A Concurrent Logical Framework

Frank Pfenning

Carnegie Mellon University

Types Workshop

Torino, Italy, April 2003

Joint work with Iliano Cervesato, David Walker, and Kevin Watkins

Logical Frameworks

- Meta-languages for deductive systems
  - Specification
  - Implementation (reasoning *within*)
  - Meta-theory (reasoning *about*)
- Applications
  - Logics
  - Programming languages

Representation Principles

- Logical framework characteristics
  - Type theory or meta-logic
  - Representation principles
- Example: LF (this talk)
  - Theory of dependent types (λ^{Π})
  - Judgments-as-types, deductions-as-objects
- Example: HHF
  - Hereditary Harrop logic
  - Judgments-as-propositions, deductions-as-proofs

Outline

- Natural semantics in LF
  - Lack of semantic modularity
- *Linear Destination-Passing* (LDP)
  - Parallel application
  - Futures
- *Concurrent Logical Framework* (CLF)
  - Linear type theory with monadic encapsulation
  - Concurrent computations as monadic expressions

Natural Semantics

- Typing rules

$$\frac{\Gamma, x{:}\tau_1 \vdash e : \tau_2}{\Gamma \vdash \mathrm{fn}\, x.e : \tau_1 \longrightarrow \tau_2} \qquad \frac{\Gamma \vdash e_1 : \tau_2 \longrightarrow \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1\ e_2 : \tau_1}$$

Natural Semantics

- Typing rules

$$\frac{\Gamma, x{:}\tau_1 \vdash e : \tau_2}{\Gamma \vdash \mathrm{fn}\, x.e : \tau_1 \longrightarrow \tau_2} \qquad \frac{\Gamma \vdash e_1 : \tau_2 \longrightarrow \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1\ e_2 : \tau_1}$$

- Intrinsic representation (with implicit quantifiers)

$$\begin{array}{lcl} \mathsf{tp} & : & \mathsf{type}. \\ \mathsf{arrow} & : & \mathsf{tp} \longrightarrow \mathsf{tp} \longrightarrow \mathsf{tp}. \end{array}$$

$$\begin{array}{lcl} \mathsf{exp} & : & \mathsf{tp} \longrightarrow \mathsf{type}. \\ \mathsf{fun} & : & (\mathsf{exp}\ \mathrm{T}_1 \longrightarrow \mathsf{exp}\ \mathrm{T}_2) \longrightarrow \mathsf{exp}\ (\mathsf{arrow}\ \mathrm{T}_1\ \mathrm{T}_2). \\ \mathsf{app} & : & \mathsf{exp}\ (\mathsf{arrow}\ \mathrm{T}_2\ \mathrm{T}_1) \longrightarrow \mathsf{exp}\ \mathrm{T}_2 \longrightarrow \mathsf{exp}\ \mathrm{T}_1. \end{array}$$

Higher-Order Abstract Syntax

- Object variables as meta-language variables
  - Example: $\ulcorner \mathrm{fn}\, f.\mathrm{fn}\, x.f\ x \urcorner = \mathsf{fun}\ (\lambda \mathsf{f}.\ \mathsf{fun}\ (\lambda \mathsf{x}.\ \mathsf{app}\ \mathsf{f}\ \mathsf{x}))$
  - Variable renaming via LF α-conversion
  - Capture-avoiding substitution via LF β-reduction
  - *Higher-order abstract syntax*
- Syntax constraints as indexed types
  - Example: $\ulcorner \mathrm{fn}\, x.x \urcorner = \mathsf{fun}\ (\lambda \mathsf{x}.\ \mathsf{x}) : \mathsf{exp}\ \mathrm{T} \rightarrow \mathsf{exp}\ \mathrm{T}$
  - Constraint satisfaction via LF type reconstruction
  - *Type families and dependent types*

Evaluation Semantics

- Judgment $e \hookrightarrow v$

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

Evaluation Semantics

- 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

$$\begin{array}{lcl}
\mathsf{eval} & : & \mathsf{exp}\ \mathrm{T} \longrightarrow \mathsf{exp}\ \mathrm{T} \longrightarrow \mathsf{type}. \\
\mathsf{evfun} & : & \mathsf{eval}\ (\mathsf{fun}\ (\lambda \mathsf{x}.\, \mathrm{E}\, \mathsf{x}))\ (\mathsf{fun}\ (\lambda \mathsf{x}.\, \mathrm{E}\, \mathsf{x})). \\
\mathsf{evapp} & : & \mathsf{eval}\ \mathrm{E}_1\ (\mathsf{fun}\ (\lambda \mathsf{x}.\, \mathrm{E}_1'\, \mathsf{x})) \\
 & & \quad \longrightarrow \mathsf{eval}\ \mathrm{E}_2\ \mathrm{V}_2 \\
 & & \quad \longrightarrow \mathsf{eval}\ (\mathrm{E}_1'\ \mathrm{V}_2)\ \mathrm{V} \\
 & & \quad \longrightarrow \mathsf{eval}\ (\mathsf{app}\ \mathrm{E}_1\ \mathrm{E}_2)\ \mathrm{V}.
\end{array}$$

