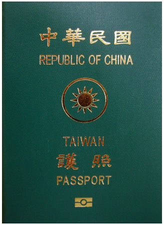
Partial Order Reduction for Security Protocols

68NQRT Seminar

Lucca Hirschi, David Baelde and Stéphanie Delaune

15th December, 2016

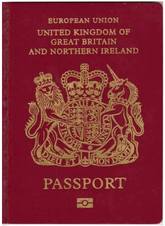

Google
31


中華民國
REPUBLIC OF CHINA
TAIWAN
護照
PASSPORT


Press the VOTE button to cast your ballot.
VOTE

wins (BlackHat'15)

wins (FMSE'08)



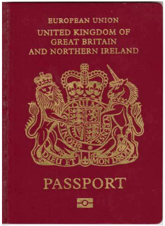
wins (CSF'10)

wins (CCS'10)

wins (CSF'11)

wins (BlackHat'15)

wins (FMSE'08)



wins (CSF'10)

wins (CCS'10)

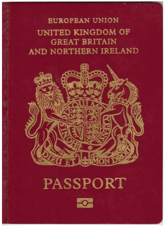
wins (CSF'11)

concurrent programs + unsecure network + active attacker

⇝ tricky attacks, hard to detect/avoid

wins (BlackHat'15)

wins (FMSE'08)



wins (CSF'10)

wins (CCS'10)

wins (CSF'11)

concurrent programs + unsecure network + active attacker

⇝ tricky attacks, hard to detect/avoid

⇝ need a mathematical framework to analyze protocols: **formal methods**

Symbolic Model

Symbolic attacker ( ) controls all the network:

Symbolic Model

[$\{n\}_k$: symmetric encryption]

Symbolic attacker () controls all the network:

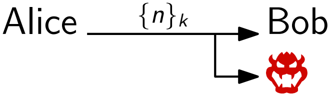
- eavesdrops messages

$$\text{Alice} \xrightarrow{\{n\}_k} \text{Bob}$$

[$\{n\}_k$: symmetric encryption]

Symbolic attacker ( ) controls all the network:

- eavesdrops messages



Symbolic Model

Symbolic attacker () controls all the network:

- eavesdrops messages
- builds new messages, applies crypto primitives

$\Big($ knows $\{n\}_k$ and $k\Big) \Rightarrow$
$\Big($ knows $n\Big)$

Symbolic Model

Symbolic attacker () controls all the network:

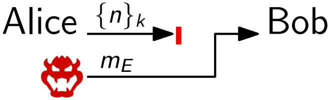
- eavesdrops messages
- builds new messages, applies crypto primitives
- injects messages

$$\text{Alice} \xrightarrow{\{n\}_k} \text{Bob}$$

Symbolic Model

Symbolic attacker ( ) controls all the network:

- eavesdrops messages
- builds new messages, applies crypto primitives
- injects messages



Symbolic Model

Symbolic attacker () controls all the network:

- eavesdrops messages
- builds new messages, applies crypto primitives
- injects messages

But  cannot break crypto primitives.

Symbolic Model

Symbolic attacker (👹) controls all the network:

- eavesdrops messages
- builds new messages, applies crypto primitives
- injects messages

But 👹 cannot break crypto primitives.

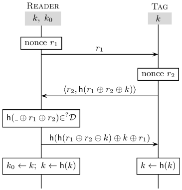
Symbolic model, pros & cons:

⊖ less precise than computational model (*i.e.,* no assumption on primitives)

⊕ allows for automation

Dolev, Yao: On the Security of Public Key Protocols. FOCS'81

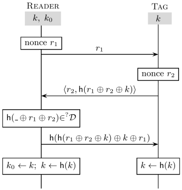
Introduction: Formal Methods for Security Protocols



Protocol's specification

Security goal
(*e.g.*, Secrecy)

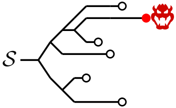
Introduction: Formal Methods for Security Protocols




```
(* ---------------------------------------------------- *)
(*              ORIGINAL LAK                            *)
(* ---------------------------------------------------- *)

let LAK = ! new k:bitstring;
            out(t, enc(secretK, k));
          !    (        (*  TAG   *)
             in(t,x:bitstring);
             new r1:bitstring;
             let m = h(xor(x, r1, k)) in
             let mout = (r1,m) in
             event TOut(k,r1, mout);
             out(t, mout);

             in(t, y:bitstring);
             event TIn(k,r1,y);

             if y = h(xor(m,x,k)) then
             event TOk(k,r1,x);
             out(t, ok)
            ) | (   (* READER *)
             new r0:bitstring;
             out(r, r0);
             in(r, (x1:bitstring,x2:bitstring));
             event RIn(k,r0,(x1,x2));

             if x2 = h(xor(r0, x1, k)) then
             event ROk(k, r0, x1);

             event ROut(k,r0, h(xor(x2,r0,k)));
             out(t, h(xor(x2,r0,k)))
            ).

process LAK
```

