Twelf and Delphin

Logic and Functional Programming in a Meta-Logical Framework

Carsten Schürmann

Yale University

Advertisement!

- Twelf Tutorial.
- July 5, 2004.
- IJCAR.
- Cork, Ireland.

What are Logical Frameworks?

We can look at the current field of problem solving by computer as a series of ideas about how to represent a problem. If a problem can be cast into one of these representations in a natural way, then it is possible to manipulate it and stand some chance of solving it.

Allen Newell

What are Logical Frameworks?

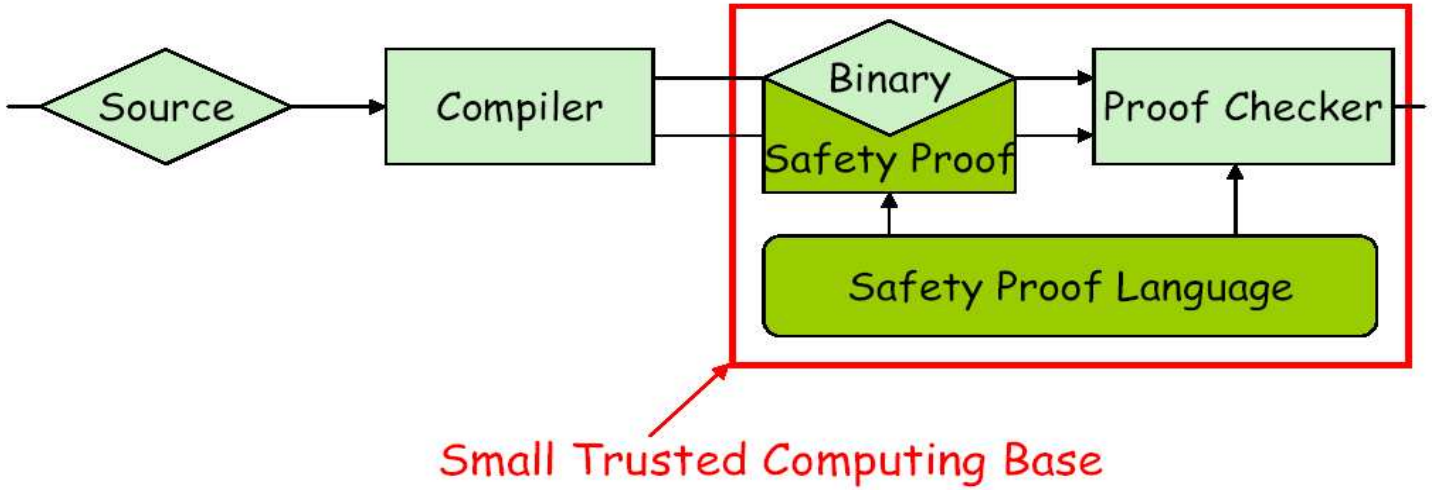
Meta-languages.

- Representation of problem domains.
- Elegance.
- Expressive.
- Beautiful.

Sound philosophical foundation.

Logically motivated.

Programming Languages



- Proof Carrying Code. [Necula, Lee]
- Typed assembly language. [Crary, et al.]

Running Example

Programmers: Think combinators!
Logicians: Think Hilbert calculus!

- Formulas: $A ::= P \mid A \supset B$
- Judgment: $\vdash A$

$$\frac{}{\vdash A \supset B \supset A}\,\mathsf{K} \qquad \frac{\vdash A \supset B \quad \vdash A}{\vdash B}\,\mathsf{MP}$$

$$\frac{}{\vdash (A \supset B \supset C) \supset (A \supset B) \supset (A \supset C)}\,\mathsf{S}$$

Other examples

Safety languages and safety proofs.

- First-order/higher-order logics.
- Temporal, modal, linear logics.

Domain specific languages.

- High-level, low-level.
- Operational, static, reduction, small-step, big-step semantics.
- Typed intermediate languages, compilers.

Sample Logical Frameworks

- Hereditary Harrop formulas.
  Isabelle, λProlog
- λ^{Π} (LF). Automath, LF, Elf, Twelf
- Substructural logical frameworks.
  Forum, LLF, OLF
- Equational logic, rewriting. Maude, ELAN
- Constructive type theories.
  ALF, Agda, Coq, LEGO, Nuprl

What can go wrong?

1. Logics may be inconsistent!
2. Logics may be incompatible!
3. Type systems may be unsound!
4. Loss of representational abstraction in implementations!
5. Maintenance of inference rules!

We need to tools to engineer, experiment, reason, and program with our encodings!

