# The Linear Logical Framework LLF

Iliano Cervesato

Department of Computer Science Stanford University

(joint work with Frank Pfenning)

# **Contents**

- Overview
- Logical frameworks
- $\bullet$  LLF
- Case study
- Implementation
- Future Work

### Overview

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

#### Aim:

• identify the principles underlying logics and programming languages [Harper, Honsell, Plotkin'87; 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; Paulson'96]

#### **Limitations**:

• ineffective with imperative formalisms [Pfenning'94]

#### State

Till 2 years ago, **no** simple, general and effective treatment of the recurring notion of **state** 

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

#### ... Linear Logic [Girard'87]

- adequate for **representing** state and imperative computation [Chirimar'95; Hodas, Miller'94; Wadler'90]
- ineffective for **reasoning** about them

#### **Achievements**

- Design of a formalism, *LLF*, that combines
  - the meta-reasoning power of traditional logical frameworks
  - the possibility of linear logic of handling state
- Based on a linear type theory
- $\bullet$  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

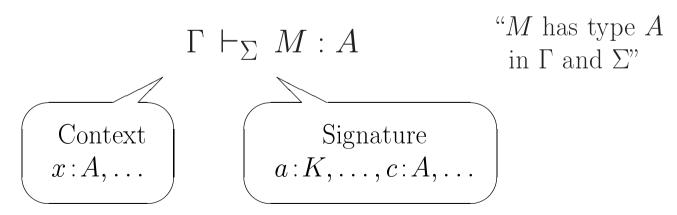
Logical framework = meta-language + representation methodology

# An Example: LF (Meta-Language)

#### • Syntax

$$Kinds$$
  $K::= type \mid \Pi x:A.K$ 
 $Type\ families$   $P::= a \mid PM$ 
 $Types$   $A::= P \mid \Pi x:A.B$ 
 $Objects$   $M::= x \mid c \mid \lambda x:A.M \mid MN$ 

#### • Typing judgment



# An Example: *LF* (Meta-Language—Cont'd)

$$\frac{\Gamma, x : A \vdash_{\Sigma} M : B}{\Gamma \vdash_{\Sigma} \lambda x : A \cdot M : \Pi x : A \cdot B} \text{ lam}$$

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

#### • Main properties

- is strongly normalizing
- admits unique canonical forms
- type checking is decidable
- can be implemented as a logic programming language (Elf [Pfenning'94])

# An Example: *LF* (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 metalanguage
- 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

# An Example: *LF* (Representation Methodology—Cont'd)

$$\begin{array}{ccc}
x_i:\tau_i,\dots \\
& \mathcal{T} \\
& \Omega \vdash e:\tau
\end{array} = M$$

$$\lceil \Omega \rceil \vdash_{\Sigma} M : \mathtt{has\_type} \lceil e \rceil \lceil \tau \rceil$$

where for each  $x_i:\tau_i$  in  $\Omega$ ,

$$\lceil x_i : \tau_i \rceil = x_i : exp, \ t_i : has\_type \ x_i \lceil \tau_i \rceil$$

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

#### Problem!

#### This does not work!

- $\bullet$  S is subject to destructive operations (e.g. assignment)
- traditional log. frameworks do not allow removing assumptions from the context

#### A way out ...

$$\cdot \vdash_{\Sigma} M : \operatorname{eval} \ulcorner S \urcorner \ulcorner K \urcorner \ulcorner e \urcorner \ulcorner a \urcorner$$

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

#### LLF

- Meta-language:  $\lambda^{\Pi \multimap \& \top}$ , a type theory based on  $\Pi$ ,  $\multimap$ , & and  $\top$
- Representation methodology: judgments-as-types, but provides direct encoding of state in the linear context
- Range of applicability: declarative and imperative formalisms

# $\lambda^{\Pi = 0 \& \top}$ , the Meta-Language of *LLF*

#### • Syntax

$$Kinds \qquad K ::= \ \mathsf{type} \ | \ \Pi x : A. \ K$$
 
$$Type \ families \qquad P ::= \ a \ | \ P \ M$$
 
$$Types \qquad A ::= \ P \ | \ \Pi x : A. \ B \ | \ A \multimap B \ | \ A \& B \ | \ \top$$
 
$$Objects \qquad M ::= \ x \ | \ c \ | \ \lambda x : A. \ M \ | \ M \ N \ | \ | \ \mathsf{FST} \ M \ | \ \mathsf{SND} \ M \ | \ \langle \rangle$$