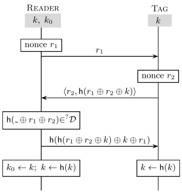
Protocol's specification ⟶ Protocol's model
(symbolic model)

Reachability
in a model



Security goal ⟶ Modelization of the property
(*e.g.*, Secrecy)
Reachability(State$_{\text{bad}}$)

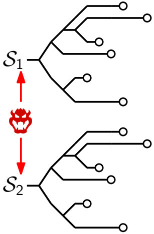
Introduction: Formal Methods for Security Protocols




```
(* ---------------------------------------------- *)
(*              ORIGINAL LAK                       *)
(* ---------------------------------------------- *)

let LAK = ! new k:bitstring;
            out(t, enc(secretK, k));
          !    (       (*  TAG   *)
             in(t,x:bitstring);
             new r1:bitstring;
             let m = h(xor(x, r1, k)) in
             let mout = (r1,m) in
             event TOut(k,r1, mout);
             out(t, mout);

             in(t, y:bitstring);
             event TIn(k,r1,y);

             if y = h(xor(m,x,k)) then
             event TOk(k,r1,x);
             out(t, ok)
            ) | (   (* READER *)
             new r0:bitstring;
             out(r, r0);
             in(r, (x1:bitstring,x2:bitstring));
             event RIn(k,r0,(x1,x2));

             if x2 = h(xor(r0, x1, k)) then
             event ROk(k, r0, x1);

             event ROut(k,r0, h(xor(x2,r0,k)));
             out(t, h(xor(x2,r0,k)))
            ).

process LAK
```

Protocol's specification → Protocol's model
(symbolic model)

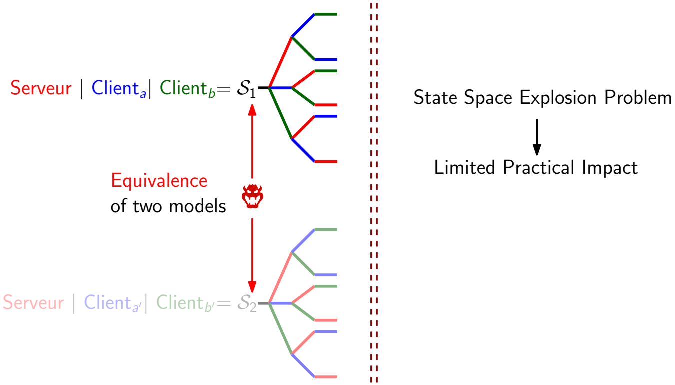
Equivalence
of two models



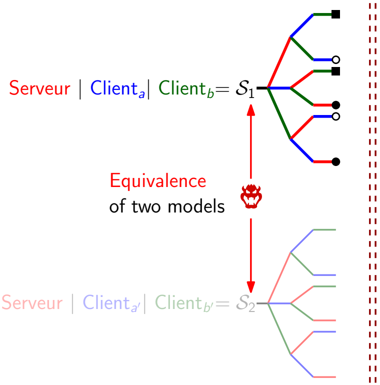
Privacy goal → Modelization of the property
(*e.g.*, Unlinkability)

$!\nu\,\mathsf{id}\,!\nu\,\mathsf{Sess}.\ P \approx !\nu\,\mathsf{id}.\,\nu\,\mathsf{Sess}.\ P$

Introduction: Formal Methods for Security Protocols



Introduction: Formal Methods for Security Protocols



State Space Explosion Problem

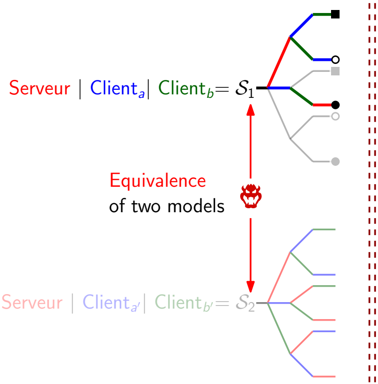
↓

Limited Practical Impact

↓

Detect Redundant States

Introduction: Formal Methods for Security Protocols



State Space Explosion Problem

Limited Practical Impact

Detect and Eliminate Redundant States

Problem

Issue: Limited practical impact

Too slow. – Bottleneck: state space explosion

e.g., verification of P.A.: 1 session $\rightarrow$ 1 sec. vs. 2 sessions $\rightarrow$ 9 days

