$) inductive definition in Figure 6. Recall that the encoding of evaluation in Figure 3 and the encoding of the isterm predicate for adequacy purposes both used inductive definitions. Here we define them both at the SL level along with the OL level typing judgment. Note that no explicit variable context is needed for this version of isterm. They are handled implicitly by the contexts of atomic assumptions of the SL, resulting in a more direct encoding. As before, in the evaluation clauses, there are no proper assumptions and two isterm assumptions. Neither kind of assumption appears in the clauses for the typing rules. None are required to prove the analogue of MC-Lemma 14 for both evaluation and typing.

MC-Lemma 20

1.	$\rhd \langle E \Downarrow V \rangle \Longrightarrow \mathsf{proper}\ E \wedge \mathsf{proper}\ V$	**(eval_proper)**
2.	$\rhd \langle E \Downarrow V \rangle \Longrightarrow \rhd \langle \mathsf{isterm}\ E \rangle \wedge \rhd \langle \mathsf{isterm}\ V \rangle$	**(eval_isterm)**
3.	$\rhd \langle E : T \rangle \Longrightarrow \mathsf{proper}\ E$	**(hastype_proper)**
4.	$\rhd \langle E : T \rangle \Longrightarrow \rhd \langle \mathsf{isterm}\ E \rangle$	**(hastype_isterm)**

$$
\begin{array}{rcl}
\textit{Inductive}\ _ \longleftarrow _ & :: & [\,atm, oo\,] \Rightarrow bool \\
 & \Longrightarrow & \mathsf{isterm}\ E_1\ @\ E_2 \longleftarrow \langle \mathsf{isterm}\ E_1 \rangle\ \mathsf{and}\ \langle \mathsf{isterm}\ E_2 \rangle \\
[\![\ \mathsf{abstr}\ E\ ]\!] & \Longrightarrow & \mathsf{isterm}\ \mathsf{fun}\, x.\, E\ x \longleftarrow \mathsf{all}\, x.\, (\mathsf{isterm}\ x)\ \mathsf{imp}\ \langle \mathsf{isterm}\ (E\ x) \rangle \\
[\![\ \mathsf{abstr}\ E\ ]\!] & \Longrightarrow & \mathsf{isterm}\ \mathsf{fix}\, x.\, E\ x \longleftarrow \mathsf{all}\, x.\, (\mathsf{isterm}\ x)\ \mathsf{imp}\ \langle \mathsf{isterm}\ (E\ x) \rangle \\
 & & \\
[\![\ \mathsf{abstr}\ E_1'\ ]\!] & \Longrightarrow & E_1\ @\ E_2 \Downarrow V \longleftarrow \\
 & & \langle E_1 \Downarrow \mathsf{fun}\, x.\, E_1'\ x \rangle\ \mathsf{and}\ \langle E_2 \Downarrow V_2 \rangle\ \mathsf{and}\ \langle (E_1'\ V_2) \Downarrow V \rangle \\
[\![\ \mathsf{abstr}\ E\ ]\!] & \Longrightarrow & \mathsf{fun}\, x.\, E\ x \Downarrow \mathsf{fun}\, x.\, E\ x \longleftarrow \langle \mathsf{isterm}\ (\mathsf{fun}\, x.\, E\ x) \rangle \\
[\![\ \mathsf{abstr}\ E\ ]\!] & \Longrightarrow & \mathsf{fix}\, x.\, E\ x \Downarrow V \longleftarrow \langle E\ (\mathsf{fix}\, x.\, E\ x) \Downarrow V \rangle\ \mathsf{and}\ \langle \mathsf{isterm}\ (\mathsf{fix}\, x.\, E\ x) \rangle \\
 & & \\
 & \Longrightarrow & (E_1\ @\ E_2) : T \longleftarrow \langle E_1 : (T' \to T) \rangle\ \mathsf{and}\ \langle E_2 : T' \rangle \\
[\![\ \mathsf{abstr}\ E\ ]\!] & \Longrightarrow & (\mathsf{fun}\, x.\, E\ x) : (T \to T') \longleftarrow \mathsf{all}\, x.\, (x : T)\ \mathsf{imp}\ \langle (E\ x) : T' \rangle \\
[\![\ \mathsf{abstr}\ E\ ]\!] & \Longrightarrow & (\mathsf{fix}\, x.\, E\ x) : T \longleftarrow \mathsf{all}\, x.\, (x : T)\ \mathsf{imp}\ \langle (E\ x) : T \rangle
\end{array}
$$

Figure 6: OL clauses: encoding of well-formedness, evaluation and typing.

Proof: All the proofs are by standard induction on the given derivation, except the last one, whose statement needs to be generalized as follows:

$$[\![\ \forall E, T.\ (E : T) \in \Gamma \longrightarrow (\mathsf{isterm}\ E) \in \Gamma';\ \Gamma \rhd_i \langle E : T \rangle\ ]\!] \quad \Longrightarrow \quad \Gamma' \rhd_i \langle \mathsf{isterm}\ E \rangle$$

□

With the new version of isterm, we restate the Validity and Completeness of Representation lemmas (Lemmas 10 and 11) as follows:

Lemma 21 (Two-level Validity of Representation) *Let* $\Gamma = \{x_1, \ldots, x_n\}$ *and let* $\overline{\Gamma}$ *be the set of atoms* $\{\mathsf{isterm}\ x_1, \ldots, \mathsf{isterm}\ x_n\}$. *If* $\Gamma \vdash e$, *then the following is provable in Isabelle/HOL:*

$$\mathsf{proper}\ \Gamma \quad \Longrightarrow \quad \overline{\Gamma} \rhd \langle \mathsf{isterm}\ \varepsilon_\Gamma(e) \rangle$$

Lemma 22 (Two-level Completeness of Representation) *Let* Γ *be the set* $\{x_1 : uexp, \ldots, x_n : uexp\}$ *and let* $\overline{\Gamma}$ *be the set of atoms* $\{\mathsf{isterm}\ x_1, \ldots, \mathsf{isterm}\ x_n\}$. *If there is a minimal derivation in Isabelle/HOL of* $\mathsf{proper}\ \Gamma \Longrightarrow \overline{\Gamma} \rhd \langle \mathsf{isterm}\ E \rangle$, *then* $\delta_\Gamma(E)$ *is defined and yields a Mini-ML expression* t *such that* $\Gamma \vdash t$ *and* $\varepsilon_\Gamma(\delta_\Gamma(E)) = E$. *Furthermore,* $\delta_\Gamma(\varepsilon_\Gamma(t)) = t$.

We will skip the statement and proof of two-level adequacy of the other OL judgments, hoping that the reader will spot the similarity with the above two lemmas. Note that, although we do not state it formally, condition 1 of Lemma 17 follows from completeness lemmas such as Lemma 22. The isterm and abstr assumptions added to the clauses of Figure 6 are exactly the ones needed to establish this fact for this OL.

We remark again that the combination of Hybrid with the use of an SL allows us to simulate definitional reflection via the built-in elimination rules of the prog inductive definition *without* the use of additional axioms. For example the inversion principle of the function typing rule is:

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

Before turning to the proof of Theorem 19, we first illustrate the use of this encoding with the following simple OL typing judgment.

MC-Lemma 23 $\exists T. \rhd \langle \mathsf{fun}\, x.\, \mathsf{fun}\, y.\, x \mathbin{@} y : T \rangle$

Proof: This goal is equivalent to: $\exists T. \exists n. \varnothing \rhd_n \langle \mathsf{fun}\, x.\, \mathsf{fun}\, y.\, x \mathbin{@} y : T \rangle$. It can be proved fully automatically by a simple tactic described below. Here, we describe the main steps in detail for illustration purposes. We use the instantiations for T and n that would be generated by the tactic and show:

$$\varnothing \rhd_8 \langle \mathsf{fun}\, x.\, \mathsf{fun}\, y.\, x \mathbin{@} y : (\mathsf{i} \to \mathsf{i}) \to (\mathsf{i} \to \mathsf{i}) \rangle.$$

We apply the last rule of the SL in Figure 5, instantiating the first premise with the OL clause from Figure 6 encoding the **tp_fun** rule for typing abstractions, leaving two premises to be proved:

$$\begin{array}{l}(\mathsf{fun}\, x.\, \mathsf{fun}\, y.\, x \mathbin{@} y) : (\mathsf{i} \to \mathsf{i}) \to (\mathsf{i} \to \mathsf{i}) \longleftarrow \mathsf{all}\, x.\, (x : \mathsf{i} \to \mathsf{i})\ \mathsf{imp}\ \langle \mathsf{fun}\, y.\, x \mathbin{@} y : \mathsf{i} \to \mathsf{i} \rangle; \\ \varnothing \rhd_7 \mathsf{all}\, x.\, (x : \mathsf{i} \to \mathsf{i})\ \mathsf{imp}\ \langle \mathsf{fun}\, y.\, x \mathbin{@} y : \mathsf{i} \to \mathsf{i} \rangle.\end{array}$$

The first now matches directly the clause in Figure 6 for **tp_fun**, resulting in the proof obligation $(\mathsf{abstr}\ \lambda x.\mathsf{fun}\, y.\, x \mathbin{@} y)$ which is handled automatically by abstr_tac discussed in Section 3. To prove the second, we apply further rules of the SL to obtain the goal:

$$[\![\ \mathsf{proper}\ x\ ]\!] \Longrightarrow \{x : \mathsf{i} \to \mathsf{i}\} \rhd_5 \langle \mathsf{fun}\, y.\, x \mathbin{@} y : \mathsf{i} \to \mathsf{i} \rangle.$$

We now have a subgoal of the same "shape" as the original theorem. Repeating the same steps, we obtain:

$$[\![\ \mathsf{proper}\ x; \mathsf{proper}\ y\ ]\!] \Longrightarrow \{x : \mathsf{i} \to \mathsf{i}, y : \mathsf{i}\} \rhd_2 \langle x \mathbin{@} y : \mathsf{i} \rangle.$$

Along the way, the proof obligation $(\mathsf{abstr}\ \lambda y.x \mathbin{@} y)$ is proved by abstr_tac. The assumption $(\mathsf{proper}\ x)$ is needed to complete this proof. At this point, we again apply the SL backchain rule using the OL clause for **tp_app**, obtaining two subgoals, the first of which is again directly provable from the OL definition. The second:

$$[\![\ \mathsf{proper}\ x; \mathsf{proper}\ y\ ]\!] \Longrightarrow \{x : \mathsf{i} \to \mathsf{i}, y : \mathsf{i}\} \rhd_1 \langle x : \mathsf{i} \to \mathsf{i} \rangle\ \mathsf{and}\ \langle y : \mathsf{i} \rangle.$$

is completed by applying the rules in Figure 5 encoding the $\wedge_R$ and **init** rules of the SL. □

Here is the code for the 2lprolog_tac tactic that can automate this proof and others involving OL goals using the SL:

```
fun 2lprolog_tac defs i =
        fast_tac(HOL_cs addIs seq.intrs @ prog.intrs
                (simpset() addSolver (abstr_solver defs))) i;
```

The tactic is based on logic programming style depth-first search (although we could switch to breath-first or iterative deepening) using a small set of initial axioms for the core of higher-order logic (`HOL_cs`), the rules of the SL (`seq.intrs`) and of the OL (`prog.intrs`). Additionally, it also employs simplification augmented with abstr_tac as discussed in Section 3.

Now we have all the elements in place for a formal HOAS proof of Theorem 19. Note that while a substitution lemma for typing plays a central role in the informal subject reduction proof, here, in the HOAS tradition, it will be subsumed by the use of the cut rule on the hypothetical encoding of the typing of an abstraction.

MC-Theorem 24 (OL_subject_reduction)

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

Proof: The proof is by complete induction on the height of the derivation of evaluation. It follows closely the proofs in [33, 62], although those theorems are for the lazy λ-calculus, while here we consider eager evaluation. Applying meta-level introduction rules and induction on n, we obtain the sequent:

$$[\![\ IH;\ \rhd_n \langle E \Downarrow V \rangle,\ \rhd \langle E : T \rangle\ ]\!] \Longrightarrow \rhd \langle V : T \rangle$$

where IH is the induction hypothesis:

$$\forall m < n.\ E, V.\ \rhd_m \langle E \Downarrow V\rangle \longrightarrow (\forall T.\ \rhd\ \langle E : T\rangle \longrightarrow \rhd\ \langle V : T\rangle).$$

Since the right side of the SL sequent in the middle hypothesis is an atom and the left side is empty, any proof of this sequent must end with the last rule of the SL in Figure 5, which implements the **bc** rule. Also, since the right side is an evaluation judgment, backchaining must occur on one of the middle three clauses of the OL in Figure 6, thus breaking the proof into three cases. In the formal proof, we obtain these three cases by applying standard inversion tactics:

$$\begin{aligned}
&[\![\, IH[i+1/n];\ \mathsf{abstr}\ E_1';\ (\rhd_i \langle E_1 \Downarrow \mathsf{fun}\, x.\, E_1'\ x\rangle \ \mathsf{and}\ \langle E_2 \Downarrow V_2\rangle \ \mathsf{and}\ \langle (E_1'\ V_2) \Downarrow V\rangle);\ \ \rhd\ \langle (E_1\ @\ E_2) : T\rangle \,]\!] \Longrightarrow \rhd\ \langle V : T\rangle \\
&[\![\, IH[i+1/n];\ \mathsf{abstr}\ E;\ \rhd_i \langle \mathsf{isterm}\ (\mathsf{fun}\, x.\, E\ x)\rangle;\ \rhd\ \langle (\mathsf{fun}\, x.\, E\ x) : T\rangle \,]\!] \Longrightarrow \rhd\ \langle (\mathsf{fun}\, x.\, E\ x) : T\rangle \qquad (2)\\
&[\![\, IH[i+1/n];\ \mathsf{abstr}\ E;\ (\rhd_i \langle E\ (\mathsf{fix}\, x.\, E\ x) \Downarrow V\rangle \ \mathsf{and}\ \langle \mathsf{isterm}\ (\mathsf{fix}\, x.\, E\ x)\rangle);\ \ \rhd\ \langle (\mathsf{fix}\, x.\, E\ x) : T\rangle \,]\!] \Longrightarrow \rhd\ \langle V : T\rangle
\end{aligned}$$

where $IH[i+1/n]$ denotes IH with the single occurrence of n replaced by $i+1$. The theorems mentioned earlier about injectivity and distinctness of the constructors fun, @, and fix are used by the inversion tactics. In contrast, in the proof in [33], because these constructors were not defined inductively, specialized inversion theorems were proved from axioms stating the necessary injectivity and distinctness properties, and then applied by hand. The second subgoal above is directly provable: no surprises here. We illustrate the first one further. Applying inversion to both the third and fourth hypotheses of the first subgoal, the subgoal reduces it to:

$$\begin{aligned}
[\![\, IH[i+3/n];\ \mathsf{abstr}\ E_1';\ \rhd_{i+1} \langle E_1 \Downarrow \mathsf{fun}\, x.\, E_1'\ x\rangle;\ \rhd_i \langle E_2 \Downarrow V_2\rangle;\\
\rhd_i \langle (E_1'\ V_2) \Downarrow V\rangle; \rhd\ \langle E_1 : T' \to T\rangle;\ \rhd\ \langle E_2 : T'\rangle \,]\!]\\
\Longrightarrow \rhd\ \langle V : T\rangle.
\end{aligned}$$

