Relating State-Based and Process-Based Concurrency through Linear Logic

Iliano Cervesato
Carnegie Mellon University - Qatar
iliano@cmu.edu
Specifying Concurrent Systems

● Two main approaches
  + Transition-based
    ➢ Petri nets, multiset rewriting, …
  + Process-based
    ➢ Process algebras, …

● No language supports both
  + Different linguistic features
  + Different analysis methods

● Ad hoc translations

● Concurrency inherent to many problems
  + Cryptographic protocols
  + …
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
Summary of Results

● **System ω**
  + Rationalization of multiset rewriting
    ➢ Traditional multiset rewriting are sublanguages
    ➢ Simpler, but much more expressive
  + Significant bridge to process algebras
    ➢ Popular algebras are sublanguages
  + Both in the same seamless formalism
  + Proof-theoretic foundations

● **MSR 3**
  + Specialization of ω to security protocols
Methodology

- Rewriting re-interpretation of linear logic
  + Open derivations
  + Left-rule semantics
- Successive refinements of LV sequent system
Further Developments

● Verifying specifications
  + Transferring methods
    ➢ Equivalence, bisimulation in \( \omega \)
  + Model checking

● Additional application domains
  + Massively distributed systems
    ➢ Claytronics
  + Molecular biology
    ➢ Modeling cellular pathways
  + Micro-economic simulation
    ➢ Predicting effect of policies
Logical Foundations

- Linear logic in LV
- Tensorial observations – $LV^{obs}_{1\otimes}$
- Tensorial-existential observations – $LV^{obs}_{1\otimes\exists}$
- Cut-elimination
- Rewriting interpretation
- The system $\omega$
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**
  
  $$\Gamma ; \Delta \rightarrow \Sigma C$$

  - Constructor: “,”
  - Empty: “•”
Some LV Rules

**Left rules**

\[ \Gamma; \Delta, A, B \rightarrow_{\Sigma} C \]
\[ \Gamma; \Delta, A \otimes B \rightarrow_{\Sigma} C \]
\[ \Gamma; \Delta' \rightarrow_{\Sigma} A \quad \Gamma; \Delta, B \rightarrow_{\Sigma} C \]
\[ \Gamma; \Delta, \Delta', A \cdot o B \rightarrow_{\Sigma} C \]
\[ \Sigma |- \top \quad \Gamma; \Delta, [\top/x] A \rightarrow_{\Sigma} C \]
\[ \Gamma; \Delta, \forall x. A \rightarrow_{\Sigma} C \]
\[ \Gamma; \Delta, A \rightarrow_{\Sigma, x} C \]
\[ \Gamma, A; \Delta \rightarrow_{\Sigma} C \]
\[ \Gamma; \Delta, ! A \rightarrow_{\Sigma} C \]

**Cut rules**

\[ \Gamma; \Delta' \rightarrow_{\Sigma} A \quad \Gamma; \Delta, A \rightarrow_{\Sigma} C \]
\[ \Gamma; \Delta, \Delta' \rightarrow_{\Sigma} C \]
\[ \Gamma; \bullet \rightarrow_{\Sigma} A \quad \Gamma, A; \Delta \rightarrow_{\Sigma} C \]
\[ \Gamma; \Delta \rightarrow_{\Sigma} C \]

**Right rules**

\[ \Gamma; \Delta \rightarrow_{\Sigma} 1 \]
\[ \Gamma; \Delta_{1} \rightarrow_{\Sigma} C_{1} \quad \Gamma; \Delta_{2} \rightarrow_{\Sigma} C_{2} \]
\[ \Gamma; \Delta_{1}, \Delta_{2} \rightarrow_{\Sigma} C_{1} \otimes C_{2} \]
\[ \Sigma |- \top \quad \Gamma; \Delta \rightarrow_{\Sigma} [\top/x] C \]
\[ \Gamma; \Delta \rightarrow_{\Sigma} \exists x. C \]

**Structural rules**

\[ \Gamma; A \rightarrow_{\Sigma} A \]
\[ \Gamma, A; \Delta \rightarrow_{\Sigma} C \]
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
Observations

- Close derivation to “observe” \( \Gamma, \Delta \) and \( \Sigma \)
  - At any point

- Use \( C \) to propagate observation to top level

- How to engineer this?
  - Restrict right rules (dramatically)
  - Get rid of binary rules
Tensorial observations – $LV_{obs}^{\otimes}$

- True in LV:
  - $+ \Gamma; \Delta \rightarrow_{\Sigma} \otimes \Delta$
  - $+ \Gamma; \Delta \rightarrow_{\Sigma} C$ \iff $\Gamma; \otimes \Delta \rightarrow_{\Sigma} C$

