A Formalization of an Ordered Logical Framework in Hybrid with Applications to Continuation Machines

Alberto Momigliano[1] (A.Momigliano@mcs.le.ac.uk) and
Jeff Polakow[2] (jp@macs.hw.ac.uk)

[1] Department of Mathematics and Computer Science, University of Leicester, U.K.
[2] Department of Computer Science, Heriot-Watt University, Edinburgh, Scotland.

Abstract. We report on work in progress devoted to the formalization of an Ordered Logical Framework (OLF) based on a two-level architecture [8] in the Hybrid system. OLF here is a second-order version of ordered linear logic to be used as a meta-language for the verification of the (meta)theory of deductive systems. It is implemented as a meta-interpreter on top of the Hybrid system, which provides the full HOAS language. We apply the framework to the formal verification of type preservation of a simple continuation machine for Mini-ML.

1 Introduction

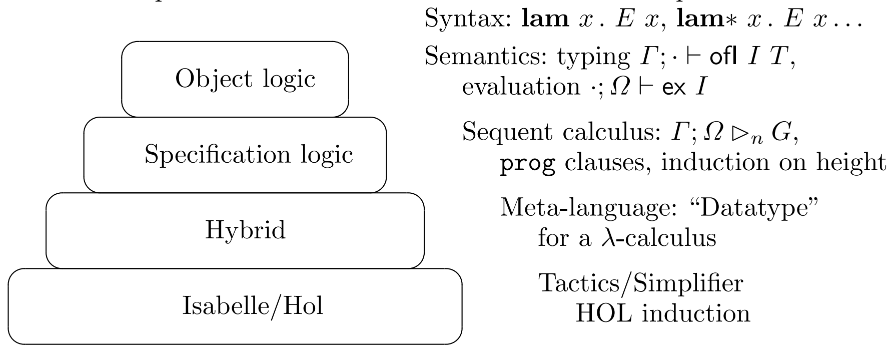
How to best encode and formally verify the (meta)theory of languages with variable binding has attracted much current research. A fair amount of it, however, mainly concerns questions of *syntax*, spanning from named syntax to De Bruijn indexes and various forms of higher-order abstract syntax (HOAS). While this is a fundamental issue, which is bound to heavily influence any development, the methodology with which one encodes *judgments* is equally important. Indeed, as initially suggested by Martin-Löf and practiced in the Edinburgh Logical Framework, (full) HOAS naturally leads to the use of hypothetical and parametric judgments to encode object level "relations-in-context". The benefits, as well as the problems, associated with this methodology are by now well-known and exemplified, for example, in the *Twelf* project. Briefly, object-level environments are represented by meta-level (logical) contexts, simplifying or even making irrelevant a substantial part of bookkeeping jobs, such as weakening and substitution lemmas. These lemmas, albeit trivial from a mathematical standpoint, in practice tend to be a bottleneck during formal verification. On the other hand, reasoning by induction in this setting has been notoriously difficult and only recently have reasonable solutions, with various degrees of satisfaction, been proposed [21, 11, 13].

As successful as these efforts have been for LF, it soon became apparent that some of the structural properties of the meta-logic were not appropriate to all situations. In particular, LF is intuitionistic and this imposes on the object level encoding weakening and contraction properties, which are not always appropriate for the domain under study; a typical example being when a notion of *state* is paramount. To fix ideas, let us consider the case of encoding the static and dynamic semantics of a language with imperative features, say Mini-ML with references (MLR). While encoding the typing system with an intuitionistic context is adequate, this does not apply to the representation of the store. If the store is modeled by a set of assumptions, say `contains C V`, where `C` is a cell and `V` a value, then *linear logic* [9] offers the right setting, since updates will consume and add resources. This is one of the motivations for investigating frameworks based on linear logics such as LLF [4]. On the other hand, work on the *automation* of reasoning in such frameworks is still in its infancy [10].

The methodology which enriches, in a *conservative* way, a logical framework so as to capture, at the right level of abstraction, different object level phenomena can be pushed further. In this paper we adopt an *ordered logical framework* [16], by which we mean here a second-order, minimal ordered linear logic. Ordered linear logic combines reasoning with unrestricted (or intuitionistic), linear and ordered hypotheses. Unrestricted 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 were assumed. We refer to Section 5 for a brief review of how this has been applied to the theory of programming languages so far.

This additional expressive power allows the possibility of the logic directly handling the notion of *stacks*. Stacks of course are ubiquitous when dealing with abstract and virtual machines; in particular, the operational semantics in the very elegant proof of type preservation of MLR in loc. cit. is based on a continuation machine. As we detail in Section 2, this entails the introduction of additional layers of *instructions, continuations and states*, and those entities need to be typed further complicating the set-up. The idea motivating the present paper is that by using OLF, the level of continuations can be disposed of via internalization in an ordered context. The latter operates as a stack of continuations to be evaluated.

