Canonical HybridLF:
Extending Hybrid with Dependent Types

Roy L. Crole & Amy Furniss

University of Leicester, UK

1 September 2015
Our work concerns reasoning about logics, programming languages . . . and other object logics by
- translating into a metalogic: $OL \leftrightarrow ML$;
- reasoning about $OL$ in $ML$;

The Hybrid metalogic was developed at Leicester, UK, by Ambler, Crole and Momigliano.

Hybrid is an implementation of HOAS in Isabelle-HOL:
- Users write syntax with “human friendly” named binders.
- Hybrid converts such syntax to “machine friendly” nameless de Bruijn notation;
- Special conversion functions are at the heart of Hybrid.
Background and Motivation

- **HYBRID** is based upon the untyped lambda-calculus. Our contribution is to show that:
  - The *conceptual ideas* behind the conversion functions extend to (simply typed and) dependently typed settings.
  - One may reason with the *judgements-as-types methodology* (within Isabelle).
  - The concept of **HYBRID** unary abstraction extends to finitary abstraction.
  - *Types* eliminate the need for well-formedness **HYBRID** predicates.

- To introduce **Canonical HYBRIDLF** we first recall the **HYBRID** metalogic in a little more detail . . .
Central to Hybrid is a Core Datatype . . .

. . . the untyped lambda calculus in nameless de Bruijn form

\[ \text{'a } \text{expr } ::=} \text{BND nat } \mid \text{VAR nat } \mid \text{CON 'a } \]

\[ \mid \text{ABS expr } \mid \text{APP expr expr } \mid \text{ERR expr $\$ expr} \]

eg 'a = \text{cForAll} \mid \text{cExists}: \text{object level } \forall \text{ rendered as CON cforAll.} \]
The Hybrid System

- the untyped lambda calculus in nameless de Bruijn form

\[
'a\ expr ::= BND\ nat \mid VAR\ nat \mid CON\ 'a
\]

\[
\mid ABS\ expr \mid APP\ expr\ expr\mid ERR
\]

\[
\text{ERR} = expr\ \$\$\ expr
\]

together with user syntax \texttt{LAM} \(v.e\) that is converted to an \texttt{ABS}-expression of the Core Datatype

- \texttt{Hybrid} provides HOAS: Object level \(\forall v.\phi\) is encoded as

\[
(CON\ cforAll) \ $(LAM\ v.\phi)$
\]
The Hybrid System

- The untyped lambda calculus in nameless de Bruijn form

```latex
\begin{align*}
' \alpha \ expr & ::= \ BND \ nat \mid VAR \ nat \mid CON \ ' \alpha \\
& \mid ABS \ expr \mid APP \ expr \ expr \mid ERR \\
& \quad \text{expr $$\$\$\$$ expr}
\end{align*}
```

together with user syntax \texttt{LAM} \texttt{v.e} that is converted to an \texttt{ABS}-expression of the Core Datatype

- \texttt{LAM} \texttt{v.e} is an abbreviation for \texttt{ABS} \texttt{(lbind 0 (\Lambda v.e))} where \texttt{\Lambda v.e} is an \textit{Isabelle function} and \texttt{lbind converts it to a de Bruijn expression}, provided that ...

- ... \texttt{\Lambda v.e} is a (unary) \texttt{HYBRID abstraction}.
Roughly, abstractions are formed from level 1 expressions such as \( \text{ABS (VAR 0 \text{\$\$ BND 1})} \) in which the dangling variable is “Isabelle-function-abstracted”

\[
\Lambda v. \text{ABS (VAR 0 \text{\$\$ v})} :: \text{expr} \Rightarrow \text{expr}
\]

... and \text{lbind} reverses this

\[
\text{lbind 0 (\Lambda v. \text{ABS (VAR 0 \text{\$\$ v})})} = \text{ABS (VAR 0 \text{\$\$ BND 1})}
\]

An abstraction is any \( e :: \text{expr} \Rightarrow \text{expr} \) for which \( \text{LAM } v. e \ v \) is level 0, ie proper, de Bruijn. (“lambda-calculus expression”)

