Protokollsicherheit

Multiset Rewriting and Complexity Results

Multiset Rewriting (MSR)

As we saw, as Horn clauses are specified using classical logic, it is not possible to encode protocols and the intruder in both **sound** and **complete** fashion.

Multiset Rewriting (MSR)

As we saw, as Horn clauses are specified using classical logic, it is not possible to encode protocols and the intruder in both **sound** and **complete** fashion.

In particular, Proverif might return **false attacks**.

Example

$$\nu a.(\overline{a}\langle c\rangle \mid a(x).a(x).\overline{x}\langle s\rangle)$$

$\supset \; msg(a[], c[])$

$msg(a[], x) \wedge msg(a[], x) \supset \; msg(x, s[])$

The order of events is important!

Multiset Rewriting (MSR)

Multiset Rewriting (MSR)

First Order Signature: A set of function, variable, and predicate symbols all with their corresponding types. Constants are function symbols with zero arity.

$$\text{enc} : \text{key} \times \text{bitstring} \rightarrow \text{bitstring}$$

Multiset Rewriting (MSR)

First Order Signature: A set of function, variable, and predicate symbols all with their corresponding types. Constants are function symbols with zero arity.

$$\text{enc} : \text{key} \times \text{bitstring} \rightarrow \text{bitstring}$$

Terms: These are expressions built by applying functions to arguments, respecting the types and arities of symbols.

Multiset Rewriting (MSR)

First Order Signature: A set of function, variable, and predicate symbols all with their corresponding types. Constants are function symbols with zero arity.

$$\text{enc} : \text{key} \times \text{bitstring} \rightarrow \text{bitstring}$$

Terms: These are expressions built by applying functions to arguments, respecting the types and arities of symbols.

Facts: Are *ground* first-order atomic formula.

Multiset Rewriting (MSR)

First Order Signature: A set of function, variable, and predicate symbols all with their corresponding types. Constants are function symbols with zero arity.

$$\text{enc} : \text{key} \times \text{bitstring} \rightarrow \text{bitstring}$$

Terms: These are expressions built by applying functions to arguments, respecting the types and arities of symbols.

Facts: Are *ground* first-order atomic formula.

States: A state is a *multiset* of facts. Here we consider only finite states.

A multiset is formally a pair $\langle \mathcal{S}, m \rangle$, where $\mathcal{S}$ is a set and $m : \mathcal{S} \rightarrow \mathbb{N}_{\geq 1}$, assigning to each element of $\mathcal{S}$ its multiplicity.

$$\{\text{p(a), p(a), q(b), r(a, b), q(b)}\}$$

Multiset Rewriting (MSR)

Rules: Rules are composed of two multiset of atomic formulas:

$$F_1, \ldots, F_n \rightarrow G_1, \ldots, G_m$$

where $F_1, \ldots, F_n$ is the pre-condition of the rule and $G_1, \ldots, G_m$ is the post-condition of the rule.

Multiset Rewriting (MSR)

Rules: Rules are composed of two multiset of atomic formulas:

$$F_1, \ldots, F_n \to G_1, \ldots, G_m$$

where $F_1, \ldots, F_n$ is the pre-condition of the rule and $G_1, \ldots, G_m$ is the post-condition of the rule.

Such a rule can be applied to a state $\mathcal{S}$, if $\mathcal{S}$ contains the facts $F_1\sigma, \ldots, F_n\sigma$ for some ground substitution σ. When applied the resulting state $\mathcal{S}'$ is obtained from $\mathcal{S}$ by removing the facts $F_1\sigma, \ldots, F_n\sigma$ and inserting the facts $G_1\sigma, \ldots, G_m\sigma$.

Multiset Rewriting (MSR)

Rules: Rules are composed of two multiset of atomic formulas:

$$F_1, \ldots, F_n \to G_1, \ldots, G_m$$

where $F_1, \ldots, F_n$ is the pre-condition of the rule and $G_1, \ldots, G_m$ is the post-condition of the rule.

Such a rule can be applied to a state $\mathcal{S}$, if $\mathcal{S}$ contains the facts $F_1\sigma, \ldots, F_n\sigma$ for some ground substitution σ. When applied the resulting state $\mathcal{S}'$ is obtained from $\mathcal{S}$ by removing the facts $F_1\sigma, \ldots, F_n\sigma$ and inserting the facts $G_1\sigma, \ldots, G_m\sigma$.

$$\{P(a), P(b)\} \xrightarrow[\sigma = \{x/a\}]{P(x) \to P(f(x))} \{P(f(a)), P(b)\}$$

Multiset Rewriting (MSR)

Rules: Rules are composed of two multiset of atomic formulas:

$$F_1, \ldots, F_n \to G_1, \ldots, G_m$$

where $F_1, \ldots, F_n$ is the pre-condition of the rule and $G_1, \ldots, G_m$ is the post-condition of the rule.

