Canonical HybridLF:
Extending Hybrid with Dependent Types

Roy L. Crole & Amy Furniss

University of Leicester, UK

1 September 2015

Background and Motivation

- Our work concerns reasoning about logics, programming languages . . . and other *object logics* by
  - translating into a *metalogic*: $\boldsymbol{OL} \mapsto \boldsymbol{ML}$;
  - reasoning *about* $\boldsymbol{OL}$ in $\boldsymbol{ML}$;
- 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 . . .

The Hybrid System

- Central to Hybrid is a Core Datatype . . .
- . . . the untyped lambda calculus in nameless de Bruijn form

$$
\begin{aligned}
{'a}\ \mathtt{expr} \quad ::= \quad & \mathtt{BND\ nat} \mid \mathtt{VAR\ nat} \mid \mathtt{CON}\ {'a} \\
& \mid \mathtt{ABS\ expr} \mid \underbrace{\mathtt{APP\ expr\ expr}}_{\mathtt{expr}\ \$\$\ \mathtt{expr}} \mid \mathtt{ERR}
\end{aligned}
$$

eg $'a$ = **cForAll** | **cExists**: object level ∀ rendered as CON **cforAll**.

The Hybrid System

- … the untyped lambda calculus in nameless de Bruijn form

$$'a\ \texttt{expr} \quad ::= \quad \texttt{BND nat} \mid \texttt{VAR nat} \mid \texttt{CON}\ 'a$$

$$\mid \texttt{ABS expr} \mid \underbrace{\texttt{APP expr expr}}_{\texttt{expr \$\$ expr}} \mid \texttt{ERR}$$

together with user syntax **LAM** $v.e$ that is converted to an `ABS`-expression of the Core Datatype

- HYBRID provides HOAS: Object level $\forall v.\phi$ is encoded as

$$(\texttt{CON}\ \textbf{cforAll})\ \$\$(\textbf{LAM}\ v.\overline{\phi})$$

The Hybrid System

- … the untyped lambda calculus in nameless de Bruijn form

$$'a\ \texttt{expr} \ ::= \ \texttt{BND nat} \mid \texttt{VAR nat} \mid \texttt{CON}\ 'a \mid \texttt{ABS expr} \mid \underbrace{\texttt{APP expr expr}}_{\texttt{expr \$\$ expr}} \mid \texttt{ERR}$$

together with user syntax **LAM** $v.e$ that is converted to an `ABS`-expression of the Core Datatype

- **LAM** $v.e$ is an abbreviation for `ABS` $(\textbf{lbind}\ \mathbf{0}\ (\Lambda v.e))$ where $\Lambda v.e$ is an *Isabelle function* and **lbind** *converts it to a de Bruijn expression*, provided that …
- … $\Lambda v.e$ is a (unary) HYBRID abstraction.

Book-Keeping: HYBRID Abstractions

- Roughly, abstractions are formed from level 1 expressions such as $\mathtt{ABS}\ (\mathtt{VAR}\ \mathbf{0}\ \$\$\ \mathtt{BND}\ \mathbf{1})$ in which the dangling variable is "Isabelle-function-abstracted"

$$\Lambda v.\ \mathtt{ABS}\ (\mathtt{VAR}\ \mathbf{0}\ \$\$\ v) :: \mathtt{expr} \Rightarrow \mathtt{expr}$$

- . . . and **lbind** reverses this

$$\textbf{lbind}\ \mathbf{0}\ (\Lambda v.\ \mathtt{ABS}\ (\mathtt{VAR}\ \mathbf{0}\ \$\$\ v)) = \mathtt{ABS}\ (\mathtt{VAR}\ \mathbf{0}\ \$\$\ \mathtt{BND}\ \mathbf{1})$$

- An abstraction is any $e :: \mathtt{expr} \Rightarrow \mathtt{expr}$ for which $\textbf{LAM}\ v.\ e\ v$ is level 0, ie proper, de Bruijn. ("lambda-calculus expression")
- We have a predicate **abstr**

$$\textbf{abstr}\ e \implies \text{proper}\ (\textbf{LAM}\ v.\ e\ v)$$

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 β-normal, η-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 HYBRIDLF

