The Logical Meeting Point of Multiset Rewriting and Process Algebra

Iliano Cervesato
iliano@math.tulane.edu

ITT Industries, Inc -> Tulane University

http://theory.stanford.edu/~iliano
Motivations

- Security protocol specifications
  - Transition-based
  - Process-based
  - Different languages and techniques
  - Ad-hoc translations

- Attempt at a unified approach
  - Rewriting re-interpretation of logic
    - Open derivations
    - Left rule semantics
  - Foundation of multiset rewriting
  - Bridge to process algebra
  - Effective protocol specification language
Outline

Linear Logic

System $\omega$

Multiset Rewriting

Process Algebra

Security Protocols


I. Cervesato: The Logical Meeting Point of MSR and PA
Linear Logic

• Formulas

\[ A, B ::= a \mid 1 \mid A \otimes B \mid A \rightarrow_o B \mid ! A \mid T \mid A & B \mid \forall x. A \mid \exists x. A \]

• LV sequents

\[ \Gamma ; \Delta \rightarrow^\Sigma \Sigma \]

- Constructor: "\,"
- Empty: "\cdot"
Some LV Rules

Left rules

\[
\begin{align*}
\Gamma; \Delta, A, B & \rightarrow_{\Sigma} C \\
\Gamma; \Delta, A \otimes B & \rightarrow_{\Sigma} C \\
\Gamma; \Delta' & \rightarrow_{\Sigma} A \\
\Gamma; \Delta, B & \rightarrow_{\Sigma} C \\
\Gamma; \Delta, \Delta', A \rightarrow_{o} B & \rightarrow_{\Sigma} C \\
\Sigma |- t & \\
\Gamma; \Delta, \lbrack t/x \rbrack A & \rightarrow_{\Sigma} C \\
\Gamma; \Delta, \forall x. A & \rightarrow_{\Sigma} C \\
\Gamma; \Delta, A & \rightarrow_{\Sigma, x} C \\
\Gamma; \Delta, \exists x. A & \rightarrow_{\Sigma} C \\
\Gamma, A; \Delta & \rightarrow_{\Sigma} C \\
\Gamma; \Delta, !A & \rightarrow_{\Sigma} C
\end{align*}
\]

Right rules

\[
\begin{align*}
\Gamma; \Delta & \rightarrow_{\Sigma} C
\end{align*}
\]

Structural rules

\[
\begin{align*}
\Gamma; A & \rightarrow_{\Sigma} A \\
\Gamma, A; \Delta, A & \rightarrow_{\Sigma} C \\
\Gamma, A; \Delta & \rightarrow_{\Sigma} C
\end{align*}
\]

Cut rules

\[
\begin{align*}
\Gamma; \Delta' & \rightarrow_{\Sigma} A \\
\Gamma; \Delta, A & \rightarrow_{\Sigma} C \\
\Gamma; \Delta, \Delta' & \rightarrow_{\Sigma} C \\
\Gamma; \bullet & \rightarrow_{\Sigma} A \\
\Gamma, A; \Delta & \rightarrow_{\Sigma} C \\
\Gamma; \Delta & \rightarrow_{\Sigma} C
\end{align*}
\]
Logical Derivations

- Proof of $C$ from $\Delta$ and $\Gamma$
  - Emphasis on $C$
    - $C$ is input
- Finite
  - Closed

- Rules shown
  - Major premise
    - Preserves $C$
  - Minor premise
    - Starts subderivation
A Rewriting Re-Interpretation

- Transition
  - From conclusion
  - To major premise
  - Emphasis on \( \Gamma, \Delta \) and \( \Sigma \)
  - \( C \) is output, at best
    - Does not change

- Possibly infinite
  - Open

- Minor premise
  - Auxiliary rewrite chain
    - Finite
  - Topped with axiom
State and Transitions

- **States**
  - $\Sigma$ is a list
  - $\Gamma$ and $\Delta$ are commutative monoids
  - No $C$
    - Does not change

- **Transitions**
  - $\Sigma; \Gamma; \Delta \rightarrow \Sigma'; \Gamma'; \Delta'$
  - $\rightarrow^*$ for reflexive and transitive closure