It is now possible to apply the induction hypothesis to the typing and evaluation judgments for E_1 and E_2 to obtain:

$$\begin{aligned}
[\![\, IH[i+3/n];\ \mathsf{abstr}\ E_1';\ \rhd_{i+1} \langle E_1 \Downarrow \mathsf{fun}\, x.\, E_1'\ x\rangle;\ \rhd_i \langle E_2 \Downarrow V_2\rangle;\ \rhd_i \langle (E_1'\ V_2) \Downarrow V\rangle; \ldots;\\
\rhd\ \langle \mathsf{fun}\, x.\, E_1'\ x : T' \to T\rangle;\ \rhd\ \langle V_2 : T'\rangle \,]\!]\\
\Longrightarrow \rhd\ \langle V : T\rangle.
\end{aligned}$$

We can now apply inversion to the hypothesis with the arrow typing judgment involving both the fun constructor of the OL and the all constructor of the SL. Inversion at the OL level gives:

$$\begin{aligned}
[\![\, IH[i+3/n];\ \mathsf{abstr}\ E_1';\ \rhd_{i+1} \langle E_1 \Downarrow \mathsf{fun}\, x.\, E_1'\ x\rangle;\ \rhd_i \langle E_2 \Downarrow V_2\rangle;\ \rhd_i \langle (E_1'\ V_2) \Downarrow V\rangle; \ldots;\\
\rhd\ \langle V_2 : T'\rangle;\ \mathsf{abstr}\ E;\ \mathsf{lambda}\ E = \mathsf{lambda}\ E_1';\ \rhd\ \mathsf{all}\, x.\, (x : T'\ \mathsf{imp}\ \langle (E\ x) : T\rangle) \,]\!]\\
\Longrightarrow \rhd\ \langle V : T\rangle.
\end{aligned}$$

The application of the inversion principle `prog.mkH_cases` similar to the one from Section 3 is evident here. MC-Theorem 4 can be applied to conclude that $E = E_1'$. Applying inversion at the SL level gives:

$$\begin{aligned}
[\![\, IH[i+3/n];\ \mathsf{abstr}\ E;\ \rhd_{i+1} \langle E_1 \Downarrow \mathsf{fun}\, x.\, E\ x\rangle;\ \rhd_i \langle E_2 \Downarrow V_2\rangle;\ \rhd_i \langle (E\ V_2) \Downarrow V\rangle; \ldots;\\
\rhd\ \langle V_2 : T'\rangle;\ \forall x.\, (\mathsf{proper}\ x \longrightarrow \rhd\ x : T'\ \mathsf{imp}\ \langle (E\ x) : T\rangle) \,]\!]\\
\Longrightarrow \rhd\ \langle V : T\rangle.
\end{aligned}$$

Inversion cannot be applied directly under the universal quantification and implication of the last premise, so we prove the following inversion lemma, which is also useful for the **fix** case of this proof.

$$\begin{aligned}
[\![\, \forall x.\, \mathsf{proper}\ x \longrightarrow \Gamma \rhd_i (x : T_1\ \mathsf{imp}\ \langle (E\ x) : T_2\rangle) \,]\!] \quad \Longrightarrow \quad & \exists j.\, i = j+1 \wedge \qquad (3)\\
& \forall x.\, \mathsf{proper}\ x \longrightarrow\\
& (x : T_1, \Gamma \rhd_j \langle (E\ x) : T_2\rangle)
\end{aligned}$$

From this lemma, and the fact that $(\mathsf{proper}\ V_2)$ holds by MC-Lemma 14, we obtain:

$$\begin{aligned}
[\![\, IH[i+3/n];\ \mathsf{abstr}\ E;\ \rhd_{i+1} \langle E_1 \Downarrow \mathsf{fun}\, x.\, E\ x\rangle;\ \rhd_i \langle E_2 \Downarrow V_2\rangle;\ \rhd_i \langle (E\ V_2) \Downarrow V\rangle; \ldots;\\
\rhd\ \langle V_2 : T'\rangle;\ ((V_2 : T') \rhd_j \langle (E\ V_2) : T\rangle) \,]\!]\\
\Longrightarrow \rhd\ \langle V : T\rangle.
\end{aligned}$$

Applying the cut rule of MC-Theorem 18 allows us to conclude $\rhd \langle (E\ V_2) : T \rangle$. We can then complete the proof by applying the induction hypothesis a third time using this fact and $\rhd_i \langle (E\ V_2) \Downarrow V \rangle$. □

A key point in this section, perhaps worth repeating, is that the clauses for typing are not inductive and would be rejected in an inductive-based proof assistant, or at best, asserted with no guarantee of consistency. Here, instead, the typing rules are encapsulated into the OL level (the prog predicate) and executed via the SL, so that OL contexts are implicitly represented as SL contexts. Therefore, we are able to reproduce full HOAS proofs, at the price of a small degree of indirectness—the need for an interpreter (the SL) for the prog clauses (the OL). While this may seem at first sight a very high price to pay, since we lose the possibility of attacking the given problem directly within the base calculus and its tools, very simple tactics as discussed next including a few safe[19] additions to Isabelle/HOL's default simplifier and rule set make the use of the SL in OL proofs hardly noticeable, as we explain next.

4.3 Tactical support

We chose to develop Hybrid as a package, rather than a stand-alone system mainly to exploit all the reasoning capabilities that a mature proof assistant can provide: decision procedures, rewrite rules, counter-model checking, and in particular support for tactical theorem proving. 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 toward mechanizing most of boilerplate scripting. We have already explained how to use specific tactics to recognize *proper* terms and *abstractions*. Now, 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[20] consists, at a high-level, of three basic steps: inversion, which subsumes instantiation of (meta-level) eigenvariables as well as (case) analysis on the shape of a given judgment, backchaining (filling, in Twelf's terminology) and recursion. This corresponds to highly stereotyped proof scripts that we have abstracted into:

1. an *inversion* tactic defL_tac, which goes through the SL inverting on the **bc** rule 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;

2. a dual *backchaining* tactic defR_tac, that calls **bc** and the applicable prog rule. The latter is the basic single step into the tactic 2lprolog_tac, which performs automatic depth first search (or other searches supported by Isabelle) on Prolog-like goals;

3. a *complete induction* tactic, to be fired when given the appropriate derivation height by the user and yielding as additional premise the result of the application of the IH.

4.4 A Variation

As mentioned, the main reason to explicitly encode a separate notion of provability is the intrinsic incompatibility of induction with non-stratifiable hypothetical judgments. On the other hand, as remarked in [70], our definition of OL evaluation, though it exploits Hybrid's HOAS to implement OL substitution, makes no use of hypothetical judgments. In fact, our encoding in Figure 3 showed that it is perfectly acceptable to define evaluation of the OL at the meta-level. Now, we can give a modified version of this definition using the new isterm defined at the SL level. The new definition is given in Figure 7. Moreover, it is easy to show (formally) that the encoding in Figure 7 is equivalent to the one in Figure 6:

MC-Theorem 25 *$E \Downarrow V$ if and only if $\rhd_n \langle E \Downarrow V \rangle$.*

Proof: Left-to right holds by straightforward structural induction on evaluation using introduction rules over sequents and prog clauses. The converse is a slightly more delicate complete induction on the height of the derivation, requiring some manual instantiations. □

[19] In Isabelle a rule is considered *safe* roughly if it does not involve backtracking on instantiation of unknowns.
[20] In Abella this is even more apparent.

$$
\begin{array}{rcl}
\textit{Inductive}\ _\Downarrow_ & :: & [\,uexp, uexp\,] \Rightarrow bool \\
[\![\, E_1 \Downarrow \mathsf{fun}\, x.\, E'\, x;\ E_2 \Downarrow V_2;\ (E'\, V_2) \Downarrow V;\ \mathsf{abstr}\, E' \,]\!] & \Longrightarrow & (E_1 \mathbin{@} E_2) \Downarrow V \\
[\![\, \rhd \langle \mathsf{isterm}\ (\mathsf{fun}\, x.\, E\ x) \rangle;\ \mathsf{abstr}\, E \,]\!] & \Longrightarrow & \mathsf{fun}\, x.\, E\ x \Downarrow \mathsf{fun}\, x.\, E\ x \\
[\![\, E\ (\mathsf{fix}\, x.\, E\ x) \Downarrow V;\ \rhd \mathsf{isterm}\ (\mathsf{fix}\, x.\, E\ x);\ \mathsf{abstr}\, E \,]\!] & \Longrightarrow & \mathsf{fix}\, x.\, E\ x \Downarrow V
\end{array}
$$

Figure 7: Alternate HOAS encoding of big step evaluation

The same remark applies also to hypothetical and parametric judgments, provided they are stratified (see the previously cited definition of applicative bisimulation). This suggests that we can, in this case, take a different approach from McDowell & Miller's architecture [62] and opt to delegate to the OL level *only* those judgments, such as typing, that would not be inductive at the meta-level. This has the benefit of limiting the indirectness of using an explicit SL. Moreover, it has the further advantage of replacing complete induction with structural induction, which is better behaved from a proof-search point of view. Complete induction, in fact, places an additional burden on the user by requiring him/her to provide the correct instantiation for the height of the derivation in question, so that the inductive hypothesis can be fired. While this is not an intellectual issue, it often limits the possibility of a complete, *i.e.*, without user intervention, mechanization of a proof via the automatic tools provided by the proof assistant.

As it turns out, this approach is again reminiscent of a fairly old idea from the theory of logic programming, namely the *amalgamation* of object and meta-language as initially suggested in [13], where clauses can be written interspersing ordinary Prolog predicates with calls to a specific meta-interpreter of the `demo` sort. This clearly also pertains to goals, *i.e.*, in our setting, theorems: subject reduction at the meta-level (a.k.a. amalgamated subject reduction) has the form:

MC-Theorem 26 (meta_subject_reduction)

$$E \Downarrow V \Longrightarrow \forall T.\, (\rhd \langle E : T \rangle) \longrightarrow (\rhd \langle V : T \rangle)$$

Proof: The proof is similar but slightly simpler than the proof of MC-Theorem 24. Instead of complete induction, we proceed by structural induction on the evaluation judgment, which breaks the proof into three cases. We again consider the application case:

$$
\begin{array}{l}
[\![\, IH_1;\ IH_2;\ IH_3;\ \mathsf{abstr}\, E_1';\ (E_1 \Downarrow \mathsf{fun}\, x.\, E_1'\ x);\ (E_2 \Downarrow V_2); \\
\qquad ((E_1'\ V_2) \Downarrow V);\ \rhd \langle (E_1 \mathbin{@} E_2) : T \rangle \,]\!] \Longrightarrow \rhd \langle V : T \rangle
\end{array}
$$

where IH_1, IH_2, and IH_3 are the following three induction hypotheses:

$$
\begin{array}{ll}
IH_1: & \forall T.\, \rhd \langle E_1 : T \rangle \Longrightarrow \rhd \langle (\mathsf{fun}\, x.\, E_1'\ x) : T \rangle \\
IH_2: & \forall T.\, \rhd \langle E_2 : T \rangle \Longrightarrow \rhd \langle V_2 : T \rangle \\
IH_3: & \forall T.\, \rhd \langle (E_1'\ V_2) : T \rangle \Longrightarrow \rhd \langle V : T \rangle
\end{array}
$$

This subgoal corresponds to subgoal (2) in the proof of MC-Theorem 24, with several differences. For instance, subgoal (2) was obtained by an application of complete induction followed by inversion on the OL and SL, while the above subgoal is a direct result of applying structural induction. Also, although both subgoals have three evaluation premises, in (2) they are inside conjunction at the SL level. Finally, the general induction hypothesis *IH* on natural numbers in (2) is replaced by three induction hypotheses here, generated from the premises of the meta-level definition of the evaluation rule for application. The remaining steps of the proof of this case are essentially the same as the steps for MC-Theorem 24. Inversion on the typing judgment is used exactly as before since in both proofs, typing is expressed via the SL. Also, the three induction hypotheses in this proof are used to reach the same conclusions as were obtained using the single induction hypothesis three times in the previous proof. □

Now that we have seen some proofs of properties of OLs, we can ask what the minimal set of theorems and tactics is that the two-level architecture needs from Hybrid. The answer is: very little. Essentially all we need is the quasi-freeness properties of the Hybrid type, which are inherited from the OL:

- clash rules to rule out impossible cases in elimination rules;
- injectivity facts, all going back to *abstr_lam_simp* to simplify equations of the form $\mathsf{lambda}\ E = \mathsf{lambda}\ F$ for second-order functions E and F;
- an abstraction solver.[21]

The reader may find in [70] other examples, such as the verification of properties of compilation, of encoding OLs using inductive predicates (types) at the meta-level for all stratifiable object-level judgments. However, this style of reasoning is viable only when there is a substantial coincidence between the meta-logical properties of the SL and the ambient (meta) logic. Were such properties to clash with an encoding that could benefit from being driven by a more exotic logic, then *all* OL predicates will have to be embedded as prog clauses. This, it may be argued, is a relatively small price to pay for the possibility of adopting an SL that better fits the logical peculiarities of interesting OLs, as we investigate next.

5 Introduction to Ordered Linear Logic

As supported by a large body of work [84], frameworks based on *intuitionistic* logic have been fairly fruitful; however, it so happens that the *structural* properties of the framework, such as weakening, contraction and exchange, are inherited by the object-level encodings. Note that we have argued that one of the keys to the success of an encoding lies in the ability of specifying judgments "in-a-context" of an OL exploiting the context of the SL itself; however those properties may not always be appropriate for every domain we want to investigate.

One case in point is the meta-theory of languages with imperative features, where the notion of (up-datable) *state* is paramount. To fix ideas and not stray away too much from the previous example, let us consider the case of the static and dynamic semantics of a functional programming language with imperative features, say Mini-ML with references (MLR). Judgments may now incorporate some representation of *memory*, in form of a store or a heap. How can HOAS effectively deal with this issue?

1. The traditional approach is to represent the memory as a (finite) map, possibly realized as an association list of `cells` and `values`; a command such as assignment then is encoded as functional (list) update. While this is theoretically non-problematic, it introduces entities that are foreign to the mathematics of the problem and which bring their own numerous, albeit trivial, proof obligations. This mixture of intuitionistic meta-logic, (full) HOAS and list-based techniques to cope with mutable data has been able to handle significant case studies: for example, Crary and Sarkar's proof of soundness for foundational certified code in typed assembly language for the x86 architecture [24] as well the more recent attempt by Lee *et al.* [57] to verify an internal language for SML, which would be elaborated into a type-theoretic reconstruction of the "full" Definition of Standard ML. As the authors have themselves remarked (Harper's invited speech at ICFP), this often required very hard work, especially the handling of the imperative features. To give a very rough estimate, the Twelf formalization of subject reduction for Winskel's *IMP* [77] (implemented by Simmons and available from the Twelf Wiki at `http://twelf.plparty.org/wiki/Mutable_state`) is *more than one* order of magnitude longer that the analogous theorem for the arguably more complex Mini-ML language, present in the Twelf distribution. Even a cursory inspection shows the preponderance of code and lemmas dealing with lists: projecting, appending, updating *etc.* As a side remark, there is also empirical evidence that the "abuse" of list-based encodings may impair efficiency when logical frameworks are used in applications such as PCC [31].
2. The other possibility is instead encoding the store as an *intuitionistic* meta-context of entries such as `contains c v` and "deal with it". Obviously, a naive encoding is doomed to be inadequate, since once an assumption is added, it cannot be "logically" removed. The idea is then to add judgments to enforce the linear use of each assumption. The rules in the operational semantics that inspect or modify the store must check that the linearity invariants are maintained. Further, this modification reverberates onto the statement of the theorems in question, *viz.* subject reduction will also state that linearity is preserved by evaluation. An early example of this

[21] Again, this is not needed anymore in a newer version of Isabelle/HOL and of our package [73].

approach can be found in the formalization of structural cut-elimination for linear logic [83], where Pfenning also points out how more complicated adequacy proofs may become.[22] While this is in principle possible, we find it complex and unpleasant enough to look for an alternative solution.

