A Linear Logical Framework

Iliano Cervesato - Frank Pfenning
Department of Computer Science
Carnegie Mellon University

11th IEEE Symposium on Logic in Computer Science

New Brunswick, NJ, July 28th, 1996
Overview

- Logical frameworks
  - definition
  - applications
  - examples (LF)
  - limitations

- The linear logical framework LLF
  - introduction
  - the linear type theory $\lambda^{\Pi-o & T}$
  - main properties
  - meta-representation in LLF

- An example: $ML_{ref}$
  - syntax
  - typing and evaluation
  - type preservation

- Related work and conclusions
Logical frameworks

Formalisms specially designed to provide effective meta-representations of formal systems

- **Formal systems:**
  - logics
  - programming languages
  - ...

- **Meta-representation:**
  - syntax
  - semantics
  - meta-theory
  - ...

- **Effectiveness:**
  - immediacy
  - executability
Applications

- Traditional logic and type theory
  - cut elimination
  - Church-Rosser property
  - soundness and completeness proofs
  - ...

- Pure logic and functional programming languages
  - interpretation
  - compilation
  - correctness of program transformations
  - representation of properties
    - type preservation, value soundness, ...
      (Mini-ML)
    - completeness of uniform provability, resolution, ...
      (Horn clauses)
  - ...

- ...

I. Cervesato, F. Pfenning - A Linear Logical Framework
Structure of a logical framework

Logical framework

= 

\textit{meta-representation language}

+ 

\textit{meta-representation methodology}
Meta-representation languages

Logics
- Horn clauses (Prolog)
- Higher-order hereditary Harrop formulas ($\lambda$Prolog, Isabelle)
- Classical linear logic (Forum)
- ...

Type theories
- $\lambda^\Pi$ (Elf)
- CIC (Coq, Lego)
- Martin-Löf's type theories (ALF, NuPrl)
- ...

I. Cervesato, F. Pfenning - A Linear Logical Framework 6
The type theory $\lambda^\Pi$

Kinds

- $K ::= \text{type} \mid \Pi x:A. K$

Type families

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

Types

- $A ::= P \mid \Pi x:A_1. A_2$

Objects

- $M ::= x \mid c \mid \lambda x:A. M \mid M_1 M_2$

```
M has type A
in $\Gamma$ and $\Sigma$
```

Principal properties

- Type checking and type synthesis are decidable
- Can be implemented as a logic programming language ($Elf$)
- Proof-terms record the inference rules used in proving the inhabittance of a type
Meta-representation methodology

\[ \Delta \rightarrow e : \tau \]

- Term-based representation

\[ |-_{\Sigma} M : \text{ofe} \Delta \vdash e \vdash \tau \]

We must encode *explicitly*
- context operations (lookup, insertion, ...)
- context-related properties (weakening, exchange, ...)

- Exploitation of the meta-language context

\[ \Gamma \vdash \tau \]

where, for each \( x_i : \tau_i \) in \( \Delta \),

\[ [x_i : \tau_i] = x_i : \text{exp}, \ t_i : \text{ofe} \ x_i \vdash \tau_i \]

- context operations reduce to meta-level primitives
- meta-theoretic properties are inherited from the meta-language

\[ LF \]
Aspects of a meta-representation

Meta-representation

\[ \text{=} \]

program

\[ \text{+} \]

adequacy theorems
Meta-representation in LF: signature

Example:

\[
\begin{align*}
\text{e ::= } & x \mid \ldots \mid \text{lam } x. \ e \mid e_1 \; e_2 \mid \\
\text{\tau ::= } & \ldots \mid \tau_1 \rightarrow \tau_2 \mid \\
\text{exp : type.} \\
\text{lam : (exp } \rightarrow \text{exp) } \rightarrow \text{exp.} \\
\text{app : exp } \rightarrow \text{exp } \rightarrow \text{exp.}
\end{align*}
\]

\[
\Delta, x : \tau_1 \vdash e : \tau_2 \quad \Delta \vdash e_1 : \tau_2 \rightarrow \tau_1 \quad \Delta \vdash e_2 : \tau_2
\]

\[
\Delta \vdash \text{lam } x. \ e : \tau_1 \rightarrow \tau_2 \quad \Delta \vdash e_1 \; e_2 : \tau_1
\]

\[
\text{ofe : exp } \rightarrow \text{tp } \rightarrow \text{type.}
\]

\[
\text{of_lam : ofe (lam E) (T1 } \rightarrow \text{T2)} \\
\quad \Leftarrow (\{x\text{:exp}\} \text{ ofe x } T1 \\
\quad \quad \rightarrow \text{ ofe (E x) } T2).
\]

\[
\text{of_app : ofe (app } \ E_1 \ E_2 \text{) } T_1 \\
\quad \Leftarrow \text{ ofe } \ E_1 \ (T_2 } \rightarrow \text{T1)} \\
\quad \Leftarrow \text{ ofe } \ E_2 \ T_2.
\]
Adequacy theorem (typing of expressions)

