Specifying Properties of Concurrent Computations in CLF

Frank Pfenning

Carnegie Mellon University

LFM Workshop
Cork, Ireland, July 2004

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

• Traditional view
  • Formalize logics in a foundationally neutral framework
  • Implement mathematics within logics

• Main object logics of interest:
  • Classical predicate calculus and set theory
  • Classical type theory
  • Constructive logic
Logical Frameworks

• Modern (my!) view
  • Isolate fundamental principles of logic and computation
  • Specify and reason about programming languages

• Many object systems of interest, for example:
  • Monadic types (effects)
  • Modal types (distributed computation)
  • Temporal types (partial evaluation)
  • Affine types (resource bounds)
  • Linear types (storage management)
  • Separation logic (reasoning with heaps)
Reasoning about Prog. Langs.

- Programming languages are mathematical objects
- Can be studied in traditional frameworks
  - Not much guidance and support
  - Appropriate especially for legacy languages
- Should be studied in modern frameworks
  - Significant design guidance
  - Useful especially for new formulations or languages
Outline

- LF via canonical forms
- $\pi$-calculus
- Concurrent LF (CLF)
- True concurrency
LF via Canonical Forms

- LF representation methodology
  - Judgments as types
  - Proofs as *canonical* objects
  - Representations $\vdash_\pi$ as compositional bijections
- Restrict LF to canonical forms
- Types and type families
  
  Atomic $P ::= a \mid P \cdot N$

  Normal $A ::= P \mid \prod x : A_1. A_2$
Normal and Atomic Objects

- Principal judgments
  \[ \Gamma \vdash_{\Sigma} N \leftrightarrow A \quad N \text{ is canonical at type } A \]
  \[ \Gamma \vdash_{\Sigma} R \Rightarrow A \quad R \text{ is atomic with type } A \]

- Signature \( \Sigma \) (fixed) for constants \( a : K \) and \( c : A \)

- Context \( \Gamma \) for variables \( x : A \)

- Objects
  \[
  \begin{align*}
  \text{Normal} & \quad N ::= \lambda x. N \mid R \\
  \text{Atomic} & \quad R ::= c \mid x \mid R \ N
  \end{align*}
  \]
Bidirectional Typing, Synthesis

• Atomic Objects

\[
\begin{align*}
\Gamma(x) &= A & \Sigma(c) &= A \\
\Gamma \vdash x \Rightarrow A & & \Gamma \vdash c \Rightarrow A
\end{align*}
\]

\[
\Gamma \vdash R \Rightarrow \Pi x : A. B & & \Gamma \vdash N \Leftarrow A \\
\Gamma \vdash R \; N \Rightarrow [N/x]_A^B & & \Pi E
\]

• \([N/x]_A^B\) is hereditary substitution
  • Returns normal type or fails finitely
  • Terminates by nested induction on \(A\) and \(B\)
  • Always succeeds if \(\Pi x : A. B \Leftarrow\) type and \(N \Leftarrow A\)
Bidirectional Typing, Checking

- Normal Objects

\[
\begin{align*}
\Gamma, x : A \vdash N & \iff B \\
\Gamma \vdash \lambda x. N & \iff \Pi x : A. B \\
\Gamma \vdash R \Rightarrow P' & \quad P' = P \\
\Gamma \vdash R & \iff P
\end{align*}
\]

- Test \( P' = P \) is \( \alpha \)-conversion
- \( \beta \) syntactically ruled out
- \( \eta \) ruled out via long normal forms
Further Representation Principles

- Object variables as LF variables
  - Renaming of bound variables for free
  - Capture-avoiding substitution for free
- Object assumptions as LF hypotheses
  - Exchange, weakening, and contraction for free
  - Logical substitution principles for free
- Remains foundationally neutral, for example:
  - Classical and intuitionistic logic
  - Call-by-name and call-by-value
Some Consequences

- LF methodology clarifies hypothetical judgments
- LF cannot distinguish between $\alpha$-equivalent terms
- LF cannot discern order of hypotheses
- LF cannot prevent unused hypotheses
- LF cannot prevent multiple use of hypotheses
Asynchronous $\pi$-Calculus

• Syntax [Gordon & Jeffrey’03]

$$
P, Q ::= \text{stop} \mid (P \mid Q) \mid \text{new}(x); P \\
\quad \mid \text{out } a\langle b \rangle \mid \text{inp } a(x); P \\
\quad \mid \text{repeat } P \mid \text{choose } P \mid Q \\
\quad \mid \text{begin } L; P \mid \text{end } L; P
$$

• $a$ for free names, $x$ for bound names
• choose is non-determinism
• [begin and end are correspondence assertions]
Syntax Representation in LF

stop : pr.
par : pr → pr → pr.
new : (nm → pr) → pr.
out : nm → nm → pr.
inp : nm → (nm → pr) → pr.
repeat : pr → pr.
choose : pr → pr → pr.

