Unrestricting restrictions in ProVerif

Vincent Cheval

Inria Paris
vincent.cheval@inria.fr

Collaborations with José Moreira and Mark Ryan, and with Bruno Blanchet

GT FMS
30/03/2023
Symbolic (Dolev-Yao) models

The attacker can...

- Read / Write
- Intercept

But they do not...

- Break cryptography
- Use side channels

Concurrent systems where dishonest parties have complete control over network communication

*but* cryptography is idealised
Automated Verification

Two main verifiers

Proverif

- Large class of cryptographic primitives
- Reachability and equivalence properties
- Unbounded number of sessions
- Push-button
- Almost no guidance

Tamarin

- Interactive interface
- Lot of guidance
- Less automated

Axioms

Restrictions
Automated Verification

Two main verifiers

Proverif

Large class of cryptographic primitives
Reachability and equivalence properties
Unbounded number of sessions

Tamarin

Interactive interface

Push-button
Almost no guidance

Lot of guidance
Less automated

Almost no guidance

Lemmas
Axioms
Restrictions

More guidance
[BCC-S&P22]

More automation
[CDDK-JCS22]
**Lemmas / Axioms / Restrictions**

**Lemmas**
- Intermediary property useful to prove the main query
- Proved by the tool

**Axioms**
- Similarly to Lemmas but assumed by the tool

**Restrictions**
- Restricts the search space of traces on which to prove the main query
- Sometimes useful to avoid heavy encoding
Lemmas / Axioms / Restrictions in ProVerif [BCC22]

Already useful

- Basis of GSVerif tool
- Allow to handle stateful protocols
- Add precision to ProVerif
- Instrumental in the verification of TLS-ECH, Voting protocols, ZCash

\[
\text{axiom } st: \text{bitstring}, x: \text{bitstring}, y: \text{bitstring};
\text{event(precise(st,x)) && event(precise(st,y))} \implies x = y.
\]
Lemmas / Axioms / Restrictions in ProVerif [BCC22]

Already useful

Basis of GSVerif tool

Allow to handle stateful protocols

Add precision to ProVerif

Instrumental in the verification of TLS-ECH, Voting protocols, ZCash

Lemmas are correspondence queries

Premise can be any predicate (events, attacker, mess, table, user-defined)

\[ F_1 \land \cdots \land F_n \Rightarrow \phi \]

Disjunctions and conjunction of events, inequalities, equalities, and disequalities

\[ \forall x_1, x_2, \ldots, x_k . F_1 \land \cdots \land F_n \Rightarrow \exists y_1, \ldots, y_{\ell} . \phi \]

Variables in the premises

Remaining variables in \( \phi \)

Axiom
\[
\text{st: bitstring, x: bitstring, y: bitstring;}
\text{event(precise(st,x)) \&\& event(precise(st,y))} \Rightarrow x = y.
\]
Lemmas / Axioms / Restrictions in ProVerif [BCC22]

Premise can be any predicate (events, attacker, mess, table, user-defined)

\[ F_1 \land \ldots \land F_n \Rightarrow \phi \]

Disjunctions and conjunction of events, inequalities, equalities, and disequalities
Lemmas / Axioms / Restrictions in ProVerif [BCC22]

Premise can be any predicate (events, attacker, mess, table, user-defined)

Disjunctions and conjunction of events, inequalities, equalities, and disequalities

Does not allow temporal variables

\[ F_1 \land \ldots \land F_n \Rightarrow \phi \]
Lemmas / Axioms / Restrictions in ProVerif [BCC22]

Premise can be any predicate (events, attacker, mess, table, user-defined)

Disjunctions and conjunction of events, inequalities, equalities, and disequalities

Does not allow temporal variables

Does not allow disequalities and inequalities in the premises

\[ F_1 \land \ldots \land F_n \Rightarrow \phi \]
Lemmas / Axioms / Restrictions in ProVerif [BCC22]

Premise can be any predicate (events, attacker, mess, table, user-defined)

Disjunctions and conjunction of events, inequalities, equalities, and disequalities

Does not allow temporal variables

Does not allow disequalities and inequalities in the premises

Events in the conclusion do not restrict traces

\[ F_1 \land \ldots \land F_n \Rightarrow \phi \]
Premise can be any predicate (events, attacker, mess, table, user-defined)