Such a rule can be applied to a state $\mathcal{S}$, if $\mathcal{S}$ contains the facts $F_1\sigma, \ldots, F_n\sigma$ for some ground substitution σ. When applied the resulting state $\mathcal{S}'$ is obtained from $\mathcal{S}$ by removing the facts $F_1\sigma, \ldots, F_n\sigma$ and inserting the facts $G_1\sigma, \ldots, G_m\sigma$.

$$\{P(a), P(b)\} \xrightarrow[\sigma = \{x/a\}]{P(x) \to P(f(x))} \{P(f(a)), P(b)\}$$

Derivation: Is a sequence of states, alternated by rules:

$$\mathcal{S}_1 \xrightarrow{r_1\sigma_1} \mathcal{S}_2 \xrightarrow{r_2\sigma_2} \cdots \mathcal{S}_{n-1} \xrightarrow{r_{n-1}\sigma_{n-1}} \mathcal{S}_n$$

Multiset Rewriting (MSR)

Rules: Rules are composed of two multiset of atomic formulas:

$$F_1, \ldots, F_n \to G_1, \ldots, G_m$$

where $F_1, \ldots, F_n$ is the pre-condition of the rule and $G_1, \ldots, G_m$ is the post-condition of the rule.

Such a rule can be applied to a state $\mathcal{S}$, if $\mathcal{S}$ contains the facts $F_1\sigma, \ldots, F_n\sigma$ for some ground substitution σ. When applied the resulting state $\mathcal{S}'$ is obtained from $\mathcal{S}$ by removing the facts $F_1\sigma, \ldots, F_n\sigma$ and inserting the facts $G_1\sigma, \ldots, G_m\sigma$.

$$\{P(a), P(b)\} \xrightarrow[\sigma = \{x/a\}]{P(x) \to P(f(x))} \{P(f(a)), P(b)\}$$

Derivation: Is a sequence of states, alternated by rules:

$$\mathcal{S}_1 \xrightarrow{r_1\sigma_1} \mathcal{S}_2 \xrightarrow{r_2\sigma_2} \cdots \mathcal{S}_{n-1} \xrightarrow{r_{n-1}\sigma_{n-1}} \mathcal{S}_n$$

Programming with MSR

Finite Automata

Programming with MSR

Finite Automata

Assume a given automata A:

Types: st for states, symb for input symbols, and string for list of input symbols.

Programming with MSR

Finite Automata

Assume a given automata A:

Types: st for states, symb for input symbols, and string for list of input symbols.

Predicates: State for the current state and Input for current input.

Programming with MSR

Finite Automata

Assume a given automata A:

Types: st for states, symb for input symbols, and string for list of input symbols.

Predicates: State for the current state and Input for current input.

Functions: cons : symb $\times$ string $\rightarrow$ string.

We write cons(a, str) as $a \cdot str$.

$q_0, \ldots, q_n$: st for the states of the automata,
a, b : symb the input symbols;
nil : string for the empy string.

Programming with MSR

Finite Automata

Assume a given automata A:

Rules: One rule for each transition of the automata. For example:

$$\text{State}(q_0), \text{Input}(a \cdot x) \rightarrow \text{State}(q_1), \text{Input}(x)$$
$$\text{State}(q_0), \text{Input}(b \cdot x) \rightarrow \text{State}(q_2), \text{Input}(x)$$
$$\text{State}(q_1), \text{Input}(a \cdot x) \rightarrow \text{State}(q_3), \text{Input}(x)$$
$$\text{State}(q_1), \text{Input}(b \cdot x) \rightarrow \text{State}(q_0), \text{Input}(x)$$

Programming with MSR

Finite Automata

Assume a given automata A:

Rules: One rule for each transition of the automata. For example:

$$\begin{aligned}
\text{State}(q_0), \text{Input}(a \cdot x) &\rightarrow \text{State}(q_1), \text{Input}(x) \\
\text{State}(q_0), \text{Input}(b \cdot x) &\rightarrow \text{State}(q_2), \text{Input}(x) \\
\text{State}(q_1), \text{Input}(a \cdot x) &\rightarrow \text{State}(q_3), \text{Input}(x) \\
\text{State}(q_1), \text{Input}(b \cdot x) &\rightarrow \text{State}(q_0), \text{Input}(x)
\end{aligned}$$

Derivation:

$$\begin{aligned}
\{\text{State}(q_0), \text{Input}(a \cdot b \cdot nil)\} &\rightarrow \{\text{State}(q_1), \text{Input}(b \cdot nil)\} \\
&\rightarrow \{\text{State}(q_0), \text{Input}(nil)\}
\end{aligned}$$

MSR with existentials

MSR with existentials

Problem: No easy way to express **freshness** in MSR.

MSR with existentials

Problem: No easy way to express **freshness** in MSR.

Inspiration on proof theory – **Natural Deduction Elimination Rule**

$$\dfrac{\exists x.\phi \qquad \begin{array}{c}\phi[c/x]\\ \vdots \\ \psi\end{array}}{\psi}\ \exists_E$$

where c does not appear in any other hypothesis, that is, it is **fresh**.

MSR with existentials

Problem: No easy way to express **freshness** in MSR.

Inspiration on proof theory – **Natural Deduction Elimination Rule**

$$\frac{\exists x.\phi \qquad \begin{array}{c}\phi[c/x]\\ \vdots \\ \psi\end{array}}{\psi}\ \exists_E$$

where c does not appear in any other hypothesis, that is, it is **fresh**.

$$P(x) \rightarrow \exists.Q(x,n)$$

$$\{P(a), P(b)\} \rightarrow \{Q(a,c), P(b)\}$$

MSR with existentials: Turing Machines

MSR with existentials: Turing Machines

Consider a Turing machine $M = \langle \Sigma, Q, \delta, q_f, q_i \rangle$.

Types: states for Turing machine states, cell for cell names, and symbol for the cell contents.

MSR with existentials: Turing Machines

Consider a Turing machine $M = \langle \Sigma, Q, \delta, q_f, q_i \rangle$.

Types: states for Turing machine states, cell for cell names, and symbol for the cell contents.