We have a predicate \text{abstr}

\[
\text{abstr } e \iff \text{proper (LAM } v. e \ v\)
\]
LF and Canonical LF

- LF is the (well known!) Edinburgh Logical Framework of Harper, Honsell and Plotkin.
- It is a dependently-typed lambda calculus, intended as a metalogic for reasoning about object logics.
  - For each judgement $J$ of the object logic an LF type $j$ is created; and
    - a proof that $J$ holds is given by an LF expression $e$ such that $e :: j$.
- This is often called the judgements-as-types approach.
LF and Canonical LF

- LF has a notion of *canonical form*: expressions in $\beta$-normal, $\eta$-long—the expressions of Canonical LF.
- Watkins, Cervesato, Pfenning and Walker give a *canonical* presentation of LF:
  - only kinds, terms and types in canonical form can be formed;
  - definitional equality is syntactic equality;
  - utilises *hereditary substitution*: ensures that any substitution yields a canonical expression.

- **Canonical HybridLF** is an implementation (we also have an implementation of LF).
The Philosophy of CANONICAL HYBRIDLF

- In HYBRID the methodology is
  - encode object logics using “lambda calculus HOAS” terms;
  - reason directly in Isabelle HOL (after conversion to de Bruijn).

- In CANONICAL HYBRIDLF the user’s methodology is different: we have a HOAS interface to an implementation of Canonical LF.
  - Theorems are defined via a signature: one can use named bound variables which are converted to de Bruijn form;
Key Ingredients of Canonical Hybrid LF

Analogous to Hybrid:

- A Core Datatype for the syntactic expressions of Canonical LF in a de Bruijn form.
- Canonical Hybrid LF Abstraction predicates.
- lbind functions that convert expressions with named variables to de Bruijn form.

Further

- An implementation of the Canonical LF formal system.
The Core Datatype of CANONICAL HYBRIDLF

The raw expressions of Canonical LF . . .

\[
K ::= K T y p e \mid \Pi x : A . K \quad A ::= a P \mid \Pi x : A . A \quad M ::= m R \mid \lambda x : A . M \\
\quad P ::= k \mid P \ M \quad R ::= x \mid [c] \mid R \ M
\]

. . . captured by the Core Datatype:

datatype

  

  

  and ('a, 'b) cterm = ATERM "('a, 'b) aterm"

  | ABS "('a, 'b) ctype" "('a, 'b) cterm"

and ('a, 'b) aterm = VAR nat | BND nat | CON 'a

  | APP "('a, 'b) aterm" "('a, 'b) cterm"

LSFA 2015, Natal, Brazil
The Core Datatype of CANONICAL HYBRIDLF

The raw expressions of Canonical LF . . .

\[ K ::=^k \text{Type} \mid \Pi x:A.K \quad A ::=_a P \mid \Pi x:A.A \quad M ::=_m R \mid \lambda x:A.M \]
\[ P ::=^P k \mid P M \quad R ::=^R x \mid c \mid R M \]

. . . captured by the Core Datatype:

datatype

: 

: 

and ('a, 'b) cterm = ATERM "('a, 'b) aterm"
\| ABS "('a, 'b) ctype" "('a, 'b) cterm"

and ('a, 'b) aterm = VAR nat \| BND nat \| \text{CON 'a}
\| APP "('a, 'b) aterm" "('a, 'b) cterm"
The Core Datatype of CANONICAL HYBRID LF

The raw expressions of Canonical LF . . .

\[
\begin{align*}
K ::= & \text{Type} \mid \Pi x: A. K \\
A ::= & \text{a} \ P \mid \Pi x: A. A \\
M ::= & \text{m} \ R \mid \lambda x: A. M \\
P ::= & \text{p} \ k \mid P \ M \\
R ::= & \text{r} \ x \mid \text{c} \mid R \ M
\end{align*}
\]

. . . captured by the Core Datatype:

datatype

. . .

