The Maude-NRL Protocol Analyzer
Lecture 1: Introduction to Maude-NPA

Catherine Meadows

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

Fifth Summer School on Formal Techniques, Menlo College, Atherton, CA, May 17-22, 2015

Purpose of These Lectures

- Introduce you to a particular protocol tool for crypto protocol analysis, Maude-NPA
  - Tool for automatic analysis of crypto protocols that takes into account equational theories of crypto operators
  - Based on unification and rewrite rules
- On the way, point out connections between research on the tool and open problems in crypto protocol analysis, rewriting logic, and unification
- Introduce you to two new related topics of research
  - Asymmetric Unification
  - Symbolic Indistinguishability

Outline

Outline

Example: Diffie-Hellman Without Authentication

1. $A \rightarrow B : g^{N_A}$
2. $B \rightarrow A : g^{N_B}$
3. A and B compute $g^{N_A * N_B} = g^{N_B * N_A}$

Well-known attack

1. $A \rightarrow I_B : g^{N_A}$
2. $I_A \rightarrow B : g^{N_I}$
3. $B \rightarrow I_A : g^{N_B}$
4. $I_B \rightarrow A : g^{N_I}$

- A thinks she shares $g^{N_I * N_A}$ with B, but she shares it with I
- B thinks he shares $g^{N_I * N_A}$ with A, but he shares it with I
- Commutative properties of $*$ and fact that $(G^X)^Y = G^{X*Y}$ crucial to understanding both the protocol and the attack

What This Example Illustrates

- There are many examples of cryptographic protocol failures that occur even when the crypto algorithms used are secure
- The attacks can be subtle and hard to find
  - Diffie and Hellman knew that their protocol would not be secure against an active attacker without authentication, but this particular attack was a surprise
- The attacks can make use of algebraic properties of the crypto-algorithm that are also needed to make the protocol work

Some Properties a Protocol Should Have

- Secrecy: If honest Alice thinks she has executed the protocol with honest Bob, an intruder should not know any secrets that were generated
- Authentication: If honest Alice thinks she has executed the protocol with honest Bob, then Bob should have executed the protocol with Alice
- Freshness: If honest Alice accepts a value as both fresh (not been used before) and shared with honest Bob, then it should not have been used before
- All of the above defined in terms of intruder knowledge and presence or absence of protocol executions
- Unauthenticated Diffie-Hellman fails Secrecy and Authentication, but satisfies freshness

"Dolev-Yao" Model for Automated Cryptographic Protocol Analysis

- Start with a signature, giving a set of function symbols and variables
- For each role, give a program describing how a principal executing that role sends and receives messages
- Give a set of inference rules the describing the deductions an intruder can make
  - E.g. if intruder knows K and $e(K, M)$, can deduce M
- Assume that all messages go through intruder who can
  - Stop or redirect messages
  - Alter messages
  - Create new messages from already sent messages using inference rules
- This problem well understood since about 2005

What We Know About Dolev-Yao

- Important notion: the "session"
  - A session is a single execution of a role in the protocol by a legitimate principal
  - Legitimate execution of unauthenticated DH and attack both involved two sessions
  - One for initiator and one for responder
- Known results
  - Secrecy undecidable in standard Dolev-Yao model (Durgin et al., 1998)
  - Secrecy NP-complete in standard DY model if number of sessions are bounded (bounded session model) (Rusinowitch and Turuani, 2001)
  - Similar results for authentication: both secrecy and authentication can be expressed in terms of reachability of states

Beyond the Free Algebra

- Crypto protocol analysis with the standard free algebra model (Dolev-Yao) well understood.
- But, not adequate to deal with protocols that rely upon algebraic properties of cryptosystems
  1. Cancellation properties, encryption-decryption
  2. Abelian groups
  3. Diffie-Hellman (exponentiation, Abelian group properties)
  4. Homomorphic encryption (distributes over an operator with also has algebraic properties, e.g. Abelian group)
  5. Etc. ..,
- In many cases, a protocol uses some combination of these

