ProVerif

Dale Vaillancourt

Outline

- Logic Programming Review
- Modeling Protocols
- Verification of Secrecy
- Demo
- ProVerif Extensions

Logic Programming

- Data given by terms.

```
M ::= X | Y | ...            ;variables
    | a | b | ...            ;names
    | f(M, ..., M)           ;constructors
```

$f \in \{$ nil/0, cons/2 $\}$

Logic Programming

- Data given by terms.

```
M ::= X | Y | ...          ;variables
    | a | b | ...          ;names
    | f(M, ..., M)         ;constructors
```

$f \in \{$ nil/0, cons/2 $\}$

nil

Logic Programming

- Data given by terms.

```
M ::= X | Y | ...            ;variables
    | a | b | ...            ;names
    | f(M, ..., M)           ;constructors
```

$f \in \{$ nil/0, cons/2 $\}$

```
nil              cons(a, nil)
```

Logic Programming

- Data given by terms.

```
M ::= X | Y | ...           ;variables
    | a | b | ...           ;names
    | f(M, ..., M)          ;constructors
```

f ∈ { nil/0, cons/2 }

```
nil                cons(a, nil)


   cons(a, cons(b, nil))
```

Logic Programming

- Behavior given by relations and rules.

```
R ::= r(M, ..., M)      ;relation app
H ::= R ∧ ... ∧ R → R ;rule
r ∈ { reverse/2, member/2, append/3 }
```

Logic Programming

- Behavior given by relations and rules.

```
R ::= r(M, ..., M)      ;relation app
H ::= R ∧ ... ∧ R → R ;rule
r ∈ { reverse/2, member/2, append/3 }


      → reverse(nil, nil)

      reverse(L, RL) ∧
      append(RL, cons(X,nil), A)
      → reverse(cons(X,L), A)
```

Logic Programming

- Behavior given by relations and rules.

```
R ::= r(M, ..., M)       ;relation app
H ::= R ∧ ... ∧ R → R ;rule
r ∈ { reverse/2, member/2, append/3 }
```

$$\frac{}{reverse(nil, nil)}$$

$$\frac{reverse(L, RL) \qquad append(RL, cons(X, nil), A)}{reverse(cons(X, L), A)}$$

Logic Programming

$$\dfrac{\dfrac{\dfrac{}{rev(nil, Z)} \qquad \dfrac{\dfrac{}{app(nil, b : nil, Y)}}{app(Z, b : nil, Y)}}{rev(b : nil, Y)} \qquad \dfrac{\dfrac{\vdots}{app(b : nil, a : nil, X)}}{app(Y, a : nil, X)}}{rev(a : b : nil, X)}$$

Modeling Protocols

- Secrecy: What does the adversary "know"?
- Need rules to deduce adversary's knowledge.

Modeling Protocols

- Secrecy: What does the adversary "know"?
- Need rules to deduce adversary's knowledge.

$$\frac{\vdots}{adversary(secret)}$$

Modeling Protocols

- Secrecy: What does the adversary “know”?
- Need rules to deduce adversary’s knowledge.

Some Syntax

$$
\begin{array}{lcll}
M & = & X \mid Y \mid \ldots & ;vars \\
 & \mid & a[M,\ldots,M] & ;names \\
 & \mid & f(M,\ldots,M) & ;constructors \\
R & = & r(M,\ldots,M) & ;relations \\
H & = & R \wedge \cdots \wedge R \rightarrow R & ;rules
\end{array}
$$

sk_A

$a[pk(sk_B)]$

$pk(sk_A)$

$a[pk(sk_B), \langle m[] \rangle_{sk_A}]$

$\{M\}_{sk_A}$

$\langle M \rangle_{sk_A}$

Names and Freshness

- Suppose we have $a[]$.

  Then we can generate "fresh" values.

Dolev-Yao Rules

- Captures adversary, independent of protocol.

$$\frac{adversary(\{M\}_k) \qquad adversary(k)}{adversary(M)}$$

$$\frac{adversary(\langle M\rangle_{sk_A})}{adversary(M)} \qquad \frac{adversary(M) \qquad adversary(k)}{adversary(\langle M\rangle_k)}$$

$$\frac{}{adversary(pk(sk_A))}$$

$$\vdots$$

Capturing Protocol Steps

$$A \rightarrow B \quad : \quad \{\langle k \rangle_{sk_A}\}_{pk_B}$$
$$B \rightarrow A \quad : \quad \{s\}_k$$

$$\frac{}{adversary(\{\langle k \rangle_{sk_A}\}_{pk(sk_B)})} \qquad \frac{}{adversary(\{s\}_k)}$$

