Case Study: Subject Reduction for Mini-ML with References, in Isabelle/HOL + Hybrid

Alan J. Martin

Department of Mathematics and Statistics,
University of Ottawa

Workshop on Mechanizing Metatheory
September 20, 2008

Outline

uOttawa

Outline

Languages with Binding Operators

Higher-order abstract syntax (HOAS) is a representation technique for languages with variable-binding operators, such as lambda-abstraction, logical quantifiers, etc.

A traditional inductive definition uses explicit variables for binding operators, taken from a countable set *var*:

Example (untyped λ-calculus)

$$\mathit{lterm} ::= \mathit{var} \mid \mathit{lterm}\ \$\ \mathit{lterm} \mid \mathbf{LAM}\ \mathit{var}.\ \mathit{lterm}$$

Reasoning about these variables (α-equivalence, capture-avoiding substitution, etc.) complicates proofs of properties of such languages.

Higher-Order Encodings

The HOAS alternative is to represent the variable-binding operators using functional-type arguments:

Example

$\mathit{lterm} ::= \mathit{lterm}\ \$\ \mathit{lterm} \mid \mathbf{LAM}\ (\mathit{lterm} \Rightarrow \mathit{lterm})$

Now the variables of the object language are represented by bound variables of the meta-language: **LAM** ‘x’. ‘x’ \$ ‘x’ becomes **LAM** $(\lambda\, x.\, x\ \$\ x)$, where λ is meta lambda-abstraction.

Higher-Order Encodings, cont.

Example

$$\mathit{lterm} ::= \mathit{lterm}\ \$\ \mathit{lterm} \mid \mathbf{LAM}\ (\mathit{lterm} \Rightarrow \mathit{lterm})$$

This is not an ordinary inductive definition, but it should have similar properties, notably freeness of the constructors. That consists of injectivity of each constructor:

$$\forall\, s\, s'\, t\, t'.\, (s\ \$\ t = s'\ \$\ t') \Longrightarrow (s = s') \wedge (t = t')$$
$$\forall\, f\, f'.\, (\mathbf{LAM}\ f = \mathbf{LAM}\ f') \Longrightarrow (f = f')$$

together with distinctness (or ‘image-disjointness’):

$$\forall\, x\, y\, f.\, (x\ \$\ y) \neq \mathbf{LAM}\ f$$

HOAS Advantages and Disadvantages

Advantages:

- α-equivalence becomes equality
- Capture-avoiding substitution becomes function application
- Structural properties may have very simple proofs

Disadvantages:

- Not all meta-reasoning can be done in the target language
- Unlike first-order encodings, adequacy is not immediate

Outline

uOttawa

Isabelle/HOL, HOAS, and Hybrid

Isabelle/HOL is an interactive proof assistant using classical higher-order logic. As such, its function space is not directly usable for HOAS, because its logic primitives allow the formation of terms at function types that do not behave like syntax.

Hybrid [Momigliano, Martin, & Felty 2008] is a package for Isabelle/ HOL that provides a type *expr* suitable for higher-order encodings. This type is built definitionally using de Bruijn indices, and it solves the problem of the function space by carving out a suitable part of it with a predicate **abstr**.

Isabelle/HOL supports two different kinds of proof: tactic-style and Isar. The formal proofs for this case study are done almost entirely in Isar form.

The Term Language of Hybrid

Definition

$expr ::= \textbf{CON}\ con \mid \textbf{VAR}\ var \mid expr\ \$\ expr \mid \textbf{LAM}\ (expr \Rightarrow expr)$

Hybrid's type *expr* is based on the untyped lambda-calculus, but with extensions designed to support higher-order encodings of other object languages (OLs): **CON** introduces OL-specific constants, and **VAR** can be used for OL free variables.

The Term Language of Hybrid

Definition

$expr ::= \textbf{CON}\ con \mid \textbf{VAR}\ var \mid expr\ \$\ expr \mid \textbf{LAM}\ (expr \Rightarrow expr)$

Hybrid's type *expr* is based on the untyped lambda-calculus, but with extensions designed to support higher-order encodings of other object languages (OLs): **CON** introduces OL-specific constants, and **VAR** can be used for OL free variables.

