MSR 3.0:

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

History of MSR

Not MicroSoft Research

- **MSR 1** [CSFW'99]
  - To formalize security protocols specification
  - First-order multiset rewriting with ∃
    - Undecidability of security protocol verification
    - Comparison with Strand Spaces

Creak!

- **MSR 2** [MMM'01]
  - Add typing infrastructure, liberalize syntax
    - Specification of Kerberos V
    - Completeness of Dolev-Yao attacker
    - Subsorting view of type-flaw attacks
    - Implementation (undergoing)
    - Comparison with Process Algebra

Hmm!

MSR 3

- From multisets to ω-multisets
  - Embeds multiset rewriting
    - MSR 1, 2
    - Paulson's inductive traces
    - Tool-specific languages
      - NRL Protocol Analyzer
      - Murφ, …
  - Encompasses Process Algebra
    - Strand spaces
    - Crypto-SPA
    - Spi-calculus
  - Founded on logic

... MSR 4 ?

- Give me 2 more years :-)
- MSR 3 is so general that it won't be needed
- … but I thought that of MSR 2 …
- Still lots of work on MSR 3

Rest of this Talk

- ω-multisets
  - Logical foundations
  - Relation to multiset rewriting
  - Relation to process algebra
- MSR 3
  - Typing
  - Example
  - State-based vs. process based representation

ω-Multisets

Instant recipe

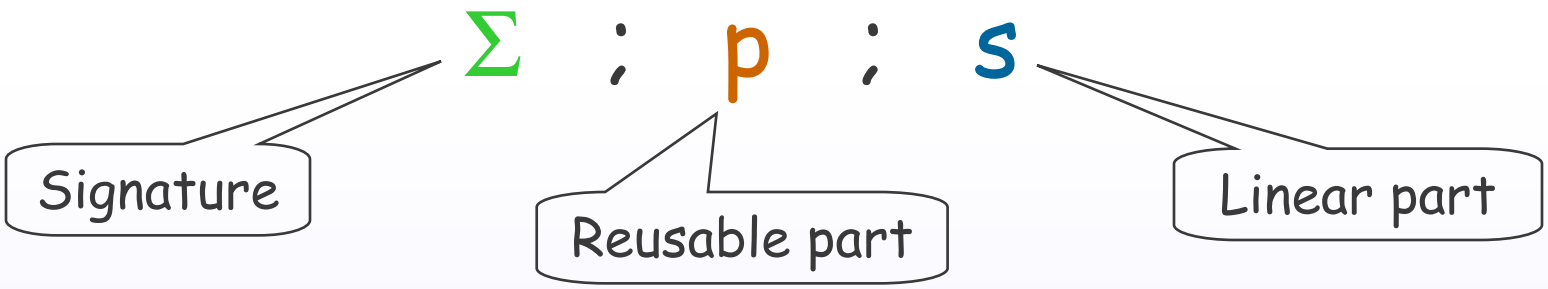
1. Take multiset rewriting
2. Fold it onto itself
3. Realize it is linear logic
4. Add more linear logic
5. Let simmer till your next presentation

- Specification language for concurrent systems
- Crossroad of
  - State transition languages
    - Petri nets, multiset rewriting, …
  - Process calculi
    - CCS, π-calculus, …
  - (Linear) logic

Syntax

ω	::=	a	atomic object
	|	1	empty
	|	$\omega \otimes \omega$	formation
	|	$\omega \multimap \omega$	rewrite
	|	$\top$	no-op
	|	$\omega \,\&\, \omega$	choice
	|	$\forall x.\ \omega$	instantiation
	|	$\exists x.\ \omega$	generation
	|	$!\,\omega$	replication

Semantics: Running State



- Constructor: ","
- Empty: "•"

- Σ is a list
- p and s are commutative monoids
- In s, we identify
  - , with ⊗
  - • with 1

ω-multiset
- logic
- rewriting
- processes

MSR 3

Target State

$$\Sigma \;;\; s$$

- Identified with $\exists x_1. \ldots \exists x_n. s$
  - For $\Sigma = x_1, \ldots, x_n$
- Mobility laws
  - $\exists x. \exists y. s = \exists y. \exists x. s$
  - $\exists x. \bullet = \bullet$
  - $\exists x. (s, s') = s, \exists x. s'$ if $x \notin FV(s)$
- In s, we still identify
  - , with $\otimes$ and $\bullet$ with 1