- $LV_{1\otimes}$: remove all right rules except for 1 and $\otimes$
  - $+ \Gamma; \Delta \rightarrow_{\Sigma} C$ \iff $\Gamma; \otimes \Delta \rightarrow_{\Sigma} C$

- $LV_{obs}^{1\otimes}$: replace right rules and id with

  $\Gamma; \Delta \rightarrow_{\Sigma} \otimes \Delta$

  $+ \Gamma; \Delta \rightarrow_{\Sigma} C$ in $LV_{1\otimes}$ \iff $\Gamma; \Delta \rightarrow_{\Sigma} C$ in $LV_{obs}^{1\otimes}$
Nominal Quantification

\[ \Gamma; \Delta \rightarrow^\Sigma, x \ C \]

\[ \Gamma; \Delta \rightarrow^\Sigma, x \ \exists x . C \]

- Binds all occurrences of \( x \) in \( C \)
  + Reification of a sequent-level binder
- All interpretations of concurrent languages rely on it
  + Often unknowingly
Nominal Observations – $\Lambda V^{\text{obs}}_{1 \otimes \exists}$

- True in LV:
  - Mobility laws
  - $\Gamma; \Delta \rightarrow_{\Sigma, \Sigma'} \exists \Sigma'. \Delta$
  - $\Gamma; \Delta \rightarrow_{\Sigma, \Sigma'} C$ iff $\Gamma; \exists \Sigma'. \Delta \rightarrow_{\Sigma} C$ if $\Sigma' \cap \text{FV}(\Gamma, C) = \emptyset$

- $\Lambda V_{1 \otimes}$: remove all right rules except $1, \otimes$ and $\exists$
  - Expressiveness limited to collecting context

- $\Lambda V^{\text{obs}}_{1 \otimes \exists}$: replace right rules and id with

\[\Gamma; \Delta \rightarrow_{\Sigma, \Sigma'} \exists \Sigma'. \Delta\]

- If $\Gamma; \Delta \rightarrow_{\Sigma} C$ in $\Lambda V_{1 \otimes \exists}$, then $C \equiv \exists \Sigma'. \Delta'$ and $\Gamma; \Delta \rightarrow_{\Sigma} \exists \Sigma'. \Delta'$ in $\Lambda V^{\text{obs}}_{1 \otimes \exists}$
- If $\Gamma; \Delta \rightarrow_{\Sigma} C$ in $\Lambda V^{\text{obs}}_{1 \otimes \exists}$, then $\Gamma; \Delta \rightarrow_{\Sigma} C$ in $\Lambda V_{1 \otimes \exists}$
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
+ $\exists x. \exists y. A = \exists y. \exists x. A$
+ $\exists x. 1 = 1$
+ $\exists x. (A \otimes B) = A \otimes \exists x. B$
  if $x \not\in FV(A)$

• Logical bi-equivalences
  ➢ Require limited right rules
• Express structure of context / binders
Dealing with Binary Rules

● Implication
  + Inline side-derivation
  + \( LV^{obs} \): replace right rule for \( \rightarrow_0 \) with

\[
\Gamma; \Delta, B \rightarrow_{\Sigma, \Sigma'} C \\
\Gamma; \Delta, \Delta', \exists \Sigma'. \Delta' \rightarrow_0 B \rightarrow_{\Sigma, \Sigma'} C
\]

+ \( \Gamma; \Delta \rightarrow_{\Sigma} C \) in \( LV^{obs}_{\otimes \exists} \) iff \( \Gamma; \Delta \rightarrow_{\Sigma} C \) in \( LV^{obs} \)

● Cut

+ Cut rules are admissible
  ➢ Simplified adaptation of usual proof
Rewriting Interpretation of $LV^{obs}$

- All rules are unary
  - Except observation rule
- States
  - $\Sigma; \Gamma; \Delta$
    - $\Sigma$ is a list
    - $\Gamma$ and $\Delta$ are commutative monoids
    - No $C$
      - Does not change
- Transitions
  - $\Sigma; \Gamma; \Delta \rightarrow \Sigma'; \Gamma'; \Delta'$

