Mechanizing the Metatheory of LF in Twelf

Chris Martens       Karl Crary

Carnegie Mellon University

August 11, 2011

Presented at ITU Denmark
Twelf and LF

**Twelf** is a proof assistant.

- Proof-carrying code \(^1\)
- SeLF/Gray (distributed security) \(^2\)
- Metatheory of ML \(^3\)

**LF:** Twelf’s underlying (dependent) type theory

---


\(^2\) [http://www.cs.cmu.edu/ self/](http://www.cs.cmu.edu/ self/)

\(^3\) [http://www.cs.cmu.edu/ dklee/tslf/](http://www.cs.cmu.edu/ dklee/tslf/)
A proof about a proof assistant: LF is “correct”.

This project is about...

- Reasoning about dependent types: syntactic (as opposed to LR) approach
- Reasoning with dependent types: proof engineering
Metalanguage and object language
Example OL: STLC

\[ e ::= \lambda x: \tau. e \mid (e \ e) \mid x \]

\[ \tau ::= o \mid \tau \rightarrow \tau \]

exp : type.  tp : type.
lam : tp \rightarrow (exp \rightarrow exp) \rightarrow exp.
app : exp \rightarrow exp \rightarrow exp.
o : tp.
arr : tp \rightarrow tp \rightarrow tp.
LF Methodology: Judgments as Types

\[ \Gamma \vdash e : \tau \]

\[
\begin{array}{c}
e_1 : \tau' \rightarrow \tau \\
e_2 : \tau'
\end{array} \quad \frac{}{(e_1 \ e_2) : \tau}
\]

of/\text{app} : of \ E1 \ (\text{arrow} \ T' \ T) \\
\rightarrow \ of \ E2 \ T' \\
\rightarrow \ of \ (\text{app} \ E1 \ E2) \ T.

of : \ exp \rightarrow \ tp \rightarrow \ type.
Twelf Methodology

Theorems $\equiv$ total relations over derivations

preservation : of E T $\to$ step E E' $\to$ of E' T $\to$ type.
%mode preservation +X1 +X2 -X3.

%% proof goes here...

%worlds () (preservation _ _ _).
%total D (preservation _ D _).

Relation on a typing derivation, a stepping derivation, and another typing derivation.
Twelf directives: %mode, %worlds, %total
Dependent types!

\[
\begin{align*}
\frac{x : A \vdash M : B}{\lambda x: A. \, M : \Pi x: A. \, B} & \quad \frac{M : \Pi x: A. \, B \quad N : A}{(M \, N) : [N/x]B}
\end{align*}
\]

of : tm -> tp -> type  \hspace{1cm} \textbf{type family} indexed by a tm and tp
of (lam o [x] x) (arr o o)  \hspace{1cm} \textbf{dependent type} inhabited by derivations
Objects $M ::= c \mid x \mid \lambda x:A. M \mid (M \ M)$
Families $A ::= a \mid \Pi x:A. A \mid \lambda x:A. B \mid (A \ M)$
Kinds $K ::= \text{type} \mid \Pi x:A. K$
Contexts $\Gamma ::= \cdot \mid \Gamma, x : A$
Signatures $\Sigma ::= \cdot \mid \Sigma, c : A \mid \Sigma, a : K$
The user populates a signature $\Sigma$, which is checked:

\[
\frac{\Sigma \text{ ok} \quad \Sigma; \vdash A : \text{type}}{
(\Sigma, c : A) \text{ ok}}
\]

\[
\frac{\Sigma \text{ ok} \quad \Sigma; \vdash K \text{ wf}}{
(\Sigma, a : K) \text{ ok}}
\]
Type formation depends on term typing.

