A formalization of an Ordered Logical Framework in Hybrid with applications to continuation machines

Alberto Momigliano
Laboratory for the Foundations of Computer Science
University of Edinburgh

Joint work with Jeff Polakow

MERλIN 2003, 26/08/2003

Overview of the Talk

- Towards reasoning HOAS-style in presence of imperative features.
- From LF to LLF and finally OLF.
- Implementing OLF with the 2-level approach in Hybrid.
- Case study: subject reduction of a continuation machine.

(Original) Motivation

- EPSRC grant: formal validation of the effect-based optimizations performed by the MLj compiler for SML into Java bytecode [Benton & Kennedy '98,'99], as (contextual) equivalence preserving transformations. Compilation into a typed intermediate language with effects.

- Need to devise techniques which are truly **scalable** to such languages. This is a function of how *abstract* the encoding can be.

- The bar has been set by Twelf, providing full HOAS and paramount use of parametric-hypo judgments: object-level environments as meta-level contexts.

- Cannot be stressed enough: no need for partial function, tables and lookup, and the relative proof obligations (weakening, substitution lemmata).

- LF being intuitionistic not suited to model **imperative** features: if the *store* is encoded as a context, you can lookup, but no elegant way to do **updates**.

- **Linear Logic** to the rescue: mutable data are modeled via *linear* assumptions, so that updates are realized by a lookup (which consumes) and a hypothetical judgment asserting the new value for the location.

- A **store** $S \equiv \{c_1 = v_1 \ldots c_n = v_n\}$ is represented as the context $c_1 : \mathit{cell} \ldots c_1 : \mathit{cell}; h_1 : \mathit{contains}\ c_1\ \ulcorner v_1 \urcorner \ldots h_n : \mathit{contains}\ c_n\ \ulcorner v_n \urcorner$.

- Traditional encodings are possible but tend to hijack the development ($1/4$ of the proof of subject red. for MLR in the Coq library is entirely dedicated to the implementation of `tables`)

- This applies to every other imperative feature: for example the operational semantics judgments of *Grail* [Stark et al'03] (TIL for Camelot) is $E \vdash \langle h, e\rangle \Downarrow \langle h', v, r\rangle$, where E maps names to values and the "mini-heap" h is a triple of maps with co-domain classes, integer and reference values.

- The catch: a linear logic encoding requires the operational semantics to be in *continuation-passing style* ($*-$ is backwards linear implication, $\{\cdot\}$ universal quantification). Take creating a reference:

```
ev_rf : eval (ref E) (rc ???)  % cell escapes scope
          *- eval E V
          *- ({c} contains c V  -* ???) % what's next?
```

- CPS sequentializes evaluation steps so that there is always a next instruction.

Linear Frameworks

```
keval: cont -> instr -> val -> type.
kev_rf : keval K (ev (ref E)) Answer
            *- keval (K ; [x] ref* x) (ev E) Answer
                % add the instr. to the cont.
ev_idfr : keval K (ref* V) (new ([c] A c))
                % bind the cell in the final answer
           *- ({c} contains C V -* keval K (return (rc c)) (A c)).
```

- This can be done in LLF [Cervesato & Pfenning'96], but the price to pay is the introduction of the level of *instructions* and of *continuations/machine states*. Those needs to be typed as well and subject reduction for MLR becomes, albeit very neat, more 'populated'.

- Moreover, LLF at the moment is only a type-checker. Meta-theorem can be expressed as relations, but there is no *coverage* checking or meta-theorem prover (yet) (see Schürmann et al's draft "A Meta Linear Logical Framework).

OLF's Basics

- However, we can be push the idea of *conservative* extension of LF further and internalize the very idea of machine state and continuation.

- Ordered Logical Framework [Polakow'00]: reasoning with unrestricted (or intuitionistic), linear and ordered hypotheses.

- Unrestricted ones may be used arbitrarily, no order. Linear ones must be used exactly once, no order. Ordered hypotheses must be used exactly once subject to the order in which they were assumed. For example:

- Typing an expression is intuitionistic.

- Accessing and updating the store is linear.

- Ordered context represents the current stack of instructions.

Prototyping OLF

- Implementing OLF from scratch is a major project, requiring:
- Devising ordered dependent unification and type reconstruction.
- Practical resource management for Olli (the logical programming language). This is particularly complicated in the presence of $\top$ [Polakow'01].
- Not to mention coverage checking or meta-theorem proving.
- Alternative: using the 2-level approach [McDowell & Miller'01] embedded in a inductive proof assistant [Felty'02, Momigliano & Ambler'03].

