Reasoning about State in a Linear Logical Framework

Iliano Cervesato

Department of Computer Science
Stanford University

Contents

- Logical frameworks
- *LLF*
- Case study
- Conclusions

Logical Frameworks

A **Logical Framework** is a formalism designed to represent and reason about deductive systems

formal system

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

meta-representation

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

effectiveness

immediacy and executability

Examples

Logics

- *Prolog*
- *λProlog* [Miller,Nadathur'88], *Isabelle* [Paulson'93]
- *Forum* [Miller'94]

Type theories

- *LF* [Harper,Honsell,Plotkin'93]
- *Coq* [Dowek&al'93], *Lego* [Pollack'94]
- *ALF* [Nordström'93], *NuPrl* [Constable&al'86]
- *LLF* [Cervesato,Pfenning'96]

Functionalities

- Specification

  Formalize (abstract) syntax, operational semantics, and meta-theory

- Analysis

  Support proof-checking, often theorem-proving

- Experimentation

  Permit (limited) execution

Identify and reify fundamental principles of classes of deductive systems

Applications

(LF biased) [Harper&al.'93]

- **Past**
  - Formalization of declarative programming languages and simple logics
  - Representation of simple properties
- **Present**
  - **State** [Cervesato,Pfenning'96; Cervesato&al.'99]
  - Program verification and certification [Necula'97; Paulson'96]
- **Future**

  Assisted design of new and better logics, programming languages, ...
  - Meta-theorem provers [Schürmann,Pfenning'98]
  - Other recurring notions [Polakow,Pfenning'99]

LLF, a Logical Framework for State

- Design
  - Extend a logical framework with linear logic constructs
  - Extend linear logic to reason about state
- Implementation
  - Automated support for *LLF* specifications
  - Higher-order linear logic programming language
- Applications
  - Reasoning about state
  - Specification of state-based problems
  - *Everything that could be done in LF*

Linear Logic in Brief

$$\Gamma; \Delta \vdash A$$

Logical assumptions

Resources

Goal

Accessing a resource **consumes** it

Main resource operators

- $A \otimes B$ = "A and B simultaneously"
- $A \,\&\, B$ = "A and B alternatively"
- $\top$ = "resource sink"
- $A \multimap B$ = "B assuming A as a resource"
- $A \rightarrow B$ = "B assuming A as a logical hypothesis"

Meta-Language

- **Syntax**

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

- **Typing judgment**

Linear context
$x \mathbin{\hat{:}} A, \ldots$

$$\Gamma; \Delta \vdash_\Sigma M : A$$

Intuitionistic context
$x : A, \ldots$

Signature
$a : K, \ldots, c : A, \ldots$

"M has type A in Γ, Δ and Σ"

Main Properties

- Decidable type checking

  $\hookrightarrow$ Automated support

- Unique canonical forms

  $\hookrightarrow$ Easy proofs of adequacy

  $\hookrightarrow$ Logic programming

- Derivations represented by terms

  $\hookrightarrow$ Meta-reasoning

  $\hookrightarrow$ Program transformation

- Conservative over *LF* [Harper&al.'93]

  $\hookrightarrow$ Inherits work done on *LF*

Applications

Reasoning

- Imperative programming languages
- Substructural logics
- Security protocols

Specification / Simulation

- Hardware architectures
- 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

Execution

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

Forthcoming ...

- Meta-theorem prover
- Innovations: reasoning about *LLF* specs, linear explicit substitutions

Limitations

With state

- Indirect representation of transition systems
- Resource modularity

Beyond state

- Extensionality (negation, extensional quantification)
- Ordering (priority, stacks, ...)

Multiset Rewriting

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

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

Computation $\dddot{X}, \dddot{Z} \xrightarrow{\dddot{X} \longrightarrow \dddot{Y}} \dddot{Y}, \dddot{Z}$

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

$\hookrightarrow$ computation relies on unification

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

Message Exchange

$$A \longrightarrow B : M$$

- Local state transitions
- Interaction with the network

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

Brand-New Nonces

- Use counter

$$A_i, \mathrm{currNonce}(n) \longrightarrow A_j, \mathrm{currNonce}(n+1), N^+(\ldots n \ldots)$$

$\hookrightarrow$ simplicistic

$\hookrightarrow$ complicates reasoning

- Use abstraction

$$A_i \longrightarrow \looparrowright n.\, A_j, N^+(\ldots n \ldots)$$

$\hookrightarrow$ not completely realistic

Cryptography

- Transcribe the encryption/decryption algorithms
  ↪ painful (but feasible)
  ↪ complicates reasoning about protocol issues
  ↪ does not allow reasoning about cryptographic issues

- Use abstraction
  ↪ constructor: $\{M\}_k$
  ↪ destructor: pattern matching
  ↪ unrealistic but often acceptable

Network

$$N^{+}(M) \;\longrightarrow\; N^{-}(M)$$

Intruder

- Dolev-Yao model

$$
\begin{array}{lrcl}
- & N^+(M) & \longrightarrow & I(M) \\
- & I(M) & \longrightarrow & N^-(M), I(M) \\
\\
- & I(< M, N >) & \longrightarrow & I(M), I(N) \\
- & I(M), I(N) & \longrightarrow & I(< M, N >), I(M), I(N) \\
\\
- & I(\{M\}_k), I(k) & \longrightarrow & I(M), I(k) \\
- & I(M), I(k) & \longrightarrow & I(\{M\}_k), I(M), I(k) \\
\\
- & & \longrightarrow & \looparrowright n.\, I(n)
\end{array}
$$

