On Order and Linearity in Logical Frameworks: The Lambek Calculus Revisited

Frank Pfenning

Workshop on Logic and Computational Linguistics

Ottawa, Canada, June 2003

Warning: Work in progress!

Joint work with Jeff Polakow and Kevin Watkins
Outline

- Logical Frameworks
- Ordered Logical Framework (OLF)
- Linear Destination-Passing
- Example: Sequential Evaluation
- Ordered Concurrent Logical Framework (OCLF)
- Examples: Parallel Evaluation, [Futures]
- A Unifying Framework for Syntax and Semantics?
Logical Frameworks

- Meta-languages for deductive systems
  - Specification (abstract syntax and rules)
  - Implementation (reasoning within)
  - Meta-theory (reasoning about)

- Applications
  - Logics
  - Programming languages
Design Criteria

• Support common concepts in logics and programming languages
  • Concise and direct specification
  • High-level implementation of algorithms
  • Effective meta-theoretic reasoning

• Example concepts
  • variable binding
  • capture-avoiding substitution
  • hypothetical judgments
  • parametric judgments
The Story So Far

- Discuss only LF-family:
  - LF [Harper, Honsell & Plotkin’93]
  - Linear LF [Cervesato & Pf’96]
  - Ordered LF [Polakow & Pf’93]
  - Concurrent LF [Watkins, Cervesato, Pf, Walker’02]
  - This talk: Ordered Concurrent LF

- All of the above are intuitionistic

- Emphasize specification in this talk

- Future work: search algorithms, meta-reasoning
Representation Principles

- Logical framework characteristics
  - Type-theoretic language ("syntax")
  - Typing and equality ("semantics")
  - Representation principles ("pragmatics")
- Examine for each framework
- Other criteria for search and meta-reasoning
The Logical Framework LF

• LF [Harper, Honsell & Plotkin’93]

• Language

  Atomic Types  \[ P ::= a \mid P \, M \]
  Types  \[ A ::= P \mid A_1 \rightarrow A_2 \mid \Pi u : A_1. \, A_2 \]
  Objects  \[ M ::= c \mid u \mid \lambda u. \, M \mid M_1 \, M_2 \]
  Contexts  \[ \Gamma ::= \cdot \mid \Gamma, u : A \]

• Main judgments

  \[ \Gamma \in \Sigma \; M : A \quad M \text{ has type } A \]
  \[ \Gamma \in \Sigma \; M = N : A \quad M \text{ is equal to } N \text{ at type } A \]
• Dependent functions

\[ \Gamma, u : A \vdash M : B(u) \]
\[ \Gamma \vdash \lambda u. M : \Pi u : A. B(u) \]

\[ \Gamma \vdash M : \Pi u : A. B(u) \quad \Gamma \vdash N : A \]
\[ \Gamma \vdash M \ N : B(N) \]

• Equality is $\beta\eta$-conversion

• Equality and type-checking are decidable

• Canonical ($\beta$-normal, $\eta$-long) forms exist
LF Representation Principles

- Higher-order abstract syntax
  - Object logic variables as framework variables
- **Judgments as types**
- **Deductions as objects**
  - Proof checking reduces to framework type checking
- Hypothetical judgments as function types
  - Object-logic assumptions as framework assumptions
  - Object-logic parameters as framework parameters
Brief Example: Natural Semantics

- Object language: call-by-value \( \lambda \)-calculus

  Expressions\[
  e ::= \text{fn } x.e \mid e_1 e_2
  \]

  Values\[
  v ::= \text{fn } x.e
  \]

- Higher-order abstract syntax representation

  \[
  \begin{align*}
  \text{exp} & : \text{type.} \\
  \text{fun} & : (\text{exp} \rightarrow \text{exp}) \rightarrow \text{exp.} \\
  \text{app} & : \text{exp} \rightarrow (\text{exp} \rightarrow \text{exp}).
  \end{align*}
  \]

- Example: \( \llbracket \text{fn } f. \text{fn } x.f \ x \rrbracket = \text{fun } (\lambda f. \text{fun } (\lambda x. \text{app } f \ x)) \)
Evaluation Semantics

