Symbolic Verification of Cryptographic Protocols

Unbounded Process Verification with Proverif

David Baelde

LSV, ENS Paris-Saclay

2019–2020

Introduction

Proverif

Protocol verifier developped by Bruno Blanchet at Inria Paris since 2000

- Analysis in formal model: secrecy, correspondences, equivalences, etc.
- Based on applied pi-calculus, Horn-clause abstraction and resolution
- The method is approximate but supports unbounded processes

Highly successful, works for most protocols including industrial ones: certified email, secure filesystem, Signal messenging, TLS draft, avionic protocols, etc.

These lectures

- Theory and practice of Proverif
- Secrecy, correspondences, equivalences

Terms

As usual in the formal model, messages are represented by terms

- built using constructor symbols from $f \in \Sigma_c$
- quotiented by an equational theory E;
- notation: $M \in \mathcal{M} = \mathcal{T}(\Sigma_c, \mathcal{N})$.

Additionally, computations are also modeled explicitly

- terms may also feature destructor symbols $g \in \Sigma_d$;
- semantics given by reduction rules $g(M_1, \ldots, M_n) \to M$;
- yields partial computation relation $\Downarrow$ over $\mathcal{T}(\Sigma, N) \times \mathcal{M}$.

Intuition:

- use constructors for total functions,
- destructors when failure is possible/observable.

Example primitives

Symmetric encryption

```
type key.
fun enc(bitstring,key):bitstring.
reduc forall m:bitstring, k:key;
  dec(enc(m,k),k) = m.
```

Block cipher

```
type key.
fun enc(bitstring,key):bitstring.
fun dec(bitstring,key):bitstring.
equation forall m:bitstring, k:key; dec(enc(m,k),k) = m.
equation forall m:bitstring, k:key; enc(dec(m,k),k) = m.
```

Exercise: how would you model signatures?

Processes

Similar to the one(s) seen before, with a few key differences:

- variables are typed (more on that later);
- private channels, phases, tables, events, etc.

Concrete syntax

```
P,Q ::= 0 | (P|Q) | !P | new n:t;P
      | in(c,x:t);P | out(c,u);P
      | if u = v then P else Q
      | let x = g(u1,..,uN) in P else Q
```

where `u`, `v` stand for constructor terms.

More details in reference manual:

http://prosecco.gforge.inria.fr/personal/bblanche/proverif/manual.pdf

First examples

File structure

- Declarations: types, constructors, destructors, public and private data, processes...
- Queries, for now only secrecy: `query attacker(s)`.
- System specification: the process/scenario to be analyzed.

Demo: `hello.pv` (basic file structure and use).

Demo: `types.pv` (on the role of types).

How does it work ?

Horn clause modeling

Encode the system as a set of Horn clauses $\mathcal{C}$:

- attacker's abilities, e.g. constructor f yields
  $\forall M_1, \ldots, M_n.\ (\bigwedge_i \mathsf{attacker}(M_i)) \Rightarrow \mathsf{attacker}(f(M_1, \ldots, M_n))$.
- protocol behaviour, e.g. $\mathsf{in}(c, x).\mathsf{out}(c, \mathsf{senc}(x, sk))$ yields
  $\forall M.\ \mathsf{attacker}(M) \Rightarrow \mathsf{attacker}(\mathsf{senc}(M, sk))$.

Clauses over-approximate behaviours, $\mathcal{C} \not\models \mathsf{attacker}(s)$ implies secrecy.

Automated reasoning

Entailment is undecidable for first-order Horn clauses but resolution (with strategies) provides practical semi-decision algorithms.
Proverif's possible outcomes:

- may not terminate, may terminate with real or false attack;
- when it declares a protocol secure, it really is.

Attacker's clauses (communication)

Predicates

Only two predicates (for now):

- attacker(M): attacker may know M
- mess(M, N): message N may be available on channel M

Variables range over messages; destructors not part of the logical language.

