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]
- Categorial 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
  – $\lambda$-lifting [Leone]
  – lazy evaluation [Okasaki]
  – monads [Gehrke’95]
Meta-Language

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

- Type theories
  - \(\lambda^\Pi\) (*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])
  - \(\lambda^{\Pi-\o&\top}\) (*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}
\text{\(x_i : \tau_i, \ldots\)} \\
\vdash \quad \tau \\
\Omega \vdash e : \tau & = & M
\end{array}
\]

• Term-based representation

\[
\begin{array}{c}
\vdash_{\Sigma} M : \text{has type} \quad \Omega \vdash e \vdash \tau
\end{array}
\]

We must encode \emph{explicitly}

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

\[ x_i : \tau_i, \ldots \vdash \tau \downarrow \]
\[ \Omega \vdash e : \tau = M \]

- Exploitation of the meta-language context

\[ \llbracket \Omega \rrbracket \vdash \Sigma \quad M : \text{has\_type} \quad \llbracket e \rrbracket \llbracket \tau \rrbracket \]

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

\[ \llbracket x_i : \tau_i \rrbracket = x_i : \text{exp}, \quad t_i : \text{has\_type} \quad x_i \llbracket \tau_i \rrbracket \]

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

- **Syntax**

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

- **Semantics**

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

  \begin{align*}
  \text{Context} & \quad x : A, \ldots \\
  \text{Signature} & \quad a : K, \ldots, c : A, \ldots
  \end{align*}
\[ \begin{align*}
\Gamma, x : A &\vdash \Sigma M : B \\
\Gamma &\vdash \Sigma \lambda x : A. M : \Pi x : A. B \\
\Gamma &\vdash \Sigma M : \Pi x : A. B \\
\Gamma &\vdash \Sigma N : A
\end{align*} \]

\[ \begin{align*}
\text{lam} & \quad \text{app}
\end{align*} \]

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

\[ c_i = v_i, \ldots \quad \vdash \quad E \quad \vdash \quad S \triangleright K \vdash e \leftrightarrow a = M \]

- **Term-based representation**
  
  \[ \vdash_{\Sigma} M : \text{eval} \quad \vdash S \quad \vdash K \quad \vdash e \quad \vdash a \]
  
  ... as before

- **Context-based representation**

  \[ \vdash S \quad \vdash_{\Sigma} M : \text{eval} \quad K \quad \vdash e \quad \vdash a \]

**This does not work!**

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

Iliano Cervesato — *A Linear Logical Framework*
Linear Logic in Brief

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

Accessing a resource consumes it

Main resource operators

- \( A \otimes B \) = “A and B simultaneously”
- \( A \& B \) = “A and B alternatively”
- \( \top \) = “resource sink”
- \( A \rightarrow_0 B \) = “B assuming \( A \) as a resource”
- \( A \rightarrow B \) = “B assuming \( A \) as a logical hypothesis”
A Simple Situation

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

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

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

\[ $ \vdash F \]

\[ $ \rightarrow C \]

\[ $ \vdash C \land F \]

"With one dollar, I can buy both a coke \textbf{and} French fries" !!
**Propositions vs. Resources**

\[ \$ \to C \text{ and } \$ \to F \text{ are propositions (logical assumptions)} \]

- either *true* or *false*

- accessible as many times as needed

\[ \$ \text{ is a resource} \]

- either *available* or *consumed*

- once consumed, it cannot be used again

**Note:** the derivation is uncontroversial if we have only propositions

\[
ss = \text{“the sun shines”} \\
sg = \text{“I wear sunglasses”} \\
in = \text{“I crave ice-cream”}
\]

\[
ss \to sg, ss \to in, ss \vdash sg \land ic
\]
**Linear Logic**

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

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

**Resource operators**

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

\[
\frac{\Gamma; \cdot \vdash \$ \multimap C \quad \Gamma; \bullet \vdash \$ \quad \Gamma; \cdot \vdash \$ \multimap F \quad \Gamma; \bullet \vdash \$}{\Gamma; \bullet \vdash C \quad \Gamma; \bullet \vdash F \quad \$ \multimap C, \$ \multimap F; \$, \$ \vdash C \otimes F}
\]