We implement OLF via the two-level architecture suggested in [11] and adopted in [13], which we refer to for more details. The basis is the higher-order meta-language Hybrid [1], within Isabelle HOL, which provides a form of HOAS for the user to represent object logics. The user level is separated from the infrastructure, in which HOAS is implemented *definitionally* via a de Bruijn style encoding. The two-level idea refers to the separation between the level of *specification* and of (inductive) *meta-reasoning* within a system. Inside the meta-language we develop a *specification logic* (SL) — in this case, ordered linear logic — which in turn is used to specify an *object-logic* (OL), which, for this paper, will be the static and dynamic semantics of a continuation machine. This partition solves the problem of meta-reasoning in the presence of negative occurrences, since hypothetical judgments are now encapsulated within the OL and therefore not required to be inductive. The architecture is depicted below:


Syntax: **lam** $x\,.\,E\ x$, **lam**$*$ $x\,.\,E\ x\ldots$
Object logic
Semantics: typing $\Gamma; \cdot \vdash \mathsf{ofI}\ I\ T$,
evaluation $\cdot; \Omega \vdash \mathsf{ex}\ I$
Specification logic
Sequent calculus: $\Gamma; \Omega \rhd_n G$,
`prog` clauses, induction on height
Hybrid
Meta-language: “Datatype”
for a λ-calculus
Isabelle/Hol
Tactics/Simplifier
HOL induction

We can view the two-level approach as a way of “fast prototyping” HOAS logical frameworks (the “next 700 _” syndrome). We can implement and experiment with a SL quickly, easily and in a way compatible with induction and tactical theorem proving. In particular, we do not need to develop a new unification algorithm, but we rely on the one provided by the proof assistant. The price to pay is the additional layer where we explicitly reference provability and thus require a sort of meta-interpreter (the SL logic) to explicitly encode it. This indirectness can be alleviated by defining simple tactics, but in the end this is no substitute for implementing a logical framework from first principles.

Our long term goal is investigating the meta-theory of languages with imperative features, starting with the formal verification of the proof of type preservation for MLR. This paper sets the project going by implementing the framework and testing it with type preservation of the CPM for pure Mini-ML only. For the sake of conciseness, we will restrict ourselves to semantics of call-by-name λ-calculus. Further, although the implementation handles all of second-order Olli (the uniform fragment of ordered linear logic) [16], we will elide references to the (unordered) linear context and linear implication, and to the ordered left implication, as they do not play any role in this case-study.

The paper is organized as follows: in the next Section 2 we introduce, at an informal level, the continuation machine. Section 3 recalls some basic notions of Hybrid and its syntax representing techniques. In Section 4 we introduce the two-level architecture, describing the SL (4.1) and the OL (4.2); this is followed by the formal verification of the proof of type preservation (4.3). We conclude with a few words on related and future work (Section 5). We use a pretty-printed version of Isabelle HOL concrete syntax; a rule (a sequent) with conclusion C and premises $H_1 \dots H_n$ is represented as $[\![\; H_1; \dots ; H_n \;]\!] \Longrightarrow C$. A type declaration is $m \; :: \; [\, t_1, \dots t_n \,] \Rightarrow t$. Isabelle HOL connectives are represented via the usual logical notation, in particular implication is $\supset$. Free variables (upper-case) are implicitly universally quantified, the sign $==$ (Isabelle meta-equality) is used for *equality by definition*. The keyword **MC-Theorem** denotes a machine-checked theorem, while *Inductive* introduces an inductive relation. We have tried to use the same notation for mathematical and formalized judgments.

2 A Continuation Machine

We assume a standard version of Mini-ML constructed using higher-order abstract syntax [15]. Values are distinguished from terms by an asterisk; so **lam** is a term while **lam*** is a value.

$$\begin{array}{rl} \text{Expressions} & e ::= \mathbf{lam}\, x.\ e \mid e_1\, e_2 \mid v \\ \text{Values} & v ::= \mathbf{lam}^* x.\ e \mid x \end{array}$$

We define the continuation machine as follows:

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

We use the following transition rules for machine states:

$$\begin{array}{rcl} \mathbf{st\_init} & :: & \mathbf{init} \diamond \mathbf{return}\, v \hookrightarrow \mathbf{answer}\, v \\ \mathbf{st\_return} & :: & K; \lambda\, x.\, i \diamond \mathbf{return}\, v \hookrightarrow K \diamond i[v/x] \\ \mathbf{st\_lam} & :: & K \diamond \mathbf{ev}\, (\mathbf{lam}\, x.\ e) \hookrightarrow K \diamond \mathbf{return}\, (\mathbf{lam}^* x.\ e) \\ \mathbf{st\_app} & :: & K \diamond \mathbf{ev}\, (e_1\, e_2) \hookrightarrow K; \lambda\, x_1.\, \mathbf{app}_1\, x_1\, e_2 \diamond \mathbf{ev}\, e_1 \\ \mathbf{st\_app1} & :: & K \diamond \mathbf{app}_1\, (\mathbf{lam}^* x.\ e)\, e_2 \hookrightarrow K \diamond \mathbf{ev}\, e[e_2/x] \end{array}$$

In order to prove type preservation of this machine we need to consider sequences of transitions by taking the reflexive-transitive closure $\hookrightarrow^*$ of the above relation. Further we add typing judgments for expressions, values, instructions and continuations [4]. We report here only the last two, but their complete encoding can be found in Subsection 4.2.

$$\frac{\Gamma \vdash_v v_1 : \tau' \to \tau \qquad \Gamma \vdash_e e_2 : \tau'}{\Gamma \vdash_i \mathbf{app}_1\, v_1\, e_2 : \tau}\, ofI\_\mathbf{app}_1 \qquad \frac{\Gamma \vdash_e e : \tau}{\Gamma \vdash_i \mathbf{ev}\, e : \tau}\, ofI\_\mathbf{ev} \qquad \frac{\Gamma \vdash_v v : \tau}{\Gamma \vdash_i \mathbf{return}\, v : \tau}\, ofI\_\mathbf{return}$$

$$\frac{}{\Gamma \vdash_K \mathbf{init} : \tau \to \tau}\, ofK\_\mathbf{init} \qquad \frac{\Gamma, x{:}\tau_1 \vdash_i i : \tau \qquad \Gamma \vdash_K K : \tau \to \tau_2}{\Gamma \vdash_K K; \lambda\, x.\, i : \tau_1 \to \tau_2}\, ofK\_\mathbf{cont}$$

Theorem 1 (Subject Reduction).
$K \diamond i \ \hookrightarrow^* \ \mathbf{answer}\, v$ and $\Gamma \vdash_i i : \tau_1$ and $\Gamma' \vdash_K K : \tau_1 \to \tau_2$ implies $\cdot \vdash_v v : \tau_2$.

