

The Logical Meeting Point of Multiset Rewriting and Process Algebra

Iliano Cervesato

iliano@itd.nrl.navy.mil

ITT Industries, inc @ NRL Washington, DC

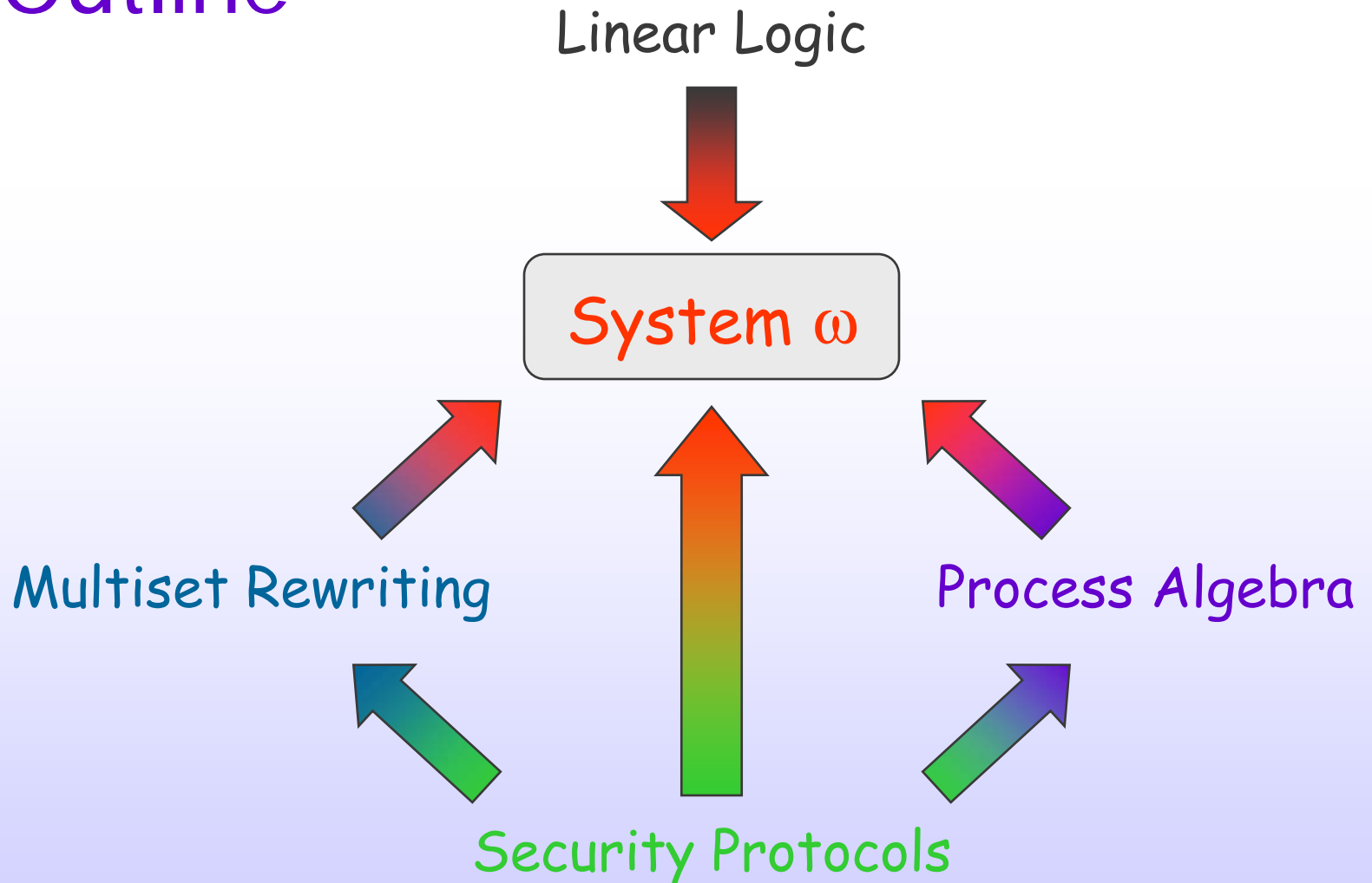
<http://theory.stanford.edu/~iliano>



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



<http://theory.stanford.edu/~iliano/forthcoming.html>

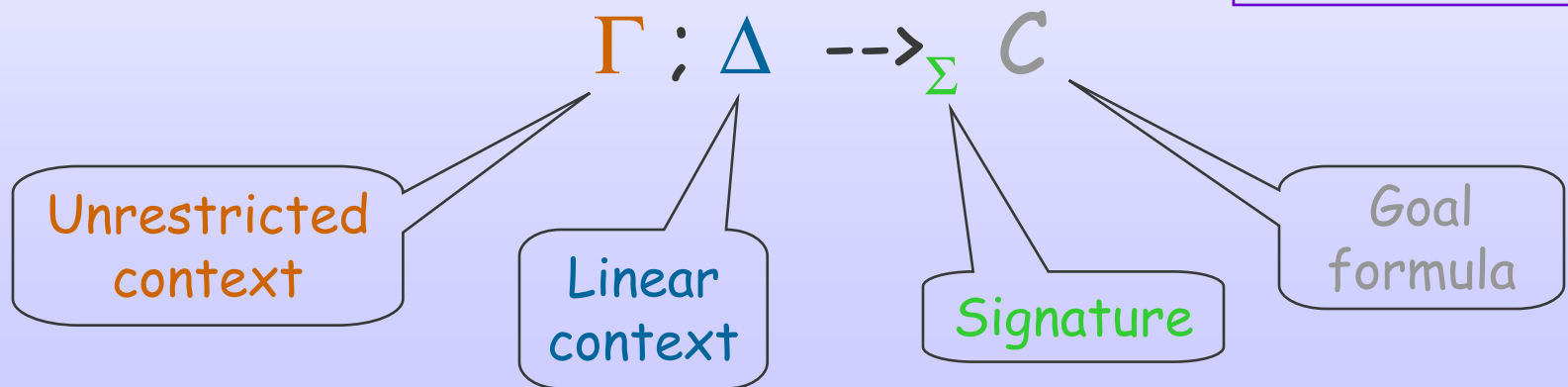
Linear Logic

• Formulas

$$A, B ::= a \mid 1 \mid A \otimes B \mid A \multimap B \mid !A \\ \mid \top \mid A \& B \mid \forall x. A \mid \exists x. A$$

• LV sequents

- Constructor: “,”
- Empty: “•”



- logic
- system ω
- rewriting
- processes
- security

Some LV Rules

Left rules

$$\frac{\Gamma; \Delta, A, B \multimap_{\Sigma} C}{\Gamma; \Delta, A \otimes B \multimap_{\Sigma} C}$$

