Maude Implementation of MSR

Mark-Oliver Stehr
Stefan Reich
University of Illinois, Urbana-Champaign

(Iliano Cervesato)
ITT Industries @ NRL

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

Protocol eXchange - UMBC
What the customer explained

What the project manager understood

What the analyst designed

What the programmer delivered

What the consultant defined

What was documented

What was installed

What the client was charged

How it was maintained

What the client needed

From http://muetze.net/links/fun/kundenprojekte-e.html
Big Picture

- **MSR**
  - Protocol specification language
  - Multiset rewriting
  - Dependent types
  - Existentials

- **Maude**
  - Flexible specification framework
  - Rewriting logic
  - Equational reasoning
  - Reflection

Protocol specs.  
Security goals
Implemented Architecture

- MSR
- MSR-
- OCC
- RWLDT
- Maude
- Security Goal
- Parsing
- Type checking
- DAS?
- Simulation
  - Execution
- Analysis
  - Search engine
  - Model checker
  - Theorem provers

This work
Bestiary

- **MSR-**
  - MSR (2) with some restrictions

- **RWLDT**
  - Rewriting Logic with Dependent Types
  - Typed version of Maude

- **OCC**
  - Open Calculus of Constructions
  - Mark-Oliver’s thesis (589 pages)
  - Prototype implemented in Maude
Advantages over MSR → Maude

• Separation of concerns
  - MSR → RWLDT
    - Preserves terms and types
    - Maps operations
  - RWLDT: takes care of 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]
Small changes to simplify encoding

- **Work-arounds**
  - Subtyping
    - Coercions
- **Omissions**
  - Data Access Specification
- **Additions**
  - Equations

- Emulated via pre-processing
- Future work
- Beta version
Supported Operations

- Parsing for MSR-
  - Minor limitations (currently worked on)
- Type reconstruction
  - Rule-level missing (currently worked on)
- Type checking
- Simulation
  - Indirect via OCC (currently worked on)
    - search \([n]\) (goal)
    - rew \([n]\) (goal)
    - choose \(n\)
Example: Otway-Rees Protocol

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

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

- **Types**
  - Subsorting
  - Constructors

- **Predicates**

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

- **Principals and keys**

---

1. $A \rightarrow B$: $n \ A \ B \{n_A \ n \ A \ B\}^{KAS}$
2. $B \rightarrow S$: $n \ A \ B \{n_A \ n \ A \ B\}^{KAS} \{n_B \ n \ A \ B\}^{KBS}$
3. $S \rightarrow B$: $n \{n_A \ k_{AB}\}^{KAS} \{n_B \ k_{AB}\}^{KBS}$
4. $B \rightarrow A$: $n \{n_A \ k_{AB}\}^{KAS}$

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

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

...
**B’s Role**

\[ \forall B:\text{princ.} \]
\[ \exists L:\Pi B:\text{princ.} \text{ nonce} \ast \text{ nonce} \ast \text{ ltK} B S \to \text{ state.} \]

\[ \left\{ \begin{array}{l}
\forall A:\text{princ.} \forall n:\text{nonce.} \forall k_{BS}:\text{ltK} B S. \forall X:\text{msg.} \\
N(n A B X) \implies \exists n_B:\text{nonce.} \\
\quad N(n A B X \{n_B n A B\}k_{BS}), \\
\quad L(A, B, n, n_B, k_{BS}) \\
\end{array} \right. \]

\[ \left\{ \begin{array}{l}
\forall A:\text{princ.} \forall n, n_B:\text{nonce.} \forall k_{BS}:\text{ltK} B S. \\
\forall Y:\text{msg.} \forall k_{AB}:\text{stK} A B. \\
N(n Y \{n_B k_{AB}\}k_{BS}), \\
L(A, B, n, n_B, k_{BS}) \implies N(n Y) \\
\end{array} \right. \]
Main Features of MSR

• **Open signatures**

• **Multiset rewriting**
  - Msets of F.O. formulas
  - Rules
    \[ \forall (LHS \rightarrow \exists n: \tau. \text{RHS}) \]
    - Existentials
  - Roles
    \[ \forall \text{A}. \exists \text{L}: \tau. \text{r} \]

• **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
    - \( \forall x:S. A = B \text{ if } C \) (generalizes equational logic)
    - \( \forall x:S. A \Rightarrow B \text{ if } C \) (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
Example: Commutative Monoid

\begin{verbatim}
state: Type.
empty: state.
union: state -> state -> state.