Ilion Cervesato — A Linear Logical Framework
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

- $\land \implies \& \quad A \& B = “A \text{ and } B \text{ alternatively}”$

\[
\begin{array}{c}
\Gamma; \cdot \vdash \$ \rightarrow C \quad \Gamma; \$ \vdash \$ \\
\Gamma; \$ \vdash C \\
\hline
\Gamma; \$ \vdash F \\
\hline
\end{array}
\]

\[
\begin{array}{c}
\Gamma; \$ \vdash C \land F; \$ \vdash C \& F \\
\hline
\end{array}
\]
Linear Operators

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

\[ \Gamma, A; \vdash A \quad \text{int} \]

\[ \Gamma; A \vdash A \quad \text{lin} \]

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

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

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

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

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

\[ \frac{\Gamma; \Delta \vdash A \& B}{\Gamma; \Delta \vdash A \quad \& \text{E}_1} \quad \frac{\Gamma; \Delta \vdash A \& B}{\Gamma; \Delta \vdash B \quad \& \text{E}_2} \]

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

Observe that $\land$ corresponds to both $\otimes$ and $\&$ when the resource context is empty. The same holds for all connectives except $\rightarrow$

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

Can we get rid of $\rightarrow$? We do not want to, but we can:

Interprete logical assumptions as *inexhaustible resources*

$!A = "as many copies of A as you wish"

\[
\Gamma, A; \Delta \vdash C \quad \iff \quad \Gamma; \Delta, !A \vdash C
\]

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

- Linear logic is a **conservative extension** of traditional logic:
  - The natural translation of judgments maintains:
    - derivability
    - derivations
  - Direct representation of resources
\textbf{LLF}

- **Meta-language:** $\lambda^{\Pi-\&-\top}$, a type theory based on $\Pi$, $\&$, $\top$

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

- **Range of applicability:** declarative and \textcolor{red}{imperative} formalisms
\( \lambda^{\Pi \rightarrow \& \top} \), the Meta-Language of LLF

- **Syntax**

  \textit{Kinds} \quad K := \text{type} \mid \Pi x : A. K

  \textit{Type families} \quad P := a \mid P M

  \textit{Types} \quad A := P \mid \Pi x : A. B \mid A \to B \mid A \& B \mid \top

  \textit{Objects} \quad M := x \mid c \mid \lambda x : A. M \mid M N \mid \hat{\lambda} x : A. M \mid \lambda^{\infty} N \mid \langle M, N \rangle \mid \text{FST } M \mid \text{SND } M \mid \langle \rangle

- **Semantics**

  Linear context
  \[
  \Gamma; \Delta \vdash_{\Sigma} M : A
  \]

  “\( M \) has type \( A \) in \( \Gamma, \Delta \) and \( \Sigma \)”

  Intuitionistic context
  \[
  x : A, \ldots
  \]

  Signature
  \[
  a : K, \ldots, c : A, \ldots
  \]
Some Inference Rules

\[ \lambda^{\Pi-\sigma} \]

\begin{align*}
\Gamma, x : A; \cdot \vdash \Sigma x : A & \quad \text{ivar} \quad \Gamma; x \diamond A \vdash \Sigma x : A & \quad \text{ivar} \\
\Gamma, x : A; \Delta \vdash \Sigma M : B & \implies \Gamma; \Delta \vdash \Sigma \lambda x : A. M : \Pi x : A. B & \quad \text{lam} \\
\Gamma; \Delta \vdash \Sigma \hat{\lambda} x : A. M : A - \sigma B & \quad \llam \\
\Gamma; \Delta \vdash \Sigma M : A & \quad \Gamma; \Delta \vdash \Sigma N : B & \quad \text{pair} \\
\Gamma; \Delta \vdash \Sigma \langle M, N \rangle : A \& B & \quad \Gamma; \Delta \vdash \Sigma M : A \& B & \quad \text{fst} \\
\Gamma; \Delta \vdash \Sigma \text{FST} M : A & \quad \Gamma; \Delta \vdash \Sigma \text{SND} M : B & \quad \text{fst} \\
\Gamma; \Delta \vdash \Sigma \langle \rangle : \top & \quad \text{unit} \\
\end{align*}
\( \lambda \Pi_{\L}^{\bullet} \& \top \): Main Properties

