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

[Extended Abstract]

Alberto Momigliano
School of Informatics, University of Edinburgh
King's Buildings
Edinburgh EH9 3JZ, Scotland
amomigl1@inf.ed.ac.uk

Jeff Polakow
Department of Computer Science, Heriot-Watt University
Riccarton, Edinburgh EH14 4AS, Scotland
jp@macs.hw.ac.uk

ABSTRACT

We report on work in progress devoted to the formalization of an Ordered Logical Framework (OLF) [16] based on a two-levels architecture [10] in the Hybrid system [2]. 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 roughly 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.

Categories and Subject Descriptors

D.3.1 [**Programming Languages**]: Formal Definitions and Theory; F.4.1 [**Mathematical Logic**]: Lambda Calculus and Related Systems—*Mechanical theorem proving, Proof theory*; I.2.3 [**Artificial Intelligence**]: Deduction and Theorem Proving—*Deduction, Inference engines, logic programming, meta theory*

General Terms

Theory, Languages, Verification

Keywords

Continuation machines, higher order abstract syntax, logical frameworks, ordered linear logic

1. INTRODUCTION

Much current research is concerned with how to encode and formally verify the (meta)theory of languages with variable binding. Sometimes, however, the main issue is merely which kind of *syntax* best fits the bill, spanning from named syntax to De Bruijn indexes, to various forms of higher-order abstract syntax (HOAS). While this choice is a fundamental one, which is bound to heavily influence any subsequent formal development, the methodology with which one encodes *judgements* is equally important. Indeed, as initially suggested by Martin-Löf and practised in the Edinburgh Logical Framework (LF), a (full) HOAS syntax naturally leads to (and benefits from) the pervasive use of hypothetical and parametric judgments to encode object level "relations-in-context". The benefits as well as the challenges associated with this methodology are by now well-known [15]. Roughly, 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 solutions, with various degrees of success, been proposed [20, 10, 12].

As fruitful as this has been for frameworks based on intuitionistic logic, it soon became apparent that some of the structural properties of such a meta-logic, namely weakening and contraction, are inherited by object level encodings, and this may not always be appropriate for every domain; case in point being when the notion of (updatable) *state* is paramount. To fix ideas, let us consider the case of encoding the static and dynamic semantics of a functional programming language with imperative features, say Mini-ML with references (MLR). While encoding the typing system with an intuitionistic context is adequate, a declarative representation of the store requires a mutable notion of context. If the store is modelled by a set of assumptions, say `contains C V`, where `C` is a cell and `V` a value, then *linear logic* is the natural choice, since updates can be obtained via linear operations that consume and/or add resources. This is one of the motivations for proposing frameworks based on linear logics such as LLF [5]. On the other hand, work on the *automation* of reasoning in such frameworks is still in its infancy [9].

In this paper we propose to push further this methodology that aims to refine, in a *conservative* way, a logical framework so as to capture, declaratively and at the right level of abstraction, different object level phenomena. Here we adopt an *ordered logical framework* (OLF) [16], by which we mean, in this incarnation, a second-order minimal ordered

MERλIN'03, August 26, 2003, Uppsala, Sweden

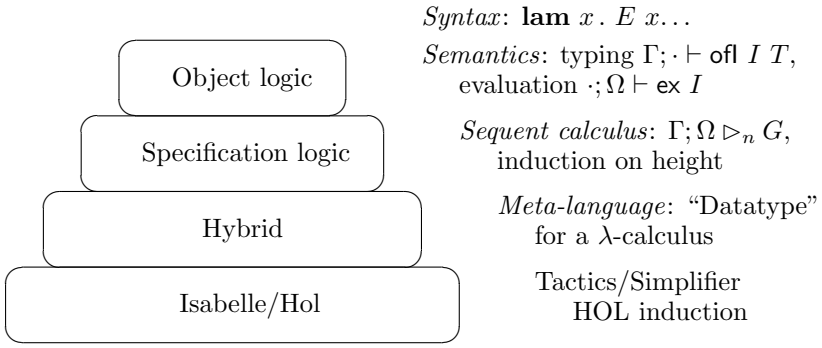

Figure 1: 2-Levels Architecture

linear logic. Ordered (formerly known as non-commutative) linear logic combines reasoning with unrestricted, 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 meta-theory of programming languages so far.