Capturing Protocol Steps

$$
\begin{aligned}
A \rightarrow B \quad &: \quad \{\langle k \rangle_{sk_A}\}_{pk_B} \\
B \rightarrow A \quad &: \quad \{s\}_k
\end{aligned}
$$

$$
\frac{}{adversary(\{\langle k \rangle_{sk_A}\}_{pk(sk_B)})}
\qquad
\frac{adversary(\{\langle k \rangle_{sk_A}\}_{pk(sk_B)})}{adversary(\{s\}_k)}
$$

Capturing Protocol Steps

$$
\begin{aligned}
A \rightarrow B &\quad : \quad \{\langle k \rangle_{sk_A}\}_{pk_B} \\
B \rightarrow A &\quad : \quad \{s\}_k
\end{aligned}
$$

$$\frac{adversary(pk(X))}{adversary(\{\langle k \rangle_{sk_A}\}_{pk(X)})}$$

$$\frac{adversary(\{\langle k \rangle_{sk_A}\}_{pk(sk_B)})}{adversary(\{s\}_k)}$$

```
(* Denning Sacco Original *)
pred c/1 elimVar,decompData.
nounif c:x.

fun pk/1.  fun encrypt/2.  fun sign/2.

query c:secret[].

reduc (* The DY attacker *)
c:c[];
c:pk(sA[]);
c:pk(sB[]);

c:x & c:encrypt(m,pk(x)) -> c:m;
c:x -> c:pk(x);
c:x & c:y -> c:encrypt(x,y);
c:sign(x,y) -> c:x;
c:x & c:y -> c:sign(x,y);

(* The protocol rules *)
(* A *)
c:pk(x) -> c:encrypt(sign(k[pk(x)], sA[]), pk(x));

(* B *)
c:encrypt(sign(k, sA[]), pk(sB[])) -> c:encrypt(secret[], pk(k)).
```

Approximations

- Protocol steps can be repeated.
- Freshness is strange.
- What are we approximating anyway?

More Precisely

- Linear Logic model by Durgin et al.
  - Use “resource aware” connectives to model states of participants.

    $$B_0 \otimes recv(M) \multimap send(M') \otimes B_1$$

  - Use existential to model freshness.