It has been frequently observed that a declarative and very elegant representation of the store may instead rely on a *volatile* notion of context (as opposed to *permanent* ones, such as in LF). A form of *linear logic* may then be the natural choice, since it offers a notion of context where each assumption must be used exactly once; a declarative encoding of store update can be obtained via linear operations that, by accessing the context, consume the old assumption and insert the new one. This is one of the motivations for proposing frameworks based on linear logics (see [67] for an overview) such as Lolli [50], Forum [66], and LLF [18], a *conservative* extension of LF with multiplicative implication, additive conjunction, and unit. Yet, at the time of writing this article, work on the *automation* of reasoning in such frameworks is still in its infancy [60] and may take other directions, such as *hybridizing* – in the sense of hybrid logics – LF [99]. In other words, the literature offers only a few formalized meta-theoretical investigations with linear logic as a framework, an impressive one being the elegant encoding of type preservation of MLR in LLF [18]. However, none of them comes with anything like a formal certification of correctness that would make people *believe* they are in the presence of a proof. Encoding in LLF lacks an analogue of Twelf's totality checker, which ensures that the relational encoding of the proof of a theorem is in effect a realizer for the latter. Moreover this effort may be reserved to LLF's extension, the Concurrent Logical Framework [111]. A $FO\lambda^{\Delta IN}$ proof of a similar result is claimed in [62], but not only the proof is not available, but it has been implemented with Eriksson's Pi, a proof checker [32] for the theory of partial inductive definitions, another software system that seems not to be available anymore.

This alone would more than justify the use of a fragment of linear logic as an SL on top of Hybrid, whose foundation, we argue, is not under discussion. Therefore, in this section, we apply the methodology of refining a logical framework in a *conservative* way, so as to capture different object-level phenomena at the right level of abstraction. Conservativity here guarantees that if a new feature (such as linearity) is not required, it does not interfere with the original system.

However, we have a duplicitous aim. First, push our agenda by showing the flexibility of the two-level architecture in changing SL in order to have a better match with the encoding on hand. Second, we want to go beyond linear logic, known as the logic of state, towards a logic for the notion of *order*.

To bring this point home we observe that the encoding of type preservation of MLR is naturally based on a *continuation* model, where evaluation is sequentialized: an instruction is executed in the context of a continuation describing the rest of the computation and eventually returning an answer; in MLR's case that is taken to be a value together with a new store. As we detail in Section 5.2, this entails the introduction of additional layers (instructions, continuations, machine states) not found in a text-book presentation of a big-step operational semantics for ML with references [90]. Moreover, those entities need to be typed, in a sense complicating the set-up. So it seems that the internalization of the store that linear logic offers is balanced off by the introduction of this new machinery, leaving the reader to wonder whether this effort is in fact worthwhile.

However, continuations follow an order, *viz.* a stack-like discipline; were we able to also *internalize* the notion of order, we would be able to simplify the presentation, and hence, the verification of properties of the continuation abstract machine itself, taking an additional step on the declarative ladder. Our contribution here to the semantics of continuation machines is, somewhat paradoxically, to dispose of the notion of continuation itself via internalization in an ordered context, in analogy with how the notion of state is realized in the linear context. In particular, the ordered context is used to encode directly the stack of continuations to be evaluated.

Therefore we adopt an *ordered logical framework* (OLF) [93], by which we mean, in this incarnation, a second-order minimal ordered linear sequent calculus. Ordered (formerly known as non-commutative) linear logic [95] combines reasoning with unrestricted, linear and ordered hypotheses. Unrestricted (*i.e.*, intuitionistic) hypotheses may be used arbitrarily often, or not at all regardless of the order in which they were assumed. Linear hypotheses must be used exactly once, also without regard to the order of their assumption. Ordered hypotheses must be used exactly once, subject to the order in which they are assumed.

This additional expressive power allows the logic to handle directly the notion of *stack*. Stacks of course are ubiquitous in computer science and in particular when dealing with abstract and virtual machines. OLF has been

[22] For a more pedagogical case study *cf.* Crary's entry at `http://twelf.plparty.org/wiki/Linear_logic`.

$$\frac{}{\Gamma; A \longrightarrow_\Pi A}\,\mathbf{init}_\Omega \qquad \frac{}{\Gamma, A; \cdot \longrightarrow_\Pi A}\,\mathbf{init}_\Gamma$$

$$\frac{(A,\Gamma);\Omega \longrightarrow_\Pi G}{\Gamma;\Omega \longrightarrow_\Pi A \to G}\,{\to}_R \qquad \frac{\Gamma;(\Omega,A) \longrightarrow_\Pi G}{\Gamma;\Omega \longrightarrow_\Pi A \twoheadrightarrow G}\,{\twoheadrightarrow}_R$$

$$\frac{\Gamma;\Omega \longrightarrow_\Pi G_1 \qquad \Gamma;\Omega \longrightarrow_\Pi G_2}{\Gamma;\Omega \longrightarrow_\Pi G_1 \wedge G_2}\,{\wedge}_R$$

$$\frac{}{\Gamma;\Omega \longrightarrow_\Pi \top}\,\top_R \qquad \frac{\Gamma;\Omega \longrightarrow_\Pi G[a/x]}{\Gamma;\Omega \longrightarrow_\Pi \forall x.\, G}\,\forall^a_R$$

$$\frac{\begin{array}{l}(A \longleftarrow [G_1,\ldots,G_m] \mid [G'_1,\ldots,G'_n]) \in [\Pi] \\ \Gamma;\cdot \longrightarrow_\Pi G_1 \ldots \Gamma;\cdot \longrightarrow_\Pi G_m \\ \Gamma;\Omega_1 \longrightarrow_\Pi G'_1 \ldots \Gamma;\Omega_n \longrightarrow_\Pi G'_n\end{array}}{\Gamma;\Omega_n \ldots \Omega_1 \longrightarrow_\Pi A}\,\mathbf{bc}$$

Figure 8: Sequent rules for $\text{Olli}^2_{\to\!\!\!\!\to}$

previously applied to the meta-theory of programming languages, but only in paper and pencil proofs: Polakow and Pfenning [96] have used OLF to formally show that terms resulting from a CPS translation obey "stackability" and linearity properties [27]. Polakow and Yi [97] later extended these techniques to languages with exceptions. Remarkably, the formalization in OLF provides a simple proof of what is usually demonstrated via more complex means, *i.e.*, an argument by logical relations. Polakow [93] has also investigated proof-search and defined a first-order logic programming language with ordered hypotheses, called *Olli*, based on the paradigm of abstract logic programming and *uniform* proofs, from which we draw inspiration for our ordered SL.

We exemplify this approach by implementing a fragment of Polakow's ordered logic as an SL and test it with a proof of type preservation of a continuation machine for Mini-ML, as we sketched in [74]. For the sake of presentation we shall deal with a call-by-name operational semantics.

It wouldn't be unreasonable to use MLR as a test case, where all the three different contexts would play a part. However, linearity has already been thoroughly studied, while we wish to analyze ordered assumptions in isolation, and for that aim, a basic continuation machine will suffice. Further, although the SL implementation handles all of second-order Olli and in particular it proves cut-elimination for the whole calculus, we will omit references to the (unordered) linear context and linear implication (lollipop), as well to to ordered left implication, since they do not play any role in this case-study.

5.1 Encoding the Specification Logic

We call our specification logic $\text{Olli}^2_{\to\!\!\!\!\to}$, as it corresponds to the aforementioned fragment of Olli, where '$\twoheadrightarrow$' denotes right-ordered implication. We follow [74] again in representing the syntax as:

$$\begin{array}{rrcl} \text{Goals} & G & ::= & A \mid A \to G \mid A \twoheadrightarrow G \mid G_1 \wedge G_2 \mid \top \mid \forall^\tau x.\, G \\ \text{Clauses} & P & ::= & \forall(A \longleftarrow [G_1,\ldots,G_m] \mid [G'_1,\ldots,G'_n]) \end{array}$$

The body of a clause $\forall(A \longleftarrow [G_1,\ldots,G_m] \mid [G'_1,\ldots,G'_n])$ consists of two *lists*, the first one of intuitionistic goals, the other of ordered ones. It represents the "logical compilation" of the formula $\forall(G_m \to \ldots \to G_1 \to G'_n \twoheadrightarrow \ldots \twoheadrightarrow G'_1 \twoheadrightarrow A)$.

We choose this compilation to emphasize that if one views the calculus as a non-deterministic logic programming interpreter, the latter would solve subgoals from innermost to outermost. Note also that this notion of clause makes additive conjunction useless, although we allow it in goals for a matter of style and consistency with the previous sections.

Our sequents have the form:

$$\Gamma; \Omega \longrightarrow_{\Pi} G$$

where Π contains the program clauses, which are unrestricted (*i.e.*, they can be used an arbitrary number of times), Γ contains unrestricted atoms, Ω contains ordered atoms and G is the formula to be derived. Contexts are lists of hypotheses, where we overload the comma to denote adjoining an element to a list at both ends. To simplify matters further, we leave eigenvariable signatures implicit. One may think of the two contexts as one big context where the ordered hypotheses are in a fixed relative order, while the intuitionistic ones may float, copy or delete themselves. The calculus is depicted in Figure 8. Again in this fragment of the logic, implications have only atomic antecedents. There are obviously two implication introduction rules, where in rule $\twoheadrightarrow_R$ the antecedent A is appended to the *right* of Ω, while in the other rule we have (A, Γ), but it could have been the other way around, since here the order does not matter. Then, we have all the other usual right sequent rules to break down the goal and they all behave additively. Note how the $\top_R$ rule can be used in discharging any unused ordered assumptions. For atomic goals there are two initial sequent rules, for the leaves of the derivation: $\mathbf{init}_{\Omega}$ enforces linearity requiring Ω to be the singleton list $[A]$, while $\mathbf{init}_{\Gamma}$ demands that all ordered assumptions have been consumed. Additionally, there is a single backchaining rule that simultaneously chooses a program formula to focus upon – the *focusing* formula in the terminology of [93] – and derives all the ensuing subgoals; rule $(\mathbf{bc})$ is applied provided there is an *instance* $A \longleftarrow [G_1 \ldots G_m] \mid [G'_1 \ldots G'_n]$ of a clause in the program Π. Note that the rule assumes that every program clause must be placed to the *left* of the ordered context. This assumption is valid for our fragment of the logic because it only contains right ordered implications ($\twoheadrightarrow$) and the ordered context is restricted to atomic formulas. Furthermore, the ordering of the Ω_i in the conclusion of the rule is forced by our compilation of the program clauses. We leave to the keen reader the task to connect formally our backchain rule to the focused uniform proof system of *op. cit.* [93].

We encode this logical language extending the datatype from Section 4.1 with right implication, where again outermost universal quantifiers will be left implicit in clauses.

$$\textit{datatype} \quad oo \quad = \quad \cdots \mid atm \twoheadrightarrow oo$$

Our encoding of the $\text{Olli}^2_{\twoheadrightarrow}$ sequent calculus is based on [62]'s approach to linearity, adapting it to right ordered implication. We use three mutually inductive definitions, motivated by the compilation of the body of clauses into additive and multiplicative lists:

$\Gamma \mid \Omega \rhd_n G$	::	$[atm\ list, atm\ list, nat, oo] \Rightarrow bool$	goal G has an ordered linear derivation from Γ and Ω of height n
$\Gamma \blacktriangleright_n Gs$	::	$[atm\ list, nat, oo\ list] \Rightarrow bool$	list of goals Gs is additively provable from Γ *etc.*
$\Gamma \mid \Omega \blacktriangleright_n Gs$	::	$[atm\ list, atm\ list, nat, oo\ list] \Rightarrow bool$	list of goals Gs is multiplicatively consumable given Γ and Ω *etc.*

The rendering of the first judgment is completely unsurprising,[23] except, perhaps, for the backchain rule, which calls the list predicates required to recur on the body of a clause:

$$[\![\, (A \longleftarrow O_L \mid I_L)\ ;\ \Gamma \mid \Omega \blacktriangleright_n O_L\ ;\ \Gamma \blacktriangleright_n I_L \,]\!] \Longrightarrow \Gamma \mid \Omega \rhd_{n+1} \langle A \rangle$$

The notation $A \longleftarrow O_L \mid I_L$ corresponds to the inductive definition of a set prog this time of type $[atm, oo\ list, oo\ list] \Rightarrow bool$, see Figure 12. Backchaining uses the two list judgments to encode, as we anticipated, execution of the (compiled) body of the focused clause. Intuitionistic list provability is just an additive recursion through the list of intuitionistic subgoals:

$$\begin{array}{rcl} & \Longrightarrow & \Gamma \blacktriangleright_n [\,] \\ [\![\, \Gamma \mid [\,] \rhd_n G\ ;\ \Gamma \blacktriangleright_n Gs \,]\!] & \Longrightarrow & \Gamma \blacktriangleright_{n+1} (G, Gs) \end{array}$$

[23] As a further simplification, the encoding of the $\forall_R$ rule will *not* introduce the proper assumption, but the reader should keep in mind the fact that morally every eigenvariable is indeed proper.

Ordered list consumption involves an analogous recursion, but it behaves multiplicatively w.r.t. the ordered context. Reading the rule bottom up, the current ordered context Ω is non-deterministically split into two ordered parts, one for the head Ω_G and one Ω_R for the rest of the list of subgoals.

$$\begin{array}{c} \Longrightarrow \Gamma \mid [\,] \blacktriangleright_n [\,] \\ [\![\, \mathsf{osplit}\ \Omega\ \ \Omega_R\ \ \Omega_G\ ;\ \Gamma \mid \Omega_G \rhd_n G\ ;\ \Gamma \mid \Omega_R \blacktriangleright_n Gs\ ]\!] \\ \Longrightarrow \Gamma \mid \Omega \blacktriangleright_{n+1} (G, Gs) \end{array}$$

Therefore the judgment relies on the inductive definition of a predicate for order-preserving splitting of a context. This corresponds to the usual logic programming predicate $\mathit{append}(\Omega_R, \Omega_G, \Omega)$ called with mode $\mathit{append}(-,-,+)$.

$$\begin{array}{rcl} & \Longrightarrow & \mathsf{osplit}\ \Omega\ \ [\,]\ \ \Omega \\ \mathsf{osplit}\ \Omega_1\ \ \Omega_2\ \ \Omega_3 & \Longrightarrow & \mathsf{osplit}\ (A, \Omega_1)\ \ (A, \Omega_2)\ \ \Omega_3 \end{array}$$

The rest of the sequent rules are encoded similarly to the previous SL (Figure 5) and the details are here omitted (though left to the web appendix of the paper). Again we define $\Gamma \mid \Omega \rhd G$ iff there exist n such that $\Gamma \mid \Omega \rhd_n G$ and simply $\rhd G$ iff $[\,] \mid [\,] \rhd G$. Similarly for the other judgments.

