A Concurrent Logical Framework

Iliano Cervesato  iliamo@itd.nrl.navy.mil
ITT Industries, inc @ NRL Washington, DC
http://www.cs.stanford.edu/~iliano

(Joint work with Frank Pfenning, David Walker, and Kevin Watkins)
CLF

Where it comes from
- Logical Frameworks
- The LF approach

What it is
- True concurrency
- Monadic encapsulation
- A canonical approach

What’s next?
All about Logical Frameworks

Represent and reason about object systems

Languages, logics, …
- Often semi-formalized as deductive systems
- Reasoning often informal

Benefits
- Formal specification of object system
- Automate verification of reasoning arguments
- Feed back into other tools
  - Theorem provers, PCC, …
The LF Way

Identify fundamental mechanisms and build them into the framework (soundly!)

- done (right) once and for all instead of each time

- Modular constructions: \([\Sigma\text{-Algebras}]\)
  - \(\text{app } f \ a\)

- Variable binding, \(\alpha\)-renaming, substitution [LF]
  - \(\lambda x. \ x+1\)

- Disposable, updateable cell [LLF]
  - \(\lambda^s'. \ f^s\)

- True concurrency [CLF]
It’s all about *Adequacy*

- Adequacy: correctness of the transcription
- LF: make adequacy as simple as possible

---

**Object system**

- Task
  - complex
  - long
  - tedious

**Representation**

**Informal**

**Automated**

rather than

(Gödel numbers)

---

I. Cervesato: *A Concurrent Logical Framework*
Representation Targets

Mottos, mottos, mottos …

\[ 3 + 5 = 8 \]

Judgment

(a statement we want to make)

\[ N : \text{ev} \ ( + \ 3 \ 5 ) \ 8 \]

object type

\[ \text{LF: judgments-as-types / proofs-as-objects} \]

\[ \text{LLF: state-as-linear-hypotheses / imperative-computations-as-linear-functions} \]

\[ \text{CLF: concurrent-computations-as-monadic-expressions / …} \]

\[ \text{nextLF: …} \]
Make it Canonical, Sam

Each object of interest has exactly 1 representation

- Canonical objects:
  - η-long, β-normal _LF term
  - Decidable, computable
But what is LLF?

- **Types**
  - (“asynchronous” constructors of ILL)
  - \( A ::= a | \Pi x:A. B | A \rightarrow B | A \land B | T \)

- **Terms**
  - \( N ::= x | \lambda x:A. N | N_1 N_2 \)
  - \( \lambda^x:A. N | N_1 ^N_2 \)
  - \( <N_1,N_2> | \text{fst } N | \text{snd } N \)
  - \( <> \)

- **Main judgment**
  - \( \Gamma ; \Delta |- N : A \)
CLF
An Example

Many instances can be executing concurrently
LLF Encoding

\[
\text{net} : \text{step} \quad o- \quad \text{net}^{\text{out}} \quad m \\
\quad o- \quad (\text{net}^{\text{in}} \quad m \quad -o \quad \text{step}) .
\]

- LLF forces continuation-passing style

- Consider 2 independent applications:
  \[
  \lambda n^1_i . \text{net} \quad ^{n^1_o} \quad (\lambda n^2_i . \text{net} \quad ^{n^2_o} \quad C) \\
  \lambda n^2_i . \text{net} \quad ^{n^2_o} \quad (\lambda n^1_i . \text{net} \quad ^{n^1_o} \quad C)
  \]

Shoulde be indistinguishable (true concurrency)

- Equate them at the meta-level

  \[
  \text{same-trace} \quad T_1 \quad T_2 \quad o- \quad ...
  \]

Never-ending even for small system!
Encoding in Linear logic

∀m. net^{out} m \rightarrow o \ net^{in} m

\begin{itemize}
  \item Much simpler
  \item In general, requires “synchronous” operators
    \begin{itemize}
      \item \otimes \ and \ 1
    \end{itemize}
  \item Concurrency given by “commuting conversions”
    \begin{align*}
      & \quad \text{let } x_1 \otimes y_1 = N_1 \text{ in (let } x_2 \otimes y_2 = N_2 \text{ in } M) \\
      & = \quad \text{let } x_2 \otimes y_2 = N_2 \text{ in (let } x_1 \otimes y_1 = N_1 \text{ in } M) \quad \text{if } x_i, y_i \not\in \text{FV}(R_{2,i})
    \end{align*}
  \item \ldots looks like what we want \ldots
\end{itemize}
However …

- Commuting conversions are too wild
  - Allow permutations we don’t care for

- Synchronous types destroy uniqueness of canonical forms
  - \texttt{nat:type. z:nat. s:nat->nat. c:1.}
  - Natural numbers: \texttt{z, sz, s(sz), …}
  - What about \texttt{let 1 = c in z}? What if \texttt{c} is linear?

- No good! 😞
Monadic Encapsulation

Separate synchronous and asynchronous types

- **Outside** the monad
  - LLF types (asynchronous)
  - $\eta$-long, $\beta$-normal forms

- **Inside** the monad
  - Synchronous types
  - Commuting conversions
    - Concurrency equation
  - $\eta$-long, $\beta$-normal forms

- Monad is a sandbox for synchronous behavior
CLF

Types

\[ A ::= a \mid \prod x:A. B \mid A \to B \mid A \& B \mid T \mid \{S\} \]

\[ S ::= A \mid !A \mid S_1 \otimes S_2 \mid 1 \mid \exists x:A. S \]

Terms

\[ N ::= x \mid \lambda x:A. N \mid N_1 N_2 \mid \lambda^x:A. N \mid N_1 \wedge N_2 \mid <N_1,N_2> \mid \text{fst } N \mid \text{snd } N \mid \leftrightarrow \mid \{E\} \]

\[ E ::= M \mid \text{let } \{p\} = N \text{ in } E \]

\[ M ::= N \mid !N \mid M_1 \otimes M_2 \mid 1 \mid [N,M] \]

\[ p ::= x \mid !x \mid p_1 \otimes p_2 \mid 1 \mid [x,p] \]
Example in CLF

\[ \text{net : net}^\text{in} m \rightarrow o \{ \text{net}^\text{out} m \}. \]

- Relating the 2 specifications
- 2 sets of CLF declarations
- Meta-level definition of trace transformation
  \[ \text{simplify-net \{T}^{i/o}\} \{T\} \]
  - Trivial mapping
  - Permutations handled automatically
    - No need to take action
    - Critical for more complex examples
Examples and Applications

- π-calculus
  - Synchronous
  - Asynchronous
- Concurrent ML
- Petri nets
  - Execution-sequence semantics
  - Trace semantics
- MSR security protocol specification language
- No implementation … yet …
Conclusions

CLF

- A logical framework that internalizes true concurrency
- Monadic encapsulation tames commuting conversions
- Canonical approach to meta-theory
- Good number of examples

- This is just the beginning ... plenty more to do!