Outline for 2nd Talk

1. Recap 1st talk
2. Logic Programming Interpretation
   (Implementation of LF)
3. Hypothetical & Parametric Reasoning
   (Mini-ML type checking and proof of type-preservation)
4. Example Representations
   - Compilation of Mini-ML
   - Intuitionistic Sequent Calculus and Cut-Elimination
5. Extensions To Basic LF Type Theory
   - Support for Integers & Minimal Equational Reasoning
   - Meta Logic for Automated Proof Search

Summary of 1st Talk

- Logical Frameworks are meta languages for representing and reasoning about deductive systems.
- LF is typed lamda calculus with dependent types.
- LF type-checking is decidable and all well-typed LF terms can be converted to canonical form (β-reduced, η-long).
- LF representation methodology:
  - Higher-Order Abstract Syntax
    (meta variables represent object variables)
  - Judgements as Types
    (derivation rules are terms)

Logic Programming Interpretation of Type-Checking

When focussing on types,
action of Π same as $\forall$ and
action of $\rightarrow$ (restricted use of Π) same as $\supset$.

$$\frac{\Gamma \vdash^{\mathrm{LF}}_{\Sigma} A \Uparrow \mathrm{type} \qquad \Gamma, x{:}A \vdash^{\mathrm{LF}}_{\Sigma} M \Uparrow B}{\Gamma \vdash^{\mathrm{LF}}_{\Sigma} \lambda x{:}A.\ M \Uparrow \Pi x{:}A.\ B}\,\mathsf{canpi}$$

$$\frac{\Gamma \vdash [N/x]B}{\Gamma \vdash \forall x.\ B}\,\forall^{N}_{I} \qquad\qquad \frac{\Gamma, A \vdash B \qquad \Gamma \vdash A \supset B}{\Gamma \vdash A}\,\supset_{I}$$

$$\frac{\Gamma \vdash^{\mathrm{LF}}_{\Sigma} M \downarrow \Pi x{:}A.\ B \qquad \Gamma \vdash^{\mathrm{LF}}_{\Sigma} N \Uparrow A}{\Gamma \vdash^{\mathrm{LF}}_{\Sigma} M\ N \downarrow [N/x]B}\,\mathsf{atmapp}$$

$$\frac{\Gamma \vdash \forall x.\ B}{\Gamma \vdash [N/x]B}\,\forall_{E} \qquad\qquad \frac{\Gamma \vdash A \supset B \qquad \Gamma \vdash A}{\Gamma \vdash B}\,\supset_{E}$$

Canonical type families are logical formulas
generated from atoms, $\supset$ and $\forall$.

Uniform Proof Search

For type-checking at Π type, only canpi rule applies.

Consider type-checking at (canonical) atomic type

$$\cdot \vdash^{\mathrm{LF}}_{\Sigma} M \Uparrow \mathrm{eval}\ (\mathrm{s}\ E)\ V$$

M must be application of $\mathsf{ev\_s}$.
No other constructor has correct result type.

Then

$$M = \mathrm{ev\_s}\ E\ V\ M'$$

where

$$\cdot \vdash^{\mathrm{LF}}_{\Sigma} M' \Uparrow \mathrm{eval}\ E\ V$$

A term for any atomic type defined in Σ, must be the application of a constructor in Σ.

Note a term at type A is a proof of formula A.

Algorithm for Type-checking

For Π types, add fresh variable of appropriate type to context.

For atomic types, search through signature for constant with matching result type, then type-check arguments.

Reverse direction of arrow to emphasize that "head" of clause is what will be matched.

$$
\begin{array}{lll}
\text{ev\_app} & : & \Pi E_1{:}\text{exp}.\ \Pi E_2{:}\text{exp}.\ \Pi E_1'{:}\text{exp} \to \text{exp}.\ \Pi V_2{:}\text{exp}.\ \Pi V{:}\text{exp} \\
& & \text{eval (app } E_1\ E_2)\ V \\
& & \quad \leftarrow \text{eval } E_1\ (\text{lam } E_1') \\
& & \quad \leftarrow \text{eval } E_2\ V_2 \\
& & \quad \leftarrow \text{eval } (E_1'\ V_2)\ V
\end{array}
$$

Can handle logic variables using unification (non-trivial extensions of normal higher-order unification).