Example

OL (mini-ML): $\textbf{fn}\ x\ y \rightarrow x$
Hybrid encoding: $(\textbf{CON}\ \mathrm{cFN})\ \$\ \big(\textbf{LAM}\ x.\ (\textbf{CON}\ \mathrm{cFN})\ \$\ (\textbf{LAM}\ y.\ x)\big)$
With concrete syntax: $\textbf{fn}\ x\ y.\ x$

Outline

uOttawa

Syntax

Expressions

$\mathit{exp} ::= \mathbf{0} \mid \mathbf{Suc}\ \mathit{exp} \mid \mathbf{Pred}\ \mathit{exp}$	*natural numbers*
$\mid \mathbf{tt} \mid \mathbf{ff} \mid \mathbf{IsZero}\ \mathit{exp} \mid \mathbf{if}\ \mathit{exp}\ \mathbf{then}\ \mathit{exp}\ \mathbf{else}\ \mathit{exp}$	*booleans*
$\mid * \mid \langle \mathit{exp}, \mathit{exp} \rangle \mid \pi_1\ \mathit{exp} \mid \pi_2\ \mathit{exp}$	*unit and pairs*
$\mid \mathbf{fn}\ \mathit{var}.\ \mathit{exp} \mid \mathit{var} \mid \mathit{exp}\ \mathit{exp} \mid \mathbf{rec}\ \mathit{exp}$	*functions & recursion*
$\mid \mathbf{let} / \mathbf{letv} / \mathbf{letrec}\ \mathit{var} = \mathit{exp}\ \mathbf{in}\ \mathit{exp}$	*let-binding*
$\mid \mathbf{ref}\ \mathit{exp} \mid !\ \mathit{exp} \mid \mathit{exp} := \mathit{exp} \mid \mathbf{cell}\ \mathit{addr}$	*mutable references*

This object language has been around for a long time; I followed the particular formulation of syntax and semantics from [Cervesato & Pfenning, 1996], with only slight changes.

Syntax

Expressions

$exp ::= \mathbf{0} \mid \mathbf{Suc}\ exp \mid \mathbf{Pred}\ exp$	*natural numbers*
$\mid \mathbf{tt} \mid \mathbf{ff} \mid \mathbf{IsZero}\ exp \mid \mathbf{if}\ exp\ \mathbf{then}\ exp\ \mathbf{else}\ exp$	*booleans*
$\mid * \mid \langle exp, exp \rangle \mid \pi_1\ exp \mid \pi_2\ exp$	*unit and pairs*
$\mid \mathbf{fn}\ var.\ exp \mid var \mid exp\ exp \mid \mathbf{rec}\ exp$	*functions & recursion*
$\mid \mathbf{let} / \mathbf{letv} / \mathbf{letrec}\ var = exp\ \mathbf{in}\ exp$	*let-binding*
$\mid \mathbf{ref}\ exp \mid !\ exp \mid exp := exp \mid \mathbf{cell}\ addr$	*mutable references*

Almost all of the interesting issues arose in the cases for functions and references.

Syntax

Expressions

$exp ::= \mathbf{0} \mid \mathbf{Suc}\ exp \mid \mathbf{Pred}\ exp$	*natural numbers*
$\mid \mathbf{tt} \mid \mathbf{ff} \mid \mathbf{IsZero}\ exp \mid \mathbf{if}\ exp\ \mathbf{then}\ exp\ \mathbf{else}\ exp$	*booleans*
$\mid * \mid \langle exp, exp \rangle \mid \pi_1\ exp \mid \pi_2\ exp$	*unit and pairs*
$\mid \mathbf{fn}\ var.\ exp \mid var \mid exp\ exp \mid \mathbf{rec}\ exp$	*functions & recursion*
$\mid \mathbf{let} / \mathbf{letv} / \mathbf{letrec}\ var = exp\ \mathbf{in}\ exp$	*let-binding*
$\mid \mathbf{ref}\ exp \mid !\ exp \mid exp := exp \mid \mathbf{cell}\ addr$	*mutable references*

Operational Semantics

To give an operational semantics for a language with mutable references, some auxiliary syntactic classes are required:

- An *addr* is an abstract memory address.
  Its syntax is unimportant, and there are infinitely many addresses.
- A *state* is a list of pairs (c, v) where c :: *addr* and v :: *exp*.
  Each abstract memory cell can hold an arbitrary value.