*Constructor: "","*
*Empty: "."*
Interpreting Unary Rules

- \( \Gamma; \Delta, A, B \rightarrow_{\Sigma} C \)
- \( \Gamma; \Delta, A \otimes B \rightarrow_{\Sigma} C \)
- \( \Sigma; \Gamma; (\Delta, A \otimes B) \rightarrow \Sigma; \Gamma; (\Delta, A, B) \)
- \( \Sigma; \Gamma; (\Delta, [t/x]A) \rightarrow \Sigma; \Gamma; (\Delta, [t/x]A) \)

- \( \Sigma \vdash t \)
- \( \Gamma; \Delta, [t/x]A \rightarrow_{\Sigma} C \)
- \( \Gamma; \Delta, \forall x. A \rightarrow_{\Sigma} C \)
- \( \Sigma; \Gamma; (\Delta, \forall x. A) \rightarrow \Sigma; \Gamma; (\Delta, [t/x]A) \)

- \( \Sigma \vdash t \)
- \( \Gamma; \Delta, \exists x. A \rightarrow_{\Sigma} C \)
- \( \Sigma; \Gamma; (\Delta, \exists x. A) \rightarrow (\Sigma, x); \Gamma; (\Delta, A) \)

- \( \Sigma \vdash t \)
- \( \Gamma, A; \Delta \rightarrow_{\Sigma} C \)
- \( \Sigma; \Gamma; (\Delta, !A) \rightarrow \Sigma; (\Gamma, A); \Delta \)

...
Binary Rules and Axiom

- Minor premise
  - Auxiliary rewrite chain
- Top of tree
  - Focus shifts to RHS
    - Axiom rule
  - Observation

\[
\begin{align*}
\Gamma; \Delta' &\rightarrow_\Sigma A & \Gamma; \Delta, B &\rightarrow_\Sigma C \\
\Gamma; \Delta, \Delta', A &\rightarrow_\Sigma B &\rightarrow_\Sigma C
\end{align*}
\]
Observations

- Observation states

  $\Sigma$ ; $\Delta$

  - In $\Delta$, we identify
    - , with $\otimes$
    - $\bullet$ with 1
  
  Categorical semantics

  - Identified with $\exists x_1. \ldots \exists x_n. \Delta$
    - For $\Sigma = x_1, \ldots, x_n$
  
  De Bruijn's telescopes

- Observation transitions

  $\Sigma; \Gamma; \Delta \rightarrow^* \Sigma'; \Delta'$

\[ \Delta = \otimes \Delta \]

\[ \Sigma; \Delta = \exists \Sigma. \otimes \Delta \]
Structural Equivalences

Monoidal laws
- $A \otimes B = B \otimes A$
- $A \otimes 1 = A$
- $(A \otimes B) \otimes C = A \otimes (B \otimes C)$

Mobility laws
- $\exists x. \exists y. \Delta = \exists y. \exists x. \Delta$
- $\exists x. \bullet = \bullet$
- $\exists x. (\Delta, \Delta') = \Delta, \exists x. \Delta'$
  if $x \notin FV(\Delta)$

• Logical bi-equivalences
  - Require limited right rules

• Express structure of context / binders

• Expand rewrite opportunities
Interpreting Binary Rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Gamma; A \rightarrow_{\Sigma} A$</td>
<td>$\Sigma; \Gamma; \Delta \rightarrow^* \Sigma; \Delta$</td>
</tr>
<tr>
<td></td>
<td>$\Sigma; \Gamma; \Delta \rightarrow^* \Sigma''; \Delta''$</td>
</tr>
<tr>
<td></td>
<td><strong>if</strong> $\Sigma; \Gamma; \Delta \rightarrow \Sigma'; \Gamma'; \Delta'$</td>
</tr>
<tr>
<td></td>
<td><strong>and</strong> $\Sigma'; \Gamma'; \Delta' \rightarrow^* \Sigma''; \Delta''$</td>
</tr>
<tr>
<td>$\Gamma; \Delta' \rightarrow_{\Sigma} A; \Gamma; \Delta, B \rightarrow_{\Sigma} C$</td>
<td>$\Sigma; \Gamma; (\Delta, \Delta', A \rightarrow_{o} B) \rightarrow \Sigma; \Gamma; (\Delta, B)$</td>
</tr>
<tr>
<td></td>
<td><strong>if</strong> $\Sigma; \Gamma; \Delta' \rightarrow^* \Sigma; A$</td>
</tr>
<tr>
<td>$\Gamma; \Delta' \rightarrow_{\Sigma} A; \Gamma; \Delta, A \rightarrow_{\Sigma} C$</td>
<td>$\Sigma; \Gamma; \Delta, \Delta' \rightarrow \Sigma; \Gamma; (A, \Delta)$</td>
</tr>
<tr>
<td></td>
<td><strong>if</strong> $\Sigma; \Gamma; \Delta' \rightarrow^* \Sigma; A$</td>
</tr>
</tbody>
</table>