State of the Art When We Started this Research (mid 2000's)

- Free algebra model well understood
- Decidability results were beginning to appear for other theories
  - Similar to work for other theories
- Tools were beginning to appear that handled different theories
- Next step needed
  - Approach equational theories in a systematic way
  - Support combination of theories to the greatest extent possible

Goal of Maude-NPA

Provide tool that

- can be used to reason about protocols with different algebraic properties in the unbounded session model
- supports combinations of algebraic properties to the greatest degree possible

Our approach

- Use rewriting logic as general theoretical framework
  - crypto protocols are specified using rewrite rules
  - algebraic identities as equational theories
- Use narrowing modulo equational theories as a symbolic reachability analysis method
- Combine with state reduction techniques of Maude-NPA's ancestor, the NRL Protocol Analyzer (grammars, optimizations, etc.)
- Implement in Maude programming environment
  - Rewriting logic gives us theoretical framework and understanding
  - Maude implementation gives us tool support

Maude-NPA

- A tool to find or prove the absence of attacks using backwards search
- Analyzes infinite state systems
  - Active intruder
  - No abstraction or approximation of nonces
  - Unbounded number of sessions
- Intruder and honest protocol transitions represented using strand space model.
- So far supports a number of equational theories: cancellation (e.g. encryption-decryption), AC, exclusive-or, Diffie-Hellman, bounded associativity, homormorphic encryption over a free theory, various combinations, working on including more

Outline

A Little Background on Unification

- Given a signature Σ and an equational theory E, and two terms s and t built from Σ:
- A unifier of $s =_E ?t$ is a substitution σ to the variables in s and t s.t. σs can be transformed into σt by applying equations from E to σs and its subterms
- Example: $\Sigma = \{d/2, e/2, m/0, k/0\}, E = \{d(K, e(K, X)) = X\}$. The substitution $\sigma = \{Z \mapsto e(T, Y)\}$ is a unifier of $d(K, Z)$ and Y.
- The set of most general unifiers of $s =?t$ is the set Γ s.t. any unifier σ is of the form $\rho\tau$ for some ρ, and some τ in Γ.
- Example: $\{Z \mapsto e(T, Y), Y \mapsto d(T, Z)\}$ mgu's of $d(T, Z)$ and Y.
- Given the theory, can have:
  - at most one mgu (empty theory)
  - a finite number (AC)
  - an infinite number (associativity)
- Unification problem in general undecidable

Rewriting Logic in a Nutshell

A rewrite theory $\mathcal{R}$ is a triple $\mathcal{R} = (\Sigma, E, R)$, with:

- Σ a signature
- (Σ, R) a set of rewrite rules of the form $t \rightarrow s$
  e.g. $e(K_A, N_A; X) \rightarrow e(K_B, X)$
- E a set of equations of the form $t = s$
  e.g. $d(K, e(K, Y)) = Y$

Intuitively, $\mathcal{R}$ specifies a concurrent system,
whose states are elements of the initial algebra $T_{\Sigma/E}$ specified by (Σ, E), and
whose concurrent transitions are specified by the rules R.
Narrowing gives us the rules for executing transitions concurrently.

Narrowing and Backwards Narrowing

Narrowing: $t \leadsto_{\sigma,R,E} s$ if there is

- a non-variable position $p \in Pos(t)$;
- a rule $l \to r \in R$;
- a unifier σ (modulo E) of $t|p =_E ?l$ such that $s = \sigma(t[r]_p)$.

Example:

- $R = \{\ X \to d(k, X)\ \}$, $E = \{\ d(K, e(K, Y)) = Y\ \}$
- $e(k,\ t) \leadsto_{\emptyset,R,E} d(k, e(k, t)) =_E t$

Backwards Narrowing: narrowing with rewrite rules reversed

A Warning About Narrowing

- Full narrowing (narrowing in every possible non-variable location) is often inefficient and even nonterminating
- We need to construct our rewrite systems so that efficient narrowing strategies can be chosen
- Maude-NPA has led to some major advances in this area

Narrowing Reachability Analysis

Narrowing can be used as a general deductive procedure for solving reachability problems of the form

$$(\exists \vec{x})\ t_1(\vec{x}) \rightarrow^* t'_1(\vec{x})\ \wedge \ldots \wedge\ t_n(\vec{x}) \rightarrow^* t'_n(\vec{x})$$

in a given rewrite theory, where the terms t_i and t'_i denote sets of states.

