A Linear Logical Framework

Iliano Cervesato

Department of Computer Science
Carnegie Mellon University
Contents

- Overview
- Linear logic
- Logical frameworks
- $LLF$
- Case study
- Future developments
Overview

A Logical Framework is a formalism designed to represent and reason about deductive systems

Aim:

- identify the principles underlying logics and programming languages [Pfenning’92; Michaylov,Pfenning’91; Shankar’94; Pfenning’95]

Intended applications:

- design of new and better logics and programming languages
- program verification and certification [Necula’97]

Limitations:

- ineffective with imperative formalisms [Pfenning’94]
State

So far, no simple, general and effective treatment of the recurring notion of state

– store of an imperative programming language
– database
– communication among concurrent processes, ...

A recent approach: Linear Logic [Girard’87]

• adequate for representing state and imperative computation [Chirimar’95; Hodas, Miller’94; Wadler’90]
• ineffective for reasoning about them
Thesis Contribution

- Design of a formalism, \( LLF \), that combines
  - the meta-reasoning power of traditional logical frameworks
  - the possibility of linear logic of handling state

- First linear type theory in literature

- Conservative over \( LF \) [Harper,Honsell,Plotkin’93]

- Used to represent
  - imperative programming languages
  - substructural logics
  - games, ...
  
  and to reason about them
Logical Frameworks

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

**formal system**

programming languages, logics, ...

**meta-representation**

represent language constructs, model their semantics, encode properties and their proofs

**effectiveness**

immediacy and executability

```
Logical framework = meta-language + representation methodology
```
Prior Achievements

- Logic
  - intuitionistic, classical, higher-order [Harper,Honsell,Plotkin’93]
  - modal [Avron,Honsell,Mason’89; Pfenning,Wong’95; Pfenning,Davies’96]
  - linear [Pfenning’95]
- Cut elimination [Pfenning’95]
- Logical interpretations [Pfenning,Rohwedder]
- Program extraction [Anderson’93]
- Categorical grammars and Lambek calculus [Penn’95]
- Church-Rosser theorem [Pfenning’92]
- Category theory [Gehrke’95]
- Theorem Proving [Pfenning’92]
- Logic programming [Pfenning’92]
Prior Achievements (Cont’d)

- **Mini-ML**
  - type preservation [Pfenning, Michaylov’91]
  - compiler correctness [Pfenning, Hannan’92]
  - compiler optimization [Hannan]
  - polymorphism [Pfenning’88; Harper’90]
  - CPS conversion, callcc [Pfenning, Danvy’95]
  - exceptions [Necula]
  - subtyping [van Stone]
  - refinement types [Pfenning’93]
  - partial evaluation [Hatcliff’95; Davies’96]

- Lazy functional programming
  - λ-lifting [Leone]
  - lazy evaluation [Okasaki]
  - monads [Gehrke’95]
Meta-Language

• Logics
  – Horn clauses (Prolog)
  – Higher-order hereditary Harrop formulas (λProlog [Miller,Nadathur’88], Isabelle [Paulson’93])
  – Classical linear logic (Forum [Miller’94])

• Type theories
  – λΠ (LF [Harper,Honsell,Plotkin’93])
  – Calculus of Constructions (Coq [Dowek&al’93], Lego [Pollack’94])
  – Martin-Löf’s type theories (ALF [Nordström’93], NuPrl [Constable&al’86])
  – λΠ−ο&⊤ (LLF [Cervesato’96])
Representation Methodology

Judgments-as-Types / Derivations-as-Objects

• Each object judgment is represented as a base type

• The context of an object judgment is encoded in the context of the meta-language

• Object-level inference rules are represented as constants that map derivations of their premisses to a derivation of their conclusion

• Derivations of an object judgment are represented as canonical terms of the corresponding base type
Representation of the Context

\[
\begin{array}{c}
x_i : \tau_i, \ldots \vdash \mathcal{T} \\
\Omega \vdash e : \tau = M
\end{array}
\]