This additional expressive power allows, e.g., the possibility for the logic to directly handle the notion of *stack*. 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 [5] is based on a continuation machine. As we detail in Section 2, this entails, however, the introduction of additional layers of *instructions, continuations and states* compared to the text-book presentation of the operational semantics based on configurations. Moreover, those entities need to be typed, in a sense complicating the set-up. On the other hand, we argue that, by using a form of OLF, the notion of continuation can be disposed of via internalization in an ordered context, in analogy on how the notion of state is internalized in the linear context. In particular, the ordered context operates as a stack of continuations to be evaluated.

The contribution of this paper is in the implementation of OLF and its use for meta-reasoning according to the $FO\lambda^{\Delta IN}$ two-levels architecture suggested in [10] and adopted in [12], which we refer to for more details. The basis is the higher-order meta-language Hybrid [2], within Isabelle HOL that provides a form of HOAS to represent object logics. The HOAS user level is separated from the infrastructure that is implemented *definitionally* by means of a de Bruijn style encoding. The two-levels 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); this, 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 in hypothetical judgments, since the latter are now encapsulated within the OL and therefore not required to be inductive. The architecture is depicted in Figure 1.

We can view our realization of the two-levels approach as a way of "fast prototyping" HOAS logical frameworks. We can implement very quickly and experiment with a SL, in particular we can do meta-reasoning in a way compatible with induction and tactical theorem proving. For example, 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, requiring thusly a sort of meta-interpreter (the SL logic) to drive it. This indirectness can be alleviated by defining appropriate tactics, but this is intrinsic in the design choice of relying on a fairly general meta-meta-logic (here HOL, in [10] some variation of $FO\lambda^{\Delta IN}$). This contrasts with the architecture proposed in [9], where the meta-meta-logic is itself sub-structural and explicitly tailored to the automation of a linear framework.

Our current goal is investigating the meta-theory of languages with imperative features, starting with the formal verification of the proof of type preservation for a language such as 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 presentation, we will restrict ourselves to the 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 omit references to the (unordered) linear context and linear implication, and to 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 the mathematical level, the continuation machine. Section 3 recalls some basic notions of Hybrid and its syntax representing techniques. In Section 4 we introduce the two-levels architecture, describing the SL (4.1) and the OL (4.2), culminating in the formal verification of the proof of type preservation. We conclude with a few words on related (Section 5) and future work (Section 6).

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 according to the usual logical notation, in particular implication is $\supset$. Free variables (upper-case) are implicitly universally quantified, the sign $\equiv$ (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 use the continuation machine for Mini-ML formulated in [14], which we refer to for motivation and full details. Values are distinguished from terms by an asterisk; so $\mathbf{lam}\, x.\ e$ is a term while $\mathbf{lam}^* x.\ e$ is a value.

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

We define the machine as follows:

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

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 may 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 [5]. These can be found in Figure 2, their complete encoding in Figure 4.

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

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

and two predicates $\mathsf{proper} :: expr \Rightarrow bool$ and $\mathsf{abstr} :: (expr \Rightarrow expr) \Rightarrow bool$. The elements of *expr* that 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. For this data to faithfully represent the syntax of the un-typed λ-calculus, it must be that CON, VAR, \$\$ are injective on proper expressions, and furthermore, lambda is injective on some suitable subset of $expr \Rightarrow expr$. This cannot the whole of $expr \Rightarrow expr$ for cardinality reasons. The predicate abstr identifies those functions that 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

$$\begin{array}{rcl} expr & ::= & \mathsf{CON}\ con \mid \mathsf{VAR}\ var \mid \mathsf{BND}\ bnd \\ & & \mid expr\ \$\$\ expr \mid \mathsf{ABS}\ expr \end{array}$$

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 rewriting to the term $\mathsf{ABS}\,(\mathsf{ABS}\,(\mathsf{BND}\ 1\ \$\$\ \mathsf{VAR}\ 3))$. Given these definitions, the essential properties of Hybrid expressions can be proved as theorems from the properties of the underlying de Bruijn representation.