A natural (a.k.a. big-step) operational semantics may then be defined as a judgment

$$(S_i, e) \hookrightarrow (S_f, v),$$

where S_i, S_f :: *state* and e, v :: *exp*.

Operational Semantics, continued

The formal proofs are based on the continuation-style operational semantics from [Cervesato & Pfenning, 1996], rather than the more straightforward natural semantics. This involves more syntactic classes:

- An instruction (*insn*) has the form "$\lambda var.\, exp$".
- A continuation (*cont*) is a list of instructions.
- An answer (*final*) is a (*state*, *exp*) pair, prefixed by an optional list of binding operators of the form "**new** *addr*."

The evaluation judgment then takes the more complicated form

$$S \triangleright K \vdash e \hookrightarrow w$$

where S :: *state* is the initial state, K :: *cont* is a continuation, e :: *exp* is the expression to evaluate, and w :: *final* is the result.

Typing

Types:

$$tp ::= \mathrm{nat} \mid \mathrm{bool} \mid \mathrm{unit} \mid tp \times tp \mid tp \to tp \mid tp\ \mathrm{ref}$$

Four separate typing judgments are needed: for expressions, continuations, states, and answers. Additionally, contexts Γ (listing types for variables) and store contexts Δ (listing types for addresses) are needed:

$$\begin{aligned} \Gamma, \Delta &\vdash_e exp : tp \\ \Gamma, \Delta &\vdash_k cont : tp \Rightarrow tp \\ \Delta &\vdash_s state : \Delta' \\ \Delta &\vdash_f final : tp \end{aligned}$$

Theorem (Subject Reduction)

If $S \triangleright \cdot \vdash e \hookrightarrow w$, $\Delta \vdash_s S : \Delta$, and $\cdot, \Delta \vdash_e e : \tau$, then $\cdot, \Delta \vdash_f w : \tau$.

Outline

uOttawa

Related Work

Previous work in Hybrid

In [Momigliano & Polakow, 2003], subject reduction for a purely functional version of Mini-ML was done in an earlier version of Hybrid.

Previous formalizations of Mini-ML with references

In [Cervesato & Pfenning, 1996], subject reduction for Mini-ML with references was formalized in a linear version of LF.

This example has been done in other systems too.

Outline

uOttawa

Syntax

Expressions of Mini-ML are encoded as Hybrid terms, using higher-order abstract syntax for bound variables:

$\mathit{exp} = \mathit{con}\ \mathit{expr}$	Hybrid expressions (untyped)
$\mathbf{fn} :: (\mathit{exp} \Rightarrow \mathit{exp}) \Rightarrow \mathit{exp}$	Higher-order abstract syntax

Example

$\mathbf{fn}\ x.\, \mathbf{Suc}\ x$ is encoded as $\mathbf{fn}\ (\lambda x.\, \mathbf{Suc}\ x) :: \mathit{exp}$

Addresses, states, continuations, and answers are similarly encoded.

Types, on the other hand, are encoded as an Isabelle/HOL datatype.

Judgments

Evaluation

eval, **cont** :: [*state*, *cont*, *exp*, *final*] $\Rightarrow$ *bool*

These predicates are defined using Isabelle/HOL inductive definitions. All four arguments are Hybrid terms.

The evaluation judgment is split into two predicates to keep track of whether the next step is to evaluate the expression (**eval**) or to execute an instruction from the continuation (**cont**).

Judgments, continued

The typing judgments follow the continuation-style presentation closely, with explicit contexts.