«stop» = stop

«P | Q» = par «P» «Q»

«new(x); P» = new (λx. «P»)

«out a⟨b⟩» = out a b

«inp a(x); P» = inp a (λx. «P»)

«repeat P» = repeat «P»

«choose P Q» = choose «P» «Q»
Operational Semantics

- Do not consider repeat for now
  
  Process states \( \Psi ::= \cdot \mid \Psi, P \)

  Channels \( \Sigma ::= \cdot \mid \Sigma, a \)

  Configurations \( \Sigma; \Psi \)

  \( \Psi \) and \( \Sigma \) are considered modulo exchange

  One-Step Transition \( \Sigma; \Psi \rightarrow \Sigma'; \Psi' \)
Transition Rules

\begin{align*}
(\Sigma; \Psi, \text{stop}) & \rightarrow (\Sigma; \Psi) \\
(\Sigma; \Psi, (P | Q)) & \rightarrow (\Sigma; \Psi, P, Q) \\
(\Sigma; \Psi, (\text{new}(x); P)) & \rightarrow (\Sigma, a; \Psi, [a/x]P) \quad a \notin \Sigma \\
(\Sigma; \Psi, (\text{out } a\langle b\rangle), (\text{inp } a(x); P)) & \rightarrow (\Sigma; \Psi, [b/x]P) \\
(\Sigma; \Psi, (\text{choose } P \quad Q)) & \rightarrow (\Sigma; \Psi, P) \\
(\Sigma; \Psi, (\text{choose } P \quad Q)) & \rightarrow (\Sigma; \Psi, Q)
\end{align*}
State as Linear Hypotheses

• Capture state via linear hypotheses
• Exchange as a structural congruence
• Represent configuration \( a_1, \ldots, a_n; P_1, \ldots, P_\kappa \) as

\[
a_1 : \text{nm}, \ldots, a_n : \text{nm}; p_1 \uparrow \text{run} \lceil P_1 \rceil, \ldots, p_\kappa \uparrow \text{run} \lceil P_\kappa \rceil
\]

• \( \text{run} : \text{pr} \rightarrow \text{type} \) a type family indexed by processes
• Hypotheses \( a : \text{nm} \) are \textit{unrestricted}
• Hypotheses \( p : \text{run} \lceil P \rceil \) are \textit{linear}
Representing State Transitions

- Need framework with unrestricted and linear hypotheses
- Represent rules as linear implications

\[ (\Sigma; \Psi, \text{stop}) \rightarrow (\Sigma; \Psi) \]

\[ \text{run (stop)} \rightarrow \circ 1 \]

\[ (\Sigma; \Psi, (P | Q)) \rightarrow (\Sigma; \Psi, P, Q) \]

\[ \Pi P : \text{pr.} \Pi Q : \text{pr. run (par } P Q) \rightarrow \circ \text{ run } P \otimes \text{ run } Q \]
Representing State Transitions

- Substitution via framework substitution

\[
(\Sigma; \Psi, (\text{new}(x); P)) \rightarrow (\Sigma, a; \Psi, [a/x]P)
\]

\[
\Pi P : \text{nm} \rightarrow \text{pr. run} \ (\text{new} \ (\lambda x. P \ x)) \rightarrow \forall a : \text{nm. run} \ (P \ a)
\]

\[
(\Sigma; \Psi, (\out a\langle b\rangle), (\inp a(x); P)) \rightarrow (\Sigma; \Psi, [b/x]P)
\]

\[
\Pi A : \text{nm. } \Pi B : \text{nm. } \Pi P : \text{nm} \rightarrow \text{pr. run} \ (\out A \ B) \rightarrow \forall \text{run} \ (\inp A \ (\lambda x. P \ x)) \rightarrow \forall \text{run} \ (P \ B)
\]

\[
(\Sigma; \Psi, (\text{choose } P \ Q)) \rightarrow (\Sigma; \Psi, P)
\]

\[
\Pi P : \text{pr. } \Pi Q : \text{pr. run} \ (\text{choose } P \ Q) \rightarrow \forall \text{run } P
\]
Lack of Canonical Forms

• Linear type theory with
  $\rightarrow, \otimes, 1, \exists, \& , \top, !, \Pi$
  does not possess canonical forms

