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

$$\boxed{\text{Logical framework} \;=\; \text{meta-language} \;+\; \text{representation methodology}}$$

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
  - λ-lifting [Leone]
  - lazy evaluation [Okasaki]
  - monads [Gehrke'95]

Meta-Language

- Logics
  - Horn clauses (*Prolog*)
  - Higher-order hereditary Harrop formulas (λ*Prolog* [Miller,Nadathur'88], *Isabelle* [Paulson'93])
  - Classical linear logic (*Forum* [Miller'94])
- Type theories
  - λ^{Π} (*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])
  - $\lambda^{\Pi\multimap\&\top}$ (*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

$x_i : \tau_i, \ldots$

$$\ulcorner \begin{array}{c} \mathcal{T} \\ \Omega \vdash e : \tau \end{array} \urcorner \quad = \quad M$$

- **Term-based representation**

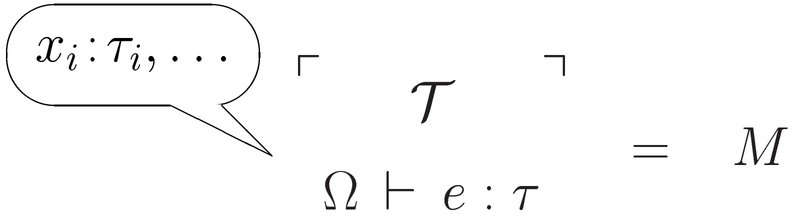
$$\cdot \vdash_\Sigma M : \texttt{has\_type}\, \ulcorner\Omega\urcorner \, \ulcorner e\urcorner \, \ulcorner\tau\urcorner$$

We must encode *explicitly*

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

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

Representation of the Context (Cont'd)



$x_i : \tau_i, \ldots$

$$\ulcorner \begin{array}{c} \mathcal{T} \\ \Omega \vdash e : \tau \end{array} \urcorner \;=\; M$$

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

$$\ulcorner\Omega\urcorner \vdash_{\Sigma} M : \texttt{has\_type}\, \ulcorner e \urcorner\, \ulcorner \tau \urcorner$$

where for each $x_i : \tau_i$ in Ω,

$$\ulcorner x_i : \tau_i \urcorner \;=\; x_i : \texttt{exp},\; t_i : \texttt{has\_type}\, x_i\, \ulcorner \tau_i \urcorner$$

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

λ^{Π}, the Meta-Language of *LF*

- **Syntax**

$$\begin{array}{rrcl}
\textit{Kinds} & K & ::= & \texttt{type} \mid \Pi x\!:\!A.\,K \\
\textit{Type families} & P & ::= & a \mid P\,M \\
\textit{Types} & A & ::= & P \mid \Pi x\!:\!A.\,B \\
\textit{Objects} & M & ::= & x \mid c \mid \lambda x\!:\!A.\,M \mid M\,N
\end{array}$$

- **Semantics**

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

"M has type A in Γ and Σ"

Context
$x\!:\!A, \ldots$

Signature
$a\!:\!K, \ldots, c\!:\!A, \ldots$

λ^{Π}, the Meta-Language of ***LF*** (Cont'd)

$$\frac{\Gamma, x\!:\!A \vdash_\Sigma M : B}{\Gamma \vdash_\Sigma \lambda x\!:\!A.\, M : \Pi x\!:\!A.\, B}\ \textbf{lam} \qquad \frac{\Gamma \vdash_\Sigma M : \Pi x\!:\!A.\, B \quad \Gamma \vdash_\Sigma N : A}{\Gamma \vdash_\Sigma M\, N : [N/x]B}\ \textbf{app}$$

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

The Problem

$c_i = v_i, \ldots$

$$\ulcorner \begin{array}{c} \mathcal{E} \\ S \rhd K \vdash e \hookrightarrow a \end{array} \urcorner \quad = \quad M$$

- **Term-based representation**

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

… as before

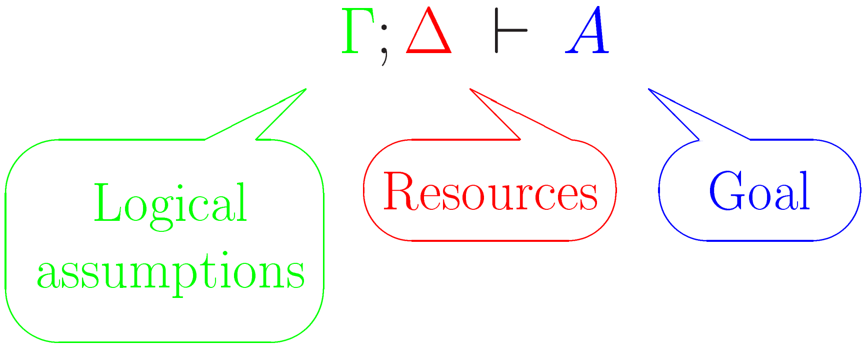
- **Context-based representation**

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

This does not work!

