Are Transition-Based Systems beyond the Reach of Logical Framework?

Iliano Cervesato

ITT Industries @ Naval Research Labs
Washington, DC

iliano@itd.nrl.navy.mil

Contents

- Logical frameworks
- *LLF*
- Transition systems
- Future work

Logical Frameworks

Formalism designed to represent and reason effectively about deductive systems

Formal systems

programming languages, logics, real-time systems, ...

Meta-representation/reasoning

- represent language constructs
- model their semantics
- encode properties and their proofs

Effectiveness

immediacy, executability

Examples

Logics

- *Prolog*
- *λProlog, Isabelle*
- *Forum*

Type theories

- *LF*
- *Coq, Lego*
- *ALF, NuPrl*
- *LLF*

Design

- "Logicity"

  You recognize a logic when you see one!

- Executability

  Proof-checking, possibly proof-search

- Identify and reify fundamental principles of classes of deductive systems
  - Recursive definitions $\rightsquigarrow$ *Horn clauses*
  - Syntactic categories $\rightsquigarrow$ *sorts / types*
  - Binders $\rightsquigarrow$ *Higher-order terms*
  - Hypothetical/parametric rules $\rightsquigarrow$ *Embedded impl./quant.*
  - Derivations as terms $\rightsquigarrow$ *Type theory*
  - State $\rightsquigarrow$ *Linearity*

Unsupported Recurring Constructions

- Negation / extensionality
- Modularity
- Ordering
- True concurrency

LLF

(*joint work with Frank Pfenning*)

- Linear type theory
- Methodology

  linear *linear*

  ∨ ∨

  – Judgment-as-types / derivations-as-terms

  – Higher-order abstract syntax

  ∧

  Linear

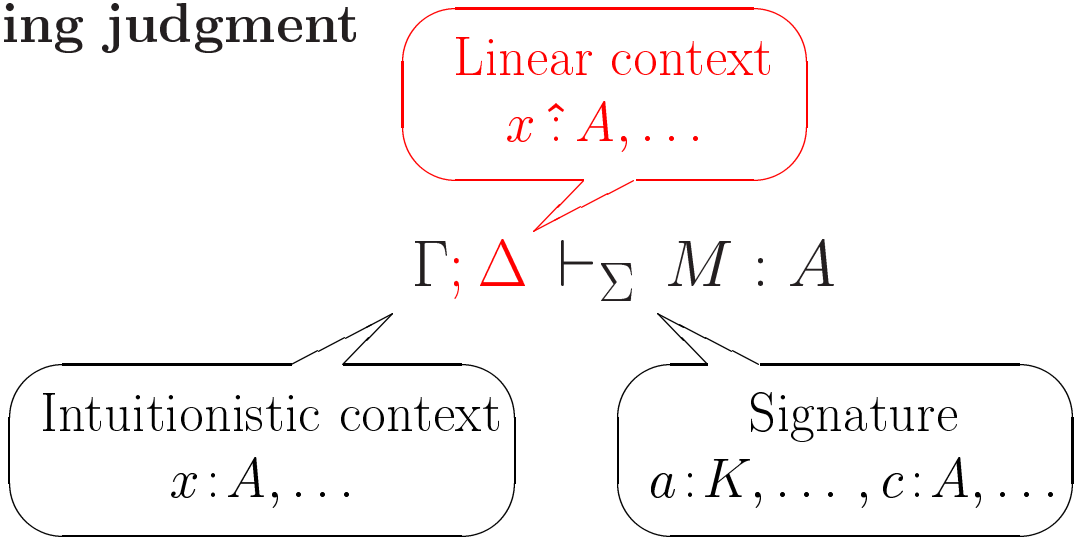
- Supports
  - representation of state-based problems
  - reasoning about state
- Implemented as a higher-order linear logic programming language

Meta-Language

- **Syntax**

$$
\begin{array}{rrcl}
\textit{Kinds} & K & ::= & \mathtt{type} \mid \Pi x\!:\!A.\, K \\
\textit{Type families} & P & ::= & a \mid P\, M \\
\textit{Types} & A & ::= & P \mid \Pi x\!:\!A.\, B \\
 & & & \mid A \multimap B \mid A \,\&\, B \mid \top \\
\textit{Objects} & M & ::= & x \mid c \mid \lambda x\!:\!A.\, M \mid M\, N \\
 & & & \mid \hat{\lambda} x\!:\!A.\, M \mid M \mathbin{\hat{}} N \mid \langle M, N \rangle \mid \textsc{fst}\, M \mid \textsc{snd}\, M \mid \langle\rangle
\end{array}
$$