$$\frac{\Gamma; \Delta' \multimap_{\Sigma} A \quad \Gamma; \Delta, B \multimap_{\Sigma} C}{\Gamma; \Delta, \Delta', A \multimap_{\Sigma} B}$$

$$\frac{\Sigma \vdash \dagger \quad \Gamma; \Delta, [\dagger/x]A \multimap_{\Sigma} C}{\Gamma; \Delta, \forall x.A \multimap_{\Sigma} C}$$

$$\frac{\Gamma; \Delta, A \multimap_{\Sigma, x} C}{\Gamma; \Delta, \exists x.A \multimap_{\Sigma} C}$$

$$\frac{\Gamma, A; \Delta \multimap_{\Sigma} C}{\Gamma; \Delta, !A \multimap_{\Sigma} C}$$

...

Structural rules

$$\frac{\Gamma; A \multimap_{\Sigma} A}{\Gamma, A; \Delta, A \multimap_{\Sigma} C}$$

$$\Gamma, A; \Delta \multimap_{\Sigma} C$$

Cut rules

$$\frac{\Gamma; \Delta' \multimap_{\Sigma} A \quad \Gamma; \Delta, A \multimap_{\Sigma} C}{\Gamma; \Delta, \Delta' \multimap_{\Sigma} C}$$

$$\frac{\Gamma; \bullet \multimap_{\Sigma} A \quad \Gamma, A; \Delta \multimap_{\Sigma} C}{\Gamma; \Delta \multimap_{\Sigma} C}$$