Communication

Send and receive on known channels:

$$\forall M, N.\ \mathsf{attacker}(M) \wedge \mathsf{attacker}(N) \Rightarrow \mathsf{mess}(M, N)$$
$$\forall M, N.\ \mathsf{mess}(M, N) \wedge \mathsf{attacker}(M) \Rightarrow \mathsf{attacker}(N)$$

Attacker's clauses (deduction)

Constructors

For each $f \in \Sigma_c$ of arity n:

$\forall M_1, \ldots, M_n.\ (\bigwedge_i \mathsf{attacker}(M_i)) \Rightarrow \mathsf{attacker}(f(M_1, \ldots, M_n))$

Similar clauses are generated for public constants and new names.

Destructors

For each $g(M_1, \ldots, M_n) \rightarrow M$:

$\forall M_1, \ldots, M_n.\ (\bigwedge_i \mathsf{attacker}(M_i)) \Rightarrow \mathsf{attacker}(M)$

Equations

Proverif attempts to turn them to rewrite rules, treated like destructors.

For instance $\mathsf{senc}(\mathsf{sdec}(x, k), k) = x$ yields

$\forall M, N.\ \mathsf{attacker}(\mathsf{sdec}(M, N)) \wedge \mathsf{attacker}(N) \Rightarrow \mathsf{attacker}(M)$.

Demo: `set verboseClauses = short/explained.`

Protocol clauses (informal)

Outputs

For each output, generate clauses:

- with all surrounding inputs as hypotheses;
- considering all cases for conditionals and evaluations.

Example:

$\mathsf{in}(c, x).\mathsf{in}(c, y).\mathsf{if}\ y = n\ \mathsf{then}\ \mathsf{let}\ z = \mathsf{sdec}(x, k)\ \mathsf{in}\ \mathsf{out}(c, \mathsf{senc}(\langle z, n\rangle, k))$

yields the following clause (assuming that c is public)

$\forall M.\ \mathsf{attacker}(\mathsf{senc}(M, k)) \wedge \mathsf{attacker}(n) \Rightarrow \mathsf{attacker}(\mathsf{senc}(\langle M, n\rangle, k))$

Protocol clauses (informal)

Outputs

For each output, generate clauses:

- with all surrounding inputs as hypotheses;
- considering all cases for conditionals and evaluations.

Example:

$\mathsf{in}(c,x).\mathsf{in}(c,y).\mathsf{if}\ y = n\ \mathsf{then}\ \mathsf{let}\ z = \mathsf{sdec}(x,k)\ \mathsf{in}\ \mathsf{out}(c, \mathsf{senc}(\langle z, n\rangle, k))$

yields the following clause (assuming that c is public)

$\forall M.\ \mathsf{attacker}(\mathsf{senc}(M,k)) \wedge \mathsf{attacker}(n) \Rightarrow \mathsf{attacker}(\mathsf{senc}(\langle M, n\rangle, k))$

Replication

Replication is ignored, as clauses can already be re-used in deduction.

Protocol clauses (informal)

Outputs

For each output, generate clauses:

- with all surrounding inputs as hypotheses;
- considering all cases for conditionals and evaluations.

Example:

$\mathsf{in}(c, x).\mathsf{in}(c, y).\mathsf{if}\ y = n\ \mathsf{then}\ \mathsf{let}\ z = \mathsf{sdec}(x, k)\ \mathsf{in}\ \mathsf{out}(c, \mathsf{senc}(\langle z, n\rangle, k))$

yields the following clause (assuming that c is public)

$\forall M.\ \mathsf{attacker}(\mathsf{senc}(M, k)) \wedge \mathsf{attacker}(n) \Rightarrow \mathsf{attacker}(\mathsf{senc}(\langle M, n\rangle, k))$

Replication

Replication is ignored, as clauses can already be re-used in deduction.

