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

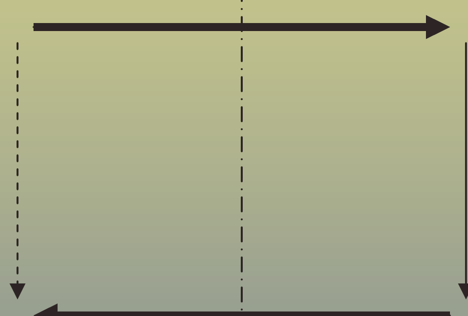
- Modular constructions: [Σ-Algebras]
  - *app f a*
- Variable binding, α-renaming, substitution [LF]
  - $\lambda x.\ x+1$
- Disposable, updateable cell [LLF]
  - $\lambda\hat{}s'.\ f\ \hat{}\ s$
- True concurrency [CLF]

It’s all about *Adequacy*

Object system | Representation

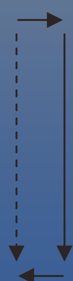
Informal

Task
- complex
- long
- tedious



Automated

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



rather than

(Gödel numbers)

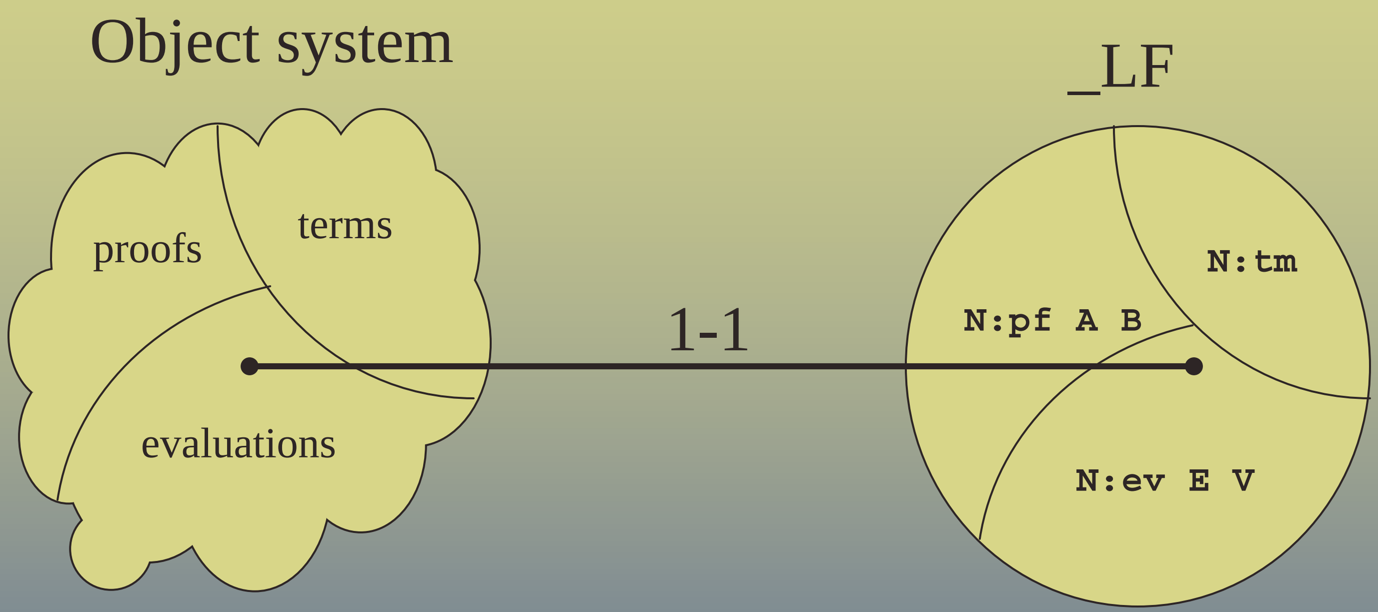
Representation Targets

Mottos, mottos, mottos …

- LF: *judgments-as-types / proofs-as-objects*
  - $\underbrace{3+5 = 8}_{\text{Judgment}}$ $\Rightarrow$ $\underbrace{\mathtt{N}}_{\text{object}}\ \mathtt{:}\ \underbrace{\mathtt{ev\ (+\ 3\ 5)\ 8}}_{\text{type}}$

    (a statement we want to make)
- LLF: *state-as-linear-hypotheses / imperative-computations-as-linear-functions*
- CLF: *concurrent-computations-as-monadic-expressions / …*
- nextLF: *blablablablablabla-**as**-blablablablablablablabla / blablablablablablablablabl-**as**-blablablablablabablablablablablablabla*