Predicates:

Curr : state $\times$ cell – Current state
Cont : cell $\times$ symbol – Contents of a cell
Adj : cell $\times$ cell – Cell ordering

MSR with existentials: Turing Machines

Consider a Turing machine $M = \langle \Sigma, Q, \delta, q_f, q_i \rangle$.

Types: states for Turing machine states, cell for cell names, and symbol for the cell contents.

Predicates:

Curr : state $\times$ cell – Current state
Cont : cell $\times$ symbol – Contents of a cell
Adj : cell $\times$ cell – Cell ordering

Constants:

$q_0, q_1, \ldots, q_n$: state – states in Q
$0, 1, \square$: cell – symbols in Σ
$c_0, c_1, \ldots, c_n, c_\#$: cell – initial used cells in the tape

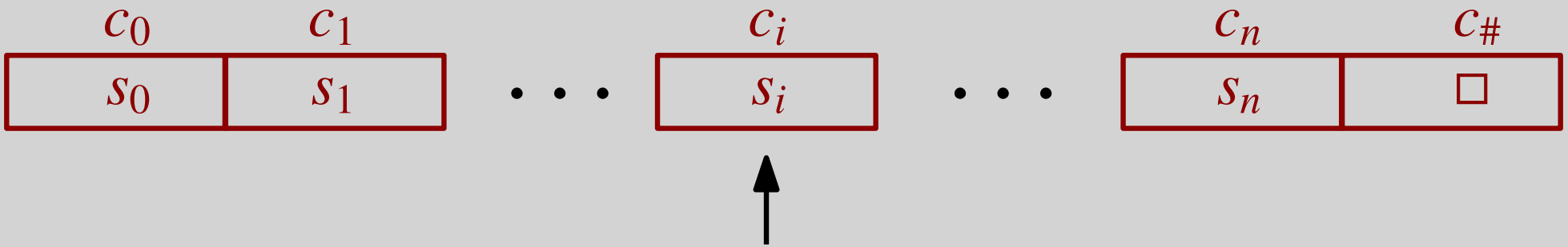
MSR with existentials: Turing Machines

Encoding Turing Machine Configurations

MSR with existentials: Turing Machines

Encoding Turing Machine Configurations

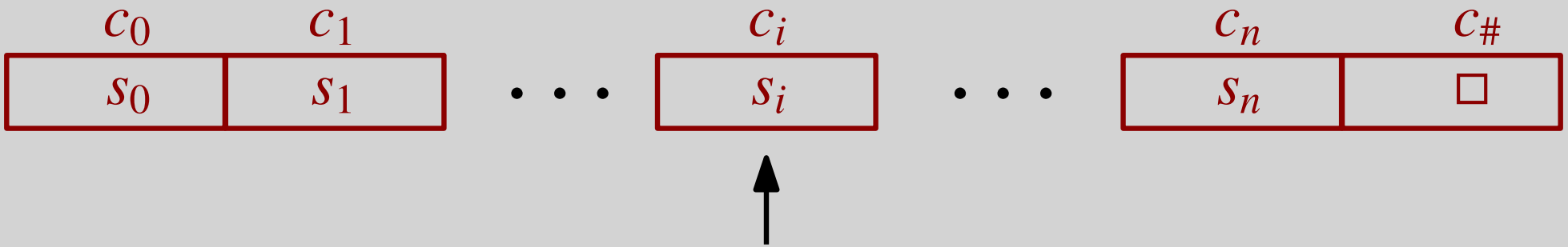
State $q_j \in Q$



MSR with existentials: Turing Machines

Encoding Turing Machine Configurations

State $q_j \in Q$



Multiset:

$\{\text{Curr}(q_j, c_i), \text{Adj}(c_0, c_1), \ldots, \text{Adj}(c_n, c_\#), \text{Cont}(c_0, s_0), \ldots, \text{Cont}(c_\#, \square)\}$

MSR with existentials: Turing Machines

Encoding Turing Machine Transitions

Maintenance Rules: Used to extend the tape.

$\mathsf{Adj}(x, c_{\#}) \longrightarrow \exists y.\mathsf{Adj}(x, y), \mathsf{Adj}(y, c_{\#}), \mathsf{Cont}(y, \square)$

MSR with existentials: Turing Machines

Encoding Turing Machine Transitions

Maintenance Rules: Used to extend the tape.

$\text{Adj}(x, c_{\#}) \longrightarrow \exists y.\text{Adj}(x, y), \text{Adj}(y, c_{\#}), \text{Cont}(y, \square)$

Move Left: Specifying transition moving the head of the tape to the left.

Write 1: $\text{Curr}(q, c), \text{Cont}(c, s), \text{Adj}(c', c) \longrightarrow \text{Curr}(q', c'), \text{Cont}(c, 1), \text{Adj}(c', c)$

Write 0: $\text{Curr}(q, c), \text{Cont}(c, s), \text{Adj}(c', c) \longrightarrow \text{Curr}(q', c'), \text{Cont}(c, 0), \text{Adj}(c', c)$

MSR with existentials: Turing Machines

Encoding Turing Machine Transitions

Maintenance Rules: Used to extend the tape.

$\text{Adj}(x, c_{\#}) \longrightarrow \exists y.\text{Adj}(x, y), \text{Adj}(y, c_{\#}), \text{Cont}(y, \square)$

Move Left: Specifying transition moving the head of the tape to the left.