...
Formal Correspondence

• Soundness

\[
\text{If } \Sigma ; \Gamma ; \Delta \Rightarrow^* \Sigma, \Sigma'; \Delta' \text{ then } \Gamma ; \Delta \Rightarrow_{\Sigma} \exists \Sigma', \otimes \Delta'
\]

• Completeness?

➢ No! We have only crippled right rules

\[
\bullet ; \bullet ; a \rightarrow o b, b \rightarrow o c \quad \overset{\text{}}{\not\rightarrow} \quad \bullet ; a \rightarrow o c
\]
System $\omega$

- With cut, rule for $\text{o}$ can be simplified to $\Sigma; \Gamma; (\Delta, A, A \rightarrow \text{o} B) \rightarrow \Sigma; \Gamma; (\Delta, B)$

- Cut elimination holds
  - = in-lining of auxiliary rewrite chains
  - But ...
    - Careful with extra signature symbols
    - Careful with extra persistent objects

- No rule for $\rightarrow$ needs a premise
  - $\rightarrow$ does not depend on $\rightarrow^*$
Discussion

- Other connectives?
  - $\oplus$, 0, $\exists$, $\bot$
    - Odd rewrite properties
  - $\otimes$, ($\_\_$)$\bot$
    - Not yet explored
- Other presentations?
- Other logics?
- Other forms of proof-as-computation?
  - Dual of logic programming
  - Similar to ACL [Kobayashi & Yonezawa, 93]
- Can logic benefit?
Type Theoretic Side

- Very close to CLF
  - Concurrent Logical Framework
  - Linear type theory with
    - Dependent function types: $\Pi$ (LF)
    - Asynchronous connectives: $\rightarrow, \&, T$ (LLF)
    - Synchronous connectives: $\otimes, 1, !, \exists$
    - Monadic sandboxing
    - Concurrency equations
  - Faithful encoding of true concurrency
    - Petri nets, MSR 2 specs, $\pi$-calculus, concurrent ML

- Details of relation still unclear
Multiset Rewriting

- **Multiset**: set with repetitions allowed
  \[ a ::= \bullet | a, a \]
  - Commutative monoid

- **Multiset rewriting** (a.k.a. Petri nets)
  - Rewriting within the monoid
  - Fundamental model of distributed computing
    - Competitor: Process Algebras
  - Basis for security protocol spec. languages
    - MSR family
    - ... several others
  - Many extensions, more or less ad hoc
First-Order Multiset Rewriting

- Multiset elements are FO atomic formulas
- Rules have the form
  \( \forall x_1 \ldots x_n. \ a(x) \rightarrow \exists y_1 \ldots y_k. \ b(x, y) \)
- Semantics

\[ \Sigma ; a(t), s \xrightarrow{R} (a(x) \rightarrow \exists y. \ b(x, y)) \quad \Sigma, y ; b(t, y), s \]

if \( \Sigma \vdash t \)

- Several encodings into linear logic
  - [Martí-Oliet, Meseguer, 91]
ω-Multisets vs. Multiset Rewriting

• MSR 1 is an instance of ω-multisets
  - Uses only ⊗, 1, ∀, ∃, and ⎯ο
  - ⎯ο never nested, always persistent

$$\Sigma ; s \rightarrow_{R} \Sigma' ; s'$$
iff
$$\Sigma ; "R" ; "s" \rightarrow^{*} \Sigma' ; "s'"$$

