Maude Implementation of MSR

Mark-Oliver Stehr

Stefan Reich

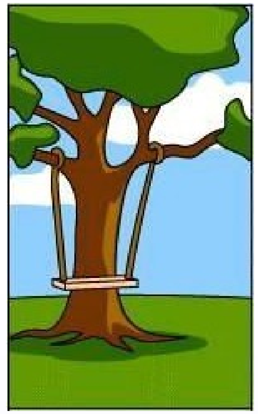
University of Illinois, Urbana-Champaign

(Iliano Cervesato)

ITT Industries ~~@ NRL~~

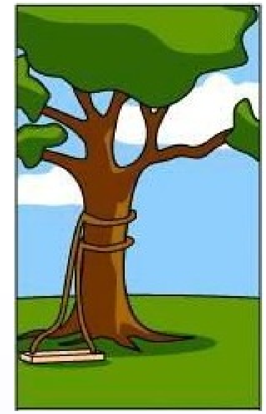
http://theory.stanford.edu/~iliano/

What the customer explained



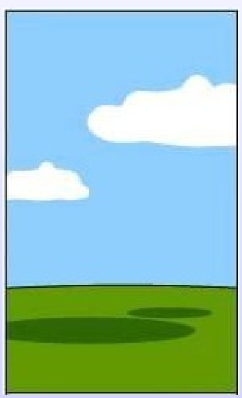
What the project manager understood

What the analyst designed

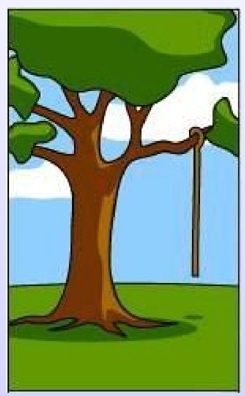


What the programmer delivered

What the consultant defined

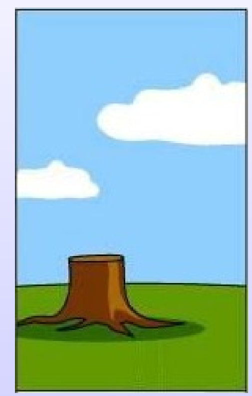


What was documented



What was installed

What the client was charged



How it was maintained

What the customer needed

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

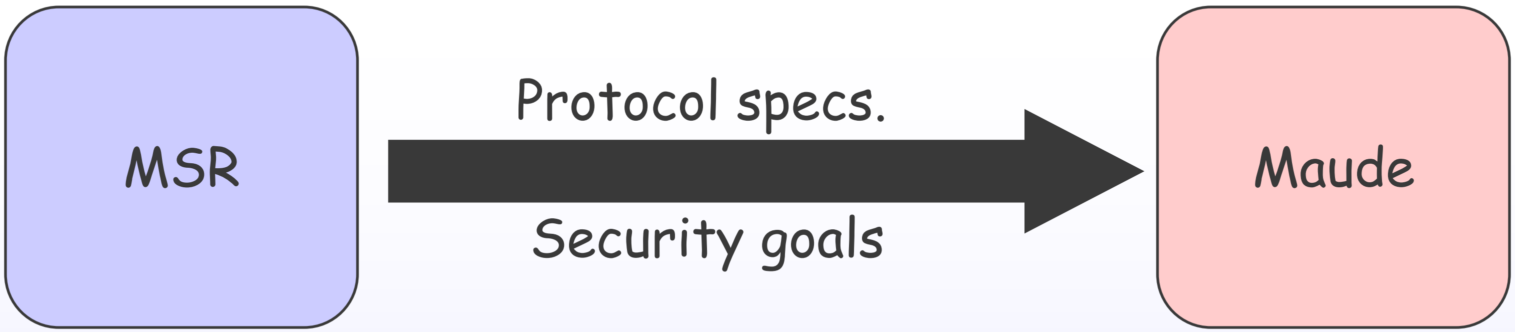
Project Objectives

MSR

- Uncommitted specification language
- Tabula rasa w.r.t. verification

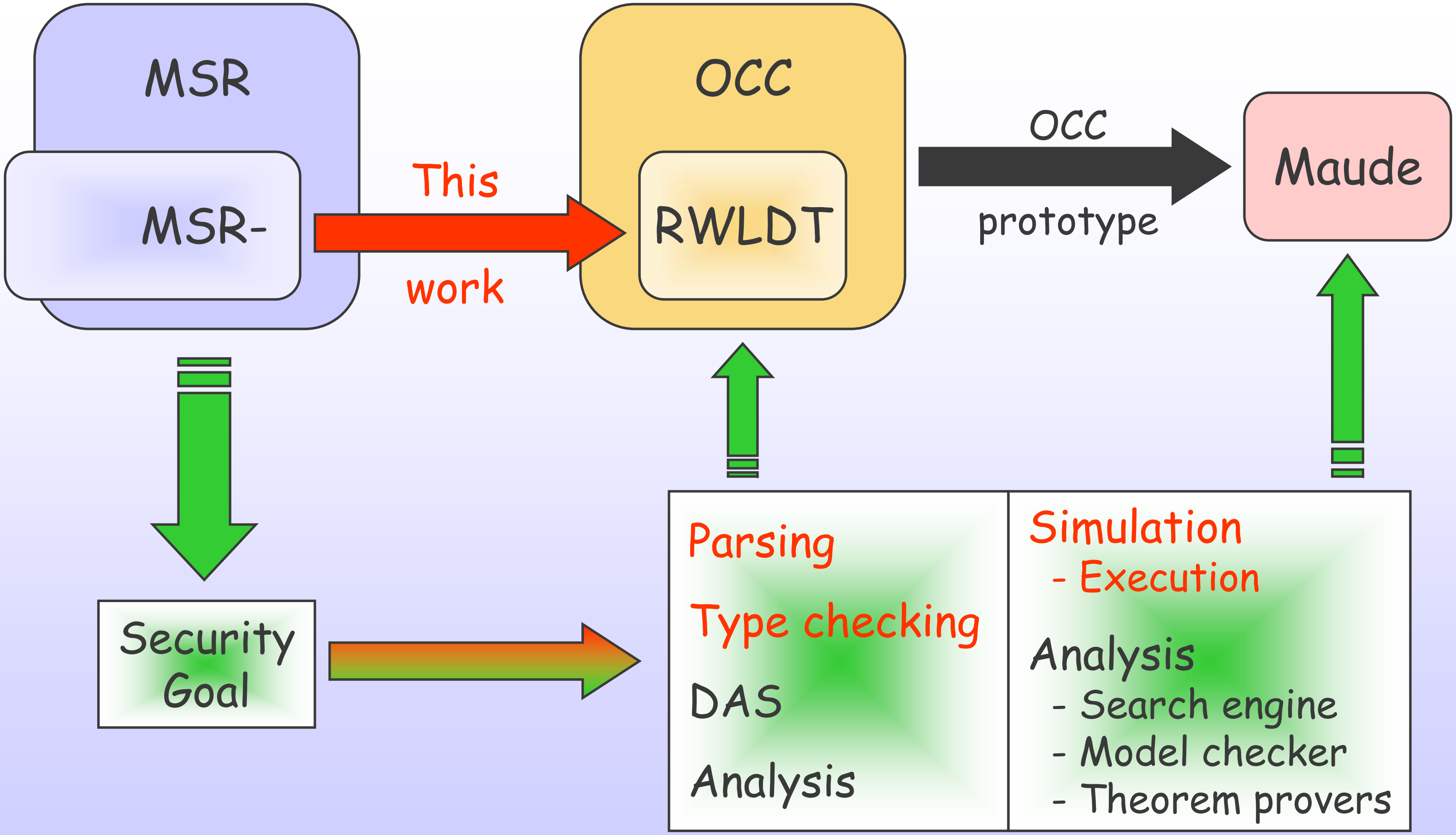
- Implement MSR in compatible language
  - Maude
