Relating the MSR Crypto-Protocol Specification Language to Rewriting Logic with Dependent Types

Iliano Cervesato **iliano@itd.nrl.navy.mil**

ITT Industries, inc @ NRL Washington, DC

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

Joint work with Mark-Oliver Stehr (UIUC)

Outline

- MSR
- Rewriting Logic with Dependent Types
- System Architecture
- Encoding details
  - Types and states
  - Roles and rules
  - Optimizations

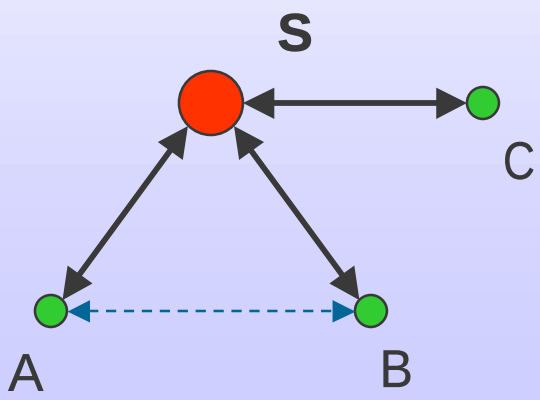
MSR

- Executable protocol specification language
  - Theoretical results
    - Decidability
    - Most powerful intruder, …
  - Practice
    - Kerberos V
    - Implementation underway
- Based on MultiSet Rewriting
  - Foundations in (linear) logic
  - Ties to Petri nets and process algebra
- MSR 2
  - Dependent types with subsorting
  - Very flexible

Example: Otway-Rees Protocol

1. A -> B: n A B $\{n_A$ n A B$\}_{KAS}$
2. B -> **S**: n A B $\{n_A$ n A B$\}_{KAS}$ $\{n_B$ n A B$\}_{KBS}$
3. **S** -> B: n $\{n_A$ $k_{AB}\}_{KAS}$ $\{n_B$ $k_{AB}\}_{KBS}$
4. B -> A: n $\{n_A$ $k_{AB}\}_{KAS}$

- A, B, C, … have keys to **S**
- A and B want to talk
- Use **S** to get common key
  - Key distribution
  - Authentication



MSR Spec.

1. A -> B: n A B $\{n_A\ n\ A\ B\}_{KAS}$
2. B -> **S**: n A B $\{n_A\ n\ A\ B\}_{KAS}$ $\{n_B\ n\ A\ B\}_{KBS}$
3. **S** -> B: n $\{n_A\ k_{AB}\}_{KAS}$ $\{n_B\ k_{AB}\}_{KBS}$
4. B -> A: n $\{n_A\ k_{AB}\}_{KAS}$

- Types
  - Subsorting

```
msg, princ, nonce: type.
shK, stK, ltK: princ -> princ -> type.
       princ, nonce, stK A B <: msg.
       stK A B, ltK A B <: shK A B.
```

- Constructors

```
_ _: msg -> msg -> msg.
{_}_: msg -> shK A B -> msg.
S : princ.
```

- Predicates

```
N: msg -> state.
```

- Roles for
  - **S**
  - A, B

Next slide

- Principals and keys

```
...
```

B's Role

1. A -> B: n A B X
2. B -> S: n A B X $\{n_B\ n\ A\ B\}_{KBS}$
3. S -> B: n Y $\{n_B\ k_{AB}\}_{KBS}$
4. B -> A: n Y

$\forall$B:princ.

$\exists$L:ΠB:princ. nonce * nonce * ltK B **S** -> state.

$\forall$A:princ. $\forall$n:nonce. $\forall k_{BS}$:ltK B **S**. $\forall$X:msg.
N(n A B X) ➔ $\exists n_B$:nonce.
N(n A B X $\{n_b\ n\ A\ B\}k_{BS}$),
L(A, B, n, n_B, k_{BS})

$\forall$A:princ. $\forall$n, n_B:nonce. $\forall k_{BS}$:ltK B **S**.
$\forall$Y:msg. $\forall k_{AB}$:stK A B.
N (n Y $\{n_B\ k_{AB}\}_{kBS}$),
L(A, B, n, n_B, k_{BS}) ➔ N (n Y)