Problem

Issue: Limited practical impact

Too slow. – Bottleneck: state space explosion

e.g., verification of P.A.: 1 session $\rightarrow$ 1 sec. vs. 2 sessions $\rightarrow$ 9 days

Our Contribution

Partial Order Reduction techniques:

- adequate with respect to specificities of security setting
- work for reachability and trace equivalence
- very effective in practice (implem $+$ bench)

Outline

I Model

II Big Picture

III Compression

IV Reduction

V Applications

VI Conclusion

I : Model

Applied-π - Term Algebra

Model of messages:

- Exchanged messages $=$ terms
- Crypto. primitives $=$ algebraic relations

Applied-π - Term Algebra

Model of messages:

- Exchanged messages $=$ terms
- Crypto. primitives $=$ algebraic relations

Terms Algebra: signature $+$ equational theory.

Example: symmetric encryption

- symbols: $\mathsf{enc}(\circ, \circ),\ \mathsf{dec}(\circ, \circ)$
- equation: $\mathsf{dec}(\mathsf{enc}(x, y), y) =_{\mathsf{E}} x$

Applied-π - Syntax

Protocols $\rightsquigarrow$ process calculus (*i.e.,* applied pi calculus)

- Process: P, Q

P, Q	$:=$	0	null
	$\mid$	$\mathrm{in}(c, x).P$	input
	$\mid$	$\mathrm{out}(c, m).P$	output
	$\mid$	$\texttt{if}\ u = v\ \texttt{then}\ P\ \texttt{else}\ Q$	conditional
	$\mid$	$P \mid Q$	parallel
	$\mid$	$!\,\nu\vec{n}.P$	replication

Applied-π - Syntax

Protocols $\leadsto$ process calculus (*i.e.,* applied pi calculus)

- **Process:**

$$
\begin{array}{lcll}
P, Q & := & 0 & \text{null} \\
 & \mid & \mathrm{in}(c, x).P & \text{input} \\
 & \mid & \mathrm{out}(c, m).P & \text{output} \\
 & \mid & \texttt{if } u = v \texttt{ then } P \texttt{ else } Q & \text{conditional} \\
 & \mid & P \mid Q & \text{parallel} \\
 & \mid & !\, \nu \vec{n}.P & \text{replication}
\end{array}
$$

- **Frame** (ϕ): the set of messages revelead to
  $\leadsto$ intuition: 's knowledge

$$\phi = \{ \underbrace{w_1}_{\text{handle}} \mapsto \underbrace{\mathrm{enc}(m, k)}_{\text{out. message}}; w_2 \mapsto k \}$$

Applied-π - Syntax

Protocols $\rightsquigarrow$ process calculus (*i.e.,* applied pi calculus)

- **Process:**

$$
\begin{array}{llll}
P, Q & := & 0 & \text{null} \\
 & \mid & \mathrm{in}(c, x).P & \text{input} \\
 & \mid & \mathrm{out}(c, m).P & \text{output} \\
 & \mid & \mathtt{if}\ u = v\ \mathtt{then}\ P\ \mathtt{else}\ Q & \text{conditional} \\
 & \mid & P \mid Q & \text{parallel} \\
 & \mid & !\ \nu \vec{n}.P & \text{replication}
\end{array}
$$

- **Frame** (ϕ): the set of messages revelead to
  $\rightsquigarrow$ intuition: 's knowledge

$$\phi = \{\underbrace{w_1}_{\text{handle}} \mapsto \underbrace{\mathrm{enc}(m, k)}_{\text{out. message}}; w_2 \mapsto k\}$$

- **Configuration:** $A = (\mathcal{P}; \phi)$

Applied-π - Semantics

- Recipes: are terms built using handles

$$e.g.,\quad \begin{array}{l} R = \mathrm{dec}(w_1, w_2) \\ R\phi =_{\mathsf{E}} m \end{array} \qquad \text{for } \phi = \{w_1 \mapsto \mathrm{enc}(m,k), w_2 \mapsto k\}$$

"How  builds messages from its knowledge"

Applied-π - Semantics

- Recipes: are terms built using handles

  $$e.g., \quad \begin{array}{l} R = \mathrm{dec}(w_1, w_2) \\ R\phi =_{\mathsf{E}} m \end{array} \qquad \text{for } \phi = \{w_1 \mapsto \mathrm{enc}(m,k), w_2 \mapsto k\}$$