- Term-based representation

\[
\vdash_{\Sigma} M : \text{has\_type} \quad \Gamma \vdash e \Gamma \vdash \tau \Gamma
\]

We must encode *explicitly*

- context operations (lookup, insertion, ...)
- context-related properties (weakening, exchange, ...)
Representation of the Context (Cont’d)

\[
\varrho_i : \tau_i, \ldots \quad \vdash \quad \mathcal{T} \quad \Downarrow \\
\Omega \vdash e : \tau 
= M
\]

- Exploitation of the meta-language context

\[
\Gamma \Omega \vdash_\Sigma M : \text{has_type} \quad \Gamma e \Gamma \quad \Gamma \tau \Gamma
\]

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

\[
\Gamma x_i : \tau_i \Gamma = x_i : \exp, \ t_i : \text{has_type} \ x_i \Gamma \tau_i \Gamma
\]

- context operations reduce to meta-level primitives
- meta-theoretic properties are inherited from the meta-language
\[ \lambda^\Pi, \text{the Meta-Language of LF} \]

- **Syntax**

  \[ \text{Kinds} \quad K := \text{type} \mid \Pi x : A. K \]
  \[ \text{Type families} \quad P := a \mid P M \]
  \[ \text{Types} \quad A := P \mid \Pi x : A. B \]
  \[ \text{Objects} \quad M := x \mid c \mid \lambda x : A. M \mid M N \]

- **Semantics**

  \[ \Gamma \vdash_{\Sigma} M : A \quad \text{“M has type A in } \Gamma \text{ and } \Sigma” \]

  \[ \begin{array}{cc}
  \text{Context} & \text{Signature} \\
  x : A, \ldots & a : K, \ldots, c : A, \ldots
  \end{array} \]
\[ \lambda \Pi, \text{the Meta-Language of LF (Cont’d)} \]

\[ \frac{\Gamma, x : A \vdash \Sigma M : B}{\Gamma \vdash \Sigma \lambda x : A. M : \Pi x : A. B} \text{ lam} \]

\[ \frac{\Gamma \vdash \Sigma M : \Pi x : A. B \quad \Gamma \vdash \Sigma N : A}{\Gamma \vdash \Sigma M N : [N/x]B} \text{ app} \]

- **Main properties**
  - is strongly normalizing
  - admits unique canonical forms
  - type checking is decidable
  - can be implemented as a logic programming language \((Elf \ [\text{Pfenning'94}])\)
The Problem

\[ c_i = v_i, \ldots \circlearrowright \]

\[ S \supset K \vdash e \leftrightarrow a = M \]

- **Term-based representation**
  
  \[ \vdash \Sigma \ M : \text{eval} \backslash S \backslash K \backslash e \backslash a \]

  ... as before

- **Context-based representation**

  \[ \Gamma S \vdash \Sigma \ M : \text{eval} \backslash K \backslash e \backslash a \]

**This does not work!**

- \( S \) is subject to *destructive operations* (e.g. assignment)
- current logical frameworks do not allow removing assumptions from the context
Linear Logic in Brief

\[ \Gamma; \Delta \vdash A \]

- **Logical assumptions**
- **Resources**
- **Goal**

**Main resource operators**

- \( A \otimes B = \text{“} A \text{ and } B \text{ simultaneously”} \)
- \( A \& B = \text{“} A \text{ and } B \text{ alternatively”} \)
- \( T = \text{“} resource \text{ sink”} \)
- \( A \rightarrow B = \text{“} B \text{ assuming } A \text{ as a resource”} \)
- \( A \Rightarrow B = \text{“} B \text{ assuming } A \text{ as a logical hypothesis”} \)

Accessing a resource  
consumes it
A Simple Situation

\[
\begin{align*}
\$ &= \text{“I have one dollar”} \\
C' &= \text{“I buy a coke”} \\
F &= \text{“I buy French fries”}
\end{align*}
\]

\[
\begin{align*}
\$ \rightarrow C' &= \text{“With one dollar, I can buy a coke”} \\
\$ \rightarrow F &= \text{“With one dollar, I can buy French fries”}
\end{align*}
\]

\[
\begin{array}{c}
\$ \rightarrow C, \$ \rightarrow F, \$ \vdash C \land F
\end{array}
\]