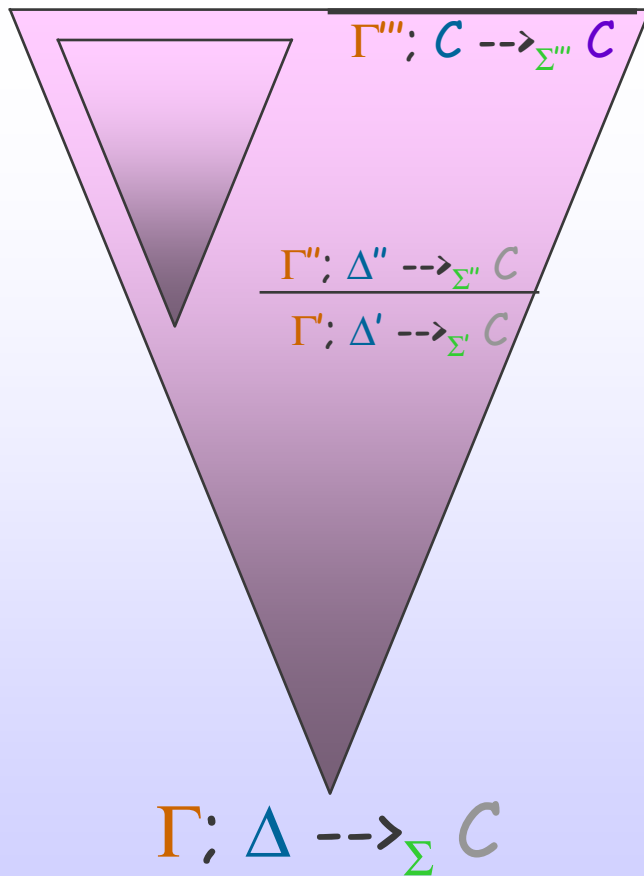
Right rules

...



- logic
- system ω
- rewriting
- processes
- security

Logical Derivations



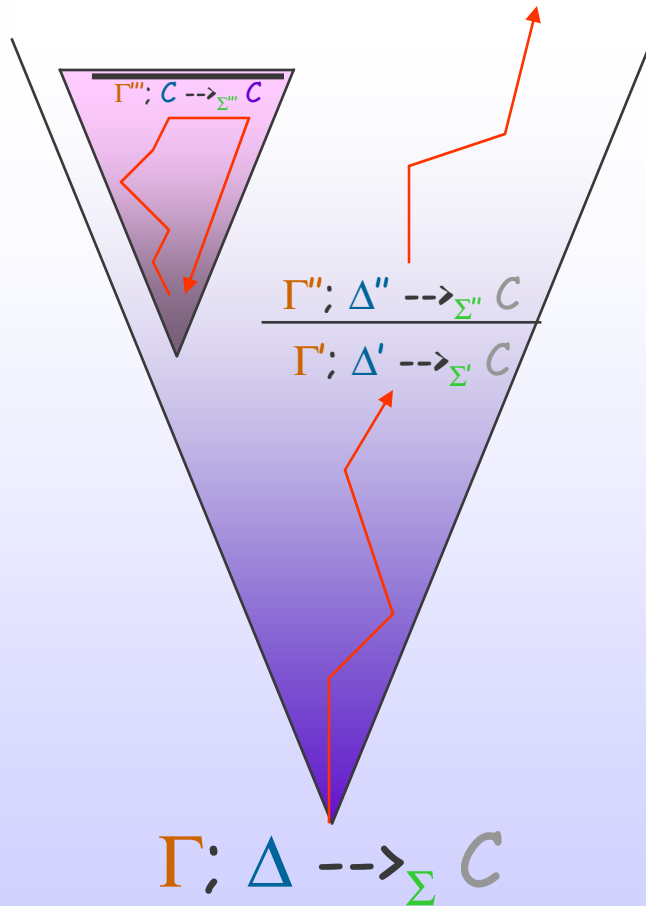
- Proof of C from Δ and Γ
 - Emphasis on C
 - C is input
- Finite
 - Closed
- Rules shown
 - Major premise
 - Preserves C
 - Minor premise
 - Starts subderivation