Given a context $\Delta = (x_1 : \tau_1, \ldots, x_n : \tau_n)$, an expression $e$ and a type $\tau$, there is a compositional bijection between derivations $T$ of

$$\Delta \rightarrow e : \tau$$

and canonical $LLF$ objects $M$ such that

$$\Gamma_\Delta \vdash_{\Sigma} M : \text{ofo}\ e \Gamma \tau$$

is derivable, where

$$\Gamma_\Delta = \begin{bmatrix}
  x_1: \text{exp}, \ t_1: \text{ofo}\ x_1 \Gamma_\tau \\
  \ldots \\
  x_1: \text{exp}, \ t_n: \text{ofo}\ x_n \Gamma_\tau
\end{bmatrix}$$
Limitations

The context-based representation methodology does not handle satisfactorily:

– linearity (affine, relevant, linear logics, ...)
– state (imperative programming languages, planning, games, ...)
– modality (modal logics, ...)

The representation of these problems involves complex encodings:

– adequacy is difficult to prove
– the meta-theory is not manageable

Exception

Forum
The problem

\[ S |- K; e \Rightarrow a = M \]

\[ \text{Store!!!} \]
\[ c_i=v_i,\ldots \]

- Term-based representation

\[ \cdot |-_\Sigma M : eval \{ S \} \{ K \} \{ e \} \{ a \} \]

... as before

- Context-based representation

\[ \{ S \} |-_\Sigma M : eval \{ K \} \{ e \} \{ a \} \]

This does not work!

- S is subject to destructive operations (e.g. assignment)
- current logical frameworks do not allow removing assumptions from the context
Design a logical framework that

– permits a direct representation of linearity/state/...

– is conservative over $LF$
  
  · language ($\lambda^{\Pi}$)
  
  · meta-representation methodology
  
  · examples

– has usable operational properties
Beyond intuitionism

Linearity/state/... are problematic because intuitionistic context management is monotonic.

The above problems require instead a non-monotonic management of the context.

Linear logic permits non-monotonic context management.
Choice of the operators

Desiderata

- model arbitrary non-monotonic context operations
- conservative extension of the operators of $\lambda^\Pi$
- existence of unique canonical forms
- completeness of uniform proof search

$\Pi \; \circ \; \& \; \top$

as type constructors. The corresponding object operators are extracted from their natural deduction style inference rules.

This is the type-theoretic version of the language of linear higher-order hereditary Harrop formulas, where

\[ \Pi x : A . \; B \quad (\equiv \; !A \; \circ \; B) \]

\[ A \rightarrow B \]

\[ \forall x . \; B \]

We are within intuitionistic linear logic
The linear type theory
\( \lambda \Pi -o & T \)

Kinds
\[ K ::= \text{type} \mid \Pi x:A. K \]

Type families
\[ P ::= a \mid P \, M \]

Types
\[ A ::= P \mid \Pi x:A_1. A_2 \mid A_1 \rightarrow A_2 \mid A_1 \land A_2 \mid T \]

Objects
\[ M ::= x \mid c \mid \lambda x:A. M \mid M_1 \, M_2 \mid \lambda x^A. M \mid M_1^A \, M_2 \mid <M_1, M_2> \mid \text{fst} \, M \mid \text{snd} \, M \]

Object (proof-term)
Type
\[ \Psi \vdash \Sigma \quad M : A \]

"M has type A in \( \Psi \) and \( \Sigma \)"

Context
\[ x:A, ... \]
\[ x^A, ... \]

Signature
\[ a:K, ... \]
\[ c:A, ... \]

\( \lambda \Pi -o & T \) is the largest propositional linear extension of \( \lambda \Pi \) admitting unique canonical forms
More $\lambda^{\Pi-o} \& T$

Intuitionistic context $x:A, \ldots$

Signature

Type

$\overline{\Psi} \vdash_{\Sigma} A : \text{type}$

"$A$ is a type in $\overline{\Psi}$ and $\Sigma$"