- Evaluation judgment \( e \rightarrow v \)

\[
\begin{align*}
\text{fn } x.e & \rightarrow \text{fn } x.e \\
\hline
\quad e_1 \rightarrow \text{fn } x.e' & \quad e_2 \rightarrow v \quad [v_2/x]e'_1 \rightarrow v \\
& \quad e_1 \ e_2 \rightarrow v
\end{align*}
\]

- Judgments as types representation (omitting quant’s)

\[
\begin{align*}
\text{eval} & : \text{exp } \rightarrow \text{exp } \rightarrow \text{type}. \\
\text{evfun} & : \text{eval } (\text{fun } (\lambda x. \text{E } x)) (\text{fun } (\lambda x. \text{E } x)). \\
\text{evapp} & : \text{eval } E_1 (\text{fun } (\lambda x. \text{E}'_1 x)) \\
& \quad \rightarrow \text{eval } E_2 \ V_2 \\
& \quad \rightarrow \text{eval } (\text{E}'_1 V_2) \ V \\
& \quad \rightarrow \text{eval } (\text{app } E_1 E_2) \ V.
\end{align*}
\]
Adequacy of Representations

- There is a *compositional bijection* between expressions and *canonical* objects $M : \text{exp}$
- There is a *compositional bijection* between deductions of $e \leftrightarrow v$ and *canonical* objects $D : \text{eval} \quad \overline{e} \quad \overline{v}$
- Canonical objects are $\beta$-normal, $\eta$-long forms
- Critical role of definitional equality
- Deemphasize in this talk
Scope of LF

● Many examples
  ● Logic: natural deduction and sequent calculi for classical, intuitionistic, modal, temporal logics; normalization and cut-elimination procedures; translations between them; program extraction and optimization
  ● Programming languages: functional and logic programming languages with a variety of features, type soundness and progress theorems, Church-Rosser theorem

● Some limitations for state and concurrency
The Linear Logical Framework

- LLF [Cervesato & Pf’96]

- Language

  Types \[ A ::= P \mid A_1 \rightarrow A_2 \mid \Pi u : A_1. A_2 \]
  \[ \mid A_1 \otimes A_2 \mid A_1 \& A_2 \mid \top \]

  Objects \[ M ::= \ldots \]

- Main judgment

  \[ \Gamma; \Delta \vdash \Sigma M : A \quad M \text{ has type } A \]

- \( \Gamma \) are \textit{unrestricted} assumptions (as before)
- \( \Delta \) are \textit{linear} assumptions (but order irrelevant)
Linear LF — Some Critical Rules

• Linear functions

\[
\frac{\Gamma; \Delta, x : A \vdash M : B}{\Gamma; \Delta \vdash \lambda x. M : A \rightarrow B}
\]

\[
\frac{\Gamma; \Delta_1 \vdash M : A \rightarrow B \quad \Gamma; \Delta_2 \vdash N : A}{\Gamma; (\Delta_1, \Delta_2) \vdash M^N : B}
\]

• Unrestricted application

\[
\frac{\Gamma; \Delta \vdash M : \Pi u : A. B(u) \quad \Gamma; \cdot \vdash N : A}{\Gamma; \Delta \vdash MN : B(N)}
\]
Linear LF Representation Principles

- Canonical forms and decidability extend
- Linear LF conservatively extends LF
- All LF representation principles still apply
- *State as linear hypotheses*
- *Imperative computations as linear objects*
New examples

- Logic: classical and intuitionistic linear logic, cut-elimination, translation
- Programming languages: imperative languages, functional languages with imperative features, lower-level languages

Some limitations for concurrency and order
Concurrency and Order

- Divergent subsequent developments:
  - Ordered Logical Framework [Polakow’01]
  - Concurrent Logical Framework [Watkins, Cervesato, Pf, Walker’02]
- This talk: a speculative synthesis of the ideas
The Ordered Logical Framework

- Ordered LF (OLF) [Polakow’01]
- Language
  
  Types $A ::= P \mid A_1 \to A_2 \mid \Pi u : A_1. A_2$
  
  $\mid A_1 \multimap A_2 \mid A_1 \& A_2 \mid \top$
  
  $\mid A_1 \setminus A_2 \mid A_2 / A_1$

- Main judgment

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

- $\Gamma$ is unrestricted, $\Delta$ is linear
- $\Omega$ is ordered (as in the Lambek calculus)
Ordered LF — Some Critical Rules

