The Maude-NRL Protocol Analyzer
Lecture 2: State Space Reduction in Maude-NPA

Catherine Meadows
Naval Research Laboratory, Washington, DC 20375
catherine.meadows@nrl.navy.mil

Formal Methods for the Science of Security Summer School,
University of Illinois, July 22-26, 2013
Title: The Maude-NRL Protocol Analyzer Lecture 2: State Space Reduction in Maude-NPA

Introduction

Outline

1. Introduction
2. Some "Easy" Ways of Reducing the Search Space
3. Grammars
4. Subsumption Partial Order Reduction
5. Super-Lazy Intruder
6. Interactions Between Subsumption Partial Order Reduction and Super Lazy Intruder
How Maude-NPA Controls the Search Space

- Left to itself, Maude-NPA will search forever
- Must use techniques for ruling out redundant or provably unreachable states to obtain finite search space
- We have developed a number of different techniques for doing this:
  - Intruder learns only once
  - Grammars
  - Subsumption
  - Super-Lazy Intruder
- We will cover these in this lecture
Important Assumptions

- Equational theory is of the form $E = R \uplus \Delta$
- $R$ is a set of rewrite rules confluent, terminating, and coherent wrt $\Delta$ and $\Delta$ is regular
- In any states produced by Maude-NPA $t \in I$, $t \notin I$, and negative terms are $R$-irreducible
- Furthermore, no substitutions produced by further search will make these terms reducible
- Reason:
  - Reason: many of the checks made by Maude-NPA for state space reduction rely on the presence of particular sub terms
  - Allowing these sub terms to vanish because of rewrite rules or further substitutions will invalidate the checks
  - We will show how these assumptions are guaranteed in the lecture on the asymmetric unification and the finite variant property
Outline

1. Introduction
2. Some "Easy" Ways of Reducing the Search Space
3. Grammars
4. Subsumption Partial Order Reduction
5. Super-Lazy Intruder
6. Interactions Between Subsumption Partial Order Reduction and Super Lazy Intruder
Execute Rule 1 First

- If there is a strand of the form \([ l_1, u^- | l_2 ]\) present, execute the rule replacing it by \([ l_1 | u^-, l_2 ]\), \(u \in \mathcal{I}\) first.
- If there are several fix an order and execute them all first, in that order.
- Removes extra step introduced by converting negative terms to intruder terms.
- Implementing this doubled the speed of the tool.
  - Not surprising, because replaced two steps by one.
Some "Easy" Ways of Reducing the Search Space

Using the Power of Strands

- Strands allow you to see the past and the future of a local execution
- Helpful since Maude-NPA very sensitive to the past and future

Things we’ve done so far

- If a term $x \notin I$ and a strand $[ l_1, -(x), l_2 | l_3 ]$ both appear in a state, then the state is unreachable
  - Reaching it would require violation of intruder-learns-once
- Let $f$ and $g$ be two terms containing $n(A, r)$. If
  - $f \in I$ appears in a state, and;
  - $[ l_1 | l_2, +(g), l_3, ]$ also appears, with strand identifier containing $r$ and no $n(A, r)$ term in $l_1$;

Then reaching the state requires the intruder to learn a nonce before it is generated and thus is unreachable.
Example

\[
\begin{align*}
\text{[ nil | (exp(E', n(B, r')))\,^{-}, (e(exp(E', n(B, r')), sec(a, r'')))\,^{-}, (sec(a, r''))\,^{+}] & \land \\
\text{[ nil | (exp(E', n(B, r')))\,^{-}, (sec(a, r''))\,^{-}, (e(exp(E', n(B, r')), sec(a, r'')))\,^{+} ] & \land \\
\text{:: r' ::} \\
\text{[ (A; B; E')\,^{-}, (B; A; exp(g, n(B, r'))))\,^{+} | (e(exp(E', n(B, r')), sec(a, r'')))\,^{-} ] & \land \\
\text{(sec(a, r''))\,\in I, \ exp(E', n(B, r'))\,\in I,} \\
\text{e(exp(E', n(B, r')), sec(a, r''))\,\in I, \ e(exp(E', n(B, r')), sec(a, r''))\,\notin I)}
\end{align*}
\]