Under this interpretation, specifications become executable logic programs.

Type Reconstruction

Most Explicit Π bindings are redundant.
Type information implicit in the application of bound term.

$$\text{ev\_s} \ : \ \Pi E{:}\text{exp}.\ \Pi V{:}\text{exp}.\ \text{eval}\ E\ V \to \text{eval}\ (\text{s}\ E)\ (\text{s}\ V)$$

$$\text{ev\_s} \ : \ \text{eval}\ E\ V \to \text{eval}\ (\text{s}\ E)\ (\text{s}\ V)$$

Greatly simplifies presentation of representations.

$$\begin{aligned} \text{ev\_app} \ : \ & \text{eval}\ E_1\ (\text{lam}\ E_1') \to \text{eval}\ E_2\ V_2 \to \text{eval}\ (E_1'\ V_2)\ V \to \\ & \text{eval}\ (\text{app}\ E_1\ E_2)\ V \end{aligned}$$

Type reconstruction algorithm very effective.
Forces user to provide type info. when necessary (rare in practice).

Compilation of Mini-ML

Expressions e ::=

$\underline{\mathrm{z}} \mid \underline{\mathrm{s}}\ e \mid (\underline{\mathrm{case}}\ e_1\ \underline{\mathrm{of}}\ \underline{\mathrm{z}} \to e_2 \mid \underline{\mathrm{s}}\ x \to e_3)$	*Natural numbers*
$\mid \langle e_1, e_2 \rangle \mid \underline{\mathrm{fst}}\ e \mid \underline{\mathrm{snd}}\ e$	*Pairs*
$\mid \underline{\mathrm{lam}}\ x.\ e \mid e_1\ e_2$	*Functions*
$\mid \underline{\mathrm{let}}\ \underline{\mathrm{val}}\ x = e_1\ \underline{\mathrm{in}}\ e_2$	*Definitions*
$\mid \underline{\mathrm{let}}\ \underline{\mathrm{name}}\ x = e_1\ \underline{\mathrm{in}}\ e_2$	
$\mid \underline{\mathrm{fix}}\ u.\ e$	*Recursion*
$\mid u$	*Variables*
$\mid v$	*Values*

Values v ::=

$\underline{\mathrm{z}}^* \mid \underline{\mathrm{s}}^*\ v$	*NaturalNumbers*
$\mid \langle v_1^*, v_2^* \rangle^*$	*Pairs*
$\mid \underline{\mathrm{lam}}^* x.\ e$	*Functions*
$\mid x$	*Variables*

$$
\begin{array}{rrcl}
\textsf{Instructions} & i & ::= & e \mid \mathrm{return}\ v \mid \underline{\mathrm{case}}_1 \mid \langle\rangle_1 \mid \underline{\mathrm{fst}}_1 \mid \underline{\mathrm{snd}}_1 \mid \mathrm{app}_1 \mid \mathrm{app}_2 \\
\textsf{Continuations} & K & ::= & \mathrm{init} \mid K; \lambda x.\ i \\
\textsf{Machine States} & S & ::= & K \diamond I \mid \mathrm{answer}\ v
\end{array}
$$