MC-Theorem 27 (Structural Rules) *The following rules are admissible:*

- *Weakening for numerical bounds:*
  1. $[\![\, \Gamma \mid \Omega \rhd_n G;\ n < m\ ]\!] \Longrightarrow \Gamma \mid \Omega \rhd_m G$
  2. $[\![\, \Gamma \mid \Omega \blacktriangleright_n Gs;\ n < m\ ]\!] \Longrightarrow \Gamma \mid \Omega \blacktriangleright_m Gs$
  3. $[\![\, \Gamma \blacktriangleright_n Gs;\ n < m\ ]\!] \Longrightarrow \Gamma \blacktriangleright_m Gs.$
- *Context weakening, where* $(\mathit{set}\ \Gamma)$ *denotes the set underlying the context* Γ.
  1. $[\![\, \Gamma \mid \Omega \rhd G;\ \mathit{set}\ \Gamma \subseteq \mathit{set}\ \Gamma'\ ]\!] \Longrightarrow \Gamma' \mid \Omega \rhd G$
  2. $[\![\, \Gamma \mid \Omega \blacktriangleright Gs;\ \mathit{set}\ \Gamma \subseteq \mathit{set}\ \Gamma'\ ]\!] \Longrightarrow \Gamma' \mid \Omega \blacktriangleright Gs$
  3. $[\![\, \Gamma \blacktriangleright Gs;\ \mathit{set}\ \Gamma \subseteq \mathit{set}\ \Gamma'\ ]\!] \Longrightarrow \Gamma' \blacktriangleright Gs.$
- *Intuitionistic atomic cut:*
  1. $[\![\, \Gamma \mid \Omega \rhd_i G;\ \mathit{set}\ \Gamma = \mathit{set}\ (A, \Gamma');\ \Gamma' \mid [\,] \rhd_j \langle A \rangle\ ]\!] \Longrightarrow \Gamma' \mid \Omega \rhd_{i+j} G.$
  2. $[\![\, \Gamma \mid \Omega \blacktriangleright_i Gs;\ \mathit{set}\ \Gamma = \mathit{set}\ (A, \Gamma');\ \Gamma' \mid [\,] \rhd_j \langle A \rangle\ ]\!] \Longrightarrow \Gamma' \mid \Omega \blacktriangleright_{i+j} Gs.$
  3. $[\![\, \Gamma \blacktriangleright_i Gs;\ \mathit{set}\ \Gamma = \mathit{set}\ (A, \Gamma');\ \Gamma' \mid [\,] \rhd_j \langle A \rangle\ ]\!] \Longrightarrow \Gamma' \blacktriangleright_{i+j} Gs.$

Proof: All the proofs are by mutual structural induction on the three sequents judgments. For the two forms of weakening, all it takes is a call to Isabelle/HOL's classical reasoner. Cut requires a little care in the implicational cases, but nevertheless it does not involve more then two dozens instructions. □

Although the sequent calculus in [93] enjoys other forms of cut-elimination, the following:

MC-Corollary 28 (seq_cut)

$$[\![\, A, \Gamma \mid \Omega \rhd G;\ \Gamma \rhd \langle A \rangle\ ]\!] \quad \Longrightarrow \quad \Gamma \mid \Omega \rhd G$$

is enough for the sake of the type preservation proof (MC-Theorem 32). Further, admissibility of contraction and exchange for the intuitionistic context is a consequence of context weakening.

5.2 A Continuation Machine and its Operational Semantics

We avail ourselves of the continuation machine for Mini-ML formulated in [85] (Chapters 6.5 and 6.6), which we refer to for motivation and additional details. We use the same language and we repeat it here for convenience:

$$\begin{array}{rrcl} \textit{Types} & \tau & ::= & i \mid \tau \to \tau' \\ \text{Expressions} & e & ::= & x \mid \mathbf{fun}\, x.\, e \mid e_1 \bullet e_2 \mid \mathbf{fix}\, x.\, e \end{array}$$

The main judgment $s \hookrightarrow s'$ (Figure 9) describes how the state of the machine evolves into a successor state s' in a *small-step* style. The machine selects an expression to be executed and a *continuation* K, which contains all the information required to carry on the execution. To achieve this we use the notion of *instruction*, *e.g.*, an intermediate command that links an expression to its value. The continuation is either empty (**init**) or it has the form of a stack $(K; \lambda\, x.\, i)$, each item of which (but the top) is a *function* from values to instructions. Instruction (**ev** e) starts the first step of the computation, while (**return** v) tells the current continuation to apply to the top element on the continuation stack the newly found value. Other instructions sequentialize the evaluation of subexpressions of constructs with more than one argument; in our language, in the case of application, the second argument is postponed until the first is evaluated completely. This yields the following categories for the syntax of the machine:

$$\begin{array}{rrcl} \text{Instructions} & i & ::= & \mathbf{ev}\, e \mid \mathbf{return}\, v \mid \mathbf{app}_1\, v_1\, e_2 \\ \text{Continuations} & K & ::= & \mathbf{init} \mid K; \lambda\, x.\, i \\ \text{Machine States} & s & ::= & K \diamond i \mid \mathbf{answer}\, v \end{array}$$

$$\begin{array}{rcl} \textbf{st_init} & :: & \mathbf{init} \diamond \mathbf{return}\, v \hookrightarrow \mathbf{answer}\, v \\ \textbf{st_return} & :: & K; \lambda\, x.\, i \diamond \mathbf{return}\, v \hookrightarrow K \diamond i[v/x] \\ \textbf{st_fun} & :: & K \diamond \mathbf{ev}\,(\mathbf{fun}\, x.\, e) \hookrightarrow K \diamond \mathbf{return}\,(\mathbf{fun}\, x.\, e) \\ \textbf{st_fix} & :: & K \diamond \mathbf{ev}\,(\mathbf{fix}\, x.\, e) \hookrightarrow K \diamond \mathbf{ev}\,(e[\mathbf{fun}\, x.\, e/x]) \\ \textbf{st_app} & :: & K \diamond \mathbf{ev}\,(e_1 \bullet e_2) \hookrightarrow K; \lambda\, x_1.\, \mathbf{app}_1\, x_1\, e_2 \diamond \mathbf{ev}\, e_1 \\ \textbf{st_app}_\mathbf{1} & :: & K \diamond \mathbf{app}_1\,(\mathbf{fun}\, x.\, e)\, e_2 \hookrightarrow K \diamond \mathbf{ev}\, e[e_2/x] \end{array}$$

Figure 9: Transition rules for machine states

The formulation of the subject reduction property of this machine follows the statement in [18], although we consider sequences of transitions by taking the reflexive-transitive closure $\hookrightarrow^*$ of the small-step relation, and a top level initialization rule **cev** (Figure 10). Of course, we need to add typing judgments for the new syntactic categories, namely instructions, continuations and states. These can be found in Figure 11, whereas we refer the reader to Figure 2 as far as typing of expressions goes.

Theorem 29 $K \diamond i \hookrightarrow^* \mathbf{answer}\, v$ *and* $\Gamma \vdash_i i : \tau_1$ *and* $\vdash_K K : \tau_1 \to \tau_2$ *implies* $\cdot \vdash_e v : \tau_2$.

Proof: By induction on the length of the execution path using inversion properties of the typing judgments. □

Corollary 30 (Subject Reduction) $e \overset{c}{\hookrightarrow} v$ *and* $\cdot \vdash_e e : \tau$ *entails* $\cdot \vdash_e v : \tau$.

$$\frac{}{s \hookrightarrow^* s}\,\textbf{stop} \qquad \frac{s_1 \hookrightarrow s_2 \qquad s_2 \hookrightarrow^* s_3}{s_1 \hookrightarrow^* s_3}\,\textbf{step}$$

$$\frac{\textbf{init} \diamond \textbf{ev}\, e \hookrightarrow^* \textbf{answer}\, v}{e \overset{c}{\hookrightarrow} v}\,\textbf{cev}$$

Figure 10: Top level transition rules

$$\frac{\Gamma \vdash_e e : \tau}{\Gamma \vdash_i \textbf{ev}\, e : \tau}\,ofI_\textbf{ev} \qquad \frac{\Gamma \vdash_e v : \tau}{\Gamma \vdash_i \textbf{return}\, v : \tau}\,ofI_\textbf{return}$$

$$\frac{\Gamma \vdash_e e_1 : \tau' \to \tau \qquad \Gamma \vdash_e e_2 : \tau'}{\Gamma \vdash_i \textbf{app}_1\, e_1\, e_2 : \tau}\,ofI_\textbf{app}_1$$

. . . . . .

$$\frac{}{\vdash_K \textbf{init} : \tau \to \tau}\,ofK_\textbf{init} \qquad \frac{x{:}\tau_1 \vdash_i i : \tau \qquad \vdash_K K : \tau \to \tau_2}{\vdash_K K; \lambda\, x.\, i : \tau_1 \to \tau_2}\,ofK_\textbf{cont}$$

. . . . . .

$$\frac{\vdash_i i : \tau_1 \qquad \vdash_K K : \tau_1 \to \tau_2}{\vdash_s K \diamond i : \tau_2}\,ofs_\diamond \qquad \frac{\vdash_e v : \tau}{\vdash_s \textbf{answer}\, v : \tau}\,ofs_\textbf{answer}$$

Figure 11: Typing rules for the continuation machine

As a matter of fact we could have obtained the same result by showing the *soundness* of the operational semantics of the continuation machine w.r.t. big step evaluation, *viz.* that $e \overset{c}{\hookrightarrow} v$ entails $e \Downarrow v$ (see Theorem 6.25 in [85]) and then appealing to type preservation of the latter. That would be another interesting case study: the equivalence of the two operational semantics (thoroughly investigated by Pfenning in Chapter 6 *op. cit.* but in the intuitionistic setting of LF), to gauge what the "Olli" approach would buy us.

5.3 Encoding the Object Logic

We now show how to write the operational semantics of the continuation machine as an Olli program, or more precisely as $\text{Olli}^2_{\twoheadrightarrow}$ OL clauses. Rather than building an explicit stack-like structure to represent the continuation K, we will simply *store instructions in the ordered context*. This is particularly striking as we end up mapping machine states not into OL data, but OL *provability*. In particular we will use the following representation to encode machine states:

$$K \diamond i \qquad \rightsquigarrow \qquad \Gamma \mid \ulcorner K \urcorner \rhd \langle \mathsf{ex} \ulcorner i \urcorner \rangle$$

where $\ulcorner K \urcorner$ is the representation, described below, of the continuation (stack) K and $\ulcorner i \urcorner$ the obvious representation of instruction.[24] In fact, if we retain the usual *type* abbreviation

$$uexp \quad == \quad con\ expr$$

the encoding of instructions can be simply realized with an Isabelle/HOL datatype, whose adequacy is standard and will not be formally discussed:

$$datatype \quad instr \quad = \quad \mathsf{ev}\ uexp \mid \mathsf{return}\ uexp \mid \mathsf{app}_1\ uexp\ uexp$$

To describe the encoding of continuations, we use our datatype *atm*, which describes the atomic formulas of the OL. This time, it is more interesting and consists of:

$$\begin{array}{lll} datatype \quad atm & = & \mathsf{ceval}\ uexp\ uexp \mid \mathsf{ex}\ instr \mid \mathsf{init}\ uexp \\ & & \mid \mathsf{cont}\ (uexp \Rightarrow instr) \mid uexp : tp \\ & & \mid \mathsf{ofl}\ instr\ tp \mid \mathsf{ofK}\ tp \end{array}$$

We have *atoms* to describe the initial continuation "init " of type $uexp \Rightarrow atm$, the continuation which just returns its value. Otherwise K is a ordered context of atoms "$\mathsf{cont}\ K$" of type $(uexp \Rightarrow instr) \Rightarrow atm$. The top level of evaluation ($\mathsf{ceval}\ \ulcorner e \urcorner\ \ulcorner v \urcorner$) unfolds to the initial goal $\mathsf{init}\ \ulcorner v \urcorner \twoheadrightarrow \mathsf{ex}\ (\mathtt{ev}\ \ulcorner e \urcorner)$; our program will evaluate the expression $\ulcorner e \urcorner$ and instantiate $\ulcorner v \urcorner$ with the resulting value. In other words, we evaluate e with the initial continuation. Overall, the instructions are treated as follows: the goal $\mathsf{ex}\ (\mathtt{return}\ \ulcorner v \urcorner)$ is intended to mean: pass v to the top continuation on the stack (i.e. the rightmost element in the ordered context): the instruction in the goal $\mathsf{ex}\ (\mathtt{app}_1\ \ulcorner v_1 \urcorner\ \ulcorner e_2 \urcorner)$ sequentializes the evaluation of application.

We have the following representations of machine states:

$$\mathbf{init} \diamond \mathbf{return}\, v \quad \rightsquigarrow \quad \Gamma \mid [\mathsf{init}\ W] \rhd \langle \mathsf{ex}\ (\mathtt{return}\ \ulcorner v \urcorner) \rangle$$

where the logic variable W is the final answer;

$$K; \lambda\, x.\, i \diamond \mathbf{return}\, v \rightsquigarrow \quad \Gamma \mid (\ulcorner K \urcorner, \mathsf{cont}\ (\lambda\, x. \ulcorner i \urcorner)) \rhd \langle \mathsf{ex}\ (\mathtt{return}\ \ulcorner v \urcorner) \rangle$$

where the ordering constraints force the proof of $\mathsf{ex}\ (\mathtt{return}\ \ulcorner v \urcorner)$ to focus on the rightmost ordered formula.

We can now give the clauses for the OL deductive systems in Figure 12, starting with typing. These judgments are intuitionistic, except typing of continuations. The judgments for expressions and instructions directly encode the corresponding judgments and derivation rules. The judgments for continuations differ from their analogs in Figure 11 in that there is no explicit continuation to type; instead, the continuation to be typed is in the ordered context. Thus, these judgments must first get a continuation from the ordered context and then proceed to type it.

The evaluation clauses of the program fully take advantage of ordered contexts. The first one is just a wrapper to put queries into the correct form. The rest directly mirrors the machine transition rules.

We remark that the choice of encoding instructions as an Isabelle/HOL datatype has the beneficial effect of eliminating the **abstr** on the (encoding of the) *ofK*_**cont** and **st_return** rules since the continuation K is of type $uexp \Rightarrow instr$ and this function space is not problematic.

A sample derivation is probably in order and so it follows as MC-Lemma 31. Note that as far as examples of evaluations go, this is not far away from total triviality, being the evaluation of something which is already a value. However, our intention here is not to illustrate the sequentialization of evaluation steps typical of a continuation machine (for which we refer again to [85]); rather we aim to emphasize the role of the ordered context, in particular the effect of non-deterministic splitting on the complexity of proof search.