- First stand is waiting for \( e(exp(E', n(B, r')), sec(a, r'')) \)
- But in order for it to receive it, intruder must send it
- Intruder can’t have learned it in the past, because \( e(exp(E', n(B, r')), sec(a, r''))\,\notin I \) records that it already”learned it sometime in the future
Outline

1. Introduction
2. Some "Easy" Ways of Reducing the Search Space
3. Grammars
4. Subsumption Partial Order Reduction
5. Super-Lazy Intruder
6. Interactions Between Subsumption Partial Order Reduction and Super Lazy Intruder
Idea Behind Grammars

- Inductively define **classes** of terms unlearnable by the intruder
- Intruder can learn a term described by the grammar only if it already knows a term described by the grammar
- Conclude intruder can never learn a term in the grammar
- Grammars are defined automatically
- Learn-only-once restriction key to their construction
Motivating Example

Consider protocol with:

- Two operators
  - \( e(K, X) \) stands for encryption of message \( X \) with key \( K \)
  - \( d(K, X) \) stands for decryption of message \( X \) with key \( K \)

- Two regular strands:
  - Two Intruder strands (Dolev-Yao):
    - \([- (X), + (d(k, X))]\)
    - \([+(e(k, r))]\)
  - Two Intruder strands
    - \([- (K), -(X), +(d(K, X))]\)
    - \([- (K), -(X), +(e(K, X))]\)

- One equation
  - \( d(K, e(K, X)) = X \)
A Partial (Backwards) Search Tree

\[
\begin{align*}
&\vdash r \\
&\vdash \left[ -(X), \mid +(d(k,X)) \right], X \mapsto e(k,r) \\
\end{align*}
\]

\[
\begin{align*}
&\vdash \{e(k, r)\} \\
&\vdash \{k, r\} \quad \text{stop} \\
&\vdash \{e(k, e(k, r))\} \\
\end{align*}
\]

\[
\begin{align*}
&\vdash \left[ nil \mid +e(k, r) \right] (initial) \\
&\vdash \{e(k, e(k, e(k, r)))\} \quad \ldots
\end{align*}
\]

Powerful tools:

1. **Learn-only-once**: terms the intruder will learn in the future and doesn’t know in the past.
2. **Unreachable states**: the intruder learns a term in a family only if he/she knew another term in that family in a past state.
(2) Grammars characterizing unreachable states

\[
Z \not\preceq r \\
\downarrow \\
\{ e(K, Z) \} \\
\downarrow \\
\{ e(K, e(K, Z)) \} \\
\downarrow \\
\{ e(K, e(K, e(K, Z))) \} \\
\downarrow \\
\vdots
\]

- Discover **Grammars** providing infinite set of terms intruder can’t learn.
  1. \( t \in L \)
  2. \( Z \in L \rightarrow e(Y, Z) \in L \)

- \( Z \notin \mathcal{I}, Z \not\preceq r \rightarrow e(A, Z) \in L \) (\( Z \not\preceq r \) means \( Z \) not subsumed by \( r \))

- \( Z \in L \rightarrow e(Y, Z) \in L \)

- If the intruder learns a term in the language, then he/she must have learned another term in a state in the past.
Grammar Generation Is Automated

- Start with initial grammar, giving a single term known by the intruder, along with conditions on the term, such as some sub term not yet known by the intruder
  - Maude-NPA uses function symbol definitions in protocol spec as source for initial grammars
  - User can define own initial grammars if desired, either in addition to or in place of Maude-NPA grammars
- Maude-NPA finds the terms the intruder needed to know to generate these terms
- Checks if new terms are also in the language defined by the grammar
- If not, uses a set of heuristics to add new grammar rules
- If no heuristic applies, adds an exception to the grammar rule
- Repeats this process until it reaches a fixed point
- In cases Maude-NPA fails to generate a grammar, it provides the reasons for its failure
Grammars