Protocol clauses (exercise)

For Proverif P is the same as $!P$.
More generally $Q = C[P]$ is the same as $Q' = C[!P]$.

Exercise

Find $Q = C[P]$ and $Q' = C[!P]$ such that

- Q ensures the secrecy of some value;
- Q' does not.

Analyze Q in Proverif; what happens?

Protocol clauses (exercise)

For Proverif P is the same as $!P$.
More generally $Q = C[P]$ is the same as $Q' = C[!P]$.

Exercise

Find $Q = C[P]$ and $Q' = C[!P]$ such that

- Q ensures the secrecy of some value;
- Q' does not.

Analyze Q in Proverif; what happens?

A possible solution: `repeat.pv`.

Protocol clauses (informal)

Nonces

Treated as (private) constructors taking surrounding inputs as argument.

For example, $\mathsf{new}\ a.\ \mathsf{in}(c, x).\mathsf{new}\ b.\mathsf{in}(c, y).\mathsf{out}(c, u(x, y, a, b))$ yields
$\forall M, N.\ \mathsf{attacker}(M) \wedge \mathsf{attacker}(N) \Rightarrow \mathsf{attacker}(u(M, N, a[], b[M]))$.

Exercise

In our process semantics, secrecy is not affected by the exchange of new and in operations. Find Q and Q' related by such exchanges such that

- both ensure the secrecy of some value;
- Proverif only proves it for Q.

Protocol clauses (informal)

Nonces

Treated as (private) constructors taking surrounding inputs as argument.

For example, new a. $\mathsf{in}(c, x).\mathsf{new}\ b.\mathsf{in}(c, y).\mathsf{out}(c, u(x, y, a, b))$ yields
$\forall M, N.\ \mathsf{attacker}(M) \wedge \mathsf{attacker}(N) \Rightarrow \mathsf{attacker}(u(M, N, a[], b[M]))$.

Exercise

In our process semantics, secrecy is not affected by the exchange of new and in operations. Find Q and Q' related by such exchanges such that

- both ensure the secrecy of some value;
- Proverif only proves it for Q.

A possible solution: `freshness.pv`.

The (long-)running example in Proverif

Demo: `nsl-secrecies.pv`

Similar to first lecture example, but generalized.

Demo HTML output with attack diagram.

Protocol clauses

$[\![0]\!]^H_\rho = \emptyset$ $\quad$ $[\![P \mid Q]\!]^H_\rho = [\![P]\!]^H_\rho \cup [\![Q]\!]^H_\rho$ $\quad$ $[\![!P]\!]^H_\rho = [\![P]\!]^H_\rho$

Protocol clauses

$[\![0]\!]^H_\rho = \emptyset \qquad [\![P \mid Q]\!]^H_\rho = [\![P]\!]^H_\rho \cup [\![Q]\!]^H_\rho \qquad [\![!P]\!]^H_\rho = [\![P]\!]^H_\rho$

$[\![\mathsf{in}(c,x).\ P]\!]^H_\rho = [\![P]\!]^{H \cup \{\mathsf{mess}(c\rho,x)\}}_{\rho+(x \mapsto x)}$

$[\![\mathsf{out}(c,u).\ P]\!]^H_\rho = \{H \Rightarrow \mathsf{mess}(c\rho, u\rho)\} \cup [\![P]\!]^{H \wedge \mathsf{mess}(c,x)}_\rho$

$[\![\mathsf{new}\ a.\ P]\!]^H_\rho = [\![P]\!]^H_{\rho+(a \mapsto a[p'_1,\dots,p'_n])}$ where $H = \wedge_i \mathsf{mess}(p_i, p'_i)$