Disjunctions and conjunction of events, inequalities, equalities, and disequalities

Does not allow temporal variables

Does not allow disequalities and inequalities in the premises

Events in the conclusion do not restrict traces

Semantics of restrictions enforce that events in the conclusion occur before at least one fact of the premise.
Lemma Axioms Restrictions in ProVerif [BCC22]

Premise can be any predicate (events, attacker, mess, table, user-defined)

Does not allow temporal variables

Does not allow disequalities and inequalities in the premises

Events in the conclusion do not restrict traces

Disjunctions and conjunction of events, inequalities, equalities, and disequalities

Conclusion cannot contain attacker, mess, table, or user-defined predicate

Semantics of restrictions enforce that events in the conclusion occur before at least one fact of the premise.
Lemmas / Axioms / Restrictions in ProVerif

- Does not allow temporal variables
- Does not allow disequalities and inequalities in the premises
- Events in the conclusion do not restrict traces
- Conclusion cannot contain attacker, mess, table, or user defined predicate
- Semantics of restrictions enforce that events in the conclusion occur before at least one fact of the premise.

Allow to be more expressive in the order of events

```plaintext
axiom id:voter, v,v':vote, i,j:time;
event(hasVoted(id,v))@i && event(hasVoted(id,v'))@j ==> i = j.
```
Lemmas / Axioms / Restrictions in ProVerif

- Does not allow temporal variables
- Does not allow disequalities and inequalities in the premises
- Events in the conclusion do not restrict traces
- Conclusion cannot contain attacker, mess, table, or user defined predicate
- Semantics of restrictions enforce that events in the conclusion occur before at least one fact of the premise.

Allow to be more expressive in the order of events

```
lemma i,j:nat;
event(A(i)) && event(B(j)) && i < j ==> event(C(i,j)).
```

instead of

```
lemma i,j:nat;
event(A(i)) && event(B(j)) ==> i >= j || ( i < j && event(C(i,j)) ).
```
Lemmas / Axioms / Restrictions in ProVerif

- Does not allow temporal variables
- Does not allow disequalities and inequalities in the premises
- Events in the conclusion do not restrict traces
- Conclusion cannot contain attacker, mess, table, or user defined predicate
- Semantics of restrictions enforce that events in the conclusion occur before at least one fact of the premise.

Allow to be more expressive in the order of events

Allow to be more efficient in the application of the lemma

Avoid non-termination scenarios

\[
\text{Lemma: } \text{id: voter, } v: \text{vote, } i,j: \text{time}; \\
\text{event(VoteCounted(id,v))@i} \implies \text{attacker(v)@j} \land i > j.
\]

\[
\text{s-event(VoteCounted(id, v))} \land H \implies \text{att(v)}
\]

Clause generated when the vote is revealed by the tally

Removed by application of the lemma
<table>
<thead>
<tr>
<th>Lemma/Restriction</th>
<th>Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does not allow temporal variables</td>
<td>Allow to be more expressive in the order of events</td>
</tr>
<tr>
<td>Does not allow disequalities and inequalities in the premises</td>
<td>Allow to be more efficient in the application of the lemma</td>
</tr>
<tr>
<td>Events in the conclusion do not restrict traces</td>
<td>Avoid non-termination scenarios</td>
</tr>
<tr>
<td>Conclusion cannot contain attacker, mess, table, or user defined predicate</td>
<td>Improve precision</td>
</tr>
<tr>
<td>Semantics of restrictions enforce that events in the conclusion occur before at least one fact of the premise.</td>
<td></td>
</tr>
<tr>
<td>Lemmas / Axioms / Restrictions in ProVerif</td>
<td></td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Does not allow temporal variables</td>
<td></td>
</tr>
<tr>
<td>Does not allow disequalities and inequalities in the premises</td>
<td></td>
</tr>
<tr>
<td>Events in the conclusion do not restrict traces</td>
<td></td>
</tr>
<tr>
<td>Conclusion cannot contain attacker, mess, table, or user defined predicate</td>
<td></td>
</tr>
<tr>
<td>Semantics of restrictions enforce that events in the conclusion occur before at least one fact of the premise.</td>
<td></td>
</tr>
</tbody>
</table>