Write 1: $\text{Curr}(q, c), \text{Cont}(c, s), \text{Adj}(c', c) \longrightarrow \text{Curr}(q', c'), \text{Cont}(c, 1), \text{Adj}(c', c)$

Write 0: $\text{Curr}(q, c), \text{Cont}(c, s), \text{Adj}(c', c) \longrightarrow \text{Curr}(q', c'), \text{Cont}(c, 0), \text{Adj}(c', c)$

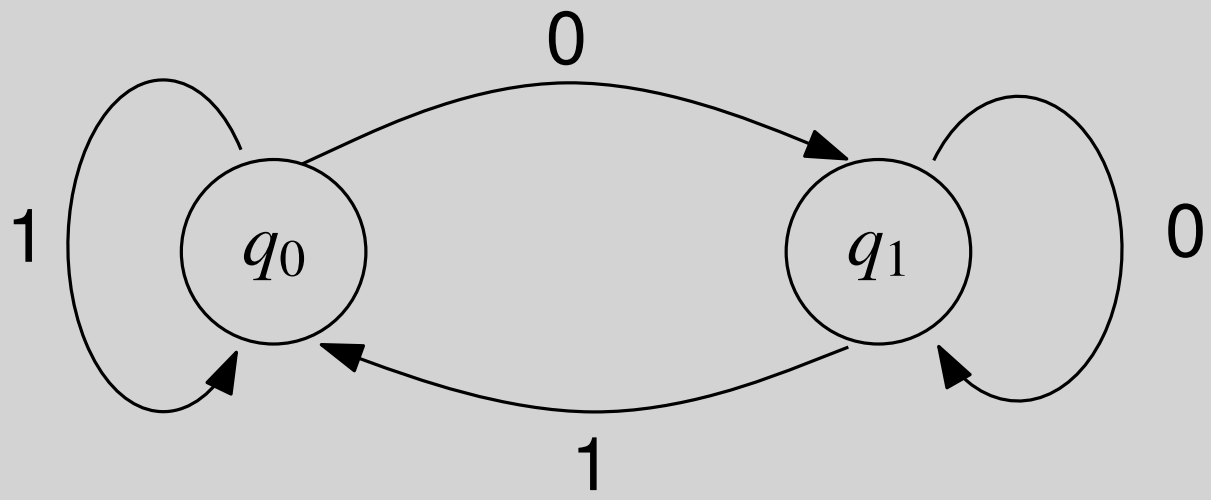
Move Right: Specifying transition moving the head of the tape to the right.

Write 1: $\text{Curr}(q, c), \text{Cont}(c, s), \text{Adj}(c, c') \longrightarrow \text{Curr}(q', c'), \text{Cont}(c, 1), \text{Adj}(c, c')$

Write 0: $\text{Curr}(q, c), \text{Cont}(c, s), \text{Adj}(c, c') \longrightarrow \text{Curr}(q', c'), \text{Cont}(c, 0), \text{Adj}(c, c')$

MSR with existentials: Turing Machines

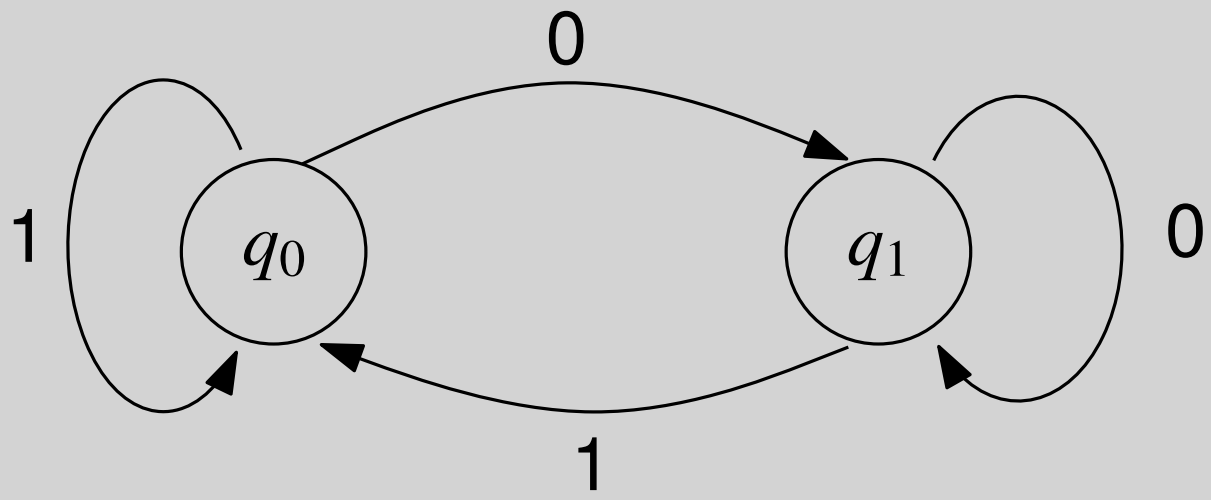
Sample Computation



When it finds two consecutive zeros it writes one at the cell after the second zero and returns the head back to the second zero.

MSR with existentials: Turing Machines

Sample Computation



When it finds two consecutive zeros it writes one at the cell after the second zero and returns the head back to the second zero.

Initial configuration

$$\left\{ \begin{array}{c} \text{Curr}(q_0, c_0), \text{Adj}(c_0, c_1), \text{Adj}(c_1, c_2), \text{Adj}(c_2, c_\#), \\ \text{Cont}(c_0, 1), \text{Cont}(c_1, 0), \text{Cont}(c_2, 0), \text{Cont}(c_\#, \square) \end{array} \right\}$$

Some Exercises

Write a MSR theory modelling the block world scenario.