- logic
- system ω
- rewriting
- processes
- security

A Rewriting Re-Interpretation



- logic
- **system** ω
- rewriting
- processes
- security



- Transition
 - From conclusion
 - To major premise
 - Emphasis on Γ , Δ and Σ
 - C is output, at best
 - Does not change
- Possibly infinite
 - Open
- Minor premise
 - Auxiliary rewrite chain
 - Finite
 - Topped with axiom

State and Transitions

- States

$$\Sigma ; \Gamma ; \Delta$$


- Σ is a list
- Γ and Δ are commutative monoids
- No \mathcal{C}
 - Does not change

- Constructor: “,”
- Empty: “•”

- Transitions

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

- \rightarrow^* for reflexive and transitive closure



- logic
- **system** ω
- rewriting
- processes
- security

Interpreting Unary Rules

$$\frac{\Gamma; \Delta, A, B \dashrightarrow_{\Sigma} C}{\Gamma; \Delta, A \otimes B \dashrightarrow_{\Sigma} C}$$

$$\Sigma; \Gamma; (\Delta, A \otimes B) \rightarrow \Sigma; \Gamma; (\Delta, A, B)$$

$$\frac{\boxed{\Sigma \vdash \dagger} \Gamma; \Delta, [\dagger/x]A \dashrightarrow_{\Sigma} C}{\Gamma; \Delta, \forall x. A \dashrightarrow_{\Sigma} C}$$

$$\Sigma; \Gamma; (\Delta, \forall x. A) \rightarrow \Sigma; \Gamma; (\Delta, [\dagger/x]A)$$

if $\boxed{\Sigma \vdash \dagger}$

$$\frac{\Gamma; \Delta, A \dashrightarrow_{\Sigma, x} C}{\Gamma; \Delta, \exists x. A \dashrightarrow_{\Sigma} C}$$

$$\Sigma; \Gamma; (\Delta, \exists x. A) \rightarrow (\Sigma, x); \Gamma; (\Delta, A)$$

$$\frac{\Gamma, A; \Delta \dashrightarrow_{\Sigma} C}{\Gamma; \Delta, !A \dashrightarrow_{\Sigma} C}$$

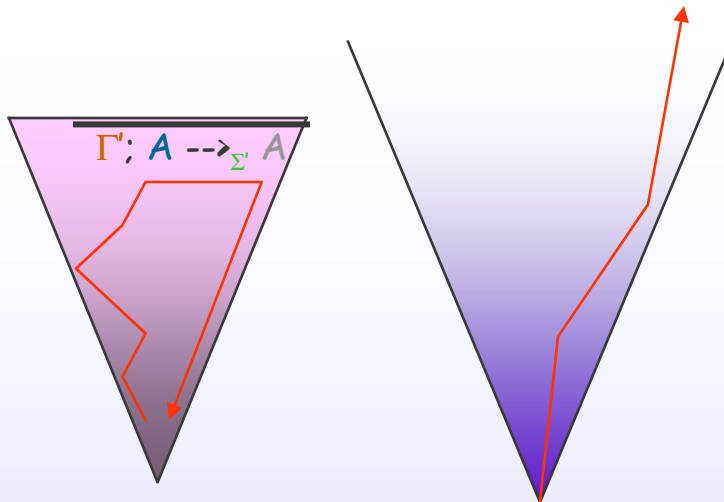
$$\Sigma; \Gamma; (\Delta, !A) \rightarrow \Sigma; (\Gamma, A); \Delta$$

...

...

