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
- π-calculus
- Concurrent LF (CLF)
- True concurrency

LF via Canonical Forms

- LF representation methodology
  - Judgments as types
  - Proofs as *canonical* objects
  - Representations $\ulcorner \_ \urcorner$ as compositional bijections
- Restrict LF to canonical forms
- Types and type families

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

  Normal $A ::= P \mid \Pi x : A_1 . A_2$

Normal and Atomic Objects

- Principal judgments

  $\Gamma \vdash_\Sigma N \Leftarrow A$ N is canonical at type A

  $\Gamma \vdash_\Sigma R \Rightarrow A$ R is atomic with type A

- Signature Σ (fixed) for constants $a : K$ and $c : A$
- Context Γ for variables $x : A$
- Objects

  Normal $N ::= \lambda x.\, N \mid R$

  Atomic $R ::= c \mid x \mid R\ N$

Bidirectional Typing, Synthesis

- Atomic Objects

$$\frac{\Gamma(x) = A}{\Gamma \vdash x \Rightarrow A} \qquad \frac{\Sigma(c) = A}{\Gamma \vdash c \Rightarrow A}$$

$$\frac{\Gamma \vdash R \Rightarrow \Pi x\!:\!A.\, B \quad \Gamma \vdash N \Leftarrow A}{\Gamma \vdash R\ N \Rightarrow [N/x]_A B}\ \Pi\mathbf{E}$$

- $[N/x]_A B$ is *hereditary substitution*
  - Returns normal type or fails finitely
  - Terminates by nested induction on A and B
  - Always succeeds if $\Pi x\!:\!A.\, B \Leftarrow \mathsf{type}$ and $N \Leftarrow A$

Bidirectional Typing, Checking

- Normal Objects

$$\frac{\Gamma, x : A \vdash N \Leftarrow B}{\Gamma \vdash \lambda x.\, N \Leftarrow \Pi x : A.\, B}\ \Pi\mathbf{I}$$

$$\frac{\Gamma \vdash R \Rightarrow P' \quad P' = P}{\Gamma \vdash R \Leftarrow P}\ \Rightarrow\Leftarrow$$

- Test $P' = P$ is α-conversion
- β syntactically ruled out
- η 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 α-equivalent terms
- LF cannot discern order of hypotheses
- LF cannot prevent unused hypotheses
- LF cannot prevent multiple use of hypotheses

Asychronous π-Calculus

- Syntax [Gordon & Jeffrey'03]

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

- a for free names, x for bound names
- choose is *non-determinism*
- [begin and end are *correspondence assertions*]

Syntax Representation in LF

$$
\begin{array}{rcl}
\mathsf{stop} & : & \mathsf{pr}. \\
\mathsf{par} & : & \mathsf{pr} \to \mathsf{pr} \to \mathsf{pr}. \\
\mathsf{new} & : & (\mathsf{nm} \to \mathsf{pr}) \to \mathsf{pr}. \\
\mathsf{out} & : & \mathsf{nm} \to \mathsf{nm} \to \mathsf{pr}. \\
\mathsf{inp} & : & \mathsf{nm} \to (\mathsf{nm} \to \mathsf{pr}) \to \mathsf{pr}. \\
\mathsf{repeat} & : & \mathsf{pr} \to \mathsf{pr}. \\
\mathsf{choose} & : & \mathsf{pr} \to \mathsf{pr} \to \mathsf{pr}.
\end{array}
$$

$$
\begin{array}{rcl}
\ulcorner \mathsf{stop} \urcorner & = & \mathsf{stop} \\
\ulcorner P \mid Q \urcorner & = & \mathsf{par}\ \ulcorner P \urcorner\ \ulcorner Q \urcorner \\
\ulcorner \mathsf{new}(x); P \urcorner & = & \mathsf{new}\ (\lambda x. \ulcorner P \urcorner) \\
\ulcorner \mathsf{out}\ a\langle b \rangle \urcorner & = & \mathsf{out}\ a\ b \\
\ulcorner \mathsf{inp}\ a(x); P \urcorner & = & \mathsf{inp}\ a\ (\lambda x. \ulcorner P \urcorner) \\
\ulcorner \mathsf{repeat}\ P \urcorner & = & \mathsf{repeat}\ \ulcorner P \urcorner \\
\ulcorner \mathsf{choose}\ P\ Q \urcorner & = & \mathsf{choose}\ \ulcorner P \urcorner\ \ulcorner Q \urcorner
\end{array}
$$

