A Concurrent Logical Framework

Iliano Cervesato
iliano@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: \(\Sigma\)-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*

- **Object system**
- **Representation**

Informal

→ Adequacy: correctness of the transcription

→ LF: make adequacy as simple as possible

rather than

Automated

(Gödel numbers)
Representation Targets

Mottos, mottos, mottos ...

▲ LF: judgments-as-types / proofs-as-objects

- 3+5 = 8 ⇒ N : ev (+ 3 5) 8

Judgment
(a statement we want to make)

- object
- type

▲ LLF: state-as-linear-hypotheses / imperative-computations-as-linear-functions

▲ CLF: concurrent-computations-as-monadic-expressions / ...

▲ nextLF: blablablablablabla-as-blablablablablablablabla / blablablablablablablablabla-as-blablablablablablablabla
Make it Canonical, Sam

Each object of interest has exactly 1 representation

Canonical objects:
- $\eta$-long, $\beta$-normal \_LF term
- Decidable, computable
But what is LLF?

Types

(``asynchronous'' constructors of ILL)

\[ A ::= a \mid \Pi x:A. B \mid A \to B \mid A \& B \mid T \]

Terms

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

Main judgment

\[ \Gamma ; \Delta |- N : A \]
An Example

Many instances can be executing concurrently
LLF Encoding

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

LLF forces continuation-passing style

Consider 2 independent applications:

\[
\lambda n^1_1 . \text{net} ^ \wedge n^o_1 ^ \wedge (\lambda n^1_2 . \text{net} ^ \wedge n^o_2 ^ \wedge C) \\
\lambda n^1_2 . \text{net} ^ \wedge n^o_2 ^ \wedge (\lambda n^1_1 . \text{net} ^ \wedge n^o_1 ^ \wedge C)
\]

Should be indistinguishable (true concurrency)

Equate them at the meta-level

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

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

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

- Much simpler
- In general, requires “synchronous” operators
  - \( \otimes \) and \( 1 \)
- Concurrency given by “commuting conversions”
  
  \[
  \begin{align*}
  &\text{let } x_1 \otimes y_1 = N_1 \text{ in (let } x_2 \otimes y_2 = N_2 \text{ in } M) \\
  &= \text{let } x_2 \otimes y_2 = N_2 \text{ in (let } x_1 \otimes y_1 = N_1 \text{ in } M) \\
  \end{align*}
  \]
  if \( x_1, y_1 \not\in \text{FV}(R_{2,i}) \)

- … looks like what we want …
However ...

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

- Synchronous types destroy uniqueness of canonical forms
  - natural z . s: nat -> nat . c: 1 .
  - Natural numbers: z, s z, s (s z), ...
  - What about let 1 = c in z? What if 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

\[ A ::= a \mid \Pi x:A. \ B \mid A \to B \mid A \& B \mid T \mid \{S\} \]
\[ S ::= A \mid !A \mid S_1 \otimes S_2 \mid 1 \mid \exists x:A. \ S \]

Terms

\[ N ::= x \mid \lambda x:A. \ N \mid N_1 \ N_2 \mid \lambda^x:A. \ N \mid N_1 \,^\uparrow N_2 \mid <N_1, N_2> \mid \text{fst} \ N \mid \text{snd} \ N \mid <> \mid \{E\} \]
\[ E ::= M \mid \text{let} \ \{p\} = N \ \text{in} \ E \]
\[ M ::= N \mid !N \mid M_1 \otimes M_2 \mid 1 \mid [N, M] \]
\[ p ::= x \mid !x \mid p_1 \otimes p_2 \mid 1 \mid [x, p] \]
Example in CLF

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

- 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
I. Cervesato: A Concurrent Logical Framework

Examples and Applications

- \(\pi\)-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!