Write a MSR theory that can solve the Tower of Hanoi problem.

MSR with existentials

Terminology

$$P, Q, Q, R \to P, Q, R, R$$

The rule above creates R facts;

The rule above preserves P facts;

The rule above consumes Q facts;

A rule $l \to r$ enables a rule $l' \to r'$ if there are substitutions σ, σ' such that some fact $P \in r\sigma$ is also in $l'\sigma'$. A theory $\mathcal{T}$ precedes a theory $\mathcal{R}$ if no rule in $\mathcal{R}$ enables a rule in $\mathcal{T}$.

Encoding Protocols

Bounded Role Theories: There is a finite list of role state predicates

$$S_0, \ldots, S_n$$

such that for every rule $l \to r$, there is one occurrence of a role state in l, say S_i and one occurrence of a role state in r, say S_j. Moreover, $i < j$.

Intuition: The state of the protocol is always advancing.

Encoding Protocols

Protocol

$$A \rightarrow B : n_a$$
$$B \rightarrow A : n_a, n_b$$
$$A \rightarrow B : n_b$$

Role State Predicates

A_0 : Alice at state 0.
$A_1(n_a)$: Alice at state 1, with her nonce.
$A_2(n_a, n_b)$: Alice at state 1, with both nonces.
B_0 : Bob at state 0.
$B_1(n_a, n_b)$: Bob at state 1, with both nonces.
$B_2(n_a, n_b)$: Bob at state 1, with both nonces.
N_i : Represent the network.

Bounded Protocol Theory

$$A_0 \rightarrow \exists x.A_1(x), N_1(x)$$
$$B_0, N_1(x) \rightarrow \exists y.B_1(x, y), N_2(x, y)$$
$$A_1(x), N_2(x, y) \rightarrow A_2(x, y), N_3(y)$$
$$B_1(x, y), N_3(y) \rightarrow B_2(x, y)$$

Encoding Protocols

Role Generation Theories: If $\mathcal{A}_1, \ldots, \mathcal{A}_n$ are bounded protocol theories, a role generation theory is a set of rules of the form:

$$P_1, \ldots, P_n \longrightarrow P_1, \ldots, P_n, S_i$$

where $P_1, \ldots, P_n$ are persistent facts not involving any role predicate and S_i is a role state of $\mathcal{A}_1, \ldots, \mathcal{A}_n$.

Intuition: Role generation theories that can be used to start new protocol sessions.

Well Founded Protocol Theories: A well founded protocol theory is of the form $\mathcal{R} \uplus \mathcal{A}_1 \uplus \cdots \uplus \mathcal{A}_n$, where $\mathcal{R}$ is a role generation theory, and $\mathcal{A}_1 \uplus \cdots \uplus \mathcal{A}_n$ are bounded role theories with disjoint role states.

Theory $\mathcal{A}$

$$A_0 \rightarrow \exists x.A_1(x), N_1(x)$$
$$A_1(x), N_2(x, y) \rightarrow A_2(x, y), N_3(y)$$

Theory $\mathcal{B}$

$$B_0, N_1(x) \rightarrow \exists y.B_1(x, y), N_2(x, y)$$
$$B_1(x, y), N_3(y) \rightarrow B_2(x, y)$$

Specifying the Intruder

As per the Dolev-Yao model, the intruder can read, intercept, compose, decompose, and generate fresh data.

Two Phase Intruder

The intruder processes data in two phases:

Decomposition Phase

$$N_1(x) \longrightarrow D(x)$$
$$N_2(x, y) \longrightarrow D(\langle x, y \rangle)$$
$$D(\langle x, y \rangle) \longrightarrow D(x), D(y)$$
$$D(x) \longrightarrow M(x)$$

Composition Phase

$$M(x) \longrightarrow C(x), M(x)$$
$$C(x) \longrightarrow N_1(x)$$
$$C(x), C(y) \longrightarrow C(\langle x, y \rangle)$$
$$C(\langle x, y \rangle) \longrightarrow N_2(x, y)$$

Nonce generation

$$\longrightarrow \exists x.M(x)$$

Specifying Encryption

Predicate:

KP(k_e, k_d) – denotes pair of public and private key of some principal.
Ann(k_e) – denotes a distributed public key of some principal.

Function: enc(k_e, m) – denotes encryption of message m using key k_e

Decryption

$$D(enc(k, x)), \mathsf{KP}(k, k'), M(k') \longrightarrow D(x), \mathsf{KP}(k, k'), M(\mathsf{enc}(k, x)), M(k')$$

Encryption

$$M(k), C(x) \longrightarrow M(k), C(\mathsf{enc}(k, x))$$

Intuition: Completeness of *normalized derivations* inspired by the same notion in proof theory:

First decompose and then compose messages.

Standard Derivations and the Secrecy Problem

Standard trace of a given well founded protocol theory $\mathcal{R} \uplus \mathbf{A}$ and a two phase intruder theory $\mathcal{M}$ is a derivation that has all steps from $\mathcal{R}$ first, and that interleaving steps from the protocol theories $\mathbf{A}$ with normalized derivations from the intruder theory $\mathcal{M}$.

$$\mathcal{S}_0 \xrightarrow{r_i} \cdots \xrightarrow{r_j} \mathcal{S}_k \xrightarrow{a_l} \mathcal{S}_{k+1} \longrightarrow \text{ Normalized Derivation } \xrightarrow{a_p}$$

Secercy Problem: Given a well founded protocol theory $\mathcal{R} \uplus \mathbf{A}$, a two phase intruder theory $\mathcal{M}$, and a secret constant s is there a standard derivation leading from an initial state $\mathcal{S}$ to a state where the intruder knows s, that is, containing the fact $M(s)$.