Analogous to HYBRID:

- A Core Datatype for the syntactic expressions of Canonical LF in a de Bruijn form.
- CANONICAL HYBRIDLF 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 . . .

$$K ::=_{\mathsf{k}} \mathrm{Type} \mid \Pi x{:}A.K \qquad A ::=_{\mathsf{a}} P \mid \Pi x{:}A.A \qquad M ::=_{\mathsf{m}} R \mid \lambda x{:}A.M$$

$$P ::=_{\mathsf{p}} k \mid P\ M \qquad R ::=_{\mathsf{r}} x \mid \boxed{c} \mid R\ M$$

. . . 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"
```

The Core Datatype of CANONICAL HYBRIDLF

The raw expressions of Canonical LF ...

$K ::=_{\mathsf{k}} \mathrm{Type} \mid \Pi x{:}A.K \qquad A ::=_{\mathsf{a}} P \mid \Pi x{:}A.A \qquad M ::=_{\mathsf{m}} R \mid \lambda x{:}A.M$

$\qquad\qquad\qquad\qquad\qquad P ::=_{\mathsf{p}} k \mid P\ M \qquad\qquad R ::=_{\mathsf{r}} x \mid \boxed{c} \mid R\ M$

... 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"
```

The Core Datatype of CANONICAL HYBRIDLF

The raw expressions of Canonical LF . . .

$$K ::=_{\mathsf{k}} \mathtt{Type} \mid \Pi x{:}A.K \qquad A ::=_{\mathsf{a}}\ P \mid \Pi x{:}A.A \qquad M ::=_{\mathsf{m}}\ R \mid \lambda x{:}A.M$$

$$P ::=_{\mathsf{p}}\ k \mid P\ M \qquad R ::=_{\mathsf{r}}\ x \mid \boxed{c} \mid R\ M$$

