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 Framwork LF

- LF [Harper, Honsell & Plotkin'93]
- Language

  Atomic Types $P ::= a \mid P\ M$

  Types $A ::= P \mid A_1 \to A_2 \mid \Pi u{:}A_1.\, A_2$

  Objects $M ::= c \mid u \mid \lambda u.\, M \mid M_1\ M_2$

  Contexts $\Gamma ::= \cdot \mid \Gamma, u{:}A$

- Main judgments

  $\Gamma \vdash_\Sigma M : A$ M has type A

  $\Gamma \vdash_\Sigma M = N : A$ M is equal to N at type A

LF — Some Critical Rules

- Dependent functions

$$\frac{\Gamma, u : A \vdash M : B(u)}{\Gamma \vdash \lambda u.\, M : \Pi u : A.\, B(u)}$$

$$\frac{\Gamma \vdash M : \Pi u : A.\, B(u) \quad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B(N)}$$

- Equality is $\beta\eta$-conversion
- Equality and type-checking are decidable
- Canonical (β-normal, η-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 λ-calculus

$$\begin{array}{rrcl} \text{Expressions} & e & ::= & \mathrm{fn}\, x.e \mid e_1\ e_2 \\ \text{Values} & v & ::= & \mathrm{fn}\, x.e \end{array}$$

- Higher-order abstract syntax representation

$$\begin{array}{lcl} \mathsf{exp} & : & \mathsf{type}. \\ \mathsf{fun} & : & (\mathsf{exp} \to \mathsf{exp}) \to \mathsf{exp}. \\ \mathsf{app} & : & \mathsf{exp} \to (\mathsf{exp} \to \mathsf{exp}). \end{array}$$

- Example: $\ulcorner \mathrm{fn}\, f.\mathrm{fn}\, x.f\ x \urcorner = \mathsf{fun}\ (\lambda \mathsf{f}.\ \mathsf{fun}\ (\lambda \mathsf{x}.\ \mathsf{app}\ \mathsf{f}\ \mathsf{x}))$

Evaluation Semantics

- Evaluation judgment $e \hookrightarrow v$

$$\frac{}{\mathrm{fn}\, x.e \hookrightarrow \mathrm{fn}\, x.e} \qquad \frac{e_1 \hookrightarrow \mathrm{fn}\, x.e_1' \quad e_2 \hookrightarrow v_2 \quad [v_2/x]e_1' \hookrightarrow v}{e_1\ e_2 \hookrightarrow v}$$

- Judgments as types representation (omitting quant's)

$$\begin{array}{lcl}
\mathsf{eval} & : & \mathsf{exp} \rightarrow \mathsf{exp} \rightarrow \mathsf{type}. \\
\mathsf{evfun} & : & \mathsf{eval}\ (\mathsf{fun}\ (\lambda \mathrm{x}.\, \mathrm{E}\, \mathrm{x}))\ (\mathsf{fun}\ (\lambda \mathrm{x}.\, \mathrm{E}\, \mathrm{x})). \\
\mathsf{evapp} & : & \mathsf{eval}\ \mathrm{E}_1\ (\mathsf{fun}\ (\lambda \mathrm{x}.\, \mathrm{E}_1'\, \mathrm{x})) \\
 & & \quad \rightarrow \mathsf{eval}\ \mathrm{E}_2\ \mathrm{V}_2 \\
 & & \quad \rightarrow \mathsf{eval}\ (\mathrm{E}_1'\ \mathrm{V}_2)\ \mathrm{V} \\
 & & \quad \rightarrow \mathsf{eval}\ (\mathsf{app}\ \mathrm{E}_1\ \mathrm{E}_2)\ \mathrm{V}.
\end{array}$$

Adequacy of Representations

- There is a *compositional bijection* between expressions and *canonical* objects $\mathrm{M} : \mathsf{exp}$
- There is a *compositional bijection* between deductions of $e \hookrightarrow v$ and *canonical* objects $\mathrm{D} : \mathsf{eval}\ \ulcorner \mathsf{e} \urcorner\ \ulcorner \mathsf{v} \urcorner$
- Canonical objects are β-normal, η-long forms
- Critical role of defi nitional 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 $A ::= P \mid A_1 \to A_2 \mid \Pi u{:}A_1.\, A_2$

  $\mid A_1 \multimap A_2 \mid A_1 \,\&\, A_2 \mid \top$

  Objects $M ::= \ldots$
- Main judgment

  $\Gamma; \Delta \vdash_\Sigma M : A$ M has type A
- Γ are *unrestricted* assumptions (as before)
- Δ are *linear* assumptions (but order irrelevant)