Status of Grammars

- Grammar generation heuristics little changed from original NRL Protocol Analyzer
- Works well on most theories we’ve tried
- Main exceptions are exclusive-or and Abelian groups: presence of inverses causes unexpected behavior
  - Grammar generation heuristics rely on assumptions about term on LHS of grammar rule being sub term of RHS of grammar rule
  - Not satisfied by grammars produced by these theories
  - Have a partial work-around for exclusive-or
- Currently planning to rethink grammars in order to address this
Outline

1. Introduction
2. Some "Easy" Ways of Reducing the Search Space
3. Grammars
4. Subsumption Partial Order Reduction
5. Super-Lazy Intruder
6. Interactions Between Subsumption Partial Order Reduction and Super Lazy Intruder
Partial order reduction standard idea in model checking, used in a lot of protocol analysis tools, too

- Identify when reachability of state $S$ implies reachability of $T$ and remove $S$
- In Maude-NPA, this happens, roughly, when $S \subseteq_\Delta \sigma T$ for some substitution $\sigma$
- Can then eliminate $S$
The Maude-NRL Protocol Analyzer  Lecture 2: State Space Reduction in Maude-NPA

Subsumption Partial Order Reduction

Idea Behind Subsumption Partial Order Reduction

- $S$ reachable means that there is a sequence of backwards narrowing steps

\[ S = S_k \leadsto_{\rho_k,R,E} S_{k-1} \leadsto_{\rho_{k-1},R,E} \ldots S_1 \leadsto_{\rho_1,R,E} S_0 \]

- Where $S_0$ is an initial state

- $S \subseteq_{=\Delta} \sigma T$ means that, considered as terms, $\sigma T|_p =_{\Delta} S$ for some position $p$

- Thus narrowing sequence from $S$ to $S_0$ lifts to a narrowing sequence from $T$ to $S_0$

\[ T \leadsto_{\rho_k\sigma,R,E} S_{k-1} \leadsto_{\rho_{k-1},R,E} \ldots S_1 \leadsto_{\rho_1,R,E} S_0 \]
Outline

1. Introduction
2. Some "Easy" Ways of Reducing the Search Space
3. Grammars
4. Subsumption Partial Order Reduction
5. Super-Lazy Intruder
6. Interactions Between Subsumption Partial Order Reduction and Super Lazy Intruder
Super-Lazy Intruder

- Based on an idea of David Basin, plus a trick used by the old NPA
- If a term $X \in I$ appears in a state, where $X$ is a variable, we assume that the intruder can easily find $x$, and so safe to drop it
- Super-lazy intruder: drop terms made out of variable terms, and terms initially known to the intruder, e.g. $X;Y$ and $e(K,a;Y)$, where $a$ is a name
- However, variables may later become instantiated
- In that case, removing the ghost term may lead to a false attack
- Solution: keep the term, and state it appears in, around as a "ghost"
  - Revive the ghost, replacing current state by ghost term and ghost state, but with current substitutions to variables if any variable subterm becomes instantiated
Example

\[
[ \text{nil} | K^- , e(K, \text{sec}(a, r''))^- , \text{sec}(a, r'')^+ ] &
\]
\[
:: r' ::
\]
\[
[ (A; B; E')^- , (B; A; \text{exp}(g, n(B, r')))^+ | (e(\text{exp}(E', n(B, r')), \text{sec}(a, r'')))^- ] &
\]
\[
(\text{exp}(E', n(B, r')), \text{sec}(a, r'')) \in \mathcal{I} , \ K \in \mathcal{I} , \ e(K, \text{sec}(a, r'')) \in \mathcal{I} , \ \text{sec}(a, r'') \notin \mathcal{I}
\]

- Intruder needs to learn some key \( K \), but any \( K \) will do
- Eliminate \( K \in \mathcal{I} \) from the state for now
Example Continued