Main Features of MSR

- Open signatures
- Multiset rewriting
  - Msets of F.O. formulas
  - Rules
    - $\forall$(LHS $\rightarrow$ $\exists \underline{n:\tau}$. RHS)
    - Existentials
  - Roles
    - $\forall A. \exists \underline{L:\tau}. \underline{r}$
- Types
  - Possibly dependent
  - Subsorting
  - Type reconstruction
- More
  - Constraints
  - Modules
  - Equations
- Static checks
  - Type checking
  - Data access spec.
- Execution

Black = implemented
Brown = work-around
Red = future work

Rewriting Logic with Dep. Types

- Combination of methodologies
  - Conditional rewriting modulo equations
    - $\forall x{:}S.\ A = B$ if C (generalizes equational logic)
    - $\forall x{:}S.\ A \Rightarrow B$ if C (generalizes rewriting logic)
  - Dependent type theory
    - $\lambda x{:}S.\ M : \Pi x{:}S\ T$ (generalizes simple types)

  Fragment of Open Calculus of Constructions
- Features
  - Open computation system
  - Proposition-as-types interpretation
    - $\forall x{:}S.\ P(x)$ interpreted as $\Pi x{:}S.\ P(x)$
      - Expressive higher-order logic
  - Model-theoretic semantics

Implementation of RWLDT

- Operational semantics
  - Conditional rewriting modulo equations
  - Automatic proof-search for conditions
  - Uniform for execution and type checking
  - Executable sublanguage
    - Equational logic
    - Rewriting logic
- OCC prototype in Maude
  - Mapping of HO logic to FO concepts

Example: Commutative Monoid

```
state: Type.

empty: state.
union: state -> state -> state.

state_comm: || {s1,s2 : state}
   (union s1 s2) = (union s2 s1).
state_assoc: || {s1,s2,s3 : state}
   (union s1 (union s2 s3)) = (union s1 (union s2 s3)).
state_id: || {s : state}
   (union s empty) = s.
```