Undecidability of the Secrecy Problem

We first show that the **existential Horn clause problem** is **undecidable** and then show that there is a sound and complete reduction to the secrecy problem.

Existential Horn Clause Problem

$$\forall \vec{x}.[P_1 \wedge \cdots \wedge P_n \Rightarrow \exists \vec{y}.[Q_1 \wedge \cdots \wedge Q_k]]$$

Given a set of existential Horn clauses, does a first-order formula F follow from the theory?

Undecidability of the Existential Horn Clause Problem

Encoding a Deterministic Turing Machine $M = \langle \Sigma, Q, \delta, q_0, q_f \rangle$

Similar to the previous encoding, but now as we are in classical logic all facts are true all the time.

If present, then head is pointing at this cell at this state.

Predicates: No more Curr.

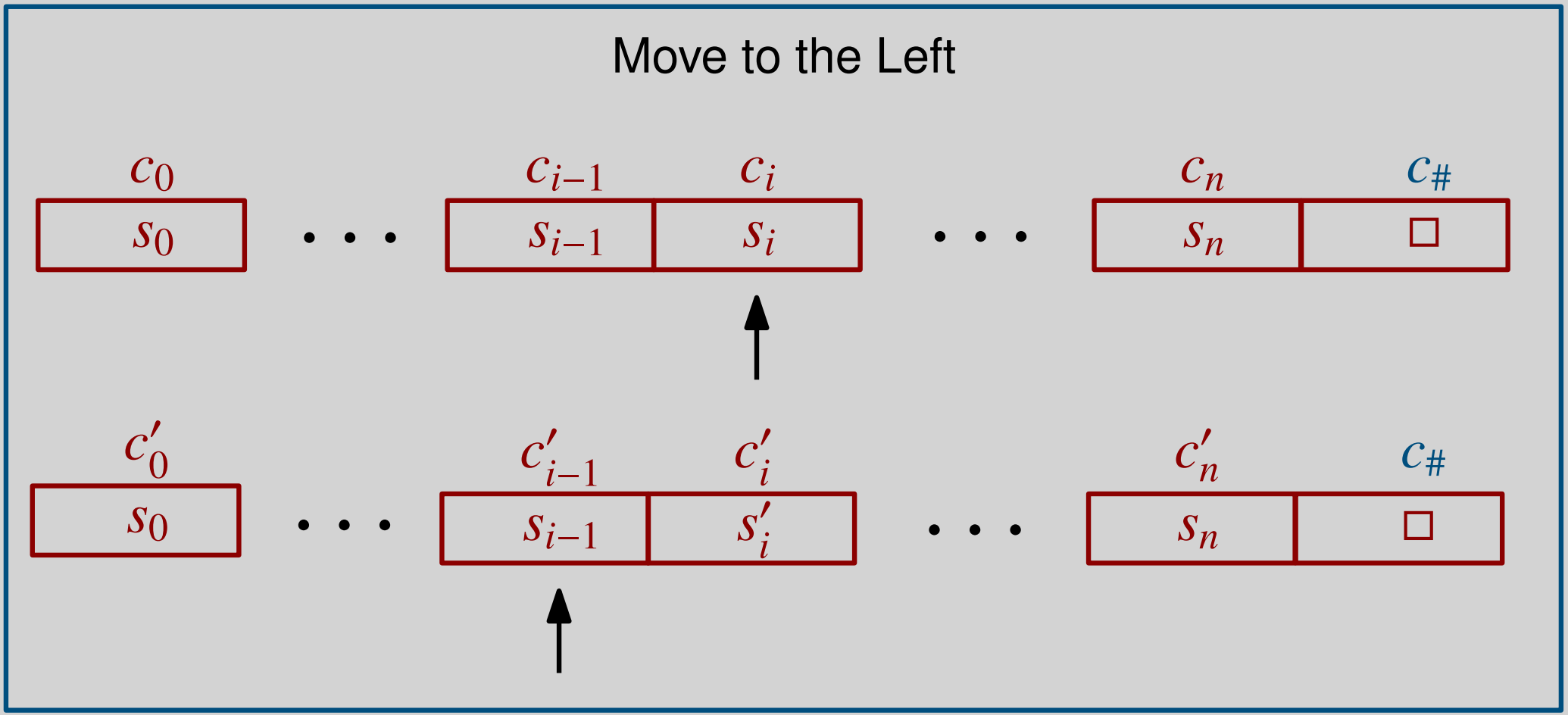
Cont : cell $\times$ symbol $\times$ state – Contents of a cell

Adj : cell $\times$ cell – Cell ordering

Below : cell $\times$ cell – Cell ordering in time

Used to model transitions.