- The terms t_i and t'_i denote sets of states.
- For what substitutions σ are $t_i(\sigma x)$ reachable from $t'_i(\sigma x)$
- No finiteness assumptions about the state space.
- Maude-NPA rewrite system supports topmost narrowing for state reachability analysis
  - Rewrite rules apply to topmost position only, so narrowing steps only need to be applied to entire state
  - Topmost narrowing complete

E-Unification

- In order to apply narrowing to search, need an E unification algorithm
- Two approaches:
  1. Built-in unification algorithms for each theory
  2. Hybrid approach with $E = R \uplus \Delta$
- Hybrid Approach
  - Δ has built-in unification algorithm
  - R confluent, terminating, and coherent rules modulo B
    - Confluent: Always reach same normal form modulo B, no matter in which order you apply rewrite rules
    - Terminating: Sequence of rewrite rules is finite
    - Coherent: Technical condition on equations needed to make narrowing on representatives of B-equivalence classes complete
    - Non-coherent equational theories can often be made coherent by adding extra (redundant) equations
- Lets us use R, Δ narrowing as a general method for E-unification

How R, Δ Narrowing Works for Unification

1. Start with a problem $s =? t$ and $U = \emptyset$.
2. Find a set of Δ-mgu's of $s =? t$ add these to U.
3. For each non-variable position p in $s =? t$ such that $s =? t|_p$ and each rewrite rule $r \to \ell \in R$, find a set of Δ-mgu's of $s|_p =? r$
4. For each unifier σ of $s|_p$ and r found in Step 3, create the problem $\sigma s|_p[\ell] = \sigma r$
5. Solve $\sigma s|_p[\ell] = \sigma r$ using Steps 1 through 4, for each unifier τ found, add $\tau\sigma$ to U.

Example

- $\Sigma = \{e/2, d/3\}$, $R = \{d(K, e(K, X) \rightarrow X\}$, $\Delta = \emptyset$
- $d(W, Z) =?Y$
- First solution $Y/d(W, Z)$
- Second Solution: $d(W, Z)$ unifies with $d(K, e(K, X))$ via $\sigma W/e(K, X)$
- New problem is $X =?Y$, solution is $\tau = X/Y$, given unifier $\tau\sigma = W/e(K, Y)$
- Only variable positions left, so we are done with two unifiers
- We were lucky: narrowing often doesn't terminate!
- Will later discuss theories for which narrowing can terminate and strategies for achieving termination and soundness

Outline

Outline

Uses Strand Space Notation

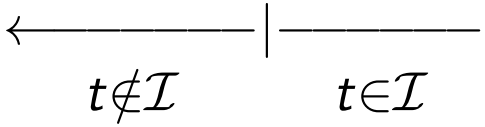
- Strand spaces: popular model introduced by Thayer, Herzog, and Guttman
- Each local execution, or session of an honest principal represented by sequence of positive and negative terms called a strand.
  - Terms made up of variables and function symbols
  - Negative term stand for received message, positive terms stand for sent messages
  - Example: $[+(pke(B, N_A; A)),\ -(pke(A, N_A; N_B)),\ +(pke(B, N_B))]$
- Each intruder computation also represented by strand
  - Example: $[-(X), +(pke(A, X))]$

Basic Structure of Maude-NPA

- Uses modified strand space model
- Each local execution and each intruder action represented by a strand, plus a marker denoting the current state
  - Searches backwards through strands from final state
  - Set of rewrite rules governs how search is conducted
  - Sensitive to past and future
- Grammars used to prevent infinite loops
- Learn-only-once rule says intruder can learn term only once
- When an intruder learns term in a backwards search, tool keeps track of this and doesn't allow intruder to learn it again
- Other optimization techniques used to reduce other infinite behavior and to cut down size of search space