Judgments as Types

- Deductions as objects
  - Proof checking via LF type checking
  - Proof search via LF logic programming
- Hypothetical proofs as LF functions
  - Powerful device (typing judgments, logics)
  - Requires admissibility of weakening and contraction
- Relaxing structural requirements
  - LLF [Cervesato & Pf'96], RLF [Ishtiaq & Pym'98]
  - OLF [Polakow & Pf'99] (Ordered Logical Framework)

Non-Modular Extensions

- Example: mutable store

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

- Others: exceptions, continuations, futures, etc.
- More abstract, *modular* presentation?
- Exploit substructural frameworks
  - Linear logic as a logic of state

Linear Destination-Passing I

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

Linear Destination-Passing II

- Judgment form

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

- Linear (ordered, affine, unrestricted also useful)
- Overall deduction and value rule

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

LDP Examples

- This talk:
  - Sequential evaluation
  - Parallel application
  - Futures
  - Continuations
- Note modularity and orthogonality
- Note substructural properties

Sequential Evaluation

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

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

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

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

Parallel Application

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

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

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

Futures

- Futures as in Multilisp [Halstead'85]
- Use destinations p as *promises*
- New value form $\mathrm{promise}\, p : \tau$ for $p{:}\tau$

$$\frac{e_1 \mapsto p_1, d{=}\mathrm{promise}\, p_1, H}{\mathrm{future}\, e_1 \mapsto d, H}\ [p_1]$$

$$\frac{d'{=}v_1, p_1{=}v_1, H}{d'{=}\mathrm{promise}\, p_1, p_1{=}v_1, H}$$

- Form $p_1{=}v_1$ must be *affine* (remain to the end)

Continuations

- Use destinations d as *continuations*
- New values $\mathrm{cont}\, d : \tau\, \mathrm{cont}$ for $d{:}\tau$

$$\frac{[\mathrm{cont}\, d/k]e_1 \mapsto d, H}{\mathrm{callcc}\, k.e_1 \mapsto d, H} \qquad \frac{e_1 \mapsto d_1, \mathrm{throw}\, d_1\, e_2 \rightarrowtail d, H}{\mathrm{throw}\, e_1\, e_2 \mapsto d, H}\ [d_1]$$

$$\frac{e_2 \mapsto d_2, H}{d_1{=}\mathrm{cont}\, d_2, \mathrm{throw}\, d_1\, e_2 \rightarrowtail d, H}$$

- All computations $f \rightarrowtail d$ must be *unrestricted*
- Reflects non-linear use of control stack

Other Examples

- ML and Haskell language families
  - Products, disjoint sums, recursive types, polymorphism
  - Call-by-name, call-by-need
  - Mutable store, exceptions
  - Concurrent ML
- Concurrent calculi
  - Petri nets
  - π-calculus
  - Multiset Rewriting (MSR)

Logical Framework Representation

- Syntax as indexed types (straightforward)
- Judgments as types (straightforward)
- State as linear hypotheses
- Rules and deductions

Representing Syntax

- Distinguish values from expressions

$$\begin{array}{lcl} \mathsf{val} & : & \mathsf{tp} \longrightarrow \mathsf{type}. \\ \mathsf{value} & : & \mathsf{val}\ \mathrm{T} \longrightarrow \mathsf{exp}\ \mathrm{T}. \end{array}$$

$$\begin{array}{lcl} \mathsf{fun} & : & (\mathsf{val}\ \mathrm{T}_1 \longrightarrow \mathsf{exp}\ \mathrm{T}_2) \longrightarrow \mathsf{val}\ (\mathsf{arrow}\ \mathrm{T}_1\ \mathrm{T}_2). \\ \mathsf{app} & : & \mathsf{exp}\ (\mathsf{arrow}\ \mathrm{T}_1\ \mathrm{T}_2) \longrightarrow \mathsf{exp}\ \mathrm{T}_1 \longrightarrow \mathsf{exp}\ \mathrm{T}_2. \end{array}$$