. . . 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"
```

Canonical HybridLF Abstractions

$$M ::=_{\mathsf{m}} \ R \mid \lambda x{:}A.M \qquad R ::=_{\mathsf{r}} \ x \mid c \mid R\ M$$

Recursively define an abstraction test on canonical terms M

$$\textbf{cterm\_abstr}\ i\ (\Lambda v.\ \texttt{ATERM}\ (\texttt{VAR}\ n)) = \texttt{True}$$

$$\textbf{cterm\_abstr}\ i\ (\Lambda v.\ \texttt{ATERM}\ (\texttt{BND}\ n)) = (n < i)$$

$$\textbf{cterm\_abstr}\ i\ (\Lambda v.\ \texttt{ATERM}\ (\texttt{CON}\ c)) = \texttt{True}$$

$$\textbf{cterm\_abstr}\ i\ (\Lambda v.\ \texttt{ATERM}\ ((f\ v)\ \$\$_{\mathrm{O}}\ (g\ v))) = \\ (\textbf{aterm\_abstr}\ i\ f \wedge \textbf{cterm\_abstr}\ i\ g)$$

$$\textbf{cterm\_abstr}\ i\ (\Lambda v.\ \texttt{ABS}\ (ty\ v)\ (f\ v)) = \\ (\textbf{ctype\_abstr}\ i\ ty \wedge \textbf{cterm\_abstr}\ (i+1)\ f)$$

CANONICAL HYBRIDLF Abstractions

$$M ::=_{\mathsf{m}} R \mid \lambda x{:}A.M \qquad R ::=_{\mathsf{r}} x \mid c \mid R\ M$$

The first two clauses deal with variables

$$\textbf{cterm\_abstr}\ i\ (\Lambda v.\ \texttt{ATERM}\ (\texttt{VAR}\ n)) = \texttt{True}$$

$$\textbf{cterm\_abstr}\ i\ \underbrace{(\Lambda v.\ \texttt{ATERM}\ (\texttt{BND}\ n))}_{\text{CANONICAL HYBRIDLF Abstractions}} = (n < i)$$

CANONICAL HYBRIDLF Abstractions

$$M ::=_{\mathsf{m}} R \mid \lambda x{:}A.M \qquad R ::=_{\mathsf{r}} x \mid c \mid R\,M$$

The third clause deals with constants

$$\textbf{cterm\_abstr}\ i\ (\Lambda v.\ \texttt{ATERM}\ (\texttt{CON}\ c)) = \texttt{True}$$

CANONICAL HYBRIDLF Abstractions

$$M ::=_{\mathsf{m}}\ R \mid \lambda x{:}A.M \qquad R ::=_{\mathsf{r}}\ x \mid c \mid R\ M$$

The fourth clause deals with $R\ M$ (coded $f\ v$ and $g\ v$)

$$\textbf{cterm\_abstr}\ i\ (\Lambda v.\ \texttt{ATERM}\ ((f\ v)\ \$\$_{\mathrm{O}}\ (g\ v))) =$$
$$(\textbf{aterm\_abstr}\ i\ f \wedge \textbf{cterm\_abstr}\ i\ g)$$

Recursive call for R ($f\ v$)

Recursive call for M ($g\ v$)

Canonical HybridLF Abstractions

$$M ::=_{\mathsf{m}}\ R \mid \lambda x{:}A.M \qquad R ::=_{\mathsf{r}}\ x \mid c \mid R\ M$$

The fifth clause deals with lambda-expressions

$$\mathbf{cterm\_abstr}\ i\ (\Lambda v.\ \mathtt{ABS}\ (ty\ v)\ (f\ v)) = \\ (\mathbf{ctype\_abstr}\ i\ ty \wedge \mathbf{cterm\_abstr}\ (i+1)\ f)$$

Recursive call for A

Recursive call for M

Conversion to de Bruijn: The lbind Functions

- Recall the conversion function **lbind** that converts a (unary) abstraction $\Lambda v.\ e$ to a level-1 de Bruijn expression.
- We define analogues of **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:

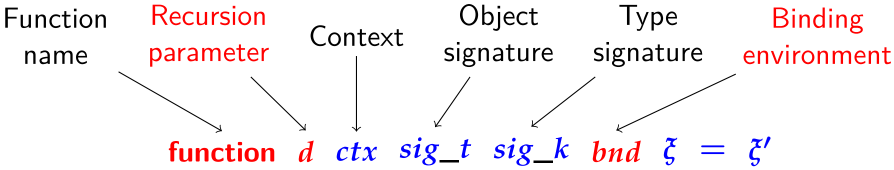
$$\textbf{cterm\_bind}\ i\ \underbrace{(\Lambda v.\ \texttt{ABS}\ (ty\ v)\ (f\ v))}_{\text{Canonical HybridLF Abstraction}} =$$

$$(\texttt{case}\ (\textbf{ctype\_bind}\ i\ ty)\ \texttt{of Some}\ t \Rightarrow$$

$$(\texttt{case}\ (\textbf{cterm\_bind}\ (i+1)\ f)\ \texttt{of Some}\ m \Rightarrow$$

$$\texttt{Some}\ (\texttt{ABS}\ t\ m)$$

The CANONICAL HYBRIDLF Formal System



$$\textbf{function}\ d\ \ ctx\ \ sig\_t\ \ sig\_k\ \ bnd\ \ \xi\ =\ \xi'$$

- Typing judgement of LF $\Gamma \vdash_\Sigma \xi : \xi'$ implemented by

$$\textbf{function}\ d\ bnd\ :\ (\Gamma, \Sigma, \xi)\ \mapsto\ \xi'$$

- d measures recursion-depth: all functions must terminate.
- bnd is a list of canonical types. When typing `ABS` $t\ m$, we recursively determine the type of the body m with $t\ \#\ bnd$.

The Canonical HybridLF Formal System

$$\frac{\Gamma \vdash_\Sigma \; P : \Pi x{:}A.K \qquad \Gamma \vdash_\Sigma \; M : A \qquad [M/x]^{\mathsf{k}}_A K = K'}{\Gamma \vdash_\Sigma \; P\ M : K'}\ \textbf{atom\_kindof}$$