- **Typing judgment**



"M has type A in Γ, Δ and Σ"

Some Inference Rules

$$\frac{}{\Gamma, x : A; \cdot \vdash_\Sigma x : A}\ \textbf{ivar} \qquad \frac{}{\Gamma; x \mathbin{\hat{:}} A \vdash_\Sigma x : A}\ \textbf{lvar}$$

$$\frac{\Gamma, x : A; \Delta \vdash_\Sigma M : B}{\Gamma; \Delta \vdash_\Sigma \lambda x : A.\, M : \Pi x : A.\, B}\ \textbf{ilam} \qquad \frac{\Gamma; \Delta \vdash_\Sigma M : \Pi x : A.\, B \quad \Gamma; \cdot \vdash_\Sigma N : A}{\Gamma; \Delta \vdash_\Sigma M\ N : [N/x]B}\ \textbf{iapp}$$

$$\frac{\Gamma; \Delta, x \mathbin{\hat{:}} A \vdash_\Sigma M : B}{\Gamma; \Delta \vdash_\Sigma \hat{\lambda} x : A.\, M : A \multimap B}\ \textbf{llam} \qquad \frac{\Gamma; \Delta_1 \vdash_\Sigma M : A \multimap B \quad \Gamma; \Delta_2 \vdash_\Sigma N : A}{\Gamma; \Delta_1, \Delta_2 \vdash_\Sigma M \mathbin{\hat{}} N : B}\ \textbf{iapp}$$

$$\frac{\Gamma; \Delta \vdash_\Sigma M : A \quad \Gamma; \Delta \vdash_\Sigma N : B}{\Gamma; \Delta \vdash_\Sigma \langle M, N \rangle : A \,\&\, B}\ \textbf{pair} \qquad \frac{\Gamma; \Delta \vdash_\Sigma M : A \,\&\, B}{\Gamma; \Delta \vdash_\Sigma \mathrm{FST}\ M : A}\ \textbf{fst} \qquad \frac{\Gamma; \Delta \vdash_\Sigma M : A \,\&\, B}{\Gamma; \Delta \vdash_\Sigma \mathrm{SND}\ M : B}\ \textbf{snd}$$

$$\frac{}{\Gamma; \Delta \vdash_\Sigma \langle\rangle : \top}\ \textbf{unit}$$

Main Properties

- Decidable type checking

  ↪ Automated support

- Unique canonical forms

  ↪ Easy proofs of adequacy

  ↪ Logic programming

- Derivations represented by terms

  ↪ Meta-reasoning

  ↪ Program transformation

- Conservative over LF

  ↪ Inherits work done on LF

Applications

Reasoning

- Imperative programming languages
- Substructural logics
- Security protocols

Specification / Simulation

- Hardware architectures
- Cryptoprotocols
- Real-time systems
- Planning
- Games

+ *LF* achievements

- Functional languages, logic programming languages
- Logics
- Category theory, ...

Implementation

Computer-aided specification

- Type-checking
- Type reconstruction
- Innovations: spine calculus, dependent explicit substitutions, linear explicit substitutions

Execution

- Higher-order linear constraint logic programming language
- Innovations: higher-order unification, context-management, compilation

Example: *MLR*

MLR is a fragment of *ML* with

- references
- value polymorphism
- recursion

$$\textit{Types}\ \ \tau ::= \ldots \mid \mathbf{1} \mid \tau_1 \to \tau_2 \mid \tau\ \mathbf{ref}$$

$$\begin{aligned}\textit{Expressions}\ \ e ::= {} & x \\ \mid {} & \langle\rangle \\ \mid {} & \mathbf{lam}\, x.\, e \\ \mid {} & e_1\, e_2 \\ \mid {} & \ldots \\ \mid {} & c \\ \mid {} & \mathbf{ref}\ e \\ \mid {} & \mathbf{!}e \\ \mid {} & e_1 := e_2\end{aligned}$$

$$\textit{Store}\ \ S ::= \cdot \mid S, c = v$$

Expressions

```
exp   : type.
cell  : type.

unit   : exp.
lam    : (exp -> exp) -> exp.
app    : exp -> exp -> exp.
       ...
loc    : cell -> exp.
ref    : exp -> exp.
deref  : exp -> exp.
assign : exp -> exp -> exp.
```

MLR: Typing

$\Omega \vdash e : \tau$ "e has type τ in Ω"