- Introduce destination parameters and frames

$$\begin{array}{lcl} \mathsf{dest} & : & \mathsf{tp} \longrightarrow \mathsf{type}. \\ \mathsf{frame} & : & \mathsf{tp} \longrightarrow \mathsf{type}. \end{array}$$

$$\begin{array}{lcl} \mathsf{app}_1 & : & \mathsf{dest}\ (\mathsf{arrow}\ \mathrm{T}_1\ \mathrm{T}_2) \longrightarrow \mathsf{exp}\ \mathrm{T}_1 \longrightarrow \mathsf{frame}\ \mathrm{T}_2. \\ \mathsf{app}_2 & : & \mathsf{dest}\ (\mathsf{arrow}\ \mathrm{T}_1\ \mathrm{T}_2) \longrightarrow \mathsf{dest}\ \mathrm{T}_1 \longrightarrow \mathsf{frame}\ \mathrm{T}_2. \end{array}$$

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

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

State as Linear Hypotheses

- Representation principle (for LLF, RLF, CLF): *State as linear hypotheses*
- First approximation: if $\mathcal{D}$ deduction of H then

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

  where
  - Γ declares all destinations in H, unrestricted
  - $\ulcorner H \urcorner$ is linear
  - C is a goal (e.g., repn. of $\exists v.\ d_{\mathrm{answer}} = v$)
- Refine for affine and unrestricted judgments

Example: Sequential Evaluation I

- Value rule

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

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

Example: Sequential Evaluation I

- Value rule

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

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

- Applications

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

evapp : eval (app E_1 E_2) D
$\multimap$ ($\exists \mathsf{d}_1$. eval E_1 d_1 $\otimes$ comp (app_1 d_1 E_2) D)

Example: Sequential Evaluation II

- App_1 frame

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

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

Example: Sequential Evaluation III

- App_2 frame

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

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

Consequences for Frameworks

- Rules have forms such as $A \otimes B \multimap \exists d.\, C \otimes D$
- Not available in LLF $(\Pi, \multimap, \&, \top)$ or RLF
- $\otimes, \exists$ 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)

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

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) [Cervesato & Pf'96]
  - 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

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

  to

$$A \multimap B \multimap \{\exists d.\, C \otimes D\}$$

  where $\{-\}$ is a monadic type constructor
- Definition similar to monadic meta-language and lax logic [Moggi'89] [Pf & Davies'01]
- Use different from functional programming

Concurrent Logical Framework

- Type theory
  - Asynchronous connectives $\multimap, \&, \top, \Pi$ as in LLF
  - Canonical forms as in LLF
  - Synchronous connectives $\otimes, 1, !, \exists$ only in monad
  - Equations for true concurrency (later in this talk)
- Representation principle:
  Concurrent computations as monadic expressions
- Conservative over LF and LLF!

Example: Futures in CLF

- Recall specification in LDP ($p{=}v$ affine)

$$\frac{e_1 \mapsto p_1, d{=}\mathrm{promise}\, p_1, H}{\mathrm{future}\, e_1 \mapsto d, H}\ [p_1] \qquad \frac{d'{=}v_1, p_1{=}v_1, H}{d'{=}\mathrm{promise}\, p_1, p_1{=}v_1, H}$$

- Encoding in CLF

$$\mathsf{evf} : \mathsf{eval}\ (\mathsf{future}\ \mathrm{E}_1)\ \mathrm{D} \multimap \{\exists \mathsf{p}_1.\, \mathsf{eval}\ \mathrm{E}_1\ \mathsf{p}_1 \otimes \mathsf{is}\ \mathrm{D}\ (\mathsf{promise}\ \mathsf{p}_1)\}$$

$$\mathsf{evp} : \mathsf{is}\ \mathrm{D}'\ (\mathsf{promise}\ \mathrm{P}_1) \multimap \mathsf{is}\ \mathrm{P}_1\ \mathrm{V}_1 \multimap \{\mathsf{is}\ \mathrm{D}'\ \mathrm{V}_1 \otimes \mathsf{is}\ \mathrm{P}_1\ \mathrm{V}_1\}$$

- Goal $\{(\exists \mathsf{v}.\, \mathsf{is}\ \mathsf{d}_0\ \mathsf{v}) \otimes \top\}$ for affine hypotheses
- Other choices possible

CLF Type Theory

- Types

  Atomic $P ::= a \mid P\ N$

  Asynch $A ::= P \mid A_1 \multimap A_2 \mid \Pi u{:}A_1.\, A_2 \mid A_1 \,\&\, A_2 \mid \top$
  $\mid \{S\}$