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

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"
- $A \multimap B$ = "B assuming A as a resource"
- $A \rightarrow B$ = "B assuming A as a logical hypothesis"

A Simple Situation

$\$$ = "I have one dollar"
C = "I buy a coke"
F = "I buy French fries"

$\$ \to C$ = "With one dollar, I can buy a coke"
$\$ \to F$ = "With one dollar, I can buy French fries"

$\$ \to C, \$ \to F, \$ \vdash C \wedge F$ "With **one** dollar, I can buy both a coke **and** French fries" !!

$$\dfrac{\dfrac{\dfrac{}{\$ \to C, \$ \to F, \$ \vdash \$ \to C} \qquad \dfrac{}{\$ \to C, \$ \to F, \$ \vdash \$}}{\$ \to C, \$ \to F, \color{red}{\$} \vdash C} \qquad \dfrac{\dfrac{}{\Gamma \vdash \$ \to F} \qquad \dfrac{}{\Gamma \vdash \$}}{\$ \to C, \$ \to F, \color{red}{\$} \vdash F}}{\underbrace{\$ \to C, \$ \to F, \$}_{\Gamma} \vdash C \wedge F}$$

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**

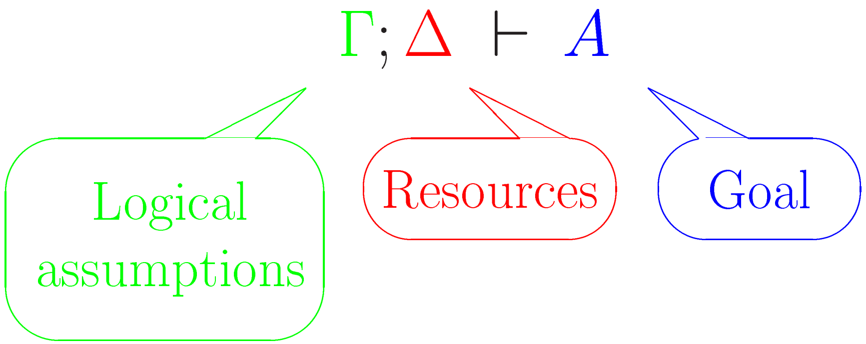
- either *available* or *consumed*
- once consumed, it cannot be used again

Note: the derivation is uncontroversial if we have only propositions

ss = "the sun shines"
sg = "I wear sunglasses"
ic = "I crave ice-cream"

$$ss \rightarrow sg, ss \rightarrow ic, ss \vdash sg \wedge ic$$

Linear Logic



$$\Gamma; \Delta \;\vdash\; A$$

Resource operators

- $\wedge \;\Longrightarrow\; \otimes$ $\qquad A \otimes B$ = "A and B simultaneously"
- $\rightarrow \;\Longrightarrow\; \multimap$ $\qquad A \multimap B$ = "B assuming A as a resource"

$$\dfrac{\dfrac{\overline{\Gamma; \cdot \vdash \$ \multimap C} \qquad \overline{\Gamma; \$ \vdash \$}}{\Gamma; \$ \vdash C} \qquad \dfrac{\overline{\Gamma; \cdot \vdash \$ \multimap F} \qquad \overline{\Gamma; \$ \vdash \$}}{\Gamma; \$ \vdash F}}{\underbrace{\$ \multimap C, \$ \multimap F}_{\Gamma}; \$, \$ \vdash C \otimes F}$$

A Step Back

$$\$ \to C, \$ \to F, \$ \vdash C \wedge F$$

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

- $\wedge \implies \&$ $\qquad A \mathbin{\&} B =$ “A and B alternatively”

$$\dfrac{\dfrac{\dfrac{}{\Gamma; \cdot \vdash \$ \multimap C} \quad \dfrac{}{\Gamma; \$ \vdash \$}}{\Gamma; \$ \vdash C} \qquad \dfrac{\dfrac{}{\Gamma; \cdot \vdash \$ \multimap F} \quad \dfrac{}{\Gamma; \$ \vdash \$}}{\Gamma; \$ \vdash F}}{\underbrace{\$ \multimap C, \$ \multimap F}_{\Gamma}; \$ \vdash C \mathbin{\&} F}$$