- logic
- **system** ω
- rewriting
- processes
- security

Binary Rules and Axiom



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

- logic
 - **system** ω
 - rewriting
 - processes
 - security

Observations

- Observation states

$$\Sigma \ ; \ \Delta$$

➤ In Δ , we identify

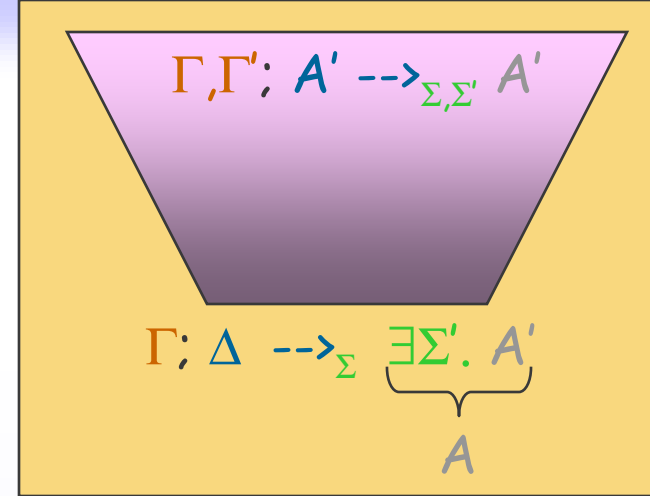
- , with \otimes
- • with 1

Categorical semantics

➤ Identified with $\exists x_1. \dots \exists x_n. \Delta$

- For $\Sigma = x_1, \dots, x_n$

De Bruijn's telescopes



$$\Delta = \otimes \Delta$$

$$\Sigma; \Delta = \exists \Sigma. \otimes \Delta$$

- Observation transitions

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

- logic
- **system** ω
- rewriting
- processes
- security

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 $x \notin \text{FV}(\Delta)$

- Logical bi-equivalences
 - Require limited right rules
- Express structure of context / binders
- Expand rewrite opportunities

- logic
- **system** ω
- rewriting
- processes
- security

Interpreting Binary Rules



$$\frac{}{\Gamma; A \dashrightarrow_{\Sigma} A}$$

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

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

$$\text{if } \Sigma; \Gamma; \Delta \rightarrow \Sigma'; \Gamma'; \Delta' \\ \text{and } \Sigma'; \Gamma'; \Delta' \rightarrow^* \Sigma''; \Delta''$$

$$\frac{\boxed{\Gamma; \Delta' \dashrightarrow_{\Sigma} A} \quad \Gamma; \Delta, B \dashrightarrow_{\Sigma} C}{\Gamma; \Delta, \Delta', A \multimap B \dashrightarrow_{\Sigma} C}$$

$$\Sigma; \Gamma; (\Delta, \Delta', A \multimap B) \rightarrow \Sigma; \Gamma; (\Delta, B) \\ \text{if } \boxed{\Sigma; \Gamma; \Delta' \rightarrow^* \Sigma; A}$$

$$\frac{\boxed{\Gamma; \Delta' \dashrightarrow_{\Sigma} A} \quad \Gamma; \Delta, A \dashrightarrow_{\Sigma} C}{\Gamma; \Delta, \Delta' \dashrightarrow_{\Sigma} C}$$

$$\Sigma; \Gamma; \Delta, \Delta' \rightarrow \Sigma; \Gamma; (A, \Delta) \\ \text{if } \boxed{\Sigma; \Gamma; \Delta' \rightarrow^* \Sigma; A}$$

...

...

- logic
- **system** ω
- rewriting
- processes
- security

Formal Correspondence


- Soundness

If $\Sigma ; \Gamma ; \Delta \rightarrow^* \Sigma, \Sigma' ; \Delta'$
then $\Gamma ; \Delta \dashrightarrow_{\Sigma} \exists \Sigma'. \otimes \Delta'$

- Completeness?

➤ **No!** We have only crippled right rules


$\bullet ; \bullet ; a \multimap b, b \multimap c \not\rightarrow^* \bullet ; a \multimap c$



- logic
- **system** ω
- rewriting
- processes
- security

System ω

- With cut, rule for \multimap can be simplified to
$$\Sigma; \Gamma; (\Delta, A, A \multimap B) \rightarrow \Sigma; \Gamma; (\Delta, B)$$
- 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^*



- logic
- **system ω**
- rewriting
- processes
- security

Discussion

- Other connectives?

- $\oplus, 0, \wp, \perp$
 - Odd rewrite properties
- $?, (_)^\perp$
 - 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?