Make it Canonical, Sam



Each object of interest has exactly 1 representation

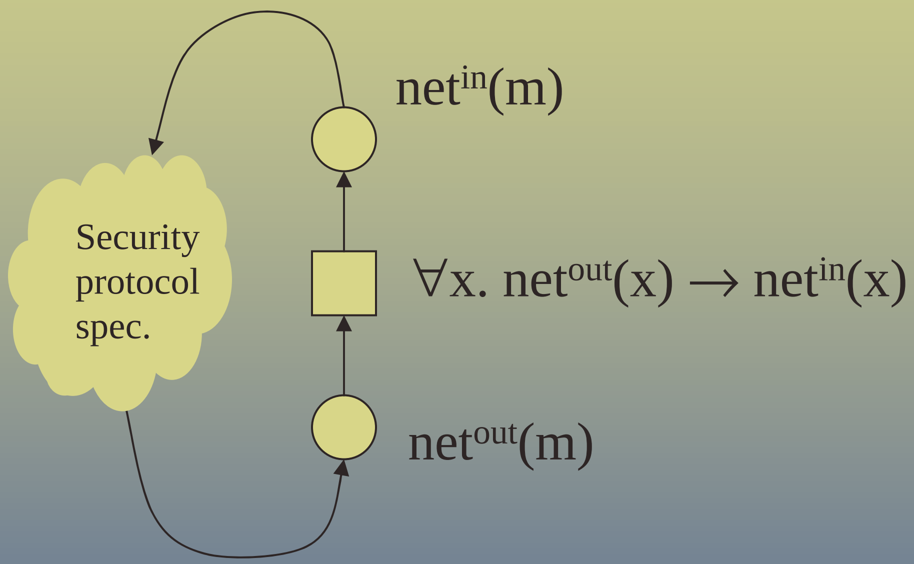
- Canonical objects:
  - η-long, β-normal _LF term
  - Decidable, computable

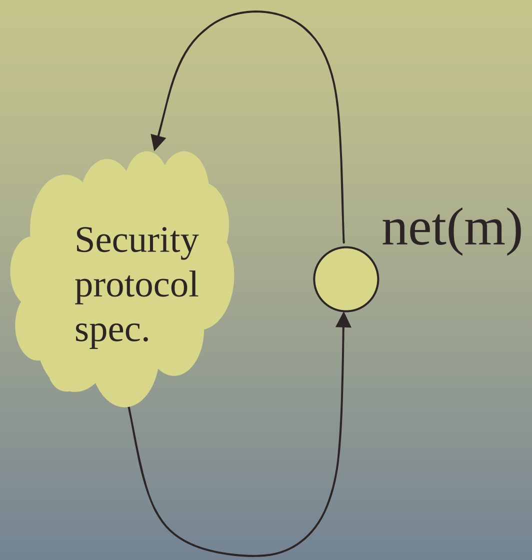
But what is LLF?

- Types (“asynchronous” constructors of ILL)
  - $A ::= a \mid \Pi x{:}A.\ B \mid A \multimap B \mid A \,\&\, B \mid \top$
- Terms
  - $N ::= x \mid \lambda x{:}A.\ N \mid N_1\ N_2$
    $\hat{\lambda} x{:}A.\ N \mid N_1 \hat{}\, N_2 \mid$
    $\langle N_1, N_2 \rangle \mid \text{fst } N \mid \text{snd } N \mid$
    $\langle\rangle$
- Main judgment
  - $\Gamma ; \Delta \vdash N : A$

CLF

An Example





Many instances can be executing concurrently

LLF Encoding

$\mathtt{net} : \mathit{step} \;\; \texttt{o-} \; \mathtt{net}^{\mathtt{out}} \; \mathtt{m}$
$\qquad\qquad\qquad\; \texttt{o-} \; (\mathtt{net}^{\mathtt{in}} \; \mathtt{m} \; \texttt{-o} \; \mathit{step}).$

- LLF forces continuation-passing style
- Consider 2 independent applications:
  - $\lambda \mathbf{n}^{\mathtt{i}}_{1}.\; \mathtt{net} \;\hat{}\; \mathbf{n}^{\mathtt{o}}_{1} \;\hat{}\; (\lambda \mathbf{n}^{\mathtt{i}}_{2}.\; \mathtt{net} \;\hat{}\; \mathbf{n}^{\mathtt{o}}_{2} \;\hat{}\; \mathtt{C})$
  - $\lambda \mathbf{n}^{\mathtt{i}}_{2}.\; \mathtt{net} \;\hat{}\; \mathbf{n}^{\mathtt{o}}_{2} \;\hat{}\; (\lambda \mathbf{n}^{\mathtt{i}}_{1}.\; \mathtt{net} \;\hat{}\; \mathbf{n}^{\mathtt{o}}_{1} \;\hat{}\; \mathtt{C})$

  Should be indistinguishable (***true concurrency***)
- Equate them at the meta-level

  $\mathtt{same\text{-}trace}\; \mathtt{T}_1 \; \mathtt{T}_2 \; \texttt{o-} \; \ldots$

  Never-ending even for small system!