Linear Operators

Context splitting $\Longrightarrow$ multiplicatives
Context sharing $\Longrightarrow$ additives

$$\begin{tikzcd}
 & \otimes \\
\wedge \arrow[ur, Rightarrow] \arrow[dr, Rightarrow] & \\
 & \&
\end{tikzcd}
\qquad
\begin{tikzcd}
 & \mathbf{1} \\
\mathbf{T} \arrow[ur, Rightarrow] \arrow[dr, Rightarrow] & \\
 & \top
\end{tikzcd}
\qquad
\begin{tikzcd}
 & \multimap \\
\rightarrow \arrow[ur, Rightarrow] \arrow[dr, Rightarrow] & \\
 & \leadsto
\end{tikzcd}$$

$$\begin{tikzcd}
 & \parr \\
\vee \arrow[ur, Rightarrow] \arrow[dr, Rightarrow] & \\
 & \oplus
\end{tikzcd}
\qquad
\begin{tikzcd}
 & \bot \\
\mathbf{F} \arrow[ur, Rightarrow] \arrow[dr, Rightarrow] & \\
 & \mathbf{0}
\end{tikzcd}
\qquad
\left\{\begin{array}{ccc}
\neg & \Longrightarrow & {}^{\perp} \\
\forall & \Longrightarrow & \forall \\
\exists & \Longrightarrow & \exists
\end{array}\right.$$

Some Inference Rules

$$\frac{}{\Gamma, A; \cdot \vdash A}\,\textbf{int} \qquad \frac{}{\Gamma; A \vdash A}\,\textbf{lin}$$

$$\frac{\Gamma, A; \Delta \vdash B}{\Gamma; \Delta \vdash A \rightarrow B}\,{\rightarrow}\,\textbf{I} \qquad \frac{\Gamma; \Delta \vdash A \rightarrow B \quad \Gamma; \cdot \vdash A}{\Gamma; \Delta \vdash B}\,{\rightarrow}\,\textbf{E}$$

$$\frac{\Gamma; \Delta, A \vdash B}{\Gamma; \Delta \vdash A \multimap B}\,{\multimap}\,\textbf{I} \qquad \frac{\Gamma; \Delta_1 \vdash A \multimap B \quad \Gamma; \Delta_2 \vdash A}{\Gamma; \Delta_1, \Delta_2 \vdash B}\,{\multimap}\,\textbf{E}$$

$$\frac{\Gamma; \Delta \vdash A \quad \Gamma; \Delta \vdash B}{\Gamma; \Delta \vdash A \,\&\, B}\,\&\,\textbf{I} \qquad \frac{\Gamma; \Delta \vdash A \,\&\, B}{\Gamma; \Delta \vdash A}\,\&\,\mathbf{E_1} \qquad \frac{\Gamma; \Delta \vdash A \,\&\, B}{\Gamma; \Delta \vdash B}\,\&\,\mathbf{E_2}$$

$$\frac{}{\Gamma; \Delta \vdash \top}\,\top\,\textbf{I}$$

Exponentials

Observe that $\wedge$ corresponds to both $\otimes$ and & when the resource context is **empty**

The same holds for all connectives **except** $\rightarrow$

$$\frac{\Gamma, A; \Delta \vdash B}{\Gamma; \Delta \vdash A \rightarrow B}\rightarrow\mathbf{I} \qquad \frac{\Gamma; \Delta \vdash A \rightarrow B \quad \Gamma; \cdot \vdash A}{\Gamma; \Delta \vdash B}\rightarrow\mathbf{E}$$

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

Interprete logical assumptions as *inexhaustible resources*

$$!A = \text{“as many copies of } A \text{ as you wish”}$$

$$\begin{array}{rcl} \Gamma, A; \Delta \vdash C & \Longleftrightarrow & \Gamma; \Delta, !A \vdash C \\ A \rightarrow B & \Longleftrightarrow & (!A) \multimap B \end{array}$$

Observations

- Linear logic is a **conservative extension** of traditional logic:

  The natural translation of judgments maintains:
  - derivability
  - **derivations**