What We Need to Represent

- Maude-NPA's use of backwards search means we have incomplete picture of what intruder learned in past. But we need the concrete moment when the intruder learns something:
- Notion of the present
  - What the intruder knows in the present (i.e., $t \in \mathcal{I}$)
  - Where the honest principals are in the present (strands)
- Notion of the future
  - What terms the intruder will learn in the future (i.e., $t \notin \mathcal{I}$)



How Protocols Are Specified in Maude-NPA

- Represent protocols and intruder actions using strands
- Terms in strands obey an equational theory specified by the user
- Terms in strands of different sorts, mostly defined by user
- Special sort Fresh
  - Terms of sort *Fresh* are always constant (used by nonces)
  - Strand annotated with fresh terms generated by the strand

$:: r :: [+(pke(B, n(A, r); A)), -(pke(A, n(A, r); NB)), +(pke(B, NB))]$

The Notion of State in NPA Strands

- A state is a set of strands plus the intruder knowledge (i.e., a set of terms)
  1. Each strand is divided into past and future
     $[\ m_1{}^{\pm},\ \ldots,\ m_i{}^{\pm} \mid m_{i+1}{}^{\pm},\ \ldots,\ m_k{}^{\pm}\ ]$
  2. Initial strand $[\ nil \mid m_1^{\pm},\ \ldots,\ m_k^{\pm}\ ]$, final strand $[\ m_1^{\pm},\ \ldots,\ m_k^{\pm} \mid nil\ ]$
  3. The intruder knowledge contains terms $m \notin \mathcal{I}$ and $m \in \mathcal{I}$
     $\{\ t_1 \notin \mathcal{I}, \ldots, t_n \notin \mathcal{I},\ s_1 \in \mathcal{I}, \ldots, s_m \in \mathcal{I}\ \}$
  4. Initial intruder knowledge $\{\ t_1 \notin \mathcal{I}, \ldots, t_n \notin \mathcal{I}\ \}$, final intruder knowledge $\{\ s_1 \in \mathcal{I}, \ldots, s_m \in \mathcal{I}\ \}$

Specifying the Protocol: Sorts

```
--- Sort Information
  sorts Name Nonce NeNonceSet Gen Exp Key
     GenvExp Secret .
  subsort Gen Exp  < GenvExp .
  subsort Name NeNonceSet GenvExp Secret Key < Msg .
  subsort Exp < Key .
  subsort Nonce < NeNonceSet .
  subsort Name < Public . --- necessary
  subsort Gen < Public . --- necessary
```

- Sorts Msg, Fresh, and Public, are built-in
- There always has to be a space before a period!

Specifying the Protocol: Function Symbols

```
--- Nonce
 op n : Name Fresh -> Nonce [frozen] .
 --- Multiplication
  subsort Nonce < NeNonceSet .
  op _*_ : NeNonceSet NeNonceSet ->
     NeNonceSet [frozen assoc comm] .
  --- Concatenation
  op _;_ : Msg Msg -> Msg [frozen gather (e E)] .
```

- Note that associativity-commutativity must be specified with the function symbol
  - Rewrite-Rule based theories are specified elsewhere
- gather (e E) means the symbol is right-associative: e.g. a;b;c = a;(b;c)
- The term [frozen] must appear at the end of each definition

Specifying the Protocol: Equational Theories

```
eq exp(exp(W:Gen,Y:NeNonceSet),Z:NeNonceSet)
  = exp(W:Gen, Y:NeNonceSet * Z:NeNonceSet) [variant] .
 eq e(K:Key,d(K:Key,M:Msg)) = M:Msg [variant] .
 eq d(K:Key,e(K:Key,M:Msg)) = M:Msg [variant] .
```

- The term [variant] needs to be after each equational rule
- K:Key means that K is a variable of sort key
  - You can also declare the sort of K beforehand