MC-Lemma 31 $\exists V.\ \rhd\ \langle \mathsf{ceval}\ (\ \mathsf{fun}\, x.\, x)\ V \rangle$

[24] The reader may be relieved to learn that, at this late stage of the paper, we will be much more informal with the issue of the adequacy of this encoding, mainly trying to convey the general intuition. This is also notationally signaled by dropping the somewhat heavy notation $\varepsilon_{\cdot}(\cdot)$ for the lighter $\ulcorner \cdot \urcorner$. It is likely that the faithfulness of our representation could be obtained following the approach in [18]. See for example Theorem 3.4.

$$
\begin{array}{rcl}
\textit{Inductive}\ _ \longleftarrow _ \mid _ & :: & [\textit{atm}, \infty\, \textit{list}, \infty\, \textit{list}] \Rightarrow \textit{bool} \\
 & \Longrightarrow & (E_1 \mathbin{@} E_2) : T \longleftarrow [\,] \mid [\langle E_1 : (T' \to T)\rangle, \langle E_2 : T'\rangle] \\
[\![\, \mathsf{abstr}\ E \,]\!] & \Longrightarrow & (\mathsf{fun}\, x.\, E\ x) : (T_1 \to T_2) \longleftarrow [\,] \mid [\mathsf{all}\, x.\, (x : T_1) \to \langle (E\ x) : T_2\rangle] \\
[\![\, \mathsf{abstr}\ E \,]\!] & \Longrightarrow & (\mathsf{fix}\, x.\, E\ x) : (T) \longleftarrow [\,] \mid [\mathsf{all}\, x.\, (x : T) \to \langle (E\ (\mathsf{fix}\, x.\, E\ x) : T\rangle] \\
 & \Longrightarrow & \mathsf{ofI}\ (\mathtt{ev}\ E)\ T \longleftarrow [\,] \mid [\langle E : T\rangle] \\
 & \Longrightarrow & \mathsf{ofI}\ (\mathtt{return}\ V)\ T \longleftarrow [\,] \mid [\langle V : T\rangle] \\
 & \Longrightarrow & \mathsf{ofI}\ (\mathtt{app}_1\ V\ E)\ T \longleftarrow [\,] \mid [\langle V : (T_2 \to T)\rangle, \langle E : T_2\rangle] \\
 & \Longrightarrow & \mathsf{ofK}\ (T \to T) \longleftarrow [\langle \mathsf{init}\ V\rangle] \mid [\,] \\
 & \Longrightarrow & \mathsf{ofK}\ (T_1 \to T_2) \longleftarrow [\langle \mathsf{cont}\ K\rangle, \langle \mathsf{ofK}\ T \to T_2\rangle] \mid \\
 & & \qquad\qquad [\mathsf{all}\, x.\, (x : T_1) \to \langle \mathsf{ofI}\ (K\ x)\ T\rangle] \\
 & & \\
 & \Longrightarrow & \mathsf{ceval}\ E\ V \longleftarrow [\mathsf{init}\ V \twoheadrightarrow \mathsf{ex}\ (\mathtt{ev}\ E)] \mid [\,] \\
 & \Longrightarrow & \mathsf{ex}\ (\mathtt{return}\ V) \longleftarrow [\langle \mathsf{init}\ V\rangle] \mid [\,] \\
 & \Longrightarrow & \mathsf{ex}\ (\mathtt{return}\ V) \longleftarrow [\langle \mathsf{cont}\ K\rangle, \langle \mathsf{ex}\ (K\ V)\rangle] \mid [\,] \\
[\![\, \mathsf{abstr}\ E \,]\!] & \Longrightarrow & \mathsf{ex}\ (\mathtt{ev}\ (\mathtt{fun}\ E)) \longleftarrow [\langle \mathsf{ex}\ (\mathtt{return}\ (\mathtt{fun}\ E))\rangle] \mid [\,] \\
 & \Longrightarrow & \mathsf{ex}\ (\mathtt{ev}\ (E_1 \mathbin{@} E_2)) \longleftarrow [\mathsf{cont}\ (\lambda v.\ \mathtt{app}_1\ v\ E_2) \twoheadrightarrow \langle \mathsf{ex}\ (\mathtt{ev}\ E_1)\rangle] \mid [\,] \\
[\![\, \mathsf{abstr}\ E \,]\!] & \Longrightarrow & \mathsf{ex}\ (\mathtt{app}_1\ (\mathtt{fun}\ E)\ E_2) \longleftarrow [\langle \mathsf{ex}\ (\mathtt{ev}\ (E\ E_2))\rangle] \mid [\,]
\end{array}
$$

Figure 12: Hybrid's encoding of program clauses

Proof: After introducing the logic variable $?V$ (here we pay no attention to the height of the derivation) we apply rule **bc**, *i.e.*, backchaining, obtaining the following 3 goals:

1. $\mathsf{ceval}\ (\ \mathsf{fun}\, x.\, x)\ ?V \longleftarrow [\mathsf{init}\ \ ?V \twoheadrightarrow \mathsf{ex}\ (\mathtt{ev}\ (\mathsf{fun}\, x.\, x))] \mid [\,]$
2. $[\,] \mid [\,] \blacktriangleright [\mathsf{init}\ \ ?V \twoheadrightarrow \mathsf{ex}\ (\mathtt{ev}\ (\mathsf{fun}\, x.\, x))]$
3. $[\,] \blacktriangleright [\,]$

Goals such as the third one (the base case of intuitionistic list evaluation) will always arise when back-chaining on evaluation, as the intuitionistic context plays no role, *i.e.*, it is empty; since they are trivially true, they will be resolved away without any further mention. So we have retrieved the body of the relevant clause and passed it to ordered list evaluation:

$$[\,] \mid [\,] \blacktriangleright [\mathsf{init}\ \ ?V \twoheadrightarrow \mathsf{ex}\ (\mathtt{ev}\ (\mathsf{fun}\, x.\, x))]$$

This leads to splitting the ordered context, *i.e.*,

1. $\mathsf{osplit}\ [\,]\ \ Og\ \ Or$
2. $[\,] \mid Og \rhd \mathsf{init}\ \ ?V \twoheadrightarrow \langle \mathsf{ex}\ (\mathtt{ev}\ (\mathsf{fun}\, x.\, x))\rangle$
3. $[\,] \mid Or \blacktriangleright [\,]$

In this case, ordered splitting is deterministic as it can only match the base case and the two resulting contexts Og and Or are both set to empty:

$$[\,] \mid [\,] \rhd \mathsf{init}\ \ ?V \twoheadrightarrow \langle \mathsf{ex}\ (\mathtt{ev}\ (\mathsf{fun}\, x.\, x))\rangle$$

The introduction rule for ordered implication (and simplification) puts the goal in the form:

$$[\,] \mid [\mathsf{init}\ \ ?V] \rhd \langle \mathsf{ex}\ (\mathtt{ev}\ (\mathsf{fun}\, x.\, x))\rangle$$

which corresponds to the execution of the identity function with the initial continuation. Another backchain yields

1. $\mathsf{abstr}\ (\lambda x.\ x)$
2. $\mathsf{osplit}\ [\mathsf{init}\ ?V]\ \ Og_1\ \ Or_1$
3. $[\,] \mid Og_1 \rhd \langle \mathsf{ex}\ (\mathtt{return}\ (\mathsf{fun}\, x.\, x))\rangle$

As usual, abstr_tac takes care of the first goal, while now we encounter the first interesting splitting case. To be able to solve the goal by assumption in the SL, we need to pass the (singleton) context to the left context Og_1. One way to achieve this is to gently push the system by proving the simple lemma $\exists A.\ \mathsf{osplit}\ [A]\ \ [A]\ \ [\,]$. Using the latter as an introduction rule for subgoal 2, we get:

$$[\,] \mid [\mathsf{init}\ ?V] \rhd \langle \mathsf{ex}\ (\mathtt{return}\ (\mathsf{fun}\, x.\, x))\rangle$$

More backchaining yields:

$$[\,] \mid [\mathsf{init}\ ?V] \blacktriangleright [\langle \mathsf{init}\ (\mathsf{fun}\, x.\, x)\rangle]$$

and with another similar ordered split to the left we have

$$[\,] \mid [\mathsf{init}\ ?V] \rhd \langle \mathsf{init}\ (\mathsf{fun}\, x.\, x)\rangle$$

which is true by the $\mathbf{init}_\Omega$ rule. This concludes the derivation, instantiating $?V$ with $\mathsf{fun}\, x.\, x$. □

If we collect in `sig_def` all the definitions pertaining to the signature in question and bundle up in `olli_intrs` all the introduction rules for the sequent calculus, (ordered) splitting and the program database

```
val olli_intrs = seq_split_seq_osplit_seq.intrs @
                 split.intrs @ osplit.intrs @ prog.intrs;

by(fast_tac(claset()  addIs olli_intrs
           (simpset() addSolver (abstr_solver sig_defs))) i);
```

the above tactic will automatically prove the above lemma in a flash, by backtracking on all the possible ordered splittings, which are, in the present case, preciously few. However, this will not be the case for practically any other goal evaluation, since splitting is highly non-deterministic in so far as all the possible partitions of the contexts need to be considered. To remedy this, we could encode a variant of the *input-output* sequent calculus described in [93] and further refined in [94], which describes efficient resource management—and hence search—in linear logic programming. Then, it would be a matter of showing it equivalent to the base calculus, which may be far from trivial. In the end of the day, our system will do fine for its aim, *i.e.*, investigation of the meta-theoretic properties of our case study.

The example may have shed some light about this peculiarity: the operational semantics of the continuation machine is small-step; a sequence of transitions are connected (via rules for its reflexive transitive closure) to compute a value, whereas our implementation looks at first sight big-step, or, at least, shows no sign of transitive closure. In fact, informally, for every transition that a machine makes from some state s_i to s_{i+1}, there is a bijective function that maps the *derivation* of $\ulcorner s \urcorner_i$, *i.e.* the sequent encoding s_i to the derivation of $\ulcorner s_{i+1} \urcorner$. The $\mathrm{Olli}^2_{\twoheadrightarrow}$ interpreter essentially simulates the informal trace of the machine obtained by transitive closure of each step $K \diamond i \hookrightarrow s'$ for some s' with a tree of attempts to establish $\Gamma \mid \ulcorner K \urcorner \rhd \langle \ulcorner i \urcorner \rangle$ by appropriate usage of the available ordered resources (the rest of $\ulcorner K \urcorner$). In the above example, the paper and pencil proof is a tree with **cev** at the root, linked by the **step** rule to the **st_fun** and **st_init** axioms. This corresponds to the $\mathrm{Olli}^2_{\twoheadrightarrow}$ proof we have described, whose skeleton consists of the statement of the lemma as root and ending with the axiom $\mathbf{init}_\Omega$.

$$[\,] \mid [\mathsf{init}\ \ ?V] \rhd \langle \mathsf{ex}\ (\mathtt{ev}\ (\mathsf{fun}\, x.\, x))\rangle \rightsquigarrow [\,] \mid [\mathsf{init}\ ?V] \rhd \langle \mathsf{ex}\ (\mathtt{return}\ (\mathsf{fun}\, x.\, x))\rangle \rightsquigarrow [\,] \mid [\mathsf{init}\ ?V] \rhd \langle \mathsf{init}\ (\mathsf{fun}\, x.\, x)\rangle$$

Now we can address the meta-theory, namely the subject reduction theorem:

MC-Theorem 32 (sub_red_aux)

$$
\begin{array}{l}
[\,] \mid \mathsf{init}\ V, \Omega \ \rhd_i \langle \mathsf{ex}\ I \rangle \Longrightarrow \\
\quad \forall T_1 T_2.\, \rhd \langle \mathsf{ofI}\ I\ T_1 \rangle \longrightarrow \\
\quad ([\,] \mid \mathsf{init}\ V, \Omega \rhd \langle \mathsf{ofK}\ (T_1 \rightarrow T_2) \rangle) \longrightarrow \rhd \langle V : T_2 \rangle)
\end{array}
$$

The proof of subject reduction again follows from first principles and does not need any weakening or substitution lemmas. The proof and proof scripts are considerably more manageable if we first establish some simple facts about typing of various syntax categories and instruct the system to aggressively apply every deterministic splitting, *e.g.*,

$$[\![\ \mathsf{osplit}\ [\,]\ \ Og\ \ Or;\ [\![\ Og = [\,];\ Or = [\,]\ ]\!] \Longrightarrow P\ ]\!] \Longrightarrow P$$

as well as a number of elimination rules stating the impossibility of some inversions such as

$$[\![\ \mathsf{cont}\ K \longleftarrow Ol \mid Il\ ]\!] \Longrightarrow P$$

The human intervention that is required is limited to providing the correct splitting of the ordered hypotheses and selecting the correct instantiations of the heights of sub-derivations in order to fire the IH. **Proof:** The proof is by complete induction on the height of the derivation of the premise. The inductive hypothesis, denoted *IH(n)* (which may be omitted next) is

$$
\begin{array}{l}
\forall m.\ m < n \longrightarrow \\
\quad (\forall I\ V\ \Omega. \\
\qquad [\,] \mid (\mathsf{init}\ V, \Omega) \rhd_m \langle \mathsf{ex}\ I \rangle \longrightarrow \\
\qquad \forall T_1\ T_2. \\
\qquad\quad [\,] \rhd \langle \mathsf{ofI}\ I\ T_1 \rangle \wedge \\
\qquad\quad (\mathsf{init}\ V, \Omega) \rhd \langle \mathsf{ofK}\ T_1 \rightarrow T_2 \rangle \longrightarrow \\
\qquad\qquad \rhd \langle \mathsf{ofV}\ V\ T_2 \rangle)
\end{array}
$$

We begin by inverting on $[\,] \mid (\mathsf{init}\ V, \Omega) \rhd_m \langle \mathsf{ex}\ I \rangle$ and then on the `prog` clauses defining execution, yielding several goals, one for each evaluation clause. The statement for the **st_return** case is as follows:

$$
\begin{array}{l}
[\![\ IH(i+1); \\
\quad [\,] \mid [\,] \rhd \langle \mathsf{ofI}\ (\mathtt{return}\ V')\ T_1 \rangle; \\
\quad [\,] \mid (\mathsf{init}\ V, \Omega) \rhd \langle \mathsf{ofK}\ T_1 \rightarrow T_2 \rangle; \\
\quad [\,] \mid (\mathsf{init}\ V, \Omega) \blacktriangleright [\langle \mathsf{ex}\ (K\ V') \rangle, \langle \mathsf{cont}\ K \rangle]\ ]\!] \\
\qquad\qquad \Longrightarrow \rhd \langle \mathsf{ofV}\ V\ T_2 \rangle
\end{array}
$$

We start by applying the typing lemma:

$$[\,] \mid [\,] \rhd \langle \mathsf{ofI}\ (\mathtt{return}\ V)\ T \rangle \quad \Longrightarrow \quad \rhd \langle \mathsf{ofI}\ V\ T \rangle$$

Inverting of the derivation of $[\,] \mid \mathsf{init}\ V, \Omega \blacktriangleright_i [\langle \mathsf{ex}\ (K\ V') \rangle, \langle \mathsf{cont}\ K \rangle]$ yields:

$$
\begin{array}{l}
[\![\ \ldots \mathsf{osplit}\ (\mathsf{init}\ V, \Omega)\ \ Og\ \ [\mathsf{cont}\ K]; \\
\quad [\,] \mid Og \rhd_i \langle \mathsf{ex}\ K\ V' \rangle; \\
\quad [\,] \mid [\,] \rhd \langle \mathsf{ofV}\ V'\ T_1 \rangle\ ]\!] \\
\qquad\qquad \Longrightarrow \rhd \langle \mathsf{ofV}\ V\ T_2 \rangle
\end{array}
$$

Now, there is only one viable splitting of the first premise, where $\bigwedge L.\ [\![\ \mathsf{osplit}\ \Omega\ \ L\ \ [\mathsf{cont}\ K];\ Og = (\mathsf{init}\ V, L)\ ]\!] \Longrightarrow P$, as the impossibility of the first one, entailing $\mathsf{cont}\ K = \mathsf{init}\ V$, is ruled out by the freeness properties of the encoding of atomic formulas. This results in

$$
\begin{array}{l}
[\![\ \ldots [\,] \mid (\mathsf{init}\ V, L) \rhd_i \langle \mathsf{ex}\ K\ V' \rangle; \\
\quad [\,] \mid (\mathsf{init}\ V, \Omega) \rhd \langle \mathsf{ofK}\ T_1 \rightarrow T_2 \rangle; \\
\quad [\,] \mid [\,] \rhd \langle \mathsf{ofV}\ V'\ T_1 \rangle; \\
\quad \mathsf{osplit}\ \Omega\ \ Ł\ \ [\mathsf{cont}\ K]\ ]\!] \\
\qquad\qquad \Longrightarrow \rhd \langle \mathsf{ofV}\ V\ T_2 \rangle
\end{array}
$$

