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, \textit{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 \textit{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**
  - λ-lifting [Leone]
  - lazy evaluation [Okasaki]
  - monads [Gehrke’95]
Meta-Language

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

- Type theories
  - ∊∞ (∊F [Harper,Honsell,Plotkin’93])
  - Calculus of Constructions (*Coq* [Dowek&al’93], *Lego* [Pollack’94])
  - Martin-Löf’s type theories (∊LF [Nordström’93], *NuPrl* [Constable&al’86])
  - ∊∞→⊂ T (∊LF [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**

\[ x_i : \tau_i, \ldots, \tau \]

\[ \Omega \vdash e : \tau = M \]

- **Term-based representation**

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

We must encode *explicitly*

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

\[ \left( x_i : \tau_i, \ldots \right) \vdash \tau \
\] \[ \Omega \vdash e : \tau = M \]

- Exploitation of the meta-language context

\[ \left\lceil \Omega \right\rceil \vdash \Sigma M : \text{has}_\text{type} \left\lceil e \right\rceil \left\lceil \tau \right\rceil \]

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

\[ \left\lceil x_i : \tau_i \right\rceil = x_i : \text{exp}, \ t_i : \text{has}_\text{type} \ x_i \left\lceil \tau_i \right\rceil \]

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

- **Syntax**

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

- **Semantics**

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

  [Diagram showing the context and signature with variables and types]
\( \lambda^{\Pi}, \) 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

\[ c_i = v_i, \ldots \Rightarrow \begin{array}{l}
\mathcal{E} \\
S \triangleright K \vdash e \leftrightarrow a = M
\end{array} \]

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

  ... as before

- **Context-based representation**

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

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**

**Main resource operators**

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

Accessing a resource \textit{consumes} it
A Simple Situation

$ = “I have one dollar”

$ \rightarrow C = “With one dollar, I can buy a coke”

$ \rightarrow F = “With one dollar, I can buy French fries”

$ \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

$\rightarrow$ is a resource

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

Note: the derivation is uncontroversial if we have only propositions

\[
\begin{align*}
ss & = \text{“the sun shines”} \\
sg & = \text{“I wear sunglasses”} \\
ic & = \text{“I crave ice-cream”}
\end{align*}
\]

\[
ss \rightarrow sg, ss \rightarrow ic, ss \vdash sg \land ic
\]
Linear Logic

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

Logical assumptions

Resources

Goal

Resource operators

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

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

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

\[
\begin{array}{c}
\$ \rightarrow C, \$ \rightarrow F; \$, \$ \vdash C \otimes F \\
\hline
\end{array}
\]
$ \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**

- \( \land \implies \& \) \; \( A \land B = “A \and B \text{ alternatively}” \)

\[
\begin{array}{c}
\Gamma; \cdot \vdash \lnot C \\
\overrightarrow{\vdash} \Gamma; \cdot \vdash C \\
\end{array} \\
\begin{array}{c}
\Gamma; \vdash \$\
\end{array} \\
\begin{array}{c}
\Gamma; \vdash F \\
\end{array} \\
\begin{array}{c}
\Gamma; \vdash \$\
\end{array} \\
\begin{array}{c}
\Gamma; \vdash \$\
\end{array}
\]

\[
\begin{array}{c}
\Gamma; \vdash \lnot F \\
\end{array} \\
\begin{array}{c}
\Gamma; \vdash F \\
\end{array} \\
\begin{array}{c}
\Gamma; \vdash \$\
\end{array} \\
\begin{array}{c}
\Gamma; \vdash \$\
\end{array}
\]

\[
\begin{array}{c}
\Gamma; \vdash \lnot C, \lnot F \\
\end{array} \\
\begin{array}{c}
\Gamma; \vdash C \land F \\
\end{array}
\]
Linear Operators

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

\[ \Gamma, A; \Delta \vdash B \] \quad \rightarrow^I

\[ \Gamma; \Delta \vdash A \rightarrow B \] \quad \rightarrow^E

\[ \Gamma; \Delta \vdash A \] \quad \rightarrow^I

\[ \Gamma; \Delta \vdash \neg B \] \quad \rightarrow^E

\[ \Gamma; \Delta \vdash A \& B \] \quad \&^I

\[ \Gamma; \Delta \vdash A \& B \& E_1 \]

\[ \Gamma; \Delta \vdash A \& B \& E_2 \]

\[ \Gamma; \Delta \vdash \top \] \quad \top^I

Iliano Cervesato — *A Linear Logical Framework* 21
Exponentials

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

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

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

Interprete logical assumptions as inexhaustible resources

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

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

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

- Direct representation of resources
- **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 *imperative* formalisms
\( \lambda^{\Pi \rightarrow \& \top} \), the Meta-Language of \textit{LLF}.

- **Syntax**

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

  \textit{Type families} 
  \[ P := a \mid PM \]

  \textit{Types} 
  \[ A := P \mid \Pi x : A. B \]
  \[ \mid A \rightarrow B \mid A \& B \mid \top \]