Undecidability of the Existential Horn Clause Problem



$$\text{Adj}(c_{i-1}, c_i), \text{Cont}(c_{i-1}, s_{i-1}, @), \text{Cont}(c_i, s_i, q)$$
$$\text{Adj}(c'_{i-1}, c'_i), \text{Cont}(c'_{i-1}, s'_{i-1}, q'), \text{Cont}(c'_i, s'_i, @)$$
$$\text{Below}(c_{i-1}, c'_{i-1}), \text{Below}(c_i, c'_i)$$

Undecidability of the Existential Horn Clause Problem

Encoding Transition Rules

Move to the Left: $\delta(q_i, s) = \{(q_j, s', L)\}$

$$\begin{aligned}
&\forall x, y, z, a, b, [\mathsf{Adj}(x, y) \wedge \mathsf{Adj}(y, z) \wedge \\
&\quad \mathsf{Cont}(x, a, @) \wedge \mathsf{Cont}(y, s, q_i) \wedge \mathsf{Cont}(z, b, @) \Rightarrow \\
&\quad\quad \exists x', y', z'.\mathsf{Adj}(x', y') \wedge \mathsf{Adj}(y', z') \wedge \\
&\quad\quad \mathsf{Below}(x, x') \wedge \mathsf{Below}(y, y') \wedge \mathsf{Below}(z, z') \wedge \\
&\quad\quad \mathsf{Cont}(x', a, q_j) \wedge \mathsf{Cont}(y', s', @) \wedge \mathsf{Cont}(z', b, @)]
\end{aligned}$$

Move to the Right: $\delta(q_i, s) = \{(q_j, s', R)\}$

$$\begin{aligned}
&\forall x, y, z, a, b, [\mathsf{Adj}(x, y) \wedge \mathsf{Adj}(y, z) \wedge \\
&\quad \mathsf{Cont}(x, a, @) \wedge \mathsf{Cont}(y, s, q_i) \wedge \mathsf{Cont}(z, b, @) \Rightarrow \\
&\quad\quad \exists x', y', z'.\mathsf{Adj}(x', y') \wedge \mathsf{Adj}(y', z') \wedge \\
&\quad\quad \mathsf{Below}(x, x') \wedge \mathsf{Below}(y, y') \wedge \mathsf{Below}(z, z') \wedge \\
&\quad\quad \mathsf{Cont}(x', a, @) \wedge \mathsf{Cont}(y', s', @) \wedge \mathsf{Cont}(z', b, q_j)]
\end{aligned}$$

Undecidability of the Existential Horn Clause Problem

Maintenance Clauses

Create new rows:

$$\forall x, y, z, a, b, c[\text{Adj}(x, y) \wedge \text{Adj}(y, z) \wedge \\ \text{Cont}(x, a, @) \wedge \text{Cont}(y, b, @) \wedge \text{Cont}(z, c, @) \Rightarrow \\ \exists y'.\text{Below}(y, y') \wedge \text{Cont}(y', b, @)]$$

Extending the tape:

$$\forall x, a.[\text{Adj}(x, c_{\#}) \wedge \text{Below}(x, y) \Rightarrow \\ \exists z.\text{Adj}(y, z) \wedge \text{Cont}(z, \square, @) \wedge \text{Adj}(z, c_{\#})]$$

Plugging rows together:

$$\forall x, x, y, y'.[\text{Adj}(x, y) \wedge \text{Below}(x, x') \wedge \text{Below}(y, y') \Rightarrow \text{Adj}(x', y')]$$

Undecidability of the Existential Horn Clause Problem

Encoding the Acceptance Condition:

$$\forall x, a.[\text{Cont}(x, a, q_f) \Rightarrow Accept]$$

Lemma: $H(M, w) \vdash Accept$ if and only if machine M halts in the accepting state on input string w.

Corollary: The existential Horn clause problem is undecidable even without function symbols.

Reduction of the Existential Horn Clause Problem to the Secrecy Problem

$$\forall \vec{x}.[P_1 \wedge \cdots \wedge P_n \Rightarrow \exists \vec{y}.[Q_1 \wedge \cdots \wedge Q_k]]$$

Encoding of Formulas

$$\lceil P_1 \wedge \cdots \wedge P_n \rceil = \text{enc}(K, \langle P_1, \cdots, P_n \rangle)$$

Secret private key

Intuition: Only the honest participants should be able to derive facts. The intruder will only pass encrypted terms around and be responsible for duplicating them.

Reduction of the Existential Horn Clause Problem to the Secrecy Problem

Three types of bounded role theories

Encoding clauses:

$$\forall \vec{x}.[P_1 \wedge \cdots \wedge P_n \Rightarrow \exists \vec{y}.[Q_1 \wedge \cdots \wedge Q_k]]$$

$$A_0, N_{Ra_0}(\lceil P_1 \wedge \cdots \wedge P_n \rceil) \longrightarrow \exists \vec{y}.A_1, N_{S\,a_1}(\lceil Q_1 \wedge \cdots \wedge Q_k \rceil)$$

Decomposing Formulas:

$$A_0, N_{Ra_0}(\lceil Q_1 \wedge \cdots \wedge Q_k \rceil) \longrightarrow A_1, N_{S\,a_1}(\lceil Q_i \rceil)$$

for each $Q_1 \wedge \ldots \wedge Q_k$ appearing at the right-hand side of clauses and for $1 \leq i \leq k$

Composing Formulas:

$$A_0, N_{Ra_0}(\lceil P_1 \rceil) \longrightarrow A_1(\lceil P_1 \rceil), N_{S\,1}()$$

$$A_1(\lceil P_1 \rceil), N_{R1}(\lceil P_2 \rceil) \longrightarrow A_1(\lceil P_1 \wedge P_2 \rceil), N_{S\,2}()$$

$$\cdots$$

$$A_{n-1}(\lceil P_1 \wedge \cdots \wedge P_{n-1} \rceil), N_{R1}(\lceil P_n \rceil) \longrightarrow A_n(), N_{S\,n}(\lceil P_1 \wedge \cdots \wedge P_n \rceil)$$

Reduction of the Existential Horn Clause Problem to the Secrecy Problem

Release Secret:

$$A_0, N_{R1}(\lceil \alpha \rceil) \longrightarrow A_1, N_{S1}(secret)$$

Lemma: The intruder can learn the secret $secret$ from $Role(HC)$ if and only if the formula α is derivable from the set of existential Horn clauses HC.