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:

$$\begin{array}{rcl} \texttt{@} & :: & expr \Rightarrow expr \Rightarrow expr \\ e_1 \texttt{@}\, e_2 & == & \mathsf{CON}\ cAPP\ \$\$\ e_1\ \$\$\ e_2 \\ \texttt{lam} & :: & (exp \Rightarrow exp) \Rightarrow exp \\ \texttt{lam}\ x\ .\ E\ x & == & \mathsf{CON}\ cABS\ \$\$\ \mathsf{lambda}\,(\lambda x.\, E\ x) \end{array}$$

where lam is indeed an Isabelle HOL binder. As shown in [2], it is now possible to *prove* the freeness properties of constructors:

MC-THEOREM 1. *The constructors have distinct images; for example:*

$$\texttt{lam}\ x\ .\ E\ x \neq e_1\ \texttt{@}\, e_2$$

Furthermore, every binding *constructor is injective on abstractions; for example:*

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

PROOF. By Isabelle HOL's simplification, using injectivity of Hybrid abstraction in the binding cases. □

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:

$$\begin{array}{rcl} \texttt{lam*} & :: & exp \Rightarrow exp \Rightarrow val \\ \texttt{app}_1 & :: & val \Rightarrow exp \Rightarrow instr \\ \texttt{ev} & :: & exp \Rightarrow instr \\ \texttt{return} & :: & val \Rightarrow instr \end{array}$$

4. TWO-LEVELS ARCHITECTURE

After having introduced the HOAS syntax of our case study, we move to encoding the specification and the object logic. We will define them using Isabelle HOL's inductive definitions and data-types. This is possible because the HOAS syntax is only a definition of standard first-order terms. However, hypothetical judgments are encapsulated in a database of Prolog-like rules, because they may not be inductive. Reasoning is conducted in the SL: inversion principles are derived by the elimination rules associated to the

$$\frac{x{:}\tau \in \Gamma}{\Gamma \vdash_e x : \tau}\, of\_\mathbf{var} \qquad \frac{\Gamma, x{:}\tau \vdash_e e : \tau'}{\Gamma \vdash_e \mathbf{lam}\, x.\ e : \tau \to \tau'}\, of\_\mathbf{lam} \qquad \frac{\Gamma \vdash_e e_1 : \tau' \to \tau \qquad \Gamma \vdash_e e_2 : \tau'}{\Gamma \vdash_e e_1\, e_2 : \tau}\, of\_\mathbf{app}$$

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

$$\frac{\Gamma, x{:}\tau \vdash_e e : \tau'}{\Gamma \vdash_v \mathbf{lam}^* x.\ e : \tau \to \tau'}\, of\_\mathbf{lam}^*$$

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

Figure 2: Typing Rules for the Continuation Machine

definition of provability and of program clause; we use complete induction on the height of the derivation 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]; in particular '$\twoheadrightarrow$' denotes right-ordered implication:

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

where a clause $\forall (A \longleftarrow [G_1, \ldots, G_m] ;; [G'_1, \ldots, G'_n])$ is meant to represent the logical compilation of the universal closure of the 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 rules are depicted in Figure 3. The derivation rules ware completely determined by the structure of the goal. Note that in this fragment of the logic implications have only atomic antecedents and this therefore yields only atomic contexts. 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 that simultaneously chooses a program formula to focus upon and derives all the ensuing sub-goals; rule (**bc**) applies the *instance* of a program clause $A \longleftarrow [G_1 \ldots G_m] ;; [G'_1 \ldots G'_n]$. 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 formulae. Furthermore, the ordering of the Ω_i, in the conclusion of the rule, is forced by our compilation of the program clauses.

We encode the above logical language with the Isabelle HOL datatype:

$$\begin{array}{rrcl} \textit{datatype} & oo & ::= & \mathsf{tt} \mid \langle atm \rangle \mid atm \to oo \mid oo \ \mathsf{with}\ oo \\ & & & \mid atm \twoheadrightarrow oo \mid \mathsf{all}\ (\mathsf{prpr} \Rightarrow oo) \end{array}$$

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 [10] and is specialized to right ordered implication. We use three mutually inductive definitions:

$$\begin{array}{rcl} \Gamma ;; \Omega \rhd_n G & :: & [\, atm\ list, atm\ list, nat, oo \,] \Rightarrow bool \\ \Gamma \rhd_n Goals & :: & [\, atm\ list, nat, oo\ list \,] \Rightarrow bool \\ \Gamma ;; \Omega \rhd_n Goals & :: & [\, atm\ list, atm\ list, nat, oo\ list \,] \Rightarrow bool \end{array}$$

where the natural number decoration represents the *height* of a proof, facilitating reasoning by complete induction.