Meta-logical frameworks

- Reasoning *about* deductive systems.
- Experimenting *with* deductive systems.
- Programming *with* deductive systems.

Meta-logical frameworks

1. If $A \vdash B$ then $\vdash A \supset B$.
2. If $\Gamma \vdash_{1} e : \tau$ then $[\Gamma] \vdash_{2} [e] : [\tau]$.
3. If $e \Longrightarrow e_1$ and $e \Longrightarrow e_2$ then there exists a e', such that $e_1 \Longrightarrow e'$ and $e_2 \Longrightarrow e'$.
4. Write a theorem prover

$$prove : \forall A : o. \Box(\vdash A)$$

5. Write a cut-elimination procedure.

Sample Meta-Logical Frameworks

- Finitary inductive definitions. FS_0
- Definitional reflection. $\mathrm{FOL}^{\Delta \mathbb{N}}$
- Higher-level judgments, regular worlds. Twelf
- Other systems used as meta-logical frameworks.
  - Constructive type theories — Agda, Coq, LEGO, Nuprl
  - Higher-order logic — HOL, Isabelle/HOL
  - Rewriting logic — Maude

Outline of this talk

- The logical framework LF.
- Logic programming in Elf.
- Meta theory of deductive systems in Twelf.
- Functional programming in Delphin.
- Conclusion.

The Logical Framework LF

The Logical Framework LF

- λ^{Π} [Harper, Honsell, Plotkin]
- Edinburgh Logical Framework.

$$
\begin{array}{lcl}
K & ::= & \text{type} \mid \Pi x : A.\, K \mid A \rightarrow K \\
A & ::= & a \mid A\ M \mid \Pi x : A_1.\, A_2 \mid A_1 \rightarrow A_2 \\
M & ::= & c \mid \lambda x : A.\, M \mid M_1\ M_2
\end{array}
$$

- Dependently-typed λ-calculus.
- Signature: declares $c : A$ and $a : K$.

The Logical Framework LF

Representation paradigm.

- Judgments-as-types.

$$\ulcorner \vdash A \urcorner : \text{type} = \text{hil}\ \ulcorner A \urcorner$$

- Derivations-as-objects.

$$\ulcorner \dfrac{\begin{array}{cc} \mathcal{H}_1 & \mathcal{H}_2 \\ \vdash A \supset B & \vdash A \end{array}}{\vdash B}\,\text{MP} \urcorner \quad \begin{array}{ll} : & \text{hil}\ \ulcorner B \urcorner \\ = & \text{MP}\ \ulcorner A \urcorner\ \ulcorner B \urcorner\ \ulcorner \mathcal{H}_1 \urcorner\ \ulcorner \mathcal{H}_2 \urcorner \end{array}$$

Deduction theorem

The Hilbert calculus in LF/Twelf.

$$
\begin{array}{lll}
\mathsf{o} & : & \text{type}. \\
\mathsf{imp} & : & \mathsf{o} \longrightarrow \mathsf{o} \longrightarrow \mathsf{o}. \\
 & & \\
\mathsf{hil} & : & \mathsf{o} \longrightarrow \text{type}. \\
\mathsf{K} & : & \mathsf{hil}\ (\mathsf{imp}\ A\ (\mathsf{imp}\ B\ A)). \\
\mathsf{S} & : & \mathsf{hil}\ (\mathsf{imp}\ (\mathsf{imp}\ A\ (\mathsf{imp}\ B\ C)) \\
 & & \quad (\mathsf{imp}\ (\mathsf{imp}\ A\ B)\ (\mathsf{imp}\ A\ C))). \\
\mathsf{MP} & : & \mathsf{hil}\ (\mathsf{imp}\ A\ B) \longrightarrow \mathsf{hil}\ A \longrightarrow \mathsf{hil}\ B.
\end{array}
$$

The Logical Framework LF (cont'd)

Hypothetical judgments.

$$\begin{array}{rcl} \ulcorner \begin{array}{c} \overline{\vdash A}\,u \\ \mathcal{H} \\ \vdash B \end{array} \urcorner & : & \mathsf{hil}\ \ulcorner A \urcorner \rightarrow \mathsf{hil}\ \ulcorner B \urcorner \\ & = & \lambda u : \mathsf{hil}\ \ulcorner A \urcorner . \ulcorner \mathcal{H} \urcorner \end{array}$$

LF function types *encode*

- inference rules,
- hypothetical judgments.