Key rule:

\[
\begin{align*}
M : A & \quad A \equiv B : \text{type} \\
\hline
\therefore M : B & 
\end{align*}
\]
LF typechecking

\[ \Gamma \vdash M \equiv N : A \]

\[ \beta - \eta \text{ equivalence:} \]

\[ ((\lambda x : A. M) \ N) \equiv [N/x]M : [N/x]B \] \( \text{beta} \)

\[ x : A \vdash (M \ x) \equiv (N \ x) : B \]

\[ M \equiv N : \Pi x : A. B \] \( \text{ext} \)
Adequacy

LF’s notion of *correctness of an encoding*:

Terms of the OL are in bijective, compositional correspondence with *canonical* LF terms of type $\text{tm}$.

Canonical $= \beta$ short, $\eta$ long
The theorems

- Decidability of type checking
- Existence of canonical forms
Prior work

Most relevant:

- Detailed paper proof using logical relations \(^4\)
- Formalization of that proof in Isabelle \(^5\)
- Crary’s proof about the singleton calculus \(^6\)

---


\(^6\)Karl Crary. A syntactic account of singleton types via hereditary substitution. LFMTP ’09.
Our approach

Proof sketch:

- Define a system in which equivalence is syntactic
- Define a translation to that system
- Show the translation sound and complete
The Canonical Forms Presentation

Syntactic separation of terms (due to Felty\textsuperscript{7}):

Terms $M ::= \lambda x. M | R$
Atoms $R ::= c | x | (R M)$
Families $A ::= \Pi x:A. A | \lambda x:A. B | P$
Atomic Families $P ::= a | (P M)$

The Canonical Forms Presentation

Bidirectional typechecking:

<table>
<thead>
<tr>
<th>Canonical Typing</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Gamma^+ \vdash R^+ \Rightarrow A^-$</td>
</tr>
<tr>
<td>$\Gamma^+ \vdash M^+ \Leftarrow A^+$</td>
</tr>
</tbody>
</table>

$\beta$-shortness: syntactic

$\eta$-length: enforced by typing

$$
\begin{align*}
R & \Rightarrow P \\
R & \Leftarrow P
\end{align*}
$$

($P$ is a base type)
**Hereditary Substitution**

Hereditary substitution\(^8\) maintains canonical forms:

To substitute \(N\) into \((R \ M)\):

- Let \(M' = [N/x]M\) and \(R' = [N/x]R\)
- If \(R' = \lambda x. \ O\), recursively do \([M'/x]O\)
- Otherwise, \((R' \ M')\)

---

Expansion

To turn an arbitrary atom into a term, expand at a simple type.

\[ \eta_o(R) = R \]
\[ \eta_{S \rightarrow T}(R) = \lambda x. \eta_T(R \eta_S(x)) \]
Canonical LF Metatheory

- **Substitution**: Given $x : A \vdash M \Leftarrow B$ and $N \Leftarrow A$, the hereditary substitution $[N/x]M$ exists and $[N/x]M \Leftarrow B$

- **Identity**: Given $R \Rightarrow A$, the expansion $\eta_{simp(A)}(R)$ exists and $\eta_{simp(A)}(R) \Leftarrow A$. 
Canonical LF Metatheory

Substitution proof

**Key lemma**: permuting substitutions

\[
[N/x][M/y]O = [[N/x]M/y][N/x]O
\]

**Termination metric**: The *simple type* of the variable and the *typing derivation* of the open term.

The type in the metric is needed for the hereditary case.

Identity proof

Straightforward induction over the expansion derivation.
Stop here?

Why not stop here?
Hereditary substitution eliminates all need for noncanonical forms...
Translation

“EL” = LF with definitional equivalence
“IL” = Canonical forms LF

\[ \Gamma \vdash M \rightsquigarrow \overline{M} : A \]

- \( \eta \)-expands constants and variables
- Translates each piece of application and uses hsub
- Transliterates the rest (type output needed for \( \lambda \))
Mechanizing the Metatheory of LF in Twelf

Proof engineering sidebar

\( \mathbf{x} \mapsto \mathbf{x} \) on paper, but in LF we have to hypothesize EL var, IL var, and a translation between them. Maintained in \textit{blocks}, which further grow to carry type translations.

\%
block cbind
: some \{EA:etp\} \{A:tp\} \{d_{tptrans}:tptrans\ EA A \textit{ktype}\}
block \{x:atm\} \{d:atof\ x A\}
\{ex:etm\} \{ed:eof\ ex EA\}
\{xt:vtrans\ ex x\}.