Context $x_i : \tau_i, \ldots, c_j : \sigma_j, \ldots$ — Expression — Type

Representation: $\ulcorner\Omega\urcorner \vdash_\Sigma \ulcorner\mathcal{T}\urcorner : \texttt{exp\_type}\ \ulcorner e\urcorner\ \ulcorner\tau\urcorner$

$x_i : \texttt{exp}, \quad t_i : \texttt{exp\_type}\ x_i\ \ulcorner\tau_i\urcorner, \quad \ldots$
$c_j : \texttt{cell}, \ l_j : \texttt{cell\_type}\ c_j\ \ulcorner\sigma_j\urcorner, \ \ldots$

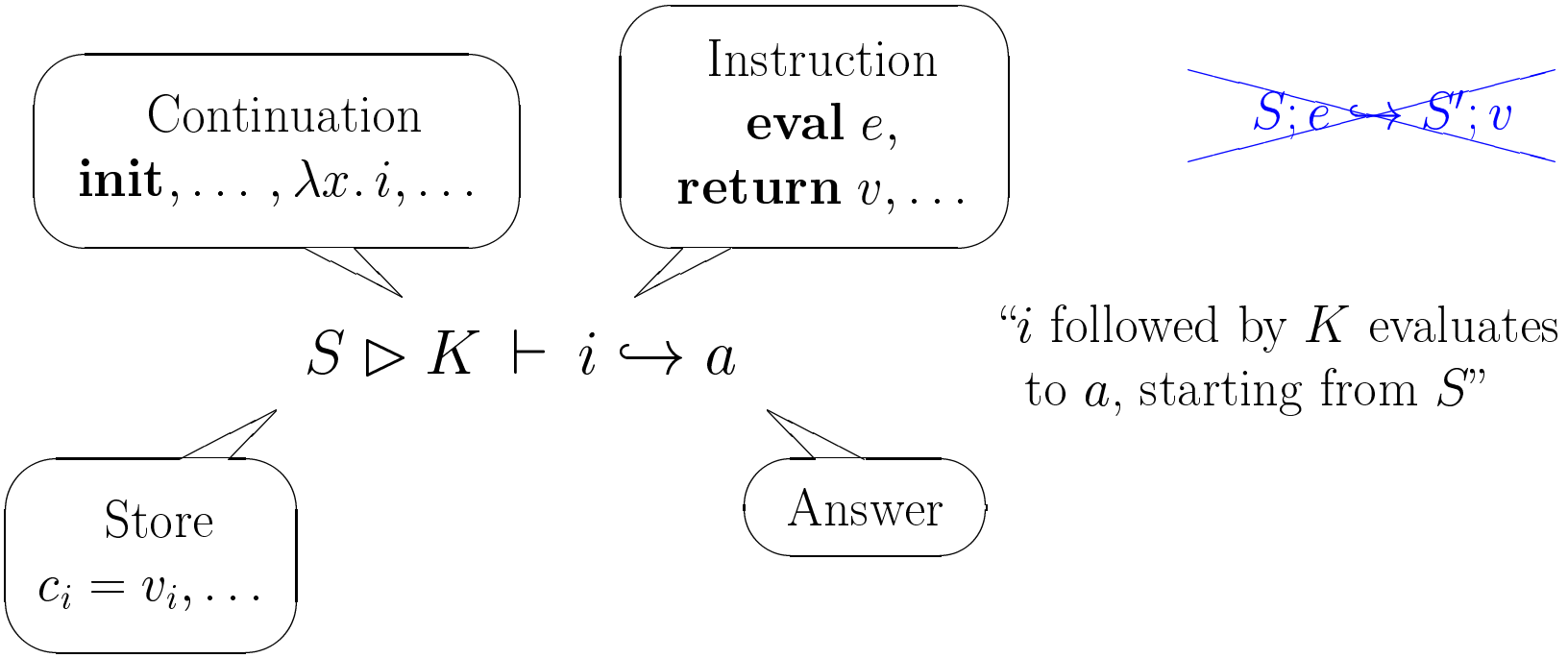
$$\frac{\Omega \vdash e_1 : \tau\ \mathbf{ref} \quad \Omega \vdash e_2 : \tau}{\Omega \vdash e_1 := e_2 : \mathbf{1}}\ \mathbf{et\_assign}$$

```
et_assign :  exp_type E1 (rf T)
          -> exp_type E2 T
          -> exp_type (assign E1 E2) 1.
```

$$\frac{\Omega \vdash e : \tau\ \mathbf{ref}}{\Omega \vdash\ !e : \tau}\ \mathbf{et\_deref}$$

```
et_deref  :  exp_type E (rf T)
          -> exp_type (deref E) T.
```