- Constructor: ";"
- Empty: "•"
**LV^{obs} Rules as Rewrite Rules**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rightarrow$</td>
<td>$(\Sigma, \Sigma') ; \Gamma ; (\Delta, \Delta', (\Sigma'.\Delta' \rightarrow B))$</td>
<td>$(\Sigma, \Sigma') ; \Gamma ; (\Delta, B)$</td>
</tr>
<tr>
<td>$\top$</td>
<td>(no rules)</td>
<td></td>
</tr>
<tr>
<td>$&amp;$</td>
<td>$\Sigma ; \Gamma ; (\Delta, A_1 &amp; A_2)$</td>
<td>$\Sigma ; \Gamma ; (\Delta, A_i)$</td>
</tr>
<tr>
<td>$\forall$</td>
<td>$\Sigma ; \Gamma ; (\Delta, \forall x. A)$</td>
<td>$\Sigma ; \Gamma ; (\Delta, [t/x]A)$ if $\Sigma</td>
</tr>
<tr>
<td>$\exists$</td>
<td>$\Sigma ; (\Gamma, A) ; \Delta$</td>
<td>$\Sigma ; (\Gamma, A) ; (\Delta, A)$</td>
</tr>
<tr>
<td>$1$</td>
<td>$\Sigma ; \Gamma ; (\Delta, 1)$</td>
<td>$\Sigma ; \Gamma ; \Delta$</td>
</tr>
<tr>
<td>$\otimes$</td>
<td>$\Sigma ; \Gamma ; (\Delta, A \otimes B)$</td>
<td>$\Sigma ; \Gamma ; (\Delta, A, B)$</td>
</tr>
<tr>
<td>$\exists$</td>
<td>$\Sigma ; \Gamma ; (\Delta, \exists x. A)$</td>
<td>$(\Sigma, x) ; \Gamma ; (\Delta, A)$</td>
</tr>
</tbody>
</table>
Formal Correspondence

• **Wrt** $LV^{obs}$
  
  + If $\Sigma ; \Gamma ; \Delta \rightarrow^* (\Sigma, \Sigma'); (\Gamma, \Gamma') ; \Delta'$, then $\Gamma; \Delta \rightarrow_{\Sigma} \exists \Sigma'. \Delta'$ in $LV^{obs}$

  + If $\Gamma; \Delta \rightarrow_{\Sigma} C$ in $LV^{obs}$, then $C \equiv \exists \Sigma'. \Delta'$ and $\Sigma ; \Gamma ; \Delta \rightarrow^* (\Sigma, \Sigma'); (\Gamma, \Gamma') ; \Delta'$

• **Wrt** $LV$
  
  + Sound

  + Not complete

    ➤ *No!* We have only crippled right rules

      $$\bullet ; \bullet ; a \rightarrow o b, b \rightarrow o c \rightarrow^* \bullet ; a \rightarrow o c$$
System $\omega$

- Monoidal equivalences allow identifying
  + $\otimes$ with linear context constructor “,”
  + $1$ with empty linear context “•”

- Correspondence with logic
  + If $\Sigma; \Gamma; \Delta \rightarrow^* (\Sigma,\Sigma’); (\Gamma,\Gamma’); \Delta’$, then $\Gamma; \Delta \rightarrow^*_\Sigma \exists \Sigma’.\Delta’ in LV^{obs}$
  + If $\Gamma; \Delta \rightarrow^*_\Sigma C in LV^{obs}$, then $C \equiv \exists \Sigma’.\Delta’$ and $\Sigma; \Gamma; \Delta \rightarrow^* (\Sigma,\Sigma’); (\Gamma,\Gamma’); \Delta’$
\( \omega - \text{multisets} \)

\[
A, B ::= a \quad \text{atomic object} \\
| \cdot \quad \text{empty} \\
| A, B \quad \text{formation} \\
| A \rightarrow^o B \quad \text{rewrite} \\
| T \quad \text{no-op} \\
| A \& B \quad \text{choice} \\
| \forall x. A \quad \text{instantiation} \\
| \exists x. A \quad \text{generation} \\
| ! A \quad \text{replication}
\]
Discussion

● Other connectives?
  + ⊕, 0, ⚬, ⊥
    ➢ Odd rewrite properties
  + ?, (⊥)
    ➢ Not yet explored

● Other presentations?

● Other logics?
  + Very close to CLF

● Other forms of proof-as-computation?
  + Dual of logic programming
  + Similar to ACL [Kobayashi & Yonezawa, 93]

● Can logic benefit?
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
    - Alternative: 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 F0 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

\[ \Sigma ; a(\uparrow), s \rightarrow_R (a(x) \rightarrow \exists y. \ b(x, y)) \quad \Sigma, y ; b(\uparrow, y), s \]

  if \( \Sigma \vdash \uparrow \)

- Several encodings into linear logic
  \[ \uparrow \quad [\text{Martí-Oliet, Meseguer, 91}] \]
ω-Multisets vs. Multiset Rewriting

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

\[ \Sigma ; s \xrightarrow{R} \Sigma' ; s' \]
iff
\[ \Sigma ; "R" ; "s" \xrightarrow{*} \Sigma' ; "s'" \]

- Interpretation of MSR as linear logic
  - Logical explanation of multiset rewriting

\[ \Sigma ; s \xrightarrow{R} \Sigma' ; s' \]

- MSR is logic
  - Guideline to design rewrite systems
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 $x \not\in \text{FN}(P)$
  