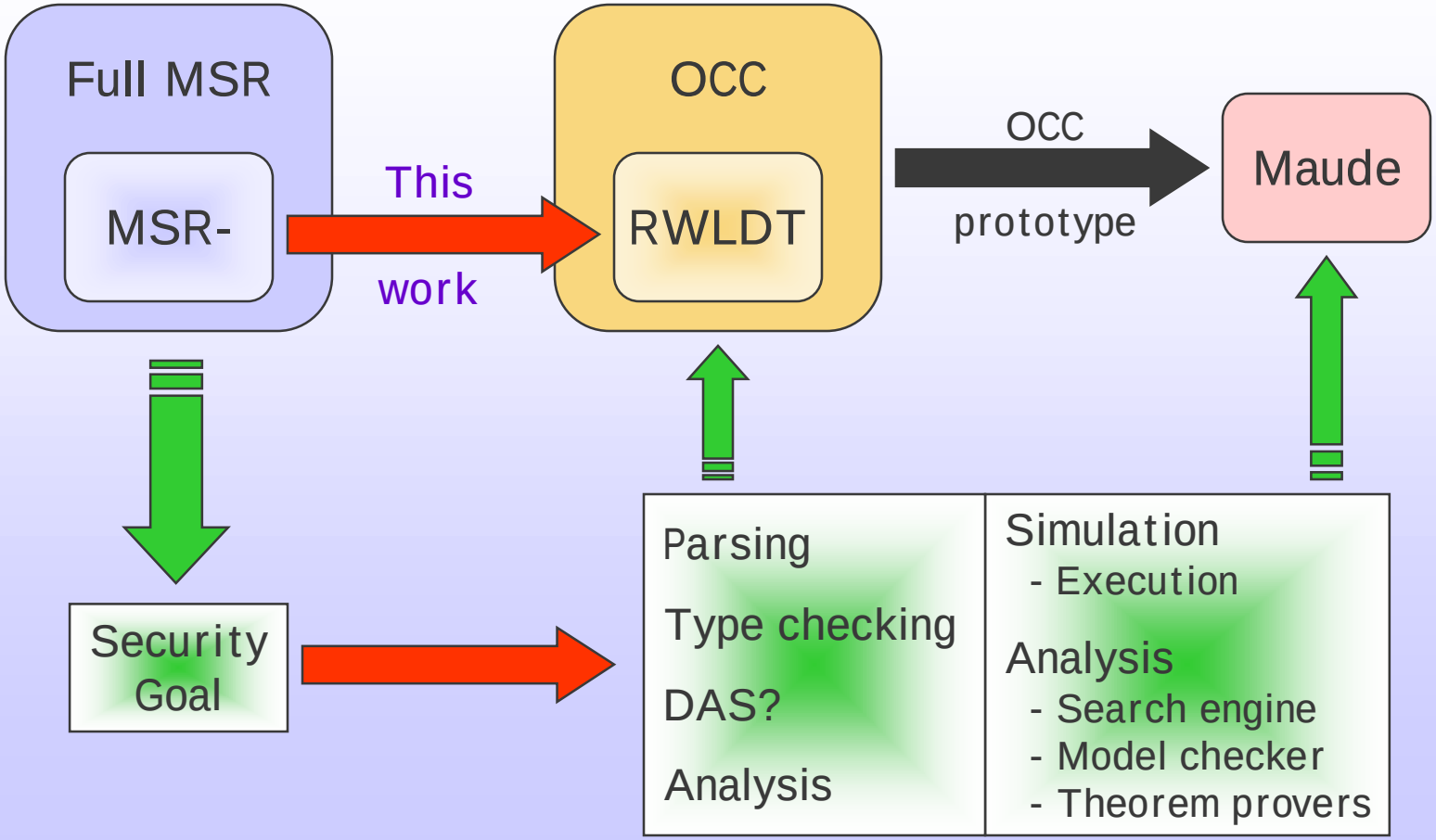
Structural equality

Π s:state. …

- This will be MSR's state

System Architecture

- Objective: implementation of MSR



Advantages over MSR → Maude

- Separation of concerns
  - MSR -> RWLDT
    - Preserves terms and types
    - Maps operations
  - RWLDT : type checking
  - Maude: untyped execution
- Abstraction
  - MSR and RWLDT have similar types and terms
  - Emulate MSR execution in RWLDT
  - Shallow encoding
- Reasoning
  - Express verification tasks in OCC [future work]

Full MSR ➔ MSR-

- Work-arounds
  - Modules
    - In-lining
  - Subtyping
    - Coercions
  - Type reconstruction
    - Inferable parts made explicit
- Omissions
  - Constraints
  - Data Access Specification
- Additions
  - Equations

Pre-processing

Future work

Encoding Strategy

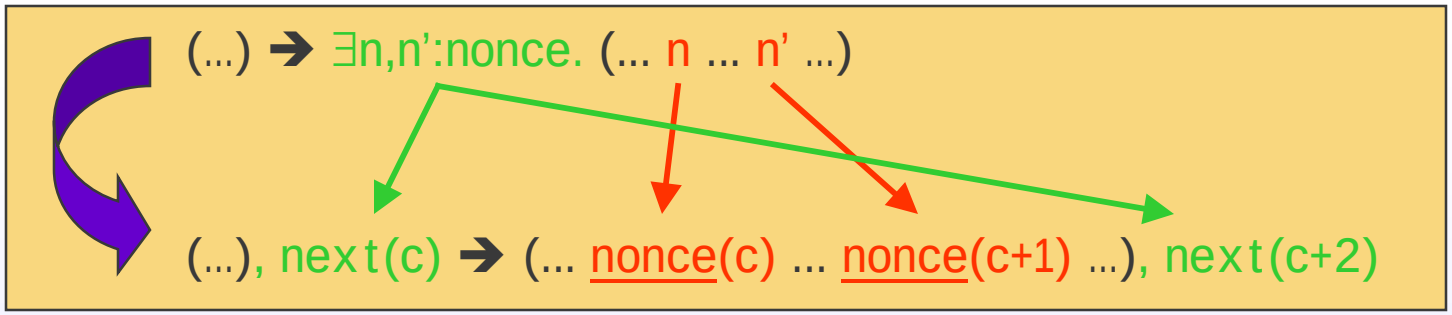
- Types and terms
  - Homomorphic mapping
    - Subsorting via coercions
- States
  - RWLDT terms
- Roles
  - Add 1 RWLDT rewrite axiom for role instantiation
  - Simulate ∃ using counters
- Rules
  - Mapped to RWLDT rewrite axioms
    - Simulate ∃ using counters