The implementation of the sequent rules (first judgment) are completely unsurprising, except maybe for the backchain rule:

$$[\![\ A \longleftarrow O_L ;; I_L \ \&\ \Gamma ;; \Omega \rhd_n O_L \ \&\ \Gamma \rhd_n I_L\ ]\!] \Longrightarrow \Gamma ;; \Omega \rhd_{n+1} \langle A \rangle$$

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 Figure 4 for examples. Incidentally, the sequent calculus is parametric in those clauses and so are its meta-theoretical properties. Backchaining uses two list judgments to encode execution of the (compiled) body of a clause. Intuitionistic list consumption is immediate:

$$\begin{array}{rcl} & \Longrightarrow & \Gamma \rhd_n [\,] \\ [\![\ \Gamma \rhd_n G \ \&\ \Gamma \rhd_n Gs\ ]\!] & \Longrightarrow & \Gamma \rhd_{n+1} G * Gs \end{array}$$

Ordered list consumption is analogous, but behaves multiplicatively w.r.t. the ordered context:

$$\begin{array}{c} \Longrightarrow \Gamma ;; [\,] \rhd_n [\,] \\ [\![\ osplit\ \Omega\ \Omega_R\ \Omega_G \ \&\ \Gamma ;; \Omega_G \rhd_n G \ \&\ \Gamma ;; \Omega_R \rhd_n Gs\ ]\!] \\ \Longrightarrow \Gamma ;; \Omega \rhd_{n+1} G * Gs \end{array}$$

Therefore the judgment relies on the inductive definition of a predicate for order-preserving split of a context. This corresponds to the usual logic programming implementation of `append` with the first argument being in input mode.

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

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

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

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

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

Figure 3: Sequent Rules

The rest of the sequent rules are as expected and left to the on-line documentation. Note that the very fact that provability is inductive makes available *inversion principles* as elimination rules of the above definitions. 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.

MC-Theorem 2 (Structural Rules). *The following rules are admissible:*

- *Weakening for numerical bounds:*
  1. $[\![\, \Gamma;;\Omega \rhd_n G;\ n < m \,]\!] \Longrightarrow \Gamma;;\Omega \rhd_m G$
  2. $[\![\, \Gamma;;\Omega \rhd_n \mathit{Goals};\ n < m \,]\!] \Longrightarrow \Gamma;;\Omega \rhd_m \mathit{Goals}$
  3. $[\![\, \Gamma \rhd_n \mathit{Goals};\ n < m \,]\!] \Longrightarrow \Gamma \rhd_m \mathit{Goals}.$
- *Context weakening:*
  1. $[\![\, \Gamma;;\Omega \rhd G;\ \Gamma \subseteq \Gamma' \,]\!] \Longrightarrow \Gamma';;\Omega \rhd G$
  2. $[\![\, \Gamma;;\Omega \rhd \mathit{Goals};\ \Gamma \subseteq \Gamma' \,]\!] \Longrightarrow \Gamma';;\Omega \rhd \mathit{Goals}$
  3. $[\![\, \Gamma \rhd \mathit{Goals};\ \Gamma \subseteq \Gamma' \,]\!] \Longrightarrow \Gamma' \rhd \mathit{Goals}.$

Proof. The proof is by mutual structural induction on the three sequents judgments and is achieved simply by a call to Isabelle HOL's classic reasoner. □

The sequent calculus in [16] enjoys various forms of cut-elimination. For the sake of the type preservation proof (MC-Theorem 3) we only need the following atomic intuitionistic cut:

$$[\![\, A * \Gamma;;[\,] \rhd G;\ \rhd \langle A \rangle \,]\!] \implies \Gamma;;[\,] \rhd G$$

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 (ignoring the intuitionistic context):

$$K \diamond i \qquad \leadsto \qquad \ulcorner K \urcorner \rhd \ulcorner i \urcorner$$

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

Given the goal: $\mathsf{init}\ V \twoheadrightarrow \mathsf{ex}\ (\mathsf{ev}\ e)$ 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 $\mathsf{ex}\ (\texttt{return}\ i)$ is intended to mean: execute instruction i. A goal $\mathsf{ex}\ (\texttt{return}\ v)$ 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 informal representations:

$$\mathbf{init} \diamond \mathbf{return}\, v \quad \leadsto \quad \mathsf{init}\ W \rhd \mathsf{ex}\ (\texttt{return}\ \ulcorner v \urcorner)$$

where the logic variable W is the final answer;

$$K; \lambda x.\, i \diamond \mathbf{return}\, v \leadsto \quad \ulcorner K \urcorner, \mathsf{cont}\ (\lambda x. \ulcorner i \urcorner) \rhd \mathsf{ex}\ (\texttt{return}\ \ulcorner v \urcorner)$$

where the ordering constraints force the proof of $\mathsf{ex}\ \texttt{return}\ \ulcorner v \urcorner$ to focus on the rightmost ordered formula. The faithfulness of our representation of evaluation could be formally stated and proved in an adequacy theorem, analogously to the Theorem 3.4 in [5].