“With one dollar, I can buy both a coke and French fries” !!
Propositions vs. Resources

$ \rightarrow C$ and $\rightarrow F$ are propositions (logical assumptions)

- either true or false
- accessible as many times as needed

$\$ is a resource

- either available or consumed
- once consumed, it cannot be used again

Note: the derivation is uncontroversial if we have only propositions

$ss = \text{“the sun shines”}$

$sg = \text{“I wear sunglasses”}$

$ic = \text{“I crave ice-cream”}$

$$ss \rightarrow sg, ss \rightarrow ic, ss \vdash sg \land ic$$
**Linear Logic**

\[ \Gamma; \Delta \vdash A \]

**Logical assumptions**

**Resources**

**Goal**

**Resource operators**

- \( \land \rightarrow \otimes \quad A \otimes B = "A \text{ and } B \text{ simultaneously}" \)
- \( \rightarrow \rightarrow \rightarrow \neg \quad A \neg B = "B \text{ assuming } A \text{ as a resource}" \)

\[
\begin{align*}
\Gamma; \cdot & \vdash \$ \multimap C \\
\Gamma; \$ & \vdash \$ \\
\hline
\Gamma; \$ & \vdash C \\
\Gamma; \$ & \vdash F \\
\hline
\$ \multimap C, \$ \multimap F \vdash \$; \$, & \vdash C \otimes F
\end{align*}
\]
A Step Back

\[ \$ \rightarrow C, \$ \rightarrow F, \$ \vdash C \land F \]

can also be interpreted as

"With one dollar, I can buy a coke and french fries, but not at the same time"

More resource operators

\[ \bullet \land \implies \& \quad A \& B = "A \text{ and } B \text{ alternatively}" \]

\[ \begin{array}{llll}
\Gamma ; \cdot \vdash \$ \leftarrow C & \Gamma ; \$ \vdash \$ & \Gamma ; \cdot \vdash \$ \leftarrow F & \Gamma ; \$ \vdash \$
\\
\hline
& \Gamma ; \$ \vdash C & \Gamma ; \$ \vdash F & \\hline
\end{array} \]

\[ \Gamma \vdash \$ \leftarrow C, \$ \leftarrow F; \$ \vdash C \& F \]
Linear Operators

Context splitting $\implies$ multiplicatives
Context sharing $\implies$ additives
Some Inference Rules

\[ \Gamma, A; \Delta \vdash A \]

\[ \Gamma; A \vdash A \]

\[ \frac{\Gamma, A; \Delta \vdash B}{\Gamma; \Delta \vdash A \rightarrow B} \rightarrow I \]

\[ \frac{\Gamma; \Delta \vdash A \rightarrow B \quad \Gamma; \cdot \vdash A}{\Gamma; \Delta \vdash B} \rightarrow E \]

\[ \frac{\Gamma; \Delta, A \vdash B}{\Gamma; \Delta \vdash A \rightarrow \circ B} \circ I \]

\[ \frac{\Gamma; \Delta_1 \vdash A \rightarrow \circ B \quad \Gamma; \Delta_2 \vdash A}{\Gamma; \Delta_1, \Delta_2 \vdash B} \circ E \]

\[ \frac{\Gamma; \Delta \vdash A \quad \Gamma; \Delta \vdash B}{\Gamma; \Delta \vdash A \& B} \& I \]

\[ \frac{\Gamma; \Delta \vdash A \& B}{\Gamma; \Delta \vdash A} \& E_1 \]

\[ \frac{\Gamma; \Delta \vdash A \& B}{\Gamma; \Delta \vdash B} \& E_2 \]

\[ \frac{\Gamma; \Delta \vdash \top}{\Gamma; \Delta \vdash \top} \top I \]
Exponentials

Observe that ∧ corresponds to both ⊗ and & when the resource context is **empty**.