2-Levels: the Idea

- Prolog 101: the vanilla meta-interpreter:

```
solve(true)
solve(A;B) :- solve(B); solve(A)
solve(A,B) :- solve(B),solve(A)
solve(A)   :- atomic(A),prog(A,Body),solve(Body).

prog(parent(P,S), (mother(P,S) ; father(P,S)).
prog(father(abraham, isaac),true).
```

- Note that `prog` clauses do not have to be Prolog-like (i.e., inductive):

```
solve(A => B) :- assert(A),!,solve(B),retract(A).
         % bad style, but you know what I mean
prog(nd(imp(A,B)),nd(A) => nd(B)).
```

2-Levels in Hybrid

- The interpreter will gladly perform (rightmost selection) evaluation, but although it can be looked at as an inductive definition, it does not know that and can't do meta-reasoning (e.g. case analysis).

- So let's take `solve` as an inductive definition in Isabelle HOL sequent-style, say. This gives structural induction and case analysis. Same for `prog`.

- Not quite enough: we need to supprt HOAS at the syntax level, so a shallow embedding into an inductive framework is out of the question, for the usual monotonicity reasons.

- Answer: implement your own λ-calculus on top of your proof assistant, consistent with the inductive side of the system, and use it as your basic metalanguage. For example, Hybrid [Ambler,Crole & Momigliano'02].

Hybrid in one slide

- Deep encoding in Isabelle/HOL of a λ-calculus as an HOAS meta-language.
- Supports *tactical theorem proving*, *induction and co-induction*, but is *definitional*, so consistent *within a classical type theory*.
- 'Hybrid syntax' gives the user HOAS, but this is a **definition** for an underlying de Bruijn representation. This gives access to all the features of HOL.
- It features a predicate $proper :: expr \Rightarrow bool$, that models the terms one-one with the untyped λ-calculus modulo α and a predicate $abstr :: (expr \Rightarrow expr) \Rightarrow bool$ to rule out exotic functional term.

- Sequents $\Gamma;\Omega \longrightarrow_\Pi G$, for Π the (unrestricted) program clauses Γ unrestricted atoms; Ω ordered atoms; and G is the formula to be derived.

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

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

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

- Encoding of provability specialized to right ordered implication, via three mutually inductive definitions:

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

- A predicate for order-preserving split of a context $(osplit\ \Omega\ \Omega_L\ \Omega_R)$ (the usual logic programming append in the other mode).

- Program clauses are $\forall(A \longleftarrow [G_1, \ldots, G_m] ;; [G'_1, \ldots, G'_n])$, i.e. atoms under two lists of intuitionistic and ordered assumptions: this subsumes multiplicative and additive conjunction.

- Backchain rule: fetch the program clause, prove intuitionistic assumptions, consume ordered ones (in the right order):

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

- Ordered list consumption (there is an analogours intuitionistic additive version): split Ω in Ω_R and Ω_G, prove G form the latter and continue.

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

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

$$\begin{array}{rcl}
\mathbf{st_init} :: & \mathbf{init} \diamond \mathbf{return}\, v \hookrightarrow \mathbf{answer}\, v \\
\mathbf{st_return} :: & K; \lambda x.\, i \diamond \mathbf{return}\, v \hookrightarrow K \diamond i[v/x] \\
\mathbf{st_lam} :: & K \diamond \mathbf{ev}\,(\mathbf{lam}\, x.e) \hookrightarrow K \diamond \mathbf{return}\,(\mathbf{lam}^*\, x.e) \\
\mathbf{st_app} :: & K \diamond \mathbf{ev}\,(e_1\, e_2) \hookrightarrow K; \lambda x_1.\, \mathbf{app}_1\, x_1\, e_2 \diamond \mathbf{ev}\, e_1 \\
\mathbf{st_app1} :: & K \diamond \mathbf{app}_1\,(\mathbf{lam}^*\, x.e)\, e_2 \hookrightarrow K \diamond \mathbf{ev}\, e[e_2/x]
\end{array}$$

- No explicit stack-like structure for K, rather we store instructions in the ordered context:

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

- The goal $\mathsf{init}\ W \twoheadrightarrow \mathsf{ex}\ (\mathtt{ev}\ e)$ evaluates e and instantiate W with the resulting value: evaluate e with the initial continuation:

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

- $\mathsf{ex}\ (\mathtt{return}\ i)$ means execute i, $\mathsf{ex}\ (\mathtt{return}\ v)$ means passing v to the top continuation on the stack:

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

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

$$
\begin{array}{rcl}
 & \Longrightarrow & \mathsf{ex}\ (\mathtt{return}\ V) \longleftarrow [\langle \mathsf{init}\ V \rangle] ;; [\,] \\
[\![\ \mathit{abstr}\ E\ ]\!] & \Longrightarrow & \mathsf{ex}\ (\mathtt{return}\ V) \longleftarrow [\langle \mathsf{cont}\ E \rangle, \langle \mathsf{ex}\ (E\ V) \rangle] ;; [\,] \\
[\![\ \mathit{abstr}\ E\ ]\!] & \Longrightarrow & \mathsf{ex}\ (\mathsf{ev}\ (\mathtt{lam}\ E)) \longleftarrow [\langle \mathsf{ex}\ (\mathtt{return}\ (\mathtt{lam}{*}\ E)) \rangle] ;; [\,] \\
 & \Longrightarrow & \mathsf{ex}\ (\mathsf{ev}\ (E_1 \mathbin{@} E_2)) \longleftarrow [\mathsf{cont}\ (\lambda v.\ \mathsf{app}_1\ v\ E_2) \twoheadrightarrow \langle \mathsf{ex}\ (\mathsf{ev}\ E_1) \rangle] ;; [\,] \\
[\![\ \mathit{abstr}\ E\ ]\!] & \Longrightarrow & \mathsf{ex}\ (\mathsf{app}_1\ (\mathtt{lam}{*}\ E)\ E_2) \longleftarrow [\langle \mathsf{ex}\ (\mathsf{ev}\ (E\ E_2)) \rangle] ;; [\,] \\
 & \cdots & \\
 & \Longrightarrow & \mathsf{ofK}\ (T\ \mathsf{arrow}\ T) \longleftarrow [\langle \mathsf{init}\ V \rangle] ;; [\,] \\
[\![\ \mathit{abstr}\ K\ ]\!] & \Longrightarrow & \mathsf{ofK}\ (T_1\ \mathsf{arrow}\ T_2) \longleftarrow [\langle \mathsf{cont}\ K \rangle, \langle \mathsf{ofK}\ T\ \mathsf{arrow}\ T_2 \rangle] ;; \\
 & & \quad [\mathit{all}\ v.\ (\mathsf{ofV}\ v\ T_1) \rightarrow \langle \mathsf{ofI}\ (K\ v)\ T \rangle]
\end{array}
$$

- No explicit continuation being typed, only an ordered assumption. First get a continuation from the ordered context and then type it.

- Evaluation a transcription of the informal rules: Subject reduction follows from:

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

- Script for subject reduction is ca. 150 lines, yet based on some infrastructure: meta-theory of SL (weakening, cut-elim), *filling* steps and

- **Resource Management**: not implemented as a IO system [Hodas & Miller'94], but *tactics-driven*:

- Deterministic splitting automatically applied and propagated in inversion rules

$$\Gamma ;; \Omega \triangleright_n [\langle \mathsf{cont}\ K \rangle] \implies \exists K'.\, \Omega = [\mathsf{cont}\ K']$$
$$\Gamma ;; \Omega \triangleright_n [\,] \implies \Omega = [\,]$$

- As intros, it is user-driven – feasible as in verification (as opposed as logic prog.) stacks are very small.

- We may look at *proof planning* to automate the splitting process more, especially to rule out impossible cases.

- Real benefits will kick in as soon as we start to add imperative features.

- No free lunch: the complexity of the encoding of environments as (finite) map is shifted to the issue of proof search in a sub-structural constructive logics (the SL).

- However, meta-theoretical properties of the latter can be re-used for other object level languages.

Related and Future Work

- Previous case study with OLF such as [Polakow & Pfenning'00] (CPS translation obeying stack-like properties wrt intermediate values) only done with paper and pencil. Related/future work includes looking at relations with:

- Bunched logics [Pym'03], stack logics [Ahmed & Walker'03], *destination-passing style* in the Concurrent Logical Framework [Watkins el al.'03].

- Applications: op. semantics of *typed intermediate languages* (MIL-lite, Grail), object calculi and their programming logics [Abadi & Leno'98], security of stack access properties on JVM.

- GSOS with priorities, string matching.