MLR: Evaluation



Representation: $\ulcorner S \urcorner \vdash_\Sigma \ulcorner \mathcal{E} \urcorner : \texttt{eval}\ \ulcorner K \urcorner\ \ulcorner i \urcorner\ \ulcorner a \urcorner$

$c_i : \texttt{cell},\ h_i \mathbin{\hat{:}} \texttt{contains}\ c_i\ \ulcorner v_i \urcorner,\ \ldots$

MLR: Some Imperative Rules

$$\frac{S', c = v, S'' \rhd K \vdash \mathbf{return}\ \langle\rangle \hookrightarrow a}{S', c = v', S'' \rhd K \vdash c := v \hookrightarrow a}\ \mathbf{ev\_assign}$$

```
ev_assign :    (contains C V   -o  eval K (return unit) A)
            -o (contains C V'  -o  eval K (assign2 (loc C) V) A).
```

$$\frac{S', c = v, S'' \rhd K \vdash \mathbf{return}\ v \hookrightarrow a}{S', c = v, S'' \rhd K \vdash\ !c \hookrightarrow a}\ \mathbf{ev\_deref}$$

```
ev_deref  :  read C V
             & eval K (return V) A
            -o eval K (ref1 (loc C)) A.

rd :  contains C V
   -o <T>
   -o read C V.
```

Adequacy theorem (*Evaluation*)

Given a store $S = (c_1 = v_1, \ldots, c_n = v_n)$, a continuation K, an instruction i and an answer a, all closed, there is a compositional bijection between derivations $\mathcal{E}$ of

$$S \rhd K \vdash i \hookrightarrow a$$

and canonical LLF objects M such that

$$\ulcorner S \urcorner \vdash_\Sigma M : \texttt{eval}\ \ulcorner i \urcorner\ \ulcorner a \urcorner$$

is derivable, where

$$\ulcorner S \urcorner = \begin{bmatrix} c_1 : \texttt{cell},\ h_1 \hat{:} \texttt{contains}\ c_1\ \ulcorner v_1 \urcorner \\ \ldots \\ c_n : \texttt{cell},\ h_n \hat{:} \texttt{contains}\ c_n\ \ulcorner v_n \urcorner \end{bmatrix}$$

MLR: Type Preservation

- Functional core: implemented in *LF*
- References: implemented in *LLF*

Theorem (*type preservation*)

If $S \rhd K \vdash i \hookrightarrow a$, with $\Omega \vdash i : \tau$, $\Omega \vdash K : \tau \Rightarrow \sigma$ and $\Omega \vdash S : \Omega$, then $\Omega \vdash a : \sigma$

Proof: by induction on the evaluation derivation

The high level of abstraction of the representation permits **transcribing** this proof into an *LLF* specification capturing its computational contents

- each case yields one declaration
- the meta-reasoning is itself *linear*

Representation

```
tpev : eval K I A -> cont_type K T S -> instr_type I T -> ans_type A S -> type.
```

Evaluation of the Encoding

- Adequacy
  ↪ good, if Ok with continuation semantics
- Execution
  ↪ fully executable
- Type-checking
  ↪ as usual
- Meta-reasoning
  ↪ elegant

Multiset Rewriting for Crypto-protocol Specification

Multiset $\mathbf{X} = X_1, \ldots, X_n$

Multiset rewrite rule $\mathbf{X} \longrightarrow \mathbf{Y}$

Computation $\mathbf{X}, \mathbf{Z} \xrightarrow{\mathbf{X} \longrightarrow \mathbf{Y}}_{\mathcal{R}} \mathbf{Y}, \mathbf{Z}$ + refl. + trans.

Parametric multisets $X_i(\vec{t})$

$\hookrightarrow$ computation relies on unification

Generative multiset rule $\mathbf{X}(\vec{t}) \longrightarrow \exists \vec{x}.\, \mathbf{Y}(\vec{t}, \vec{x})$

Message Exchange

$$A \longrightarrow B : M$$

- Local state transitions
- Interaction with the network

$$A_i(\vec{a}), \ldots \longrightarrow A_{i'}(\vec{a}), N^+(M)$$
$$B_j(\vec{b}), N^-(M) \longrightarrow B_{j'}(\vec{b}), \ldots$$