  "How builds messages from its knowledge"

- Semantics of configurations:
  - Protocol's output:

  $$(\{\mathrm{out}(c,u).P\} \cup \mathcal{P}; \phi) \xrightarrow{\mathrm{out}(c,w)} (\{P\} \cup \mathcal{P}; \phi \cup \{w \mapsto u\}) \quad \text{if } w \text{ fresh}$$

   learns outputted message

Applied-π - Semantics

- Recipes: are terms built using handles

$$e.g., \quad \begin{array}{l} R = \mathrm{dec}(w_1, w_2) \\ R\phi =_{\mathsf{E}} m \end{array} \qquad \text{for } \phi = \{w_1 \mapsto \mathrm{enc}(m,k), w_2 \mapsto k\}$$

"How builds messages from its knowledge"

- Semantics of configurations:
  - Protocol's output:

$$(\{\mathrm{out}(c,u).P\} \cup \mathcal{P}; \phi) \xrightarrow{\mathrm{out}(c,w)} (\{P\} \cup \mathcal{P}; \phi \cup \{w \mapsto u\}) \quad \text{if } w \text{ fresh}$$

 learns outputted message

  - Protocol's input:

$$(\{\mathrm{in}(c,x).P\} \cup \mathcal{P}; \phi) \xrightarrow{\mathrm{in}(c,R)} (\{P\{x \mapsto R\phi\}\} \cup \mathcal{P}; \phi)$$

 injects any message he can builds

Applied-π - Semantics

- Recipes: are terms built using handles

$$e.g., \quad \begin{array}{l} R = \mathrm{dec}(w_1, w_2) \\ R\phi =_E m \end{array} \qquad \text{for } \phi = \{w_1 \mapsto \mathrm{enc}(m,k), w_2 \mapsto k\}$$

"How builds messages from its knowledge"

- Semantics of configurations:
  - Protocol's output:

$$(\{\mathrm{out}(c,u).P\} \cup \mathcal{P}; \phi) \xrightarrow{\mathrm{out}(c,w)} (\{P\} \cup \mathcal{P}; \phi \cup \{w \mapsto u\}) \quad \text{if } w \text{ fresh}$$

learns outputted message

  - Protocol's input:

$$(\{\mathrm{in}(c,x).P\} \cup \mathcal{P}; \phi) \xrightarrow{\mathrm{in}(c,R)} (\{P\{x \mapsto R\phi\}\} \cup \mathcal{P}; \phi)$$

injects any message he can builds

  - $+$ expected rules for conditional and other constructs

$\rightsquigarrow$ controls all the network

Applied-π - Trace Equivalence

1. Reachability (*e.g.,* secret, authentification) and
2. Trace equivalence (*e.g.,* anonymity, unlinkability).

Applied-π - Trace Equivalence

1. Reachability (*e.g.*, secret, authentification) and
2. Trace equivalence (*e.g.*, anonymity, unlinkability).

Static Equivalence (intuitively)

$\Phi \sim \Psi$ when

- $\mathrm{dom}(\Phi) = \mathrm{dom}(\Psi)$ and
- for all tests, it holds on $\Phi \iff$ it holds on Ψ

Applied-π - Trace Equivalence

1. Reachability (*e.g.,* secret, authentification) and
2. Trace equivalence (*e.g.,* anonymity, unlinkability).

Static Equivalence (intuitively)

$\Phi \sim \Psi$ when

- $\mathrm{dom}(\Phi) = \mathrm{dom}(\Psi)$ and
- for all tests, it holds on $\Phi \iff$ it holds on Ψ

Trace Equivalence

$A \sqsubseteq B$: for any $A \xrightarrow{\mathrm{tr}} A'$ there exists $B \xrightarrow{\mathrm{tr}} B'$ such that $\Phi(A') \sim \Phi(B')$.

$$A \approx B, \text{ when } A \sqsubseteq B \text{ and } B \sqsubseteq A.$$

(bisimulation: too strong)

II : Big Picture

Redundancies

- Motivation: Improve algorithms checking trace equivalence
- How: Remove redundant interleavings via a reduced semantics

Redundancies

- **Motivation:** Improve algorithms checking trace equivalence
- **How:** Remove redundant interleavings via a reduced semantics

Two types of redundancies:

1. $\mathrm{in}(c_1, x) \mid \mathrm{out}(c_2, m) \rightsquigarrow \begin{array}{l} \mathrm{tr}_1 = \mathrm{out}(c_2, w).\mathrm{in}(c_1, M) \\ \mathrm{tr}_2 = \mathrm{in}(c_1, M).\mathrm{out}(c_2, w) \end{array}$

Redundancies

- **Motivation:** Improve algorithms checking trace equivalence
- **How:** Remove redundant interleavings via a reduced semantics

Two types of redundancies:

1. $\mathrm{in}(c_1, x) \mid \mathrm{out}(c_2, m) \rightsquigarrow \begin{array}{l} \mathrm{tr}_1 = \mathrm{out}(c_2, w).\mathrm{in}(c_1, M) \\ \cancel{\mathrm{tr}_2 = \mathrm{in}(c_1, M).\mathrm{out}(c_2, w)} \end{array}$

Redundancies

- Motivation: Improve algorithms checking trace equivalence
- How: Remove redundant interleavings via a reduced semantics

Two types of redundancies:

1. $\mathrm{in}(c_1, x) \mid \mathrm{out}(c_2, m) \rightsquigarrow$ $\mathrm{tr}_1 = \mathrm{out}(c_2, w).\mathrm{in}(c_1, M)$ $\cancel{\mathrm{tr}_2 = \mathrm{in}(c_1, M).\mathrm{out}(c_2, w)}$
2. $\mathrm{in}(c_1, x).\mathrm{out}(c_1, m_1) \mid \mathrm{in}(c_2, y).\mathrm{out}(c_2, m_2) \rightsquigarrow$
   - $\mathrm{tr}_1 = \mathrm{in}(c_1, M_1).\mathrm{out}(c_1, w_1).\mathrm{in}(c_2, M_2).\mathrm{out}(c_2, w_2)$
   - $\mathrm{tr}_2 = \mathrm{in}(c_2, M_2).\mathrm{out}(c_2, w_2).\mathrm{in}(c_1, M_1).\mathrm{out}(c_1, w_1)$

Redundancies

- Motivation: Improve algorithms checking trace equivalence
- How: Remove redundant interleavings via a reduced semantics

Two types of redundancies:

1. $\mathrm{in}(c_1, x) \mid \mathrm{out}(c_2, m) \rightsquigarrow$ $\mathrm{tr}_1 = \mathrm{out}(c_2, w).\mathrm{in}(c_1, M)$
   $\cancel{\mathrm{tr}_2 = \mathrm{in}(c_1, M).\mathrm{out}(c_2, w)}$

2. $\mathrm{in}(c_1, x).\mathrm{out}(c_1, m_1) \mid \mathrm{in}(c_2, y).\mathrm{out}(c_2, m_2) \rightsquigarrow$
   - $\mathrm{tr}_1 = \mathrm{in}(c_1, M_1).\mathrm{out}(c_1, w_1).\mathrm{in}(c_2, M_2).\mathrm{out}(c_2, w_2)$
   - $\cancel{\mathrm{tr}_2 = \mathrm{in}(c_2, M_2).\mathrm{out}(c_2, w_2).\mathrm{in}(c_1, M_1).\mathrm{out}(c_1, w_1)}$
     when M_1 does not use w_2

Redundancies

- **Motivation:** Improve algorithms checking trace equivalence
- **How:** Remove redundant interleavings via a reduced semantics

Two types of redundancies:

1. $\mathsf{in}(c_1, x) \mid \mathsf{out}(c_2, m) \rightsquigarrow$ $\mathsf{tr}_1 = \mathsf{out}(c_2, w).\mathsf{in}(c_1, M)$; $\cancel{\mathsf{tr}_2 = \mathsf{in}(c_1, M).\mathsf{out}(c_2, w)}$

2. $\mathsf{in}(c_1, x).\mathsf{out}(c_1, m_1) \mid \mathsf{in}(c_2, y).\mathsf{out}(c_2, m_2) \rightsquigarrow$
   - $\mathsf{tr}_1 = \mathsf{in}(c_1, M_1).\mathsf{out}(c_1, w_1).\mathsf{in}(c_2, M_2).\mathsf{out}(c_2, w_2)$
   - $\cancel{\mathsf{tr}_2 = \mathsf{in}(c_2, M_2).\mathsf{out}(c_2, w_2).\mathsf{in}(c_1, M_1).\mathsf{out}(c_1, w_1)}$
   when M_1 does not use w_2

- what about trace equivalence ($\approx$) ?

$$\text{e.g., } (\mathsf{in}(c_1, x) \mid \mathsf{out}(c_2, m)) \not\approx (\mathsf{out}(c_2, m).\mathsf{in}(c_1, x))$$