Typing

ofe $C\ D\ e\ t$	$C, D \vdash_e e : t$
ofk $C\ D\ k\ t\ r$	$C, D \vdash_k k : t \Rightarrow r$
ofs $D\ s\ D'$	$D \vdash_s s : D'$
off $D\ w\ t$	$D \vdash_f w : t$

where

$$C :: (exp \times tp)\ set; \qquad D, D' :: (addr \times tp)\ set$$

$$e :: exp; \qquad k :: cont; \qquad s :: state; \qquad w :: final; \qquad t, r :: tp$$

Subject reduction

Isabelle/HOL Theorem

$[\![\, \textbf{eval}\ s\ \textbf{kDone}\ e\ w;\ \textbf{ofs}\ D\ s\ D;\ \textbf{ofe}\ \{\}\ D\ e\ t\,]\!] \Longrightarrow \textbf{off}\ \{\}\ D\ w\ t$

This also corresponds directly to the continuation-style presentation.

Observations

- It was necessary to prove weakening and substitution properties for the typing context, similar to those of a logic.
- The proof was by induction on **eval**, with many cases proved automatically (except those involving functions and references).
- Auxiliary predicates were needed for reasoning about states (s) and store typings (D).

Outline

uOttawa

The Two-Level Approach

It is often useful to split up a HOAS encoding into two levels: first a specification logic (SL) is encoded in the meta-logic (Isabelle/HOL), then the object language (OL) is encoded in the SL.

Benefits of this approach:

- The 'object' part of the encoding is separated from the 'logic' part.
- By using different encoding styles for the two levels, it is possible to combine some of their advantages.

Specification Logic

The SL is a fragment of first-order minimal logic:

$$prp ::= \textbf{tt} \mid prp\ \textbf{and}\ prp \mid atm\ \textbf{imp}\ prp \mid \textbf{all}\ (expr \Rightarrow prp) \mid \langle atm \rangle$$

Its sequent rules are encoded as an inductive definition:

$$\begin{array}{l} \textit{inductive} \quad \_ \rhd \_ :: [\,atm\ list, prp\,] \Rightarrow bool \\ \textit{intros} \end{array}$$

$$\begin{array}{rcl} & \Longrightarrow & \Gamma \rhd \textbf{tt} \\ [\![\ \Gamma \rhd G_1;\ \Gamma \rhd G_2\ ]\!] & \Longrightarrow & \Gamma \rhd (G_1\ \textbf{and}\ G_2) \\ [\![\ (A, \Gamma) \rhd G\ ]\!] & \Longrightarrow & \Gamma \rhd (A\ \textbf{imp}\ G) \\ [\![\ \forall x.\ \Gamma \rhd G\ x\ ]\!] & \Longrightarrow & \Gamma \rhd (\textbf{all}\ x.\ G\ x) \\ [\![\ A \in \Gamma\ ]\!] & \Longrightarrow & \Gamma \rhd \langle A \rangle \\ [\![\ A \longleftarrow G;\ \Gamma \rhd G\ ]\!] & \Longrightarrow & \Gamma \rhd \langle A \rangle \end{array}$$

The highlighted items are OL-specific and specified later.

Two-level Intuitionistic: Judgments

The syntax and evaluation judgments are inductively defined predicates, unchanged from the one-level version.

The typing judgments are encoded via the SL (except for **off** which is encoded directly in Isabelle/HOL). The datatype *atm* has one constructor for each such judgment:

$$\begin{aligned} atm ::= &\ \mathbf{ofe}\ (addr \times tp\ set)\ exp\ tp \mid \mathbf{ofk}\ (addr \times tp\ set)\ cont\ tp\ tp \\ \mid &\ \mathbf{ofs}\ (addr \times tp\ set)\ state\ (addr \times tp\ set) \end{aligned}$$