Syntax and Environment

Nonces $A_i \longrightarrow \exists n.\, A_j, N(\ldots n \ldots)$

↪ not completely realistic

Cryptography

↪ constructor: $\{M\}_k$, $< M_1, M_2 >$

↪ destructor: pattern matching

↪ unrealistic but often acceptable

Network $N^+(M) \longrightarrow N^-(M)$

Intruder Dolev-Yao model

Example

Needham-Schroeder key exchange (simplified)

$$A \longrightarrow B : \{<n_a, A>\}_{k_b}$$
$$B \longrightarrow A : \{<n_a, n_b, B>\}_{k_a}$$
$$A \longrightarrow B : \{n_b\}_{k_b}$$

$$\begin{aligned}
\cdot &\longrightarrow A_0 \\
A_0 &\longrightarrow \exists n_a.\, N^+(\{<n_a, A>\}_{k_b}), A_1(B, n_a) \\
A_1(B, n_a), N^-(\{<n_a, n, B>\}_{k_a}) &\longrightarrow N^+(\{n\}_{k_b}), A_2(B, n_a, n)
\end{aligned}$$

$$\begin{aligned}
\cdot &\longrightarrow B_0 \\
B_0, N^-(\{<n, A>\}_{k_b}) &\longrightarrow \exists n_b.\, N^+(\{<n, n_b, B>\}_{k_a}), B_1(A, n, n_b) \\
B_1(A, n, n_b), N^-(\{n_b\}_{k_b}) &\longrightarrow B_2(A, n, n_b)
\end{aligned}$$

Formalization in Linear Logic

Generative multiset rewriting is linear logic in disguise

$$\mathbf{X}(\vec{x}) \longrightarrow \exists \vec{y}.\, \mathbf{Y}(\vec{x}, \vec{y})$$

$$\Updownarrow$$

$$\forall \vec{x}.\, \bigotimes \mathbf{X}(\vec{x}) \multimap \exists \vec{y}.\, \bigotimes \mathbf{Y}(\vec{x}, \vec{y})$$

The translation preserves the semantics

Coding in *LLF*

No $\otimes$, no $\exists$!?

$$\forall\vec{x}.\ \ X_1(\vec{x}) \ \otimes\ \ldots\ \otimes\ X_m(\vec{x}) \ \multimap\ \exists\vec{y}.\ \ Y_1(\vec{x},\vec{y}) \ \otimes\ \ldots\ \otimes\ Y_m(\vec{x},\vec{y})$$

$$\Downarrow$$

$$\begin{aligned}
\forall\vec{x}.\,\mathrm{loop} \ \circ\!\!- \ & X_1(\vec{x}) \\
& \ldots \\
\circ\!\!- \ & X_n(\vec{x}) \\
\circ\!\!- \ & \forall\vec{y}.\,(Y_1(\vec{x},\vec{y}) \ \multimap \\
& \qquad \ldots \\
& \qquad Y_m(\vec{x},\vec{y}) \ \multimap\ \mathrm{loop})
\end{aligned}$$

Adequacy

Theorem

For every transition sequence $\vec{r}$ such that

$$\mathbf{X} \xrightarrow{\vec{r}}_{\mathcal{R}} \mathbf{Z}$$

there is an *LLF* proof-term M such that

$$\ulcorner\mathcal{R}\urcorner;\ulcorner\mathbf{X}\urcorner,\ulcorner\mathbf{Y}\urcorner \multimap \mathtt{loop} \vdash_{\Sigma} M : \mathtt{loop}$$

is derivable, and viceversa.

No Net

Theorem

For every run $\mathcal{R}$ there is a run $\mathcal{R}'$ that

- does not use the network rule
- exchanges the same messages in the same order
- has the same or bigger intruder knowledge

Proof: Replace network uses with interception + resend by the intruder □

- This proof can been represented in *LLF*
- It is executable and implements the transformation
- Same technique has been applied to more involved problems

Evaluation of the Encoding

- Adequacy
  ↪ indirect

- Simulation
  ↪ trivial, but unfocused

- Attack detection (*model checking*)
  ↪ at best inefficient

- Reasoning
  ↪ feasible but painful

What's wrong?

Continuation-based model of LLF?

↪ No, but makes things worse

Sequential specification!

↪ same difficulties in multiset rewriting

↪ same difficulties in linear logic

Alternatives

- Strands
- Colored Petri nets

Recurrent enough?

State Transition Systems

- Concurrency
- Programming languages
- Cryptoprotocols
- Real-time system
- Planning
- Automata
- Database transactions
- ...

Future Directions

Logical frameworks that can handle transition systems effectively

- Study existing formalism
  - colored Petri nets
  - process calculi
  - proof nets
  - ...

- Logics whose derivations are acyclic graphs