Linear LF — Some Critical Rules

- Linear functions

$$\frac{\Gamma; \Delta, x : A \vdash M : B}{\Gamma; \Delta \vdash \hat{\lambda} x.\, M : A \multimap B}$$

$$\frac{\Gamma; \Delta_1 \vdash M : A \multimap B \quad \Gamma; \Delta_2 \vdash N : A}{\Gamma; (\Delta_1, \Delta_2) \vdash M \hat{} N : B}$$

- Unrestricted application

$$\frac{\Gamma; \Delta \vdash M : \Pi u : A.\, B(u) \quad \Gamma; \cdot \vdash N : A}{\Gamma; \Delta \vdash M\ N : B(N)}$$

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*

Scope of Linear LF

- 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 $A ::= P \mid A_1 \to A_2 \mid \Pi u{:}A_1.\, A_2$

  $\mid A_1 \multimap A_2 \mid A_1 \,\&\, A_2 \mid \top$

  $\mid A_1 \backslash A_2 \mid A_2 / A_1$

- Main judgment

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

- Γ is *unrestricted*, Δ is *linear*
- Ω is *ordered* (as in the Lambek calculus)

Ordered LF — Some Critical Rules

- Left implication

$$\frac{\Gamma; \Delta; (w : A, \Omega) \vdash M : B}{\Gamma; \Delta; \Omega \vdash \lambda^{<} w.\, M : A \backslash B}$$

$$\frac{\Gamma; \Delta_1; \Omega_1 \vdash N : A \quad \Gamma; \Delta_2; \Omega_2 \vdash M : A \backslash B}{\Gamma; (\Delta_1, \Delta_2); (\Omega_1, \Omega_2) \vdash N \backslash M : B}$$

- Functions expect argument on the left
- Note Ω_1, Ω_2 is ordered, Δ_1, Δ_2 just linear

Ordered LF — Some Critical Rules

- Right implication

$$\frac{\Gamma; \Delta; (\Omega, w : A) \vdash M : B}{\Gamma; \Delta; \Omega \vdash \lambda^{>} w.\, M : B/A}$$

$$\frac{\Gamma; \Delta_1; \Omega_1 \vdash M : B/A \quad \Gamma; \Delta_2; \Omega_2 \vdash N : A}{\Gamma; (\Delta_1, \Delta_2); (\Omega_1, \Omega_2) \vdash M/N : B}$$

- Functions expect argument on the right
- Note Ω_1, Ω_2 is ordered, Δ_1, Δ_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)

Framework Applications

- 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 $e \hookrightarrow v$ to

$$\frac{}{\langle s, \mathrm{fn}\, x.e \rangle \hookrightarrow \langle s, \mathrm{fn}\, x.e \rangle} \qquad \frac{\langle s_1, e_1 \rangle \hookrightarrow \langle s_2, \mathrm{fn}\, x.e_1' \rangle \quad \langle s_2, e_2 \rangle \hookrightarrow \langle s_3, v_2 \rangle \quad \langle s_3, [v_2/x]e_1' \rangle \hookrightarrow \langle s_4, v \rangle}{\langle s_1, e_1\, e_2 \rangle \hookrightarrow \langle s_4, v \rangle}$$

- 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
  - $f \rightarrowtail d$ compute f with destination d
  - $d{=}v$ value of destination d is v

Linear Destination-Passing

- Judgment form H

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

- H ordered (later consider also linear, unrestricted)
- Overall deduction and value rule

$$\frac{}{\begin{array}{c} d_0{=}v, \cdot \\ \vdots \\ e \mapsto d_0, \cdot \end{array}} \qquad \frac{d{=}v, H}{v \mapsto d, H}$$

LDP Examples

- This talk:
  - Sequential evaluation
  - Parallel application
  - Futures
- Other have been worked out:
  continuations, mutable references, call-by-need, exceptions, heaps, Petri nets, π-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 $[-]$)

$$\frac{e_1 \mapsto d_1, d_1\, e_2 \rightarrowtail d, H}{e_1\, e_2 \mapsto d, H}\ [d_1]$$

$$\frac{e_2 \mapsto d_2, d_1{=}v_1, d_1\, d_2 \rightarrowtail d, H}{d_1{=}v_1, d_1\, e_2 \rightarrowtail d, H}\ [d_2]$$

$$\frac{[v_2/x]e_1' \mapsto d, H}{d_2{=}v_2, d_1{=}(\mathrm{fn}\, x.e_1'), d_1\, d_2 \rightarrowtail d, H}$$

Representing Basic Judgments

- Judgments as types

$$\begin{array}{lcll} \ulcorner e \mapsto d \urcorner & = & \mathsf{eval}\ \ulcorner e \urcorner\ d & : \quad \mathsf{type} \\ \ulcorner f \rightarrowtail d \urcorner & = & \mathsf{comp}\ \ulcorner f \urcorner\ d & : \quad \mathsf{type} \\ \ulcorner d{=}v \urcorner & = & \mathsf{is}\ d\ \ulcorner v \urcorner & : \quad \mathsf{type} \end{array}$$