and (’a, ’b) cterm = ATERM "(’a, ’b) aterm"

| ABS "(’a, ’b) ctype" "(’a, ’b) cterm"

and (’a, ’b) aterm = VAR nat | BND nat | CON ’a

| APP "(’a, ’b) aterm" "(’a, ’b) cterm"


**Canonical Hybrid LF Abstractions**

\[ M ::= \text{m} R | \lambda x : A. M \quad R ::= \text{r} x | c | R M \]

Recursively define an abstraction test on canonical terms \( M \):

- \( \text{cterm\_abstr} \ i \ (\Lambda v. \ \text{ATERM} \ (\text{VAR} \ n)) = \text{True} \)
- \( \text{cterm\_abstr} \ i \ (\Lambda v. \ \text{ATERM} \ (\text{BND} \ n)) = (n < i) \)
- \( \text{cterm\_abstr} \ i \ (\Lambda v. \ \text{ATERM} \ (\text{CON} \ c)) = \text{True} \)
- \( \text{cterm\_abstr} \ i \ (\Lambda v. \ \text{ATERM} \ ((f \ v) \ $$ \circ \ (g \ v))) = \\
  (\text{aterror\_abstr} \ i \ f \land \text{cterm\_abstr} \ i \ g) \)
- \( \text{cterm\_abstr} \ i \ (\Lambda v. \ \text{ABS} \ (ty \ v) \ (f \ v)) = \\
  (\text{ctype\_abstr} \ i \ ty \land \text{cterm\_abstr} \ (i + 1) \ f) \)
The first two clauses deal with variables

\[ M ::=_{m} R \mid \lambda x: A. M \quad R ::=_{r} x \mid c \mid R \, M \]

\[ \text{cterm\_abstr} \, i \, (\Lambda v. \, \text{ATERM} \, (\text{VAR} \, n)) = \text{True} \]

\[ \text{cterm\_abstr} \, i \, (\Lambda v. \, \text{ATERM} \, (\text{BND} \, n)) = (n < i) \]

**Canonical HybridLF Abstractions**
The third clause deals with constants

\[ \text{cterm\_abstr } i \ (\Lambda v. \ \text{ATERM} \ (\text{CON } c)) = \text{True} \]
The fourth clause deals with $R M$ (coded $f \, v$ and $g \, v$)

$$\text{cterm\_abstr}\ i\ (\Lambda v.\ \text{ATERM}\ \((f\ v)\ \circ\ (g\ v))\)) = (\text{atrm\_abstr}\ i\ f\ \land\ \text{cterm\_abstr}\ i\ g)$$

Recursive call for $R\ (f\ v)$

Recursive call for $M\ (g\ v)$
$M ::= _m R | \lambda x:A.M \quad R ::= _r x | c | R M$

The fifth clause deals with lambda-expressions

\[
\text{cter}_\text{m}_\text{abstr} \ i \ (\Lambda v. \ \text{ABS} (ty \ v) \ (f \ v)) = \\
(\text{ctype}_\text{abstr} \ i \ ty \land \text{cter}_\text{m}_\text{abstr} \ (i+1) \ f)
\]

Recursive call for $A$

Recursive call for $M$
Conversion to de Bruijn: The lbind Functions

- Recall the conversion function \textbf{lbind} that converts a (unary) abstraction \( \Lambda v. e \) to a level-1 de Bruijn expression.
- We define analogues of \textbf{lbind}, by mutual recursion, over the canonical and atomic types; and the canonical and atomic terms.
- Here is part of the definition over canonical terms:

\[
\text{cterm_bind } i \ (\Lambda v. \text{ABS } (ty\ v)\ (f\ v)) = \\
\text{Canonical HybridLF Abstraction}
\]

\[
(\text{case } (\text{ctype_bind } i\ ty) \text{ of Some } t \Rightarrow \\
\ (\text{case } (\text{cterm_bind } (i + 1)\ f) \text{ of Some } m \Rightarrow \\
\text{Some } (\text{ABS } t\ m))
\]
The CANONICAL HYBRIDLF Formal System

