A Linear Logical Framework

Iliano Cervesato

Department of Computer Science
Carnegie Mellon University
Contents

• Overview

• Linear logic

• Logical frameworks

• \textit{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

\[
\text{Logical framework} = \text{meta-language} + \text{representation methodology}
\]
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 (*λProlog* [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])
  – $\lambda^{\Pi-\omega\&\top}$ (*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

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

- Term-based representation

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

We must encode \textit{explicitly}

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

\[ \Gamma(x_i : \tau_i, \ldots) \vdash \mathcal{T} \Downarrow \]
\[ \Omega \vdash e : \tau \Downarrow M \]

- Exploitation of the meta-language context

\[ \Gamma(\Omega) \vdash \Sigma M : \text{has\_type} \Gamma(e) \Gamma(\tau) \]

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

\[ \Gamma(x_i : \tau_i) = x_i : \text{exp}, t_i : \text{has\_type} x_i \Gamma(\tau_i) \]

- 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**

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

\[ Type \ families \quad P ::= a \mid PM \]

\[ Types \quad A ::= P \mid \Pi x: A. B \]

\[ Objects \quad M ::= x \mid c \mid \lambda x: A. M \mid MN \]

**Semantics**

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

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

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

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

\[ c_i = v_i, \ldots \]\n
\[ \vdash S \triangleright K \vdash e \rightarrow a = M \]

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

  ... as before

- **Context-based representation**
  \[ \vdash S \triangleright \vdash_{\Sigma} M : \text{eval} \vdash K \triangleright e \triangleright a \]

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**

Accessing a resource consumes it

**Main resource operators**

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

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

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

"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**

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

\[
\begin{align*}
\Gamma; \cdot \vdash \$ \rightarrow C & \quad \Gamma; \$ \vdash \$ & \quad \Gamma; \cdot \vdash \$ \rightarrow F & \quad \Gamma; \$ \vdash \$
\hline
\Gamma; \$ \vdash C & \quad \Gamma; \$ \vdash F
\end{align*}
\]

\[
\frac{\$ \rightarrow C, \$ \rightarrow F; \$, \$ \vdash C \otimes F}{\Gamma}
\]
$ \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 = \text{“A and B alternatively”} \quad$

\[
\begin{align*}
\Gamma ; \cdot \vdash $ \rightarrow C & \quad \Gamma ; $ \vdash $ \quad \Gamma ; \cdot \vdash $ \rightarrow F & \quad \Gamma ; $ \vdash $ \\
\Gamma ; $ \vdash C & \quad \Gamma ; $ \vdash F \\
\hline
\Gamma ; $ \vdash C \land F \quad \Gamma
\end{align*}
\]
**Linear Operators**

Context splitting  \(\implies\) multiplicatives  
Context sharing  \(\implies\) additives

\[
\begin{array}{cccc}
\land & \otimes & \top & \neg \\
\& & & & \\
\lor & \vee & \bot & \\
\oplus & & & \\
\end{array}
\]

\[
\begin{array}{cccc}
\neg & \iff & \perp & \\
\forall & \iff & \forall & \\
\exists & \iff & \exists & \\
\end{array}
\]
Some Inference Rules

\[
\begin{align*}
\Gamma, A; \cdot & \vdash A \quad \text{int} \\
\Gamma; A & \vdash A \quad \text{lin}
\end{align*}
\]

\[
\begin{align*}
\Gamma, A; \Delta & \vdash B \\
\Gamma; \Delta & \vdash A \rightarrow B \\
\Gamma; \Delta & \vdash A \rightarrow B \quad \rightarrow_I
\end{align*}
\]

\[
\begin{align*}
\Gamma; \Delta, A & \vdash B \\
\Gamma; \Delta & \vdash A \rightarrow B \\
\Gamma; \Delta & \vdash A \rightarrow B \quad \rightarrow_I
\end{align*}
\]