  + **Reaction law**
    
    - $x<y> || x(z). P || Q \Rightarrow [y/z]P || Q$
    - $!P \Rightarrow !P || P$
\[ \pi\text{-calculus in } \omega\text{-Multisets} \]

- \( 0 \Leftrightarrow 1 \)  
- \( \| \Leftrightarrow \otimes \)  
- \( \nu \Leftrightarrow \exists \)  
- \( ! \Leftrightarrow ! \)

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

- Structural equivalence  
  + Monoidal congr. of \( \| \) and \( 0 \Leftrightarrow \) monoidal laws of \( \otimes \) and \( 1 \)  
  + Mobility congr. of \( \nu \Leftrightarrow \) mobility laws of \( \exists \)  
  + \( !P \equiv !P \| P \)
    - Only \( \Rightarrow \) in \( \omega\)-multisets
Properties

- If $P \Rightarrow^* Q$,
  then $\Sigma_P; \bullet; "P" \rightarrow^* (\Sigma_P, \Sigma); \Gamma; \Delta$

  where "$Q$" = $\exists \Sigma. !\Gamma \otimes \Delta$

- If $\Sigma_P; \bullet; "P" \rightarrow^* (\Sigma_P, \Sigma); \Gamma; \Delta$,
  then there exists $Q$ such that "$Q$" = $\exists \Sigma. !\Gamma \otimes \Delta$
  and $P \Rightarrow^* Q$
\(\omega\)-Multisets vs. Process Algebra

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

- Other languages
  - Join calculus
  - Strand spaces
  - To do: Synchronous \(\pi\)-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
Example

Needham-Schroeder public-key protocol

- $A \rightarrow B: \{n_A, A\}_{kB}$
- $B \rightarrow A: \{n_A, n_B\}_{kA}$
- $A \rightarrow B: \{n_B\}_{kB}$

- 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.
{∃L: princ × ∑B: princ.pubK B × nonce → mset.

∀B: princ. ∀kB: pubK B.

•
→ ∃nA: nonce.
net ({nA, A}kB), L (A, B, kB, nA)

∀B: princ. ∀kB: pubK B.
∀kA: pubK A. ∀kA': prvK kA.
∀nA: nonce. ∀nB: nonce.
net ({nA, nB}kA), L (A, B, kB, nA)
→ net ({nB}kB)

Interpretation of L

➢ Rule invocation
  ▪ Implementation detail
  ▪ Control flow

➢ Local state of role
  ▪ Explicit view
  ▪ Important for DOS
Process-Based

∀A: princ.
∀B: princ. ∀kB: pubK B.

• → ∃nA: nonce.

net (\{nA, A\}_{kB}),

(∀kA: pubK A. ∀kA': prvK kA. ∀nB: nonce.

net (\{nA, nB\}_{kA}) → net (\{nB\}_{kB}))

• Succinct
• Continuation-passing style
  ➢ Rule asserts what to do next
  ➢ Lexical control flow

• State is implicit
  ➢ Abstract
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

Other distance

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

● ω-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
**Interpreting Unary Rules**

\[
\begin{align*}
\Gamma; \Delta, A, B &\rightarrow_\Sigma C \\
\Gamma; \Delta, A \otimes B &\rightarrow_\Sigma C \\
\Sigma |- \top; \Gamma; \Delta, [\top/x]A &\rightarrow_\Sigma C \\
\Gamma; \Delta, \forall x. A &\rightarrow_\Sigma C \\
\Gamma; \Delta, A &\rightarrow_\Sigma x C \\
\Gamma; \Delta, \exists x. A &\rightarrow_\Sigma C \\
\Gamma, A; \Delta &\rightarrow_\Sigma C \\
\Gamma; \Delta, !A &\rightarrow_\Sigma C \\
\end{align*}
\]

**Example Rules:**

\[
\begin{align*}
\Sigma; \Gamma; (\Delta, A \otimes B) &\rightarrow \Sigma; \Gamma; (\Delta, A, B) \\
\Sigma; \Gamma; (\Delta, \forall x. A) &\rightarrow \Sigma; \Gamma; (\Delta, [\top/x]A) \\
\Sigma; \Gamma; (\Delta, \exists x. A) &\rightarrow (\Sigma, x); \Gamma; (\Delta, A) \\
\Sigma; \Gamma; (\Delta, !A) &\rightarrow \Sigma; (\Gamma, A); \Delta \\
\end{align*}
\]
Observations

- Observation states
  \[ \Sigma ; \Delta \]
  + In \( \Delta \), we identify
    - \( \o \), 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
  \[ \Sigma; \Gamma; \Delta \rightarrow^* \Sigma'; \Delta' \]
Type Theoretic Side

- Very close to CLF
  - Concurrent Logical Framework
  - Linear type theory with
    - Dependent function types: $\Pi$ (LF)
    - Asynchronous connectives: $\leftarrow o$, &, $T$ (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