```
atom_kindof (d + 1) ctx sig_t sig_k bnd (FAPP p m) =
  (case atom_kindof d ctx sig_t sig_k bnd p of Some (KPI a k)
    ⇒ (case canon_typeof d ctx sig_t sig_k bnd m of Some a
      ⇒ kind_subst_bv d ctx sig_t sig_k bnd m 0 0 k
```

The CANONICAL HYBRIDLF Formal System

$$\frac{\Gamma \vdash_{\Sigma} P : \Pi x{:}A.K \qquad \Gamma \vdash_{\Sigma} M : A \qquad [M/x]^{\mathsf{k}}_{A} K = K'}{\Gamma \vdash_{\Sigma} P\ M : K'}\ \textbf{atom\_kindof}$$

atom_kindof $(d+1)$ *ctx sig_t sig_k bnd* (`FAPP` *p m*) $=$

(`case` **atom_kindof** *d ctx sig_t sig_k bnd p* `of Some` (`KPI` *a k*)
  $\Rightarrow$ (`case` **canon_typeof** *d ctx sig_t sig_k bnd m* `of Some` *a*
    $\Rightarrow$ **kind_subst_bv** *d ctx sig_t sig_k bnd m* 0 0 *k*

The Canonical HybridLF Formal System

$$\frac{\Gamma \vdash_{\Sigma} P : \Pi x{:}A.K \qquad \Gamma \vdash_{\Sigma} M : A \qquad [M/x]^{\mathsf{k}}_{A} K = K'}{\Gamma \vdash_{\Sigma} P\ M : K'}\ \textbf{atom\_kindof}$$

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

$(\texttt{case}\ \textbf{atom\_kindof}\ d\ ctx\ sig\_t\ sig\_k\ bnd\ p\ \texttt{of Some}\ (\texttt{KPI}\ a\ k)$
$\quad \Rightarrow (\texttt{case}\ \textbf{canon\_typeof}\ d\ ctx\ sig\_t\ sig\_k\ bnd\ m\ \texttt{of Some}\ a$
$\qquad \Rightarrow \textbf{kind\_subst\_bv}\ d\ ctx\ sig\_t\ sig\_k\ bnd\ m\ \mathbf{0}\ \mathbf{0}\ k$

Case Study: Simply Typed Lambda Calculus

- 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 | ...
```

Case Study: Simply Typed Lambda Calculus

Object level `lam`-functions are values:

(i) first order syntax

`(∀ E : tm)(∀ T : tp)(val (lam x : T. E))`

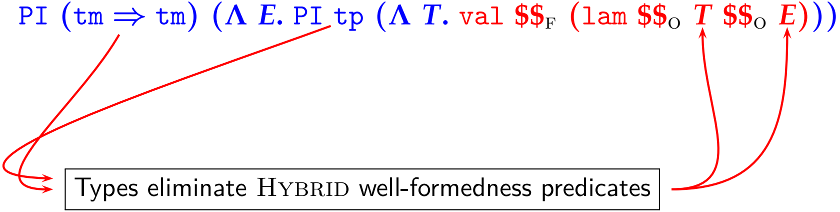
Case Study: Simply Typed Lambda Calculus

Object level `lam`-functions are values:

(i) first order syntax

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

(ii) CANONICAL HYBRIDLF



$\mathtt{PI}\ (\mathtt{tm} \Rightarrow \mathtt{tm})\ (\mathbf{\Lambda}\ E.\ \mathtt{PI}\ \mathtt{tp}\ (\mathbf{\Lambda}\ T.\ \mathtt{val}\ \$\$_{\mathrm{F}}\ (\mathtt{lam}\ \$\$_{\mathrm{O}}\ T\ \$\$_{\mathrm{O}}\ E)))$

Types eliminate HYBRID well-formedness predicates

Case Study: Simply Typed Lambda Calculus

Object level `lam`-functions are values:

(ii) Canonical HybridLF

$\mathtt{PI}\ (\mathtt{tm} \Rightarrow \mathtt{tm})\ (\Lambda\ E.\ \mathtt{PI}\ \mathtt{tp}\ (\Lambda\ T.\ \mathtt{val}\ \$\$_{\mathrm{F}}\ (\mathtt{lam}\ \$\$_{\mathrm{O}}\ T\ \$\$_{\mathrm{O}}\ E)))$