  Synch $S ::= S_1 \otimes S_2 \mid 1 \mid \exists u{:}A.\, S \mid\ !A \mid A$

- Signatures and contexts

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

  Linear $\Delta ::= \cdot \mid \Delta, x \hat{:} A$

  Global $\Sigma ::= \cdot \mid \Sigma, a{:}K \mid \Sigma, c{:}A$

Objects

- Permit only canonical forms
- New approach to defining logical frameworks
- Objects

  Normal $N ::= \hat{\lambda}x.\, N \mid \lambda u.\, N \mid \langle N_1, N_2 \rangle \mid \langle\rangle \mid R$

  $\mid \{E\}$

  Atomic $R ::= c \mid u \mid x \mid R \hat{} N \mid R\ N \mid \pi_1 R \mid \pi_2 R$

- Normal $\sim$ intro, atomic $\sim$ elim
- No types inside objects

Expressions

- New objects in CLF

Normal	$N ::= \ldots \mid \{E\}$
Atomic	$R ::= \ldots$
Expressions	$E ::= \mathsf{let}\ \{p\} = R\ \mathsf{in}\ E \mid M$
Monadic	$M ::= M_1 \otimes M_2 \mid 1 \mid [N, M] \mid !N \mid N$
Patterns	$p ::= p_1 \otimes p_2 \mid 1 \mid [u, p] \mid !u \mid x$

- Expressions are synchronous eliminations
- Monadic objects are synchronous introductions
- Future extension: synchronous additives $(\oplus, 0)$

Judgments Defining CLF

- Bi-directional (Γ, Δ, Σ always given)

$$\Gamma; \Delta \vdash_\Sigma N \Leftarrow A \quad N, A \text{ given}$$

$$\Gamma; \Delta \vdash_\Sigma R \Rightarrow A \quad R \text{ given, } A \text{ computed}$$

$$\Gamma; \Delta \vdash_\Sigma E \leftarrow S \quad E, S \text{ given}$$

$$\Gamma; \Delta; \Psi \vdash_\Sigma E \leftarrow S \quad \Psi, E, S \text{ given}$$

$$\Gamma; \Delta \vdash_\Sigma M \Leftarrow S \quad M, S \text{ given}$$

- Pattern contexts $\Psi ::= p \hat{:} S, \Psi \mid \cdot$
- Omit judgments for types, kinds, contexts, sigs

Selected Rules, Expressions

- Expressions $\Gamma; \Delta \vdash E \leftarrow S$

$$\frac{\Gamma; \Delta \vdash E \leftarrow S}{\Gamma; \Delta \vdash \{E\} \Leftarrow \{S\}} \{\}\mathbf{I}$$

$$\frac{\Gamma; \Delta_1 \vdash R \Rightarrow \{S_0\} \quad \Gamma; \Delta_2; p \hat{:} S_0 \vdash E \leftarrow S}{\Gamma; \Delta_1, \Delta_2 \vdash (\mathsf{let}\ \{p\} = R\ \mathsf{in}\ E) \leftarrow S} \{\}\mathbf{E}$$

$$\frac{\Gamma; \Delta \vdash M \Leftarrow S}{\Gamma; \Delta \vdash M \leftarrow S} \Leftarrow\leftarrow$$

Selected Rules, Patterns

- Decompose patterns deterministically (Ψ ordered)

$$\frac{\Gamma; \Delta; p_1 \hat{:} S_1, p_2 \hat{:} S_2, \Psi \vdash E \leftarrow S}{\Gamma; \Delta; p_1 \otimes p_2 \hat{:} S_1 \otimes S_2, \Psi \vdash E \leftarrow S} \otimes\mathbf{L}$$

$$\frac{\Gamma, u : A; \Delta; p \hat{:} S_0, \Psi \vdash E \leftarrow S}{\Gamma; \Delta; [u, p] \hat{:} \exists u : A.\, S_0, \Psi \vdash E \leftarrow S} \exists\mathbf{L}$$

$$\frac{\Gamma; \Delta, x \hat{:} A; \Psi \vdash E \leftarrow S}{\Gamma; \Delta; x \hat{:} A, \Psi \vdash E \leftarrow S} \mathrm{A}\mathbf{L}$$

$$\frac{\Gamma; \Delta \vdash E \leftarrow S}{\Gamma; \Delta; \cdot \vdash E \leftarrow S} \leftarrow\leftarrow$$

Definitional Equality

- Maintain types in canonical form [Watkins et al.'02]

$$\frac{\Gamma;\Delta \vdash R \Rightarrow \Pi u\!:\!A.\,B \quad \Gamma;\cdot \vdash N \Leftarrow A}{\Gamma;\Delta \vdash R\ N \Rightarrow [N/u\!:\!A^-]^{\mathrm{a}}B}\ \Pi\mathbf{E}$$