- logic
- **system** ω
- rewriting
- processes
- security

Type Theoretic Side

- Very close to CLF

Concurrent Logical Framework

➤ Linear type theory with

- Dependent function types: Π
- Asynchronous connectives: \multimap , $\&$, \top
- Synchronous connectives: \otimes , 1 , $!$, \exists
- Monadic sandboxing
- Concurrency equations

(LF)

(LLF)

➤ Faithful encoding of true concurrency

- Petri nets, MSR 2 specs, π -calculus, concurrent ML

- Details of relation still unclear



- logic
- **system** ω
- rewriting
- processes
- security

Multiset Rewriting

- Multiset: set with repetitions allowed

$$\underline{a} ::= \bullet \mid a, \underline{a}$$

- 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



- logic
- system ω
- **rewriting**
- processes
- security

First-Order Multiset Rewriting

- Multiset elements are FO atomic formulas
- Rules have the form

$$\forall x_1 \dots x_n. \underline{a}(\mathbf{x}) \rightarrow \exists y_1 \dots y_k. \underline{b}(\mathbf{x}, \mathbf{y})$$

- Semantics

$$\Sigma ; \underline{a}(\mathbf{t}), \underline{s} \rightarrow_{R, (\underline{a}(\mathbf{x}) \rightarrow \exists \mathbf{y}. \underline{b}(\mathbf{x}, \mathbf{y}))} \Sigma, \mathbf{y} ; \underline{b}(\mathbf{t}, \mathbf{y}), \underline{s} \quad \text{if } \Sigma \vdash \mathbf{t}$$

- Several encodings into linear logic
 - [Martí-Oliet, Meseguer, 91]


- logic
- system ω
- rewriting
- processes
- security

ω -Multisets vs. Multiset Rewriting

- MSR 1 is an instance of ω -multisets
 - Uses only \otimes , 1 , \forall , \exists , and \multimap
 - \multimap never nested, always persistent

➤ iff
$$\begin{array}{ccc} \Sigma ; \underline{s} & \xrightarrow{R} & \Sigma' ; \underline{s}' \\ \Sigma ; "R" ; \underline{s} & \xrightarrow{*} & \Sigma' ; \underline{s}' \end{array}$$

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



- logic
- system ω
- rewriting
- processes
- security

ω -Rewriting

A, B	$::=$	a	atomic object
	$ $	1	empty
	$ $	$A \otimes B$	formation
	$ $	$A \multimap B$	rewrite
	$ $	T	no-op
	$ $	$A \& B$	choice
	$ $	$\forall x. A$	instantiation
	$ $	$\exists x. A$	generation
	$ $	$! A$	replication

- logic
- system ω
- **rewriting**
- processes
- security

The Asynchronous π -Calculus

Another fundamental model of distributed computing

- Language

$$P ::= 0 \mid P \parallel Q \mid \nu x. P \mid !P \mid x(y).P \mid \underline{x}\langle y \rangle$$


- Semantics

- Structural equivalence

- Comm. monoidal congruence of \parallel and 0
 - Binder mobility congruence of ν
 - $\nu x. \nu y. P \equiv \nu y. \nu x. P$
 - $0 \equiv \nu x. 0$
 - $P \parallel \nu x. Q \equiv \nu x. (P \parallel Q)$ if $x \notin \text{FN}(P)$
 - $!P \equiv !P \parallel P$

- Reaction law

- $\underline{x}\langle y \rangle \parallel x(z). P \parallel Q \Rightarrow [y/z]P \parallel Q$



- logic
- system ω
- rewriting
- **processes**
- security

π -calculus in ω -Multisets

- $0 \Leftrightarrow 1$
- $|| \Leftrightarrow \otimes$
- $\nu \Leftrightarrow \exists$
- $! \Leftrightarrow !$
- $x(y). P \Leftrightarrow \forall y. ch(x,y) \multimap "P"$
- $\underline{x}\langle y \rangle \Leftrightarrow ch(x,y)$

- Reaction law

➤ $\Sigma; \Gamma; ch(x,y), \forall z. ch(x,z) \multimap P, \Delta \rightarrow^2 \Sigma; \Gamma; [y/z]P, \Delta$


- 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 ω -multisets
- Oversight in the π -calculus?



- logic
- system ω
- rewriting
- processes
- security

Properties

- If $P \Rightarrow^* Q$
then $\bullet; \bullet; "P" \rightarrow^* \Sigma; \Gamma; \Delta$
where $"Q" = \exists \Sigma. !\Gamma \otimes \Delta \quad \text{mod } !A = !A \otimes A$


➤ Note: with $!P \rightarrow !P \parallel P$ as a transition

- If $P \Rightarrow^* Q$
then $\bullet; \bullet; "P" \rightarrow^* \Sigma; \Gamma; \Delta$
where $"Q" = \exists \Sigma. !\Gamma \otimes \Delta$

- logic
- system ω
- rewriting
- **processes**
- security

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



- logic
- system ω
- rewriting
- **processes**
- security

MSR 3

- Instance of ω -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



- logic
- system ω
- rewriting
- processes
- security

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 net
- Memory M_A
- Intruder I
- ...