(iii) which equals

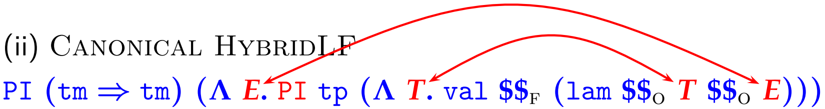
$\boxed{\mathtt{ctype\_bind2}}\ (\mathtt{tm} \Rightarrow \mathtt{tm})$

$(\Lambda\ E.\ \mathtt{tp})$

$(\boxed{\Lambda\ E.\ \Lambda T.\ \ \mathtt{val}\ \$\$_{\mathrm{F}}\ \ (\mathtt{lam}\ \$\$_{\mathrm{O}}\ T\ \$\$_{\mathrm{O}}\ E\ )})$

Case Study: Simply Typed Lambda Calculus

Object level `lam`-functions are values:



(ii) Canonical HybridLF

$\texttt{PI}\ (\texttt{tm} \Rightarrow \texttt{tm})\ (\Lambda\, E.\ \texttt{PI}\ \texttt{tp}\ (\Lambda\, T.\ \texttt{val}\ \$\$_{\mathrm{F}}\ (\texttt{lam}\ \$\$_{\mathrm{O}}\ T\ \$\$_{\mathrm{O}}\ E)))$

(iii) which equals

$\boxed{\texttt{ctype\_bind2}}\ (\texttt{tm} \Rightarrow \texttt{tm})$

$(\Lambda\, E.\ \texttt{tp})$

$(\boxed{\Lambda\, E.\ \Lambda T.\ \texttt{val}\ \$\$_{\mathrm{F}}\ (\texttt{lam}\ \$\$_{\mathrm{O}}\ T\ \$\$_{\mathrm{O}}\ E\ )})$

(iv) which evaluates to

$\texttt{PI}\ (\texttt{tm} \Rightarrow \texttt{tm})\ (\texttt{PI}\ \texttt{tp}\ (\texttt{val}\ \$\$_{\mathrm{F}}\ ((\texttt{lam}\ \$\$_{\mathrm{O}}\ (\texttt{BND}\ \mathbf{0}))\ \$\$_{\mathrm{O}}\ (\texttt{BND}\ \mathbf{1}))))$

Case Study: Simply Typed Lambda Calculus

Object level `lam`-functions are values:

(ii) Canonical HybridLF

$$\texttt{PI}\ (\texttt{tm} \Rightarrow \texttt{tm})\ (\boldsymbol{\Lambda}\ E.\ \texttt{PI}\ \texttt{tp}\ (\boldsymbol{\Lambda}\ T.\ \texttt{val}\ \$\$_{\mathrm{F}}\ (\texttt{lam}\ \$\$_{\mathrm{O}}\ T\ \$\$_{\mathrm{O}}\ E)))$$

(iii) which equals

$$\boxed{\texttt{ctype\_bind2}}\ (\texttt{tm} \Rightarrow \texttt{tm})$$

$$(\boldsymbol{\Lambda}\ E.\ \texttt{tp})$$

$$(\boxed{\boldsymbol{\Lambda}\ E.\ \boldsymbol{\Lambda} T.\ \texttt{val}\ \$\$_{\mathrm{F}}\ (\texttt{lam}\ \$\$_{\mathrm{O}}\ T\ \$\$_{\mathrm{O}}\ E\ )})$$

(iv) which evaluates to

$$\texttt{PI}\ (\texttt{tm} \Rightarrow \texttt{tm})\ (\texttt{PI}\ \texttt{tp}\ (\texttt{val}\ \$\$_{\mathrm{F}}\ ((\texttt{lam}\ \$\$_{\mathrm{O}}\ (\texttt{BND}\ \mathbf{0}))\ \$\$_{\mathrm{O}}\ (\texttt{BND}\ \mathbf{1}))))$$

Hybrid syntax

Conclusions

- Can the techniques of HYBRID be migrated to a dependently typed setting? Yes.
- The type system replaces well-formedness predicates of HYBRID (eg **isTerm** E and **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.