Optimizations

- Reduce non-determinism

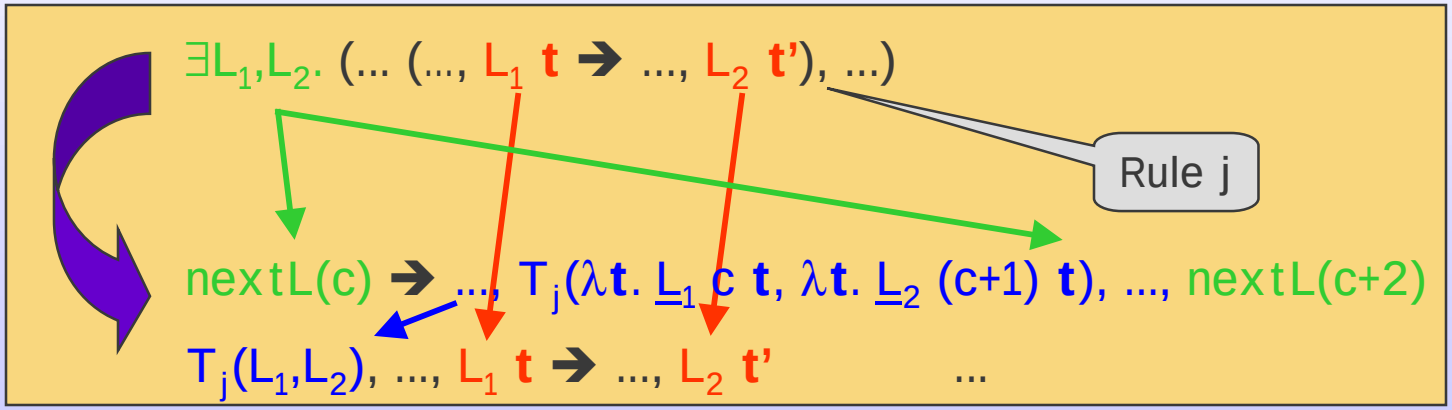
Representing Fresh Objects

- In rules



➢ nonce : nat -> nonce is an injection

- In roles



➢ L_i : nat -> τ_i -> state are injections

(done using conditional rewriting)

Representing Roles

$\forall A{:}princ.\ \exists Ls.\ (lhs_1 \rightarrow rhs_1, \ldots, lhs_n \rightarrow rhs_n)$

$princ(A), nextL(c) \rightarrow T_1(A,Ls), \ldots, T_n(A,Ls), princ(A), nextL(c')$

$T_1(A,Ls), lhs_1 \rightarrow rhs_1$

...

$T_n(A,Ls), lhs_n \rightarrow rhs_n$

Enhancement

- Force rule application upon activation
  - $princ(A), nextL(c), lhs_i \rightarrow T_1(A,Ls), \ldots, rhs_i \ldots, T_n(A,Ls), princ(A), nextL(c')$
  - $T_i(A,Ls), lhs_i \rightarrow rhs_i$

Representing Rules

$\forall x{:}\tau.$ lhs ➔ rhs

$\underline{\tau}(x)$, ..., ..., lhs ➔ $\underline{\tau}(x)$, ..., rhs

- Handles x's occurring only in rhs
  - Allows encoding to untyped rewrite systems
  - Types τ must be finite and enumerated in state
- Enhancement
  - Limit to x's occurring only on rhs

Optimizations

- Use single counter
  - ∀A. ∃L. (lhs ➔ ∃n. rhs)
- Minimal control-flow analysis
  - Trace uses of L's
  - Do not generate unreachable rules
    - T's often duplicates L's

Substantial code reduction

- Could be further improved

Otway-Rees (1)