\[
\begin{align*}
\Gamma; \Delta & \vdash A \\
\Gamma; \Delta & \vdash B \\
\Gamma; \Delta & \vdash A \& B \quad \&_I
\end{align*}
\]

\[
\begin{align*}
\Gamma; \Delta & \vdash A \& B \\
\Gamma; \Delta & \vdash A \\
\Gamma; \Delta & \vdash A \& B \quad \&_{E_1}
\end{align*}
\]

\[
\begin{align*}
\Gamma; \Delta & \vdash A \& B \\
\Gamma; \Delta & \vdash A \& B \\
\Gamma; \Delta & \vdash A \& B \quad \&_{E_2}
\end{align*}
\]

\[
\begin{align*}
\Gamma; \Delta & \vdash \top \\
\Gamma; \Delta & \vdash T \quad \top_I
\end{align*}
\]
Exponentials

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

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

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

Interprete logical assumptions as inexhaustible resources

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

\[
\frac{}{\Gamma, A; \Delta \vdash C} \iff \frac{}{\Gamma; \Delta, !A \vdash C} \\
\frac{A \rightarrow B}{(!A) \rightarrow B}
\]
Observations

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

- Direct representation of resources
\[ \mathcal{LLF} \]

- **Meta-language**: \( \lambda^{\Pi-\text{o} \& \top} \), a type theory based on \( \Pi \), \( \text{o} \), \( \& \) and \( \top \)

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

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

- **Syntax**

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

  \( Type \ families \quad P ::= a \mid P \ M \)

  \( Types \quad A ::= P \mid \Pi x : A. B \)

  \[ \mid A \rightarrow B \mid A \land B \mid \top \]

  \( Objects \quad M ::= x \mid c \mid \lambda x : A. M \mid M \ N \)

  \[ \mid \hat{x} : A. M \mid M ^\bowtie N \mid \langle M, N \rangle \mid \text{fst} \ M \mid \text{snd} \ M \mid \emptyset \]

- **Semantics**

  Linear context

  \[ x ? A, \ldots \]

  \[ \Gamma ; \Delta \vdash \Sigma \quad M : A \]

  “\( M \) has type \( A \)

  in \( \Gamma, \Delta \) and \( \Sigma \)”

  Intuitionistic context

  \[ x : A, \ldots \]

  Signature

  \[ a : K, \ldots, c : A, \ldots \]
$\chi_{\Pi \rightarrow (\Lambda \otimes \top)}$, Some Inference Rules

\[
\Gamma; x : A; \cdot \vdash_{\Sigma} x : A \quad \text{iVar}
\]

\[
\Gamma; \Delta \vdash_{\Sigma} M : B 
\quad \Gamma; \Delta \vdash_{\Sigma} \lambda x : A. M : \Pi x : A. B 
\]

\[
\Gamma; \Delta \vdash_{\Sigma} \hat{\lambda} x : A. M : A \rightarrow B 
\quad \text{lam}
\]

\[
\Gamma; \Delta \vdash_{\Sigma} M : A 
\quad \Gamma; \Delta \vdash_{\Sigma} N : B 
\]

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

\[
\Gamma; \Delta \vdash_{\Sigma} \langle \bullet \rangle : \top 
\quad \text{unit}
\]

\[
\Gamma; \Delta \vdash_{\Sigma} M : A \rightarrow B 
\quad \Gamma; \Delta \vdash_{\Sigma} N : A 
\]

\[
\Gamma; \Delta \vdash_{\Sigma} M \rightarrow N : B 
\quad \text{iapp}
\]

\[
\Gamma; \Delta \vdash_{\Sigma} M : A \& B 
\quad \Gamma; \Delta \vdash_{\Sigma} M : A \& B 
\]

\[
\Gamma; \Delta \vdash_{\Sigma} \text{FST} M : A 
\quad \text{fST}
\]

\[
\Gamma; \Delta \vdash_{\Sigma} \text{SND} M : B 
\quad \text{fST}
\]