- $[N/u\!:\!A^-]^{\mathrm{a}}B$ defined inductively on A and B
- Returns canonical form
- Avoids explicit β and η conversion
- Allows bi-directional checking
- Subtle for expressions [Pf & Davies'01]

True Concurrency

- True concurrency (my definition):

  We cannot observe the order of independent, concurrent actions

- Internalize via definitional equality on expressions

$$\mathsf{let}\ \{p_1\} = R_1\ \mathsf{in}\ (\mathsf{let}\ \{p_2\} = R_2\ \mathsf{in}\ E)$$
$$\stackrel{c}{=}_1 \quad \mathsf{let}\ \{p_2\} = R_2\ \mathsf{in}\ (\mathsf{let}\ \{p_1\} = R_1\ \mathsf{in}\ E)$$

  - No variable in p_1 free in R_2
  - No variable in p_2 free in R_1
  - No variable bound in p_1 and p_2

Concurrency Equations

- Extend $\stackrel{c}{=}_1$ to congruence
- Alternative defn. via bisimulation [Watkins et al.'02]
- Capture traces on Petri nets [Cervesato et al.'02]
- Summary of definitional equality
  - α-conversion
  - Canonical forms avoid $\beta\eta$-conversion
  - Concurrency equations
- Captures only minimal object-language equality
  - For example, *not* π-calculus bisimulation or testing

Example: Parallel Application I

- Application rule in LDP

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

- Representation in CLF (omitting rule name)

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

Example: Parallel Application II

- Frame rule in LDP

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

- Representation in CLF (omitting rule name)

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

Example: Parallel Application III

- Adequacy, assuming no effects
- Computations from $(e \mapsto d_0, \cdot)$ to $(d_0{=}v, \cdot)$ for $e : \tau$ correspond to expressions E such that

$$\mathsf{d}_0 : \mathsf{dest}\ulcorner\tau\urcorner; \mathsf{h} \Updownarrow \mathsf{eval}\ulcorner e\urcorner\ \mathsf{d}_0 \vdash \mathrm{E} \leftarrow \mathsf{is}\ \mathsf{d}_0\ulcorner v\urcorner$$

- *Exactly one* such E (mod concurrent equality)
- On this fragment, bijection to computations in natural semantics $e \hookrightarrow v$
- *Concurrent computations as monadic expressions*

Example: Futures Revisited

- Recall encoding in CLF

$$\mathsf{evf} : \mathsf{eval}\ (\mathsf{future}\ \mathrm{E}_1)\ \mathrm{D} \multimap \{\exists \mathsf{p}_1.\, \mathsf{eval}\ \mathrm{E}_1\ \mathsf{p}_1 \otimes \mathsf{is}\ \mathrm{D}\ (\mathsf{promise}\ \mathsf{p}_1)\}$$

$$\mathsf{evp} : \mathsf{is}\ \mathrm{D}'\ (\mathsf{promise}\ \mathrm{P}_1) \multimap \mathsf{is}\ \mathrm{P}_1\ \mathrm{V}_1 \multimap \{\mathsf{is}\ \mathrm{D}'\ \mathrm{V}_1 \otimes \mathsf{is}\ \mathrm{P}_1\ \mathrm{V}_1\}$$

- Hypothesis is $\mathrm{P}_1\ \mathrm{V}_1$ is consumed and re-created
- Two independent reads *do not* commute
- Spurious synchronization

Example: Futures Reformulated I

- Explicit step from linear to unrestricted dests.

$$\frac{e_1 \mapsto d_1, \mathrm{deliver}\, d_1 \rightarrowtail p_1, d{=}\mathrm{promise}\, p_1, H}{\mathrm{future}\, e_1 \mapsto d, H}\ [d_1,p_1]$$

$$\mathsf{eval}\ (\mathsf{future}\ \mathrm{E}_1)\ \mathrm{D} \multimap \{\exists \mathsf{d}_1.\ \exists \mathsf{p}_1.\ \mathsf{eval}\ \mathrm{E}_1\ \mathsf{d}_1 \otimes \mathsf{comp}\ (\mathsf{deliver}\ \mathsf{d}_1)\ \mathsf{p}_1 \otimes \mathsf{is}\ \mathrm{D}\ (\mathsf{promise}\ \mathsf{p}_1)\}.$$

Example: Futures Reformulated II

- Creating unrestricted destinations (note !)

$$\frac{H, p_1{=}v_1}{d_1{=}v_1, \mathrm{deliver}\, d_1 \rightarrowtail p_1, H}$$

is $\mathrm{D}_1\ \mathrm{V}_1 \multimap \mathsf{comp}\ (\mathsf{deliver}\ \mathrm{D}_1)\ \mathrm{P}_1 \multimap \{!\ \mathsf{is}\ \mathrm{P}_1\ \mathrm{V}_1\}$.