Function name  | Recursion parameter | Context | Object signature | Type signature | Binding environment
----------------|---------------------|---------|------------------|----------------|-----------------------

\[ \text{function} \quad d \quad \text{ctx} \quad \text{sig}_t \quad \text{sig}_k \quad \text{bnd} \quad \zeta = \zeta' \]

- Typing judgement of LF \[ \Gamma \vdash_{\Sigma} \xi : \xi' \] implemented by

  \[ \text{function} \quad d \quad \text{bnd} \quad : \quad (\Gamma, \Sigma, \xi) \quad \mapsto \quad \xi' \]

- \(d\) measures recursion-depth: all functions must terminate.

- \(\text{bnd}\) is a list of canonical types. When typing \(\text{ABS} \ t \ m\), we recursively determine the type of the body \(m\) with \(t \not\in \text{bnd}\).
The CANONICAL HYBRIDLF Formal System

\[ \Gamma \vdash_{\Sigma} P : \Pi x : A. K \quad \Gamma \vdash_{\Sigma} M : A \quad [M/x]_{A}^{k} K = K' \]

\[ \Gamma \vdash_{\Sigma} P \ M : K' \]

atom_kindof \((d + 1)\) ctx sig_t sig_k bnd (FAPP p m) =

\( \begin{cases} \text{case atom_kindof } d \ \text{ctx sig}_t \ \text{sig}_k \ \text{bnd } p \ \text{of } \text{Some } (\text{KPI } a \ k) \\ \Rightarrow \text{case canon_typeof } d \ \text{ctx sig}_t \ \text{sig}_k \ \text{bnd } m \ \text{of } \text{Some } a \\ \Rightarrow \text{kind_subst_bv } d \ \text{ctx sig}_t \ \text{sig}_k \ \text{bnd } m \ 0 \ 0 \ k \end{cases} \)
The CANONICAL HYBRIDLF Formal System

\[ \Gamma \vdash \Sigma \quad P : \Pi x : A. K \quad \Gamma \vdash \Sigma \quad M : A \quad [M/x]^k_A K = K' \]
\[ \Gamma \vdash \Sigma \quad P \ M : K' \]

\text{atom\_kindof} \quad (d + 1) \ \text{ctx sig\_t sig\_k bnd} \ (\text{FAPP} \ p \ m) =

\text{(case atom\_kindof} \ d \ \text{ctx sig\_t sig\_k bnd} \ p \ \text{of Some} \ (\text{KPI} \ a \ k) \quad \Rightarrow \ \text{(case canon\_typeof} \ d \ \text{ctx sig\_t sig\_k bnd} \ m \ \text{of Some} \ a \quad \Rightarrow \ \text{kind\_subst\_bv} \ d \ \text{ctx sig\_t sig\_k bnd} \ m \ 0 \ 0 \ k)
The **CANONICAL HYBRIDLF** Formal System

\[ \Gamma \vdash \Sigma \; P : \Pi x : A.K \quad \Gamma \vdash \Sigma \; M : A \quad [M/x]_A^k K = K' \]

\[ \Gamma \vdash \Sigma \; P \; M : K' \]

\text{atom\_kindof} \ (d + 1) \ ctx \ sig\_t \ sig\_k \ bnd \ (\text{FAPP} \ p \ m) =

\[ (\text{case} \ \text{atom\_kindof} \ d \ ctx \ sig\_t \ sig\_k \ bnd \ p \ of \ \text{Some} \ (\text{KPI} \ a \ k) \Rightarrow (\text{case} \ \text{canon\_typeof} \ d \ ctx \ sig\_t \ sig\_k \ bnd \ m \ of \ \text{Some} \ a \Rightarrow \text{kind\_subst\_bv} \ d \ ctx \ sig\_t \ sig\_k \ bnd \ m \ 0 \ 0 \ k) \]
We implemented an analogue of the case study of the simply typed lambda calculus in Twelf (from 2007):

- Simple types generated from a unit type;
- a type assignment system;
- a single-step operational semantics;
- a proof of type preservation.