The target of term translation is **canonical terms**.

How should a family $A : \prod x : B . K$ translate?

Our approach: add $\lambda$ at the family level; treat objects and families uniformly.

Perhaps a simpler avenue: translate to a disjunctive class $(P + A)$. 
Correctness of the Translation

Completeness

If $M \equiv N : A$, then
$M \rightsquigarrow Q : \overline{A}$ and $N \rightsquigarrow Q : \overline{A}$

Soundness

If $M \rightsquigarrow Q : A$, $N \rightsquigarrow Q : A$ and $\hat{A} \rightsquigarrow A : \text{type}$
Then $M \equiv N : \hat{A}$
Completeness

If $M \equiv N : A$, then
$M \leadsto Q : \overline{A}$ and $N \leadsto Q : \overline{A}$

**Proof:** By induction over the structure of $M \equiv N : A$.
Hard cases: beta and ext (as expected)
**Key lemma:** Permutability of translation and substitution.
Soundness

If $M \rightsquigarrow Q : A$, $N \rightsquigarrow Q : A$ and $\hat{A} \rightsquigarrow A : \text{type}$
Then $M \equiv N : \hat{A}$

Proven via “transliteration”: $\Gamma \vdash M : A \mapsto \hat{M}$
Approximately id; needs the type for $\lambda$
Theorem: the transliterated translation of $M$ is $\equiv M$. 
Soundness

Key lemma:

Transliteration “almost” permutes with substitution

If $x : A \vdash (M : B) \mapsto \hat{M}$
and $(N : A) \mapsto \hat{N}$
then $[\hat{N}/x]M \equiv [\hat{N}/x]M : \hat{A}$

This held us up for about a year.
Solution: syntactic reduction approach.

\[ \Gamma \vdash M \rightarrow N \]

Transliteration “almost” permutes with substitution

If \( x : A \vdash (M : B) \mapsto \hat{M} \)

and \( (N : A) \mapsto \hat{N} \)

then \( [\hat{N}/x]\hat{M} \rightarrow^\ast [\hat{N}/x]M \)

Relating reduction and equivalence

If \( M \rightarrow M' \) and \( M : A \) then \( M \equiv M' : A \).
Relating reduction and equivalence

If $M \rightarrow M'$ and $M : A$ then $M \equiv M' : A$.

Must be proved simultaneously with inversion of $\lambda$ typing

- reduce-equiv uses $\lambda$ inversion in the $\beta$ case
- $\lambda$ inversion uses reduce-equiv in the ext case

\[
\frac{x : A' \vdash ((\lambda x : A. M) \, x) : B}{\lambda x : A. M : \Pi x : A'. B} \quad \text{ext}
\]

- Both cases need $\Pi$ injectivity
Relating reduction and equivalence

Π injectivity

If \( \Pi x:A. B \equiv \Pi x:A'. B' : \text{type} \) then
\( A \equiv A' : \text{type} \) and \( x : A \vdash B \equiv B' : \text{type} \)

Prior approaches \(^9\) based on logical relations.
Our approach:

Generalized Π injectivity

If \( A \equiv B : K \) and \( A \Downarrow A' \), then \( B \Downarrow B' \) and \( A' \sim B' : K \).

\(^9\)Harper-Pfenning;
Pi Injectivity

Generalized Π injectivity

If $A \equiv B : K$ and $A \downarrow A'$, then $B \downarrow B'$ and $A' \sim B' : K$.