Encoding in Linear logic

$$\forall m.\ net^{out}\ m\ \multimap\ net^{in}\ m$$

- Much simpler
- In general, requires "synchronous" operators
  - $\otimes$ and $\mathbf{1}$
- Concurrency given by "commuting conversions"

$$\begin{aligned} & \text{let } x_1 \otimes y_1 = N_1 \text{ in } (\text{let } x_2 \otimes y_2 = N_2 \text{ in } M) \\ = \ & \text{let } x_2 \otimes y_2 = N_2 \text{ in } (\text{let } x_1 \otimes y_1 = N_1 \text{ in } M) \qquad \text{if } x_i, y_i \notin FV(R_{2-i}) \end{aligned}$$

- … looks like what we want …

However …

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

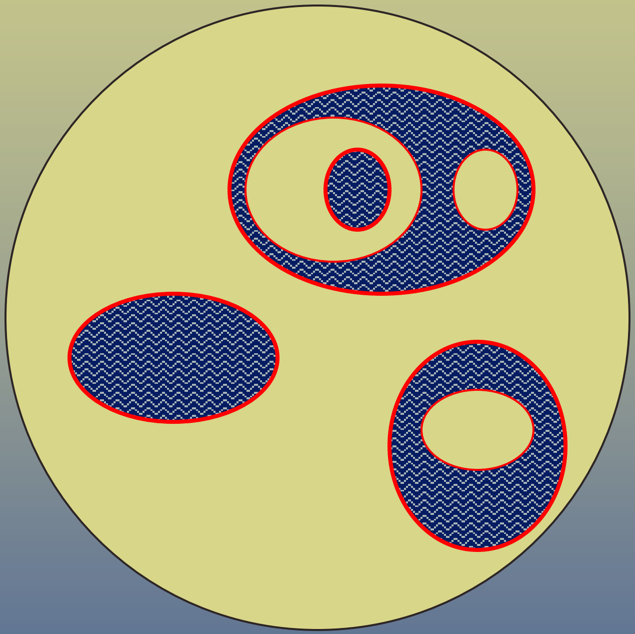
- Synchronous types destroy uniqueness of canonical forms
  - `nat:type.  z:nat.  s:nat->nat.  c:1.`
  - Natural numbers: `z`, `s z`, `s (s z)`, …
  - What about `let 1 = c in z` ? What if `c` is linear?

- No good! ☹

Monadic Encapsulation

Separate synchronous and asynchronous types

- *Outside* the monad
  - LLF types (asynchronous)
  - η-long, β-normal forms
- *Inside* the monad
  - Synchronous types
  - Commuting conversions
    - *Concurrency equation*
  - η-long, β-normal forms
- Monad is a sandbox for synchronous behavior



CLF

- Types
  - $A ::= a \mid \Pi\, x{:}A.\ B \mid A \multimap B \mid A \,\&\, B \mid \top \mid \{S\}$
  - $S ::= A \mid\ !A \mid S_1 \otimes S_2 \mid \mathbf{1} \mid \exists x{:}A.\ S$
- Terms
  - $N ::= x \mid \lambda\, x{:}A.\ N \mid N_1\ N_2 \mid \hat\lambda x{:}A.\ N \mid N_1 \hat{}\, N_2 \mid \langle N_1, N_2 \rangle \mid \mathrm{fst}\ N \mid \mathrm{snd}\ N \mid \langle\rangle \mid \{E\}$
  - $E ::= M \mid \mathrm{let}\ \{p\} = N\ \mathrm{in}\ E$
  - $M ::= N \mid\ !N \mid M_1 \otimes M_2 \mid 1 \mid [N, M]$
  - $p ::= x \mid\ !x \mid p_1 \otimes p_2 \mid 1 \mid [x, p]$

Example in CLF

$$\mathtt{net} : \mathtt{net}^{\mathtt{in}}\ \mathtt{m}\ \mathtt{-o}\ \{\ \mathtt{net}^{\mathtt{out}}\ \mathtt{m}\ \}.$$

- Relating the 2 specifications
  - 2 sets of CLF declarations
  - Meta-level definition of trace transformation

    $$\mathtt{simplify\text{-}net}\ \{\mathtt{T}^{\mathtt{i/o}}\}\ \{\mathtt{T}\}$$

    - 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!*