- Port range of verification methodologies
  - MSR implementation as verification middleware
  - Compositional verification
- Verify large protocol suites
  - Kerberos (in fine detail)
  - Too hard using a single methodology

Big Picture



- **MSR**
  - Protocol specification language
  - Multiset rewriting
  - Dependent types
  - Existentials
- **Maude**
  - Flexible specification framework
  - Rewriting logic
  - Equational reasoning
  - Reflection

Implemented Architecture



Bestiary

- MSR-
  - MSR 2.0 with some restrictions and extensions
- 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]

MSR → MSR-

Small changes to simplify encoding

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

Emulated via pre-processing

Future work

Beta version

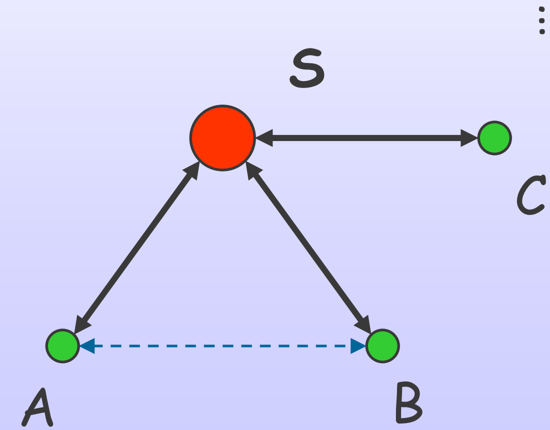
Supported Operations

- 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 -> 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

∀B:princ.