The Logical Framework LF (cont'd)

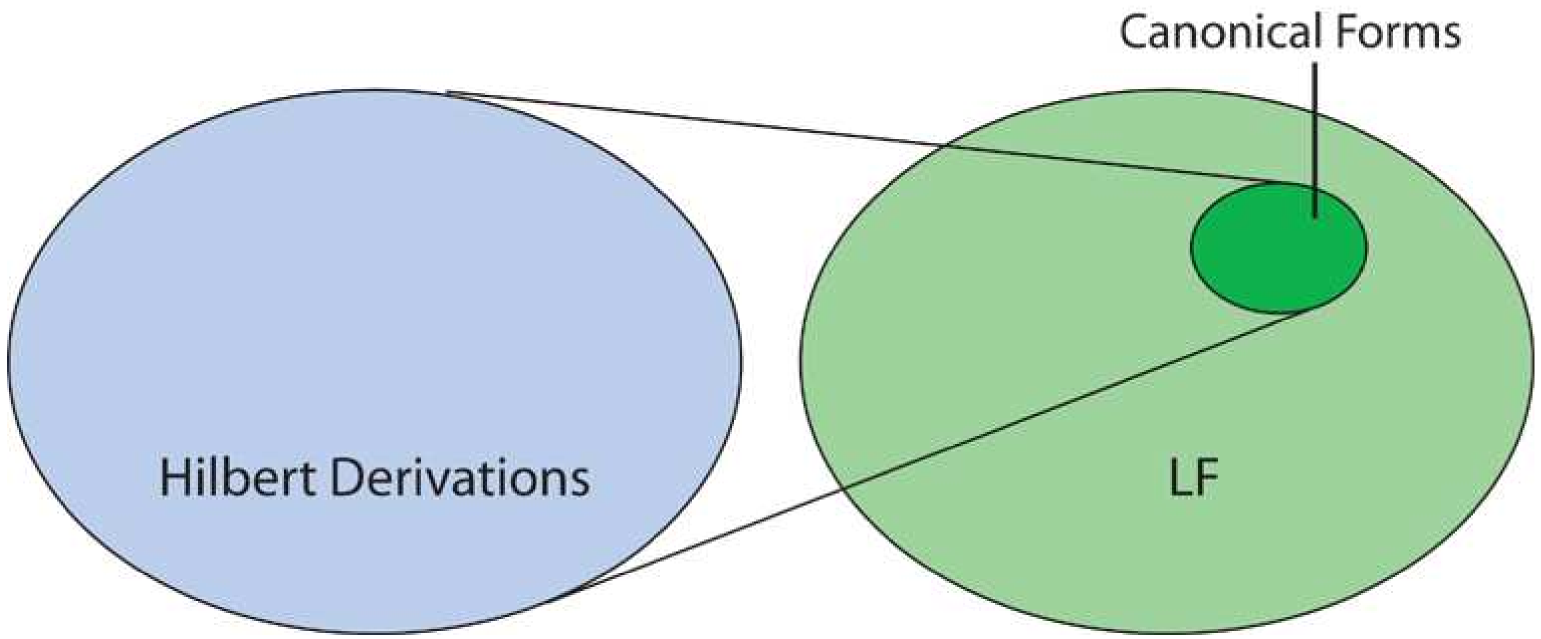
Definitional equality.

- LF terms are alive.
- $(\lambda x : A.\, M)N \equiv [N/x]M$ $\qquad (\beta)$
- $(\lambda x : A.\, Mx) \equiv M$ $\qquad (\eta)$
- Canonical forms: β-normal, η-long form.
- Object language contexts/environments disappear.

Theorem: Every well-typed object in LF reduces to a unique canonical form.

The Logical Framework LF (cont'd)

Theorem: [Adequacy] There exists a bijection between $\mathcal{H} :: A_1 \ldots A_n \vdash A$ and $u_1 : \ulcorner A_1 \urcorner, \ldots, u_n : \ulcorner A_n \urcorner \vdash \ulcorner \mathcal{H} \urcorner \Uparrow \ulcorner A \urcorner$.



The Logical Framework LF (cont'd)

Parametric Function Space.

- Example: $\mathsf{ded}\ A \rightarrow \mathsf{ded}\ B$

$$\lambda x : \mathsf{ded}\ A.\ x$$
$$\lambda x : \mathsf{ded}\ A.\ \mathsf{K}$$
$$\lambda x : \mathsf{ded}\ A.\ \mathsf{S}$$
$$\lambda x : \mathsf{ded}\ A.\ \mathsf{MP}\ (H_1\ x)\ (H_2\ x)$$

- Parametric functions are good for *representation* but not *programming*.