- Allow to be more expressive in the order of events
- Allow to be more efficient in the application of the lemma
- Avoid non-termination scenarios
- Improve precision
- Allow to prove properties on complex data structure
Lemmas / Axioms / Restrictions in ProVerif

- Does not allow temporal variables
- Does not allow disequalities and inequalities in the premises
- Events in the conclusion do not restrict traces
- Conclusion cannot contain attacker, mess, table, or user defined predicate
- Semantics of restrictions enforce that events in the conclusion occur before at least one fact of the premise.

Allow to be more expressive in the order of events
Allow to be more efficient in the application of the lemma
Avoid non-termination scenarios
Improve precision
Allow to prove properties on complex data structure
Allow to prove liveness and accountability properties
Liveness properties [BDKK-EuroSnP17]

Local progress
Processes need to be reduced as far as possible, that is until they wait for a message

Resilient channels
Messages sent on resilient channel must be delivered

External non-determinism
Any process $P + Q$ reduces to $R$ if $P$ or $Q$ reduces to $R$. 

All these properties can be enforced by "forward" restrictions
Liveness properties [BDKK-EuroS&P17]

External non-determinism

Any process $P + Q$ reduces to $R$ if $P$ or $Q$ reduces to $R$.

All these properties can be enforced by "forward" restrictions

$P + Q$

translated into

$P = a; P'$

$Q = b; Q'$

$\text{event } B; (\text{event } M; a; \text{event } E; P' | \text{event } M; b; \text{event } E; Q')$

restriction

$\text{event}(B) \implies \text{event}(E);$

$\text{event}(M)@i \land \text{event}(M)@j \implies i = j.$
Properties on complex data structures [CMR-EuroS&P23]

Merkle trees

- **Digest of the tree**
  - \( h(h(h(d_1), h(d_2)), h(d_3)) \)
- Data in the leaves
  - \( h(d_1) \)
  - \( h(d_2) \)
  - \( h(d_3) \)

- Good to model ledgers
- Append only structure
- Proof of presence in \( O(\log(n)) \)
- Proof of extension in \( O(\log(n)) \)
Properties on complex data structures [CMR-EuroS&P23]

Merkle trees

In green, proof of extension between the two trees
Properties on complex data structures [CMR-EuroS&P23]

Defining verification predicates through Horn clauses

```prolog
(* Proof of presence *)
fun PP(list):proof_of_presence [data].
clauses
  forall x:bitstring;
       verify_pp(PP(nil),x,hash(leaf(x)));
  forall pl:list, x:bitstring, d_left,d_right:digest;
       verify_pp(PP(pl),x,d_left) ->
       verify_pp(PP(cons((left,d_right),pl)),x,hash(node(d_left,d_right)));
  forall pl:list, x:bitstring, d_left,d_right:digest;
       verify_pp(PP(pl),x,d_right) ->
       verify_pp(PP(cons((right,d_left),pl)),x,hash(node(d_left,d_right)))
.
```

Will often not terminate if these Horn clauses are given with the protocol
Properties on complex data structures [CMR-EuroS&P23]

Prove the protocol in two phases

**Phase 1**

- Define predicate with Horn Clauses
- Extract properties on the data structures
- Prove them as lemmas with the empty protocol

**Interface**

**Phase 2**

- Predicate given with unspecified semantics
- Predicate under specified with restrictions
- Prove the protocol

**Lemmas -> Restrictions**

Phase 2 proves the protocol for all implementations satisfying the interface.
Properties on complex data structures [CMR-EuroS&P23]

Extract of the interface

(* Transitivity of proof of extension *)
lemma pe1,pe2,pe3:proof_of_extension, d1,d2,d3:digest;
  verify_pe(pe1,d1,d2) && verify_pe(pe2,d2,d3) ==> verify_pe(pe3,d1,d3).