Redundancies

- Motivation: Improve algorithms checking trace equivalence
- How: Remove redundant interleavings via a reduced semantics

Two types of redundancies:

1. $\mathrm{in}(c_1, x) \mid \mathrm{out}(c_2, m) \rightsquigarrow$ $\mathrm{tr}_1 = \mathrm{out}(c_2, w).\mathrm{in}(c_1, M)$
   $\cancel{\mathrm{tr}_2 = \mathrm{in}(c_1, M).\mathrm{out}(c_2, w)}$

2. $\mathrm{in}(c_1, x).\mathrm{out}(c_1, m_1) \mid \mathrm{in}(c_2, y).\mathrm{out}(c_2, m_2) \rightsquigarrow$
   - $\mathrm{tr}_1 = \mathrm{in}(c_1, M_1).\mathrm{out}(c_1, w_1).\mathrm{in}(c_2, M_2).\mathrm{out}(c_2, w_2)$
   - $\cancel{\mathrm{tr}_2 = \mathrm{in}(c_2, M_2).\mathrm{out}(c_2, w_2).\mathrm{in}(c_1, M_1).\mathrm{out}(c_1, w_1)}$
     when M_1 does not use w_2

- what about trace equivalence ($\approx$) ?

  e.g., $(\mathrm{in}(c_1, x) \mid \mathrm{out}(c_2, m)) \not\approx (\mathrm{out}(c_2, m).\mathrm{in}(c_1, x))$
- $\rightsquigarrow$ same swaps are possible ($\equiv$ same sequential dependencies)

Big Picture

$$
\boxed{\begin{array}{c} \longrightarrow \\ \approx \end{array}}
\xRightarrow[\text{Strong Sym: } \approx = \approx_a]{\text{Annot. Sem.}}
\boxed{\begin{array}{c} \longrightarrow_a \\ \approx_a \end{array}}
\xRightarrow[\text{Theorem 1: } \approx_a = \approx_c]{\text{Compression}}
\boxed{\begin{array}{c} \longrightarrow_c \\ \approx_c \end{array}}
\xRightarrow[\text{Theorem 2: } \approx_c = \approx_r]{\text{Reduction}}
\boxed{\begin{array}{c} \longrightarrow_r \\ \approx_r \end{array}}
$$

Required properties

$\rightarrow_r$ is such that:

- reachability properties coincide on $\rightarrow_r$ and $\rightarrow$;
- for **action-determinate** processes, trace-equivalence coincides on $\rightarrow_r$ and $\rightarrow$.

Big Picture

$$\boxed{\begin{array}{c}\longrightarrow \\ \approx\end{array}} \xRightarrow[\text{Strong Sym: } \approx = \approx_a]{\text{Annot. Sem.}} \boxed{\begin{array}{c}\longrightarrow_a \\ \approx_a\end{array}} \xRightarrow[\text{Theorem 1: } \approx_a = \approx_c]{\text{Compression}} \boxed{\begin{array}{c}\longrightarrow_c \\ \approx_c\end{array}} \xRightarrow[\text{Theorem 2: } \approx_c = \approx_r]{\text{Reduction}} \boxed{\begin{array}{c}\longrightarrow_r \\ \approx_r\end{array}}$$

Required properties

$\rightarrow_r$ is such that:

- reachability properties coincide on $\rightarrow_r$ and $\rightarrow$;
- for **action-determinate** processes, trace-equivalence coincides on $\rightarrow_r$ and $\rightarrow$.

Action-determinism

A is *action-deterministic* if: two actions in parallel must be $\neq$

Attacker knows to/from whom he is sending/receiving messages.

Big Picture

$$\boxed{\begin{array}{c}\longrightarrow\\ \approx\end{array}} \xRightarrow[\text{Strong Sym: } \approx = \approx_a]{\text{Annot. Sem.}} \boxed{\begin{array}{c}\longrightarrow_a\\ \approx_a\end{array}} \xRightarrow[\text{Theorem 1: } \approx_a = \approx_c]{\text{Compression}} \boxed{\begin{array}{c}\longrightarrow_c\\ \approx_c\end{array}} \xRightarrow[\text{Theorem 2: } \approx_c = \approx_r]{\text{Reduction}} \boxed{\begin{array}{c}\longrightarrow_r\\ \approx_r\end{array}}$$

Required properties

$\rightarrow_r$ is such that:

- reachability properties coincide on $\rightarrow_r$ and $\rightarrow$;
- for **action-determinate** processes, trace-equivalence coincides on $\rightarrow_r$ and $\rightarrow$.