Fully definable

- logic
- system ω
- rewriting
- processes
- security

Types

- Simple types

- $A : \text{princ}$
- $n : \text{nonce}$
- $m : \text{msg}, \dots$

- Dependent types

- $k : \text{shK } A \ B$
- $K : \text{pubK } A$
- $K' : \text{privK } K, \dots$

- S
u
b
s
o
r
t
i
n
g

Fully definable

- Powerful abstraction mechanism

- At various user-definable level
 - Finely tagged messages
 - Untyped: msg only

- Simplify specification and reasoning

- Automated type checking

- logic
- system ω
- rewriting
- processes
- **security**


Example

Needham-Schroeder public-key protocol

1. $A \rightarrow B: \{n_A, A\}_{k_B}$
2. $B \rightarrow A: \{n_A, n_B\}_{k_A}$
3. $A \rightarrow B: \{n_B\}_{k_B}$

- 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)



- logic
- system ω
- rewriting
- processes
- security

State-Based

$$\begin{aligned} A &\rightarrow B: \{n_A, A\}_{k_B} \\ B &\rightarrow A: \{n_A, n_B\}_{k_A} \\ A &\rightarrow B: \{n_B\}_{k_B} \end{aligned}$$

MSR 2 spec.

$\forall A: \text{princ.}$

$\{ \exists L: \text{princ} \times \Sigma B: \text{princ. pubK } B \times \text{nonce} \rightarrow \text{mset.}$

$\forall B: \text{princ. } \forall k_B: \text{pubK } B.$

•
 $\rightarrow \exists n_A: \text{nonce.}$

$\text{net} (\{n_A, A\}_{k_B}), L (A, B, k_B, n_A)$

$\forall B: \text{princ. } \forall k_B: \text{pubK } B.$

$\forall k_A: \text{pubK } A. \forall k_A': \text{prvK } k_A.$

$\forall n_A: \text{nonce. } \forall n_B: \text{nonce.}$

$\text{net} (\{n_A, n_B\}_{k_A}), L (A, B, k_B, n_A)$

$\rightarrow \text{net} (\{n_B\}_{k_B})$

}

Interpretation of L

- Rule invocation
 - Implementation detail
 - Control flow
- Local state of role
 - Explicit view
 - Important for DOS

- logic
- system ω
- rewriting
- processes
- security

Process-Based

$$\begin{aligned} A &\rightarrow B: \{n_A, A\}_{KB} \\ B &\rightarrow A: \{n_A, n_B\}_{KA} \\ A &\rightarrow B: \{n_B\}_{KB} \end{aligned}$$

$\forall A: \text{princ.}$

$\forall B: \text{princ. } \forall k_B: \text{pubK } B.$

• $\rightarrow \exists n_A: \text{nonce.}$

$\text{net } (\{n_A, A\}_{KB}),$

$(\forall k_A: \text{pubK } A. \forall k_A': \text{prvK } k_A. \forall n_B: \text{nonce.}$

$\text{net } (\{n_A, n_B\}_{KA}) \rightarrow \text{net } (\{n_B\}_{KB}))$

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

- logic
- system ω
- rewriting
- processes
- security

NSPK in Process Algebra

$$\begin{array}{l} A \rightarrow B: \{n_A, A\}_{KB} \\ B \rightarrow A: \{n_A, n_B\}_{KA} \\ A \rightarrow B: \{n_B\}_{KB} \end{array}$$

$\forall A: \text{princ.}$

$\forall B: \text{princ. } \forall k_B: \text{pubK } B.$

$\forall k_A: \text{pubK } A. \forall k_A': \text{prvK } k_A. \forall n_B: \text{nonce.}$

$\forall n_A: \text{nonce.}$

$\text{net } (\{n_A, A\}_{KB}).$

$\text{net } \langle \{n_A, n_B\}_{KA} \rangle.$

$\text{net } (\{n_B\}_{KB}). 0$

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-basedrepresentations of a problem

- logic
- system ω
- rewriting
- processes
- security

State-Based vs. Process-Based

- State-based languages

- Multiset Rewriting
- NRL Prot. Analyzer, CAPSL/CIL, Paulson's approach, ...