- Resulting signature

$$\begin{array}{lcl} \mathsf{eval} & : & \mathsf{exp} \rightarrow \mathsf{dest} \rightarrow \mathsf{type}. \\ \mathsf{comp} & : & \mathsf{frame} \rightarrow \mathsf{dest} \rightarrow \mathsf{type}. \\ \mathsf{is} & : & \mathsf{dest} \rightarrow \mathsf{val} \rightarrow \mathsf{type}. \end{array}$$

State as Ordered Hypotheses

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

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

  where
  - Γ declares all destinations in H, unrestricted
  - $\ulcorner H \urcorner$ 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

$$\frac{d{=}v, H}{v \mapsto d, H}$$

evval : eval (value V) D $\multimap$ is D V.

- Here $A \multimap B$ stands for $A \backslash B$ or B / A when the choice does not matter.

Example: Sequential Evaluation

- Applications

$$\frac{e_1 \mapsto d_1, d_1\, e_2 \rightarrowtail d, H}{e_1\, e_2 \mapsto d, H}\ [d_1]$$

$$\mathsf{evapp} \;:\; \mathsf{eval}\ (\mathsf{app}\ \mathrm{E}_1\ \mathrm{E}_2)\ \mathrm{D} \multimap (\exists \mathsf{d}_1.\, \mathsf{eval}\ \mathrm{E}_1\ \mathsf{d}_1 \bullet \mathsf{comp}\ (\mathsf{app}_1\ \mathsf{d}_1\ \mathrm{E}_2)\ \mathrm{D})$$

- Use $\exists$ and $\bullet$ (ordered conjunction) freely
- Add to framework later

Example: Sequential Evaluation

- App_1 frame

$$\frac{e_2 \mapsto d_2, d_1{=}v_1, d_1\, d_2 \rightarrowtail d, H}{d_1{=}v_1, d_1\, e_2 \rightarrowtail d, H}\ [d_2]$$

is $\mathrm{D}_1\ \mathrm{V}_1 \bullet \mathsf{comp}\ (\mathsf{app}_1\ \mathrm{D}_1\ \mathrm{E}_2)\ \mathrm{D}$
$\quad \multimap \bullet\ (\exists \mathsf{d}_2.\ \mathsf{eval}\ \mathrm{E}_2\ \mathsf{d}_2 \bullet \mathsf{is}\ \mathrm{D}_1\ \mathrm{V}_1 \bullet \mathsf{comp}\ (\mathsf{app}_2\ \mathrm{D}_1\ \mathsf{d}_2)\ \mathrm{D})$

Example: Sequential Evaluation

- App_2 frame

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

$\text{is}\ \mathrm{D}_2\ \mathrm{V}_2 \bullet \text{is}\ \mathrm{D}_1\ (\text{fun}\ (\lambda \mathrm{x}.\ \mathrm{E}_1'\ \mathrm{x})) \bullet \text{comp}\ (\text{app}_2\ \mathrm{D}_1\ \mathrm{D}_2)\ \mathrm{D}$
$\quad \multimap \bullet\ \text{eval}\ (\mathrm{E}_1'\ \mathrm{V}_2)\ \mathrm{D}.$

Consequences for Frameworks

- Rules have forms such as $A \bullet B \multimap \exists d.\, C \bullet D$
- Not available in LLF $(\Pi, \rightarrow, \multimap, \&, \top)$ or OLF
- $\bullet, \exists$ do not permit unique canonical forms
- Two prior approaches
  - Convert to classical linear logic (LO, Forum)

$$A \parr B \circ\!\!- \forall d.\, C \parr D$$

  - Convert to continuation-passing style (LLF, OLF)

$$(\Pi d.\, C \multimap D \multimap g) \multimap (A \multimap B \multimap g)$$

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)
  - Depencies 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

$$A \bullet B \multimap \exists d.\, C \bullet D$$

  to

$$B \backslash (A \backslash \{\exists d.\, C \bullet D\})$$

  where $\{-\}$ is a monadic type constructor
- Defi nition 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 $\backslash, /, \multimap, \&, \top, \rightarrow, \Pi$ as in OLF
  - Canonical forms as in OLF
  - Synchronous connectives $\bullet, 1, !, \mathsf{i}, \exists$ 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 $$A ::= P \mid A_1 \to A_2 \mid \Pi u{:}A_1.\, A_2 \mid A_1 \multimap A_2 \mid A_1 \,\&\, A_2 \mid \top \mid A_1 \backslash A_2 \mid A_2 / A_1 \mid \{S\}$$