Protocol clauses

$$[\![0]\!]^H_\rho = \emptyset \qquad [\![P \mid Q]\!]^H_\rho = [\![P]\!]^H_\rho \cup [\![Q]\!]^H_\rho \qquad [\![!P]\!]^H_\rho = [\![P]\!]^H_\rho$$

$$[\![\mathsf{in}(c,x).\ P]\!]^H_\rho = [\![P]\!]^{H \cup \{\mathsf{mess}(c\rho,x)\}}_{\rho+(x\mapsto x)}$$

$$[\![\mathsf{out}(c,u).\ P]\!]^H_\rho = \{H \Rightarrow \mathsf{mess}(c\rho, u\rho)\} \cup [\![P]\!]^{H \wedge \mathsf{mess}(c,x)}_\rho$$

$$[\![\mathsf{new}\ a.\ P]\!]^H_\rho = [\![P]\!]^H_{\rho+(a\mapsto a[p'_1,\dots,p'_n])} \qquad \text{where } H = \wedge_i \mathsf{mess}(p_i, p'_i)$$

$$[\![\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q]\!]^H_\rho = [\![P]\!]^{H\sigma}_{\rho\sigma} \cup [\![Q]\!]^H_\rho \qquad \text{where } \sigma = \mathrm{mgu}(u\rho, v\rho)$$

Protocol clauses

$$[\![0]\!]^H_\rho = \emptyset \qquad [\![P \mid Q]\!]^H_\rho = [\![P]\!]^H_\rho \cup [\![Q]\!]^H_\rho \qquad [\![!P]\!]^H_\rho = [\![P]\!]^H_\rho$$

$$[\![\mathsf{in}(c,x).\ P]\!]^H_\rho = [\![P]\!]^{H \cup \{\mathsf{mess}(c\rho,x)\}}_{\rho+(x\mapsto x)}$$

$$[\![\mathsf{out}(c,u).\ P]\!]^H_\rho = \{H \Rightarrow \mathsf{mess}(c\rho, u\rho)\} \cup [\![P]\!]^{H \wedge \mathsf{mess}(c,x)}_\rho$$

$$[\![\mathsf{new}\ a.\ P]\!]^H_\rho = [\![P]\!]^H_{\rho+(a\mapsto a[p'_1,\dots,p'_n])} \qquad \text{where } H = \wedge_i \mathsf{mess}(p_i, p'_i)$$

$$[\![\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q]\!]^H_\rho = [\![P]\!]^{H\sigma}_{\rho\sigma} \cup [\![Q]\!]^H_\rho \qquad \text{where } \sigma = \mathrm{mgu}(u\rho, v\rho)$$

$$[\![\mathsf{let}\ x = g(u_1,\dots,u_n)\ \mathsf{in}\ P\ \mathsf{else}\ Q]\!]^H_\rho = \left(\bigcup_{(p',\sigma)\in X} [\![P]\!]^{H\sigma}_{\rho\sigma+(x\mapsto p'\sigma)}\right) \cup [\![Q]\!]^H_\rho$$
$$\text{where } X = \{\,(p',\sigma) \mid g(p'_1,\dots,p'_n) \to p',\ \sigma = \mathrm{mgu}(\wedge_i u_i\sigma = p'_i)\,\}$$

Example:

$\mathsf{in}(c,x).\mathsf{in}(c,y).\mathsf{if}\ y = n\ \mathsf{then}\ \mathsf{let}\ z = \mathsf{sdec}(x,k)\ \mathsf{in}\ \mathsf{out}(c, \mathsf{senc}(\langle z, n\rangle, k))$

Semi-deciding non-derivability

Let $\mathcal{C}$ be the encoding of a system.

Proposition

If m is not secret then (roughly) attacker(m) is derivable from $\mathcal{C}$ using the consequence rule:

$$\frac{H_1\sigma \quad \dots \quad H_n\sigma \quad (\vec{H} \Rightarrow C) \in \mathcal{C}}{C\sigma}$$

Equivalently: if attacker(m) is not derivable, then m is secret.

Goal

Find a semi-decision procedure that allows to conclude often enough that a fact is not derivable from $\mathcal{C}$.

