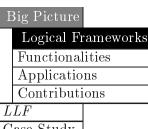
# Reasoning about State in a Linear Logical Framework

Iliano Cervesato

Department of Computer Science Stanford University

## **Contents**

- Logical frameworks
- $\bullet$  LLF
- Case study
- Conclusions



## **Logical Frameworks**

Case Study Conclusions

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

#### formal system

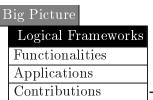
programming languages, logics, real-time systems, ...

#### meta-representation

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

#### effectiveness

immediacy and executability



# **Examples**

*LLF*Case Study

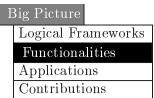
Conclusions

## 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**

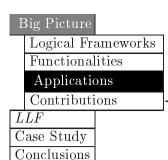
LLF
Case Study
Conclusions

- 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

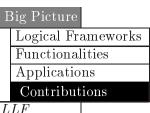
#### • Present

- 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

 $\frac{LLF}{\text{Case Study}}$ Conclusions

#### • Design

- Extend a logical framework with linear logic constructs
- Extend linear logic to reason about state

#### • Implementation

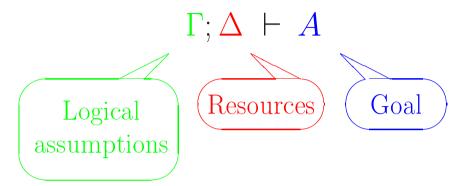
- Automated support for *LLF* specifications
- Higher-order linear logic programming language

## • Applications

- Reasoning about state
- Specification of state-based problems
- Everything that could be done in LF



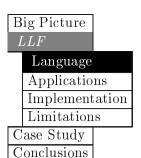
# Linear Logic in Brief



Accessing a resource consumes it

#### Main resource operators

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

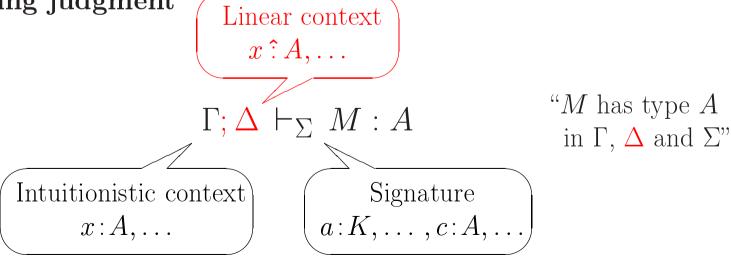


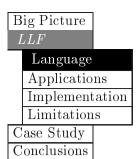
# Meta-Language

• Syntax

$$Kinds \qquad K ::= \mathsf{type} \mid \Pi x : A. \, K$$
 
$$Type \ families \qquad P ::= a \mid P \, M$$
 
$$Types \qquad A ::= P \mid \Pi x : A. \, B \\ \mid A \multimap B \mid A \& B \mid \top$$
 
$$Objects \qquad M ::= x \mid c \mid \lambda x : A. \, M \mid M \, N \\ \mid \hat{\lambda} x : A. \, M \mid M \, \hat{N} \mid \langle M, N \rangle \mid \mathsf{FST} \, M \mid \mathsf{SND} \, M \mid \langle \rangle$$

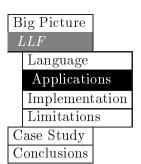
• Typing judgment





## **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

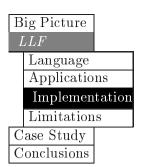
- Imperative programming languages
- Substructural logics
- Security protocols

#### Specification / Simulation

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

#### + LF achievements

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



## **Implementation**

#### Computer-aided specification

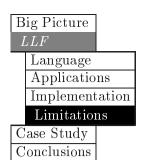
- Type-checking
- Type reconstruction
- Innovations: spine calculus, dependent explicit substitutions

#### 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



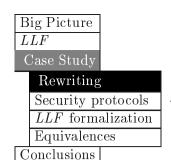
## Limitations

#### With state

- Indirect representation of transition systems
- Resource modularity

#### Beyond state

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



## Multiset Rewriting

Multiset

$$\ddot{X} = X_1, \ldots, X_n$$

Multiset rewrite rule

$$\ddot{X} \longrightarrow \ddot{Y}$$

Computation

$$X, Z \xrightarrow{X \to Y} Y, Z$$

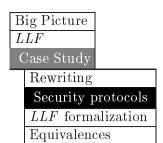
Parametric multisets

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

 $\hookrightarrow$  computation relies on unification

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

$$\ddot{X}(\vec{t}) \longrightarrow \hookrightarrow \vec{x}. \ \ddot{Y}(\vec{t}, \vec{x})$$



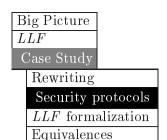
## Message Exchange

$$A \longrightarrow B : M$$

- Local state transitions
- Interaction with the network

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

$$B_j(\vec{b}), N^-(M) \longrightarrow B_{j'}(\vec{b}), \dots$$



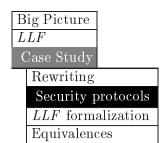
#### **Brand-New Nonces**

• Use counter

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

- $\hookrightarrow$  simplicistic
- $\hookrightarrow$  complicates reasoning
- Use abstraction