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

- **Semantics**

  \begin{center}
  \begin{tikzpicture}
  \node (linear) [circle, draw] {Linear context};
  \node (intuitionistic) [below left=of linear] {Intuitionistic context};
  \node (signature) [below right=of linear] {Signature};
  \node (context) [above=of linear] {\( \Gamma; \Delta \vdash \Sigma M : A \)};
  \node (statement) [right=of context] {“\( M \) has type \( A \) in \( \Gamma, \Delta \) and \( \Sigma \)”;};
  \end{tikzpicture}
  \end{center}
\( \lambda \Pi \to \& \top \), Some Inference Rules

\[
\begin{align*}
&\frac{\Gamma, x : A; \cdot \vdash x : A}{\Gamma; \cdot \vdash x : A} \text{ ivar} \\
&\frac{\Gamma; x \in A \vdash x : A}{\Gamma; \cdot \vdash x : A} \text{ ivar}
\end{align*}
\]

\[
\begin{align*}
&\frac{\Gamma, x : A; \Delta \vdash M : B}{\Gamma; \Delta \vdash \lambda x : A. M : \Pi x : A. B} \text{ lam} \\
&\frac{\Gamma; \Delta \vdash M : \Pi x : A. B}{\Gamma; \cdot \vdash N : A} \text{ iapp}
\end{align*}
\]

\[
\begin{align*}
&\frac{\Gamma; \Delta, x : A \vdash M : B}{\Gamma; \Delta \vdash \lambda x : A. M : A \to B} \text{ lam} \\
&\frac{\Gamma; \Delta \vdash M : A \to B}{\Gamma; \Delta_1, \Delta_2 \vdash M \cdot N : B} \text{ iapp}
\end{align*}
\]

\[
\begin{align*}
&\frac{\Gamma; \Delta \vdash M : A \quad \Gamma; \Delta \vdash N : B}{\Gamma; \Delta \vdash \langle M, N \rangle : A \& B} \text{ pair} \\
&\frac{\Gamma; \Delta \vdash M : A \& B}{\Gamma; \Delta \vdash \text{FST} \ M : A} \text{ fst} \\
&\frac{\Gamma; \Delta \vdash M : A \& B}{\Gamma; \Delta \vdash \text{SND} \ M : B} \text{ fst}
\end{align*}
\]

\[
\begin{align*}
&\frac{\Gamma; \Delta \vdash \langle \rangle : \top}{\Gamma; \Delta \vdash \langle \rangle : \top} \text{ unit}
\end{align*}
\]
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 form 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 \( \lambda \)-abstraction
- derivations as proof-terms
- state manipulation via linear constructs
Computational Properties of \textit{LLF}

- Allows automatic proof verification

\textbf{Theorem} (\textit{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

\textbf{Theorem} (\textit{abstract logic programming language})

\[ \lambda^{\Pi_{\neg o \& \top}} \] is an \textit{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

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

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

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

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**

**Type**

**Representation:**

\[ \Gamma \vdash \Sigma \quad \Gamma \vdash \tau : \text{exp_type} \quad \Gamma \vdash e : \tau \]

\[ x_i : \text{exp} \quad t_i : \text{exp_type} \quad x_i \vdash \tau_i \quad \ldots \]

\[ c_j : \text{cell} \quad l_j : \text{cell_type} \quad c_j \vdash \sigma_j \quad \ldots \]

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

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

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

- **et_assign**
  - \( \text{et_assign} : \text{exp_type} \ E1 \ (\text{rf} \ T) \)
  - \( \rightarrow \text{exp_type} \ E2 \ T \)
  - \( \rightarrow \text{exp_type} \ (\text{assign} \ E1 \ E2) \ 1. \)

- **et_deref**
  - \( \text{et_deref} : \text{exp_type} \ E \ (\text{rf} \ T) \)
  - \( \rightarrow \text{exp_type} \ (\text{deref} \ E) \ T. \)
**MLR: Evaluation**

- **Continuation**
  - \textbf{init}, \ldots, \lambda x. i, \ldots

- **Instruction**
  - \textbf{eval} e,
  - 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:**
\[
\Gamma S \vdash \Sigma \quad \mathcal{E} \vdash \text{eval} K \vdash i \vdash a
\]

- \( c_i: \text{cell}, \quad h_i \uparrow \text{contains} c_i \downarrow v_i, \ldots \)
**MLR: Some Imperative Rules**

\[
S', c = v, S'' \triangleright K \vdash \text{return } \langle \rangle \rightarrow a \\
S', c = v', S'' \triangleright K \vdash c := v \rightarrow a \quad \text{ev_assign}
\]

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

\[
S', c = v, S'' \triangleright K \vdash \text{return } v \rightarrow a \quad \text{ev_deref}
\]

\[
S', c = v, S'' \triangleright K \vdash !c \rightarrow a \quad \text{ev_deref}
\]

\[
\text{ev_deref} : \quad \begin{align*}
&\text{read } C \ V \\
&\quad &\&\text{eval } K \ (\text{return } V) \ A \\
&\quad -o \ \text{eval } K \ (\text{ref1 } (\text{loc } C)) \ A.
\end{align*}
\]

\[
\text{rd} : \quad \text{contains } C \ V \\
&\quad -o \ <T> \\
&\quad -o \ \text{read } C \ V.
\]

Ilario Cervesato — *A Linear Logical Framework* 34
**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

\[
\mathcal{E} \vdash \Sigma \quad M : \text{eval } \overline{i} \overline{a}
\]

is derivable, where

\[
\overline{S} = \begin{bmatrix}
c_1 : \text{cell}, & h_1 \hat{\text{contains}} \ c_1 \overline{v_1}
\end{bmatrix}
\]

\[
\ldots
\]

\[
\begin{bmatrix}
c_n : \text{cell}, & h_n \hat{\text{contains}} \ c_n \overline{v_n}
\end{bmatrix}
\]
**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 \( S \triangleright K \vdash i \leftrightarrow a \), 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**

\[
\text{tpev : eval K I A \rightarrow cont_type K T S \rightarrow instr_type I T \rightarrow ans_type A S \rightarrow type.}
\]
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 Nechaev)
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 \cite{PfenningRohwedder96}, meta-logical frameworks \cite{Schuermann95})

• Educational software for logic and the theory of programming languages