∃L:ΠB:princ. nonce * nonce * ltK B **S** -> state.

∀A:princ. ∀n:nonce. $\forall k_{BS}$:ltK B **S**. ∀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})

∀A:princ. ∀n, n_B:nonce. $\forall k_{BS}$:ltK B **S**.
∀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

    $\forall(\mathrm{LHS} \rightarrow \exists \mathbf{n}{:}\tau.\ \mathrm{RHS})$
    - Existentials
  - Roles

    $\forall A.\ \exists \mathbf{L}{:}\tau.\ r$
- 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

```
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 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 ∃ using counters
- Rules
  - Mapped to RWLDT rewrite axioms
    - Simulate ∃ using counters

Optimizations [not implemented]

- Reduce non-determinism

Representing Fresh Objects

- In rules

$$(\ldots) \rightarrow \exists n,n'\!:\!nonce.\ (\ldots\ n\ \ldots\ n'\ \ldots)$$

$$(\ldots),\ next(c) \rightarrow (\ldots\ nonce'(c)\ \ldots\ nonce'(c+1)\ \ldots),\ next(c+2)$$

  - nonce' : nat -> nonce is an injection

- In roles

$$\exists L_1,L_2.\ (\ldots\ (\ldots,\ L_1\ t \rightarrow \ldots,\ L_2\ t'),\ \ldots)$$

Rule j

$$nextL(c) \rightarrow \ldots,\ T_j(\lambda t.\ L'_1\ c\ t,\ \lambda t.\ L'_2\ (c+1)\ t),\ \ldots,\ nextL(c+2)$$

$$T_j(L_1,L_2),\ \ldots,\ L_1\ t \rightarrow \ldots,\ L_2\ t' \qquad \ldots$$

  - L'_i : nat -> τ_i -> state are injections

(done using conditional rewriting)

Representing Roles

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

princ(A), nextL(c) $\rightarrow$ $T_1(A,Ls)$, …, $T_n(A,Ls)$, 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), $lhs_i \rightarrow T_1(A,Ls)$, …, rhs_i …, $T_n(A,Ls)$, princ(A), nextL(c')
  - $T_i(A,Ls)$, $lhs_i \rightarrow rhs_i$

Representing Rules

$\forall x{:}\tau.\ lhs \rightarrow rhs$

$\underline{\tau}(x), \ldots, \ldots, lhs \rightarrow \underline{\tau}(x), \ldots, rhs$

- 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 [not implemented]

- Use single counter
  - $\forall A.\ \exists L.\ (lhs \rightarrow \exists n.\ rhs)$
- 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 : ... )      Optimized away

( B22 : ... )


<Axioms for S>
```

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)))
```

Trivia

- Versions
  - Alpha (current)
    - 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 …

Playing with MSR

`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
- Medium/long-term – Verification
  - Implement various methodologies
  - MSR as verification middleware

Next slides

MSR 3

Meeting point of

- multiset rewriting (state-transition model)
- process algebra (process-based computation)

- Rules can rewrite rules

  $a \rightarrow b, (c,d \rightarrow e)$

  - Drop distinction between state and rules
- Strong logical underpinning
  - Large freely-generated fragment of linear logic
- Strong connection to process calculus
  - Direct embedding of asynch. π- and join calculus
- Protocol specification
  - Choose and mix approaches

Data Access Specification – DAS

Check that principals entitled to operations

- Crypto only with known/allowed keys
- Local state is private

- Characterize the Dolev-Yao attacker
  - DY intruder uses same operations as regular principals
- Intro/elim rules for constructors

$$\frac{k \qquad m}{\{m\}_k} \qquad\qquad \frac{\{m\}_k \qquad k}{m}$$

- Free algebra / cancellation laws dilemma
  - Dec(k, Enc(k, m)) = m