Action-determinism

A is *action-deterministic* if: two actions in parallel must be $\neq$

Attacker knows to/from whom he is sending/receiving messages.

D. Baelde, S. Delaune and L. Hirschi: Partial Order Reduction for Security Protocols. CONCUR'15

D. Baelde, S. Delaune and L. Hirschi: A reduced semantics for deciding trace equivalence using constraint systems. POST'14

Outline

Annotated Semantics

- embeds labels into produced actions
- one can extract sequential dependencies from labelled actions

e.g., $\mathrm{in}(c_1, x) \mid \mathrm{out}(c_2, m) \xrightarrow{[\mathrm{out}(c_2,w)]^{1.2}.[\mathrm{in}(c_1,M_1)]^{1.1}}_a \cdot$ labels: in parallel

while $\mathrm{out}(c_2, m).\mathrm{in}(c_1, x) \xrightarrow{[\mathrm{out}(c_2,w)]^{1}.[\mathrm{in}(c_1,M_1)]^{1}}_a \cdot$ labels: in sequence

Annotated Semantics

- embeds labels into produced actions
- one can extract sequential dependencies from labelled actions

e.g., $\mathrm{in}(c_1, x) \mid \mathrm{out}(c_2, m) \xrightarrow{[\mathrm{out}(c_2,w)]^{1.2}.[\mathrm{in}(c_1,M_1)]^{1.1}}_a \cdot$ labels: in parallel

while $\mathrm{out}(c_2, m).\mathrm{in}(c_1, x) \xrightarrow{[\mathrm{out}(c_2,w)]^{1}.[\mathrm{in}(c_1,M_1)]^{1}}_a \cdot$ labels: in sequence

Strong Symmetry Lemma

- mismatch on labels $\rightsquigarrow$ systematically used to show $\not\approx$
- for action-deterministic, ($\approx +$ labels) coincides with $\approx$

Compression - Intuitions

The Idea

Follow a particular **strategy** that reduces the number of choices by looking at the **nature** of available actions.

Polarities of processes:

- *negative*: $\mathrm{out}().P, (P_1 \mid P_2), 0$
  Bring new data or choices, execution **independent** on the context

Compression - Intuitions

The Idea

Follow a particular **strategy** that reduces the number of choices by looking at the **nature** of available actions.

Polarities of processes:

- *negative*: $\mathrm{out}().P, (P_1 \mid P_2), 0$
  Bring new data or choices, execution **independent** on the context
- *positive*: $\mathrm{in}().P$
  Execution **depends** on the context

Compression - Intuitions

The Idea

Follow a particular **strategy** that reduces the number of choices by looking at the **nature** of available actions.

Polarities of processes:

- *negative*: $\text{out}().P, (P_1 \mid P_2), 0$
  Bring new data or choices, execution **independent** on the context
  $\rightsquigarrow$ to be performed as soon as possible in a given order
- *positive*: $\text{in}().P$
  Execution **depends** on the context

Compression - Intuitions

The Idea

Follow a particular **strategy** that reduces the number of choices by looking at the **nature** of available actions.

Polarities of processes:

- *negative*: $\mathrm{out}().P, (P_1 \mid P_2), 0$
  Bring new data or choices, execution **independent** on the context
  $\rightsquigarrow$ to be performed as soon as possible in a given order
- *positive*: $\mathrm{in}().P$
  Execution **depends** on the context
  $\rightsquigarrow$ can be performed only if no *negative*

Compression - Intuitions

The Idea

Follow a particular **strategy** that reduces the number of choices by looking at the **nature** of available actions.

Polarities of processes:

- *negative*: $\mathrm{out}().P, (P_1 \mid P_2), 0$
  Bring new data or choices, execution **independent** on the context
  $\rightsquigarrow$ to be performed as soon as possible in a given order
- *positive*: $\mathrm{in}().P$
  Execution **depends** on the context
  $\rightsquigarrow$ can be performed only if no *negative*
  $\rightsquigarrow$ **choose** one *positive*, put it **under focus**
  $\rightsquigarrow$ focus released when *negative*

Compression - Intuitions

The Idea

Follow a particular **strategy** that reduces the number of choices by looking at the **nature** of available actions.