The same holds for all connectives except →

$$\frac{\Gamma; \Delta \vdash B}{\Gamma; \Delta \vdash A \rightarrow B} \rightarrow I$$

$$\frac{\Gamma; \Delta \vdash A \rightarrow B \Gamma; \cdot \vdash A}{\Gamma; \Delta \vdash B} \rightarrow E$$

Can we get rid of →? We do not want to, but we can:

Interprete logical assumptions as *inexhaustible resources*

!A = “as many copies of A as you wish”

$$\frac{\Gamma, A; \Delta \vdash C}{\Gamma; \Delta, !A \vdash C} \iff$$

$$A \rightarrow B \iff (!A) \rightarrow B$$
Observations

• Linear logic is a **conservative extension** of traditional logic:
  The natural translation of judgments maintains:
  – derivability
  – derivations

• Direct representation of resources
Meta-language: $\lambda^{\Pi \rightarrow \& \top}$, a type theory based on $\Pi$, $\rightarrow$, $\&$ and $\top$

Representation methodology: judgments-as-types, but provides direct encoding of state in the linear context

Range of applicability: declarative and imperative formalisms
\( \lambda^{\Pi \vdash \circ \& \top}, \text{ the Meta-Language of } LLF \)

- **Syntax**

  \[
  \begin{align*}
  Kinds & \quad K := \text{type} \mid \Pi x: A. K \\
  Type \ families & \quad P := a \mid P M \\
  Types & \quad A := P \mid \Pi x: A. B \mid A \rightarrow B \mid A \& B \mid \top \\
  Objects & \quad M := x \mid c \mid \lambda x: A. M \mid M N \mid \hat{x}: A. M \mid M^\triangleleft N \mid \langle M, N \rangle \mid \text{FST } M \mid \text{SND } M \mid \langle \rangle
  \end{align*}
  \]

- **Semantics**

  \[
  \Gamma; \Delta \vdash \Sigma M : A
  \]
  
  "\( M \) has type \( A \) in \( \Gamma, \Delta \) and \( \Sigma \)"

  - Linear context: \( x ? A, \ldots \)
  - Intuitionistic context: \( x : A, \ldots \)
  - Signature: \( a : K, \ldots, c : A, \ldots \)
$\chi^{\Pi \vdash o & T}$, Some Inference Rules

\[ \Gamma, x : A; \Delta \vdash x : A \quad \text{ivar} \]
\[ \Gamma, x : A; \Delta \vdash x : A \quad \text{ivar} \]

\[ \Gamma, x : A; \Delta \vdash M : B \]
\[ \Gamma ; \Delta \vdash \lambda x : A. M : \Pi x : A. B \quad \text{ilam} \]
\[ \Gamma ; \Delta \vdash \lambda x : A. M : A \to B \quad \text{ilam} \]
\[ \Gamma ; \Delta \vdash \lambda x : A. M : A \to B \]
\[ \Gamma ; \Delta \vdash M N : [N/x]B \quad \text{iapp} \]
\[ \Gamma ; \Delta \vdash M N : [N/x]B \]
\[ \Gamma ; \Delta \vdash M N : [N/x]B \]

\[ \Gamma ; \Delta \vdash M : A \]
\[ \Gamma ; \Delta \vdash N : B \]
\[ \Gamma ; \Delta \vdash \langle M, N \rangle : A \& B \quad \text{pair} \]
\[ \Gamma ; \Delta \vdash \langle M, N \rangle : A \& B \]
\[ \Gamma ; \Delta \vdash \langle M, N \rangle : A \& B \]
\[ \Gamma ; \Delta \vdash \langle M, N \rangle : A \& B \]
\[ \Gamma ; \Delta \vdash \langle M, N \rangle : A \& B \]

\[ \Gamma ; \Delta \vdash \langle \rangle : \top \quad \text{unit} \]
\[ \Gamma ; \Delta \vdash \langle \rangle : \top \]
\[ \Gamma ; \Delta \vdash \langle \rangle : \top \]
Lemma (Church-Rosser property)