The Logical Framework LF (cont’d)

Summary.

- Adequate higher-order encodings.
- Encodings necessarily non-inductive.
- Rapid prototyping of deductive systems.

Computational weakness

$\approx$

Representational strength

Elf: Logic programming in LF

Elf: Logic programming in LF

[Pfenning 89]

- Overcoming the computational weakness.
- Strict separation data and programs ...
  ... using the same syntax.
- Idea: Don’t just use β for computation.
- Instead: Search for canonical forms.

Elf: Logic programming in LF

$$\frac{}{\vdash A \supset B \supset A}\,\mathsf{K} \qquad \frac{\vdash (A \supset B) \quad \vdash A}{\vdash B}\,\mathsf{MP}$$

$$\frac{}{\vdash (A \supset B \supset C) \supset (A \supset B) \supset (A \supset C)}\,\mathsf{S}$$

Challenge: Give a derivation of the identity.

$$\vdash A \supset A$$

Elf: Logic programming in LF

By S with A/C and $(A \supset B) \supset A/B$.

$$\vdash \quad (A \supset ((A \supset B) \supset A) \supset A) \supset (A \supset (A \supset B) \supset A) \supset (A \supset A)$$

By K with $(A \supset B) \supset A/B$ and $(A \supset B)/B$.

$$\vdash A \supset ((A \supset B) \supset A) \supset A$$
$$\vdash A \supset (A \supset B) \supset A$$

By two applications of MP: $\vdash A \supset A$. $\square$

Elf: Logic programming in LF

Search for canonical forms.

$$
\begin{array}{llllll}
? & & & & & : \mathsf{ded}\ (A \supset A) \\
\mathsf{MP} & ? & & & ? & \\
\mathsf{MP} & (\mathsf{MP} & ? & ?) & K & \\
\mathsf{MP} & (\mathsf{MP} & S & K) & K &
\end{array}
$$

- Model of computation: search.
- Signature: logic program.
- Here: search space infinite.

Elf: Logic programming in LF

Programmers: Think bracket abstraction!

Logicians: Think deduction theorem!

Programming Exercise: [Gentzen]

Convert a “hypothetical combinator” of type

$$\begin{array}{c} \overline{\vdash A}\,u \\ \vdots \\ \vdash B \end{array}$$

into a combinator of type $\vdash A \supset B$.

Elf: Logic programming in LF

Representation in LF/Twelf

$$
\begin{array}{l}
\text{ded} : (\text{hil}\ A \rightarrow \text{hil}\ B) \rightarrow \text{hil}\ (A\ \text{imp}\ B) \rightarrow \text{type}.\\
\text{ded\_id} : \text{ded}\ (\lambda u{:}\text{hil}\ A.\ u)\ (\text{MP}\ (\text{MP}\ \text{S}\ \text{K})\ \text{K}).\\
\text{ded\_K} : \text{ded}\ (\lambda u{:}\text{hil}\ A.\ \text{K})\ (\text{MP}\ \text{K}\ \text{K}).\\
\text{ded\_S} : \text{ded}\ (\lambda u{:}\text{hil}\ A.\ \text{S})\ (\text{MP}\ \text{K}\ \text{S}).\\
\text{ded\_MP} : \text{ded}\ (\lambda u{:}\text{hil}\ A.\ \text{MP}\ (H_1\ u)\ (H_2\ u))\\
\qquad\qquad (\text{MP}\ (\text{MP}\ \text{S}\ H_1')\ H_2')\\
\qquad\qquad \leftarrow \text{ded}\ (\lambda u{:}\text{hil}\ A.\ H_1\ u)\ H_1'\\
\qquad\qquad \leftarrow \text{ded}\ (\lambda u{:}\text{hil}\ A.\ H_2\ u)\ H_2'.
\end{array}
$$

Elf: Logic programming in LF

The two function spaces.

$c : A \to B$ is for *representation*.

$c(x) = M$ (Reduction to $\beta\eta$-canonical form).

$f : A \Rightarrow B$ is for *programming*.

$f(x) = M$ if and only if $\exists D : f \; x \; M$.

Elf: Logic programming in LF

Operational interpretation. [Pfenning]

$$
\begin{array}{rcl}
G & ::= & P \mid \Pi x : A.\, G \mid D \to G \\
D & ::= & P \mid \Pi x : A.\, D \mid G \to D
\end{array}
$$

$$
P \;::=\; a \mid P\ M
$$

- "$\longrightarrow$" triggers search, "Π" does not. [Pym]
- $x : A$ existential variable.
- $x : A$ parameter.