Specifying the Intruder (Doley-Yao) Rules

```
eq STRANDS-DOLEVYAO =
    :: nil :: [ nil | -(M1 ; M2), +(M1), nil ] &
    :: nil :: [ nil | -(M1 ; M2), +(M2), nil ] &
    :: nil :: [ nil | -(M1), -(M2), +(M1 ; M2), nil ] &
    :: nil :: [ nil | -(Ke), -(M), +(e(Ke,M)), nil ] &
    :: nil :: [ nil | -(Ke), -(M), +(d(Ke,M)), nil ] &
    :: nil :: [ nil | -(NS1), -(NS2), +(NS1 * NS2), nil ] &
    :: nil :: [ nil | -(GE), -(NS), +(exp(GE,NS)), nil ] &
    :: r ::   [ nil | +(n(i,r)), nil ] &
    :: nil :: [ nil | +(g), nil ] &
    :: nil :: [ nil | +(A), nil ]
 [nonexec] .
```

- DY Rules specify how the intruder constructs and deconstructs terms
- DY rules separated by the symbol "&"
- In this example, sorts of variables declared beforehand

Specifying the Intruder (Doley-Yao) Rules

```
vars NS1 NS2 NS3 NS : NeNonceSet .  ...
eq STRANDS-DOLEVYAO =
    :: nil :: [ nil | -(M1 ; M2), +(M1), nil ] &
    :: nil :: [ nil | -(M1 ; M2), +(M2), nil ] &
    :: nil :: [ nil | -(M1), -(M2), +(M1 ; M2), nil ] &
    :: nil :: [ nil | -(Ke), -(M), +(e(Ke,M)), nil ] &
    :: nil :: [ nil | -(Ke), -(M), +(d(Ke,M)), nil ] &
    :: nil :: [ nil | -(NS1), -(NS2), +(NS1 * NS2), nil ] &
    :: nil :: [ nil | -(GE), -(NS), +(exp(GE,NS)), nil ] &
    :: r ::   [ nil | +(n(i,r)), nil ] &
    :: nil :: [ nil | +(g), nil ] &
    :: nil :: [ nil | +(A), nil ]
 [nonexec] .
```

- DY Rules specify how the intruder builds terms
- DY rules separated by the symbol "&"
- In this example, sorts of variables declared beforehand
  - Scope is both Dolev-Yao and honest principal strands

Specifying the Honest Principals

```
eq STRANDS-PROTOCOL =
    :: r,r' ::
    [nil | +(A ; B ; exp(g,n(A,r))),
           -(A ; B ; XE),
           +(e(exp(XE,n(A,r)),sec(A,r'))), nil] &
    :: r ::
    [nil | -(A ; B ; XE),
           +(A ; B ; exp(g,n(B,r))),
           -(e(exp(XE,n(B,r)),Sr)), nil]
 [nonexec] .
```

- Fresh variables generated by a principal declared at beginning
- Terms received by principal whose structure can't be verified represented by variables
- If assume principal can verify sort, can specify variable to be of desired sort

Specifying Attack States

```
  eq ATTACK-STATE(0)
 = :: r ::
   [nil, -(a ; b ; XE),
         +(a ; b ; exp(g,n(b,r))),
         -(e(exp(XE,n(b,r)),sec(a,r'))) | nil]
   || sec(a,r') inI || nil || nil || never
   *** Pattern for authentication
   (:: R:FreshSet ::
   [nil | +(a ; b ; XE),
          -(a ; b ; exp(g,n(b,r))),
          +(e(YE,sec(a,r'))), nil]
    & S:StrandSet || K:IntruderKnowledge)
[nonexec] .
```

- Attack states specify insecure states you want to prove unreachable

Anatomy of an Attack State

1. First part: Strands that are executed in the attack:

```
:: r :: [nil, -(a ; b ; XE), +(a ; b ; exp(g,n(b,r))),
          -(e(exp(XE,n(b,r)),sec(a,r'))) | nil]}
```