- More powerful models are possible

Example

Needham-Schroeder key exchange (simplified)

$$
\begin{aligned}
&A \longrightarrow B : \{<n_a, A>\}_{k_b} \\
&B \longrightarrow A : \{<n_a, n_b, B>\}_{k_a} \\
&A \longrightarrow B : \{n_b\}_{k_b} \\
&\ldots
\end{aligned}
$$

$$
\begin{aligned}
A_0 &\longrightarrow \looparrowright n_a.\, N^+(\{<n_a, A>\}_{k_b}), A_1(B, n_a) \\
B_0, N^-(\{<n, A>\}_{k_b}) &\longrightarrow \looparrowright n_b.\, N^+(\{<n, n_b, B>\}_{k_a}), B_1(A, n, n_b) \\
A_1(B, n_a), N^-(\{<n_a, n, B>\}_{k_a}) &\longrightarrow N^+(\{n\}_{k_b}), A_2(B, n_a, n) \\
B_1(A, n, n_b), N^-(\{n_b\}_{k_b}) &\longrightarrow B_2(A, n, n_b), \ldots
\end{aligned}
$$

Linear Logic Strikes Back

Generative multiset rewriting is linear logic undercover

$$\dddot{X}(\vec{x}) \longrightarrow \looparrowright \vec{y}.\, \dddot{Y}(\vec{x}, \vec{y})$$

$$\rightsquigarrow$$

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

The translation preserves the semantics

Coding in *LLF*

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

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

$$\rightsquigarrow$$

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

Needham-Schroeder

```
nsA1 : loop
        o- annKey B
        o- a0
        o- ({Na:atm}
                 a1 B (@ Na)
              -o toNet (crypt ((@ Na) * (@ (k2m A))) B)
              -o loop).

nsB1 : loop
        o- b0
        o- fromNet (crypt (X * (@ (k2m A))) B)
        o- annKey A
        o- (     {Nb:atm} b1 A X (@ Nb)
              -o toNet (crypt (X * (@ Nb) * (@ (k2m B)) A)
              -o loop).
```

Needham-Schroeder

```
nsA2 : loop
        o- a1 B X
        o- fromNet (crypt (X * Y * (@ (k2m B)) A)
        o- (     toNet (crypt Y B)
              -o a2 B X Y
              -o loop).

nsB2 : loop
        o- b1 A X Y
        o- fromNet (crypt Y B)
        o- (b2 A X Y -o loop).
```

Uses of *LLF*

- Simulation

  $\hookrightarrow$ trivial

- Attack detection

  $\hookrightarrow$ tricky

- Reasoning $\Longleftarrow$

  $\hookrightarrow$ feasible

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 $\square$

Yields huge savings during protocol analysis

LLF Formalization

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

Summary

***LLF*,**

- combines the meta-reasoning power of logical frameworks with the ability of handling state of linear logic
- conservative extension of the logical framework *LF*
- implemented as a linear logic programming language
- used for the representation of
  - imperative programming languages
  - substructural and modal logics
  - state transition systems, ...

Future Directions

- Experimentation with *LLF*: more state-based systems, new limitations
- Complete *LLF*: efficiency, environment
- Meta-theorem prover: get help proving things
- Beyond *LLF*: direct support for transition systems, modularity, negation, ...

An Example: *LF* (Meta-Language)

Typing judgment

$$\Gamma \vdash_{\Sigma} M : A$$

"M has type A in Γ and Σ"

Context
$x\!:\!A, \ldots$

Signature
$a\!:\!K, \ldots, c\!:\!A, \ldots$

An Example: *LF* (Representation Methodology—Cont'd)

$x_i : \tau_i, \ldots$

$$\ulcorner \frac{\mathcal{T}}{\Omega \vdash e : \tau} \urcorner \quad = \quad M$$

$$\ulcorner \Omega \urcorner \vdash_\Sigma M : \texttt{has\_type} \ulcorner e \urcorner \ulcorner \tau \urcorner$$

where for each $x_i : \tau_i$ in Ω,

$$\ulcorner x_i : \tau_i \urcorner \;=\; x_i : \texttt{exp},\; t_i : \texttt{has\_type}\, x_i \ulcorner \tau_i \urcorner$$

- context operations reduce to meta-level primitives
- meta-theoretic properties are inherited from the meta-language

Problem!

$$\ulcorner \begin{array}{c} \mathcal{E} \\ S \rhd K \vdash e \hookrightarrow a \end{array} \urcorner = M$$

$c_i = v_i, \ldots$

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

This does not work!

- S is subject to *destructive operations* (e.g. assignment)
- traditional log. frameworks do not allow removing assumptions from the context

A way out ...

$$\cdot \vdash_\Sigma M : \texttt{eval} \ulcorner S \urcorner \ulcorner K \urcorner \ulcorner e \urcorner \ulcorner a \urcorner$$

... but, we must encode *explicitly*

- context operations (lookup, insertion, ...)
- context-related properties (weakening, exchange, ...)