We now use the reading of ordered split as "reversed" append to force Ω to be the concatenation of L and $[\mathsf{cont}\ K]$:

$$\begin{array}{l}[\![\ldots [\,] \mid (\mathsf{init}\ V, L) \rhd_i \langle \mathsf{ex}\ K\ V' \rangle; \\ \quad [\,] \mid (\mathsf{init}\ V, L)@[\mathsf{cont}\ K] \rhd \langle \mathsf{ofK}\ T_1 \to T_2 \rangle; \\ \quad [\,] \mid [\,] \rhd \langle \mathsf{ofV}\ V'\ T_1 \rangle\ ]\!] \\ \qquad\qquad \Longrightarrow \rhd \langle \mathsf{ofV}\ V\ T_2 \rangle\end{array}$$

we now invert on the typing of continuation:

$$\begin{array}{l}[\![\ldots [\,] \mid [\,] \blacktriangleright_i [\mathsf{all}\, v.\, \mathsf{ofV}\ v\ T_1\ \mathsf{imp}\ \langle \mathsf{ofI}\ K'\ v\ T \rangle]; \\ \quad [\,] \mid [\,] \rhd \langle \mathsf{ofV}\ V'\ T_1 \rangle; \\ \quad [\,] \mid (\mathsf{init}\ V, L)@[\mathsf{cont}\ K] \blacktriangleright [\langle \mathsf{ofK}\ T \to T_2 \rangle, \langle \mathsf{cont}\ K' \rangle]; \\ \quad [\,] \mid (\mathsf{init}\ V, L) \rhd_i \langle \mathsf{ex}\ K\ V' \rangle\ ]\!] \\ \qquad\qquad \Longrightarrow \rhd \langle \mathsf{ofV}\ V\ T_2 \rangle\end{array}$$

The informal proof would require an application of the substitution lemma. Instead here we use cut to infer

$$[\,] \mid [\,] \rhd \langle \mathsf{ofI}\ (K'\ V')\ T \rangle$$

We first have to invert on the hypothetical statement $\mathsf{all}\, v.\, \mathsf{ofV}\ v\ T_1\ \mathsf{imp}\ \langle \mathsf{ofI}\ (K'\ v)\ T \rangle$ and instantiate v with V':

$$\begin{array}{l}[\![\ldots [\,] \mid [\,] \rhd \langle \mathsf{ofV}\ V'\ T_1 \rangle; \\ \quad [\,] \mid (\mathsf{init}\ V, L) \rhd_i \langle \mathsf{ex}\ (K\ V') \rangle; \\ \quad [\,] \mid (\mathsf{init}\ V, L)@[\mathsf{cont}\ K] \blacktriangleright_{ic+2} [\langle \mathsf{ofK}\ T \to T_2 \rangle, \langle \mathsf{cont}\ K' \rangle]; \\ \quad [\mathsf{ofV}\ V'\ T_1] \mid [\,] \rhd_{ic} \langle \mathsf{ofI}\ (K'\ V')\ T \rangle\ ]\!] \\ \qquad\qquad \Longrightarrow \rhd \langle \mathsf{ofV}\ V\ T_2 \rangle\end{array}$$

Now one more inversion on $[\,] \mid (\mathsf{init}\ V, L)@[\mathsf{cont}\ K] \blacktriangleright [\langle \mathsf{ofK}\ T \to T_2 \rangle, \langle \mathsf{cont}\ K' \rangle]$ brings us to split $\mathsf{osplit}\ (\mathsf{init}\ V, L)@[\mathsf{cont}\ K]\ \ Og\ \ [\mathsf{cont}\ K']$ so that $(\mathsf{init}\ V, L) = Og$ and $K = K'$:

$$\begin{array}{l}[\![IH(i+1); \\ \quad [\,] \mid [\,] \rhd \langle \mathsf{ofV}\ V'\ T_1 \rangle; \\ \quad [\,] \mid (\mathsf{init}\ V, L) \rhd_i \langle \mathsf{ex}\ (K'\ V') \rangle; \\ \quad [\,] \mid [\,] \rhd \langle \mathsf{ofI}\ (K'\ V')\ T \rangle; \\ \quad [\,] \mid (\mathsf{init}\ V, L) \rhd_{ic+2} \langle \mathsf{ofK}\ T \to T_2 \rangle\ ]\!] \\ \qquad\qquad \Longrightarrow \rhd \langle \mathsf{ofV}\ V\ T_2 \rangle\end{array}$$

This final sequent follows from complete induction for height i. □

MC-Corollary 33 (subject_reduction) $[\![\rhd \mathsf{ceval}\ E\ V;\ \rhd E : T\ ]\!] \Longrightarrow \rhd V : T$

6 Related Work

There is nowadays extensive literature on approaches to representing and reasoning about what we have called "object logics," (OLs) where the notion of variable bindings is paramount. These approaches are supported (or are in the process of building such support) by implementations in the form of proof checkers, proof assistants and theorem provers. We will compare our approach to others according to two categories: whether the system uses different levels for different forms of reasoning and whether it is relational (*i.e.*, related to proof search) or functional (based on evaluation).

6.1 Two-level, Relational Approaches

Our work started as a way of porting most of the ideas of $FO\lambda^{\Delta I\!N}$ [62] into the mainstream of current proof assistants, so that they can enjoy the facilities and support that such assistants provide. As mentioned in the introduction, Isabelle/HOL or Coq plays the role of $FO\lambda^{\Delta I\!N}$, the introduction/elimination rules of inductive definitions (types) simulate the *defR* and *defL* rules of PIDs and the Hybrid meta-language provides $FO\lambda^{\Delta I\!N}$'s λ-calculus. In addition, our approach went beyond $FO\lambda^{\Delta I\!N}$, featuring meta-level induction and coinduction, which were later proved consistent with the theory of (partial) inductive definitions [75]. These features are now standard in $FO\lambda^{\Delta I\!N}$'s successor, *Linc* [105].

One of the more crucial advances given by *Linc*-like logic lies in the treatment of induction over *open* terms, offered by the proof-theory of [69, 105]. This proof theory has been recently modified [106] to simplify the proof-theory of ∇-quantification by removing local contexts of ∇-bounded variables so as to enjoy properties closer to the *fresh* quantifier of nominal logic, such as strengthening and permutation (see later in this section). Finally the $\mathscr{G}$ logic [39] brings together PIDs and ∇-quantification by allowing the latter to occur in the head of definitions. This gives excellent new expressive power, allowing for example to *define* the notion of freshness. Furthermore it eases induction over open terms and even gives a logical reading to the notion of "regular worlds" that are crucial in the meta-theory of Twelf.

At this stage, it is not obvious whether or how much of this can be incorporated in the Hybrid approach and it is a subject of future work.

Very recently, the two-level approach has received a brand new implementation from first principles. Firstly *Bedwyr* [6] is meant as a model-checker of higher-order specifications, such as whether terms of the π-calculus are provably bisimilar. This crucially uses ∇-quantification and case analysis, but does not aim to be a theorem prover. The already cited *Abella* [38] implements a large part of the $\mathscr{G}$ logic and has been used to give elegant proofs of the POPLMARK challenge [5] as well of proofs by logical relations [40], an issue which has been contentious in the theorem proving world. The proof is based on a notion of arbitrarily cascading substitutions, which shares with encodings based on nominal logic the problem that once nominal constants have been introduced, the user often needs to spend some time and effort (and lemmas) controlling their spread. One example is the lemma showing that such constants do not occur in types [40]. Similarly there is currently some need to control occurrences of free variables in terms. This can be taken as a need to rely on "technical" lemmas which have no counterpart in the informal proof. This is not a problem of the prover itself, but it derives by the nominal flavor that logics such as LG^{Ω} and $\mathscr{G}$ have introduced since the times of *Linc*.

The so far more established contender in the two-level relational approach is *Twelf* [86]. Here, the LF type theory is used to encode OLs as judgments and to specify meta-theorems as relations (type families) among them; a logic programming-like interpretation provides an operational semantics to those relations, so that an external check for totality (incorporating termination, well-modedness, coverage [87, 102]) verifies that the given relation is indeed a realizer for that theorem. In this sense the Twelf totality checker can be seen to work at a different level than the OL specifications.

Hickey *et al.* [48] build a theory for two-level reasoning within the MetaPRL system. Their use of levels is different from ours, and is based on reflection. A higher-order syntax representation is used at the level of reflected terms. A de Bruijn representation that is computationally equivalent to the higher-order representation is also defined. Principles of induction are automatically generated for a reflected theory, but it is stated that they are difficult to use interactively because of their size. In fact, there is little experience using the system for reasoning about OLs. An example formal proof is described showing the simple property that the subtyping relation for the type system of the POPLMARK challenge [5] is reflexive. The main property of the challenge—transitivity of this relation—is stated formally, but not proved.

6.2 Two-level, Functional Approaches

There exists a second approach to reasoning in LF that is built on the idea of devising an explicit (meta-)meta-logic for reasoning (inductively) about the framework, in a fully automated way [100]. $\mathscr{M}_\omega$ can be seen as a constructive first-order inductive type theory, whose quantifiers range over possibly open LF objects over a signature. In this calculus

it is possible to express and inductively prove meta-logical properties of an OL. 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. $\mathscr{M}_\omega$ can be also seen as a dependently-typed functional programming language, and as such it has been refined first into the *Elphin* programming language [103] and finally in *Delphin* [98]. ATS^{LF} [26] is an instantiation of Xi's *applied type systems* combining programming with proofs and can be used as a logical framework. In a similar vein the context modal logic of Pientka, Pfenning and Naneski [76] provides a basis for a different foundation for programming with HOAS based on hereditary substitutions as explicitly formulated as a programming language in [88, 89]. Because all of these systems are programming languages, we refrain from a deeper discussion.

6.3 One-level, Functional Approaches

Modal λ-calculi were formulated in the early attempts by Schürmann, Despeyroux, and Pfenning [101] to develop a calculus that allows the combination of HOAS with a primitive recursion principle in the *same* framework, while preserving the adequacy of representations. For every type A there is a type $\Box A$ of *closed* objects of type A. In addition to the regular function type $A \Rightarrow B$, there is a more restricted type $A \rightarrow B \equiv \Box A \Rightarrow B$ of "parametric" functions. Functions used as arguments for higher-order constructors are of this kind and thus roughly correspond to our notion of abstraction. The dependently-typed case is considered in [30] but the approach has been somehow abandoned in view of [76]. Washburn and Weirich [110] show how standard first-class polymorphism can be used instead of a special modal operator to restrict the function space to "parametric" functions. They encode and reason about higher-order iteration operators.

We have mentioned earlier the work by Gordon and Melham [43, 44], which we used as a starting point for Hybrid. Norrish building on this work, improves the recursion principles [80], allowing greater flexibility in defining recursive functions on this syntax.

6.4 Other One-Level Approaches

Weak[25] *higher-order abstract syntax* [29] is an approach that strives to co-exist with an inductive setting, where the positivity condition for datatypes and hypothetical judgments must be obeyed. In weak HOAS, the problem of negative occurrences in datatypes is handled by replacing them with a new type. For example, the fun constructor for Mini-ML introduced in Section 3 has type $(\underline{var} \Rightarrow uexp) \Rightarrow uexp$, where *var* is a type of variables, isomorphic to natural numbers. *Validity* predicates are required to weed out exotic terms, stemming from case analysis on the *var* type, which at times is inconvenient. The approach is extended to hypothetical judgments by introducing distinct predicates for the negative occurrences. Some axioms are needed to reason about hypothetical judgments, to mimic what is inferred by the cut rule in our architecture. Miculan *et al.*'s framework [22, 51, 65] embraces an *axiomatic* approach to meta-reasoning with weak HOAS in an inductive setting. It has been used within Coq, extended with a "theory of contexts" (ToC), which includes a set of axioms parametric to an HOAS signature. The theory includes the reification of key properties of names akin to *freshness*. Exotic terms are avoided by taking the *var* to be a parameter and assuming axiomatically the relevant properties. Furthermore, higher-order induction and recursion schemata on expressions are also assumed. To date, the consistency with respect to a categorical semantics has been investigated for higher-order logic [14], rather than w.r.t. a (co)inductive dependent type theory such as the one underlying Coq [42].

From our perspective, ToC can be seen as a stepping stone towards Gabbay and Pitts *nominal logic*, which aims to be a foundation of programming and reasoning with *names*, in a one-level architecture. This framework started as a variant of the Frankel-Mostowski set theory based on permutations [35], but it is now presented as a first-order theory [91], which includes primitives for variable renaming and variable freshness, and a (derived) new "freshness" quantifier. Using this theory, it is possible to prove properties by structural induction and also to define functions by recursion over syntax [92]. The proof-theory of nominal logic has been thoroughly investigated in [19, 36], and the latter also investigates the proof-theoretical relationships between the ∇ and the "freshness" quantifier, by providing a translation of the former to the latter.

Gabbay has tried to implement nominal sets on top of Isabelle [37]. A better approach has turned out to be Urban *et al.*'s; namely to engineer a *nominal datatype package* inside Isabelle/HOL [79, 108] analogous to the standard

[25] For the record, the by now standard terminology "weak" HOAS was coined by the second author of the present paper in [71].

datatype package but defining equivalence classes of term constructors. In more recent versions, principles of primitive recursion and strong induction have been added [107] and many case studies tackled successfully, such as proofs by logical relations (see [79] for more examples). The approach has also been compared in detail with de Bruijn syntax [11] and in hindsight owes to McKinna and Pollack's "nameless" syntax [63]. Nominal logic is beginning to make its way into Coq; see [4].

It is fair to say that while Urban's nominal package allows the implementation of informal proofs obeying the Barendregt convention almost literally, a certain number of lemmas that the convention conveniently hides must still be proved w.r.t. the judgment involved; for example to choose a *fresh* atom for an object x, one has to show that x has *finite support*, which may be tricky for x of functional type, notwithstanding the aid of general tactics implemented in the package. HOAS, instead, aims to make α-conversion disappear and tries to extract the abstract higher-order nature of calculi and proofs thereof, rather than follow line-by-line the informal development. On the other hand, it would be interesting to look at versions of the freshness quantifier at the SL level, especially for those applications where the behavior of the OL binder is not faithfully mirrored by HOAS, namely with the traditional universal quantification at the SL-level; well known examples of this case include (mis)match in the π-calculus and closure-conversion in functional programming.

Chlipala [20] recently introduced an alternate axiomatic approach to reasoning with weak HOAS. Object-level terms are identified as meta-terms belonging to an inductive type family, where the type of terms is parameterized by the type of variables. Exotic terms are ruled out by parametricity properties of these polymorphic types. Clever encodings of OLs are achieved by instantiating these type variables in different ways, allowing data to be recorded inside object-level variables (a technique borrowed from [110]). Example proofs developed with this technique include type preservation and semantic preservation of program transformations on functional programming languages.

6.5 Hybrid Variants

Some of our own related work has involved alternative versions of Hybrid as well as improvements to Hybrid, which we describe here.

Constructive Hybrid. A *constructive* version of Hybrid implemented in Coq [16] provides an alternative that could also serve as the basis for a two-level architecture. This version provides a new approach to defining induction and non-dependent recursion principles aimed at simplifying reasoning about OLs. In contrast to [103], where built-in primitives are provided for the reduction equations for the higher-order case, the recursion principle is defined on top of the base de Bruijn encoding, and the reduction equations proved as lemmas.

In order to define induction and recursion principles for particular OLs, terms of type *expr* are paired with proofs showing that they are in a form that can represent an object-level term. A dependent type is used to store such pairs; here we omit the details and just call it $expr'$, and sometimes oversimplify and equate $expr'$ with *expr*. For terms of Mini-ML for example, in addition to free variables and bound variables, terms of the forms $(\mathsf{CON}\ cAPP\ \$\ E_1\ \$\ E_2)$, $(\mathsf{CON}\ cABS\ \$\ \mathsf{LAM}\, x.E\ x)$ and $(\mathsf{CON}\ cFIX\ \$\ \mathsf{LAM}\, x.E\ x)$, which correspond to the bodies of the definitions of @, fun, and fix, are the only ones that can be paired with such a proof. Analogues of the definitions for constructing object-level terms of type *expr* are defined for type $expr'$. For example, $(e_1 \overline{@} e_2)$ is defined to be the dependent term whose first component is an application (using @) formed from the first components of e_1 and e_2, and whose second component is formed from the proof components of e_1 and e_2.