Operational Semantics

- Do not consider repeat for now

  Process states $\Psi ::= \cdot \mid \Psi, P$

  Channels $\Sigma ::= \cdot \mid \Sigma, a$

  Configurations $\Sigma; \Psi$

- Ψ and Σ are considered modulo exchange
- One-Step Transition $\Sigma; \Psi \longrightarrow \Sigma'; \Psi'$

Transition Rules

$$
\begin{aligned}
(\Sigma; \Psi, \mathsf{stop}) &\longrightarrow (\Sigma; \Psi) \\
(\Sigma; \Psi, (P \mid Q)) &\longrightarrow (\Sigma; \Psi, P, Q) \\
(\Sigma; \Psi, (\mathsf{new}(x); P)) &\longrightarrow (\Sigma, a; \Psi, [a/x]P) \quad a \notin \Sigma \\
(\Sigma; \Psi, (\mathsf{out}\ a\langle b\rangle), (\mathsf{inp}\ a(x); P)) &\longrightarrow (\Sigma; \Psi, [b/x]P) \\
(\Sigma; \Psi, (\mathsf{choose}\ P\ Q)) &\longrightarrow (\Sigma; \Psi, P) \\
(\Sigma; \Psi, (\mathsf{choose}\ P\ Q)) &\longrightarrow (\Sigma; \Psi, Q)
\end{aligned}
$$

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

$$a_1 : \mathsf{nm}, \ldots, a_n : \mathsf{nm}; p_1 \mathbin{\hat{:}} \mathsf{run}\ulcorner P_1 \urcorner, \ldots, p_\kappa \mathbin{\hat{:}} \mathsf{run}\ulcorner P_\kappa \urcorner$$

- $\mathsf{run} : \mathsf{pr} \longrightarrow \mathsf{type}$ a type family indexed by processes
- Hypotheses $a : \mathsf{nm}$ are *unrestricted*
- Hypotheses $p \mathbin{\hat{:}} \mathsf{run}\ulcorner P \urcorner$ are *linear*

Representing State Transitions

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

$$(\Sigma; \Psi, \mathsf{stop}) \quad \longrightarrow (\Sigma; \Psi)$$

$$\mathsf{run}\ (\mathsf{stop}) \quad \multimap 1$$

$$(\Sigma; \Psi, (P \mid Q)) \quad \longrightarrow (\Sigma; \Psi, P, Q)$$

$$\Pi P : \mathsf{pr}.\, \Pi Q : \mathsf{pr}.\, \mathsf{run}\ (\mathsf{par}\ P\ Q) \quad \multimap \mathsf{run}\ P \otimes \mathsf{run}\ Q$$

Representing State Transitions

- Substitution via framework substitution

$$(\Sigma; \Psi, (\mathsf{new}(x); P)) \longrightarrow (\Sigma, a; \Psi, [a/x]P)$$

$$\Pi P : \mathsf{nm} \rightarrow \mathsf{pr}.\, \mathsf{run}\ (\mathsf{new}\ (\lambda x.\, P\ x)) \multimap \exists a : \mathsf{nm}.\, \mathsf{run}\ (P\ a)$$

$$(\Sigma; \Psi, (\mathsf{out}\ a\langle b\rangle), (\mathsf{inp}\ a(x); P)) \longrightarrow (\Sigma; \Psi, [b/x]P)$$

$$\Pi A : \mathsf{nm}.\, \Pi B : \mathsf{nm}.\, \Pi P : \mathsf{nm} \rightarrow \mathsf{pr}.$$
$$\mathsf{run}\ (\mathsf{out}\ A\ B) \multimap \mathsf{run}\ (\mathsf{inp}\ A\ (\lambda x.\, P\ x)) \multimap \mathsf{run}\ (P\ B)$$

$$(\Sigma; \Psi, (\mathsf{choose}\ P\ Q)) \longrightarrow (\Sigma; \Psi, P)$$

$$\Pi P : \mathsf{pr}.\, \Pi Q : \mathsf{pr}.\, \mathsf{run}\ (\mathsf{choose}\ P\ Q) \multimap \mathsf{run}\ P$$