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

3 The Hybrid Meta-Language

We briefly recall that the theory Hybrid [1] provides support for a deep embedding of higher order abstract syntax within Isabelle HOL. In particular, it provides a model of the the untyped λ-calculus with constants. Let *con* denote a suitable type of constants. The model comprises a type *expr* together with functions

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

and two predicates $\mathsf{proper} :: expr \Rightarrow bool$ and $\mathsf{abstr} :: (expr \Rightarrow expr) \Rightarrow bool$. The elements of *expr* which satisfy proper are in one-one correspondence with the terms of the untyped λ-calculus modulo α-equivalence. The function CON is the inclusion of constants into terms, VAR is the enumeration of an infinite supply of free variables, and $$ is application. The function lambda is declared as a binder and we write $\mathsf{lambda}\ (\lambda\, v.\, e)$. For this data to faithfully represent the syntax of the un-typed λ-calculus, it must be that $\mathsf{CON}, \mathsf{VAR}, \$\$$ are injective on proper expressions, and furthermore, lambda is injective on some suitable subset of $expr \Rightarrow expr$. This cannot the whole of $expr \Rightarrow expr$ for cardinality reasons. The predicate abstr identifies those functions which are sufficiently parametric to be realized as the body of a λ-term, and lambda is injective on these. Hybrid is implemented in a definitional style using a translation into de Bruijn notation. The type *expr* is defined by the grammar

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

The translation of terms is best explained by example. Let $T_O = \Lambda\, V_1.\, \Lambda\, V_2.\, V_1\ V_3$ be an expression in the concrete syntax of the λ-calculus. This is rendered in Hybrid as $T_H = \mathsf{lambda}\ (\lambda\, v_1.\, (\mathsf{lambda}\ (\lambda\, v_2.\, (v_1\ \$\$\ \mathsf{VAR}\ 3))))$ where λv_i is Isabelle HOL's meta-abstraction. The function $\mathsf{lambda} :: (expr \Rightarrow expr) \Rightarrow expr$ is defined so as to map any function satisfying abstr to a corresponding proper de Bruijn expression. The expression T_H is reduced by higher order rewriting to the de Bruijn term $\mathsf{ABS}\ (\mathsf{ABS}\ (\mathsf{BND}\ 1\ \$\$\ \mathsf{VAR}\ 3))$. Given these definitions, the essential properties of Hybridexpressions can be proved as theorems from the properties of the underlying de Bruijn representation.

With this in place we recall how to represent the fragment of Mini-ML in question in Hybrid. First, we need constants for object-level constructors. Thus, we declare these constants (for example *cApp*) to belong to *con* and then make the following definitions:

$$\begin{array}{cc} @ \ :: \ [\, exp, exp\,] \Rightarrow exp & \mathtt{lam} \ :: \ (exp \Rightarrow exp) \Rightarrow exp \\ e_1\ @\ e_2 == \mathsf{CON}\ cAPP\ \$\$\ e_1\ \$\$\ e_2 & \mathtt{lam}\ x\,.\ E\ x == \mathsf{CON}\ cABS\ \$\$\ \mathsf{lambda}\ (\lambda\, x.\, E\ x) \end{array}$$

where `lam` is indeed an Isabelle HOL binder. As shown in [1], it is now possible to *prove* the freeness properties of constructors, for example:

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

We also need to introduce *values val* and *instructions instr* and related constructors, but not continuations. We only show their types glossing over the definitions:

$$\begin{array}{rlrl} \mathtt{lam*} & ::[\,exp, exp\,] \Rightarrow val & \mathtt{app_1} & ::[\,val, exp\,] \Rightarrow instr \\ \mathtt{ev} & ::exp \Rightarrow instr & \mathtt{return} & ::val \Rightarrow instr \end{array}$$

4 Two-level Architecture

After having introduced the HOAS syntax of our case study, we move to encoding the specification and the object logic. They will be defined via Isabelle HOL's inductive definitions and data-types. However, hypothetical judgments are encapsulated in a database of Prolog-like rules and need not be inductive. Reasoning is conducted in the SL: inversion principles are derived by the elimination rules associated to the definition of provability and of program clause and complete induction on the height of the derivation is used to simulate structural induction.

4.1 Encoding the Specification Logic

We introduce our specification logic, which corresponds to the aforementioned fragment of second-order Olli [16]:

$$\begin{array}{rl} \text{Atoms} & A \quad \text{a countably infinite set of atomic formulae} \\ \text{Goals} & G ::= A \mid A \to G \mid A \twoheadrightarrow G \mid G_1 \wedge G_2 \mid \top \mid \forall x.\, G \\ \text{Program Clauses} & P ::= \forall(A \longleftarrow [G_1, \ldots, G_m] \mathbin{;;} [G'_1, \ldots, G'_n]) \end{array}$$

where a clause $\forall(A \longleftarrow [G_1, \ldots, G_m] \mathbin{;;} [G'_1, \ldots, G'_n]$ is meant to represent the logical compilation of the universal closure of formula $G_m \to \ldots \to G_1 \to G'_n \twoheadrightarrow \ldots \twoheadrightarrow G'_1 \twoheadrightarrow A$. We choose this "compilation" to emphasize that the operational semantics of proof search will solve subgoals from innermost to outermost. Our sequents have the form

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

where Π contains the program clauses, which are unrestricted (i.e. can be used an arbitrary number of times); Γ contains unrestricted atoms; Ω contains ordered atoms; and G is the formula to be derived. The derivation rules will be completely determined by the structure of the goal. We have the usual right sequent rules to break down the goal. For atomic goals, we have two initial sequent rules, for the leaves of the derivation, and a single backchaining rule which simultaneously chooses a program formula to focus upon and derives all the ensuing sub-goals.