Types and kinds are linearly closed:
no linear dependencies
Properties of $\lambda^{\Pi-0} & T$

We restricted the semantics of $\lambda^{\Pi-0} & T$ to terms that are in $\eta$-long form:

$$\Psi \vdash_\Sigma U \uparrow V$$

- simpler
- sufficient

- **Church-Rosser property**
  
  If $U' \equiv U''$, there exists a term $V$ such that $U' \rightarrow^* V$ and $U'' \rightarrow^* V$

- **Strong normalization**
  
  If $\Psi \vdash_{-\Sigma} U \uparrow V$ is derivable, then $U$ is strongly normalizing

- **Decidability of type checking and type synthesis**
  
  It can be recursively decided whether there exists a derivation and a term $V$ for the judgment $\Psi \vdash_{-\Sigma} U \uparrow V$

- **Conservativity over LF**
  
  If $\Psi, \Sigma, U$ and $V$ do not mention linear constructs, then $\Psi \vdash_{-\Sigma} U \uparrow V$ is derivable in $LLF$
  iff $\Psi \vdash_{-\Sigma}^{LF} U \uparrow V$ is derivable in $LF$
Logic programming in $\lambda^\Pi$-o & T

Proof-search in $\lambda^\Pi$-o & T can be efficiently mechanized.

$LLF$ is adequate for an implementation as a logic programming language:

- it is complete for uniform proof search
- $\lambda^\Pi$-o & T is the largest propositional linear extension of $\lambda^\Pi$ that is complete for uniform proof search

- it admits a form of resolution

- further non-determinism:
  - resource distribution: context management
  - conjunctive: sequentialization
  - disjunctive: backtracking
  - existential: unification
Meta-representation methodology

Intuitionistic $LLF$ assumptions
– part of the object-level context managed \textit{monotonically}
– object-level parameters

Linear $LLF$ assumptions
– part of the object-level context managed \textit{linearly}

The operators of $\lambda^\pi^\text{o} \& T$ are sufficient to express arbitrary non-monotonic context manipulations
Case study: $ML^{ref}$

$ML^{ref}$ is a fragment of $ML$ with

- references
- value polymorphism
- recursion

**Types:**

\[
\tau ::= \ldots | \tau_1 \rightarrow \tau_2 | \tau^{ref} | 1
\]

**Expressions:**

\[
e ::= x \ | \ldots | \text{lam} \ x. \ e \ | \ e_1 \ e_2 \ | \ldots
\]
\[
\quad | \ c \ | \langle \rangle
\]
\[
\quad | \text{ref} \ e \ | \ !e \ | \ e_1 := e_2 \ | \ e_1 ; e_2
\]

**Store:**

\[
S ::= \cdot \ | \ S, \ c=v
\]

Expressions

exp : type.
cell : type.

\[
\ldots
\]

loc : cell -> exp.
unit : exp.
ref : exp -> exp.
deref : exp -> exp.
assign : exp -> exp -> exp.
seq : exp -> exp -> exp.
**ML_{ref}**: typing

\[ \Delta |- e : \tau \]

“\(e\) has type \(\tau\) in \(\Delta\)”

**Representation:**

\[ \begin{align*}
\Delta |\Sigma & \vdash T : ofe \\
\Delta |\Sigma & \vdash e : \tau
\end{align*} \]

\[ x_i : \text{exp}, \quad t_i : \text{ofe} \quad x_i : \tau_i, \ldots \\
\quad c_j : \text{cell}, \quad l_j : \text{ofc} \quad c_j : \sigma_j, \ldots \]

\[
\Delta |- e : \tau_{\text{ref}} \\
\Delta |- \text{!} e : \tau
\]

\[
\Delta |- e_1 : \tau_{\text{ref}} \quad \Delta |- e_2 : \tau \\
\Delta |- e_1 ::= e_2 : 1
\]

\[
\text{ofe}_{\text{derefer}} : \\
ofe \text{ (derefer E) T} \\
\quad \leftarrow \text{ofe E (rf T)}. \\
\]

\[
\text{ofe}_{\text{asign}} : \\
ofe \text{ (assign E1 E2) l} \\
\quad \leftarrow \text{ofe E1 (rf T)} \\
\quad \leftarrow \text{ofe E2 T}. \\
\]
**ML_{ref}: evaluation**

Continuation

init, ..., λx.i, ...

Instruction:

eval c, return v, ...