• Left implication

\[ \Gamma; \Delta; (w : A, \Omega) \vdash M : B \]
\[ \Gamma; \Delta; \Omega \vdash \lambda w. M : A \backslash B \]

\[ \Gamma; \Delta_1; \Omega_1 \vdash N : A \quad \Gamma; \Delta_2; \Omega_2 \vdash M : A \backslash B \]
\[ \Gamma; (\Delta_1, \Delta_2); (\Omega_1, \Omega_2) \vdash N \backslash M : B \]

• Functions expect argument on the left
• Note \( \Omega_1, \Omega_2 \) is ordered, \( \Delta_1, \Delta_2 \) just linear
Ordered LF — Some Critical Rules

• Right implication

\[
\Gamma; \Delta; (\Omega, w : A) \vdash M : B \\
\Gamma; \Delta; \Omega \vdash \lambda^> w. M : B/A \\
\Gamma; \Delta_1; \Omega_1 \vdash M : B/A \\
\Gamma; \Delta_2; \Omega_2 \vdash N : A \\
\Gamma; (\Delta_1, \Delta_2); (\Omega_1, \Omega_2) \vdash M/N : B
\]

• Functions expect argument on the right

• Note \( \Omega_1, \Omega_2 \) is ordered, \( \Delta_1, \Delta_2 \) just linear
OLF Representation Principles

- Canonical forms and decidability extend
- OLF conservatively extends Linear LF
- All Linear LF principles still apply
- Ordered structures as ordered hypotheses
- Examples: queues, stacks, CPS transformations
- Example: parsing, in the style of Lambek
- Limitation: concurrency
- “Missing” connectives (e.g., ordered conjunction)
Operational semantics of programming languages

Earlier: Natural Semantics
  - Semantics via natural deduction
  - Not modular

Next: Linear Destination-Passing
  - Semantics via substructural deduction
  - Modular
Semantic Modularity

- Natural semantics is not modular:
  - Need to change judgments and rules
- Example: mutable store — from $e \leftrightarrow v$ to

\[
\begin{align*}
\langle s_1, e_1 \rangle & \leftrightarrow \langle s_2, \text{fn } x.e'_1 \rangle \\
\langle s_2, e_2 \rangle & \leftrightarrow \langle s_3, v_2 \rangle \\
\langle s_3, [v_2/x]e'_1 \rangle & \leftrightarrow \langle s_4, v \rangle \\
\langle s, \text{fn } x.e \rangle & \leftrightarrow \langle s, \text{fn } x.e \rangle \\
\langle s_1, e_1 e_2 \rangle & \leftrightarrow \langle s_4, v \rangle
\end{align*}
\]

- Also: concurrency, exceptions, continuations, etc.
- More abstract, *modular* presentation?
Linear Destination-Passing

• New semantic presentation: *Linear Destination-Passing* (LDP)
• Usually: dest-passing as a compiler optimization
• Here: destinations $d$ as names for values
• Frames $f$ for intermediate states
• Basic judgments $J$
  • $e \mapsto d$ evaluate $e$ with destination $d$
  • $f \mapsto d$ compute $f$ with destination $d$
  • $d=v$ value of destination $d$ is $v$
Linear Destination-Passing

- Judgment form $H$

$$H ::= \cdot \mid e \mapsto d, H \mid f \mapsto d, H \mid d=v, H$$

- $H$ ordered (later consider also linear, unrestricted)

- Overall deduction and value rule

$$d_0=v, \cdot$$

$$\vdots$$

$$e \mapsto d_0, \cdot$$

$$d=v, H$$

$$v \mapsto d, H$$
LDP Examples

• This talk:
  • Sequential evaluation
  • Parallel application
  • Futures

• Other have been worked out:
  continuations, mutable references, call-by-need, exceptions, heaps, Petri nets, $\pi$-calculus, concurrent ML
Design Criteria

- Modularity
  - Do not revise earlier specifications
- Orthogonality
  - No cross-references between features
- Substructural properties
  - Which judgments are ordered, linear, affine, unrestricted
Sequential Evaluation