• Interpretation of MSR as linear logic
  - Logical explanation of multiset rewriting
    - MSR is logic
    - Guideline to design rewrite systems
ω-Rewriting

\( A, B ::= a \) \quad \text{atomic object}

\[ 1 \quad \text{empty} \]

\[ A \otimes B \quad \text{formation} \]

\[ A \rightarrow_0 B \quad \text{rewrite} \]

\[ T \quad \text{no-op} \]

\[ A \& B \quad \text{choice} \]

\[ \forall x. A \quad \text{instantiation} \]

\[ \exists x. A \quad \text{generation} \]

\[ ! A \quad \text{replication} \]
The Asynchronous $\pi$-Calculus

Another fundamental model of distributed computing

- **Language**
  
  $$P ::= 0 \mid P || Q \mid \nu x. P \mid !P \mid x(y).P \mid x<y>$$

- **Semantics**
  
  - **Structural equivalence**
    - Comm. monoidal congruence of $||$ and $0$
    - Binder mobility congruence of $\nu$
      
      - $\nu x. \nu y. P \equiv \nu y. \nu x. P$
      - $0 \equiv \nu x. 0$
      - $P || \nu x. Q \equiv \nu x. (P || Q)$ if $x \notin FN(P)$
    - $!P \equiv !P || P$

  - **Reaction law**
    
    $$x<y> || x(z).P || Q \Rightarrow [y/z]P || Q$$

I.Cervesato: The Logical Meeting Point of MSR and PA
\[\pi\text{-calculus in } \omega\text{-Multisets}\]\begin{itemize}
  \item $0 \equiv 1$
  \item $\mid\mid \equiv \otimes$
  \item $\nu \equiv \exists$
  \item $! ! \equiv ! !$
  \item $x(y). P \equiv \forall y. ch(x,y) \rightarrow P$
  \item $x\langle y \rangle \equiv ch(x,y)$
\end{itemize}

- **Reaction law**
  \[\Sigma; \Gamma; ch(x,y), \forall z. ch(x,z) \rightarrow P, \Delta \rightarrow^2 \Sigma; \Gamma; [y/z]P, \Delta\]

- **Structural equivalence**
  \[\text{Monoidal congr. of } \mid\mid \text{ and } 0 \equiv \text{monoidal laws of } \otimes \text{ and } 1\]
  \[\text{Mobility congr. of } \nu \equiv \text{mobility laws of } \exists\]
  \[!P \equiv !P \mid\mid P\]
    \begin{itemize}
      \item Only $\Rightarrow$ in $\omega$-multisets
      \item Oversight in the $\pi$-calculus?
    \end{itemize}
Properties

• If $P \Rightarrow^* Q$
  
  then $\bullet; \bullet; "P" \Rightarrow^* \Sigma; \Gamma; \Delta$

  where "Q" = $\exists \Sigma. !\Gamma \otimes \Delta$  \text{ mod } !A = !A \otimes A$

  ➢ Note: with $!P \parallel P \Rightarrow !P$ as a transition
    - If $P \Rightarrow^* Q$
      
      then $\bullet; \bullet; "P" \Rightarrow^* \Sigma; \Gamma; \Delta$

      where "Q" = $\exists \Sigma. !\Gamma \otimes \Delta$
ω-Multisets vs. Process Algebra

- Simple encoding of asynchronous π-calculus into ω-multisets
  - Doesn’t show that π-calculus is logic
  - Uses only a fraction of ω-multiset syntax
  - Inverse encoding?
    - As hard as going from multiset rewriting to π-calculus

- Other languages
  - Join calculus
  - Strand spaces
  - To do: Synchronous π-calculus
MSR 3

- Instance of $\omega$-multisets for cryptographic protocol specification
  - Security-relevant signature
  - Typing infrastructure
  - Modules, equations, ...