(* Proofs of presence are stable by proofs of extension *)
lemma x:bitstring, pe:proof_of_extension, pp1,pp2:proof_of_presence, d1,d2:digest;
  verify_pp(pp1,x,d1) && verify_pe(pe,d1,d2) ==> verify_pp(pp2,x,d2).
Properties on complex data structures

Useful to model other predicates: is_subterm

In work with Véronique Cortier and Alexandre Debant on Election Verifiability

(* We do not implement the full subterm semantics but only a sufficient subset. *)

pred is_subterm(bitstring,bitstring).

clauses
  forall x,y:bitstring; is_subterm(x,hash((y,x)));
  forall x,y:bitstring; is_subterm(x,x);
  forall x,y,z:bitstring; is_subterm(x,y) -> is_subterm(x,hash((y,z)))
.

lemma x,y:bitstring, uuid:election_id, j1,j2,i1,i2:nat,
h1,h2,ballot1,ballot2:bitstring;
  event(Ballot_In_Bulletin_Board(uuid,j1,i1,ballot1,h1)) &&
  event(Ballot_In_Bulletin_Board(uuid,j2,i2,ballot2,h2)) && i1 <= i2 ==> is_subterm(ballot1,h2)
.
Lemmas / Axioms / Restrictions in ProVerif

- Does not allow temporal variables

- Does not allow disequalities and inequalities in the premises

- Events in the conclusion do not restrict traces

- Conclusion cannot contain attacker, mess, table, or user defined predicate

- Semantics of restrictions enforce that events in the conclusion occur before at least one fact of the premise.

- Allow to be more expressive in the order of events

- Allow to be more efficient in the application of the lemma

- Avoid non-termination scenarios

- Improve precision

- Allow to prove properties on complex data structure

- Allow to prove liveness and accountability properties
### Lemmas / Axioms / Restrictions in ProVerif (next release)

- **Allow temporal variables**
  - Allow to be more expressive in the order of events
- **Allow disequalities and inequalities in the premises**
  - Allow to be more efficient in the application of the lemma
- **Events in the conclusion restrict traces**
  - Avoid non-termination scenarios
- **Conclusion can contain any predicate**
  - Improve precision
- **No semantics constraints on the occurrence order of events in restrictions.**
  - Allow to prove properties on complex data structure
  - Allow to prove liveness and accountability properties

Includes [CMR-EuroS&P23]
Under the hood: any predicate in query conclusion

**Process** → **Translation into Horn clauses** → **Saturation of Horn clauses**

Lemmas / Axioms / Restrictions → Applied on each Horn clauses

**Verification of query**

**Lemma**

\[ F_1 \land F_2 \implies G_1 \land G_2 \land G_3 \]

**Clause**

\[ H \rightarrow C \]

If there is a substitution \( \sigma \) such that \( F_1\sigma \land F_2\sigma \subseteq H \) then

\[ H \rightarrow C \text{ is replaced by } H \land G_1\sigma \land G_2\sigma \land G_3\sigma \rightarrow C \]
Under the hood: any predicate in query conclusion

Process → Translation into Horn clauses → Saturation of Horn clauses

Lemmas / Axioms / Restrictions → Applied on each Horn clauses

Verification of query

Lemma $F_1 \land F_2 \Rightarrow G_1 \land G_2 \land G_3$

Clause $H \rightarrow C$

If there is a substitution $\sigma$ such that $F_1\sigma \land F_2\sigma \subseteq H$ then $H \rightarrow C$ is replaced by $H \land G_1\sigma \land G_2\sigma \land G_3\sigma \rightarrow C$

Not always sound! events
attacker facts
Under the hood: any predicate in query conclusion

Two predicates for events

$s$-event  
$s$-event: occurs only in hypotheses of Horn clauses

$m$-event  
$m$-event: occurs only in conclusions of Horn clauses

Consequence: facts with $s$-event predicates are never resolved!

Applying a lemma is sound if no added facts can be resolved.
Two predicates for events

\( s\)-event \hspace{1cm} \textit{sure-event}: occurs only in hypotheses of Horn clauses
\( m\)-event \hspace{1cm} \textit{may-event}: occurs only in conclusions of Horn clauses

**Consequence:** facts with \( s\)-event predicates are never resolved!

Applying a lemma is sound if no added facts can be resolved.