Corollary: The secrecy problem with an unbounded number of roles is undecidable even if the size of terms is bounded.

Bounding the number of protocol sessions

The secrecy problem is NP-complete when the number of protocol sessions and the size of messages is bounded.

NP-hardness is shown by encoding the 3-SAT problem.

Propositional variables: $v_1, \ldots, v_n$

Propositional literals: $L = v$ or $L = \neg v$

Clauses: $C = L_1 \vee L_2 \vee L_3$

Formulas: $F = C_1 \wedge \cdots \wedge C_n$

Problem: Is a formula F satisfiable?

Try to find such encoding.

Bounding the number of protocol sessions

NP-membership

Recall that a well founded protocol theory is $\mathcal{R} \uplus \mathbf{A}$.
Task: Can we check in **polynomial time** whether a derivation is an attack?
Assumption: There is a bound on the number of protocol sessions, that is, a bound, r, on the number of role state instances and a bound k on the size of messages.

Check: Do attacks necessarily have polynomial length?

$$\mathcal{S}_0 \xrightarrow{r_i} \cdots \xrightarrow{r_j} \mathcal{S}_k \xrightarrow{a_l} \mathcal{S}_{k+1} \longrightarrow \text{ Normalized Derivation } \xrightarrow{a_p}$$

There are at most r $\mathcal{R}$-steps. Moreover, there are at most rR $\mathbf{A}$-steps, where R is the number of steps of the longest protocol.

Moreover, there at most kr values for a given role. Thus the upper bound on the distinct values that can appear in an attack is $B = krR$. Hence, there are at most B nonces in an attack. These can be generated in the role generation phase.

Bounding the number of protocol sessions

NP-membership

Recall that a well founded protocol theory is $\mathcal{R} \uplus \mathbf{A}$.

Task: Can we check in **polynomial time** whether a derivation is an attack?

Assumption: There is a bound on the number of protocol sessions, that is, a bound, r, on the number of role state instances and a bound k on the size of messages.

Bounding the number of protocol sessions

Check: Do attacks necessarily have polynomial length?

$$\mathcal{S}_0 \xrightarrow{r_i} \cdots \xrightarrow{r_j} \mathcal{S}_k \xrightarrow{a_l} \mathcal{S}_{k+1} \longrightarrow \text{ Normalized Derivation } \xrightarrow{a_p}$$

There are at most r $\mathcal{R}$-steps. Moreover, there are at most rR $\mathbf{A}$-steps, where R is the number of steps of the longest protocol.

Moreover, there at most kr values for a given role and at most $B = krR$ distinct values (including nonces) that can appear in an attack These nonces can be generated in the role generation phase.

1- Guess B nonces to be used by the intruder. These can be included in the intruder's knowledge;

2- Guess a selection of r role states. Call this set I_r;

3- Guess a candidate sequence with at most rR protocol steps. Label each protocol step with s_i for $0 \leq i \leq rR$.

Bounding the number of protocol sessions

Check: Can we verify in polynomial time whether a derivation is an attack?

Let $S_0 = I \cup I_r$, where I is the initial set of facts.

For each step s_i:

$$s_i : A_i(\vec{m}), N_R(\vec{n}) \ldots A_j(\vec{m'}), N_S(\vec{n})$$

1- Check whether $A_i(\vec{m}) \in S_{i-1}$; if not then REJECT;

2- Decompose the message $N_R(\vec{n})$ into M facts by using the intruder's rules until only persistent facts remain. Call this set $\mathcal{M}$;

3- Check whether $\mathcal{M} \subseteq S_{i-1}$; if not REJECT;

4- Decompose the message $N_S(\vec{n'})$ into M facts by using the intruder's rules until only persistent facts remain. Call this set $\mathcal{M}'$;

5- Assign $S_i := (S_{i-1} \uplus \{A_j(\vec{m'})\} \uplus \mathcal{M}) - \{A_i(\vec{m})\}$;

6- Increment i.

Bounding the number of protocol sessions

Check: Can we verify in polynomial time whether a derivation is an attack?

Let $S_0 = I \cup I_r$, where I is the initial set of facts.

For each step s_i:

$$s_i : A_i(\vec{m}), N_R(\vec{n}) \ldots A_j(\vec{m'}), N_S(\vec{n})$$

1- Check whether $A_i(\vec{m}) \in S_{i-1}$; if not then REJECT;

2- Decompose the message $N_R(\vec{n})$ into M facts by using the intruder's rules until only persistent facts remain. Call this set $\mathcal{M}$;

3- Check whether $\mathcal{M} \subseteq S_{i-1}$; if not REJECT;

4- Decompose the message $N_S(\vec{n'})$ into M facts by using the intruder's rules until only persistent facts remain. Call this set $\mathcal{M}'$;

5- Assign $S_i := (S_{i-1} \uplus \{A_j(\vec{m'})\} \uplus \mathcal{M}) - \{A_i(\vec{m})\}$;

6- Increment i.