2. Intruder knowledge section saying what terms the intruder learns: `sec(a,r') inI`
3. "Never patterns" saying what strands should not be executed in the attack
4. If a state contains an instance of a never pattern, it is discarded as unreachable

```
never
      *** Pattern for authentication
      (:: R:FreshSet ::
      [nil | +(a ; b ; XE),  -(a ; b ; exp(g,n(b,r))),
             +(e(YE,sec(a,r'))), nil]
        & S:StrandSet || K:IntruderKnowledge)
```

Two Uses of Never Patterns

1. To define authentication properties
   - To specify an authentication attack property: Can X happen without Y happening first?
   - This was done in previous example
2. To narrow the search space
   - Tell Maude-NPA to ignore states that contain components that lead to state explosion
   - If you have previously proven never pattern unreachable, this does not affect completeness
   - If you have not, it can still help in searching for attacks

Examples of State-Space Reduction Never Patterns

```
|| never(
    *** Avoid infinite useless path
     (:: nil ::
     [ nil | -(exp(GE,NS1 * NS2)), -(NS3),
         +(exp(GE,NS1 * NS2 * NS3)), nil ]
     & S:StrandSet || K:IntruderKnowledge)
    *** Pattern to avoid unreachable states
    (:: nil ::
     [nil | -(exp(#1:Exp, N1:Nonce)),
            -(sec(A:Name, #2:Fresh)),
            +(e(exp(#1:Exp, N2:Nonce),
            sec(A:Name, #2:Fresh))), nil]
     & S:StrandSet || K:IntruderKnowledge)
```

- You can specify as many never patterns as you like

Outline

How Maude-NPA Uses Narrowing

- What We Need
  - A set of rewrite rules describing protocol execution
  - A narrowing strategy
  - A unification algorithm
    - To use in unifying left-hand side of rewrite rules with sub terms

Maude-NPA's Strategy: Top-Most Narrowing

- Essential idea - suppose the following:
  - If a rewrite rule $\ell \rightarrow r$ can be applied to a term t, it can be applied to the top of t (ℓ unified with t)
  - Applying a rule to a proper subterm t is either not possible or is redundant
- Assume also E has a finite complete unification algorithm
- In that case, (E, R) top-most narrowing (left-hand side of rewrite rule unified with the entire term) is sound and complete
- Moreover, narrowing only at the top is more efficient

Terms of Sort *State* and Rewrite Rules in Maude-NPA

SS & s_1 & ... & s_k & $\{m_1 \in \mathcal{I}, \ldots, m_n \in \mathcal{I}, n_1 \notin \mathcal{I}, \ldots, n_s \notin \mathcal{I}, IK\}$

- _&_ is AC and has arguments of sort Strand Space
- _,_ is AC and has arguments of sort Intruder Facts
- Two types of rewrite rules
  - Those that describe how strands already in the state behave
  - Those that describe how new strands enter that state

Protocol Rules and Their Execution With Strands Already in State

To execute a protocol $\mathcal{P}$ associate to it a rewrite theory on sets of strands as follows. Let $\mathcal{I}$ informally denote the set of terms known by the intruder, and K the facts known or unknown by the intruder

1. $[\ L \mid M^{-}, L'\ ]\ \&\ \{M{\in}\mathcal{I}, K\} \rightarrow [\ L, M^{-} \mid L'\ ]\ \&\ \{M{\in}\mathcal{I}, K\}$
   Moves input messages into the past
2. $[\ L \mid M^{+}, L'\ ]\ \&\ \{K\} \rightarrow [\ L, M^{+} \mid L'\ ]\ \&\ \{K\}$
   Moves output message that are not read into the past
3. $[\ L \mid M^{+}, L'\ ]\ \&\ \{M{\notin}\mathcal{I}, K\} \rightarrow [\ L, M^{+} \mid L'\ ]\ \&\ \{M{\in}\mathcal{I}, K\}$
   Joins output message with term in intruder knowledge.