Judgments

- Base step

$$\Sigma ; p ; s \rightarrow \Sigma' ; p' ; s'$$

- Finite iteration

$$\Sigma ; p ; s \rightarrow^* \Sigma' ; s'$$

  - Reflexive and transitive closure of →

- Infinite iteration

$$\Sigma ; p ; s \rightarrow^*$$

  - Limit of _→*_

ω-Multisets: Semantics

1	$\Sigma ; p ; (s, 1) \rightarrow \Sigma ; p ; s$
$\otimes$	$\Sigma ; p ; (s, a \otimes b) \rightarrow \Sigma ; p ; (s, a, b)$
$\multimap$	$\Sigma ; p ; (s, s', a \multimap b) \rightarrow \Sigma ; p ; (s, b)$ if $\Sigma ; p ; s' \rightarrow^* \Sigma ; a$
$\top$	(no rule)
&	$\Sigma ; p ; (s, a_1 \,\&\, a_2) \rightarrow \Sigma ; p ; (s, a_i)$
$\forall$	$\Sigma ; p ; (s, \forall x.\, a) \rightarrow \Sigma ; p ; (s, [t/x]a)$ if $\Sigma \vdash t$
$\exists$	$\Sigma ; p ; (s, \exists x.\, a) \rightarrow (\Sigma, x) ; p ; (s, a)$

ω-multiset
- logic
- rewriting
- processes

MSR 3

ω-Multisets: Semantics (cont'd)

!	$\Sigma ; p ; (s, !a)$	$\rightarrow$	$\Sigma ; (p, a) ; s$
	$\Sigma ; (p, a) ; s$	$\rightarrow$	$\Sigma ; (p, a) ; (s, a)$

$$\Sigma ; p ; s, s' \rightarrow \Sigma ; p ; a, s \quad \text{if} \quad \Sigma ; p ; s' \rightarrow^* \Sigma ; a$$

$$\Sigma ; p ; s \rightarrow \Sigma ; p, a ; s \quad \text{if} \quad \Sigma ; p ; \bullet \rightarrow^* \Sigma ; a$$

$$\Sigma ; p ; s \rightarrow^* \Sigma ; s$$

$$\Sigma ; p ; s \rightarrow^* \Sigma'' ; s'' \quad \text{if} \quad \Sigma ; p ; s \rightarrow \Sigma' ; p' ; s' \quad \text{and} \quad \Sigma' ; p' ; s' \rightarrow^* \Sigma'' ; s''$$

ω-multiset
- logic
- rewriting
- processes

MSR 3

Logical Foundations

→ is exactly

- Left rules of linear logic + Cut rules

→* is

- Axiom rule + Transitive closure of →

- Which linear logic?
  - Pfenning's LV / Barber's DILL
  - Judgment:

$$\Gamma ; \Delta \dashrightarrow_{\Sigma} C$$

A Few Examples

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

$$\Sigma;\ p;\ (s, s', a \multimap b) \to \Sigma;\ p;\ (s, b) \quad \text{if } \Sigma;\ p;\ s' \to^* \Sigma;\ a$$

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

$$\Sigma;\ p;\ (s, \forall x.\ a) \to \Sigma;\ p;\ (s, [t/x]a) \quad \text{if } \Sigma \vdash t$$