1. A -> B: n A B X
2. B -> S: n A B X $\{n_B\ n\ A\ B\}_{KBS}$
3. S -> B: n Y $\{n_B\ k_{AB}\}_{KBS}$
4. B -> A: n Y

```
<Initial context>
<Declarations for types and terms>
<Axioms for A>

(LB : nat ->
     ({B : princ} princ -> nonce -> nonce -> (ltK B S) -> state))
(TB1: princ -> princ ->
     ({B:princ} princ -> nonce -> nonce -> (ltK B S) -> state) -> state)
(TB2: princ -> princ ->
     ({B:princ} princ -> nonce -> nonce -> (ltK B S) -> state) -> state)

( B11 : ... )
( B12 : ... )
( B21 : ... )
( B22 : ... )

<Axioms for S>
```

Optimized away

Otway-Rees (2)

1. A -> B: n A B X
2. B -> **S**: n A B X $\{n_B\ n\ A\ B\}_{KBS}$
3. **S** -> B: n Y $\{n_B\ k_{AB}\}_{KBS}$
4. B -> A: n Y

```
B11 : !! {B : princ}
 {L : {B : princ} princ -> nonce -> nonce -> (ltK B S) -> state}
 {A : princ}{kBS : (ltK B S)}{X : msg}
 {fresh,fresh' : nat} {n,nB : nonce}

 (nB := (NONCE fresh)) -> (L := (LB (suc fresh))) ->
 (fresh' := (suc (suc fresh))) ->

 [LB11]: (union (EL (ltK B S) kBS) (union (F fresh) (union (START-2 B)
                (N (append (nonce-msg n) (append (princ-msg A)
                   (append (princ-msg B) X)))))))
      =>
         (union (EL (ltK B S) kBS) (union (F fresh')
         (union (N (append (nonce-msg n) (append (princ-msg A)
                   (append (princ-msg B)
                   (append X (encrypt B S (append (nonce-msg nB)
                                          (append (nonce-msg n)
                                          (append (princ-msg A)
                                                  (princ-msg B))))
                                    (ltK-shK B S kBS)))))))
         (union (L B A n nB kBS) (TB2 A B L)))))
```

Otway-Rees (3)

1. A -> B: n A B X
2. B -> **S**: n A B X $\{n_B\ n\ A\ B\}_{KBS}$
3. **S** -> B: n Y $\{n_B\ k_{AB}\}_{KBS}$
4. B -> A: n Y

```
B22 : !! {B : princ}
 {L : {B : princ} princ -> nonce -> nonce -> (ltK B S) -> state}
 {A : princ}{kAB : (stK A B)}{kBS : (ltK B S)}{Y : msg}
 {n,nB : nonce}

 [LB22]: (union (N (append (nonce-msg n)
                           (append Y (encrypt B S (append (nonce-msg nB)
                                                          (stK-msg A B kAB))
                                                  (ltK-shK B S kBS)))))
                (union (L B A n nB kBS) (TB2 A B L)))
      =>
         (union (N (append (nonce-msg n) Y)) (TERMINATED-2 B))
```

Execution

- Encoding typechecks in OCC
- Executes on top of Maude

```
A:princ . B:princ . kAS:(ltK A S) . kBS:(ltK B S) .

rew (union ((F 0),
            (E P A), (E P B), (E (ltK A S) kAS), (E (ltK B S) kBS),
            (START1 A), (START2 B), (START3 S))) .
trace:
    LA11 LB11 LS11 LB22 LA22
result:
    (union ((F 6),
            (E P A), (E P B), (E (ltK A S) kAS), (E (ltK B S) kBS),
            (TERMINATED1 A), (TERMINATED2 B), (TERMINATED3 S)))
```

Summary

Encoding of MSR to RWLDT

- Captures most of MSR
- Shallow embedding
  - Type-preserving to OCC
- Proof of adequacy
  - Soundness
  - Completeness
- Executable on Maude engine
  - Optimized
- Preliminary experiments
  - **http://formal.cs.uiuc.edu/stehr/msr.html**
- Forthcoming implementation

Future Work

- Extensions
  - Constraints
  - Equations
  - Data access specification
- Implementation
  - Experimentation
  - Fine-tuning of optimizations
  - Verification tasks
- Longer-term
  - MSR 3
    - embedded rules and more

Monitor web site

- **http://formal.cs.uiuc.edu/stehr/msr.html**