Elf: Logic programming in LF

- Existential variables.
- Back-tracking.
- Embedded implications.

$+$ Works with higher-order encodings.

$+$ Same syntax as LF signatures.

$-$ No user control on search.

$*$ No extra logical constants.

Elf: Logic programming in LF

Applications.

- Programming language design.
  Type systems.
  Operational semantics.
  Compilation.
- Logics.
  Transformations. [Logosphere]
  Cut-elimination. [Pfenning]

Meta theory of deductive systems

Meta Theory

Programmers: Think λ-calculus!
Logicians: Think natural deductions!

$$\frac{\Gamma, A \Vdash B}{\Gamma \Vdash A \supset B}\,\text{lam} \qquad \frac{\Gamma \Vdash A \supset B \quad \Gamma \Vdash A}{\Gamma \Vdash B}\,\text{app}$$

nd : o $\longrightarrow$ type.
lam : (nd A $\longrightarrow$ nd B) $\longrightarrow$ nd (A imp B).
app : nd (A imp B) $\longrightarrow$ nd A $\longrightarrow$ nd B.

Meta Theory

Theorem: [Natural Deduction - Hilbert]

For all $\overset{\mathcal{D}}{\Gamma \Vdash A}$ there exists a derivation $\overset{\mathcal{H}}{\Gamma \vdash A}$.

- Realizability interpretation.

$$\text{ndhil} \;:\; \Pi A{:}\texttt{o}.\ \text{nd}\ A \longrightarrow \text{hil}\ A \longrightarrow \text{type}$$

- *Total* logic programs encode meta proofs.

Meta Theory

Lemma [Deduction]

$$\text{If } \overset{\mathcal{H}}{\Gamma, A \vdash B} \text{ then } \overset{\mathcal{H}'}{\Gamma \vdash A \supset B} .$$

Proof: by structural induction on $\mathcal{H}$.

Cases K, S, MP same as above.

Case: $B \in \Gamma$

$\mathcal{H}_1 :: \Gamma \vdash B \supset A \supset B$ by K

$\mathcal{H}' :: \Gamma \vdash A \supset B$ by MP

Meta Theory

Proof (of ndhil): by structural induction on $\mathcal{D}$.

$$\text{Case: } \mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}_1\\ \Gamma, A \Vdash B\end{array}}{\Gamma \Vdash A \supset B}\ \mathsf{lam}$$

$\mathcal{H}_1 :: \Gamma, A \vdash B$ by induction hypothesis
$\mathcal{H} :: \Gamma \vdash (A \supset B)$ by deduction lemma.

Case app straightforward.

Meta Theory

$$
\begin{array}{l}
\mathsf{caselam}\texttt{:}\ \mathsf{ndhil}\ \texttt{(}\mathsf{lam}\ \texttt{(}\lambda u\texttt{:}\mathsf{nd}\ A\texttt{.}\ D_1\ u\texttt{))}\ H \\
\qquad\qquad \leftarrow\ \texttt{(}\Pi u\texttt{:}\mathsf{nd}\ A\texttt{.}\ \Pi h\texttt{:}\mathsf{hil}\ A\texttt{.} \\
\qquad\qquad\qquad\qquad \mathsf{ndhil}\ u\ h\ \rightarrow \\
\qquad\qquad\qquad\qquad \texttt{(}\Pi B\texttt{:}\mathsf{o}\texttt{.}\ \mathsf{ded}\ \texttt{(}\lambda z\texttt{:}\mathsf{hil}\ B\texttt{.}\ h\texttt{)} \\
\qquad\qquad\qquad\qquad\qquad\qquad \texttt{(}\mathsf{MP}\ \mathsf{K}\ h\texttt{))} \\
\qquad\qquad\qquad\qquad \rightarrow\ \mathsf{ndhil}\ \texttt{(}D_1\ u\texttt{)}\ \texttt{(}H_1\ h\texttt{))} \\
\qquad\qquad \leftarrow\ \mathsf{ded}\ \texttt{(}\lambda h\texttt{:}\mathsf{hil}\ A\texttt{.}\ H_1\ h\texttt{)}\ H\texttt{.} \\
\mathsf{caseapp}\texttt{:}\ \mathsf{ndhil}\ \texttt{(}\mathsf{app}\ D_1\ D_2\texttt{)}\ \texttt{(}\mathsf{MP}\ H_1\ H_2\texttt{)} \\
\qquad\qquad \leftarrow\ \mathsf{ndhil}\ D_1\ H_1 \\
\qquad\qquad \leftarrow\ \mathsf{ndhil}\ D_2\ H_2\texttt{.}
\end{array}
$$

What makes a proof a proof?