- Using unrestricted destinations (note $\rightarrow$)

$$\frac{d'{=}v_1, H, p_1{=}v_1}{d'{=}\mathrm{promise}\, p_1, H, p_1{=}v_1}$$

is $\mathrm{D}'\ (\mathsf{promise}\ \mathrm{P}_1) \multimap \mathsf{is}\ \mathrm{P}_1\ \mathrm{V}_1 \rightarrow \{\mathsf{is}\ \mathrm{D}'\ \mathrm{V}_1\}$.

Example: Futures Reformulated III

- Using unrestricted destinations (note $\rightarrow$)

$$\frac{d'{=}v_1, H, p_1{=}v_1}{d'{=}\mathrm{promise}\, p_1, H, p_1{=}v_1}$$

$\mathsf{is}\ \mathrm{D}'\ (\mathsf{promise}\ \mathrm{P}_1) \multimap \mathsf{is}\ \mathrm{P}_1\ \mathrm{V}_1 \rightarrow \{\mathsf{is}\ \mathrm{D}'\ \mathrm{V}_1\}.$

- $A \rightarrow B = \Pi x\!:\!A.\, B \sim\, !A \multimap B$, x new

Example: Continuations

- LDP specification ($f \rightarrowtail d$ *unrestricted*)

$$\frac{[\mathrm{cont}\, d/k]e_1 \mapsto d, H}{\mathrm{callcc}\, k.e_1 \mapsto d, H} \qquad \frac{e_1 \mapsto d_1, \mathrm{throw}\, d_1\, e_2 \rightarrowtail d, H}{\mathrm{throw}\, e_1\, e_2 \mapsto d, H}\ [d_1]$$

$$\frac{e_2 \mapsto d_2, H}{d_1{=}\mathrm{cont}\, d_2, \mathrm{throw}\, d_1\, e_2 \rightarrowtail d, H}$$

- CLF encoding

$\mathsf{eval}\ (\mathsf{callcc}\ (\lambda \mathsf{k}.\, \mathrm{E}_1\ \mathsf{k}))\ \mathrm{D} \multimap \{\mathsf{eval}\ (\mathrm{E}_1\ (\mathsf{cont}\ \mathrm{D}))\ \mathrm{D}\}.$

$\mathsf{eval}\ (\mathsf{throw}\ \mathrm{E}_1\ \mathrm{E}_2)\ \mathrm{D}$
$\quad \multimap \{\exists \mathsf{d}_1.\, \mathsf{eval}\ \mathrm{E}_1\ \mathsf{d}_1 \otimes\, !\, \mathsf{comp}\ (\mathsf{throw}\ \mathsf{d}_1\ \mathrm{E}_2)\ \mathrm{D}\}.$

$\mathsf{is}\ \mathrm{D}_1\ (\mathsf{cont}\ \mathrm{D}_2) \multimap \mathsf{comp}\ (\mathsf{throw}\ \mathrm{D}_1\ \mathrm{E}_2)\ \mathrm{D} \rightarrow \{\mathsf{eval}\ \mathrm{E}_2\ \mathrm{D}_2\}.$

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 concurrent computation
  - *Affine* for store
  - *Unrestricted* for memoization, continuations

Other Modular Approaches

- Monadic Metalanguage [Moggi'89]
  - Insulate effects *inside* the language
  - Exploit encapsulation instead in CLF
- MSOS [Mosses'02]
  - Small-step *structured operational semantics*
  - Add effect annotations
  - Not as flexible or modular in effect notation
- Contextual semantics [Wright & Felleisen'92]
  - Well-suited for continuations
  - Not as appropriate for concurrency

Summary: CLF

- *Concurrent Logical Framework*
  as meta-language for concurrent systems and programming languages
- Type theory
  - LF ($\Pi, \rightarrow$) and LLF ($\multimap, \&, \top$) — canonical forms
  - CLF ($\{-\}, \otimes, 1, !, \exists$) — concurrency equations
- Representation principles
  - Judgments as types, proofs as objects
  - State as linear hypotheses
  - Concurrent computations as monadic expressions

Debate

- How much do we want the framework to internalize and how much do we want to encode?
- Logical frameworks should conceptualize and intrinsically support recurring notions
  - Variable renaming (LF α-conversion)
  - Capture-avoiding substitution (LF β-reduction)
  - True concurrency (CLF concurrency equation)
- Payoff in clarity and elegance of specification
- LDP was reverse engineered from CLF representations!