- Abstractions handled by value rule
- Applications (new parameters noted [-])

\[
\begin{align*}
  e_1 & \mapsto d_1, \ d_1 \ e_2 \mapsto d, \ H \\
  e_1 \ e_2 & \mapsto d, \ H \\
  d_1 = v_1, \ d_1 \ d_2 & \mapsto d, \ H [d_1] \\
  d_1 = v_1, \ d_1 \ e_2 & \mapsto d, \ H [d_2] \\
  [v_2/x] e'_1 & \mapsto d, \ H \\
  d_2 = v_2, \ d_1 = (\text{fn} \ x. e'_1), \ d_1 \ d_2 & \mapsto d, \ H
\end{align*}
\]
Representing Basic Judgments

• Judgments as types

\[ e \mapsto d \Downarrow = \text{eval } e \Downarrow d : \text{type} \]
\[ f \mapsto d \Downarrow = \text{comp } f \Downarrow d : \text{type} \]
\[ d = v \Downarrow = \text{is } d \Downarrow v \Downarrow : \text{type} \]

• Resulting signature

eval : exp → dest → type.
comp : frame → dest → type.
is : dest → val → type.
State as Ordered Hypotheses

- First approximation: if $\mathcal{D}$ deduction of $H$ then

$$\Gamma; \cdot; \mathcal{H} \vdash \mathcal{D} : C$$

where
- $\Gamma$ declares all destinations in $H$, unrestricted
- $\mathcal{H}$ is ordered
- $C$ is a goal (e.g., repn. of $\exists v. d_0 = v$)
- Extend for more complex examples
Example: Sequential Evaluation

- Value rule

\[
\frac{d=v, H}{v \leftrightarrow d, H}
\]

\[
\text{evval} : \quad \text{eval (value V)} \quad D \rightarrow \bullet \quad \text{is} \quad D \lor V.
\]

- Here \( A \rightarrow \bullet B \) stands for \( A \setminus B \) or \( B / A \) when the choice does not matter.
Example: Sequential Evaluation

- **Applications**

\[ e_1 \mapsto d_1, \quad d_1 \cdot e_2 \mapsto d, \quad H \]

\[ e_1 \cdot e_2 \mapsto d, \quad H \quad \text{[d1]} \]

\[
\text{evapp} : \quad \text{eval} \ (\text{app E}_1 \ E_2) \ D \\
\quad \rightarrow (\exists d_1. \text{eval E}_1 \ d_1 \bullet \text{comp} \ (\text{app}_1 \ d_1 \ E_2) \ D)
\]

- Use \( \exists \) and \( \bullet \) (ordered conjunction) freely
- Add to framework later
Example: Sequential Evaluation

- App₁ frame

\[
\begin{align*}
e₂ & \mapsto d₂, \ d₁ = v₁, \ d₁ \ d₂ \mapsto d, \ H \\
d₁ = v₁, \ d₁ \ e₂ \mapsto d, \ H \quad [d₂]
\end{align*}
\]

is \( D₁ \ V₁ \bullet \ comp \ (app₁ \ D₁ \ E₂) \ D \)

\( \rightarrow \bullet \ (\exists d₂. \ eval \ E₂ \ d₂ \bullet \ is \ D₁ \ V₁ \bullet \ comp \ (app₂ \ D₁ \ d₂) \ D) \)
Example: Sequential Evaluation

- $\text{App}_2$ frame

\[
\frac{[v_2/x]e_1' \rightarrow d, H}{d_2 = v_2, d_1 = (\text{fn } x.e_1'), d_1 \ d_2 \rightarrow d, H}
\]

is $D_2 \ V_2 \cdot$ is $D_1 \ (\text{fun } (\lambda x. \ E_1' \ x)) \cdot$ comp $\ (\text{app}_2 \ D_1 \ D_2) \ D$

$\rightarrow \text{eval } (E_1' \ V_2) \ D.$
Consequences for Frameworks

- Rules have forms such as \( A \bullet B \rightarrow \exists d. C \bullet D \)
- Not available in LLF (\( \Pi, \rightarrow, \rightarrow, \&, \top \)) or OLF
- \( \bullet, \exists \) do not permit unique canonical forms
- Two prior approaches
  - Convert to classical linear logic (LO, Forum)
    \[
    A \otimes B \leftarrow \forall d. C \otimes D
    \]
  - Convert to continuation-passing style (LLF, OLF)
    \[
    (\Pi d. C \rightarrow D \rightarrow g) \rightarrow (A \rightarrow B \rightarrow g)
    \]
Limitations of Prior Frameworks

- Classical linear logic (Forum) [Miller’94] [Chirimar’95]
  - No dependencies or internal notation for proofs
  - No distinguished goal
  - Which deductions are equal?
  - Operational semantics?