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

$$\Sigma;\ p;\ (s, \exists x.\ a) \to (\Sigma, x);\ p;\ (s, a)$$

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

$$\Sigma;\ p;\ (s, !a) \to \Sigma;\ (p, a);\ s$$

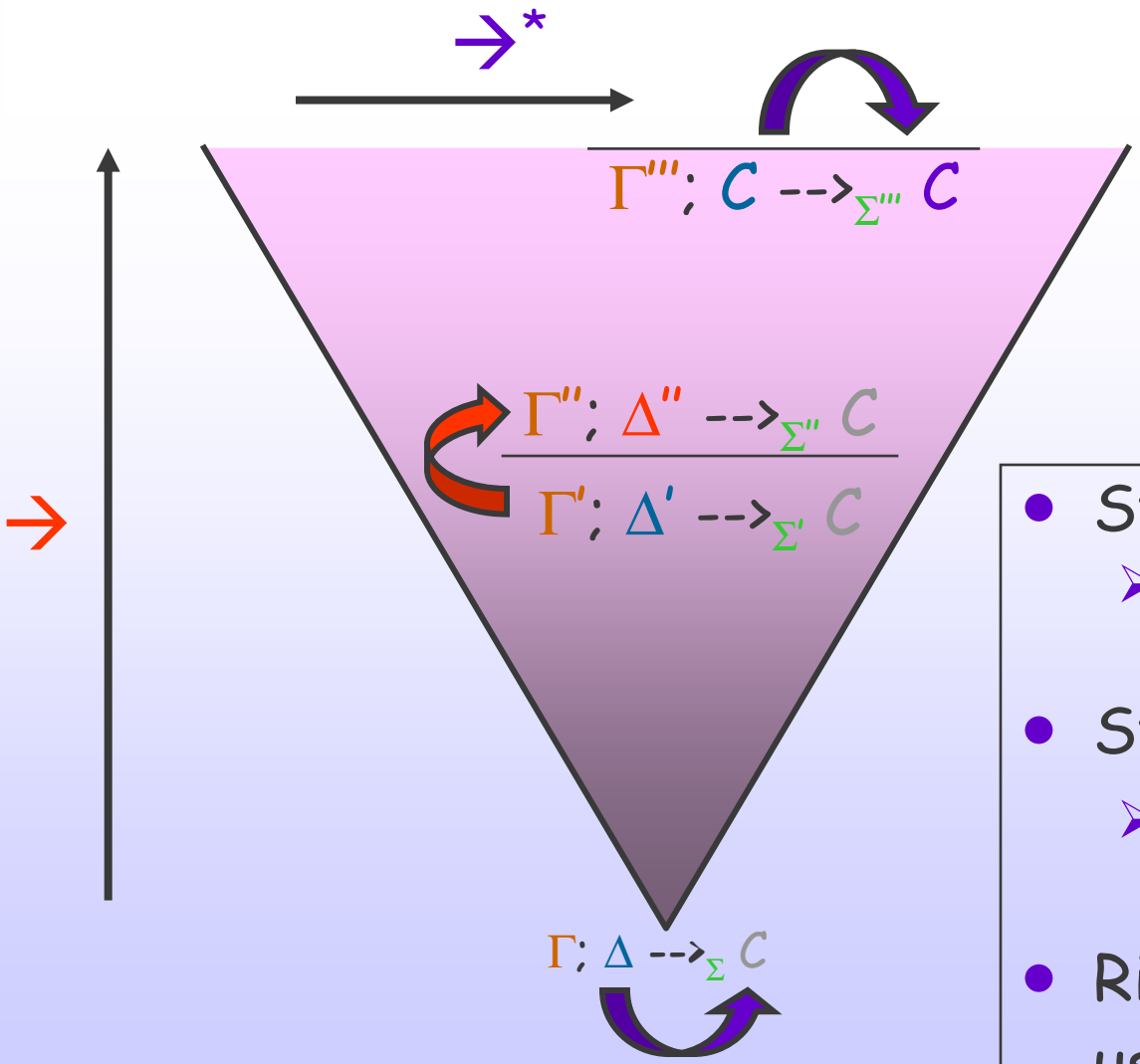
$$\frac{\Gamma;\ \Delta' \dashrightarrow_\Sigma A \qquad \Gamma;\ \Delta, A \dashrightarrow_\Sigma C}{\Gamma;\ \Delta, \Delta' \dashrightarrow_\Sigma C}$$

$$\Sigma;\ p;\ s, s' \to \Sigma;\ p;\ a, s \quad \text{if } \Sigma;\ p;\ s' \to^* \Sigma;\ a$$