Option 1: Propositions-as-types.

$$\forall A. \forall D. \forall \Gamma. \mathsf{isctx}(\Gamma) \wedge \mathsf{wff}(A) \wedge \mathsf{nd}(D, \Gamma, A) \supset \exists H. \mathsf{hil}(H, \Gamma, A)$$

- Logical derivations.
- Inductive types.
- Predominantly used technique. [Coq, . . .]
- Incompatible with higher-order encodings.
- Explicit notion of equality.
- Disprove impossible cases.

What makes a proof a proof?

Option 2: Jugments-as-types.

- Total logic programs *are* proofs.
- Induction on canonical form derivations.
- Non-standard induction principles exist.
- Impossible cases omitted.
- Adequacy replaces validity propositions.
- But: Need to decide totality!

Meta Theory

1. Mode criterion.
   (Fixed input/output behavior of arguments)
2. World criterion.
   (Form of the local context is regular)
3. Termination criterion.
   (Does not run on forever)
4. Coverage criterion.
   (Covers all cases)

Meta Theory

Definition: [*Mode criterion*] During execution, ground inputs are being mapped onto output ground outputs.

[Rohwedder, Pfenning]

%mode (ded $+H$ $-H'$).
%mode (ndhil $+D$ $-H$).

Meta Theory (Mode Criterion)

$$
\begin{array}{l}
\mathsf{caselam}\texttt{:}\ \ \mathsf{ndhil}\ \boxed{\texttt{(}\mathsf{lam}\ \texttt{(}\lambda u\texttt{:}\mathsf{nd}\ A\texttt{.}\ D_1\ u\texttt{))}}\ H \\
\qquad \leftarrow\ \texttt{(}\Pi u\texttt{:}\mathsf{nd}\ A\texttt{.}\ \Pi h\texttt{:}\mathsf{hil}\ A\texttt{.} \\
\qquad\qquad \mathsf{ndhil}\ \boxed{u}\ h\ \rightarrow \\
\qquad\qquad \texttt{(}\Pi B\texttt{:}\mathsf{o}\texttt{.}\ \mathsf{ded}\ \boxed{\texttt{(}\lambda z\texttt{:}\mathsf{hil}\ B\texttt{.}\ h\texttt{)}} \\
\qquad\qquad\qquad \texttt{(}\mathsf{MP}\ \mathsf{K}\ h\texttt{))} \\
\qquad\quad \rightarrow\ \mathsf{ndhil}\ \texttt{(}D_1\ u\texttt{)}\ \boxed{\texttt{(}H_1\ h\texttt{))}} \\
\qquad \leftarrow\ \mathsf{ded}\ \texttt{(}\lambda h\texttt{:}\mathsf{hil}\ A\texttt{.}\ H_1\ h\texttt{)}\ \boxed{H}\texttt{.} \\
\mathsf{caseapp}\texttt{:}\ \ \mathsf{ndhil}\ \boxed{\texttt{(}\mathsf{app}\ D_1\ D_2\texttt{)}}\ \texttt{(}\mathsf{MP}\ H_1\ H_2\texttt{)} \\
\qquad \leftarrow\ \mathsf{ndhil}\ D_1\ \boxed{H_1} \\
\qquad \leftarrow\ \mathsf{ndhil}\ D_2\ \boxed{H_2}\texttt{.}
\end{array}
$$

Meta Theory

Definition: [*World criterion*] During execution the local context is always regular formed.

[Schürmann]

$\texttt{\%world dyn}\ [A\texttt{:o}]$

$\quad\{\texttt{u:nd}\ A,$

$\quad\ \texttt{h:hil}\ A,$

$\quad\ \texttt{p:}(\Pi B\texttt{:o. ded}\ (\lambda z\texttt{:hil}\ B.\ h)\ (\texttt{MP K}\ h))$