 $\hookrightarrow$  not completely realistic



# Cryptography

- Transcribe the encryption/decryption algorithms
  - $\hookrightarrow$  painful (but feasible)
  - → complicates reasoning about protocol issues
  - $\hookrightarrow$  does not allow reasoning about cryptographic issues
- Use abstraction
  - $\hookrightarrow$  constructor:  $\{M\}_k$
  - $\hookrightarrow$  destructor: pattern matching
  - → unrealistic but often acceptable

Big Picture

LLF

Case Study

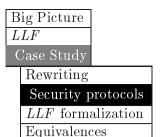
Rewriting

Security protocols LLF formalization

Network

Equivalences
Conclusions

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



#### Intruder

• Doley-Yao model

• More powerful models are possible



Equivalences
Conclusions

## Example

Needham-Schroeder key exchange (simplified)

$$A \longrightarrow B : \{\langle n_a, A \rangle\}_{k_b}$$

$$B \longrightarrow A : \{\langle n_a, n_b, B \rangle\}_{k_a}$$

$$A \longrightarrow B : \{n_b\}_{k_b}$$

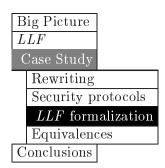
. . .

$$A_{0} \longrightarrow \oplus n_{a}. N^{+}(\{\langle n_{a}, A \rangle\}_{k_{b}}), A_{1}(B, n_{a})$$

$$B_{0}, N^{-}(\{\langle n_{a}, A \rangle\}_{k_{b}}) \longrightarrow \oplus n_{b}. N^{+}(\{\langle n_{a}, n_{b}, B \rangle\}_{k_{a}}), B_{1}(A, n, n_{b})$$

$$A_{1}(B, n_{a}), N^{-}(\{\langle n_{a}, n_{a}, B \rangle\}_{k_{a}}) \longrightarrow N^{+}(\{n\}_{k_{b}}), A_{2}(B, n_{a}, n)$$

$$B_{1}(A, n, n_{b}), N^{-}(\{n_{b}\}_{k_{b}}) \longrightarrow B_{2}(A, n, n_{b}), \dots$$



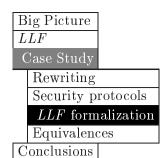
## Linear Logic Strikes Back

Generative multiset rewriting is linear logic undercover

$$\ddot{X}(\vec{x}) \longrightarrow \varphi \vec{y}. \ddot{Y}(\vec{x}, \vec{y})$$

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

The translation preserves the semantics



## Coding in *LLF*

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

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

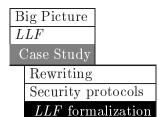
$$\forall \vec{x}. \log \circ - X_1(\vec{x})$$

$$\cdots$$

$$\circ - X_n(\vec{x})$$

$$\circ - \forall \vec{y}. (Y_1(\vec{x}, \vec{y}) \multimap$$

$$Y_m(\vec{x}, \vec{y}) \multimap \log$$

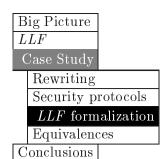


Equivalences
Conclusions

## 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).
```

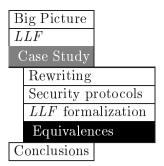
#### Needham-Schroeder



## Uses of LLF

- Simulation
  - $\hookrightarrow$  trivial
- Attack detection
  - $\hookrightarrow$  tricky
- Reasoning
  - $\hookrightarrow$  feasible





## No Net

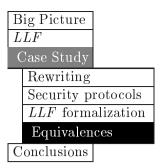
#### Theorem

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

Yields huge savings during protocol analysis



## **LLF** Formalization

- ullet 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,

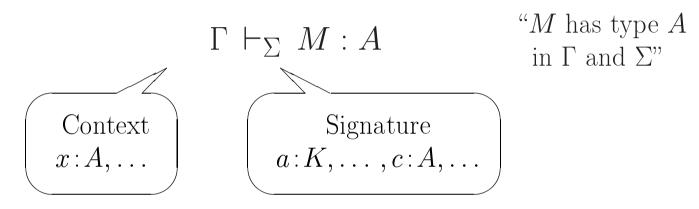
- combines the meta-reasoning power of logical frameworks with the ability of handling state of linear logic
- ullet conservative extension of the logical framework LF
- implemented as a linear logic programming language
- used for the representation of
  - imperative programming languages
  - substructural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

## **Future Directions**

- $\bullet$  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



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

$$x_i: au_i,\ldots$$
 $T$ 
 $\Omega \vdash e: au$ 
 $M$ 

$$\lceil \Omega \rceil \vdash_{\Sigma} M$$
 : has\_type  $\lceil e \rceil \lceil \tau \rceil$ 

where for each  $x_i : \tau_i$  in  $\Omega$ ,

$$\lceil x_i : \tau_i \rceil = x_i : exp, \ t_i : has\_type \ x_i \lceil \tau_i \rceil$$

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

#### Problem!

$$\begin{array}{cccc}
\hline
c_i = v_i, \dots \\
& \mathcal{E} \\
& S \triangleright K \vdash e \hookrightarrow a
\end{array} = M$$

$$\lceil S \rceil \vdash_{\Sigma} M : \operatorname{eval} \lceil K \rceil \lceil e \rceil \lceil a \rceil$$

#### This does not work!

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

#### A way out ...

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

- ... but, we must encode explicitly
  - context operations (lookup, insertion, ...)
  - context-related properties (weakening, exchange, ...)