ω-multiset
- logic
- rewriting
- processes

MSR 3

ω-Multiset View of Derivations



- Step up: →
  - Left rules
- Step across: →*
  - Axiom
- Right rules not used

ω-multiset
- logic
- rewriting
- processes

MSR 3

Formal Correspondence

- Soundness

  If $\Sigma ; p ; s \rightarrow^* \Sigma ; s'$

  then $p ; s \dashrightarrow_{\Sigma} \otimes s'$

- Notes
  - Monoidal laws of $\otimes$ are congruence modulo -->
  - Mobility laws of $\exists$ are equivalence modulo -->

- Completeness?
  - No! We have only crippled right rules

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

Notes on →*

- With cut, rule for —o can be simplified to

  Σ; p; (s, a, a —o b) → Σ; p; (s, b)

  (without minor premise)

- Cut elimination seems to hold
  - Turn subderivation into prefix
  - But …
    - Careful with extra signature symbols
    - Careful with extra persistent objects
  - Trivial without ! nor ∃

- No rule for → needs a premise
  - → does not depend on →*

Other Connectives ?

- Possibly, but need more work
- Looked into
  - ⊕, 0, ⅋, ⊥
  - Odd rewrite properties
- Not yet explored
  - ?, $(\_)^{\perp}$
- Beyond linear logic?

Type Theoretic Side

- Very close to CLF
  Concurrent Logical Framework
  - Linear type theory with
    - Dependent function types: Π (LF)
    - Asynchronous connectives: —o, &, ⊤ (LLF)
    - Synchronous connectives: ⊗, 1, !, ∃
    - Monadic sandboxing
    - Concurrency equations
  - Faithful encoding of true concurrency
    - Petri nets, MSR 2 specs, π-calculus, concurrent ML
- Details of relation still unclear

ω-multiset
- logic
- rewriting
- processes

MSR 3

Multiset Rewriting

- Multiset: set with repetitions allowed

  $a ::= \bullet \mid a, a$

  - 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

ω-multiset
- logic
- rewriting
- processes

MSR 3

First-Order Multiset Rewriting

a.k.a. MSR 1.0

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

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

- Semantics

$$\Sigma\ ;\ \underline{a}(\mathbf{t}),\ \underline{S} \rightarrow_{R,\ (\underline{a}(\mathbf{x}) \rightarrow \exists \mathbf{y}.\ \underline{b}(\mathbf{x},\mathbf{y}))} \Sigma,\mathbf{y}\ ;\ \underline{b}(\mathbf{t},\mathbf{y}),\ \underline{S}$$

if $\Sigma \vdash t$

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

ω-Multisets vs. Multiset Rewriting

- MSR 1 is an <u>instance</u> of ω-multisets
  - Uses only ⊗, 1, ∀, ∃, and —o
  - —o never nested, always persistent

➢ If $\Sigma ; s \rightarrow_R \Sigma' ; s'$
then $\Sigma ; R ; s \rightarrow^* \Sigma' ; s'$

- Interpretation of MSR <u>as</u> linear logic
  - ➢ Logical explanation of multiset rewriting
    - MSR is logic
  - ➢ Guideline to design rewrite systems

Compiling ω-Multisets to MSR 1

- Introduce intermediate tokens
  - Examples
    - a & b => (u → a), (u -> b) ; u
    - !a => (u → u, a) ; u
    - ∃x. a => (u → ∃x. a) ; u
    - ∀x. b => ∀x. (u → b) ; u
    - a —o (b —o c) => (a,u → v), (b,v → c) ; u
    - ...
- Must keep track of variables
  - Somewhat tricky (and tedious)

The Asynchronous π-Calculus

Another fundamental model of distributed computing

- Language

  P ::= 0 | P||Q | ν x. P | !P | x(y).P | $\underline{x}$<y>

- Semantics
  - Structural equivalence
    - Comm. monoidal congruence of || and 0
    - Binder mobility congruence of ν
      - 0 ≡ ν x. 0
      - P || ν x. Q ≡ ν x. (P || Q) if x ∉ FN(P)
      - ν x. ν y. P ≡ ν y. ν x. P
    - !P ≡ !P || P
  - Reaction law
    - $\underline{x}$<y> || x(z). P || Q ➔ [y/z]P || Q

