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 $\lambda^{\Pi\text{ -o }\&\ \top}$
  - 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)
  - ...
- ...

Structure of a logical framework

Logical framework

=

meta-representation language

+

meta-representation methodology

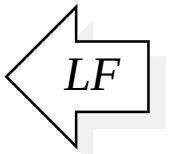
Meta-representation languages

Logics

- Horn clauses (*Prolog*)
- Higher-order hereditary Harrop formulas (λ*Prolog*, *Isabelle*)
- Classical linear logic (*Forum*)
- ...

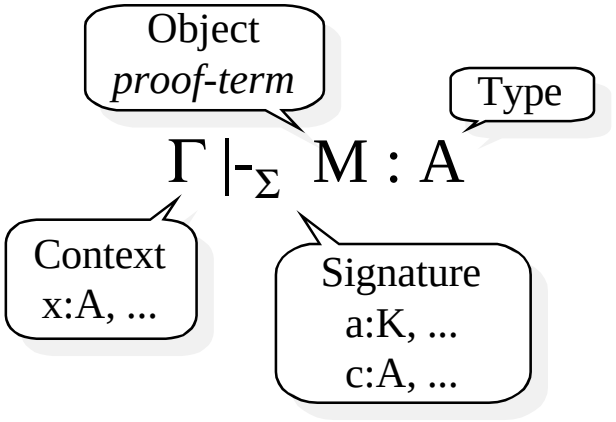
Type theories

- λ^{Π} (*Elf*)
- CIC (*Coq*, *Lego*)
- Martin-Löf's type theories (*ALF*, *NuPrl*)
- ...



The type theory λ^{Π}

Kinds	K ::= `type` | $\Pi x{:}A.\ K$
Type families	P ::= a | P M
Types	A ::= P | $\Pi x{:}A_1.\ A_2$
Objects	M ::= x | c | $\lambda x{:}A.\ M$ | $M_1\ M_2$



"M has type A in Γ and Σ"

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 inhabitance of a type

Meta-representation methodology

Context!!! $x_i : \tau_i, \ldots$

$$\ulcorner \quad \mathcal{T} \quad \urcorner \qquad \ulcorner \quad \urcorner$$
$$\Delta \rightarrow e : \tau \qquad = \qquad M$$

- **Term-based representation**

$$\cdot \mid\!-_\Sigma \; M : \texttt{ofe} \; \ulcorner\Delta\urcorner \; \ulcorner e \urcorner \; \ulcorner \tau \urcorner$$

We must encode *explicitly*

– context operations (lookup, insertion, ...)

– context-related properties (weakening, exchange, ...)

- **Exploitation of the meta-language context** *LF*

$$\ulcorner\Delta\urcorner \; \mid\!-_\Sigma \; M : \texttt{ofe} \; \ulcorner e \urcorner \; \ulcorner \tau \urcorner$$

where, for each $x_i : \tau_i$ in Δ,

$$\ulcorner x_i : \tau_i \urcorner \; = \; x_i : \texttt{exp}, \quad t_i : \texttt{ofe} \; \; x_i \; \ulcorner \tau_i \urcorner$$

– context operations reduce to meta-level primitives

– meta-theoretic properties are inherited from the meta-language

Aspects of a meta-representation

Meta-representation

=

program

+

adequacy theorems

Meta-representation in *LF*: signature

Example:

$e ::= x \mid \ldots \mid \mathbf{lam}\ x.\ e \mid e_1\ e_2 \mid \ldots$

$\tau ::= \ldots \mid \tau_1 \rightarrow \tau_2 \mid \ldots$

```
exp : type.

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

$$\frac{\Delta, x{:}\tau_1 \vdash e : \tau_2}{\Delta \vdash \mathbf{lam}\ x.\ e : \tau_1 \rightarrow \tau_2} \qquad \frac{\Delta \vdash e_1 : \tau_2 \rightarrow \tau_1 \quad \Delta \vdash e_2 : \tau_2}{\Delta \vdash e_1\ e_2 : \tau_1}$$

```
ofe : exp -> tp -> type.