- 3rd generation
  - MSR 1: First-order multiset rewriting with $\exists$
    - Undecidability of protocol analysis
  - MSR 2: MSR 1 + typing
    - Actual specification language
    - More theoretical results
    - Implementation underway
The Atomic Objects of MSR 3

**Atomic terms**
- Principals \( A \)
- Keys \( K \)
- Nonces \( N \)
- Other
  - Raw data, timestamp, ...

**Constructors**
- Encryption \( \{\} \)\
- Pairing \( (\_ , \_ ) \)
- Other
  - Signature, hash, MAC, ...

**Predicates**
- Network \( \text{net} \)
- Memory \( \text{MA} \)
- Intruder \( I \)
- ...

**Fully definable**
Types

- **Simple types**
  - $A : \text{princ}$
  - $n : \text{nonce}$
  - $m : \text{msg, ...}$

- **Dependent types**
  - $k : \text{shK A B}$
  - $K : \text{pubK A}$
  - $K' : \text{privK K, ...}$

**Fully definable**

- **Powerful abstraction mechanism**
  - At various user-definable level
    - Finely tagged messages
    - Untyped: $\text{msg only}$

- **Simplify specification and reasoning**
- **Automated type checking**
Example

Needham-Schroeder public-key protocol

1. $A \rightarrow B: \{n_A, A\}_kB$
2. $B \rightarrow A: \{n_A, n_B\}_kA$
3. $A \rightarrow B: \{n_B\}_kB$

• Can be expressed in several ways
  - State-based
    - Explicit local state
    - As in MSR 2
  - Process-based: embedded
    - Continuation-passing style
    - As in process algebra
  - (Intermediate approaches)
∀A: princ.

{ ∀B: princ. ∀k_B: pubK B. 
• 
→ ∀n_A: nonce.
net ({n_A, A}_{k_B}), L (A, B, k_B, n_A) 

∀B: princ. ∀k_B: pubK B. ∀k_A: prvK k_A. 
∀n_A: nonce. ∀n_B: nonce.
net ({n_A, n_B}_{k_A}), L (A, B, k_B, n_A) 
→ net ({n_B}_{k_B}) 

}
Process-Based

\( \forall A: \text{princ.} \)
\( \forall B: \text{princ.} \forall k_B: \text{pubK B.} \)

- \( \rightarrow \exists n_A: \text{nonce.} \)
  \[
  \text{net (} \{n_A, A\}_kB) ,
  \]

- \( (\forall k_A: \text{pubK A.} \forall k'_A: \text{prvK k}_A. \forall n_B: \text{nonce.} \)
  \[
  \text{net (} \{n_A, n_B\}_kA) \rightarrow \text{net (} \{n_B\}_kB) \)
  \]

- **Succinct**
- **Continuation-passing style**
  - Rule asserts what to do next
  - Lexical control flow
- **State is implicit**
  - Abstract

---

I. Cervesato: The Logical Meeting Point of MSR and PA
NSPK in Process Algebra

∀A: princ.
∀B: princ. ∀kB: pubK B.
∀kA: pubK A. ∀kA': prvK kA. ∀nB: nonce.

∀nA: nonce.
net [{nA, A}kB].

net <{nA, nB}kA>.
net [{nB}kB]. 0

Same structure!

- Not a coincidence
- MSR 3 very close to Process Algebra
  - ω-multiset encodings of π-calculus
  and Join Calculus

- MSR 3 is promising middle-ground for relating
  - State-based
  - Process-based

representations of a problem
State-Based vs. Process-Based

- **State-based languages**
  - Multiset Rewriting
  - NRL Prot. Analyzer, CAPSL/CIL, Paulson’s approach, ...
  - State transition semantics

- **Process-based languages**
  - Process Algebra
  - Strand spaces, spi-calculus, ...
  - Independent communicating threads
MSR 3 Bridges the Gap

- Difficult to go from one to the other
  - Different paradigms

State vs. process distance

State ↔ Process translation done once and for all in MSR 3

I. Cervesato: The Logical Meeting Point of MSR and PA
Conclusions

• \(\omega\)-multisets
  - Logical foundation of multiset rewriting
  - Relationship with process algebras
  - Unified logical view
    - Better understanding of where we are
    - Hint about where to go next

• MSR 3.0
  - Language for security protocol specification
  - Succinct representations
    - Simpler specifications
    - Economy of reasoning
  - Bridge between
    - State-based representation
    - Process-based representation