ω-multiset
- logic
- rewriting
- processes

MSR 3

π-calculus in ω-Multisets

preliminary results

- 0 ⇔ 1
- || ⇔ ⊗
- ν ⇔ ∃
- ! ⇔ !
- x(y). P ⇔ ∀y. ch(x,y) —o "P"
- x<y> ⇔ ch(x,y)

- Reaction law
  - Σ; p; ch(x,y), ∀z. ch(x,z) —o P, s →² Σ; p; [y/z]P, s
- Structural equivalence
  - Monoidal congr. of || and 0 ⇔ monoidal congr. of ⊗ and 1
  - Mobility congr. of ν ⇔ mobility congr. of ∃
  - !P ≡ !P || P
    - Only ⇒ in ω-multisets
    - Oversight in the π-calculus?

ω-multiset
- logic
- rewriting
- processes

MSR 3

Properties

- If P ➔* Q

  then •; •; "P" →^{n} Σ; p; s for some n

  where "Q" = ∃Σ. !p ⊗ s mod !a = !a⊗a

  - Note: with !P → !P || P as a transition
    - If P ➔* Q

      then •; •; "P" →^{n} Σ; p; s for some n

      where "Q" = ∃Σ. !p ⊗ s

ω-multiset
- logic
- rewriting
- processes

MSR 3

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
  - Strand spaces
  - Synchronous π-calculus
  - Join calculus

} To do

ω-multiset
- logic
- rewriting
- processes

MSR 3

MSR 3.0

- Instance of ω-multisets for cryptographic protocol specification
  - Security-relevant signature
    - Network
    - Encryption, …
  - Typing infrastructure
    - Dependent types
    - Subsorting
  - Data Access Specification (DAS)
  - Module system
  - Equations

From MSR 1

From MSR 2

From MSR 2 implementation

ω-multiset

MSR 3
- typing
- example
- MSR<->PA

Messages

Atomic terms

- Principals — A
- Keys — K
  - Shared — K_{AB}
  - Public — K_A
  - Private — K'_A
- Nonces — N
- Other
  - Raw data — M
  - Timestamps — t
  - …

Constructors

- Encryption — $\{\_\}\_$
  - Symmetric — $\{\_\}\_$
  - Asymmetric — $\{\{\_\}\}\_$
- Pairing — $(\_, \_)$
- Other
  - Signature — $[\_]\_$
  - Hash — $h(\_)$
  - MAC — $h\_(\_)$
  - …

Fully definable

ω-multiset

MSR 3
- typing
- example
- MSR<->PA

Types

- Simple types
  - A : princ
  - n : nonce
  - m : msg, …
- Dependent types
  - k : shK A B
  - K : pubK A
  - K' : privK K, …

Fully definable

- Powerful abstraction mechanism
  - At various user-definable level
    - Finely tagged messages
    - Untyped: msg only
- Simplify specification and reasoning
- Automated type checking

Subsorting

$$\tau <: \tau'$$

- Allows atomic terms in messages
- Definable
  - Non-transmittable terms
  - Sub-hierarchies
- Discriminant for type-flaw attacks

Data Access Specification

- Prevent illegitimate use of information
  - Protocol specification divided in roles
    - Owner = principal executing the role
  - A signing/encrypting with B's key
  - A accessing B's private data, …
- Simple static check
- Central meta-theoretic notion
  - Detailed specification of Dolev-Yao access model
- Gives meaning to Dolev-Yao intruder
- Current effort towards integration in type system
  - Definable
    - Possibility of going beyond Dolev-Yao model

Modules and Equations

- Modules
  - Bundle declarations with simple import/export interface
  - Keep specifications tidy
  - Reusable
- Equations

  (For free from underlying Maude engine)
  - Specify useful algebraic properties
    - Associativity of pairs
  - Allow to go beyond free-algebra model