of_lam : ofe (lam E) (T1 => T2)
          <- ({x:exp} ofe x T1
                   -> ofe (E x) T2).
of_app : ofe (app E1 E2) T1
          <- ofe E1 (T2 => T1)
          <- ofe E2 T2.
```

Meta-representation in *LF*: adequacy

Adequacy theorem (*typing of expressions*)

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

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

and canonical *LLF* objects M such that

$$\ulcorner\Delta\urcorner \ \vdash_\Sigma M : \mathtt{ofe}\ \ulcorner e\urcorner\ \ulcorner\tau\urcorner$$

is derivable, where

$$\ulcorner\Delta\urcorner = \begin{bmatrix} x_1 : \mathtt{exp},\ t_1 : \mathtt{ofe}\ x_1\ \ulcorner\tau_1\urcorner \\ \ldots \\ x_1 : \mathtt{exp},\ t_n : \mathtt{ofe}\ x_n\ \ulcorner\tau_n\urcorner \end{bmatrix}$$

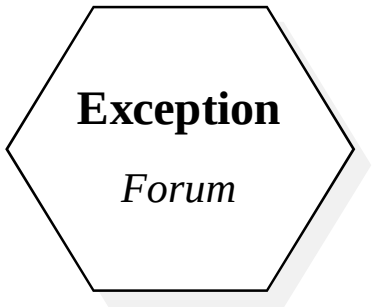
Limitations

The context-based representation methodology does not handle satisfactorilly:

- 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



The problem

$$\ulcorner \quad \mathcal{E} \quad \urcorner = M$$
$$S \mid\!- K; e \Rightarrow a$$

Store!!!
$c_i = v_i, \ldots$

- Term-based representation

$$\cdot \mid\!-_{\Sigma} M : \texttt{eval}\ \ulcorner S \urcorner\ \ulcorner K \urcorner\ \ulcorner e \urcorner\ \ulcorner a \urcorner$$

... as before

- Context-based representation

$$\ulcorner S \urcorner \mid\!-_{\Sigma} M : \texttt{eval}\ \ulcorner K \urcorner\ \ulcorner e \urcorner\ \ulcorner a \urcorner$$

This does not work!

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

Goal

Design a logical framework that

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

– is conservative over *LF*

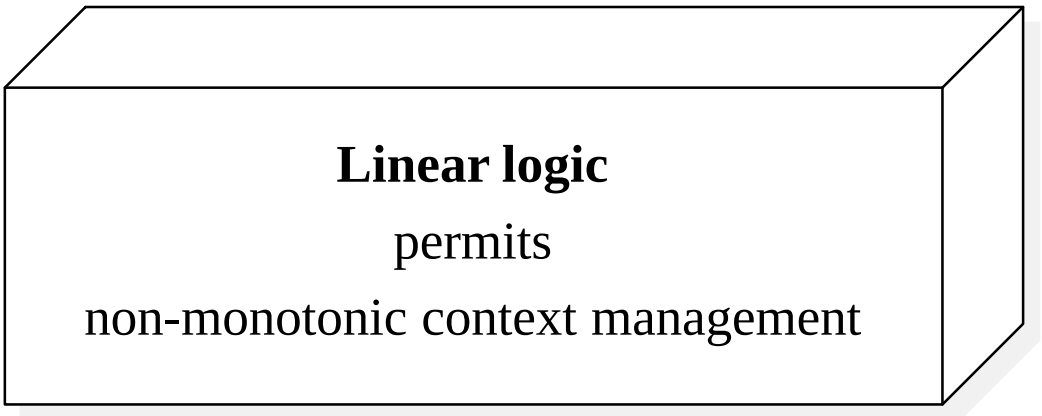
- language (λ^{Π})
- meta-representation methodology
- examples

– 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



Choice of the operators

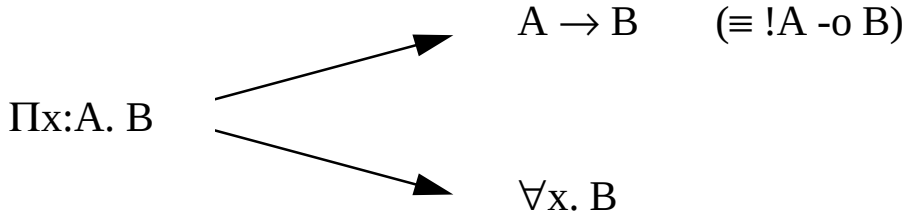
Desiderata

- model arbitrary non-monotonic context operations
- conservative extension of the operators of λ^{Π}
- existence of unique canonical forms
- completeness of uniform proof search

$$\Pi \quad \text{-o} \quad \& \quad \top$$

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

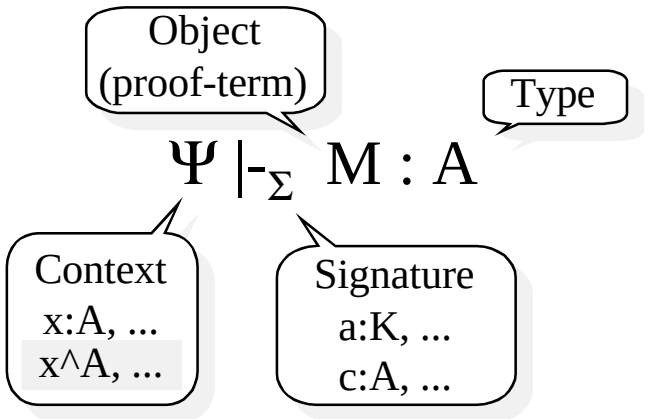
This is the type-theoretic version of the language of *linear higher-order hereditary Harrop formulas*, where