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

$$\frac{\Gamma A; \Omega \longrightarrow_{\Pi} G}{\Gamma; \Omega \longrightarrow_{\Pi} A \to G}\,\to_R \qquad \frac{\Gamma; \Omega A \longrightarrow_{\Pi} G}{\Gamma; \Omega \longrightarrow_{\Pi} A \twoheadrightarrow G}\,\twoheadrightarrow_R$$

$$\frac{\Gamma; \Omega \longrightarrow_{\Pi} G_1 \qquad \Gamma; \Omega \longrightarrow_{\Pi} G_2}{\Gamma; \Omega \longrightarrow_{\Pi} G_1 \wedge G_2}\,\wedge_R \qquad \frac{}{\Gamma; \Omega \longrightarrow_{\Pi} \top}\,\top_R \qquad \frac{\Gamma; \Omega \longrightarrow_{\Pi} G[a/x]}{\Gamma; \Omega \longrightarrow_{\Pi} \forall x.\, G}\,\forall^a_R$$

$$\frac{\Gamma; \cdot \longrightarrow_{\Pi} G_1 \ \ldots\ \Gamma; \cdot \longrightarrow_{\Pi} G_m \qquad \Gamma; \Omega_1 \longrightarrow_{\Pi} G'_1 \ \ldots\ \Gamma; \Omega_n \longrightarrow_{\Pi} G'_n}{\Gamma; \Omega_n \ldots \Omega_1 \longrightarrow_{\Pi} A}\,\mathbf{backchain}$$

where $A \longleftarrow [G_1 \dots G_m] \,;;\, [G'_1 \dots G'_n]$ is an instance of a program clause in Π. Note that the **backchain** 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 formulae. Furthermore, the ordering of the Ω_i, in the conclusion of the rule, is forced by our compilation of the program clauses.

The above logical language can be encoded with the Isabelle HOL datatype:

$$datatype\ oo ::= \mathsf{tt} \mid \langle atm \rangle \mid atm \rightarrow oo \mid oo\ \mathsf{with}\ oo \mid atm \twoheadrightarrow oo \mid \mathsf{all}\ (\mathsf{prpr} \Rightarrow oo)$$

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

The encoding of provability is inspired by [11] and is more general of the above calculus, as it can also handle left ordered implication. Given an inductive definition of a predicate for order-preserving split of a context $(osplit\ \Omega\ \Omega_L\ \Omega_R)$, the above sequent calculus is defined with three mutually inductive definitions:

$$\begin{aligned}
\Gamma \,;;\, \Omega \rhd_n G &:: [\, atm\ list, atm\ list, nat, oo \,] \Rightarrow bool \\
\Gamma \rhd_n Goals &:: [\, atm\ list, nat, oo\ list \,] \Rightarrow bool \\
\Gamma \,;;\, (\Omega_L ; \Omega_R) \rhd_n Goals &:: [\, atm\ list, atm\ list, atm\ list, nat, oo\ list \,] \Rightarrow bool
\end{aligned}$$

where the list judgments are employed in the implementation of the backchain rule:

$$[\![\ A \longleftarrow O_L \,;;\, I_L\ \&\ (osplit\ \Omega\ \Omega_L\ \Omega_R)\ \&\ \Gamma \,;;\, (\Omega_L ; \Omega_R) \rhd_i O_L\ \&\ \Gamma \rhd_i I_L\ ]\!] \Longrightarrow \Gamma \,;;\, \Omega \rhd_{i+1} \langle A \rangle$$

Ordered list consumption is as follows:

$$\begin{aligned}
&\Longrightarrow \Gamma \,;;\, ([\,];[\,]) \rhd_i [\,] \\
[\![\ (osplit\ \Omega_R\ \Omega_G\ \Omega_r)\ \&\ \Gamma \,;;\, \Omega_G \rhd_i G\ \&\ \Gamma \,;;\, (\Omega_L ; \Omega_r) \rhd_i Gs\ ]\!] &\Longrightarrow \Gamma \,;;\, (\Omega_L ; \Omega_R) \rhd_{i+1} G \# Gs
\end{aligned}$$

while unordered list consumption is analogous, but behaves additively. The rest of the sequent rules are as expected and left to the on-line documentation. Note that in this logic implications have only atomic antecedents which therefore yields only atomic contexts. Atoms are provable either by assumption (intuitionistic or ordered) or via *backchaining.* The notation $A \longleftarrow O_L \,;;\, I_L$ corresponds to an inductive definition of a set `prog` of type $[\, atm, oo\ list, oo\ list \,] \Rightarrow bool$, see Subsection 4.2 for examples. The sequent calculus is parametric in those clauses and so are its meta-theoretical properties. Sequents are decorated with natural numbers which represent the *height* of a proof; this measure allows reasoning by complete induction. For convenience we define $\Gamma \,;;\, \Omega \rhd G$ iff there exist n such that $\Gamma \,;;\, \Omega \rhd_n G$ and $\rhd G$ iff $[\,] \,;;\, [\,] \rhd G$. Similarly for the other judgments. The very fact that provability is inductive makes available *inversion principles* as elimination rules of the aforementioned definitions.