$\quad\ \texttt{d:ndhil}\ u\ v,$

$\quad\}$

Meta Theory (World criterion)

$$
\begin{array}{l}
\texttt{caselam}: \ \texttt{ndhil}\ (\texttt{lam}\ (\lambda u{:}\texttt{nd}\ A.\ D_1\ u))\ H \\
\qquad \leftarrow \boxed{(\Pi u{:}\texttt{nd}\ A.\ \Pi h{:}\texttt{hil}\ A.} \\
\qquad\qquad \boxed{\texttt{ndhil}\ u\ h \rightarrow} \\
\qquad\qquad \boxed{(\Pi B{:}\texttt{o}.\ \texttt{ded}\ (\lambda z{:}\texttt{hil}\ B.\ h)} \\
\qquad\qquad\qquad \boxed{(\texttt{MP}\ \texttt{K}\ h))} \\
\qquad\qquad \rightarrow \texttt{ndhil}\ (D_1\ u)\ (H_1\ h)) \\
\qquad \leftarrow \texttt{ded}\ (\lambda h{:}\texttt{hil}\ A.\ H_1\ h)\ H. \\
\texttt{caseapp}: \ \texttt{ndhil}\ (\texttt{app}\ D_1\ D_2)\ (\texttt{MP}\ H_1\ H_2) \\
\qquad \leftarrow \texttt{ndhil}\ D_1\ H_1 \\
\qquad \leftarrow \texttt{ndhil}\ D_2\ H_2.
\end{array}
$$

Meta Theory

Definition: [*Termination criterion*] The execution will eventually terminate.

[Rohwedder, Pfenning, Pientka]

- In general undecidable.
- Well-founded subterm ordering.
- Lexicographic and simultaneous extensions.

%terminates H (ded $+H$ $-H'$).
%terminates D (ndhil $+D$ $-H$).

Meta Theory (Termination criterion)

$$
\begin{array}{l}
\mathsf{caselam}\texttt{:}\ \ \mathsf{ndhil}\ \boxed{\texttt{(}\mathsf{lam}\ \texttt{(}\lambda u\texttt{:}\mathsf{nd}\ A\texttt{.}\ D_1\ u\texttt{))}}\ H \\
\qquad \leftarrow\ \texttt{(}\Pi u\texttt{:}\mathsf{nd}\ A\texttt{.}\ \Pi h\texttt{:}\mathsf{hil}\ A\texttt{.} \\
\qquad\qquad\qquad \mathsf{ndhil}\ u\ h\ \rightarrow \\
\qquad\qquad\qquad \texttt{(}\Pi B\texttt{:}\mathsf{o}\texttt{.}\ \mathsf{ded}\ \texttt{(}\lambda z\texttt{:}\mathsf{hil}\ B\texttt{.}\ h\texttt{)} \\
\qquad\qquad\qquad\qquad \texttt{(}\mathsf{MP}\ \mathsf{K}\ h\texttt{))} \\
\qquad\qquad \rightarrow\ \mathsf{ndhil}\ \boxed{\texttt{(}D_1\ u\texttt{)}}\ \texttt{(}H_1\ h\texttt{))} \\
\qquad \leftarrow\ \mathsf{ded}\ \texttt{(}\lambda h\texttt{:}\mathsf{hil}\ A\texttt{.}\ H_1\ h\texttt{)}\ H\texttt{.} \\
\mathsf{caseapp}\texttt{:}\ \ \mathsf{ndhil}\ \boxed{\texttt{(}\mathsf{app}\ D_1\ D_2\texttt{)}}\ \texttt{(}\mathsf{MP}\ H_1\ H_2\texttt{)} \\
\qquad \leftarrow\ \mathsf{ndhil}\ \boxed{D_1}\ H_1 \\
\qquad \leftarrow\ \mathsf{ndhil}\ \boxed{D_2}\ H_2\texttt{.}
\end{array}
$$

Meta Theory

Definition: [*Coverage criterion*] The execution will always make progress.

[Schürmann, Pfenning]

- In general undecidable. [Coquand]
- Very difficult but extremely important.
  - Non-local assumptions.
  - Input coverage.
  - Output coverage.
- Open for 10 years.

Meta Theory (Coverage Criterion)

$$
\begin{array}{l}
\mathsf{caselam}:\ \mathsf{ndhil}\ \boxed{(\mathsf{lam}\ (\lambda u{:}\mathsf{nd}\ A.\ D_1\ u))}\ H \\
\quad \leftarrow (\Pi u{:}\mathsf{nd}\ A.\ \Pi h{:}\mathsf{hil}\ A. \\
\qquad \mathsf{ndhil}\ \boxed{u}\ h \rightarrow \\
\qquad (\Pi B{:}\mathsf{o}.\ \mathsf{ded}\ (\lambda z{:}\mathsf{hil}\ B.\ h) \\
\qquad\qquad (\mathsf{MP}\ \mathsf{K}\ h)) \\
\qquad \rightarrow \mathsf{ndhil}\ (D_1\ u)\ \boxed{(H_1\ h)}) \\
\quad \leftarrow \mathsf{ded}\ (\lambda h{:}\mathsf{hil}\ A.\ H_1\ h)\ H. \\
\mathsf{caseapp}:\ \mathsf{ndhil}\ \boxed{(\mathsf{app}\ D_1\ D_2)}\ (\mathsf{MP}\ H_1\ H_2) \\
\quad \leftarrow \mathsf{ndhil}\ D_1\ \boxed{H_1} \\
\quad \leftarrow \mathsf{ndhil}\ D_2\ \boxed{H_2}.
\end{array}
$$