ω-multiset

MSR 3
- typing
- example
- MSR<->PA

Example: NSPK Initiator

$A \to B: \{N_A, A\}_{KB}$
$B \to A: \{N_A, N_B\}_{KA}$
$A \to B: \{N_B\}_{KB}$

$\forall A$:princ.
  $\forall B$: princ. $\forall K_B$: pubK B.
    • —o $\exists N_A$: nonce.
      net ($\{N_A, A\}_{KB}$),
      ($\forall K_A$: pubK A. $\forall K_A'$: prvK K_A. $\forall N_B$: nonce.
        net ($\{N_A, N_B\}_{KA}$) —o net ($\{N_B\}_{KB}$))

Compare with (untyped) process algebra

νN_A. net ($\{N_A, A\}_{KB}$). net<$\{N_A, N_B\}_{KA}$>. net ($\{N_B\}_{KB}$). 0

ω-multiset

MSR 3
- typing
- example
- MSR<->PA

NSPK in MSR 2.0

$A \rightarrow B: \{N_A, A\}_{KB}$
$B \rightarrow A: \{N_A, N_B\}_{KA}$
$A \rightarrow B: \{N_B\}_{KB}$

$\forall A$: princ.
{ $\exists L$: princ $\times \sum B$:princ.pubK B $\times$ nonce $\rightarrow$ mset.

$\forall B$: princ. $\forall K_B$: pubK B.
•
$\rightarrow \exists N_A$: nonce.
net ($\{N_A, A\}_{KB}$), L (A, B, K_B, N_A)

$\forall B$: princ. $\forall K_B$: pubK B.
$\forall K_A$: pubK A. $\forall K_A'$: prvK K_A.
$\forall N_A$: nonce. $\forall N_B$: nonce.
net ($\{N_A, N_B\}_{KA}$), L (A, B, K_B, N_A)
$\rightarrow$ net ($\{N_B\}_{KB}$)

}

ω-multiset

MSR 3
- typing
- example
- MSR<->PA

MSR vs. PA

Multiset Rewriting

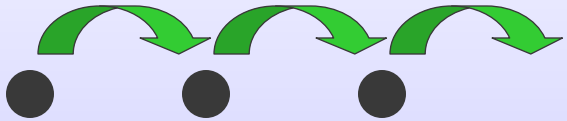
- NRL Prot. Analyzer, CAPSL/CIL, Paulson's approach, …

and Process Algebra

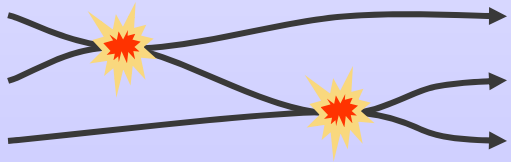
- Strand spaces, spi-calculus, other process-based lang.

operate in very different ways:

- State transitions

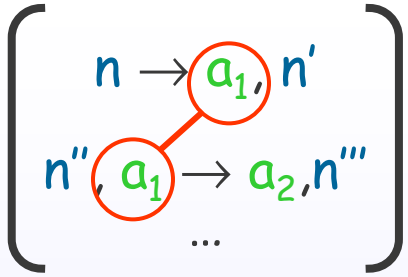


- Contact evolution



Representing Protocols

- MSR 1



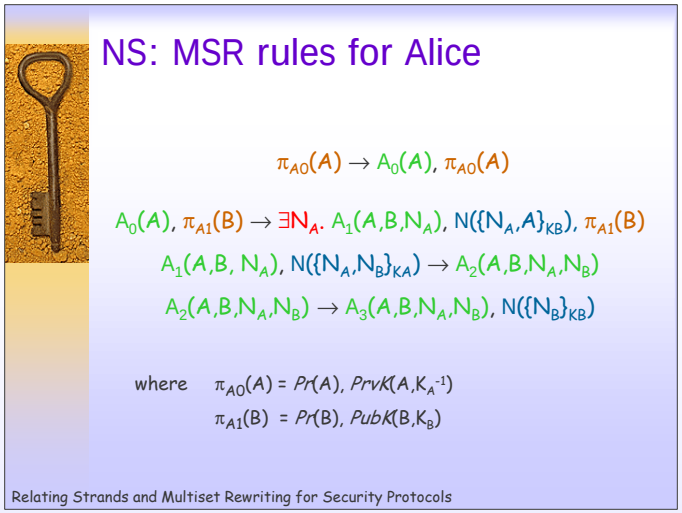
  - a_i pass control/data to the next rule