MC-Theorem 1 (Structural Rules). *The following rules are admissible:*

1. *Weakening for numerical bounds:* $[\![\ \Gamma \,;;\, \Omega \rhd_n G;\ n < m\ ]\!] \Longrightarrow \Gamma \,;;\, \Omega \rhd_m G$ *and* $[\![\ \Gamma \,;;\, (\Omega_L ; \Omega_R) \rhd_n Goals;\ n < m\ ]\!] \Longrightarrow \Gamma \,;;\, (\Omega_L ; \Omega_R) \rhd_m Goals$ *and* $[\![\ \Gamma \rhd_n Goals;\ n < m\ ]\!] \Longrightarrow \Gamma \rhd_m Goals.$
2. *Context weakening:* $[\![\ \Gamma \,;;\, \Omega \rhd G;\ \Gamma \subseteq \Gamma'\ ]\!] \Longrightarrow \Gamma' \,;;\, \Omega \rhd G$ *and* $[\![\ \Gamma \,;;\, (\Omega_L ; \Omega_R) \rhd Goals;\ \Gamma \subseteq \Gamma'\ ]\!] \Longrightarrow \Gamma' \,;;\, (\Omega_L ; \Omega_R) \rhd Goals$ *and* $[\![\ \Gamma \rhd Goals;\ \Gamma \subseteq \Gamma'\ ]\!] \Longrightarrow \Gamma' \rhd Goald.$

Proof. The proof is by a fully automated mutual structural induction on the three sequents judgments.

The sequent calculus in [16] enjoys various forms of cut-elimination. For the sake of the type preservation proof (Theorem 2) we only need the following atomic intuitionistic cut: $[\![\ (\Gamma\#A) ;; [\,] \rhd G; \ \ \rhd \langle A\rangle \ ]\!] \Longrightarrow \Gamma ;; [\,] \rhd G$ and similarly for judgments on lists. This proof is work-in-progress.

4.2 Encoding the Object Logic

We now show how the continuation machine can be written as an Olli program. Rather than building an explicit stack-like structure to represent the continuation K, we will simply store instructions in the ordered context. Thus we will use the following representation to encode the machine:

$$K \diamond i \qquad \rightsquigarrow \qquad \ulcorner K \urcorner \Longrightarrow \ulcorner i \urcorner$$

where $\ulcorner K \urcorner$ is the representation, described below, of the continuation (stack) K and similarly for $\ulcorner i \urcorner$.

Given the goal: $\mathsf{init}\ V \twoheadrightarrow \mathsf{ex}\ (\mathtt{ev}\ e)$ our program will evaluate the expression e and instantiate V with the resulting value. The intended reading of this query is: evaluate e with the initial continuation (the continuation which just returns its value). A goal of $\mathsf{ex}\ (\mathtt{return}\ i)$ is intended to mean: execute instruction i. A goal of $\mathsf{ex}\ (\mathtt{return}\ v)$ is intended to mean: pass v to the top continuation on the stack (i.e. the rightmost element in the ordered context).

We have the following representations:

$$\mathbf{init} \diamond \mathbf{return}\, v \qquad \rightsquigarrow \qquad \mathsf{init}\ W \Longrightarrow \mathsf{ex}\ (\mathtt{return}\ \ulcorner v \urcorner)$$

where the logic variable V is the final answer;

$$K; \lambda\, x.\, i \diamond \mathbf{return}\, v \qquad \rightsquigarrow \qquad \ulcorner K \urcorner\, (\mathsf{cont}\ (\lambda\, x.\, \ulcorner i \urcorner)) \Longrightarrow \mathsf{ex}\ (\mathtt{return}\ \ulcorner v \urcorner)$$

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

As usual in the two-level approach [13] we introduce a datatype $\mathtt{atm}$ to encode the atomic formulae of the OL, which in this case study consists of:

$$\begin{array}{l} \textit{datatype atm} ::= \mathsf{ceval}\ \textit{exp val} \mid \mathsf{ex}\ \textit{instr} \mid \mathsf{init}\ \textit{val} \mid \mathsf{cont}\ (\textit{val} \Rightarrow \textit{instr}) \mid \\ \qquad\qquad\qquad\ \ \mathsf{of}\ \textit{exp tp} \mid \mathsf{ofI}\ \textit{instr tp} \mid \mathsf{ofV}\ \textit{val tp} \mid \mathsf{ofK}\ \textit{tp} \end{array}$$

We can now give the clauses for the OL deductive systems, starting with typing:

$$\begin{array}{rl}
\textit{Inductive}\ \_ \longleftarrow \_ ;; \_ \ :: & [\,\textit{atm}, \textit{oo list}, \textit{oo list}\,] \Rightarrow \textit{bool} \\
& \Longrightarrow \mathsf{of}\ (E_1\ \mathtt{@}\ E_2)\ T \longleftarrow [\,] ;; [\langle \mathsf{of}\ E_1\ (T'\ \mathtt{arrow}\ T)\rangle, \langle \mathsf{of}\ E_2\ T'\rangle] \\
[\![\ \mathsf{abstr}\ E\ ]\!] & \Longrightarrow \mathsf{of}\ (\mathtt{lam}\ x\,.\ E\ x)\ (T_1\ \mathtt{arrow}\ T_2) \longleftarrow [\,] ;; [\mathsf{all}\, x.\, (\mathsf{of}\ x\ T_1) \rightarrow \langle \mathsf{of}\ (E\ x)\ T_2\rangle] \\
[\![\ \mathsf{abstr}\ E\ ]\!] & \Longrightarrow \mathsf{ofV}\ (\mathtt{lam*}\ E)\ T \longleftarrow [\,] ;; [\mathsf{of}\ (\mathtt{lam}\ E)\ T] \\
& \Longrightarrow \mathsf{ofI}\ (\mathtt{ev}\ E)\ T \longleftarrow [\,] ;; [\langle \mathsf{of}\ E\ T\rangle] \\
& \Longrightarrow \mathsf{ofI}\ (\mathtt{return}\ V)\ T \longleftarrow [\,] ;; [\langle \mathsf{ofV}\ V\ T\rangle] \\
& \Longrightarrow \mathsf{ofI}\ (\mathtt{app}_1\ V\ E)\ T \longleftarrow [\,] ;; [\langle \mathsf{ofV}\ V\ (T_2\ \mathtt{arrow}\ T)\rangle, \langle \mathsf{of}\ E\ T_2\rangle] \\
& \Longrightarrow \mathsf{ofK}\ (T\ \mathtt{arrow}\ T) \longleftarrow [\langle \mathsf{init}\ V\rangle] ;; [\langle \mathsf{ofV}\ V\ T\rangle] \\
[\![\ \mathsf{abstr}\ K\ ]\!] & \Longrightarrow \mathsf{ofK}\ (T_1\ \mathtt{arrow}\ T_2) \longleftarrow [\langle \mathsf{cont}\ K\rangle, \langle \mathsf{ofK}\ T\ \mathtt{arrow}\ T_2\rangle] ;; \\
& \qquad\qquad\qquad\qquad\qquad [\mathsf{all}\, v.\, (\mathsf{ofV}\ v\ T_1) \rightarrow \langle \mathsf{ofI}\ (K\ v)\ T\rangle]
\end{array}$$

Typing judgments are intuitionistic, except typing of continuations. The judgments for expressions, values and instructions directly encode the corresponding judgments and derivation rules. The judgments for continuations differ from their analogs in Section 2 in that there is no explicit continuation being typed; 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.

We now show the evaluation clauses of the program, which fully takes advantage of ordered contexts. The first one is just a wrapper to put queries into the correct form. The rest directly mirror the machine transition rules:

$$\begin{aligned}
&\Longrightarrow \mathsf{ceval}\ E\ V \longleftarrow [\mathsf{init}\ V \twoheadrightarrow \mathsf{ex}\ (\mathsf{ev}\ E)]\,;;[\,]\\
&\Longrightarrow \mathsf{ex}\ (\mathtt{return}\ V) \longleftarrow [\langle \mathsf{init}\ V\rangle]\,;;[\,]\\
[\![\ \mathsf{abstr}\ E\ ]\!] &\Longrightarrow \mathsf{ex}\ (\mathtt{return}\ V) \longleftarrow [\langle \mathsf{cont}\ E\rangle, \langle \mathsf{ex}\ (E\ V)\rangle]\,;;[\,]\\
[\![\ \mathsf{abstr}\ E\ ]\!] &\Longrightarrow \mathsf{ex}\ (\mathtt{ev}\ (\mathtt{lam}\ E)) \longleftarrow [\langle \mathsf{ex}\ (\mathtt{return}\ (\mathtt{lam}{*}\ E))\rangle]\,;;[\,]\\
&\Longrightarrow \mathsf{ex}\ (\mathtt{ev}\ (E_1\ \mathtt{@}\ E_2)) \longleftarrow [\mathsf{cont}\ (\lambda v.\ \mathsf{app}_1\ v\ E_2) \twoheadrightarrow \langle \mathsf{ex}\ (\mathtt{ev}\ E_1)\rangle]\,;;[\,]\\
[\![\ \mathsf{abstr}\ E\ ]\!] &\Longrightarrow \mathsf{ex}\ (\mathsf{app}_1\ (\mathtt{lam}{*}\ E)\ E_2) \longleftarrow [\langle \mathsf{ex}\ (\mathtt{ev}\ (E\ E_2))\rangle]\,;;[\,]
\end{aligned}$$

Note the presence of the *abstraction* annotations as Isabelle HOL premises in rules mentioning binding construct. This in turn allows to simulate definitional reflection via the built-in elimination rules of the `prog` inductive definition without the use of freeness axioms [8].

4.3 Formal Verification of Type Preservation

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

MC-Theorem 2.

$$\begin{aligned}
&[\,]\,;;\mathsf{init}\ V, \Omega\ \ \rhd_i \langle \mathsf{ex}\ I\rangle \Longrightarrow\\
&\quad \forall T_1 T_2 \Omega'.\,(\ \rhd \langle \mathsf{ofI}\ I\ T_1\rangle) \supset .\,([\,]\,;;\mathsf{init}\ V, \Omega' \rhd \langle \mathsf{ofK}\ (T_1\ \mathtt{arrow}\ T_2)\rangle) \supset (\ \rhd \langle \mathsf{ofV}\ V\ T_2\rangle)
\end{aligned}$$

Proof. The proof is by complete induction on the height of the derivation. Legenda:

$[\,]\,;;\mathsf{init}\ V, \Omega \rhd_i \langle \mathsf{ex}\ I\rangle$ corresponds to	`[] ;; init V * Ome |- < ex I > ::: i`
$\rhd \langle \mathsf{ofI}\ I\ T_1\rangle$ corresponds to	`[] ;; [] |-- < ofI I T1 >`
$[\,]\,;;(\Omega_l;\Omega_r) \rhd_i Goals$ corresponds to	`[] ;; Ol ;; Or ||> Goals ::: i`

The induction hypothesis (which will be elided next) is

```
[| ALL m. m < n -->
      (ALL I V Ome.
          [] ;; init V * Ome |- < ex I > ::: m -->
          (ALL T1 T2 Ome'.
              [] ;; [] |-- < ofI I T1 >  &
              [] ;; [init V * Ome'] |-- < ofK (T1 ar T2) >  -->
              [] ;; [] |-- < ofV V T2 > ));
```