- After more backwards narrowing steps, suppose
  \( e(K, \text{sec}(a, r??)) \) is unified with \( e(\text{exp}(X, n(a, r)), \text{sec}(a, r'')) \)
- \( K \) is replaced with \( \text{exp}(X, n(a, r)) \).
- Replace current state with resurrected state

\[
[nil \mid K^-, e(K, \text{sec}(a, r''))^-, \text{sec}(a, r'')^+] \& \\
:: r' :: \\
[ (A; B; E')^-, (B; A; \text{exp}(g, n(B, r')))^+ \mid (e(\text{exp}(E', n(B, r')), \text{sec}(a, r''))^-] \& \\
(e(\text{exp}(E', n(B, r')), \text{sec}(a, r'')) \in I, \text{exp}(X, n(a, r)) \in I, \text{exp}(X, n(a, r)), \text{sec}(a, r'')) \in I, \text{sec}(a, r'') \notin I)
\]

- Not quite as simple as instantiating ghost term variables:
  some other special concerns such as propagating nonces
- Detailed in Escobar et al. 2013.
The Maude-NRL Protocol Analyzer  Lecture 2: State Space Reduction in Maude-NPA
Interactions Between Subsumption Partial Order Reduction and Super Lazy Intruder

Outline

1. Introduction
2. Some "Easy" Ways of Reducing the Search Space
3. Grammars
4. Subsumption Partial Order Reduction
5. Super-Lazy Intruder
6. Interactions Between Subsumption Partial Order Reduction and Super Lazy Intruder
How Partial Order Reduction and Super Lazy Intruder May Interact

- When ghost created, node (without super-lazy term) remains
- When ghost resuscitated, it is identical to its ancestor node except
  - the super-lazy term is now included
  - the variables in the ghost terms are now instantiated
- In other words, the ancestor state dominates the resuscitated ghost state in the subsumption partial order!
- If the subsumption partial order check is not modified, Maude-NPA will not find a path to the resuscitated node, even if one exists
- Thus, potential incompleteness is introduced
Solving the Problem

- Most straightforward solution: do not apply subsumption partial order to super-lazy nodes and their resuscitations.
- However, keeping track of and verifying the information necessary to do this would have a negative impact on performance.
- Instead, we keep a history of state transitions:
  - When a message is sent or received.
  - When a ghost is created or resuscitated.
- By examining this history, we can identify conditions that necessary, to identify a super-lazy node and its resuscitated descendant.
- Thus, we achieve completeness at the cost of potentially missing some genuine cases of the subsumption partial order.
Experimental Results 1

<table>
<thead>
<tr>
<th>Protocol</th>
<th>none</th>
<th>Grammars</th>
<th></th>
<th></th>
<th></th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSPK</td>
<td>5 19 136 642 4021</td>
<td>4 12 49 185 758</td>
<td>758</td>
<td>81</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NSL</td>
<td>5 19 136 642 4019</td>
<td>4 12 50 190 804</td>
<td>804</td>
<td>79</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SecReT06</td>
<td>1 6 22 119 346</td>
<td>1 2 6 15 36</td>
<td>36</td>
<td>89</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SecReT07</td>
<td>6 20 140 635 4854</td>
<td>6 17 111 493 3823</td>
<td>3823</td>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DH</td>
<td>1 14 38 151 816</td>
<td>1 6 14 37 105</td>
<td>105</td>
<td>87</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Protocol</th>
<th>none</th>
<th>Input First</th>
<th></th>
<th></th>
<th></th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSPK</td>
<td>5 19 136 642 4021</td>
<td>11 123 1669 26432</td>
<td>N/A</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NSL</td>
<td>5 19 136 642 4019</td>
<td>11 123 1666 26291</td>
<td>N/A</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SecReT06</td>
<td>1 6 22 119 346</td>
<td>11 133 1977 32098</td>
<td>N/A</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SecReT07</td>
<td>6 20 140 635 4854</td>
<td>11 127 3402 N/A</td>
<td>N/A</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DH</td>
<td>1 14 38 151 816</td>
<td>14 135 1991 44157</td>
<td>N/A</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Protocol</th>
<th>none</th>
<th>Inconsistency</th>
<th></th>
<th></th>
<th></th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSPK</td>
<td>5 19 136 642 4021</td>
<td>5 18 95 310 650</td>
<td>650</td>
<td>83</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NSL</td>
<td>5 19 136 642 4019</td>
<td>5 18 95 310 650</td>
<td>650</td>
<td>83</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SecReT06</td>
<td>1 6 22 119 346</td>
<td>1 6 22 114 326</td>
<td>326</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SecReT07</td>
<td>6 20 140 635 4854</td>
<td>6 18 107 439 3335</td>
<td>3335</td>
<td>31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DH</td>
<td>1 14 38 151 816</td>
<td>1 12 12 56 128</td>
<td>128</td>
<td>84</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Experimental Results 2