We are within *intuitionistic* linear logic

The linear type theory
$\lambda^{\Pi\text{ -o \& T}}$

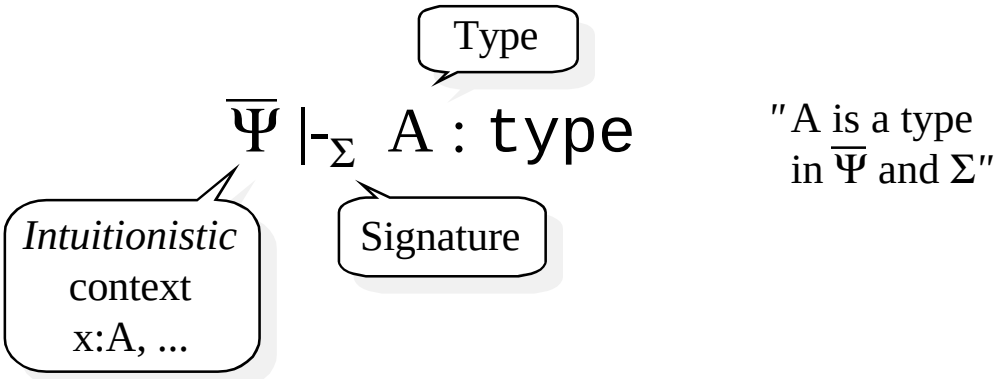
Kinds	K ::= `type` | $\Pi x{:}A.\ K$
Type families	P ::= a | P M
Types	A ::= P | $\Pi x{:}A_1.\ A_2$ | A_1 -o A_2 | A_1 & A_2 | T
Objects	M ::= x | c | $\lambda x{:}A.\ M$ | $M_1\ M_2$ | $\lambda x\hat{}A.\ M$ | $M_1\ \hat{}\ M_2$ | $\langle M_1, M_2\rangle$ | fst M | snd M | $\langle\rangle$



"M has type A in Ψ and Σ"

$\lambda^{\Pi\text{ -o \& T}}$ is the largest propositional linear extension of λ^{Π} admitting unique canonical forms

More λ^{Π} -o & T



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

Types and kinds are *linearly closed*: no linear dependencies

Properties of $\lambda^{\Pi\text{-}o\ \&\ T}$

We restricted the semantics of $\lambda^{\Pi\text{-}o\ \&\ T}$ to terms that are in η-*long form*:

$$\Psi \mid\!-_{\Sigma}\ U \Uparrow V$$

- 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 $\Psi \mid\!-_{\Sigma}\ U \Uparrow V$ 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 $\Psi \mid\!-_{\Sigma}\ U \Uparrow V$

- Conservativity over *LF*

  If Ψ, Σ, U and V do not mention linear constructs, then $\Psi \mid\!-_{\Sigma}\ U \Uparrow V$ is derivable in *LLF* iff $\Psi \mid\!-_{\Sigma}^{LF}\ U \Uparrow V$ is derivable in *LF*

Logic programming in $\lambda^{\Pi\text{-o \& T}}$

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

LLF is adequate for an implementation as a logic programming language

- it is complete for uniform proof search

> $\lambda^{\Pi\text{-o \& T}}$ is the largest propositional linear extension of λ^{Π} 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 *monotonically*

– object-level parameters

Linear *LLF* assumptions

– part of the object-level context managed *linearly*

The operators of $\lambda^{\Pi\text{-o}\,\&\,\top}$ 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: $\tau ::= \ldots \mid \tau_1 \rightarrow \tau_2 \mid \tau\ \mathbf{ref} \mid \mathbf{1}$

Expressions: $e ::= x \mid \ldots \mid \mathbf{lam}\ x.\ e \mid e_1\ e_2 \mid \ldots \mid c \mid <> \mid \mathbf{ref}\ e \mid \mathbf{!}e \mid e_1 := e_2 \mid e_1; e_2$

Store: $S ::= \cdot \mid S, c=v$

Expressions