Two-level Intuitionistic: Judgments, continued

$atm ::= \textbf{ofe}\ (addr \times tp\ set)\ exp\ tp \mid \ldots$

The definition of $\longleftarrow$ encodes the corresponding rules:

$$\begin{array}{lll}
\textit{inductive} & \_ \longleftarrow \_ :: [\,atm, prp\,] \Rightarrow bool & \\
\textit{intros} & & \\
\textbf{abstr}\ f & \Longrightarrow & \textbf{ofe}\ D\ (\textbf{fn}\, x.\, f\ x)\ (\text{tFun}\ s\ t) \longleftarrow \\
 & & \quad \textbf{all}\, x.\, \langle \textbf{ofe}\ D\ x\ s \rangle\ \textbf{imp}\ \langle \textbf{ofe}\ D\ (f\ x)\ t \rangle \\
 & \Longrightarrow & \textbf{ofe}\ D\ (g \cdot x)\ t \longleftarrow \\
 & & \quad \langle \textbf{ofe}\ D\ g\ (\text{tFun}\ s\ t) \rangle\ \textbf{and}\ \langle \textbf{ofe}\ D\ x\ s \rangle \\
 & \text{etc.} &
\end{array}$$

Two-level Intuitionistic: Subject Reduction

The statement of subject reduction now refers to provability in the SL:

Isabelle/HOL Theorem

$[\![\, \textbf{eval}\ s\ \textbf{kDone}\ e\ w;\ \textbf{ofs}\ D\ s\ D;\ \cdot \rhd \textbf{ofe}\ D\ e\ t \,]\!] \Longrightarrow \textbf{off}\ D\ w\ t$

The typing context C has disappeared, being replaced with the SL's context of assumptions.

Although this context is used in type-checking an ML^{ref} term, it remains empty throughout the proof of subject reduction, thanks to the substitution property of the SL.

Two-level Intuitionistic: Observations

- Evaluation was kept at the Isabelle/HOL (meta) level.
- Weakening for typing contexts followed directly from the structural properties of the specification logic. However, weakening for store typings was a bit tricky, requiring induction over SL judgments and definition of functions on SL propositions.
- The specification logic rule for universal quantification uses HOAS directly in Isabelle/HOL. This may limit the ability to reason about SL derivations.
- Some freshness reasoning was required, though it was not complicated.
- The proof size was only a little bit larger than the previous version, and the easy cases of subject reduction were still mostly automatic.

Outline

uOttawa

Linear Specification Logic

The SL's provability judgment is extended with a linear context Δ:

inductive $\_ , \_ \rhd \_ :: [\, atm\ list, atm\ list, prp \,] \Rightarrow bool$

There are now two conjunctions, additive (&) and multiplicative($\otimes$); also two implications, one for each context ($\rightarrow$ and $\multimap$).

$$\begin{array}{rcl}
[\![\, \Gamma, \Delta \rhd G_1 ;\ \Gamma, \Delta \rhd G_2 \,]\!] & \Longrightarrow & \Gamma, \Delta \rhd (G_1 \ \&\ G_2) \\
[\![\, \Gamma, \Delta_1 \rhd G_1 ;\ \Gamma, \Delta_2 \rhd G_2 \,]\!] & \Longrightarrow & \Gamma, (\Delta_1, \Delta_2) \rhd (G_1 \otimes G_2) \\
[\![\, (A, \Gamma), \Delta \rhd G \,]\!] & \Longrightarrow & \Gamma, \Delta \rhd (A \rightarrow G) \\
[\![\, \Gamma, (A, \Delta) \rhd G \,]\!] & \Longrightarrow & \Gamma, \Delta \rhd (A \multimap G) \\
[\![\, A \in \Gamma ;\ \Delta = \{\} \,]\!] & \Longrightarrow & \Gamma, \Delta \rhd \langle A \rangle \\
[\![\, \Delta = \{A\} \,]\!] & \Longrightarrow & \Gamma, \Delta \rhd \langle A \rangle
\end{array}$$

uOttawa

Two-level Linear: Judgments

The syntax and evaluation judgments are unchanged. The typing judgments lose their store-typing argument, instead placing that information in the linear context in the form of new atoms **ofc** *addr tp* and **ofcL** *addr tp*.

Now all four of the typing judgments are encoded in the SL, using the linear context in the case of **ofs** and **off**.