Functional Programming

Functional programming in LF

Delphin

- $\Box A$ embeds LF types in Delphin.
- box M embeds LF objects in Delphin.
- λ-calculus with recursion and case.
- Strict separation of LF and meta level.
- Parametric function space.
- Primitive recursive function space.
- Automated theorem prover.

Functional programming in LF

Advantages.

- No existential variables.
- Back-tracking.
- Higher-order encodings.
- Computation under λ-binders.
- Local let statements.

Applications.

- Coverage checking (order ≥ 3).
- Compilation of mode-correct programs.

Functional programming in LF

The two function spaces.

$c : A \to B$ is for *representation*.

$c(x) = M$ (Reduction to $\beta\eta$-canonical form)

$f : \Box A \Rightarrow \Box B$ is for *programming*.

$f(x) = M$ (Function definition by cases).

Functional programming in LF

Basic idea: Use worlds to describe datatypes.

```
%world static
   {imp : o → o → o,
    K   : hil (imp A (imp B A)),
    S   : hil (imp (imp A (imp B C))
    MP  : hil (imp A B) → hil A → hil B}

%world dynamic [A:o]
   {y : hil A }
```

Functional programming in LF

$$
\begin{aligned}
\mu f.\quad & (\nabla s:\mathsf{static}.\nabla d:\mathsf{dynamic}.\exists A:\mathsf{o}.\mathsf{box}\ (\lambda u:\mathsf{hil}\ A.d.\mathsf{y}) \\
& \quad \mapsto \mathsf{box}\ (s.\mathsf{MP}\ s.\mathsf{K}\ d.\mathsf{y})) \\
\mid\ & (\nabla s:\mathsf{static}.\exists A:\mathsf{o}.\mathsf{box}\ (\lambda u:\mathsf{hil}\ A.u) \\
& \quad \mapsto \mathsf{box}\ (s.\mathsf{MP}\ (s.\mathsf{MP}\ s.\mathsf{S}\ s.\mathsf{K})\ s.\mathsf{K})) \\
\mid\ & (\nabla s:\mathsf{static}.\exists A:\mathsf{o}.\exists B:\mathsf{o}.\exists C:\mathsf{o}.\exists H_2:\mathsf{hil}\ A \to \mathsf{hil}\ B. \\
& \quad \exists H_1:\mathsf{hil}\ A \to \mathsf{hil}\ (B \to C). \\
& \quad\quad \mathsf{box}\ (\lambda u:\mathsf{hil}\ A.s.\mathsf{MP}\ (H_1\ u)\ (H_2\ u)) \\
& \quad \mapsto\ (\mathsf{box}\ (s.\mathsf{MP}))[\cdot](\mathsf{box}\ (s.\mathsf{MP}) \\
& \quad\quad\quad [\cdot](\mathsf{box}\ (s.\mathsf{S}))[\cdot](f\ (\mathsf{box}\ (\lambda u:\mathsf{hil}\ A.H_1\ u)))) \\
& \quad\quad\quad [\cdot](f\ (\mathsf{box}\ (\lambda u:\mathsf{hil}\ A.H_2\ u)))) \\
\vdots &
\end{aligned}
$$

Functional programming in LF

Meta(-meta) Theory.

Type soundness. Operational semantics is type preserving.

Conversion lemma. Let $a : A \rightarrow B \rightarrow type$, *mode correct*. Then there exists a Delphin function $f_a : \Box A \Rightarrow \Box B$, such that

$$\text{If } \exists D : a\ M\ N \text{ then } f_a(\text{box } M) = \text{box } N.$$

Implementation

Implementation

Twelf. `www.twelf.org`

- Type reconstruction.
- Logic programming.
- Mode, world, termination, coverage.

Delphin.
`www.cs.yale.edu/~carsten/delphin`

- Prototype exists.
- Functional programming.
- Converter from Elf logic programs.
- Factoring. [Poswolsky]

Conclusion

Meta-logical framework Twelf and Delphin.

- Lots of applications.
- Automated deduction.
- Great rapid prototyping tool.
- Programming with variable binders.
- Supports representational strength.
- Provides computational power.