Reasoning about State
in a
Linear Logical Framework

Iliano Cervesato

Department of Computer Science
Stanford University
Contents

- Logical frameworks
- \( LLF \)
- Case study
- Conclusions
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*
- $\lambda$*Prolog* [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

\[ \Gamma; \Delta \vdash A \]

Accessing a resource consumes it

Main resource operators

- \( A \otimes B = \text{“A and B simultaneously”} \)
- \( A \& B = \text{“A and B alternatively”} \)
- \( \top = \text{“resource sink”} \)
- \( A \circ B = \text{“B assuming A as a resource”} \)
- \( A \rightarrow B = \text{“B assuming A as a logical hypothesis”} \)
Meta-Language

- Syntax

Kinds \[ K ::= \text{type} | \Pi x: A. K \]

Type families \[ P ::= a | P M \]

Types \[ A ::= P | \Pi x: A. B | A \rightarrow B | A \& B | \top \]

Objects \[ M ::= x | c | \lambda x: A. M | M N | \hat{\lambda} x: A. M | M^N | \langle M, N \rangle | \text{FST} \ M | \text{SND} \ M | \langle \rangle \]

- Typing judgment

\[ \Gamma; \Delta \vdash \Sigma \ M : A \]

“\( M \) has type \( A \)

in \( \Gamma, \Delta \) and \( \Sigma \)”

Intuitionistic context\[ x: A, \ldots \]

Signature \[ a: K, \ldots, c: A, \ldots \]
Main Properties

• Decidable type checking
  \[\rightarrow\text{Automated support}\]

• Unique canonical forms
  \[\rightarrow\text{Easy proofs of adequacy}\]
  \[\rightarrow\text{Logic programming}\]

• Derivations represented by terms
  \[\rightarrow\text{Meta-reasoning}\]
  \[\rightarrow\text{Program transformation}\]

• Conservative over \(LF\) [Harper\&al.'93]
  \[\rightarrow\text{ Inherits work done on } LF\]
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, ...
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, ...)

Iliano Cervesato — *Reasoning about State in a Linear Logical Framework*
Multiset Rewriting

Multiset \( \bar{X} = X_1, \ldots, X_n \)

Multiset rewrite rule \( \bar{X} \rightarrow \bar{Y} \)

Computation \( \bar{X}, \bar{Z} \xrightarrow{\bar{x} \rightarrow \bar{y}} \bar{Y}, \bar{Z} \)

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

\( \rightarrow \) computation relies on unification

Generative multiset rule \( \bar{X}(\vec{t}) \rightarrow \forall \vec{x}. \bar{Y}(\vec{t}, \vec{x}) \)
$A \rightarrow B : M$

- Local state transitions
- Interaction with the network

$A_i(\vec{a}), \ldots \rightarrow A_{i'}(\vec{a}), N^+(M)$

$B_j(\vec{b}), N^-(M) \rightarrow B_{j'}(\vec{b}), \ldots$
Brand-New Nonces

- Use counter

\[ A_i, \text{currNonce}(n) \rightarrow A_j, \text{currNonce}(n+1), N^+ (\ldots n \ldots) \]

\[ \leftrightarrow \text{simplicistic} \]

\[ \leftrightarrow \text{complicates reasoning} \]

- Use abstraction

\[ A_i \rightarrow \forall n. A_j, N^+ (\ldots n \ldots) \]

\[ \leftrightarrow \text{not completely realistic} \]
• 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

\[ N^+(M) \longrightarrow N^-(M) \]
**Dolev-Yao model**

- \[ N^+(M) \rightarrow I(M) \]
- \[ I(M) \rightarrow N^-(M), I(M) \]

- \[ I(<M,N>) \rightarrow I(M), I(N) \]
- \[ I(M), I(N) \rightarrow I(<M,N>), I(M), I(N) \]

- \[ I(\{M\}_k), I(k) \rightarrow I(M), I(k) \]
- \[ I(M), I(k) \rightarrow I(\{M\}_k), I(M), I(k) \]

- \[ \rightarrow \ q\rightarrow n. I(n) \]

**More powerful models are possible**
Example

Needham-Schroeder key exchange (simplified)

\[ A \rightarrow B : \{<n_a, A>\}_{k_b} \]

\[ B \rightarrow A : \{<n_a, n_b, B>\}_{k_a} \]

\[ A \rightarrow B : \{n_b\}_{k_b} \]

...?

\[ A_0 \rightarrow \leftrightarrow n_a \cdot N^+(\{<n_a, A>\}_{k_b}), A_1(B, n_a) \]

\[ B_0, N^-(\{<n, A>\}_{k_b}) \rightarrow \leftrightarrow n_b \cdot N^+(\{<n, n_b, B>\}_{k_a}), B_1(A, n, n_b) \]

\[ A_1(B, n_a), N^-(\{<n_a, n, B>\}_{k_a}) \rightarrow N^+(\{n\}_{k_b}), A_2(B, n_a, n) \]

\[ B_1(A, n, n_b), N^-(\{n_b\}_{k_b}) \rightarrow B_2(A, n, n_b), ... \]
Generative multiset rewriting is linear logic undercover

\[ \ddot{X}(\vec{x}) \rightarrow \forall y. \dot{Y}(\vec{x}, \vec{y}) \]

\[ \Downarrow \]

\[ \forall \vec{x}. \bigotimes \ddot{X}(\vec{x}) \rightarrow \exists \vec{y}. \bigotimes \dot{Y}(\vec{x}, \vec{y}) \]

The translation preserves the semantics
Coding in LLF

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

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

$\downarrow$

$$\forall \vec{x}. \text{loop} \implies X_1(\vec{x})$$

$$\ldots$$

$$\implies X_n(\vec{x})$$

$$\implies \forall \vec{y}. \; (Y_1(\vec{x}, \vec{y}) \implies \ldots$$

$$\implies Y_m(\vec{x}, \vec{y}) \implies \text{loop})$$
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**
  $\rightarrow$ trivial

- **Attack detection**
  $\rightarrow$ tricky

- **Reasoning**
  $\leftarrow$ feasible
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

Yields huge savings during protocol analysis
• 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

\[ \Gamma \vdash \Sigma \quad M : A \]

"\(M\) has type \(A\) in \(\Gamma\) and \(\Sigma\)"

**Context**

\[ x : A, \ldots \]

**Signature**

\[ a : K, \ldots, c : A, \ldots \]
An Example: \emph{LF} (Representation Methodology—Cont’d)

\[
\begin{array}{c}
\vdash \mathcal{T} \\
\Omega \vdash e : \tau = M
\end{array}
\]

\[
\Gamma \vdash \Sigma \ M : \text{has\_type} \Gamma e \Gamma \Gamma \tau \Gamma
\]

where for each \( x_i : \tau_i \) in \( \Omega \),

\[
\Gamma x_i : \tau_i = x_i : \text{exp}, \ t_i : \text{has\_type} x_i \Gamma \tau_i \Gamma
\]

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

\[ c_i = v_i, \ldots \quad \because \quad \mathcal{E} \quad \therefore \quad S \triangleright K \vdash e \leftrightarrow a = M \]

\[ \neg S \vdash \Sigma \quad M : \mathsf{eval} \neg K \neg e \neg a \]

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 ...

\[ \neg \Sigma \quad M : \mathsf{eval} \neg S \neg K \neg e \neg a \]

... but, we must encode explicitly

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