"i followed by K evaluates to a, starting in S"

**Representation:**

\[
S \vdash K; i \Rightarrow a
\]

\[
\left[ S \right]^\_ \vdash \left[ E \right] \overset{\text{eval}}{\rightarrow} \left[ K \right] \left[ i \right] \left[ a \right]
\]

\[
c_i: \text{cell, } h_i^\wedge \text{contains } c_i[ v_i ] , ...
\]
\[ S',c=v,S'' |- K; \textbf{return} v \Rightarrow a \]

\[ S',c=v,S'' |- K; !c \Rightarrow a \]

\[
\text{ev\_deref1} : \text{eval} \ K (\text{ref1} \ (\text{loc} \ C)) \ A \\
\quad o- \ \text{read} \ C \ V \\
\quad \quad \& \ \text{eval} \ K (\text{return} \ V) \ A.
\]

\[
\text{rd} : \text{read} \ C \ V \\
\quad o- \ \text{contains} \ C \ V \\
\quad o- <T>.
\]

\[ S',c=v,S'' |- K; \textbf{return} <> \Rightarrow a \]

\[ S',c=v',S'' |- K; c := v \Rightarrow a \]

\[
\text{ev\_assign2} : \text{eval} \ K (\text{assign2} \ (\text{loc} \ C) \ V) \ A \\
\quad o- \ \text{contains} \ C \ V' \\
\quad o- \ (\text{contains} \ C \ V \\
\quad \quad -o \ \text{eval} \ K (\text{return} \ \text{unit}) \ A.
\]
\textbf{ML}^\text{ref}: adequacy

\textbf{Adequacy theorem (evaluation)}

Given a store \( S = (c_1 = v_1, \ldots, c_n = v_n) \), a continuation \( K \), an instruction \( i \) and an answer \( a \), all closed, there is a compositional bijection between derivations \( \mathcal{X} \) of

\[ S \vdash K; i \Rightarrow a \]

and canonical LLF objects \( M \) such that

\[ \llbracket S \rrbracket \vdash M \uparrow \text{eval} \llbracket K \rrbracket \llbracket i \rrbracket \llbracket a \rrbracket \]

is derivable, where

\[ \llbracket S \rrbracket = \begin{bmatrix}
\text{\( c_i \): cell, \( h_i \) ^ contains c_i \( \llbracket v_i \rrbracket \)} \\
\ldots \\
\text{\( c_n \): cell, \( h_n \) ^ contains c_n \( \llbracket v_n \rrbracket \)}
\end{bmatrix} \]
**ML^{ref}:** type preservation

**Theorem (type preservation)**

If $S \vdash K; i \Rightarrow a$, with $\Delta \vdash i : \tau$, $\Delta \vdash K : \tau \Rightarrow \sigma$ and $\Delta \vdash S : \Delta$, then $\Delta \vdash a : \sigma$

**Proof:** by induction on the evaluation derivation

The *high level of abstraction* of the representation permits *transcribing* this proof into an *LLF* program capturing its computational contents:

– each case yields one clause

– the meta-reasoning is itself *linear*

**Representation:**

```ml
val tpev =
  eval K I A -> ofk K T S -> ofi I T ->
  off A S -> type.
```
Related work

• Historical perspective
  – Elf [Pfenning 94]
  – Lolli [Hodas & Miller 94]

• Forum [Miller 94, Chirimar 95]
  – the language of formulas is (full) classical linear logic
  – the language of terms is (traditional) Church's simply typed λ-calculus (no linearity)
  – the representation of linear provability relies on techniques similar to LLF's
  – linear derivations are not representable (no linear objects)
  – the relationship between provability and derivations is external (no proof-terms)
Conclusions

*LLF* is a conservative linear extension of *LF*

It has been used for the representation of

- imperative languages (*ML*\textsuperscript{ref}, polyC)
- non-traditional logics (CLL, S4)
- languages with non-standard binders (linear *λ*-calculi)

Direct implementations of

- cut-elimination for CLL
- games (Mahjongg, tic-tac-toe, connect 4, ...)
- planning (block world)
- imperative graph search
Future work

Implementation

- context management [ELP'96, with J. Hodas]
- unification
  - Huet's style pre-unification [CADE WP-6]
  - pattern fragment
  - constraints
- type/term reconstruction

Dependent versions of -o and & [Ishtiaq-Pym]

Schema checking [Pfenning-Rohwedder]

Non-commutative linear framework [Penn-Pfenning]