  Synch Types $$S ::= S_1 \bullet S_2 \mid 1 \mid !A \mid \mathsf{i}A \mid \exists u{:}A.\, S \mid A$$

- Main judgments

  $\Gamma; \Delta; \Omega \vdash_\Sigma M : A$ object M has type A

  $\Gamma; \Delta; \Omega \vdash_\Sigma E \div S$ monadic expression E has synch type S

OCLF — Some Critical Rules

- Omit proof terms

$$\frac{\Gamma; \Delta; \Omega \vdash \quad \div S}{\Gamma; \Delta; \Omega \vdash \quad : \{S\}}$$

$$\frac{\Gamma; \Delta; \Omega \vdash \quad : S_1 \bullet S_2 \quad \Gamma; \Delta'; (\Omega_1, S_1, S_2, \Omega_2) \vdash \quad \div S}{\Gamma; (\Delta, \Delta'); (\Omega_1, \Omega, \Omega_2) \vdash \quad \div S}$$

$$\frac{\Gamma; \Delta_1; \Omega_1 \vdash \quad \div S_1 \quad \Gamma; \Delta_2; \Omega_2 \vdash \quad \div S_2}{\Gamma; (\Delta_1, \Delta_2); (\Omega_1, \Omega_2) \vdash \quad \div S_1 \bullet S_2}$$

$$\frac{\Gamma; \Delta; \Omega \vdash \quad : A}{\Gamma; \Delta; \Omega \vdash \quad \div A}$$

OCLF Properties

- Offi cial rules permit only canonical forms
- Important for adequacy theorems
- Outside monad ($:$) just as in OLF
- Inside monad ($\div$) “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:

$$\frac{H', e_1 \mapsto d_1, e_2 \mapsto d_2, d_1\, d_2 \rightarrowtail d, H}{H', \qquad\qquad e_1\, e_2 \mapsto d, H}\ [d_1,d_2]$$

$$\frac{H', [v_2/x]e_1' \mapsto d, H}{H', d_1{=}(\mathrm{fn}\, x.e_1'), d_2{=}v_2, d_1\, d_2 \rightarrowtail d, H}$$

Example: Parallel Application

- Application rule in LDP

$$\frac{H', e_1 \mapsto d_1, e_2 \mapsto d_2, d_1\, d_2 \rightarrowtail d, H}{H', \qquad\qquad e_1\, e_2 \mapsto d, H}\ [d_1,d_2]$$

- Representation in OCLF (omitting rule name)

$$\mathsf{eval}\ (\mathsf{app}\ \mathrm{E}_1\ \mathrm{E}_2)\ \mathrm{D}$$
$$\multimap \{\exists \mathsf{d}_1.\, \exists \mathsf{d}_2.\, \mathsf{eval}\ \mathrm{E}_1\ \mathsf{d}_1 \bullet \mathsf{eval}\ \mathrm{E}_2\ \mathsf{d}_2 \bullet \mathsf{comp}\ (\mathsf{app}_2\ \mathsf{d}_1\ \mathsf{d}_2)\ \mathrm{D}\}$$

Example: Parallel Application

- Frame rule in LDP

$$\frac{H', [v_2/x]e_1' \mapsto d, H}{H', d_1{=}(\mathrm{fn}\, x.e_1'), d_2{=}v_2, d_1\, d_2 \rightarrowtail d, H}$$

- Representation in CLF (omitting rule name)

$$\mathsf{is}\ \mathrm{D}_1\ (\mathsf{fun}\ (\lambda \mathrm{x}.\, \mathrm{E}_1'\ \mathrm{x})) \bullet \mathsf{is}\ \mathrm{D}_2\ \mathrm{V}_2 \bullet \mathsf{comp}\ (\mathsf{app}_2\ \mathrm{D}_1\ \mathrm{D}_2)\ \mathrm{D} \multimap \{\mathsf{eval}\ (\mathrm{E}_1'\ \mathrm{V}_2)\ \mathrm{D}\}$$

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

$$\mathrm{A} \bullet \mathrm{B} \multimap \{\mathrm{C} \bullet \mathrm{D}\} \equiv \mathrm{B}\backslash\mathrm{A}\backslash\{\mathrm{C} \bullet \mathrm{D}\}$$

Example: Parallel Application

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

$$\mathsf{d}_0 : \mathsf{dest}; \cdot; \mathsf{h} \mathrel{\hat{:}} \mathsf{eval} \ulcorner e \urcorner \ \mathsf{d}_0 \vdash \mathrm{E} \div \mathsf{is} \ \mathsf{d}_0 \ \ulcorner v \urcorner$$

- *Exactly one* such E (mod concurrent equality)
- *Concurrent computations as monadic expressions*

Sequential and Parallel Computation

- Retain order in specifi cation
  - 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 [Mosses'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 specifi ed 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?