Other Considerations

- Meta-programming (next)
- Exploiting framework expressiveness for meta-reasoning
  - For LF: $\mathcal{M}_2^+$ [Schürmann'00]
  - For HHF: $\mathrm{FOL}^{\Delta N}$ [McDowell & Miller'97]
  - For HHF: Hybrid [Ambler, Crole & Momigliano'02]
  - For LHHF: [Tiu & Miller'03]
  - For LLF: [McCreight & Schürmann'03]
  - For CLF: future work

Operational Semantics

- Operational semantics of LF (Elf)
  - Computation via proof search
  - Backchaining search
  - Don’t-know non-determinism (backtracking)
  - Pattern unification, constraints
- Sound and non-deterministically complete
  - Discovered proof represents deduction
  - Finite failure means no proof exists
  - May not find all proofs (depth-first)
  - Recently improved with tabling [Pientka’02]

Example: Evaluation Semantics

- Judgment $e \hookrightarrow v$

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

- Signatures as logic programs

$$\begin{array}{lcl}
\mathsf{eval} & : & \mathsf{exp}\ \mathrm{T} \rightarrow \mathsf{exp}\ \mathrm{T} \rightarrow \mathsf{type}. \\
\mathsf{evfun} & : & \mathsf{eval}\ (\mathsf{fun}\ (\lambda \mathrm{x}.\, \mathrm{E}\, \mathrm{x}))\ (\mathsf{fun}\ (\lambda \mathrm{x}.\, \mathrm{E}\, \mathrm{x})). \\
\mathsf{evapp} & : & \mathsf{eval}\ (\mathsf{app}\ \mathrm{E}_1\ \mathrm{E}_2)\ \mathrm{V} \\
 & & \quad \leftarrow \mathsf{eval}\ \mathrm{E}_1\ (\lambda \mathrm{x}.\, \mathrm{E}_1'\, \mathrm{x}) \\
 & & \quad \leftarrow \mathsf{eval}\ \mathrm{E}_2\ \mathrm{V}_2 \\
 & & \quad \leftarrow \mathsf{eval}\ (\mathrm{E}_1'\ \mathrm{V}_2)\ \mathrm{V}.
\end{array}$$

Extension to LLF

- Imperative logic programming
- Linear hypotheses as resources (store)
- Generalizes first-order linear logic programming [Hodas & Miller'94]
- Efficient implementation
- Conservative extension of LF search
- Linear higher-order pattern unification open

Extension to CLF

- Mostly speculative
- Outside monad $\rightarrow, \Pi$ (LF), $\multimap, \&, \top$ (LLF)
  - Backward chaining
  - Don’t-know non-determinism
  - Backtracking
- Inside monad $\otimes, 1, !, \exists$
  - Forward chaining
  - Don’t-care non-determinism
  - No backtracking

Semantic Justification

- Don’t-care non-deterministic search finds at most one representative of equivalence class of proofs
- Signature must avoid deadlock
- Appropriate to execute concurrent programs or specifications
  - Futures
  - Concurrent ML
  - π-calculus

Future Work

- Linear Destination-Passing
  - Proof techniques for linear destination-passing
  - Use in undergraduate programming language course
- Concurrent Logical Framework
  - Synchronous additives $(\oplus, 0)$
  - Proof irrelevance, affine, ordered extensions
  - Co-inductive interpretation of rules
  - Operational semantics, implementation
  - Meta-theoretic reasoning

Conclusions

- Linear Destination-Passing
  - Modular presentation style
  - Well-suited for wide range of languages
- Concurrent Logical Framework
  - Monadic encapsulation of concurrency
  - Concurrent computations as monadic expressions

Extra Slides Follow

Mini-ML with Store I

- New types and expressions

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \mathrm{ref}\, e : \tau\, \mathrm{ref}} \qquad \frac{\Gamma \vdash e : \tau\, \mathrm{ref}}{\Gamma \vdash\, !\, e : \tau}$$

$$\frac{\Gamma \vdash e_1 : \tau\, \mathrm{ref} \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash (e_1 := e_2) : \mathrm{unit}}$$

Mini-ML with Store I

- New types and expressions

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \mathrm{ref}\, e : \tau\, \mathrm{ref}} \qquad \frac{\Gamma \vdash e : \tau\, \mathrm{ref}}{\Gamma \vdash\, !\, e : \tau}$$

$$\frac{\Gamma \vdash e_1 : \tau\, \mathrm{ref} \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash (e_1 := e_2) : \mathrm{unit}}$$