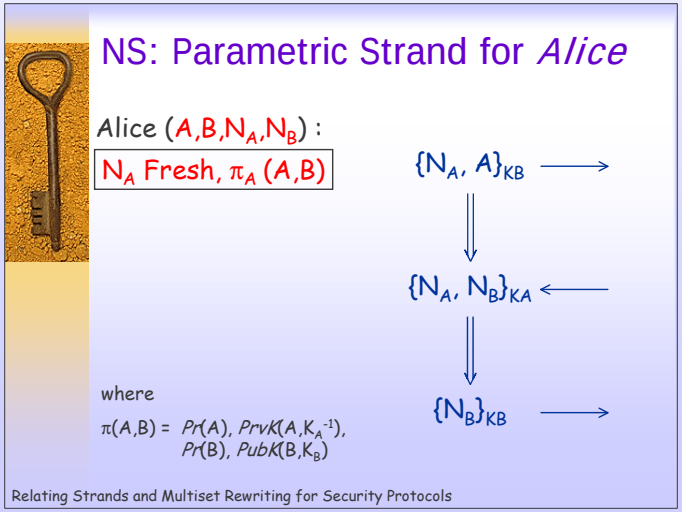
- PA $\quad n.\underline{n'}.n''.\underline{n'''}. \ldots .0$
- MSR 3 $\quad n \multimap n', (n'' \multimap n''', (\ldots))$

  - Control is implicit

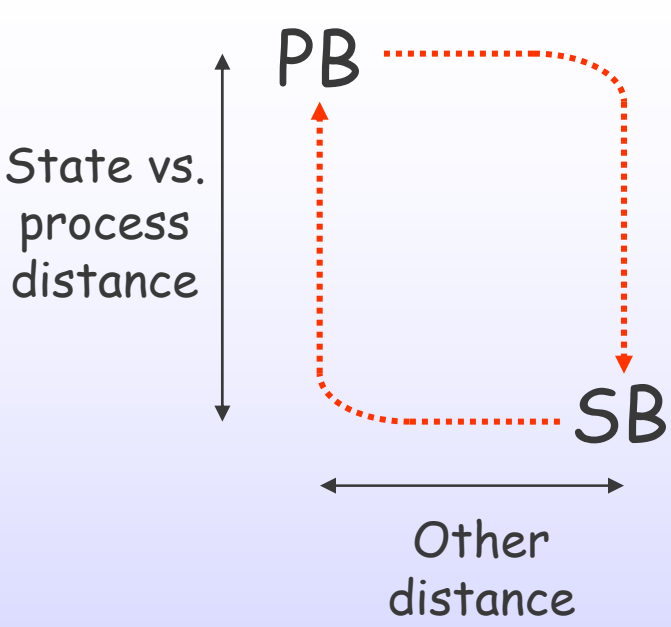
ω-multiset

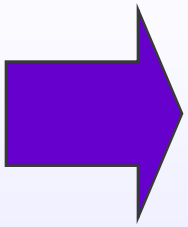
MSR 3
- typing
- example
- MSR<->PA

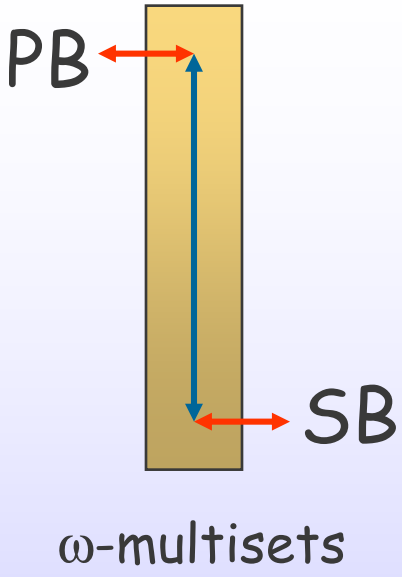




Encoding Distributed Algorithms







State ↔ Process translation done once and for all

ω-multiset

MSR 3
- typing
- example
- MSR<->PA

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