As usual in the two-levels approach [12] we introduce a datatype `atm` to encode the atomic formulae of the OL, which in this case study consists of:

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

We can now give the clauses for the OL deductive systems in Figure 4, starting with typing. These 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 Figure 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.

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: Note the presence of the *abstraction* annotations as Isabelle HOL premises in rules mentioning second-order terms. 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, 12].

$$
\begin{array}{rcl}
\mathit{Inductive}\ \_ \longleftarrow \_;;\_ & :: & [\,\mathit{atm}, \mathit{oo\ list}, \mathit{oo\ list}\,] \Rightarrow \mathit{bool} \\
 & \Longrightarrow & \mathsf{of}\ (E_1\ \texttt{@}\ E_2)\ T \longleftarrow [\,];;[\langle \mathsf{of}\ E_1\ (T'\ \texttt{arr}\ T)\rangle, \langle \mathsf{of}\ E_2\ T'\rangle] \\
[\![\ \mathsf{abstr}\ E\ ]\!] & \Longrightarrow & \mathsf{of}\ (\texttt{lam}\ E)\ (T_1\ \texttt{arr}\ 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}\ (\texttt{lam}*\ E)\ T \longleftarrow [\,];;[\mathsf{of}\ (\texttt{lam}\ E)\ T] \\
 & \Longrightarrow & \mathsf{ofI}\ (\texttt{ev}\ E)\ T \longleftarrow [\,];;[\langle \mathsf{of}\ E\ T\rangle] \\
 & \Longrightarrow & \mathsf{ofI}\ (\texttt{return}\ V)\ T \longleftarrow [\,];;[\langle \mathsf{ofV}\ V\ T\rangle] \\
 & \Longrightarrow & \mathsf{ofI}\ (\mathsf{app}_1\ V\ E)\ T \longleftarrow [\,];;[\langle \mathsf{ofV}\ V\ (T_2\ \texttt{arr}\ T)\rangle, \langle \mathsf{of}\ E\ T_2\rangle] \\
 & \Longrightarrow & \mathsf{ofK}\ (T\ \texttt{arr}\ T) \longleftarrow [\langle \mathsf{init}\ V\rangle];;[\,] \\
[\![\ \mathsf{abstr}\ K\ ]\!] & \Longrightarrow & \mathsf{ofK}\ (T_1\ \texttt{arr}\ T_2) \longleftarrow [\langle \mathsf{cont}\ K\rangle, \langle \mathsf{ofK}\ T\ \texttt{arr}\ T_2\rangle];;[\mathsf{all}\, v.\, (\mathsf{ofV}\ v\ T_1) \rightarrow \langle \mathsf{ofI}\ (K\ v)\ T\rangle] \\
 & \Longrightarrow & \mathsf{ceval}\ E\ V \longleftarrow [\mathsf{init}\ V \twoheadrightarrow \mathsf{ex}\ (\mathsf{ev}\ E)];;[\,] \\
 & \Longrightarrow & \mathsf{ex}\ (\texttt{return}\ V) \longleftarrow [\langle \mathsf{init}\ V\rangle];;[\,] \\
[\![\ \mathsf{abstr}\ E\ ]\!] & \Longrightarrow & \mathsf{ex}\ (\texttt{return}\ V) \longleftarrow [\langle \mathsf{cont}\ E\rangle, \langle \mathsf{ex}\ (E\ V)\rangle];;[\,] \\
[\![\ \mathsf{abstr}\ E\ ]\!] & \Longrightarrow & \mathsf{ex}\ (\texttt{ev}\ (\texttt{lam}\ E)) \longleftarrow [\langle \mathsf{ex}\ (\texttt{return}\ (\texttt{lam}*\ E))\rangle];;[\,] \\
 & \Longrightarrow & \mathsf{ex}\ (\texttt{ev}\ (E_1\ \texttt{@}\ E_2)) \longleftarrow [\mathsf{cont}\ (\lambda v.\ \mathsf{app}_1\ v\ E_2) \twoheadrightarrow \langle \mathsf{ex}\ (\mathsf{ev}\ E_1)\rangle];;[\,] \\
[\![\ \mathsf{abstr}\ E\ ]\!] & \Longrightarrow & \mathsf{ex}\ (\mathsf{app}_1\ (\texttt{lam}*\ E)\ E_2) \longleftarrow [\langle \mathsf{ex}\ (\mathsf{ev}\ (E\ E_2))\rangle];;[\,]
\end{array}
$$