```
exp  : type.
cell : type.

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

ML^{ref}: typing

Expression: e — Type: τ

$$\Delta \vdash e : \tau$$

Context: $x_i:\tau_i, \ldots, c_j:\sigma_j, \ldots$

"e has type τ in Δ"

Representation: $\ulcorner\Delta\urcorner \vdash_\Sigma \ulcorner\mathcal{T}\urcorner : \mathtt{ofe}\ \ulcorner e\urcorner\ \ulcorner\tau\urcorner$

$x_i : \mathtt{exp}, \quad t_i : \mathtt{ofe}\ x_i\ \ulcorner\tau_i\urcorner, \ldots$

$c_j : \mathtt{cell}, \quad l_j : \mathtt{ofc}\ c_j\ \ulcorner\sigma_j\urcorner, \ldots$

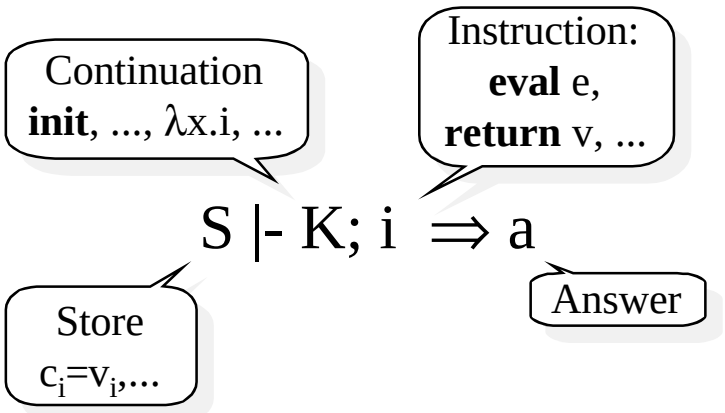
$$\frac{\Delta \vdash e : \tau\ \mathbf{ref}}{\Delta \vdash\ !\, e : \tau}$$

$$\frac{\Delta \vdash e_1 : \tau\ \mathbf{ref} \quad \Delta \vdash e_2 : \tau}{\Delta \vdash e_1 := e_2 : \mathbf{1}}$$

```
ofe_deref :
  ofe (deref E) T
   <- ofe E (rf T).


ofe_assign :
  ofe (assign E1 E2) 1
   <- ofe E1 (rf T)
   <- ofe E2 T.
```

ML^{ref}: evaluation



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

Representation:

$$\ulcorner S \urcorner \;\; \text{|-} \;\; \ulcorner \mathcal{E} \urcorner \Uparrow \texttt{eval} \;\; \ulcorner K \urcorner \;\; \ulcorner i \urcorner \;\; \ulcorner a \urcorner$$

$c_i : \texttt{cell}, \;\; h_i \wedge \texttt{contains} \; c_i \; \ulcorner v_i \urcorner, \ldots$

ML^{ref}: imperative rules

$$\frac{S',c=v,S'' \mid\!- K; \textbf{return } v \Rightarrow a}{S',c=v,S'' \mid\!- K; \textbf{!}\, c \Rightarrow a}$$

```
ev_deref1 : eval K (ref1 (loc C)) A
               o- read C V
                & eval K (return V) A.


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

$$\frac{S',c=v,S'' \mid\!- K; \textbf{return } <> \Rightarrow a}{S',c=v',S'' \mid\!- K; c := v \Rightarrow a}$$

```
ev_assign2 : eval K (assign2 (loc C) V) A
               o- contains C V'
               o- (contains C V
                   -o eval K (return unit) A.
```

ML^{ref}: 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 $\mathcal{X}$ of

$$S \mid\!- K; i \Rightarrow a$$

and canonical *LLF* objects M such that

$$\ulcorner S \urcorner \mid\!- M \Uparrow \texttt{eval}\ \ulcorner K \urcorner\ \ulcorner i \urcorner\ \ulcorner a \urcorner$$

is derivable, where

$$\ulcorner S \urcorner = \begin{bmatrix} c_1 : \texttt{cell},\ h_1 \wedge \texttt{contains}\ c_1\ \ulcorner v_1 \urcorner \\ \ldots \\ c_1 : \texttt{cell},\ h_n \wedge \texttt{contains}\ c_n\ \ulcorner v_n \urcorner \end{bmatrix}$$

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:

```
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^{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]