Two-level Linear: Observations

- For this version, it was necessary to allow non-empty contexts in the induction hypothesis for subject reduction. This was encapsulated in a separate predicate called the context invariant.
- A substitution lemma for the linear context was proved, but never used.
- Some auxiliary predicates involving the store typing needed major changes to move them from the meta-level to the SL level.
- There was a major loss of proof automation with a corresponding increase in the size of the proofs.
- Longer and uglier freshness reasoning was required.

Outline

uOttawa

Two-level Linear (incl. eval.): Judgments

The specification logic and typing judgments are unchanged; the difference in this version is the encoding of evaluation. The **eval** and **cont** predicates are replaced with SL atoms. New atoms in the context are used to store the state. And two more defined atoms are used to "pack" the state from the linear context into the answer.

Two-level Linear (incl. eval.): Observations

With all OL judgments encoded in the SL, it was no longer possible to structure the proofs using induction on Isabelle/HOL datatypes. Structural induction over SL derivations is possible but very awkward. So I added a natural-number argument to the SL provability judgment, in the style of [McDowell & Miller, 2002], allowing a form of size induction.

A provability judgment without this extra argument was obtained by existential quantification, with all but one of the old rules as derived rules. The one exception is the infinitely-branching introduction rule for the SL's universal quantifier.

Induction on the natural-number argument was workable, but certain aspects of proof automation were completely lost; there were no fully automatic cases in the proof of subject reduction.

Two-level Linear (incl. eval.): Observations, cont.

- The context invariant was modularized, and parts of it were reused in the induction hypotheses for various lemmas leading up to subject reduction.
- It was still necessary to represent answers as Hybrid expressions, so the encoding had to include packing of the linear context into such expressions.
- Establishing the context invariant from the premises of subject reduction was not as trivial as before.
- The last of the meta-level auxiliary lemmas involving states were moved to the SL level.

Outline

uOttawa

Ordered Specification Logic

The SL's provability judgment is extended again, this time with an ordered context Ω:

$$\textit{inductive} \quad \_\,,\,\_\,,\,\_ \rhd \_ :: [\,\textit{atm list}, \textit{atm list}, \textit{atm list}, \textit{prp}\,] \Rightarrow \textit{bool}$$

There are *two* new implication connectives, differing in whether the new assumption appears at the left or the right of Ω.

uOttawa

Two-level Ordered: Judgments

The evaluation judgments lose their continuation argument, with this information being encoded in the ordered context.

Observations

- The differences in the proofs from the previous version were surprisingly "shallow", indeed close to being merely a syntactic change.
- Having just one ordered context could be limiting for larger examples; it should be possible to have more, with proper "logic engineering".
- A substitution lemma for the ordered context was proved, but never used.

Outline

uOttawa

Proof Statistics

Proof	Lines (all)
One-level	675
Two-level intuitionistic	802
Two-level linear	1893
Two-level linear (including evaluation)	2247
Two-level ordered	2534

As more use was made of HOAS and substructural features, the form of the OL judgments became simpler; the encodings grew a little, but remained reasonable. But the proofs became much longer and less automated when substructural SLs were used.

Future Work

- Hybrid could be applied to other examples, such as the POPLmark challenge.
- For larger examples, especially with substructural SLs, improvements to Hybrid are needed to keep proof sizes reasonable. Support for object-language types (in the style of Isabelle's datatype package, or the Nominal package) would be best, but special-purpose tactics might be sufficient.
- It remains to prove adequacy of the encodings.

References

Cervesato, I., and F. Pfenning, *A Linear Logical Framework*, in: *Proceedings of the Eleventh Annual IEEE Symposium on Logic in Computer Science – LICS'96*, New Brunswick, NJ, July 27–30, 1996.

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

Momigliano, A., A. Martin, and A. Felty, *Two-Level Hybrid: A System for Reasoning Using Higher-Order Abstract Syntax*, in: *Electronic Notes in Theoretical Computer Science* 196 (2008), pp. 85-93.

uOttawa

References

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