- Continuation-passing style (LLF, OLF)
  - Dependencies and internal notation for proofs
  - Distinguished, but generic goal $g$
  - Too few deductions are equal
  - Inappropriate don’t-know nondeterminism
Monadic Encapsulation

- Idea: Encapsulate state in a monad!
- Move from

  \[ A \bullet B \rightarrow \exists d. C \bullet D \]

  to

  \[ B \setminus (A \setminus \{ \exists d. C \bullet D \}) \]

  where \( \setminus \) is a monadic type constructor

- Definition similar to monadic meta-language and lax logic [Moggi’89] [Pf & Davies’01]

- Use different from functional programming
Ordered Concurrent LF

- Type theory
  - Asynchronous connectives \(\setminus, /, \neg, \&, \top, \to, \Pi\) as in OLF
  - Canonical forms as in OLF
  - Synchronous connectives \(\bullet, 1,!, i, \exists\) only in monad
  - Equations for true concurrency [omitted from this talk]

- Representation principle: *Concurrent computations as monadic expressions*
- Conservative over LF, LLF, and OLF!
Ordered Concurrent LF

- **Language**

  Types

  \[ A ::= P \mid A_1 \rightarrow A_2 \mid \Pi u : A_1. A_2 \]

  \[ \mid A_1 \rightarrow A_2 \mid A_1 \& A_2 \mid \top \]

  \[ \mid A_1 \setminus A_2 \mid A_2 / A_1 \]

  \[ \mid \{S\} \]

  Synch Types

  \[ S ::= S_1 \bullet S_2 \mid 1 \mid !A \mid iA \mid \exists u : A. S \mid A \]

- **Main judgments**

  \[ \Gamma; \Delta; \Omega \vdash_{\Sigma} M : A \quad \text{object } M \text{ has type } A \]

  \[ \Gamma; \Delta; \Omega \vdash_{\Sigma} E \div S \quad \text{monadic expression } E \text{ has synch type } S \]
OCLF — Some Critical Rules

- Omit proof terms

\[
\begin{align*}
\Gamma; \Delta; \Omega \vdash & \quad \div S \\
\Gamma; \Delta; \Omega \vdash & \quad : \{S\} \\
\Gamma; \Delta; \Omega \vdash & \quad : S_1 \bullet S_2 \\
\Gamma; \Delta'; (\Omega_1, S_1, S_2, \Omega_2) \vdash & \quad \div S \\
\Gamma; (\Delta, \Delta'); (\Omega_1, \Omega, \Omega_2) \vdash & \quad \div S \\
\Gamma; \Delta_1; \Omega_1 \vdash & \quad \div S_1 \\
\Gamma; \Delta_2; \Omega_2 \vdash & \quad \div S_2 \\
\Gamma; (\Delta_1, \Delta_2); (\Omega_1, \Omega_2) \vdash & \quad \div S_1 \bullet S_2 \\
\Gamma; \Delta; \Omega \vdash & \quad : A \\
\Gamma; \Delta; \Omega \vdash & \quad \div A
\end{align*}
\]
OCLF Properties

- Official rules permit only canonical forms
- Important for adequacy theorems
- Outside monad (\(\vdash\)) just as in OLF
- Inside monad (\(\vdash\)) “true” concurrency
  - Independent elimination forms can be commuted
  - Cannot observe order of independent concurrent computation steps
- Type checking and equality are decidable
Example: Parallel Application

- Execute function and argument in parallel
- Replace application rules by:

\[
\begin{align*}
H', e_1 & \mapsto d_1, e_2 \mapsto d_2, d_1 d_2 \mapsto d, H & [d_1, d_2] \\
H', e_1 e_2 & \mapsto d, H
\end{align*}
\]

\[
\begin{align*}
H', [v_2/x]e'_1 & \mapsto d, H \\
H', d_1 = (\text{fn } x.e'_1), d_2 = v_2, d_1 d_2 & \mapsto d, H
\end{align*}
\]
Example: Parallel Application

- Application rule in LDP

\[
\frac{H', e_1 \rightarrow d_1, e_2 \rightarrow d_2, d_1 d_2 \rightarrow d, H}{H', e_1 e_2 \rightarrow d, H} \quad [d_1,d_2]
\]