Instead of defining a general lambda operator, a version of lbind that does not rely on classical constructs is defined for each OL. Roughly, $(\mathsf{lbind}\ e)$ is obtained by applying e to a *new* free variable and then replacing it with de Bruijn index 0. A new variable for a term e of type $expr \Rightarrow expr$ is defined by adding 1 to the maximum index in subterms of the form $(\mathsf{VAR}\ x)$ in $(e(\mathsf{BND}\ 0))$. Note that terms that do not satisfy abstr may have a different set of free variables for every argument, but for those which do satisfy abstr, choosing $(\mathsf{BND}\ 0)$ as the argument to which e is applied does give an authentic free variable. Replacing free variable $(\mathsf{VAR}\ n)$ in $(e(\mathsf{VAR}\ n))$ with $(\mathsf{BND}\ 0)$ involves defining a substitution operator that increases bound indices as appropriate as it descends through ABS operators. This description of lbind is informal and hides the fact that these definitions are actually given on dependent pairs, *i.e.*, e has type $expr' \rightarrow expr'$. Thus, the definition of lbind depends on the OL because $expr'$ is defined for each OL.

Induction and recursion are also defined directly on type $expr'$. To obtain a recursion principle, it is shown that for any type t, a function f of type $expr' \to t$ can be defined by specifying its results on each "constructor" of the OL. For example, for the @ and fun cases of Mini-ML, defining f involves defining Happ and Hfun of the following types:

$$\mathsf{Happ} : expr' \to expr' \to B \to B \to B$$
$$\mathsf{Hfun} : (expr' \to expr') \to B \to B$$

and then the following reduction equations hold.

$$\mathsf{f}\,(e_1 @ e_2) = \mathsf{Happ}\, e_1\, e_2\, (\mathsf{f}\, e_1)\, (\mathsf{f}\, e_2)$$
$$\mathsf{f}\,(\mathsf{fun}\ \lambda x.\ f\,x) = \mathsf{Hfun}\,(\mathsf{canon}(\lambda x.\ f\,x))\,(\mathsf{f}\,(\mathsf{lbind}\ (\lambda x.\ f\,x)))$$

In these equations, we oversimplify, showing functions f, Happ, and Hfun applied to terms of type *expr*; in the actual equations, proofs paired with terms on the left are used to build proofs of terms appearing on the right. The canon function in the equation for fun uses another substitution operator to obtain a "canonical form," computed by replacing de Bruijn index 0 in $(\mathsf{lbind}\ (\lambda x.\ f\,x))$ with x. This function is the identity function on terms that satisfy abstr.

Another version of constructive Hybrid [15] in Coq has been proposed, in which theorems such as induction and recursion principles are proved once at a general level, and then can be applied directly to each OL. An OL is specified by a *signature*, which can include sets of sorts, operation names, and even built-in typing rules. A signature specifies the binding structure of the operators, and the recursion and induction principles are formulated directly on the higher-order syntax.

Hybrid 0.2**.** During the write-up of this report, the infrastructure of Hybrid has developed significantly, thanks to the work by Alan Martin (see [73]), so that we informally talk of Hybrid 0.2. Because those changes have been so recent and only relatively influence the two-level approach, we have decided not to update the whole paper, but mention here the relevant differences.

The main improvement concerns an overall reorganization of the infrastructure described in Section 2, based on the internalization as a type of the set of *proper* terms. Using Isabelle/HOL's *typedef* mechanism, the type *prpr* is defined as a *bijective image* of the set $\{\, s :: expr \mid \mathsf{level}\, 0\, s \,\}$, with inverse bijections $\mathsf{expr} :: prpr \Rightarrow expr$ and $\mathsf{prpr} :: expr \Rightarrow prpr$. In effect, *typedef* makes *prpr* a subtype of *expr*, but since Isabelle/HOL's type system does not have subtyping, the conversion function must be explicit. Now that OL terms can only be well-formed de Bruijn terms, we can replace the *proper_abst* property (MC-Lemma 3) with the new lemma

MC-Lemma 34 (abstr_const)

$$\mathsf{abstr}\ (\lambda\, v.\, t :: prpr)$$

From the standpoint of two-level reasoning we can dispose of all proper assumptions: in particular the SL universal quantification clause (Figure 5) becomes:

$$[\![\, \forall x.\ \Gamma \rhd_n (G\,x) \,]\!] \quad \Longrightarrow \quad \Gamma \rhd_{n+1} (\mathsf{all}\, x.\, G\,x)$$

Therefore, in the proof of MC-Lemma 23 no proper assumptions are generated. The proof of OL Subject Reduction (MC-Theorem 24) does not need to appeal to property (3) or, more importantly, to part 1 of MC-Lemma 14. While this is helpful, it does not eliminate the need for adding well-formedness annotations in OL judgments for the sake of establishing adequacy of the encoding.

Further, a structural definition of abstraction allows us to state the crucial quasi-injectivity property of the Hybrid binder LAM, strengthening MC-Theorem 4 by requiring only one of e and f to satisfy this condition (instead of both), thus somewhat simplifying the elimination rules for inductively defined OL judgments:

MC-Theorem 35 (strong_lambda_inject)

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

The new definition replaces the $\mathsf{abstr_tac}$ with plain Isabelle/HOL simplification, and the same applies, *a fortiori* to $\mathsf{proper_tac}$.

7 Conclusions and Future Work

We have presented a multi-level architecture that allows reasoning about objects encoded using HOAS in well-known systems such as Isabelle/HOL and Coq that implement well-understood logics. The support for reasoning includes induction and co-induction as well as various forms of automation available in such systems such as tactical-style reasoning and decision procedures. We have presented several examples of its use, including an arguably innovative case study. As we have demonstrated, there are a variety of advantages of this kind of approach:

- It is possible to replicate in a well-understood and interactive setting the style of proof used in systems such as *Linc* designed specially for reasoning using higher-order encodings. The reasoning can be done in such a way that theorems such as subject reduction proofs are proven without "technical" lemmas foreign to the mathematics of the problem.
- Results about the intermediate layer of specification logics, such as cut elimination, are proven once and for all; in fact it is possible to work with different specification logics without changing the infrastructure.
- It is possible to use this architecture as a way of "fast prototyping" HOAS logical frameworks since we can quickly implement and experiment with a potentially interesting SL, rather than building a new system from scratch.

Since our architecture is based on a very small set of theories that definitionally builds an HOAS meta-language on top of a standard proof-assistant, this allows us to do without any axiomatic assumptions, in particular freeness of HOAS constructors and extensionality properties at higher types, which in our setting are now theorems. Furthermore, we have shown that mixing of meta-level and OL specifications make proofs more easily mechanizable. Finally, by the simple reason that the Hybrid system sits on top of Isabelle/HOL or Coq, we benefit from the higher degree of automation of the latter.

Some of our current and future work will concentrate on the practical side, such as continuing the development and the testing of the new infrastructure to which we have referred as *Hybrid* 0.2 (see Section 6.5 and [73]), especially to exploit the new features offered by Isabelle/HOL 2008. One case study of particular interest is being tackled by Alan Martin and consists in investigating the meta-theory of a functional programming language with references using a linear ordered SL. This work extends the work in Section 5 and [74], as well as offering a different encoding of Mini-ML with references than the one analyzed with a linear logical framework [18]. Moreover, our approach seems to be exempt from the problems connected to verifying meta-theoretical *linear* properties in LF-style, as addressed in [99].

Still on the practical side we envisage developing a package similar in spirit to Urban's nominal datatype package for Isabelle/HOL [79]. For Hybrid, such a package would automatically supply a variety of support from a user specification of an OL, such as validity predicates like isterm, a series of theorems expressing freeness of the constructors of such a type including injectivity and clash theorems, and an induction principle on the shape of expressions analogous to MC-Theorem 6. To work at two levels, such a package would include a number of pre-compiled SLs (including cut-elimination proofs and other properties) as well as some lightweight tactics to help with two-level inference. Ideally, the output of the package could be in itself generated by a tool such as *OTT* ([104]) so as to exploit the tool's capabilities of supporting work on large programming language definitions, where "the scale makes it hard to keep a definition internally consistent, and hard to keep a tight correspondence between a definition and implementations", *op. cit*.

Crucially, more work is required at the theoretical level. Note that in all our examples, proofs by induction over derivations were always on closed judgment such as evaluation, be it encoded as a direct inductive definition at the meta-level or as prog clauses used by the SL. In both cases, this judgment was encoded without the use of hypothetical and parametric judgments, and thus induction was over *closed* terms, although we essentially used case analysis on open terms. Inducting over open terms and hypothetical judgments is a challenge [100] that we are currently addressing, both by developing techniques specific to our Hybrid framework, and trying to endow the ambient logic with a form of the "nabla" quantifier [39, 106], possibly using HOL/Nominal as the meta-meta-logic.

Another thread we are planning to to investigate is the use of our framework to aid in gaining a better understanding and "popularization" of Twelf proofs. In Twelf, as mentioned, much of the work in constructing proofs is handled by

an external check for properties such as termination and coverage [87, 102]. We plan to investigate Hybrid as the target of a sort of "compilation" of such proofs into the well-understood higher-order logic of Isabelle/HOL. In doing so we should be able to browse such proofs so that we can better understand them and their correspondence to proofs in our framework.

Acknowledgements. Some of the material in this paper is based on previous and current joint work with Simon Ambler and Roy Crole [2, 3, 70–72], Jeff Polakow [74], Venanzio Capretta [16], and Alan Martin [73], whose contribution we gratefully acknowledge. The paper has also benefited from discussions with Andy Gordon, Dale Miller, Marino Miculan, Brigitte Pientka, Randy Pollack, Frank Pfenning, and Carsten Schürman.

References

[1] Samson Abramsky and C.-H. Luke Ong. Full abstraction in the lazy lambda calculus. *Inf. Comput.*, 105(2):159–267, 1993.

[2] S. J. Ambler, R. L. Crole, and Alberto Momigliano. A definitional approach to primitive recursion over higher order abstract syntax. In *MERλIN '03: Proceedings of the 2003 ACM SIGPLAN workshop on MEchanized Reasoning about Languages with varIable biNding*, pages 1–11, New York, NY, USA, 2003. ACM Press.

[3] Simon Ambler, Roy L. Crole, and Alberto Momigliano. Combining higher order abstract syntax with tactical theorem proving and (co)induction. In Carreño et al. [17], pages 13–30.

[4] Brian Aydemir, Aaron Bohannon, and Stephanie Weirich. Nominal reasoning techniques in Coq. *Electron. Notes Theor. Comput. Sci.*, 174(5):69–77, 2007.

[5] Brian E. Aydemir, Aaron Bohannon, Matthew Fairbairn, J. Nathan Foster, Benjamin C. Pierce, Peter Sewell, Dimitrios Vytiniotis, Geoffrey Washburn, Stephanie Weirich, and Steve Zdancewic. Mechanized metatheory for the masses: the POPLMARK challenge. In Joe Hurd and T. Melham, editors, *Theorem Proving in Higher Order Logics, 18th International Conference*, Lecture Notes in Computer Science, pages 50–65. Springer, 2005.

[6] David Baelde, Andrew Gacek, Dale Miller, Gopalan Nadathur, and Alwen Tiu. The Bedwyr system for model checking over syntactic expressions. In Frank Pfenning, editor, *CADE*, volume 4603 of *Lecture Notes in Computer Science*, pages 391–397. Springer, 2007.

[7] Clemens Ballarin. Locales and locale expressions in Isabelle/Isar. In Berardi et al. [9], pages 34–50.

[8] Nick Benton and Andrew Kennedy. Monads, effects and transformations. *Electr. Notes Theor. Comput. Sci.*, 26, 1999.

[9] Stefano Berardi, Mario Coppo, and Ferruccio Damiani, editors. *Types for Proofs and Programs, International Workshop, TYPES 2003, Torino, Italy, April 30 - May 4, 2003, Revised Selected Papers*, volume 3085 of *Lecture Notes in Computer Science*. Springer, 2004.

[10] Stefan Berghofer and Tobias Nipkow. Proof terms for simply typed higher order logic. In J. Harrison and M. Aagaard, editors, *Theorem Proving in Higher Order Logics*, volume 1869 of *LNCS*, pages 38–52. Springer, 2000.

[11] Stefan Berghofer and Christian Urban. A head-to-head comparison of de Bruijn indices and names. *Electr. Notes Theor. Comput. Sci.*, 174(5):53–67, 2007.

[12] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. Springer, 2004.

[13] K. A. Bowen and R. A. Kowalski. Amalgamating language and metalanguage in logic programming. In K. L. Clark and S. A. Tarnlund, editors, *Logic programming, vol 16 of APIC studies in data processing*, pages 153–172. Academic Press, 1982.

[14] Anna Bucalo, Furio Honsell, Marino Miculan, Ivan Scagnetto, and Martin Hoffman. Consistency of the theory of contexts. *J. Funct. Program.*, 16(3):327–372, 2006.

[15] Venanzio Capretta and Amy Felty. Higher order abstract syntax in type theory. http://www.cs.ru.nl/ venanzio/publications/HOUA.pdf, 2006.

[16] Venanzio Capretta and Amy P. Felty. Combining de Bruijn indices and higher-order abstract syntax in Coq. In Thorsten Altenkirch and Conor McBride, editors, *TYPES*, volume 4502 of *Lecture Notes in Computer Science*, pages 63–77. Springer, 2006.

[17] Victor Carreño, César Muñoz, and Sofiène Tashar, editors. *Theorem Proving in Higher Order Logics, 15th International Conference, TPHOLs 2002, Hampton, VA, USA, August 20-23, 2002, Proceedings*, volume 2410 of *Lecture Notes in Computer Science*. Springer, 2002.

[18] Iliano Cervesato and Frank Pfenning. A linear logical framework. *Inf. Comput.*, 179(1):19–75, 2002.

[19] James Cheney. A simpler proof theory for nominal logic. In Vladimiro Sassone, editor, *FoSSaCS*, volume 3441 of *Lecture Notes in Computer Science*, pages 379–394. Springer, 2005.

[20] Adam Chlipala. Parametric higher-order abstract syntax for mechanized semantics. In *13th ACM SIGPLAN International Conference on Functional Programming*, September 2008.

[21] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[22] Alberto Ciaffaglione, Luigi Liquori, and Marino Miculan. Imperative object-based calculi in co-inductive type theories. In Moshe Y. Vardi and Andrei Voronkov, editors, *LPAR*, volume 2850 of *Lecture Notes in Computer Science*, pages 59–77. Springer, 2003.

[23] 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, August 1986.

[24] Karl Crary and Susmit Sarkar. Foundational certified code in a metalogical framework. In Franz Baader, editor, *CADE*, volume 2741 of *Lecture Notes in Computer Science*, pages 106–120. Springer, 2003.

[25] Roy Crole. Hybrid adequacy. Technical Report CS-06-011, School of Mathematics and Computer Sience, University of Leicester, UK, November 2006.

[26] Sa Cui, Kevin Donnelly, and Hongwei Xi. ATS: A language that combines programming with theorem proving. In Bernhard Gramlich, editor, *FroCos*, volume 3717 of *Lecture Notes in Computer Science*, pages 310–320. Springer, 2005.

[27] Olivier Danvy, Belmina Dzafic, and Frank Pfenning. On proving syntactic properties of CPS programs. In Andrew Gordon and Andrew Pitts, editors, *Proceedings of HOOTS'99*, Paris, September 1999. Electronic Notes in Theoretical Computer Science, Volume 26.

[28] N.G. de Bruijn. A plea for weaker frameworks. In G. Huet and G. Plotkin, editors, *Logical Frameworks*, pages 40–67. Cambridge University Press, 1991.