We begin by inverting on `[] ;; [] ;; init V * Ome |- < ex I> ::: n` and then on the `prog` clauses, yielding several goals each for each evaluation clause. Let's look at a simple case for `lam`:

```
[| osplit (init V * Ome) Ol Or;
   [] ;; Ol ;; Or ||> [< ex (return (lam* E)) >, True)] ::: ia;
   [] ;; [] |-- < ofI (ev (lam E)) T1 > ;
   [] ;; [init V * Ome'] |-- < ofK (T1 ar T2) > ; abstr E |]
==> [] ;; [] |-- < ofV V T2 >
```

First we prove that `T1` must be of functional form `T1' ar T2'`. Note that this requires several inversion steps as we need to move back and forth between the `prog` and provability level. Then we invert on splitting, yielding two goals, the first one of which we try to show impossible. Case one: `Ol = init V * L2` and `Or = []`.

```
 [] ;; init V * L2 ;; Or ||> [< ex (return (lam* E)) >, True)] ::: ia;
          osplit Ome L2 Or .... |]
       ==> [] ;; [] |-- < ofV V T2 >
```

Inverting again on the first assumptions yields:

```
  [] ;; Og |- < ex (return (lam* E)) > ::: ia;
          [] ;; init V * L2 ;; Or' ||> [] ::: ia ... |] ==> ...
```

This is contradictory, since we have `init V * L2 = []`. The second case has `Or = init V * L2` and `Ol = []`. On more inversion and

```
 osplit (init V * Ome) Og Or';
          [] ;; Og |- < ex (return (lam* E)) > ::: ia;
          [] ;; [] ;; Or' ||> [] ::: ia |]
       ==> [] ;; [] |-- < ofV V T2 >
```

Here we note that it must be `Or' = []`, so that inversion on `osplit (init V * Ome) Og []` yields only one case:

```
...  [] ;; [] |-- < ofI (ev (lam E)) (T1' ar T2') > ;
          [] ;; [init V * L''] |-- < ofK ((T1' ar T2') ar T2) > ; abstr E;
          [] ;; init V * L2 |- < ex (return (lam* E)) > ::: ia;
          osplit Ome L2 [] |]
       ==> [] ;; [] |-- < ofV V T2 >
```

This follows by IH and the fact that:

```
[] ;; [] |-- < ofI (ev (lam E)) (T1 ar T2) > ==>
  [] ;; [] |-- < ofI (return (lam* E)) (T1 ar T2) >
```

Corollary 1 (Subject Reduction). $[\![\, \rhd \mathsf{ceval}\ E\ V;\ \rhd \mathsf{of}\ E\ T \,]\!] \Longrightarrow (\rhd \mathsf{ofV}\ V\ T)$.

5 Related Work and Conclusions

We refer to [1] for a review of related work about HOAS. The present paper generalizes the approach in [13], which in turn was inspired by [8, 11]. The latter in particular presents a two-level proof of type preservation for MLR in a second-order linear specification logic. This is a variant of the proof implemented in the linear logical framework LLF [4], where, in the Elf tradition, a meta-theorem is a relation (type family) between judgments whom the logic programming-like interpretation provides an operational semantics to. Finally, external coverage checking (which are, for the time being, limited to LF [22]) verifies that the given relation is indeed a realizer for that theorem. Only as we speak, $\mathcal{M}_\omega$ [20], the meta-logic of LF has been extended to $\mathcal{L}^+_\omega$, a meta-logic for LLF [10].

In the same vein, Polakow and Pfenning [17] have used an Ordered Logical Framework to formally show that terms resulting from a CPS translation obey stack-like ordering properties with respect to intermediate values [7, 6]. Polakow and Yi [18] later extended these techniques to a CPS translation for a language with exceptions which employed a continuation pair (one for success, one for failure). While both of these efforts carried out a formal proof in OLF, neither of them were automated in any way.

We have presented a two-level approach to formalize an ordered logical framework on top of the Hybrid system, which allows inductive reasoning about objects defined via HOAS in a well-known environment such as Isabelle HOL. This replicates, in a well-understood and interactive setting, the style of proof of $FO\lambda^{\Delta I\!N}$, so all results are proven without "technical" lemmas foreign to the mathematics of the problem. The specification logic with its meta-theoretical properties are proven once and for all and it can be varied depending on the application under study without changing infrastructure.

As we said in the Introduction, this paper is merely a stepping stone towards investigating the meta-theory of languages with imperative features. The next objective is to replay the cited proof of type preservation for MLR, which will be simplified by the internalization of the instruction stack. From this viewpoint it is somewhat disappointing that we still have to retain a notion of *typing* of continuations ofK T. An intriguing possibility is to move to the third-oder machine described in [16], which, by using third-order clauses and left implication $\twoheadleftarrow$, does not need the level of instructions. For example evaluating an application would be in Olli syntax:

```
ev_app : ev (E1 @ E2) <<- ({V1} return V1 <-< app1 V1 E2) ->> (ev E1)
```

where now `app1 :: val -> exp -> bool, return :: val -> bool, ev :: exp -> bool`. This would entail some changes in the SL, generalizing the structure of implicational clauses and of backchaining. In a sense, there seems to be a natural progression from intuitionistic second-order logic [13] to second and finally third order ordered linear logic whereby we simplify the machine, first internalizing explicit continuations in the CPM [15] then removing instructions by making use of the more expressive logic.