$$S \Longrightarrow S'$$

S goes to S' in one computation step

$$\begin{array}{rcl}
\textsf{st_z} & :: & K \diamond \underline{\mathrm{z}} \;\Longrightarrow\; K \diamond \mathrm{return}\ \underline{\mathrm{z}}^* \\
\textsf{st_s} & :: & K \diamond (\underline{\mathrm{s}}\ e) \;\Longrightarrow\; (K; \lambda x.\ \mathrm{return}\ (\underline{\mathrm{s}}^*\ x)) \diamond e \\
\textsf{st_case} & :: & K \diamond (\underline{\mathrm{case}} e_1 \underline{\mathrm{of}}\ \underline{\mathrm{z}} \Rightarrow e_2 | \underline{\mathrm{s}}\ x \Rightarrow e_3) \;\Longrightarrow \\
 & & \quad (K; \lambda x_1.\ \underline{\mathrm{case}}_1 x_1 \underline{\mathrm{of}}\ \underline{\mathrm{z}} \Rightarrow e_2 | \underline{\mathrm{s}}\ x \Rightarrow e_3) \diamond e_1 \\
\textsf{st_case1_z} & :: & K \diamond (\underline{\mathrm{case}}_1 \underline{\mathrm{z}}^* \underline{\mathrm{of}}\ \underline{\mathrm{z}} \Rightarrow e_2 | \underline{\mathrm{s}}\ x \Rightarrow e_3) \;\Longrightarrow\; K \diamond e_2 \\
\textsf{st_case1_s} & :: & K \diamond (\underline{\mathrm{case}}_1 (\underline{\mathrm{s}}^*\ v_1) \underline{\mathrm{of}}\ \underline{\mathrm{z}} \Rightarrow e_2 | \underline{\mathrm{s}}\ x \Rightarrow e_3) \;\Longrightarrow\; K \diamond [v_1/x] e_3 \\
\textsf{st_pair} & :: & K \diamond \langle e_1, e_2 \rangle \;\Longrightarrow\; (K; \lambda x.\ \langle x, e_2 \rangle_1) \diamond e_1 \\
\textsf{st_pair}_1 & :: & K \diamond \langle v_1, e_2 \rangle_1 \;\Longrightarrow\; (K; \lambda x.\ \mathrm{return}\ \langle v_1, x \rangle^*) \diamond e_2 \\
\textsf{st_fst} & :: & K \diamond (\underline{\mathrm{fst}}\ e) \;\Longrightarrow\; (K; \lambda x.\ \underline{\mathrm{fst}}_1\ x) \diamond e \\
\textsf{st_fst}_1 & :: & K \diamond (\underline{\mathrm{fst}}_1\ \langle v_1, v_2 \rangle^*) \;\Longrightarrow\; K \diamond \mathrm{return}\ v_1) \\
\textsf{st_snd} & :: & K \diamond (\underline{\mathrm{snd}}\ e) \;\Longrightarrow\; (K; \lambda x.\ \underline{\mathrm{snd}}_1\ x) \diamond e \\
\textsf{st_snd}_1 & :: & K \diamond (\underline{\mathrm{snd}}_1\ \langle v_1, v_2 \rangle^*) \;\Longrightarrow\; K \diamond \mathrm{return}\ v_2) \\
\textsf{st_lam} & :: & K \diamond \underline{\mathrm{lam}}\ x.\ e \;\Longrightarrow\; K \diamond \mathrm{return}\ \underline{\mathrm{lam}}^*\ x.\ e \\
\textsf{st_app} & :: & K \diamond e_1\ e_2 \;\Longrightarrow\; (K; \lambda x.\ \mathrm{app}_1\ x\ e_2) \diamond e_1 \\
\textsf{st_app}_1 & :: & K \diamond \mathrm{app}_1\ (\lambda^* x.\ e)\ e_2 \;\Longrightarrow\; (K; \lambda x_2.\ \mathrm{app}_2\ (\lambda^* x.\ e)\ x_2) \diamond e_2 \\
\textsf{st_app}_2 & :: & K \diamond \mathrm{app}_2\ (\lambda^* x.\ e)\ v_2 \;\Longrightarrow\; K \diamond [v_2/x] e \\
\textsf{st_letv} & :: & K \diamond \underline{\mathrm{let}}\ \underline{\mathrm{val}}\ x = e_1 \underline{\mathrm{in}} e_2 \;\Longrightarrow\; (K; \lambda x_1.\ [x_1/x] e_2) \diamond e_1 \\
\textsf{st_letn} & :: & K \diamond \underline{\mathrm{let}}\ \underline{\mathrm{name}}\ x = e_1 \underline{\mathrm{in}} e_2 \;\Longrightarrow\; K \diamond [e_1/x] e_2 \\
\textsf{st_fix} & :: & K \diamond \underline{\mathrm{fix}}\ u.\ e \;\Longrightarrow\; K \diamond [\underline{\mathrm{fix}} u.\ e/u] e
\end{array}$$