[29] Joëlle Despeyroux, Amy Felty, and André Hirschowitz. Higher-order abstract syntax in Coq. In *Second International Conference on Typed Lambda Calculi and Applications*, pages 124–138. Springer, *Lecture Notes in Computer Science*, April 1995.

[30] Joelle Despeyroux and Pierre Leleu. Metatheoretic results for a modal λ-calculus. *Journal of Functional and Logic Programming*, 2000(1), 2000.

[31] D.Wu, A.Appel, and A. Stump. Foundational proof checkers with small witnesses. In *PPDP '03: Proceedings of the 5th ACM SIGPLAN international conference on principles and practice of declarative programming*, volume ACM Press, pages 264–274, 2003.

[32] Lars-Henrik Eriksson. Pi: an interactive derivation editor for the calculus of partial inductive definitions. In Alan Bundy, editor, *CADE*, volume 814 of *Lecture Notes in Computer Science*, pages 821–825. Springer, 1994.

[33] Amy P. Felty. Two-level meta-reasoning in Coq. In Carreño et al. [17], pages 198–213.

[34] Jonathan Ford and Ian A. Mason. Formal foundations of operational semantics. *Higher-Order and Symbolic Computation*, 16(3):161–202, 2003.

[35] M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13:341–363, 2001.

[36] Murdoch Gabbay and James Cheney. A sequent calculus for nominal logic. In *LICS*, pages 139–148. IEEE Computer Society, 2004.

[37] Murdoch J. Gabbay. Automating Fraenkel-Mostowski syntax. Technical Report CP-2002-211736, NASA, 2002. Track B Proceedings of TPHOLs'02.

[38] Andrew Gacek. The Abella interactive theorem prover (system description). In *IJCAR*, 2008.

[39] Andrew Gacek, Dale Miller, and Gopalan Nadathur. Combining generic judgments with recursive definitions. In *LICS*, pages 33–44. IEEE Computer Society, 2008.

[40] Andrew Gacek, Dale Miller, and Gopalan Nadathur. Reasoning in Abella about structural operational semantics specifications. In Andreas Abel and Christian Urban, editors, *Informal proceedings of LFMTP'08*. To appearin ENTCS, 2008.

[41] Guillaume Gillard. A formalization of a concurrent object calculus up to α-conversion. In David A. McAllester, editor, *CADE*, volume 1831 of *Lecture Notes in Computer Science*, pages 417–432. Springer, 2000.

[42] Eduardo Gimenez. A Tutorial on Recursive Types in Coq. Technical Report RT-0221, Inria, 1998.

[43] Andrew Gordon. A mechanisation of name-carrying syntax up to α-conversion. In J.J. Joyce and C.-J.H. Seger, editors, *International Workshop on Higher Order Logic Theorem Proving and its Applications*, volume 780 of *Lecture Notes in Computer Science*, pages 414–427, Vancouver, Canada, August 1994. Springer.

[44] Andrew D. Gordon and Tom Melham. Five axioms of α-conversion. In J. von Wright, J. Grundy, and J. Harrison, editors, *Proceedings of the 9th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'96)*, pages 173–191, Turku, Finland, August 1996. Springer-Verlag LNCS 1125.

[45] Elsa L. Gunter. Why we can't have SML-style datatype declarations in HOL. In Luc J. M. Claesen and Michael J. C. Gordon, editors, *TPHOLs*, volume A-20 of *IFIP Transactions*, pages 561–568. North-Holland/Elsevier, 1992.

[46] L. Hallnas. Partial inductive definitions. *Theoretical Computer Science*, 87(1):115–147, July 1991.

[47] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993.

[48] Jason Hickey, Aleksey Nogin, Xin Yu, and Alexei Kopylov. Mechanized meta-reasoning using a hybrid HOAS/de Bruijn representation and reflection. In John H. Reppy and Julia L. Lawall, editors, *ICFP 2006*, pages 172–183. ACM Press, 2006.

[49] P. M. Hill and J. Gallagher. Meta-programming in logic programming. In Dov Gabbay, Christopher J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 5: Logic programming*, pages 421–498. Oxford University Press, Oxford, 1998.

[50] Joshua S. Hodas and Dale Miller. Logic programming in a fragment of intuitionistic linear logic. *Inf. Comput.*, 110(2):327–365, 1994.

[51] Furio Honsell, Marino Miculan, and Ivan Scagnetto. An axiomatic approach to metareasoning on nominal algebras in HOAS. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *ICALP*, volume 2076 of *Lecture Notes in Computer Science*, pages 963–978. Springer, 2001.

[52] Douglas J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation*, 124(2):103–112, 1996.

[53] Hybrid Group. Hybrid: A package for higher-order syntax in Isabelle and Coq. www.hybrid.dsi.unimi.it, 2008.

[54] Isar Group. Isar - intelligible semi-automated reasoning. http://isabelle.in.tum.de/Isar/, 2000, Accessed 3 Sept. 2008.

[55] I. Johansson. Der minimalkalkl, ein reduzierter intuitionistischer formalismus. *Compositio Math*, 4:119–136, 1937.

[56] Søren B. Lassen. Head normal form bisimulation for pairs and the $\lambda\mu$-calculus. In *LICS*, pages 297–306. IEEE Computer Society, 2006.

[57] Daniel K. Lee, Karl Crary, and Robert Harper. Towards a mechanized metatheory of standard ML. In *POPL '07: Proceedings of the 34th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 173–184, New York, NY, USA, 2007. ACM Press.

[58] Hanbing Liu and J. Strother Moore. Executable JVM model for analytical reasoning: A study. *Sci. Comput. Program.*, 57(3):253–274, 2005.

[59] Per Martin-Löf. On the meanings of the logical constants and the justifications of the logical laws. *Nordic Journal of Philosophical Logic*, 1(1):11–60, 1996.

[60] Andrew McCreight and Carsten Schürmann. A meta linear logical framework. Informal Proceedings of LFM'04.

[61] Raymond McDowell and Dale Miller. Cut-elimination for a logic with definitions and induction. *Theoretical Computer Science*, 232:91–119, 2000.

[62] Raymond McDowell and Dale Miller. Reasoning with higher-order abstract syntax in a logical framework. *ACM Transactions on Computational Logic*, 3(1):80–136, January 2002.

[63] James McKinna and Robert Pollack. Some lambda calculus and type theory formalized. *J. Autom. Reasoning*, 23(3-4):373–409, 1999.

[64] Thomas F. Melham. A mechanized theory of the π-calculus in HOL. *Nord. J. Comput.*, 1(1):50–76, 1994.

[65] Marino Miculan. On the formalization of the modal μ-calculus in the calculus of inductive constructions. *Information and Computation*, 164(1):199–231, 2001.

[66] Dale Miller. Forum: A multiple-conclusion specification logic. *Theor. Comput. Sci.*, 165(1):201–232, 1996.

[67] Dale Miller. Overview of linear logic programming. In Thomas Ehrhard, Jean-Yves Girard, Paul Ruet, and Phil Scott, editors, *Linear Logic in Computer Science*, volume 316 of *London Mathematical Society Lecture Note*, pages 119 – 150. Cambridge University Press, 2004.

[68] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[69] Dale Miller and Alwen Tiu. A proof theory for generic judgments. *ACM Trans. Comput. Logic*, 6(4):749–783, 2005.

[70] Alberto Momigliano and Simon Ambler. Multi-level meta-reasoning with higher order abstract syntax. In A. Gordon, editor, *FOSSACS'03*, volume 2620 of *LNCS*, pages 375–392. Springer Verlag, 2003.

[71] Alberto Momigliano, Simon Ambler, and Roy Crole. A comparison of formalisations of the meta-theory of a language with variable binding in Isabelle. In R.J. Boulton and P. Jackson, editors, *14th International Conference on Theorem Proving in Higher Order Logics (TPHOLs01), Supplemental Proceedings*, pages 267–282. Informatics Research Report EDI-INF-RR-01-23, 2001.

[72] Alberto Momigliano, Simon Ambler, and Roy L. Crole. A Hybrid encoding of Howe's method for establishing congruence of bisimilarity. *Electr. Notes Theor. Comput. Sci.*, 70(2), 2002.

[73] Alberto Momigliano, Alan J. Martin, and Amy P. Felty. Two-level Hybrid: A system for reasoning using higher-order abstract syntax. *Electr. Notes Theor. Comput. Sci.*, 196:85–93, 2008.

[74] Alberto Momigliano and Jeff Polakow. A formalization of an ordered logical framework in Hybrid with applications to continuation machines. In *MERLIN*. ACM, 2003.

[75] Alberto Momigliano and Alwen Fernanto Tiu. Induction and co-induction in sequent calculus. In Berardi et al. [9], pages 293–308.

[76] Aleksandar Nanevski, Brigitte Pientka, and Frank Pfenning. Contextual modal type theory. *ACM Transactions on Computational Logic*, 200? To appear.

[77] Tobias Nipkow. Winskel is (almost) right: Towards a mechanized semantics. *Formal Asp. Comput.*, 10(2):171–186, 1998.

[78] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer Verlag, 2002.

[79] Nominal Methods Group. Nominal Isabelle. isabelle.in.tum.de/nominal/, 2008, Accessed 2 July 2008.

[80] Michael Norrish. Recursive function definition for types with binders. In *Seventeenth International Conference on Theorem Proving in Higher Order Logics*, pages 241–256. Springer-Verlag Lecture Notes in Computer Science, 2004.

[81] Christine Paulin-Mohring. Inductive definitions in the system Coq: Rules and properties. In M. Bezem and J.F. Groote, editors, *Proceedings of the International Conference on Typed Lambda Calculi and Applications*, pages 328–345. Springer-Verlag LNCS 664, March 1993.

[82] Lawrence C. Paulson. A fixedpoint approach to implementing (co)inductive definitions. In Alan Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, pages 148–161, Nancy, France, June 1994. Springer-Verlag LNAI 814.

[83] Frank Pfenning. Structural cut elimination in linear logic. Technical Report CMU-CS-94-222, Carnegie Mellon University, 1994.

[84] Frank Pfenning. Logical frameworks. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*. Elsevier Science Publishers, 1999.

[85] Frank Pfenning. *Computation and Deduction*. Cambridge University Press, In preparation. Draft from March 2001 available at www.cs.cmu.edu/∼fp/courses/comp-ded/handouts/cd.pdf. Accessed 30 July 2008.

[86] Frank Pfenning and Carsten 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.

[87] Brigitte Pientka. Verifying termination and reduction properties about higher-order logic programs. *J. Autom. Reasoning*, 34(2):179–207, 2005.

[88] Brigitte Pientka. A type-theoretic foundation for programming with higher-order abstract syntax and first-class substitutions. In George C. Necula and Philip Wadler, editors, *POPL*, pages 371–382. ACM, 2008.

[89] Brigitte Pientka and Joshua Dunfield. Programming with proofs and explicit contexts. In *PPDP*. ACM Press, 2008.

[90] Benjamin C. Pierce. *Types and Programming Languages*. The MIT Press, Cambridge, MA, 2002.

[91] Andrew M. Pitts. Nominal logic, a first order theory of names and binding. *Information and Computation*, 186(2):165–193, 2003.

[92] Andrew M. Pitts. Alpha-structural recursion and induction. *J. ACM*, 53(3):459–506, 2006.

[93] Jeff Polakow. *Ordered Linear Logic and Applications*. PhD thesis, CMU, 2001.

[94] Jeff Polakow. Linearity constraints as bounded intervals in linear logic programming. *J. Log. Comput.*, 16(1):135–155, 2006.

[95] Jeff Polakow and Frank Pfenning. Relating natural deduction and sequent calculus for intuitionistic non-commutative linear logic. In Andre Scedrov and Achim Jung, editors, *Proceedings of the 15th Conference on Mathematical Foundations of Programming Semantics*, New Orleans, Louisiana, April 1999. *Electronic Notes in Theoretical Computer Science, Volume 20*.

[96] Jeff Polakow and Frank Pfenning. Properties of terms in continuation-passing style in an ordered logical framework. In Joëlle Despeyroux, editor, *2nd Workshop on Logical Frameworks and Meta-languages (LFM'00)*, Santa Barbara, California, June 2000. Proceedings available as INRIA Technical Report.

[97] Jeff Polakow and Kwangkeun Yi. Proving syntactic properties of exceptions in an ordered logical framework. In Herbert Kuchen and Kazunori Ueda, editors, *Proceedings of the 5th International Symposium on Functional and Logic Programming (FLOPS'01)*, pages 61–77, Tokyo, Japan, March 2001. Springer-Verlag LNCS 2024.

[98] Adam Poswolsky and Carsten Schürmann. Practical programming with higher-order encodings and dependent types. In Sophia Drossopoulou, editor, *ESOP*, volume 4960 of *Lecture Notes in Computer Science*, pages 93–107. Springer, 2008.

[99] Jason Reed. Hybridizing a logical framework. *Electron. Notes Theor. Comput. Sci.*, 174(6):135–148, 2007.

[100] Carsten Schürmann. *Automating the Meta-Theory of Deductive Systems*. PhD thesis, Carnegie-Mellon University, 2000. CMU-CS-00-146.

[101] Carsten Schürmann, Joëlle Despeyroux, and Frank Pfenning. Primitive recursion for higher-order abstract syntax. *Theor. Comput. Sci.*, 266(1-2):1–57, 2001.

[102] Carsten Schürmann and Frank Pfenning. A coverage checking algorithm for LF. In David A. Basin and Burkhart Wolff, editors, *TPHOLs*, volume 2758 of *Lecture Notes in Computer Science*, pages 120–135. Springer, 2003.

[103] Carsten Schürmann, Adam Poswolsky, and Jeffrey Sarnat. The $\bigtriangledown$-calculus. Functional programming with higher-order encodings. In *Seventh International Conference on Typed Lambda Calculi and Applications*, pages 339–353. Springer, *Lecture Notes in Computer Science*, April 2005.

[104] Peter Sewell, Francesco Zappa Nardelli, Scott Owens, Gilles Peskine, Tom Ridge, Susmit Sarkar, and Rok Strnisa. Ott: effective tool support for the working semanticist. In Ralf Hinze and Norman Ramsey, editors, *ICFP 2007*, pages 1–12. ACM, 2007.

[105] Alwen Tiu. *A Logical Framework for Reasoning about Logical Specifications*. PhD thesis, Pennsylvania State University, May 2004.

[106] Alwen Tiu. A logic for reasoning about generic judgments. *Electr. Notes Theor. Comput. Sci.*, 174(5):3–18, 2007.

[107] Christian Urban and Stefan Berghofer. A recursion combinator for nominal datatypes implemented in Isabelle/HOL. In Ulrich Furbach and Natarajan Shankar, editors, *IJCAR*, volume 4130 of *Lecture Notes in Computer Science*, pages 498–512. Springer, 2006.

[108] Christian Urban and Christine Tasson. Nominal techniques in Isabelle/HOL. In R. Nieuwenhuis, editor, *Proceedings of the 20th International Conference on Automated Deduction (CADE)*, volume 3632 of *LNCS*, pages 38–53. Springer, 2005.

[109] René Vestergaard and James Brotherston. A formalised first-order confluence proof for the λ-calculus using one-sorted variable names. *Inf. Comput.*, 183(2):212–244, 2003.

[110] Geoffrey Washburn and Stephanie Weirich. Boxes go bananas: Encoding higher-order abstract syntax with parametric polymorphism. *Journal of Functional Programming*, 18(1):87–140, January 2008.

[111] Kevin Watkins, Iliano Cervesato, Frank Pfenning, and David Walker. A concurrent logical framework: The propositional fragment. In Berardi et al. [9], pages 355–377.