- Direct representation of resources

LLF

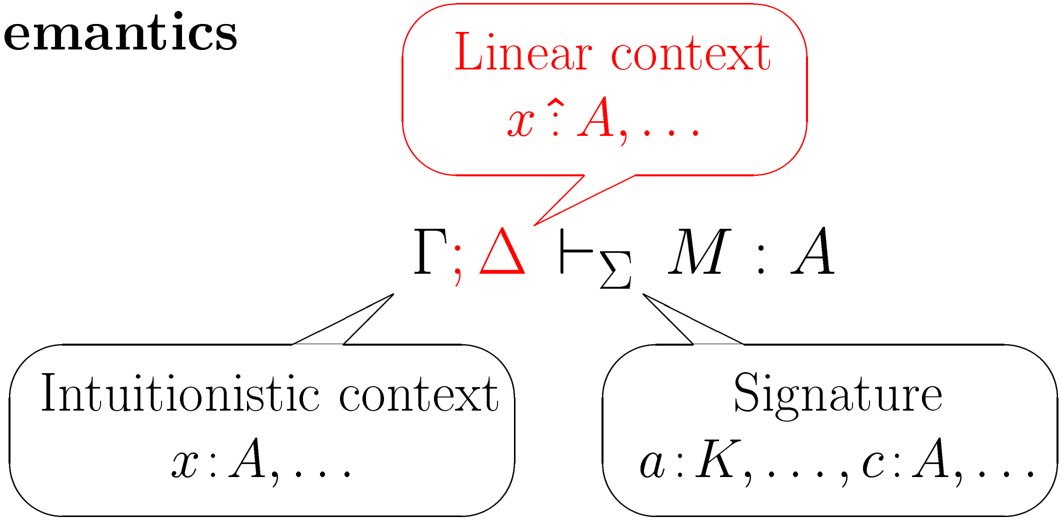
- **Meta-language**: $\lambda^{\Pi\multimap\&\top}$, a type theory based on Π, $\multimap$, $\&$ and $\top$
- **Representation methodology**: judgments-as-types, but provides direct encoding of state in the linear context
- **Range of applicability**: declarative and imperative formalisms

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

- **Syntax**

$$\begin{array}{rrcl}
\textit{Kinds} & K & ::= & \texttt{type} \mid \Pi x\,{:}\,A.\,K \\
\textit{Type families} & P & ::= & a \mid P\,M \\
\textit{Types} & A & ::= & P \mid \Pi x\,{:}\,A.\,B \\
 & & & \mid A \multimap B \mid A\,\&\,B \mid \top \\
\textit{Objects} & M & ::= & x \mid c \mid \lambda x\,{:}\,A.\,M \mid M\;N \\
 & & & \mid \hat{\lambda} x\,{:}\,A.\,M \mid M\,\hat{}\,N \mid \langle M, N\rangle \mid \textsc{fst}\;M \mid \textsc{snd}\;M \mid \langle\rangle
\end{array}$$

- **Semantics**



Linear context

$x \,\hat{:}\, A, \ldots$

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

Intuitionistic context

$x : A, \ldots$

Signature

$a : K, \ldots, c : A, \ldots$

"M has type A in Γ, Δ and Σ"

$\lambda^{\Pi\multimap\&\top}$, Some Inference Rules

$$\frac{}{\Gamma, x{:}A; \cdot \vdash_\Sigma x : A}\ \textbf{ivar} \qquad \frac{}{\Gamma; x \mathrel{\hat{:}} A \vdash_\Sigma x : A}\ \textbf{lvar}$$

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

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

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

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

$\lambda^{\Pi\multimap\&\top}$: Main Properties

Lemma (*Church-Rosser property*)

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

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 from such that $M \longrightarrow^* N$ 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*)

  $\lambda^{\Pi \multimap \& \top}$ 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 Γ, 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

$$
\begin{array}{rccl}
\textit{Types} & \tau & ::= & \ldots \mid \mathbf{1} \mid \tau_1 \rightarrow \tau_2 \mid \tau\ \mathbf{ref} \\
 & & & \\
\textit{Expressions} & e & ::= & x \\
 & & \mid & \langle\rangle \\
 & & \mid & \mathbf{lam}\ x.\, e \\
 & & \mid & e_1\ e_2 \\
 & & \mid & \ldots \\
 & & \mid & c \\
 & & \mid & \mathbf{ref}\ e \\
 & & \mid & \mathbf{!}e \\
 & & \mid & e_1 := e_2 \\
 & & & \\
\textit{Store} & S & ::= & \cdot \mid S, c = v
\end{array}
$$

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

$$\Omega \vdash e : \tau$$ "e has type τ in Ω"

Context

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

Expression

Type

Representation: $\ulcorner \Omega \urcorner \vdash_\Sigma \ulcorner \mathcal{T} \urcorner : \texttt{exp\_type}\, \ulcorner e \urcorner\, \ulcorner \tau \urcorner$

$$\begin{array}{lll} x_i : \texttt{exp}, & t_i : \texttt{exp\_type}\, x_i\, \ulcorner \tau_i \urcorner, & \ldots \\ c_j : \texttt{cell}, & l_j : \texttt{cell\_type}\, c_j\, \ulcorner \sigma_j \urcorner, & \ldots \end{array}$$