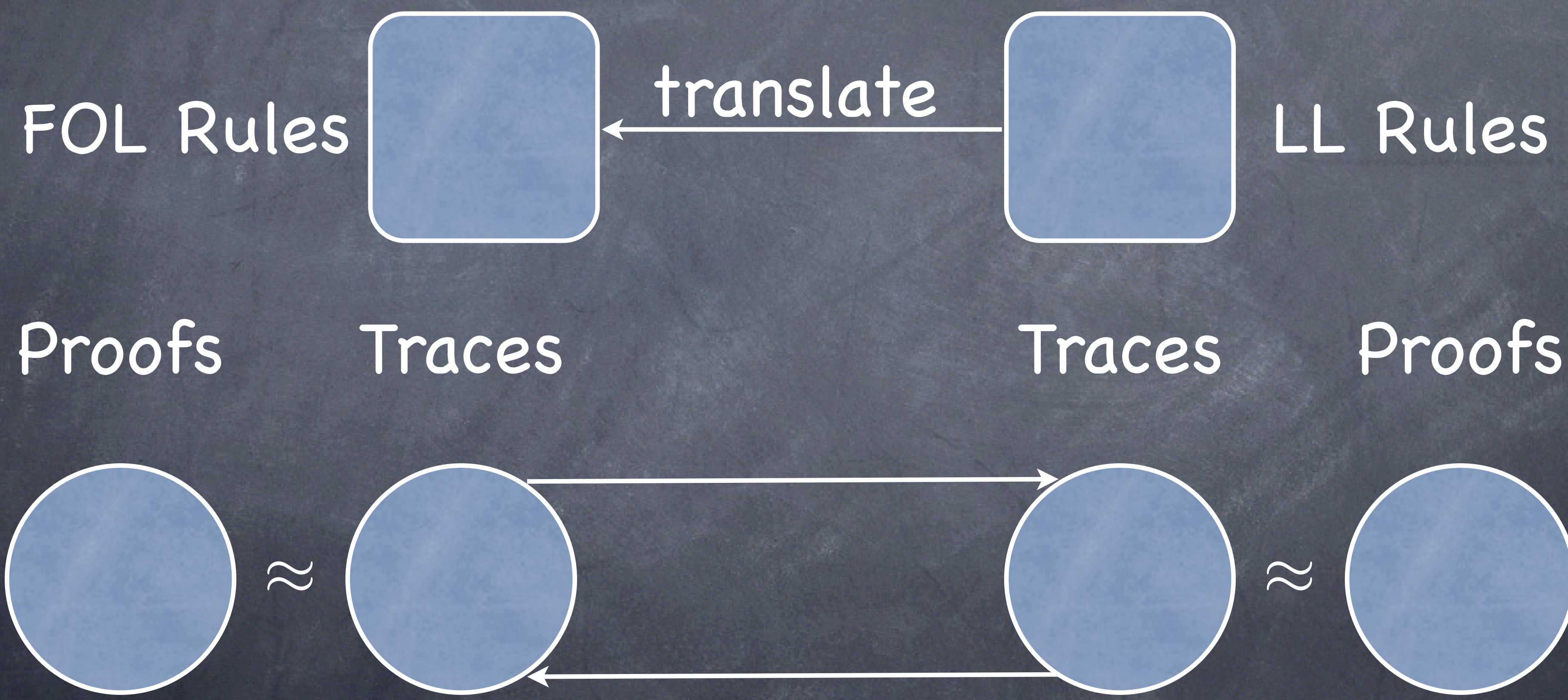
Traces

$$\frac{}{A} \qquad \frac{}{(A \otimes A) \multimap B} \qquad \frac{A \qquad A \multimap B}{B} \qquad \frac{A \qquad B}{A \otimes B}$$

$$\begin{aligned}
\emptyset &\mapsto \{A\} \\
&\mapsto \{A, A\} \\
&\mapsto \{A \otimes A\} \\
&\mapsto \{A \otimes A, (A \otimes A) \multimap B\} \\
&\mapsto \{B\}
\end{aligned}$$

Safety

- Theorem: FOL model is safe.



Verification

- Now we can just run the program:

$$adversary(secret[])$$

- Prolog will tell us if it fails or succeeds!

Example

$$\frac{adversary(\{M\}_k) \qquad adversary(k)}{adversary(M)}$$

Example

$$\dfrac{\dfrac{\dfrac{\vdots}{adversary(\{\{s\}_k\}_{k'})} \qquad \dfrac{\vdots}{adversary(k')}}{adversary(\{s\}_k)} \qquad \dfrac{\vdots}{adversary(k)}}{adversary(s)}$$

$$\dfrac{adversary(\{M\}_k) \qquad adversary(k)}{adversary(M)}$$

Be More Clever!

- Need to avoid selecting rules concluding:

$$adversary(X)$$

- Standard extension to Prolog's algorithm: Resolution with Selection.

```
(* Denning Sacco Original *)
pred c/1 elimVar,decompData.
nounif c:x.

fun pk/1.  fun encrypt/2.  fun sign/2.

query c:secret[].

reduc (* The DY attacker *)
c:c[];
c:pk(sA[]);
c:pk(sB[]);

c:x & c:encrypt(m,pk(x)) -> c:m;
c:x -> c:pk(x);
c:x & c:y -> c:encrypt(x,y);
c:sign(x,y) -> c:x;
c:x & c:y -> c:sign(x,y);

(* The protocol rules *)
(* A *)
c:pk(x) -> c:encrypt(sign(k[pk(x)], sA[]), pk(x));

(* B *)
c:encrypt(sign(k, sA[]), pk(sB[])) -> c:encrypt(secret[], pk(k)).
```

```
(* Denning Sacco Original *)
pred c/1 elimVar,decompData.
nounif c:x.

fun pk/1.  fun encrypt/2.  fun sign/2.

query c:secret[].

reduc (* The DY attacker *)
c:c[];
c:pk(sA[]);
c:pk(sB[]);

c:x & c:encrypt(m,pk(x)) -> c:m;
c:x -> c:pk(x);
c:x & c:y -> c:encrypt(x,y);
c:sign(x,y) -> c:x;
c:x & c:y -> c:sign(x,y);

(* The protocol rules *)
(* A *)
c:pk(x) -> c:encrypt(sign(k[pk(x)], sA[]), pk(x));

(* B *)
c:encrypt(sign(k, sA[]), pk(sB[])) -> c:encrypt(secret[], pk(k)).
```

Pros and Cons

- Fast Analysis.
- Fully Automatic.
- Does not limit number of sessions.
- Produces proof of attack when discovered.

- May not terminate.
- False Negatives.

Extensions

- Terminate in more cases.
- Optimize.
- Analyze authentication properties.
- Use computational model for adversary.

Fin

- Blanchet, B. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In CSFW'01.
- Blanchet, B. An Efficient Crytographic Protocol Verifier Based on Logic Programming. (Tech Rept of above.)
- Blanchet, B. From Secrecy to Authenticity in Security Protocols. In SAS'02.