If $M_1 \equiv M_2$, then there is $N$ such that $M_1 \rightarrow^* N$ and $M_2 \rightarrow^* N$

Lemma (strong normalization)

If $\Gamma; \Delta \vdash_{\Sigma} M : A$ is derivable, then $M$ is strongly normalizing

Theorem (canonical forms)

If $\Gamma; \Delta \vdash_{\Sigma} M : A$, then there exist a unique term $N$ in canonical form such that $M \rightarrow^* N$ and $\Gamma; \Delta \vdash_{\Sigma} N : A$
Immediacy in $LLF$

Direct correlation between an object system and its encoding

$LLF$ gives direct support to recurrent representation patterns

- binding constructs via $\lambda$-abstraction
- derivations as proof-terms
- state manipulation via linear constructs
Computational Properties of $LLF$

- Allows automatic proof verification

**Theorem** (*decidability of type checking*)

It can be recursively decided whether there exist a derivation for the judgment

$\Gamma; \Delta \vdash_{\Sigma} M : A$

- Supports proof search

**Theorem** (*abstract logic programming language*)

$\lambda^\Pi_{=\circ \& \top}$ is an *abstract logic programming language*
**LLF, Summary**

- combines the meta-reasoning power of logical frameworks with the ability of handling state of linear logic
- is a conservative extension of the logical framework *LF*

**Theorem** (*conservativity over LF*)

If \( \Gamma, M \) and \( A \) do not mention linear constructs, \( \Gamma; \cdot \vdash_{\Sigma} M : A \) is derivable in *LLF* [iff] \( \Gamma \vdash_{\Sigma} M : A \) is derivable in *LF*

- can be implemented as a linear logic programming language
- has been used for the representation of
  - imperative programming languages
  - non-traditional logics
  - languages with non-standard binders
  - puzzles and solitaires
  - planning
  - imperative graph search
Case Study: MLR

MLR is a fragment of ML with

- references
- value polymorphism
- recursion

\[
\text{Types } \tau ::= \ldots \mid 1 \mid \tau_1 \rightarrow \tau_2 \mid \tau \text{ ref}
\]

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

\[
\text{Store } S ::= \cdot \mid S, c = v
\]
**MLR: Typing**

\[ \Omega \vdash e : \tau \quad \text{“e has type } \tau \text{ in } \Omega \]  

**Context**  \[ x_i : \tau_i, \ldots, c_j : \sigma_j, \ldots \]  

**Expression**  

**Type**

**Representation:**  

\[ \mathcal{J}_\Omega \vdash_{\Sigma} \mathcal{J}_\tau : \text{exp_type} \mathcal{J}_e \mathcal{J}_\tau \]

\[ x_i : \text{exp}, \ t_i : \text{exp_type} \ x_i \mathcal{J}_{\tau_i}, \ldots \]  

\[ c_j : \text{cell}, \ l_j : \text{cell_type} \ c_j \mathcal{J}_{\sigma_j}, \ldots \]

\[ \Omega \vdash e_1 : \tau \text{ ref} \quad \Omega \vdash e_2 : \tau \quad \begin{array}{c} \text{et_assign} \\ \hline \end{array} \quad \Omega \vdash e_1 := e_2 : 1 \]

\[ \Omega \vdash !e : \tau \quad \begin{array}{c} \text{et_deref} \\ \hline \end{array} \quad \Omega \vdash !e : \tau \]

\[ \begin{array}{l} \text{et_assign} : \text{exp_type E1 (rf T)} \\
\quad \rightarrow \text{exp_type E2 T} \\
\quad \rightarrow \text{exp_type (assign E1 E2) 1.} \end{array} \]

\[ \begin{array}{l} \text{et_deref} : \text{exp_type E (rf T)} \\
\quad \rightarrow \text{exp_type (deref E) T.} \end{array} \]
**MLR: Evaluation**

Continuation
\[ \text{init}, \ldots, \lambda x. i, \ldots \]