datatype
type_cons =
    tp | tm | var_of_type | pres | val | step

object_cons =
    unit | arrow | singleton | app | lam | ...
Object level \texttt{lam}-functions are values:

(i) first order syntax

\[(\forall E:tm)(\forall T:tp)(\text{val (lam } x:T.E))\]
Object level $\text{lam}$-functions are values:

(i) first order syntax

$$\forall E : \text{tm} \forall T : \text{tp} (\text{val} (\text{lam } x : T. E))$$

(ii) Canonical HybridLF

$$\text{PI} (\text{tm} \Rightarrow \text{tm}) (\Lambda E. \text{PI} \text{tp} (\Lambda T. \text{val} \ F (\text{lam} \ F_0 T \ F_0 E)))$$

Types eliminate Hybrid well-formedness predicates
Object level \texttt{lam}-functions are values:

(ii) \texttt{Canonical HybridLF}

\begin{align*}
\text{PI } (tm \Rightarrow tm) & \left( \Lambda E. \text{PI } \text{tp} \left( \Lambda T. \text{val } $$F (\text{lam } $$O T $$O E) \right) \right) \\
\text{(iii) which equals} & \hspace{1cm} \texttt{ctype\_bind2 } (tm \Rightarrow tm) \\
& \left( \Lambda E. \text{tp} \right) \\
& \left( \Lambda E. \Lambda T. \text{val } $$F (\text{lam } $$O T $$O E) \right)
\end{align*}
Case Study: Simply Typed Lambda Calculus

Object level lam-functions are values:

(ii) Canonical HybridLF
\[
\text{PI (tm } \Rightarrow \text{ tm) (} \Lambda \ E. \ \text{PI tp} (\Lambda \ T. \ \text{val } \$$f$$ (\text{lam } \$$o$$ \ T \ $$o$$ E))\]

(iii) which equals
\[
\text{ctype_bind2 (tm } \Rightarrow \text{ tm)} (\Lambda \ E. \ \text{tp})
\]
\[
(\Lambda \ E. \ \Lambda T. \ \text{val } \$$F$$ (\text{lam } \$$O$$ \ T \ $$O$$ E))
\]

(iv) which evaluates to
\[
\text{PI (tm } \Rightarrow \text{ tm) (PI tp (val } \$$F$$ ((\text{lam } \$$O$$ (BND 0)) $$O$$ (BND 1))))
\]
Case Study: Simply Typed Lambda Calculus

Object level \textit{lam}-functions are values:

(ii) \textbf{Canonical HybridLF}
\begin{equation}
\Pi (tm \Rightarrow tm) (\Lambda E. \Pi tp (\Lambda T. \text{val } \frac{F}{O} (\text{lam } \frac{O}{O} T \frac{O}{O} E)))
\end{equation}

(iii) which equals
\begin{equation}
\text{ctype\_bind2} (tm \Rightarrow tm)
\end{equation}
\begin{equation}
(\Lambda E. \text{tp})
\end{equation}
\begin{equation}
(\Lambda E. \Lambda T. \text{val } \frac{F}{O} (\text{lam } \frac{O}{O} T \frac{O}{O} E))
\end{equation}

(iv) which evaluates to
\begin{equation}
\Pi (tm \Rightarrow tm) (\Pi tp (\text{val } \frac{F}{F} ((\text{lam } \frac{O}{O} (\text{BND } 0)) \frac{O}{O} (\text{BND } 1))))
\end{equation}
Conclusions

- Can the techniques of Hybrid be migrated to a dependently typed setting? Yes.
- The type system replaces well-formedness predicates of Hybrid (e.g., \texttt{isTerm E} and \texttt{isType T}).
- In Canonical HybridLF we make a number of advances over the existing Hybrid systems, such as implementing finitary abstractions rather than just unary abstractions.
- An interesting topic might be formal adequacy proofs like the one for Hybrid.
- A journal paper will outline similar work for (standard) LF.
- Canonical HybridLF proof search is often long and tedious; in Twelf this is automatic. Canonical HybridLF currently lacks automation of unification and proofs of totality.