Iliano Cervesato — A Linear Logical Framework
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
\[ \Gamma; \Delta \vdash \Sigma \quad M : A \]

- Supports proof search

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

\( \lambda^{\Pi_{\text{o&}\top}} \) 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 \( LLF \)

\[ \text{iff} \quad \Gamma \vdash_{\Sigma} \ M : A \] 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

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

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

\[
\text{Store} \quad S ::= \cdot \mid S, c = v
\]

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**

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

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

**Expression**
\[ \text{Expression} \]

**Type**
\[ \text{Type} \]

**Representation:**
\[ \Gamma \vdash \Sigma \quad \Gamma^e \vdash \text{exp}_{\text{type}}(\Gamma^e) \]

\[ x_i : \text{exp}, \quad t_i : \text{exp}_{\text{type}} \quad x_i \Gamma^e_{\tau_i}, \ldots \]

\[ c_j : \text{cell}, \quad l_j : \text{cell}_{\text{type}} \quad c_j \Gamma^e_{\sigma_j}, \ldots \]

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

\[ \Omega \vdash e_1 := e_2 : 1 \quad \text{et_assign} \]

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

\[ \Omega \vdash !e : \tau \]

**et_assign**
\[ \begin{align*}
\text{et_assgin} : & \quad \text{exp}_{\text{type}} \text{ E1 (rf T)} \\
& \rightarrow \text{exp}_{\text{type}} \text{ E2 T} \\
& \rightarrow \text{exp}_{\text{type}} \text{ (assign E1 E2) 1.}
\end{align*} \]

**et_deref**
\[ \begin{align*}
\text{et_deref} : & \quad \text{exp}_{\text{type}} \text{ E (rf T)} \\
& \rightarrow \text{exp}_{\text{type}} \text{ (deref E) T.}
\end{align*} \]
**MLR: Evaluation**

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

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

**Continuation**

\textbf{init}, \ldots, \lambda x. i, \ldots

**Instruction**

\textbf{eval} e, \\
\textbf{return} v, \ldots

**Store**

\( c_i = v_i, \ldots \)

**Answer**

**Representation:**

\[ \Gamma S \vdash_{\Sigma} \Gamma \mathcal{E} \vdash_{\Sigma} \text{eval} \Gamma K \vdash_{\Gamma} i \vdash_{\Gamma} a \]

\( c_i : \text{cell}, \ h_i : \text{contains} \ c_i \ \Gamma v_i, \ldots \)
**MLR:** Some Imperative Rules

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

**ev_assign**:
\[\text{(contains } C \text{ } V\quad \neg o \quad \text{eval } K \text{ (return unit) } A)\]
\[\neg o \text{ (contains } C \text{ } V'\quad \neg o \quad \text{eval } K \text{ (assign2 (loc } C) \text{ } V) \text{ } A).\]

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

**ev_deref**:
\[\text{read } C \text{ } V\]
\[\& \text{ eval } K \text{ (return } V) \text{ } A\]
\[\neg o \text{ eval } K \text{ (ref1 (loc } C)) \text{ } A.\]

**rd**:
\[\text{contains } C \text{ } V\]
\[\neg o \text{ } <T>\]
\[\neg o \text{ read } C \text{ } V.\]
**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

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

is derivable, where

$$\Gamma S = \left[ \begin{array}{c}
\text{c_1 : cell, } h_1 \triangleright \text{contains } c_1 \Gamma v_1 \\
\ldots \\
\text{c_n : cell, } h_n \triangleright \text{contains } c_n \Gamma v_n
\end{array} \right]$$
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 \( S ⊢ K ⊢ i \mapsto a \), with \( Ω ⊢ i : τ \), \( Ω ⊢ K : τ \Rightarrow σ \) and \( Ω ⊢ S : Ω \), then \( Ω ⊢ a : σ \)

**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 Necha)
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