The possibility of handling in such an elegant fashion *both state and order* opens up literally dozens of applications, typically abstract machines, which up to now have been encoded in a rather indirect fashion. One long term project is the verification of properties of typed monadic intermediate languages, such as MIL-lite [3], possibly deriving an abstract machine following [5]. Further there are several other applications beyond programming languages for an ordered framework such as GSOS with priorities [23]. The latter may require a more sophisticated notion of order, such as branching. It is conceivable that this could be mirrored by refining linear ordered context in the sense of *bunches* in BI [19].

As far as the infrastructure is concerned, note that similarly to [13] in this case study we only needed to induct *closed* terms, although we reason (typically by inversion) in presence of hypothetical judgments. Inducting HOAS-style over open terms is a major challenge [20]; in this setting *generic* judgments are particularly problematic, but can be dealt with by switching to a more expressive SL, based on a eigenvariable encoding [12]. The new theory of *terms-in-infinite-context* underlying the new version of Hybrid [2] directly supports this syntax. With that in place, we will be able, for example, to replay in a full HOAS style a notion of program equivalence based on bisimilarity [14] and finally approach at the right level of abstraction the verification of the compiler optimizations of MIL-lite [3].

Source files for the Isabelle HOL code can be found at

`www.mcs.le.ac.uk/~amomigliano/isabelle/2Levels/Oll2`

Acknowledgments Alberto Momigliano is supported by EPSRC grant GR/M98555. Jeff Polakow is supported by grant IST-2001-33477.

References

1. S. Ambler, R. Crole, and A. Momigliano. Combining higher order abstract syntax with tactical theorem proving and (co)induction. In V. A. Carreño, editor, *Proceedings of the*

15th International Conference on Theorem Proving in Higher Order Logics, Hampton, VA, 1-3 August 2002, volume 2342 of *LNCS*. Springer Verlag, 2002.

2. S. Ambler, R. Crole, and A. Momigliano. A definitional approach to primitive recursion over higher order abstract syntax. 2003. Submitted.
3. N. Benton and A. Kennedy. Monads, effects and transformations. *Electronic Notes in Theoretical Computer Science*, 26, 1999.
4. I. Cervesato and F. Pfenning. A linear logical framework. *Information and Computation*, 1998. To appear in a special issue with invited papers from LICS'96, E. Clarke, editor.
5. O. Danvy and al. A functional correspondence between evaluators and abstract machines. 2003. To appear at PPDP'03.
6. O. Danvy, B. Dzafic, and F. Pfenning. On proving syntactic properties of CPS programs. In A. Gordon and A. Pitts, editors, *Proceedings of HOOTS'99*, Paris, Sept. 1999. Electronic Notes in Theoretical Computer Science, Volume 26.
7. O. Danvy and F. Pfenning. The occurrence of continuation parameters in CPS terms. Technical Report CMU-CS-95-121, Department of Computer Science, Carnegie Mellon University, Feb. 1995.
8. A. Felty. Two-level meta-reasoning in Coq. In V. A. Carreño, editor, *Proceedings of the 15th International Conference on Theorem Proving in Higher Order Logics, Hampton, VA, 1-3 August 2002*, volume 2342 of *LNCS*. Springer Verlag, 2002.
9. J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
10. A. McCreight and C. Schürmann. A meta linear logica framework. Draft, 2003.
11. R. McDowell and D. Miller. Reasoning with higher-order abstract syntax in a logical framework. *ACM Transactions on Computational Logic*, 3(1):80–136, January 2002.
12. D. Miller and A. Tiu. Encoding generic judgments. In M. Agrawal and A. Seth, editors, *FST TCS 2002: Foundations of Software Technology and Theoretical Computer Science*, number 2556 in LNCS, pages 18–34. Springer-Verlag, 2002.
13. A. Momigliano and S. 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.
14. A. Momigliano, S. Ambler, and R. Crole. A Hybrid encoding of Howe's method for establishing congruence of bisimilarity. *ENTCS*, 70(2), 2002.
15. F. Pfenning. Computation and deduction. Lecture notes, 277 pp. Revised 1994, 1996, to be published by Cambridge University Press, 1992.
16. J. Polakow. *Ordered Linear Logic and Applications*. PhD thesis, CMU, 2001.
17. J. Polakow and F. Pfenning. Properties of terms in continuation-passing style in an ordered logical framework. In J. Despeyroux, editor, *2nd Workshop on Logical Frameworks and Meta-languages (LFM'00)*, Santa Barbara, California, June 2000. Proceedings available as INRIA Technical Report.
18. J. Polakow and K. Yi. Proving syntactic properties of exceptions in an ordered logical framework. In H. Kuchen and K. Ueda, editors, *Proceedings of the 5th International Symposium on Functional and Logic Programming (FLOPS'01)*, pages 61–77, Tokyo, Japan, Mar. 2001. Springer-Verlag LNCS 2024.
19. D. J. Pym. On bunched predicate logic. In G. Longo, editor, *Proceedings of the 14th Annual Symposium on Logic in Computer Science (LICS'99)*, pages 183–192, Trento, Italy, July 1999. IEEE Computer Society Press.
20. C. Schürmann. *Automating the Meta-Theory of Deductive Systems*. PhD thesis, Carnegie-Mellon University, 2000. CMU-CS-00-146.
21. C. Schürmann. A type-theoretic approach to induction with higher-order encodings. In *Proceedings of Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2001)*, volume 2142 of *Lecture Notes in Computer Science*, pages 266–281, 2001.
22. C. Schürmann and F. Pfenning. A coverage checking algorithm for LF. 2003. To appear at TPHOLs, Roma, Italy, September 2003.
23. I. Ulidowski and I. Phillips. Ordered SOS process languages for branching and eager bisimulations. *Information and Computation*, 178, 2002.