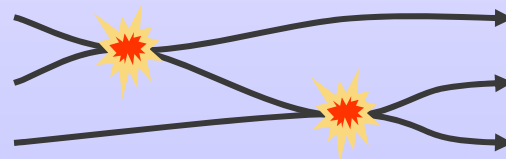
➤ State
transition
semantics



- Process-based languages

- Process Algebra
- Strand spaces, spi-calculus, ...

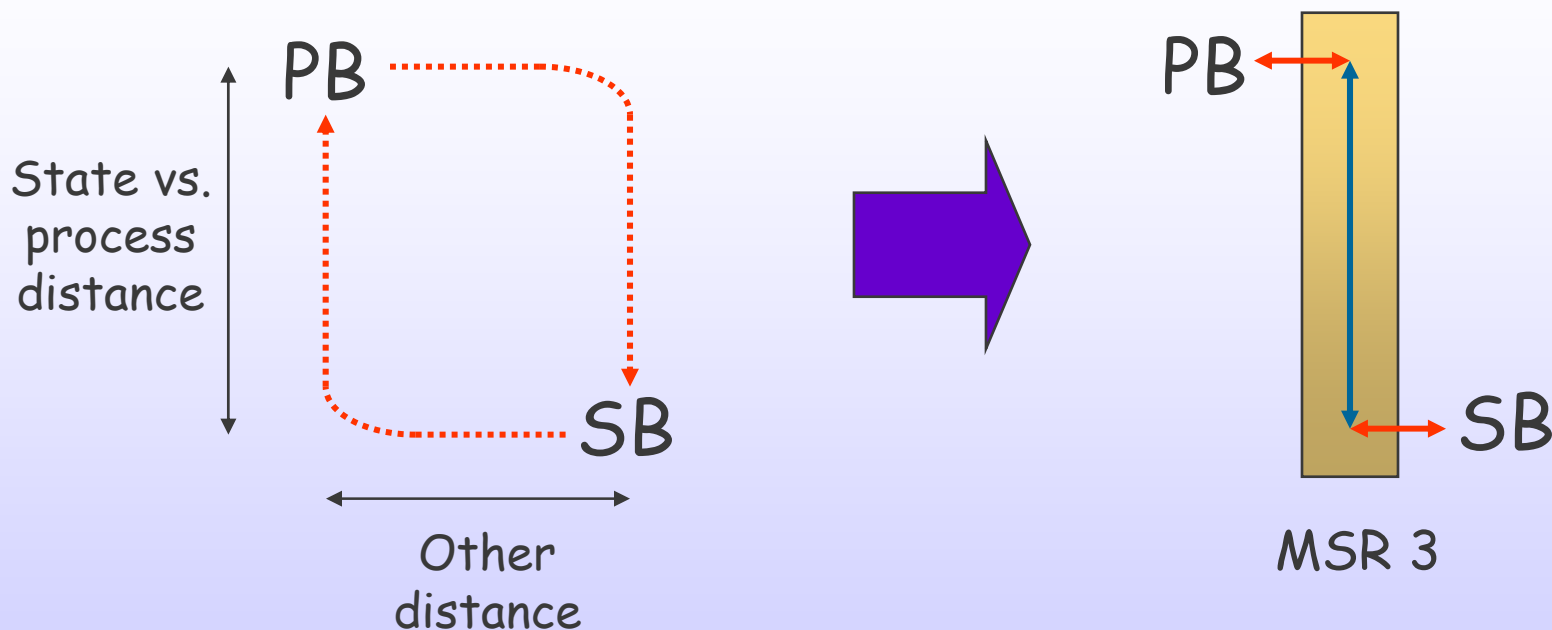
➤ Independent
communicating
threads



- logic
- system ω
- rewriting
- processes
- security

MSR 3 Bridges the Gap

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



State \leftrightarrow 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