#### • Typing judgment

Linear context  $x : A, \dots$ "M has type A $\Gamma; \Delta \vdash_{\Sigma} M : A$ in  $\Gamma$ ,  $\Delta$  and  $\Sigma$ " Intuitionistic context Signature  $a:K,\ldots,c:A,\ldots$  $x:A,\ldots$ 

# $\lambda^{\Pi - 0 \& \top}$ , Some Inference Rules

$$\frac{1}{\Gamma, x : A; \cdot \vdash_{\Sigma} x : A} \text{ ivar} \qquad \frac{\Gamma; x : A \vdash_{\Sigma} x : A}{\Gamma} \text{ lvar}$$

$$\frac{1}{\Gamma; x \hat{\cdot} A \vdash_{\Sigma} x : A}$$
 lvar

$$\frac{\Gamma, x : A; \Delta \vdash_{\Sigma} M : B}{\Gamma; \Delta \vdash_{\Sigma} \lambda x : A, M : \Pi x : A, B}$$
ilam

$$\frac{\Gamma, x : A; \Delta \vdash_{\Sigma} M : B}{\Gamma; \Delta \vdash_{\Sigma} \lambda x : A. M : \Pi x : A. B} \mathbf{ilam} \qquad \frac{\Gamma; \Delta \vdash_{\Sigma} M : \Pi x : A. B \quad \Gamma; \cdot \vdash_{\Sigma} N : A}{\Gamma; \Delta \vdash_{\Sigma} M N : [N/x]B} \mathbf{iapp}$$

$$\frac{\Gamma; \Delta, x : A \vdash_{\Sigma} M : B}{\Gamma; \Delta \vdash_{\Sigma} \hat{\lambda}x : A. M : A \multimap B}$$
llam

$$\frac{\Gamma; \Delta, x \mathbin{\widehat{:}} A \vdash_{\Sigma} M : B}{\Gamma; \Delta \vdash_{\Sigma} \hat{\lambda} x : A. M : A \multimap B} \text{ llam} \qquad \frac{\Gamma; \Delta_1 \vdash_{\Sigma} M : A \multimap B \quad \Gamma; \Delta_2 \vdash_{\Sigma} N : A}{\Gamma; \Delta_1, \Delta_2 \vdash_{\Sigma} M \mathbin{\widehat{\cdot}} N : B} \text{ iapp}$$

$$\frac{\Gamma; \Delta \vdash_{\Sigma} M : A \quad \Gamma; \Delta \vdash_{\Sigma} N : B}{\Gamma; \Delta \vdash_{\Sigma} \langle M, N \rangle : A \& B} \text{ pair }$$

$$\frac{\Gamma; \Delta \vdash_{\Sigma} M : A \quad \Gamma; \Delta \vdash_{\Sigma} N : B}{\Gamma; \Delta \vdash_{\Sigma} \langle M, N \rangle : A \& B} \, \mathbf{pair} \qquad \qquad \frac{\Gamma; \Delta \vdash_{\Sigma} M : A \& B}{\Gamma; \Delta \vdash_{\Sigma} \mathbf{FST} M : A} \, \mathbf{fst} \quad \frac{\Gamma; \Delta \vdash_{\Sigma} M : A \& B}{\Gamma; \Delta \vdash_{\Sigma} \mathbf{SND} M : B} \, \mathbf{snd}$$

$$\Gamma; \Delta \vdash_{\Sigma} \langle \rangle : \top$$
 unit

# LLF, Main Properties

- Church-Rosser property
- strongly normalizing
- unique canonical forms
- decidability of type checking
- abstract logic programming language
- $\bullet$  conservative over LF

# Immediacy in *LLF*

Direct correlation between an object system and its encoding

LLF gives direct support to recurrent representation patterns

- $\bullet$  binding constructs via  $\lambda$ -abstraction
- derivations as proof-terms
- state manipulation via linear constructs

# Case Study: MLR

#### MLR is a fragment of ML with

- references
- value polymorphism
- recursion

```
Types \ \tau \ ::= \ \ldots \ | \ 1 \ | \ \tau_1 \rightarrow \tau_2 \ | \ \tau \ \mathbf{ref} Expressions \ e \ ::= \ x \ | \ \langle \rangle \ | \ \mathbf{lam} \ x. \ e \ | \ e_1 \ e_2 \ | \ \ldots \ | \ c \ | \ \mathbf{ref} \ e \ | \ !e \ | \ e_1 := e_2 Store \ S \ ::= \ \cdot \ | \ S, c = v
```

```
Expressions

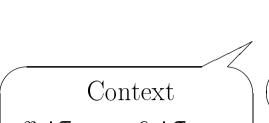
exp : type.
cell : type.

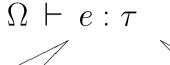
unit : exp.
lam : (exp -> exp) -> exp.
app : exp -> exp -> exp.

loc : cell -> exp.
ref : exp -> exp.
deref : exp -> exp.
assign : exp -> exp -> exp.
```