Figure 4: Hybrid's Encoding of Program Clauses

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

MC-Theorem 3.

$$
\begin{array}{l}
[\,];;\mathsf{init}\ V, \Omega\ \ \rhd_i \langle \mathsf{ex}\ I\rangle \Longrightarrow \\
\quad \forall T_1 T_2.\, \rhd\ \langle \mathsf{ofI}\ I\ T_1\rangle \supset \\
\quad ([\,];;\mathsf{init}\ V, \Omega \rhd\ \langle \mathsf{ofK}\ (T_1\ \texttt{arr}\ T_2)\rangle) \supset (\ \rhd\ \langle \mathsf{ofV}\ V\ T_2\rangle)
\end{array}
$$

Proof. The proof is by complete induction on the height i of the derivation of the premise and one case is detailed in Figure 5. □

As far as proof search is concerned, once we have instructed the system to aggressively apply every deterministic splitting, the proof script for the above case consists of three instructions, including giving the correct instantiation of the height of the proof, in order to fire the IH. Other cases, typically the ones which increase the current continuation need more care. In particular, the user is required to provide the correct splitting of the ordered hypothesis.

Corollary 1 (Subject Reduction).

$$[\![\ \rhd \mathsf{ceval}\ E\ V;\ \rhd \mathsf{of}\ E\ T\ ]\!] \ \Longrightarrow\ (\rhd \mathsf{ofV}\ V\ T)$$

5. RELATED WORK

We refer to [2] for a review of related work about HOAS. The present paper generalizes the approach in [12], which in turn was inspired by [8, 10]. The latter in particular presents a two-levels 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 [5], 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 currently does not extend to LLF [21]) verifies that the given relation is indeed a realizer for that theorem. 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 that 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. Only very recently, $\mathcal{M}_\omega$ [20], the meta-logic of LF has been extended to $\mathcal{L}^+_\omega$, a meta-logic for LLF [9].

6. CONCLUSIONS

We have presented a two-levels approach to formalize an ordered logical framework on top of the Hybrid system that allows inductive reasoning about objects defined via HOAS in an established environment such as Isabelle HOL. This replicates, in a well-understood and interactive setting, the architecture of $FO\lambda^{\Delta IN}$ [10], so all results are proved without ad-hoc lemmas that are not warranted by the mathematics of the problem. The specification logic is implemented and its meta-theoretical properties are proved 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 toward investigating the meta-theory of languages with imperative features. The next objective would be to replay the cited proof of type preservation for MLR, which will be simplified by the internalization of the instructions stack. From this viewpoint, it is somewhat disappointing that we still have to retain a notion of *typing* of continuations $\mathsf{ofK}\ T$. An intriguing possibility is to move to the third-order machine described in [16], which, by using left implication $\hookleftarrow$, does not need the level of instructions. For example evaluating an application would be in Olli syntax:

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 >`
$[\,];;\Omega \rhd_i Goals$	corresponds to	`[] ;; Ome ||> Goals ::: i`

The inductive hypothesis (which will be omitted next) is

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

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

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

First we apply the lemma that `T1` must be of functional form `T1' arr 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 the `||>` statement:

```
[| ...[] ;;  [] |-- (< ofI (ev (lam E)) (T1' arr T2') >) ;
      [] ;;  init V * Ome |-- (< ofK ((T1' arr T2') arr T2) >) ;
      osplit (init V * Ome) Og Or; [] ;; Or ||> [] ::: i';
      [] ;; Og |- (< ex (return (lam* E)) >) ::: i' |]
                      ==> [] ;; [] |-- < ofV V T2 >
```