For backwards execution, just reverse

Introducing New Strands

- If we want an unbounded number of strands, need some way of introducing new strands in the backwards search
- Specialize rule r1 using each strand $[\ l_1,\ u^+,\ l_2\ ]$ of the protocol $\mathcal{P}$:

$$[\ l_1 \mid u^+] \ \&\ \{u \notin \mathcal{I}, K\} \rightarrow \{u \in \mathcal{I}, K\}$$

- Gives us a natural way of switching between bounded and unbounded sessions
  - Put a bound on the number of times r3 could be invoked with non-intruder strands

Reachability Analysis

- Backwards narrowing protocol execution defines a backwards reachability relation $St \rightsquigarrow^*_{\mathcal{P}} St'$
- In initial step, prove lemmas that identify certain states unreachable
- Specify a state describing the attack state, including a set of final strands plus terms $m \notin \mathcal{I}$ and $m \in \mathcal{I}$
- Execute the protocol backwards to an initial state, if possible
- For each intermediate state found, check if it has been proved unreachable and discard if it is

Outline

What Is Needed in an E-Unification Algorithm for Crypto Protocol Analysis

- Should apply to a wide class of theories
- In particular, should apply to AC theories: Abelian Groups, Diffie-Hellman, exclusive-or
- Should be easy to combine: protocols often combine different algorithms that obey different equational theories
- Should be reasonably efficient
- The solution: variant unification

Variants

- Let $(\Sigma, R \uplus \Delta)$ be a an equational theory where
  - Δ is regular (if a variable appears on one side of an equation, then it appears on the other
  - R is a set of rewrite rules confluent, terminating and coherent wrt Δ.
- Let t be a term. A variant of tt is a pair $(\theta, \theta t \downarrow)$.
- A set of most general variants of $t$is a set of variants V such that for every variant $(\theta, \theta t \downarrow)$ there is a variant $(\sigma t \downarrow) \in V$ such that $\theta t \downarrow = \tau(\sigma t \downarrow)$.
- Example: Find a set of most general variants $d(Y, Z)$ in $(\{e/2.d/2\}, \{d(K, e(K, X)) \rightarrow W\}, \emptyset)$
  - We already know how to do it : variant narrowing!
  - You find it by solving $d(Y, Z) =? W$ using narrowing.
  - The set is $\{(\iota, d(Y, Z)), (Z/e(K, X), X)\}$

The Finite Variant Property

- A theory decomposition $(\Sigma, E = R \uplus \Delta)$ has the finite variant property if
  - It satisfies all the conditions described on the previous slide
  - Every term has a finite most general set of variants.
- Concept of variants and finite variants originally due to (Comon and Delaune, 2005); we use a slightly stronger definition of variant due to (Escobar, Sasse, Escobar, 2008)
- Variants and the finite variant property originally introduced to characterize theories for which there exist sound, complete, and terminating narrowing strategies

What Has the Finite Variant Property, and What Hasn't

- Finite Variant Theories
  - $\mathbf{Z}_n, \quad \Delta = \text{AC}$
  - $R = \{exp(exp(X, Y), Z)) = exp(X, Y * Z)\} \quad , \Delta = \text{AC}$ (for *)
  - Various forms of cancellation of encryption-decryption
- Non Finite Variant Theory
  - h homomorphic over operator * (doesn't matter whether it's free or AC)
  - If $t = h(X)$, then $\Sigma = \{\iota, X/X_1 * X_2, X/X_1 * X_2 * X_3, \ldots\}$
- Sufficient and/or necessary conditions studied by Escobar, Sasse, and Meseguer, 2008
- A method for using narrowing to compute a most general set of variants, guaranteed to terminate for theories with the finite variant property, is given in (Escobar, Sasse, and Meseguer, JLAP 2011)