Resolution with selection

Conventions

Let $\phi = \forall M_1, \ldots, M_k.\ H_1 \wedge H_n \Rightarrow C$ be a clause.
Quantifiers may be omitted: free variables implicitly universally quantified.
Hypotheses' order is irrelevant: $\{H_i\}_i \Rightarrow C$, where $\{H_i\}_i$ is a multiset.

Resolution with selection

For each clause ϕ, let $\mathrm{sel}(\phi)$ be a subset of its hypotheses.

$$\frac{\phi = (H'_1 \wedge \ldots \wedge H'_m \Rightarrow C') \quad \psi = (H_1 \wedge \ldots \wedge H_n \Rightarrow C)}{(\bigwedge_i H'_i \wedge \bigwedge_{j \neq k} H_j \Rightarrow C)\sigma}$$

With $\sigma = \mathrm{mgu}(C', H_k)$, $\mathrm{sel}(\phi) = \emptyset$, $H_k \in \mathrm{sel}(\psi)$
and variables of ϕ and ψ disjoint.

Logical completeness (1)

If $\mathcal{C}'$ is a set of clauses, let $\mathsf{solved}(\mathcal{C}') = \{ \phi \in \mathcal{C}' \mid \mathsf{sel}(\phi) = \emptyset \}$.

Proposition

Let $\mathcal{C}$ and $\mathcal{C}'$ be two sets of clauses such that

- $\mathcal{C} \subseteq \mathcal{C}'$ *and*
- $\mathcal{C}'$ *is closed under resolution with selection.*

If F is derivable from $\mathcal{C}$ then it is derivable from $\mathsf{solved}(\mathcal{C}')$, with a derivation of size (number of nodes) $\leq$ the original size.

Goal: saturate the initial set of clauses by resolution?

Resolution examples

- The selection strategy is crucial to obtain termination:

$$\mathsf{attacker}(x) \wedge \mathsf{attacker}(y) \Rightarrow \mathsf{attacker}(\mathsf{aenc}(x, y))$$

Resolution examples

- The selection strategy is crucial to obtain termination:

$$\text{attacker}(x) \wedge \text{attacker}(y) \Rightarrow \text{attacker}(\text{aenc}(x, y))$$

Resolution examples

- The selection strategy is crucial to obtain termination:

$$\mathsf{attacker}(x) \wedge \mathsf{attacker}(y) \Rightarrow \mathsf{attacker}(\mathsf{aenc}(x, y))$$

- Termination not achieved in general, as seen in NS shared-key:

$$\begin{array}{lcl} B \to A & : & \mathsf{senc}(n_b, k) \\ A \to B & : & \mathsf{senc}(n_b - 1, k) \end{array}$$

Logical completeness (2)

Subsumption

$(\{H_i\}_i \Rightarrow C) \sqsubseteq (\{H'_j\}_j \Rightarrow C')$ if there exists σ such that

- $C'\sigma = C$ and
- for all j, $H'_j\sigma = H_i$ for some i.

Given a set of clauses, let $\mathrm{elim}(\mathcal{C})$ be a set of clauses such that for all $\phi \in \mathcal{C}$ there is $\psi \in \mathrm{elim}(\mathcal{C})$ such that $\phi \sqsubseteq \psi$.

Saturation of an initial set of clauses $\mathcal{C}_0$

1. initialize $\mathcal{C} := \mathrm{elim}(\mathcal{C}_0)$
2. for each ϕ generated from $\mathcal{C}$ by resolution, let $\mathcal{C} := \mathrm{elim}(\mathcal{C} \cup \{\phi\})$
3. repeat step 2 until a fixed point is reached, let $\mathcal{C}'$ be the result.

Theorem

If F is derivable from $\mathcal{C}_0$ then it is derivable from $\mathrm{solved}(\mathcal{C}')$.