• Problem: let-forms and commuting conversions
  
  $\left( \text{let } x_1 \otimes x_2 = M \text{ in } N \right) N' = \left( \text{let } x_1 \otimes x_2 = N \text{ in } NN' \right)$
  
  • Normal forms are not type-directed

• Representation is not a compositional bijection

• [Correct definition of equality is open]

• [Decidability of definitional equality is open]
Monadic Encapsulation

- Encapsulate connectives $\otimes, \mathbf{1}, \exists, \!$ in a monad
- Concurrent computations in the monad $\{\_\}$
- LF ($\Pi$) and Linear LF ($\rightarrow, \&,$, $\top$) outside the monad
- Restores canonical forms
  - $\{\text{let } x_1 \otimes x_2 = M \text{ in } N\} \ N'$ no longer well-typed
- Ensures conservativity over LF and LLF
  - All LF and LLF encodings work as before
  - Representations are still compositional bijections
CLF Type Theory

• Types

  Atomic \( P ::= a \mid P \, N \)

  Asynch \( A ::= P \mid A_1 \rightarrow A_2 \mid \Pi u : A_1. A_2 \mid A_1 \& A_2 \mid \top \mid \{ S \} \)

  Synch \( S ::= S_1 \otimes S_2 \mid 1 \mid \exists u : A. S \mid !A \mid A \)

• Signatures and contexts

  Unrestricted \( \Gamma ::= \cdot \mid \Gamma, x : A \)

  Linear \( \Delta ::= \cdot \mid \Delta, x : A \)

  Global \( \Sigma ::= \cdot \mid \Sigma, a : K \mid \Sigma, c : A \)
CLF Objects

• As before, permit only canonical forms

• Objects

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

\mid \{E\}

Atomic \( R ::= c \mid x \mid R^\wedge N \mid R N \mid \pi_1 R \mid \pi_2 R \)

• No types inside objects
CLF Expressions

- New objects in CLF

  Expressions \( E ::= \text{let } \{p\} = R \text{ in } E \mid M \)

  Monadic \( M ::= M_1 \otimes M_2 \mid 1 \mid \langle N, M \rangle \mid !N \mid N \)

  Patterns \( p ::= p_1 \otimes p_2 \mid 1 \mid \langle x, p \rangle \mid !x \mid x \)

- Expressions are synchronous eliminations
- Monadic objects are synchronous introductions
• Bi-directional ($\Gamma, \Delta, \Sigma$ always given)

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

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

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

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

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

• Pattern contexts $\Psi ::= p \vdash S, \Psi | \cdot$

• Omit judgments for types, kinds, contexts, sigs
Selected Rules, Expressions

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

\[
\frac{\Gamma; \Delta \vdash E \leftarrow S}{\Gamma; \Delta \vdash \{E\} \leftarrow \{S\}} \quad \text{ } \{\} I
\]

\[
\frac{\Gamma; \Delta_1 \vdash R \Rightarrow \{S_0\} \quad \Gamma; \Delta_2; p:S_0 \vdash E \leftarrow S}{\Gamma; \Delta_1, \Delta_2 \vdash (\text{let } \{p\} = R \text{ in } E) \leftarrow S} \quad \text{ } \{\} E
\]

\[
\frac{\Gamma; \Delta \vdash M \leftarrow S}{\Gamma; \Delta \vdash M \leftarrow S} \quad \leftrightarrow
\]
Selected Rules, Patterns

- Decompose patterns deterministically ($\Psi$ ordered)

\[
\frac{\Gamma; \Delta; p_1 : S_1, p_2 : S_2, \Psi \vdash E \leftarrow S}{\Gamma; \Delta; p_1 \otimes p_2 : S_1 \otimes S_2, \Psi \vdash E \leftarrow S} \quad \otimes_L
\]

\[
\frac{\Gamma, u : A; \Delta; p : S_0, \Psi \vdash E \leftarrow S}{\Gamma; \Delta; [u, p] : \exists u : A. S_0, \Psi \vdash E \leftarrow S} \quad \exists_L
\]

\[
\frac{\Gamma; \Delta, x : A; \Psi \vdash E \leftarrow S}{\Gamma; \Delta; x : A, \Psi \vdash E \leftarrow S} \quad \text{AL}
\]

\[
\frac{\Gamma; \Delta \vdash E \leftarrow S}{\Gamma; \Delta; \cdot \vdash E \leftarrow S} \quad \leftrightarrow
\]
Example Revisited