<table>
<thead>
<tr>
<th>Protocol</th>
<th>none</th>
<th>Transition Subsumption</th>
<th>Super-lazy Intruder</th>
<th>All optimizations</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSPK</td>
<td>5 19</td>
<td>136 642 4021</td>
<td>5 15 61 107 237</td>
<td>4 6 4 2 1</td>
<td>99</td>
</tr>
<tr>
<td>NSL</td>
<td>5 19</td>
<td>136 642 4019</td>
<td>5 15 61 107 237</td>
<td>4 7 6 2 0</td>
<td>99</td>
</tr>
<tr>
<td>SecReT06</td>
<td>1 6</td>
<td>22 119 346</td>
<td>1 6 22 119 340</td>
<td>2 3 2 - -</td>
<td>99</td>
</tr>
<tr>
<td>SecReT07</td>
<td>6 20</td>
<td>140 635 4854</td>
<td>6 16 44 134 424</td>
<td>5 1 1 1 -</td>
<td>99</td>
</tr>
<tr>
<td>DH</td>
<td>1 14</td>
<td>38 151 816</td>
<td>1 14 38 138 525</td>
<td>4 6 10 9 12</td>
<td>99</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Protocol</th>
<th>none</th>
<th>Super-lazy Intruder</th>
<th>All optimizations</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSPK</td>
<td>5 19</td>
<td>136 642 4021</td>
<td>5 19 136 641 3951</td>
<td>1</td>
</tr>
<tr>
<td>NSL</td>
<td>5 19</td>
<td>136 642 4019</td>
<td>5 19 136 641 3949</td>
<td>2</td>
</tr>
<tr>
<td>SecReT06</td>
<td>1 6</td>
<td>22 119 346</td>
<td>1 6 22 119 340</td>
<td>2</td>
</tr>
<tr>
<td>SecReT07</td>
<td>6 20</td>
<td>140 635 4854</td>
<td>6 16 44 134 424</td>
<td>91</td>
</tr>
<tr>
<td>DH</td>
<td>1 14</td>
<td>38 151 816</td>
<td>1 14 38 138 525</td>
<td>35</td>
</tr>
</tbody>
</table>
Different from rewrite-rule based grammar behavior, because infinite behavior results from substitution
- Root term grows larger instead of leaf terms
- This behavior becomes more common as theories grow more complex
- Currently just cut off branch after a certain point
- But, would like a method that guarantees completeness
A Possible Path to a Solution

- Add more equations to the theory that make the search tree finite
- E.g., fix a bound $k$ and have $g^{X_1 \cdots X_k \cdot X_{k+1}} = g^{X_1 \cdots X_k}$
- Currently, this breaks soundness guarantees
- However, we can conjecture that, for $k$ large enough, the terms $g^{X_1 \cdots X_k}$ will be useless for finding attacks
  - How do we prove this?
- Bae at al. 2013 gives sufficient conditions, but requires that equations be topmost
  - This means equations can only be applicable to whole state
  - Not practical for us, terms like $g^{X_1 \cdots X_k}$ can appear anywhere in the state
  - What other conditions can we prove?
State Space Reduction References


- Kyungmin Bae, Santiago Escobar, José Meseguer: Abstract Logical Model Checking of Infinite-State Systems Using Narrowing. RTA 2013: 81-96