$A \downarrow A'$: Syntactic normalization of family-level reduction (maintains terms)

$A \sim B : K$: A stronger notion of equivalence; implies injectivity
Family normalization

\[ \Pi x : A. B \downarrow \Pi x : A. B \quad \text{norm/pi} \]
\[ x \vdash B \downarrow B' \quad \lambda x : A. B \downarrow \lambda x : A. B' \quad \text{norm/\lambda lam} \]
\[ (B \ M) \downarrow [M/x]C \quad \text{norm/app} \]
**Similarity**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ A \equiv A' : \text{type} \quad x : A \vdash B \equiv B' : \text{type} ]</td>
<td>sim/(\pi)</td>
</tr>
<tr>
<td>[ \Pi x : A.B \sim \Pi x : A'.B' : \text{type} ]</td>
<td>sim/(\pi)</td>
</tr>
<tr>
<td>[ x : C \vdash B \sim B' : K ]</td>
<td>sim/(\lambda)</td>
</tr>
<tr>
<td>[ \lambda x : A.B \sim \lambda x : A'.B' : \Pi x : C.K ]</td>
<td>sim/(\lambda)</td>
</tr>
</tbody>
</table>
Pi Injectivity

Generalized $\Pi$ injectivity

If $A \equiv B : K$ and $A \downarrow A'$,
then $B \downarrow B'$ and $A' \sim B' : K$.

Key: Separating family-level from term-level computation
Recap

- Syntactic proof of LF’s metatheory using hereditary substitution
- Fully mechanized
- Novel (AFAWK) approach to Π injectivity for LF with η expansion and family-level λ
Proof engineering challenges

Explicit Contexts

Need to define a copy of the language in certain cases to avoid dependencies on the ambient context (example follows)

---

Karl Crary. Explicit Contexts in LF. LFMTP 2009.
Explicit contexts

\[ y \vdash [N/x]M_{xy} = M'_y \]

\[ [N/x](\lambda y.M_{xy}) = \lambda y.M'_y \quad \text{sub/lam} \]

\[ y : A \vdash M_y : B_y \]

\[ \lambda y.M_y : \Pi y : A. B_y \quad \text{of/lam} \]
Explicit contexts

\[ y \vdash [N/x]M_{xy} = M'_y \]
\[ [N/x](\lambda y.M_{xy}) = \lambda y.M'_y \] \textit{sub/lam}

\[ x : C, y : A_x \vdash M_{xy} : B_{xy} \]
\[ x : C \vdash \lambda y.M_{xy} : \Pi y:A_x.B_{xy} \] \textit{of/lam}

By induction we want \( y : A_x \vdash M'_y : B'_y \)
In Twelf...

- : subst
  (sub/lam ([y] DsubM y : sub ([x] M x y) N (M’ y)))
  ([x] [d:of x C]
   of/lam
   ([y] [e:of y (A x)]
     DofM x d y e : of (M x y) (B x y)))
  [...] 
XXX
<- ({y} {d: of y (A x)} % x not bound!
    subst (DsubM y) (DofM y) [...] (DofM’ y))
Proof engineering challenges

Explicit Contexts

ctx : type. [...]  
ofe : ctx -> tm -> tp -> type.

Can shift between ofe nil M A and of M A.  
Unfortunately, significant overhead, and no good way to “librarify”.  

Proof engineering challenges

Manual equality reasoning

```
    tm-eq : tm -> tm -> type.
    tm-eq/i : tm-eq M M.
```

For every type family (no polymorphism)...

- plus congruence, compatibility, and respects lemmas
- Library?
- Automation?
Proof engineering challenges

No internalization of functionality

Effectiveness and uniqueness lemmas abound
Proof Engineering

Joys:
- Type reconstruction: thinking is plan B
- Visible derivation manipulation
- HOAS and blocks

Regrets:
- Well-formedness checks in inference rules
- Adding family-level $\lambda$ (mixed blessing)
Proof Engineering

- 36503 lines of (heavily spaced, annotated) Twelf
  - about 14 KLOC for Canonical LF
  - about 13 KLOC just on explicit contexts
- Checks in a few seconds
- On my webpage
  - http://www.cs.cmu.edu/~cmartens
Future Work

- Formalize lack of need for (or do away with) family $\lambda$
- Intrinsic encoding$^{11}$
- Polymorphism; other extensions to LF

---

Current interests

- Logic programming and dependent types in other settings
- Two-level logics e.g. Abella
- Substructural/polarized logic programming