Instruction
\[ \text{eval } e, \text{ return } v, \ldots \]

\[ S \triangleright K \vdash i \mapsto a \]

“\( i \) followed by \( K \) evaluates to \( a \), starting from \( S \)”

Store
\[ c_i = v_i, \ldots \]

Answer

**Representation:**
\[ \vdash S \vdash \Sigma \vdash \mathcal{E} \vdash \text{eval } \vdash K \vdash i \vdash a \]

\[ c_i: \text{cell}, \quad h_i \text{ contains } c_i \vdash v_i, \ldots \]

Iliano Cervesato — *A Linear Logical Framework*
MLR: Some Imperative Rules

\[ S', c = v, S'' \triangleright K \vdash \text{return} \langle \rangle \rightarrow a \]
\[ S', c = v', S'' \triangleright K \vdash c := v \rightarrow a \]  

\text{ev\_assign} :  (contains C V  \ -o eval K (return unit) A) 
\ -o (contains C V'  \ -o eval K (assign2 (loc C) V) A).

\[ S', c = v, S'' \triangleright K \vdash \text{return} v \rightarrow a \]  
\[ S', c = v, S'' \triangleright K \vdash !c \rightarrow a \]  

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

\text{rd} :  contains C V  
\ -o <T>  
\ -o read C V.
**MLR: Adequacy**

**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 $\mathcal{E}$ of

$$S \triangleright K \vdash i \leftrightarrow a$$

and canonical $LLF$ objects $M$ such that

$$\triangledown S \triangleright \Sigma M : \text{eval} \triangledown i \triangledown a$$

is derivable, where

$$\triangledown S \triangleright = \left[\begin{array}{c}
    c_1:\text{cell}, \ h_1:\text{contains } c_1 \triangledown v_1 \\
    \ldots \\
    c_n:\text{cell}, \ h_n:\text{contains } c_n \triangledown v_n
\end{array}\right]$$
MLR: Type Preservation

- Functional core: implemented in $LF$ [Michaylov, Pfenning’91]
- References [Tofte’90; Harper’94]: implemented in $LLF$ [Cervesato’96]

**Theorem** (*type preservation*)

If $S \triangleright K \vdash i \leftrightarrow a$, with $\Omega \vdash i : \tau$, $\Omega \vdash K : \tau \Rightarrow \sigma$ and $\Omega \vdash S : \Omega$, then $\Omega \vdash a : \sigma$

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

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

- each case yields one declaration
- the meta-reasoning is itself *linear*

**Representation**

\[
\text{tpev : eval K I A } \to \text{ cont_type K T S } \to \text{ instr_type I T } \to \text{ ans_type A S } \to \text{ type.}
\]
Future Developments: Implementation

Indispensable for tackling larger applications

- **Interpreter**
  - context management [Hodas, Miller’94; Cervesato, Hodas, Pfenning’96]
  - unification [Cervesato, Pfenning’96]
  - term reconstruction

- **Compiler**
  - WAM [Warren’83]
  - embedded implication/quantification [Nadathur, Jayaraman, Kwon’95]
  - types [Kwon, Nadathur, Wilson’91]
  - higher-order unification
  - proof-terms
  - linearity
**Future Developments: Applications**

- Specification and verification of
  - real-world programming languages (e.g. *SML ’96, Java*)
  - communication protocols
  - logics

- Proof-Carrying Code [Necula’97]
  Use of logical frameworks technology to determine that it is safe to execute code provided by an untrusted producer
  - user extensions to the kernel of the operating system
  - mobile code in distributed/Web computing
  - foreign code extensions to a safe programming language

  *LLF can provide a direct handling of resources and a better representation of memory*
(courtesy George Nechaia)
Future Developments: Miscellaneous

- Type theoretic extensions of $LLF$ (e.g. dependent linear types, non-commutativity)

- Computer-assisted development environments for logics and programming languages (schema checking [Pfenning,Rohwedder’96], meta-logical frameworks [Schürmann’95])

- Educational software for logic and the theory of programming languages