Summing up: Proverif's procedure

Procedure for secrecy

- Encode system as $\mathcal{C}_0$.
- Saturate it to obtain $\mathcal{C}'$.
- Declare secrecy of m if solved($\mathcal{C}'$) contains no clause with conclusion attacker(m') with $m'\sigma = m$.

Summing up: Proverif's procedure

Procedure for secrecy

- Encode system as $\mathcal{C}_0$.
- Saturate it to obtain $\mathcal{C}'$.
- Declare secrecy of m if solved($\mathcal{C}'$) contains no clause with conclusion attacker(m') with $m'\sigma = m$.

Remarks

- Choice of selection function: at most one hypothesis, of the form attacker(u) where u is not a variable.
- Not covered here: treatment of equations, several optimizations.
- Differences with standard resolution: focus on deducible facts rather than consistency; factorisation not needed (Horn).

Termination and decidability

Proverif's procedure works very well in practice, but offers no guarantee.
This can be improved under additional assumptions.

Tagging

Secrecy is decidable for (reasonable classes of) tagged protocols.

- Blanchet & Podelski 2003: termination of resolution
- Ramanujan & Suresh 2003: decidability, but forbid blind copies

At most one blind copy

- Comon & Cortier 2003: decidability through (ordered) resolution

Illustration: resolution with selection on tagged NS shared-key

Correspondences

Roughly, express that if X happens then Y must have happened.

- If B thinks he's completed the protocol with A, then A thinks he's completed the protocol with B.

Events

Add events to the syntax of protocols:

```
(* Declaration *)
event evName(type1,..,typeN).
(* Use inside processes *)
P ::= ... | event evName(u1,..,uN); P
```

Semantics extended as follows:

$$(\text{event } E.\ P \mid Q, \Phi) \xrightarrow{\tau} (P \mid Q, \Phi)$$

Queries

Definition

The query

```
query x1:t1, .., xN:tK;
  event(E(u1,..,uN)) ==> event(E'(v1,..,vM))
```

holds if for all traces of the system

- if the trace ends with an event rule for an event of the form $E(u_i)_i$,
- there is a prior execution of the rule for an event of the form $E'(v_j)_j$.

Note that variables of u_i are universally quantified
while those only ocurring in v_j are existentially quantified.

Example

```
query na:bitstring, nb:bitstring;
  event(endR(pka,pkb,na,nb)) ==> event(endI(pka,pkb,na,nb)).
```

How does it work ?

It is natural to encode events as outputs using a dedicated predicate.
For example,

$$(\mathrm{in}(c, x).\ \mathrm{if}\ x = n_a\ \mathrm{then\ event}\ E)$$

would yield

$$(\mathrm{attacker}(n_a) \Rightarrow \mathrm{occurs}(E)).$$

Problem # 1

This approximate encoding would only express that the event may occur.
When checking $E ==> E'$ we cannot over-approximate E'!

- We will see how "must occur" can be encoded in the language of Horn clauses and resolution.

How does it work ?

Problem # 2

Because of the approximate encoding of fresh names, messages in the logic do not correspond uniquely to messages in the semantics.

The process

$$
\begin{array}{ll}
 & \mathsf{new}\ d : \mathsf{channel}; \\
! & \mathsf{new}\ a : \mathsf{bitstring}; \\
 & \mathsf{in}(c, x : \mathsf{bool}); \\
 & \mathsf{if}\ x = \mathsf{true}\ \mathsf{then}\ \mathsf{event}\ A(a); \mathsf{out}(d, \mathsf{ok})\ \mathsf{else} \\
 & \mathsf{if}\ x = \mathsf{false}\ \mathsf{then}\ \mathsf{in}(d, x : \mathsf{bitstring}); \mathsf{event}\ B(a)
\end{array}
$$

should not have any trace satisfying

$\mathsf{query}\ x : \mathsf{bitstring};\ \mathsf{event}(B(x)) ==> \mathsf{event}(A(x)).$

How does it work ?