**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 λ-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*)

\[ \lambda^{\Pi_{\omega & \top}} \; is \; an \; abstract \; logic \; programming \; language \]
**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

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

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

\[
\text{Store} \quad S ::= \cdot \mid S, c = v
\]
\[ \Omega \vdash e : \tau \quad \text{“\( e \) has type \( \tau \) in \( \Omega \)”} \]

**Representation:**

\[ \Gamma \vdash \Sigma \quad \Gamma \vdash \text{exp\_type} \quad \Gamma \vdash e \quad \Gamma \vdash \tau \]

\[ x_i: \text{exp}, \quad t_i: \text{exp\_type} \quad x_i \Gamma \vdash \tau_i, \quad ... \]

\[ c_j: \text{cell}, \quad l_j: \text{cell\_type} \quad c_j \Gamma \vdash \sigma_j, \quad ... \]

**Inference Rules:**

\[ \Omega \vdash e_1 : \tau \quad \text{ref} \quad \Omega \vdash e_2 : \tau \]

\[ \frac{}{\Omega \vdash e_1 \equiv e_2 : 1} \quad \text{et\_assign} \]

\[ \Omega \vdash e : \tau \quad \text{ref} \quad \text{et\_deref} \]

\[ \Omega \vdash \text{et\_assign} : \text{exp\_type E1 (rf T)} \]

\[ \quad \rightarrow \text{exp\_type E2 T} \]

\[ \quad \rightarrow \text{exp\_type (assign E1 E2) 1.} \]

\[ \Omega \vdash \text{et\_deref} : \text{exp\_type E (rf T)} \]

\[ \quad \rightarrow \text{exp\_type (deref E) T}. \]
**MLR: Evaluation**

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

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

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

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

**Store**
\[ c_i = v_i, \ldots \]

**Answer**

**Representation:**
\[ \Gamma \vdash_\Sigma \Gamma \mathcal{E} \vdash \text{eval } \Gamma K \vdash_i \Gamma a \]

\[ c_i : \text{cell, } h_i \uparrow \text{contains } c_i \Gamma v_i, \ldots \]
**MLR: Some Imperative Rules**

\[
\begin{align*}
S', c = v, S'' \triangleright K & \vdash \text{return } \langle \rangle \leftrightarrow a \quad \text{ev_assign} \\
S', c = v', S'' \triangleright K & \vdash c := v \leftrightarrow a
\end{align*}
\]

```
ev_assign :  (contains C V  -o  eval K (return unit) A) \\
            -o  (contains C V' -o  eval K (assign2 (loc C) V) A).
```

\[
\begin{align*}
S', c = v, S'' \triangleright K & \vdash \text{return } v \leftrightarrow a \quad \text{ev_deref} \\
S', c = v, S'' \triangleright K & \vdash !c \leftrightarrow a
\end{align*}
\]

```
ev_deref :  read C V \\
            & eval K (return V) A \\
            -o eval K (ref1 (loc C)) A.
```

```
rd :  contains C V \\
      -o <T> \\
      -o read C V.
```

Iliano Cervesato — A Linear Logical Framework
**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

\[
\Gamma S \vdash_{\Sigma} M : \text{eval} \Gamma i \Gamma a
\]

is derivable, where

\[
\Gamma S = \begin{cases}
  c_1: \text{cell}, & h_1 \hat{\text{contains}} c_1 \Gamma v_1 \\
  \ldots \\
  c_n: \text{cell}, & h_n \hat{\text{contains}} c_n \Gamma v_n
\end{cases}
\]
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**

\[
\text{tpenv : eval K I A} \rightarrow \text{cont_type K T S} \rightarrow \text{instr_type I T} \rightarrow \text{ans_type A S} \rightarrow \text{type.}
\]
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 Neguia
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