- Representation in OCLF (omitting rule name)

\[
eval (\text{app} \ E_1 \ E_2) \ D \\
\quad \rightarrow \{\exists d_1. \exists d_2. \eval E_1 \ d_1 \bullet \eval E_2 \ d_2 \bullet \comp (\text{app}_2 \ d_1 \ d_2) \ D\}
\]
Example: Parallel Application

- Frame rule in LDP

\[
\frac{H', [v_2/x]e_1' \rightarrow d, H}{H', d_1=(\text{fn } x.e_1'), d_2=v_2, d_1 d_2 \rightarrow d, H}
\]

- Representation in CLF (omitting rule name)

\[
\text{is } D_1 (\text{fun } (\lambda x. E_1' x)) \bullet \text{ is } D_2 V_2 \bullet \text{ comp } (\text{app}_2 D_1 D_2) D \\
\rightarrow \bullet \{\text{eval } (E_1' V_2) D\}
\]

- Curry $\bullet$ and $\rightarrow \bullet$ to reduce to pure OCLF, e.g.

\[
A \bullet B \rightarrow \bullet \{C \bullet D\} \equiv B \backslash A \\{C \bullet D\}
\]
Example: Parallel Application

- Adequacy
- Computations from $\delta(e \mapsto d_0, \cdot)$ to $\delta(d_0=v, \cdot)$ correspond to expressions $E$ such that

$$d_0 : \text{dest}; \cdot; h \hat{=} \text{eval } \lnot e \ \delta d_0 \vdash E \triangleright \text{is } d_0 \lnot v \lnot$$

- *Exactly one* such $E$ (mod concurrent equality)
- *Concurrent computations as monadic expressions*
Sequential and Parallel Computation

- Retain order in specification
  - Sequential computation
  - Non-communicating parallel computation
- Relax order for communication
  - Example: encode Milner’s structural congruence via structural properties of hypotheses
  - Example: mutable references
- Generalize judgment form to $H; L; P$ where $H$ is ordered, $L$ is linear, $P$ is unrestricted
Other Modular Approaches

• Monadic Metalanguage [Moggi’89]
  - Insulate effects *inside* the language

• Contextual semantics [Wright & Felleisen’92]
  - Well-suited for continuations
  - Not as appropriate for concurrency?

• MSOS [Moses’02]
  - Small-step *structured operational semantics*
  - Add effect annotations
  - Not as flexible or modular in effect notation
Future Work: More Examples

- Parsing (into higher-order abstract syntax!)
- Spatial computation [Cardelli & Gordon’98] [Moody’03]
  - Index destinations by location
- Other concurrent calculi (action, join)
- Garbage collection
  - Index destinations by to-space or from-space
- Saturation-based procedures [MacAllester,Ganzinger]
- Protocols [Cervesato] [Bozzano’02]
Future Work: Implementation

- Linear Destination Passing reverse engineered from Concurrent Logical Framework!
- With minor changes, all examples here can be readily implemented in OCLF . . .
- . . . when an implementation of OCLF exists
- Issues
  - Executing LDP using OCLF operational semantics
  - Interleaving don’t-know (search) and don’t-care (concurrency) non-determinism
  - Representation of meta-theoretic proofs
Future Work: Slick Proofs

- Best formulation of meta-theoretic properties?
  - Type preservation
  - Progress
  - Termination
  - Infinite computations
- Some modularity of proofs?
Summary: LDP

- *Linear Destination Passing* as uniform and modular semantic framework for functional, imperative, and concurrent languages

- Structural properties
  - *Ordered* for pure, sequential computation
  - *Linear* for communicating concurrent computation; store
  - *Unrestricted* for memoization, continuations

- Readily specified in OCLF
Summary: OCLF

- Based on Lambek calculus, intuitionistic linear logic, and intuitionistic logic
- Conservatively extends LF, LLF, OLF
- Representation principle: 
  State as ordered or linear hypotheses
- Monadic encapsulation of state for concurrency
- True concurrency [omitted in this talk]
Discussion

- Uniform treatment of syntax (parsing), static semantics (typing), dynamic semantics (execution), and meta-theory (type soundness) of logic and programming languages in a single framework?
- Operational semantics for OCLF and parsing algorithms?
- Encoding of CCG in OCLF?
- Other applications in computational linguistics?