Problem # 2

Because of the approximate encoding of fresh names, messages in the logic do not correspond uniquely to messages in the semantics.

The process

$$
\begin{array}{ll}
 & \text{new } d : \text{channel}; \\
! & \text{new } a : \text{bitstring}; \\
 & \text{in}(c, x : \text{bool}); \\
 & \text{if } x = \text{true then event } A(a); \text{out}(d, \text{ok}) \text{ else} \\
 & \text{if } x = \text{false then in}(d, x : \text{bitstring}); \text{event } B(a)
\end{array}
$$

should not have any trace satisfying

query x : bitstring; event($B(x)$) ==> event($A(x)$).

We will ignore this problem in this lecture.

How does it work?

Translation

Use a predicate begin$(\cdot)$ for events that must occur,
and end$(\cdot)$ for events that may occur.
Treat event(M) actions in processes using both may and must:

$$[\![\text{event } M; P]\!]_\rho^H = [\![P]\!]_\rho^{H \wedge \text{begin}(\text{event}(M\rho))} \cup \{H\rho \Rightarrow \text{end}(\text{event}(M\rho))\}$$

We may look at `nspk-auth.pv` for concrete examples.

Verification problem

query $x_1, \ldots, x_n$; event$(E(u_i)_i)$ ==> event$(E'(v_j)_j)$

$\Leftrightarrow$ deriving end$(E(u_i)_i)$ requires to derive an instance of begin$(E'(v_j)_j)$ (ignoring problem # 2)

$\Leftrightarrow$ for all sets $\mathcal{E}$ of begin(M) open facts, end$(E(u_i)_i)$ is derivable from $\mathcal{C} \cup \mathcal{E}$ only if $\mathcal{E}$ contains begin$(E'(v_j)_j)$ (or a generalization of it)

Verifying correspondences through resolution

So we want to verify the following:

for all sets $\mathcal{E}$ of begin(M) open facts, end($E(u_i)_i$) is derivable from $\mathcal{C} \cup \mathcal{E}$ only if $\mathcal{E}$ contains begin($E'(v_j)_j$) (or a generalization of it).

But we don't know $\mathcal{E}$ and can't enumerate all of them!

Verifying correspondences through resolution

So we want to verify the following:

for all sets $\mathcal{E}$ of begin(M) open facts, end($E(u_i)_i$) is derivable from $\mathcal{C} \cup \mathcal{E}$ only if $\mathcal{E}$ contains begin($E'(v_j)_j$) (or a generalization of it).

But we don't know $\mathcal{E}$ and can't enumerate all of them!

Key observation

If we never select on begin(M) hypotheses, saturating on $\mathcal{C} \cup \mathcal{E}$ is the same as saturating on $\mathcal{C}$ and adding $\mathcal{E}$ afterwards.

Verifying correspondences through resolution

So we want to verify the following:

for all sets $\mathcal{E}$ of begin(M) open facts, end$(E(u_i)_i)$ is derivable from $\mathcal{C} \cup \mathcal{E}$ only if $\mathcal{E}$ contains begin$(E'(v_j)_j)$ (or a generalization of it).

But we don't know $\mathcal{E}$ and can't enumerate all of them!

Key observation

If we never select on begin(M) hypotheses, saturating on $\mathcal{C} \cup \mathcal{E}$ is the same as saturating on $\mathcal{C}$ and adding $\mathcal{E}$ afterwards.

Procedure

- Build $\mathcal{C}$ by translating the process and adding attacker clauses.
- Get $\mathcal{C}'$ by saturating $\mathcal{C}$, without selecting begin$(\cdot)$ hypotheses.
- Check that for all clauses $(\{H_i\}_i \Rightarrow C) \in \text{solved}(\mathcal{C}')$, and all σ such that $C\sigma = \text{end}(E(u_i)_i)\sigma$, there exists i such that $H_i\sigma$ is an instance of begin$(E'(v_j)_j)\sigma$.