# MLR: Typing

Type





"e has type  $\tau$  in  $\Omega$ "



#### Representation:

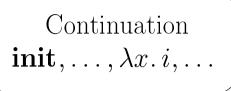
$$\lceil \Omega \rceil \vdash_{\Sigma} \lceil \mathcal{T} \rceil : exp\_type \lceil e \rceil \lceil \tau \rceil$$

$$x_i$$
:exp,  $t_i$ :exp\_type  $x_i \lceil \tau_i \rceil$ , ...  $c_j$ :cell,  $l_j$ :cell\_type  $c_j \lceil \sigma_j \rceil$ , ...

$$\frac{\Omega \vdash e_1 : \tau \text{ ref } \Omega \vdash e_2 : \tau}{\Omega \vdash e_1 := e_2 : \mathbf{1}} \text{ et\_assign}$$

$$\frac{\Omega \vdash e : \tau \text{ ref}}{\Omega \vdash !e : \tau} \text{ et\_deref}$$

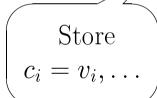
#### MLR: Evaluation



Instruction  $\mathbf{eval}\ e,$   $\mathbf{return}\ v, \dots$ 

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

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



Answer

Representation:

$$\lceil S \rceil \vdash_{\Sigma} \lceil \mathcal{E} \rceil : \operatorname{eval} \lceil K \rceil \lceil i \rceil \lceil a \rceil$$

 $c_i$ :cell,  $h_i$ :contains  $c_i \, \lceil v_i \rceil$ , ...

# MLR: Some Imperative Rules

$$\frac{S', c = v, S'' \rhd K \vdash \mathbf{return} \; \langle \rangle \hookrightarrow a}{S', c = v', S'' \rhd K \vdash c := v \hookrightarrow a} \text{ ev\_assign}$$

$$\frac{S', c = v, S'' \triangleright K \vdash \mathbf{return} \ v \hookrightarrow a}{S', c = v, S'' \triangleright K \vdash !c \hookrightarrow a} \text{ ev\_deref}$$

```
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.
```

# MLR: Adequacy

#### Adequacy theorem (Evaluation)

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

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

and canonical LLF objects M such that

$$\lceil S \rceil \vdash_{\Sigma} M : \operatorname{eval} \lceil K \rceil \lceil i \rceil \lceil a \rceil$$

is derivable, where

$$\lceil S \rceil = \begin{bmatrix} c_1 : \texttt{cell}, & h_1 : \texttt{contains} & c_1 \lceil v_1 \rceil \\ & \dots \\ c_n : \texttt{cell}, & h_n : \texttt{contains} & c_n \lceil v_n \rceil \end{bmatrix}$$

## 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 \hookrightarrow 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  $\mathbf{transcribing}$  this proof into an LLF specification capturing its computational contents

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

#### Representation

```
tpev : eval K I A -> cont_type K T S -> instr_type I T -> ans_type A S -> type.
```

# **Implementation**

LLF is implemented as part of the Twelf project

- Twelf, a successor to *Elf* [Pfenning'94]
  - higher-order constraint logic programming language based on LF and LLF
  - automated theorem prover in a meta-logic for LF [Schürmann,Pfenning'98]
  - internals: explicit substitutions, spine calculus, compilation

#### • Linear aspects

- linearity check
- resource management [Hodas, Miller'94; Cervesato, Pfenning'96]
- linear unification [Cervesato, Pfenning'97]

# LLF, Summary

- combines the meta-reasoning power of logical frameworks with the ability of handling state of linear logic
- ullet conservative extension of the logical framework LF
- implemented as a linear logic programming language
- used for the representation of
  - imperative programming languages
  - substructural and modal logics
  - puzzles and solitaires
  - planning
  - imperative graph search

#### **Future Work**

- Specification and verification of
  - "real" programming languages (e.g. SML'97, Java)
  - communication protocols
  - logics
- Proof-Carrying Code [Necula'97]
- Computer-assisted development environments for logics and programming languages (meta-logical frameworks)
- Type theoretic extensions of LLF (e.g. dependent linear types [Ishtiaq,Pym'97], non-commutativity [Pfenning'98])