• Suppress $\Pi$-quantifier prefix (reconstructed)

\[
\begin{align*}
\text{ev\_stop} & : \text{run\ stop} \implies \{1\}. \\
\text{ev\_par} & : \text{run} (\text{par} P Q) \implies \{\text{run } P \otimes \text{run } Q\}\}. \\
\text{ev\_new} & : \text{run} (\text{new} (\lambda x. P x)) \implies \{\exists x : \text{nm. run } (P x)\}. \\
\text{ev\_sync} & : \text{run} (\text{out} A B) \implies \text{run} (\text{inp} A (\lambda x. P x)) \implies \{\text{run } (P B)\}.
\end{align*}
\]
Other $\pi$-Calculus Constructs

- Non-deterministic choice
  
  \[
  \text{ev}_\text{choose}_1 : \text{run } (\text{choose } P_1 P_2) \rightarrow \{-\text{run } P_1\}.
  \]
  
  \[
  \text{ev}_\text{choose}_2 : \text{run } (\text{choose } P_1 P_2) \rightarrow \{-\text{run } P_2\}.
  \]

- Process replication
  
  \[
  \text{ev}_\text{repeat} : \text{run } (\text{repeat } P) \rightarrow \{-!\text{run } P\}.
  \]

- Extended process states of informal presentation
  
  Process states \(\Psi ::= \cdot | \Psi, P^1 | \Psi, P^\omega\)
New Representation Principle

• **Concurrent computations as monadic expressions**

• Consider configuration $a, b; \text{out } a\langle b\rangle, (\text{inp } a(x); \text{stop})$

• Represented by

\[
    \Gamma_0 = (a : \text{nm}, b : \text{nm})
\]
\[
    \Delta_0 = (p_1 \uparrow \text{out } a \ b, p_2 \uparrow \text{inp } a \ (\lambda x. \text{stop}))
\]

• **Two-step computation**

\[
    \Gamma_0; \Delta_0 \vdash \text{let } \{p_3\} = \text{ev\_sync}^{\uparrow p_1 \uparrow p_2} \text{ in}
\]
\[
    \text{let } \{1\} = \text{ev\_stop}^{\uparrow p_3} \text{ in } \langle \rangle \leftarrow \top
\]
True Concurrency

• True concurrency (my definition):
  \[\text{We cannot observe the order of independent actions}\]

• Internalize via definitional equality on expressions

\[
(\text{let } \{p_1\} = R_1 \text{ in } (\text{let } \{p_2\} = R_2 \text{ in } E))
= (\text{let } \{p_2\} = R_2 \text{ in } (\text{let } \{p_1\} = R_1 \text{ in } E))
\]

• 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 \(p_2\)
• Both sides well-typed
Consequences for CLF

- Only one rule is affected

\[ \Gamma \vdash R \Rightarrow P' \quad P' = P \]

\[ \Gamma \vdash R \Leftarrow P \]

- Decidability is easily retained
- Can be formulated as bi-simulation
- Let-forms under concurrency equations constitute a syntax for directed acyclic graphs
Computations as Objects

• Can manipulate concurrent computations
  • Index families with monadic expressions
  • Example: correspondence assertions
    (see preliminary proceedings)
  • Framework cannot discern the order of independent computation steps!

• Universal properties of computations are harder
Summary of CLF

- LF (Π)
  - Judgments as types
  - Hypothetical judgments
- LLF (→, &, ⊤)
  - State as linear hypotheses
- CLF (1, ⊗, ∃, !)
  - Concurrent computations as monadic expressions
Selected Related Work

• LF [Harper, Honsell & Plotkin’93] [Felty’91]
• LLF [Hodas & Miller’94] [Cervesato & Pfenning’96] [Ishtiaq & Pym’98]
• Multiset rewriting [Cervesato’03] [Cervesato & Stehr’03]
• Forum [Chirimar’95] [Miller’96]
• Meta-reasoning [McDowell & Miller’97] [Miller & Tiu’03] [McCreight & Schürmann’04] [Affeldt & Kobayashi’04] [Meseguer’04]
• Further examples [Watkins et al.’03,04] [Cervesato et al.’03]
Future Work

• Proof irrelevant, affine, ordered extensions
• Families of monads
• Unification of CLF objects
• Operational semantics for CLF signatures
• Co-inductive rule interpretation
• Meta-theoretic reasoning
Summary

• Logical frameworks to study fundamental principles of logic and computation
• CLF: true concurrency
• Joint work with Iliano Cervesato, David Walker, and Kevin Watkins
• For more, see CMU-CS-02-101 and CMU-CS-02-102