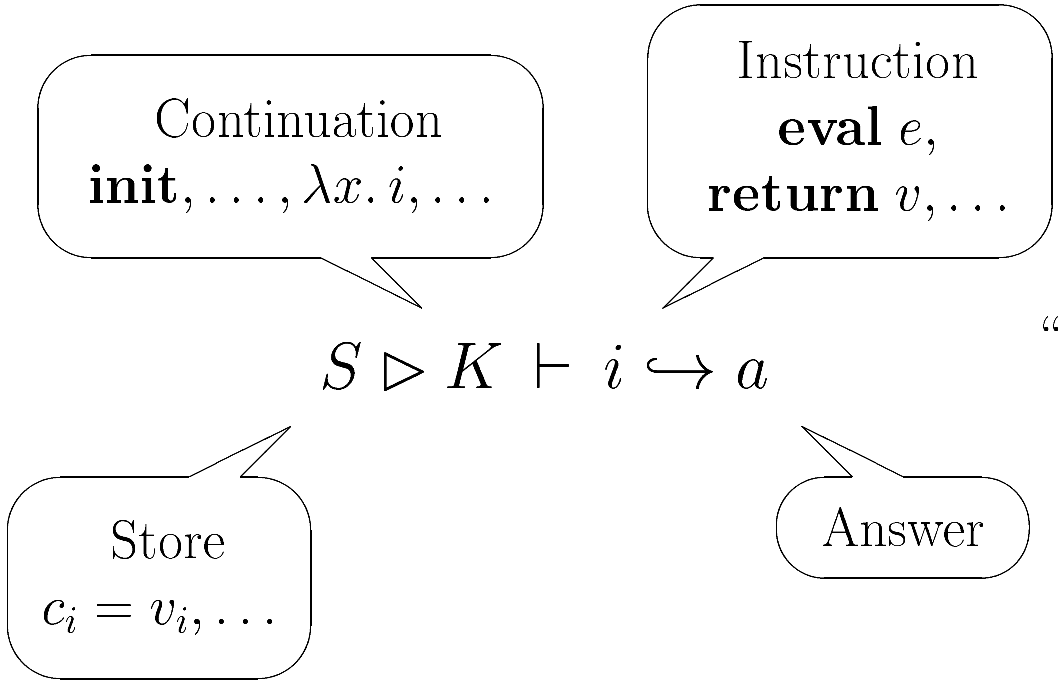
$$\frac{\Omega \vdash e_1 : \tau\ \mathbf{ref} \quad \Omega \vdash e_2 : \tau}{\Omega \vdash e_1 := e_2 : \mathbf{1}}\ \textbf{et\_assign}$$

```
et_assign :  exp_type E1 (rf T)
          -> exp_type E2 T
          -> exp_type (assign E1 E2) 1.
```

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

```
et_deref  :  exp_type E (rf T)
          -> exp_type (deref E) T.
```

MLR: Evaluation



Continuation

$\mathbf{init}, \ldots, \lambda x.\, i, \ldots$

Instruction

$\mathbf{eval}\ e,$

$\mathbf{return}\ v, \ldots$

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

Store

$c_i = v_i, \ldots$

Answer

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

Representation: $\ulcorner S \urcorner \vdash_{\Sigma} \ulcorner \mathcal{E} \urcorner : \mathtt{eval}\ \ulcorner K \urcorner\ \ulcorner i \urcorner\ \ulcorner a \urcorner$

$c_i : \mathtt{cell},\ h_i \,\hat{:}\, \mathtt{contains}\ c_i\ \ulcorner v_i \urcorner, \ \ldots$

***MLR*: Some Imperative Rules**

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

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

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

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

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

and canonical *LLF* objects M such that

$$\ulcorner S \urcorner \vdash_{\Sigma} M : \texttt{eval}\ \ulcorner i \urcorner\ \ulcorner a \urcorner$$

is derivable, where

$$\ulcorner S \urcorner = \left[\begin{array}{c} c_1 : \texttt{cell},\ \ h_1 \,\hat{:}\, \texttt{contains}\ c_1\ \ulcorner v_1 \urcorner \\ \ldots \\ c_n : \texttt{cell},\ \ h_n \,\hat{:}\, \texttt{contains}\ c_n\ \ulcorner v_n \urcorner \end{array} \right]$$

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 $S \rhd K \vdash i \hookrightarrow a$, 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 **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.
```

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 Necula)

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