- Representation

$$\begin{array}{lcl} \mathsf{ref} & : & \mathsf{tp} \longrightarrow \mathsf{tp}. \\ \\ \mathsf{newref} & : & \mathsf{exp}\ \mathrm{T} \longrightarrow \mathsf{exp}\ (\mathsf{ref}\ \mathrm{T}). \\ \mathsf{assign} & : & \mathsf{exp}\ (\mathsf{ref}\ \mathrm{T}) \longrightarrow \mathsf{exp}\ \mathrm{T} \longrightarrow \mathsf{exp}\ (\mathsf{unit}). \\ \mathsf{deref} & : & \mathsf{exp}\ (\mathsf{ref}\ \mathrm{T}) \longrightarrow \mathsf{exp}\ \mathrm{T}. \end{array}$$

Mini-ML with Store II

- Locations l (a new value, not in source)
- Stores $s ::= \cdot \mid s, l{=}v$
- Store typings $\Lambda ::= \cdot \mid \Lambda, l{:}\tau$
- Operations: allocate, read, write
- Judgment $\langle s, e\rangle \hookrightarrow \langle s', v\rangle$

$$\frac{\langle s, e\rangle \hookrightarrow \langle s', v\rangle \quad l = \mathrm{alloc}(s')}{\langle s, \mathrm{ref}\, e\rangle \hookrightarrow \langle (s', l{=}v), l\rangle} \qquad \frac{\langle s, e\rangle \hookrightarrow \langle s', l\rangle \quad v = \mathrm{read}(s', l)}{\langle s, !\, e\rangle \hookrightarrow \langle s', v\rangle}$$

$$\frac{\langle s, e_1\rangle \hookrightarrow \langle s_1, l\rangle \quad \langle s_1, e_2\rangle \hookrightarrow \langle s_2, v_2\rangle \quad s' = \mathrm{write}(s_2, l, v_2)}{\langle s, e_1 := e_2\rangle \hookrightarrow \langle s', \langle\,\rangle\rangle}$$

Sequential Evaluation with Store I

- Use destinations l as mutable locations
- New value form $\mathrm{cell}\, l : \tau\, \mathrm{ref}$ for $l{:}\tau$
- Allocate ($\mathrm{ref}\, e_1$) and read ($!\, e_1$)

$$\frac{e_1 \mapsto d_1, \mathrm{ref}\ d_1 \rightarrowtail d, H}{\mathrm{ref}\ e_1 \mapsto d, H}\ [d_1] \qquad \frac{d{=}\mathrm{cell}\, l_1, H, l_1{=}v_1}{d_1{=}v_1, \mathrm{ref}\ d_1 \rightarrowtail d, H}\ [l_1]$$

$$\frac{e_1 \mapsto d_1, !\, d_1 \rightarrowtail d, H}{!\, e_1 \mapsto d, H}\ [d_1] \qquad \frac{d{=}v, H, l{=}v}{d_1{=}\mathrm{cell}\, l, !\, d_1 \rightarrowtail d, H, l{=}v}$$

- Form $l{=}v$ must be *affine*

Sequential Evaluation with Store II

- Write $(e_1 := e_2)$

$$\frac{e_1 \mapsto d_1, (d_1 := e_2) \rightarrowtail d, H}{(e_1 := e_2) \mapsto d, H}\ [d_1]$$

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

$$\frac{d{=}\langle\,\rangle, H, l_1{=}v_2}{d_2{=}v_2, d_1{=}\mathrm{cell}\, l_1, (d_1 := d_2) \rightarrowtail d, H, l_1{=}v_1}$$

Call-by-Need (Lazy Evaluation)

- Thunks as destinations t

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

$$\frac{[\mathrm{force}\, t_2/x]e_1' \mapsto d, H, \mathrm{delay}\, e_2 \rightarrowtail t_2}{d_1{=}(\mathrm{fn}\, x.e_1'), d_1\, e_2 \rightarrowtail d, H}\ [t_2]$$

$$\frac{e_2 \mapsto t_2, \mathrm{force}\, t_2 \mapsto d, H}{\mathrm{force}\, t_2 \mapsto d, H, \mathrm{delay}\, e_2 \rightarrowtail t_2} \qquad \frac{d{=}v_2, H, t_2{=}v_2}{t_2{=}v_2, \mathrm{force}\, t_2 \mapsto d, H}$$

- Forms $t{=}v$, $\mathrm{delay}\, e \rightarrowtail t$ must be *affine*