state_comm: || \{s_1, s_2 : state\}
            (union s_1 s_2) = (union s_2 s_1).

state_assoc: || \{s_1, s_2, s_3 : state\}
             (union s_1 (union s_2 s_3)) = (union s_1 (union s_2 s_3)).

state_id: || \{s : state\}
         (union s empty) = s.
\end{verbatim}

- This implements MSR’s state
Encoding Strategy

• Types and terms
  ➢ Homomorphic mapping
    ▪ Subsorting via coercions

• States
  ➢ RWLDT terms

• Roles
  ➢ Add 1 RWLDT rewrite axiom for role instantiation
  ➢ Simulate $\exists$ using counters

• Rules
  ➢ Mapped to RWLDT rewrite axioms
    ▪ Simulate $\exists$ using counters

Optimizations [not implemented]
  ➢ Reduce non-determinism
Representing Fresh Objects

- **In rules**

  \[(...) \Rightarrow \exists n, n':\text{nonce.} \ (\ldots \ n \ldots \ n' \ldots)\]

  \[(\ldots), \text{next}(c) \Rightarrow (\ldots \ \text{nonce}(c) \ldots \ \text{nonce}(c+1) \ldots), \ \text{next}(c+2)\]

  - **nonce : nat -> nonce** is an injection

- **In roles**

  \[\exists L_1, L_2. \ (\ldots \ (\ldots, \ \lambda t. \ L_1 \ t \Rightarrow \ldots, \ \lambda t. \ L_2 \ (c+1) \ t), \ldots, \ \text{nextL}(c+2)\]

  \[\text{nextL}(c) \Rightarrow \ldots, \ T_j(\lambda t. \ L_1 \ c \ t, \ \lambda t. \ L_2 \ (c+1) \ t), \ldots, \ \text{nextL}(c+2)\]

  \[T_j(L_1, L_2), \ldots, \ L_1 \ t \Rightarrow \ldots, \ L_2 \ t' \ldots \]

  - **L_i : nat -> \tau_i -> state** are injections
Representing Roles

∀A:princ. ∃Ls. (lhs₁ \rightarrow rhs₁, ..., lhsₙ \rightarrow rhsₙ)

princ(A), nextL(c) \rightarrow T₁(A,Ls), ..., Tₙ(A,Ls), princ(A), nextL(c')
T₁(A,Ls), lhs₁ \rightarrow rhs₁
...
Tₙ(A,Ls), lhsₙ \rightarrow rhsₙ

Enhancement

• Force rule application upon activation
  ➢ princ(A), nextL(c), lhs_i \rightarrow T₁(A,Ls), ..., rhs_i ..., Tₙ(A,Ls), princ(A), nextL(c')
  ➢ T_i(A,Ls), lhs_i \rightarrow rhs_i
Representing Rules

\[ \forall x: \tau. \text{lhs} \rightarrow \text{rhs} \]

\[ \tau(x), ..., ..., \text{lhs} \rightarrow \tau(x), ..., \text{rhs} \]

- Handles \( x \)'s occurring only in \( \text{rhs} \)
  - Allows encoding to untyped rewrite systems
  - Types \( \tau \) must be finite and enumerated in state
- Enhancement
  - Limit to \( x \)'s occurring only on \( \text{rhs} \)
Optimizations [not implemented]

• 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
Trivia

- **Versions**
  - Alpha (this)
    - Partial reconstruction
    - Non-integrated search (exit MSR; call OCC)
    - No equations
    - Not-so-pretty-printing
  - Beta (mid-October - already working, mostly)

- **Space and Time**
  - 3,700 lines of Maude (1,300 for testing)
  - 6 months designing, 3 months coding

- **Examples**
  - Otway-Rees
  - Needham-Schroeder PK
  - Kerberos (abstract, full, cross-realm - soon)
  - ... more soon ...
Wanna Play?

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

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

• Download
  ➢ Currently alpha-release
  ➢ Soon beta-release

• Papers

• News
Future Work

- **Short-term**
  - Complete beta-released
  - Get degree (Stefan)

- **Medium term - language**
  - Library of protocols
  - Data Access Specification
  - MSR 3
    - Embedded rules and more

- **Medium/long-term - Verification**
  - Implement various methodologies
  - MSR as verification middleware
Demo Time!!!