Variant Unification

- Suppose E with a finite variant decomposition (Δ, R)
- To find a most general set of unifiers $s =_E ? t$:
  - First find a complete set of R-variants $\sigma_1 s \downarrow, \ldots, \sigma_k s$ of s and $\tau_1 t, \ldots, \tau_n t$ of t
  - Then find find a most general set of unifiers of $\sigma_i s =_\Delta \tau_j t$
  - The set of solutions will be $\{\ldots, \rho_{ijk}\sigma_i\tau_j \ldots\}$, where the ρ_{ijk} are the solutions of $\sigma_i s =_\Delta \tau_j t$

Using Narrowing to Find Variants

- Let $(\Sigma, E, \mathcal{X})$ be an algebra where $E = (R \uplus \Delta)$
- To find all the R variants of a term t, construct a narrowing tree for (t, ι)
- For each leaf in the tree (σ, s) find each substitution ρ and term r such that $s \leadsto_\rho r'$
- If $(\rho\sigma, r)$ doesn't already appear in the tree, add it as a child of (σ, s)
- If E has the finite variant property, then this will terminate with all variants found (Escobar et al., 2011)

Example : Solve $X \oplus a =? Y \oplus b$ where $\oplus$ = exclusive-or

$X \oplus a / Y \oplus b$	$(X \oplus a, \iota)$	$(Z, X \mapsto Z \oplus a)$	$(a, X \mapsto 0)$	$(0, X \mapsto a)$
$(Y \oplus b, \iota)$	$X \mapsto Y$	$Z \mapsto Y \oplus b$	none	none
$(W, Y \mapsto W \oplus b)$	$W \mapsto X \oplus a$	$W \mapsto Z$	$W \mapsto a$	$W \mapsto 0$
$(b, Y \mapsto 0)$	none	$Z \mapsto b$	none	none
$(0, Y \mapsto b)$	none	$Z \mapsto 0$	none	none

Table : AC-unifiers of variants

$X \oplus a / Y \oplus b$	$(X \oplus a, \iota)$	$(Z, X \mapsto Z \oplus a)$	$(a, X \mapsto 0)$	$(0, X \mapsto a)$
$(Y \oplus b, \iota)$	$X \mapsto Y$	$X \mapsto Y \oplus a \oplus b$	none	none
$(W, Y \mapsto W \oplus b)$	$Y \mapsto X \oplus a \oplus b$	$Y \mapsto Z \oplus b$, $X \mapsto Z \oplus a$	$X \mapsto 0$, $Y \mapsto a \oplus b$	$X \mapsto a$, $Y \mapsto b$
$(b, Y \mapsto 0)$	none	$X \mapsto a \oplus b$ $Y \mapsto 0$	none	none
$(0, Y \mapsto b)$	none	$X \mapsto a, Y \mapsto b$	none	none

Table : Solutions to $X \oplus a =? Y \oplus b$ derived from AC-unifiers of variants

Limitations of Variant Unification

- Variant (R, Δ) unification generates many Δ unification problems
- Many of these don't have solutions
- Many have equivalent solutions
- So you waste a lot of time checking redundant and unsolvable problems
- In the next lecture we'll talk about a possible better way

Maude-NPA References

- Maude-NPA 2.0 and relevant papers available at `http://maude.cs.uiuc.edu/tools/Maude-NPA/` . We are using an advance version of Maude-NPA 3.0 that has additional features and improved performance.
- S. Escobar, C. Meadows, J. Meseguer. Maude-NPA: Cryptographic Protocol Analysis Modulo Equational Properties. FOSAD 2007/2008/2009 Tutorial Lectures, LNCS 5705, pages 1-50. Springer-Verlag, 2009.

Variant Unification References

- H. Comon-Lundh and S. Delaune. The finite variant property: How to get rid of some algebraic properties. In RTA'05, LNCS 3467, pages 294-307. Springer, 2005.
- Santiago Escobar, José Meseguer, Ralf Sasse. Effectively Checking or Disproving the Finite Variant Property In proceedings of 19th International Conference on Rewriting Techniques and Applications (RTA 2008), LNCS 5117, pages 79-93. 2008.
- Santiago Escobar, Ralf Sasse, José Meseguer. Folding variant narrowing and optimal variant termination. The Journal of Logic and Algebraic Programming, 2011.