Lack of Canonical Forms

- Linear type theory with
  $\multimap, \otimes, 1, \exists, \&, \top, !, \Pi$
  does not possess canonical forms
- Problem: let-forms and commuting conversions
  - $(\text{let } x_1 \otimes x_2 = M \text{ in } N)N' = (\text{let } x_1 \otimes x_2 = N \text{ in } NN')$
  - 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 $\otimes, 1, \exists, !$ in a *monad*
- Concurrent computations in the monad $\{\_\}$
- LF (Π) and Linear LF ($\multimap, \&, \top$) 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 $A ::= P \mid A_1 \multimap A_2 \mid \Pi u{:}A_1.\, A_2 \mid A_1\ \&\ A_2 \mid \top$
  $\mid \{S\}$

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

- Signatures and contexts

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

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

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

CLF Objects

- As before, permit only canonical forms
- Objects

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

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

- No types inside objects

CLF Expressions

- New objects in CLF

  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 $p ::= p_1 \otimes p_2 \mid 1 \mid [x, p] \mid !x \mid x$

- Expressions are synchronous eliminations
- Monadic objects are synchronous introductions

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)

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

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

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

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

Example Revisited

- Suppress Π-quantifier prefix (reconstructed)

$$\begin{aligned}
\mathsf{ev_stop} &: \mathsf{run}\ \mathsf{stop} \multimap \{1\}. \\
\mathsf{ev_par} &: \mathsf{run}\ (\mathsf{par}\ P\ Q) \multimap \{\mathsf{run}\ P \otimes \mathsf{run}\ Q\}. \\
\mathsf{ev_new} &: \mathsf{run}\ (\mathsf{new}\ (\lambda x.\, P\ x)) \multimap \{\exists x : \mathsf{nm}.\, \mathsf{run}\ (P\ x)\}. \\
\mathsf{ev_sync} &: \mathsf{run}\ (\mathsf{out}\ A\ B) \multimap \mathsf{run}\ (\mathsf{inp}\ A\ (\lambda x.\, P\ x)) \\
&\qquad \multimap \{\mathsf{run}\ (P\ B)\}.
\end{aligned}$$

Other π-Calculus Constructs

- Non-deterministic choice

$$\mathsf{ev\_choose}_1 \;:\; \mathsf{run}\,(\mathsf{choose}\; P_1\; P_2) \multimap \{\mathsf{run}\; P_1\}.$$

$$\mathsf{ev\_choose}_2 \;:\; \mathsf{run}\,(\mathsf{choose}\; P_1\; P_2) \multimap \{\mathsf{run}\; P_2\}.$$

- Process replication

$$\mathsf{ev\_repeat} \;:\; \mathsf{run}\,(\mathsf{repeat}\; P) \multimap \{!\mathsf{run}\; P\}.$$

- Extended process states of informal presentation

Process states $\Psi ::= \cdot \mid \Psi, P^1 \mid \Psi, P^\omega$

New Representation Principle

- *Concurrent computations as monadic expressions*
- Consider configuration $a, b; \mathsf{out}\ a\langle b\rangle, (\mathsf{inp}\ a(x); \mathsf{stop})$
- Represented by

$$\Gamma_0 = (a : \mathsf{nm}, b : \mathsf{nm})$$

$$\Delta_0 = (p_1 \stackrel{\wedge}{:} \mathsf{out}\ a\ b, p_2 \stackrel{\wedge}{:} \mathsf{inp}\ a\ (\lambda x.\, \mathsf{stop}))$$

- Two-step computation

$$\Gamma_0; \Delta_0 \vdash \mathsf{let}\ \{p_3\} = \mathsf{ev\_sync}\hat{}p_1\hat{}p_2\ \mathsf{in}$$

$$\mathsf{let}\ \{1\} = \mathsf{ev\_stop}\hat{}p_3\ \mathsf{in}\ \langle\rangle \leftarrow \top$$

True Concurrency

- True concurrency (my definition):

  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

$$\frac{\Gamma \vdash R \Rightarrow P' \quad P' = P}{\Gamma \vdash R \Leftarrow P} \Rightarrow\Leftarrow$$

- 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 ($\multimap, \&, \top$)
  - State as linear hypotheses
- CLF ($\{1, \otimes, \exists, !\}$)
  - 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