For every predicate, we consider a \textit{blocking} predicate that cannot be resolved.
Under the hood: any predicate in query conclusion

\[
\text{lemma id:voter, v:vote; event(VoteCounted(id,v)) } \implies \text{ attacker(v).}
\]

\textbf{Clause} \quad \text{b-event(VoteCounted}(id, C_1)) \land H \implies \text{att}(C_1)
Under the hood: any predicate in query conclusion

Lemma \text{ id: voter, v: vote; event(VoteCounted(id,v)) ==> attacker(v).}

Clause \quad \text{b-event(VoteCounted(id, C_1)) \land H \rightarrow att(C_1)}

After applying the lemma

\quad \text{\text{b-att}(C_1) \land \text{b-event(VoteCounted(id, C_1)) \land H \rightarrow att(C_1)}}
Under the hood: any predicate in query conclusion

**Lemma**

\[
\text{id: voter, v: vote; event(VoteCounted(id,v)) \implies attacker(v).}
\]

**Clause**

\[
b\text{-event(VoteCounted(id, } C_1) \land H \rightarrow \text{att(C}_1)\]

**After applying the lemma**

\[
b\text{-att(C}_1) \land b\text{-event(VoteCounted(id, } C_1)) \land H \rightarrow \text{att(C}_1)\]

**Transformation rules are adapted to take blocking predicate into account**

The clause is removed by tautology
Under the hood: forward restriction

```prolog
process event Send | (event Goal; event Received).
```

```prolog
query event(Send) ==> event(Goal).  True
```

False
Under the hood: forward restriction

```
process event Send | (event Goal; event Received).
```

```
query event(Send) ==> event(Goal).
```

```
restriction event(Send) ==> event(Received).
```

But ProVerif can’t prove it… Why?
Under the hood: forward restriction

\[
\text{process event } \text{Send} \mid (\text{event Goal; event Received}).
\]

\[
\text{query event(Send)} \implies \text{event(Goal)}.
\]

\[
\text{restriction event(Send)} \implies \text{event(Received)}.
\]

But ProVerif can’t prove it… Why?

Clauses generated:

\[
\rightarrow \text{event(Send)}
\]

\[
\text{b-event(Goal)} \rightarrow \text{event(Received)}
\]

\[
\rightarrow \text{event(Goal)}
\]
Under the hood: forward restriction

```
process event Send | (event Goal; event Received).
```

```
query event(Send) ==> event(Goal).
```

```
restriction event(Send) ==> event(Received).
```

But ProVerif can’t prove it… Why?

**Clauses after saturation:**

```
b-event(Received)  →  event(Send)
```

```
b-event(Goal)  →  event(Received)
```

```
→  event(Goal)
```
Under the hood: forward restriction

Idea: Two rounds of saturation

restriction event(Send) \implies event(Received).

\text{1st Saturation of Horn clauses}

\begin{align*}
\text{b-event(Received)} \implies & \text{event(Send)} \\
\text{becomes} & \\
\text{b-event(Received)} \land \text{event(Received)} \implies & \text{event(Send)}
\end{align*}

\text{2nd Saturation of Horn clauses}

\begin{align*}
\text{b-event(Received)} \land \text{b-event(Goal)} \implies & \text{event(Send)} \\
\text{b-event(Goal)} \implies & \text{event(Received)} \\
\implies & \text{event(Goal)}
\end{align*}

Verification of the query
Timetable of ProVerif next releases

- **Now**
  - GT MFS 2023

- **2-3 months**
  - Next release with unrestricted lemmas/axioms/restrictions

- **End of 2023**
  - Memory optimisation (first prototype already developed)

- **Mid 2024**
  - ProVerif with DH, XOR, AC (early work with Caroline Fontaine)

- **??**
  - Certificate generator and machine checked verifier
    - Auto-detection of cycles with noselect suggestions
    - Auto-detection of secrecy assumptions
    - GSVerif integration
    - MultiCore ProVerif
    - Trace equivalence, simulation and bisimulation
    - Algorithm optimisations (subsumption, redundancy, lemma applications, …)
    - Symmetry-based query verification

Available at
- https://gitlab.inria.fr/bblanche/proverif
- https://bblanche.gitlabpages.inria.fr/proverif/

Interns wanted!