However, by inversion on the `||>`, it must be that `Or = []`, hence splitting is deterministic returning `Og = init V * Ome`:

```
[| ...[] ;;  [] |-- (< ofI (ev (lam E)) (T1' arr T2') >) ;
      [] ;;  init V * Ome |-- (< ofK ((T1' arr T2') arr T2) >) ;
      [] ;;  init V * Ome |- (< ex (return (lam* E)) >) ::: i' |]
                      ==>  [] ;; [] |-- < ofV V T2 >
```

That follows by IH and the easy inversion lemma:

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

Figure 5: The Lambda Case in the Proof of Subject Reduction

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

where now

$$\begin{array}{rcl} \texttt{app}_1 & :: & [\,val, exp\,] \Rightarrow bool \\ \texttt{ev} & :: & exp \Rightarrow bool \\ \texttt{return} & :: & val \Rightarrow bool \end{array}$$

that is they are predicates. This would entail some changes in the SL, namely generalizing the structure of implicational clauses and of backchaining. It may be argued that there seems to be a natural progression from intuitionistic second-order logic [12] to second and finally third order ordered linear logic; whereby we simplify the machine, first internalizing explicit continuations in the CPM [14] then removing instructions by making use of a 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 current project is the verification of compilation based on monadic intermediate languages, such as MIL-lite [4]. Further there are several other applications beyond programming languages for an ordered framework, for example GSOS with priorities [22]. The latter may require a more sophisticated notion of order, typically branching. It is conceivable that this could be mirrored by refining linear ordered context in the sense of BI's *bunches* along the lines of [19, 1].

As far as the infrastructure is concerned, note that similarly to [12] in this case study we only needed to induct over *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 [11]. The new theory of *terms-in-infinite-context* underlying the new version of Hybrid [3] 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 [13] and finally approach at the right level of abstraction the verification of the compiler optimizations of MIL-lite [4].

Source files for the Isabelle HOL code can be found at `www.mcs.le.ac.uk/~amomigliano/isabelle/2Levels/Oll2`

7. ACKNOWLEDGMENTS

Alberto Momigliano was supported by EPSRC grant GR/M98555 and partly by the MRG project (IST-2001-33149), funded by the EC under the FET proactive initiative on Global Computing. Jeff Polakow was supported by grant IST-2001-33477.

8. REFERENCES

[1] A. Ahmed and D. Walker. The logical approach to stack typing. *ACM SIGPLAN Notices*, 38(3):74–85, Mar. 2003.

[2] S. Ambler, R. Crole, and A. Momigliano. Combining higher order abstract syntax with tactical theorem proving and (co)induction. In V. A. Carreño, editor, *Proceedings of the 15th International Conference on Theorem Proving in Higher Order Logics, Hampton, VA, 1-3 August 2002*, volume 2342 of *LNCS*, pages 13-30. Springer Verlag, 2002.

[3] S. Ambler, R. Crole, and A. Momigliano. A definitional approach to primitive recursion over higher order abstract syntax. In *Proceedings of MERLIN'03, Second ACM SIGPLAN Workshop on MEchanized Reasoning about Languages with varIable biNding*, pages 1–18, Uppsala, Sweden, 2003.

[4] N. Benton and A. Kennedy. Monads, effects and transformations. *Electronic Notes in Theoretical Computer Science*, 26, 1999.

[5] I. Cervesato and F. Pfenning. A linear logical framework. *Information & Computation*, 179(1):19–75, Nov. 2002.

[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-levels 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*, pages 198-213. Springer Verlag, 2002.

[9] A. McCreight and C. Schürmann. A meta linear logical framework. Draft, 2003.

[10] 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.

[11] D. Miller and A. Tiu. A proof theory for generic judgments: an extended abstract. In P. Kolaitis editor, *LICS* 2003, Ottawa, July 2003, pp. 118 - 127.

[12] 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.

[13] A. Momigliano, S. Ambler, and R. Crole. A Hybrid encoding of Howe's method for establishing congruence of bisimilarity. *ENTCS*, 70(2), 2002.

[14] F. Pfenning. Computation and deduction. Lecture notes, 277 pp. Revised 1994, 1996, to be published by Cambridge University Press, 1992.

[15] F. Pfenning. Logical frameworks. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*. Elsevier Science Publishers, 1999.

[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.

[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 and F. Pfenning. A coverage checking algorithm for LF. 2003. TPHOLs, Roma, Italy, September 2003.

[22] I. Ulidowski and I. Phillips. Ordered SOS process languages for branching and eager bisimulations. *Information and Computation*, 178, 2002.