Polarities of processes:

- *negative*: $\mathrm{out}().P, (P_1 \mid P_2), 0$
  Bring new data or choices, execution **independent** on the context
  $\rightsquigarrow$ to be performed as soon as possible in a given order
- *positive*: $\mathrm{in}().P$
  Execution **depends** on the context
  $\rightsquigarrow$ can be performed only if no *negative*
  $\rightsquigarrow$ **choose** one *positive*, put it **under focus**
  $\rightsquigarrow$ focus released when *negative*

(Replication: $!\, \nu \overrightarrow{n}.\ P$ is *positive* but releases the focus)

Compression - Example

$$\mathcal{P} = \{ \quad !\ \nu n.\ \mathsf{in}(c, x).\mathsf{out}(c, \mathsf{enc}(\langle x, n \rangle\}, k)).0 \quad \}$$

Compressed interleavings:
$t =$

Compression - Example

$$\mathcal{P} = \{ \quad !\nu n.\ \mathrm{in}(c, x).\mathrm{out}(c, \{< x, n >\}_k).0; \\ \boxed{\mathrm{in}(\mathrm{c}_1, x).\mathrm{out}(c_1, \mathrm{enc}(\langle x, n_1\rangle, k)).0} \quad \}$$

Compressed interleavings:

$t = \mathrm{sess}(a, c_1)$

Compression - Example

$$\mathcal{P} = \{ \quad !\nu n.\ \mathsf{in}(c, x).\mathsf{out}(c, \{< x, n >\}_k).0; \quad \mathsf{out}(c_1, \mathsf{enc}(\langle x, n_1 \rangle, k)).0 \quad \}$$

Compressed interleavings:
$t = \mathsf{sess}(a, c_1).\mathsf{in}(c_1, M_1)$

Compression - Example

$$\mathcal{P} = \{!\nu n.\ \mathrm{in}(c, x).\mathrm{out}(c, \{< x, n >\}_k).0\}$$

Compressed interleavings:

$t = \ \mathrm{sess}(a, c_1).\mathrm{in}(c_1, X_1).\mathrm{out}(c_1, w_1)$

Compression - Example

$$\mathcal{P} = \{!\nu n.\ \mathsf{in}(c,x).\mathsf{out}(c,\{<x,n>\}_k).0\}$$

Compressed interleavings:
$t =\ \mathsf{sess}(a,c_1).\mathsf{in}(c_1,X_1).\mathsf{out}(c_1,w_1)$

Only traces of the form:
$\mathsf{sess}_1.\mathsf{in}_1.\mathsf{out}_1.\ \mathsf{sess}_2.\mathsf{in}_2.\mathsf{out}_2.\ \ldots$

Compression - Results

Reachability:

- Soundness: $A \xrightarrow{t}_c A' \;\Rightarrow\; A \xrightarrow{t} A'$
- Completeness: for complete execution $A \xrightarrow{t} A' \;\Rightarrow$ $\exists t_c,$ permutation of $t,\; A \xrightarrow{t_c}_c A'$

Compression - Results

Reachability:

- Soundness: $A \xrightarrow{t}_c A' \;\Rightarrow\; A \xrightarrow{t} A'$
- Completeness: for complete execution $A \xrightarrow{t} A' \;\Rightarrow$ $\exists t_c,$ permutation of t, $A \xrightarrow{t_c}_c A'$

Equivalence:

Theorem: $\approx_c = \approx$

Let A and B be two action-deterministic configurations.

$$A \approx B \text{ if, and, only if, } A \approx_c B.$$

Outline

Reduction - Intuitions

By building upon $\rightarrow_c, \approx_c$:

- compressed semantics produces *blocks* of actions of the form:

$$b = (\mathit{sess}).\mathit{in} \ldots \mathit{in}.\mathit{out} \ldots \mathit{out}$$

- but we still need to make choices (which *positive* process/block?)
- some of them are redundant.

Reduction - Intuitions

By building upon $\rightarrow_c, \approx_c$:

- compressed semantics produces *blocks* of actions of the form:

$$b = (sess).in \ldots in.out \ldots out$$

- but we still need to make choices (which *positive* process/block?)
- some of them are redundant.

$$P = \text{in}(c_1, x).\text{out}(c_1, m_1) \mid \text{in}(c_2, y).\text{out}(c_2, m_2)$$

Compressed traces:

- $\text{tr}_1 = \text{in}(c_1, M_1).\text{out}(c_1, w_1).\text{in}(c_2, M_2).\text{out}(c_2, w_2)$
- $\cancel{\text{tr}_2 = \text{in}(c_2, M_2).\text{out}(c_2, w_2).\text{in}(c_1, M_1).\text{out}(c_1, w_1)}$
  when M_1 does not use w_2

Reduction - Monoid of traces

Definition

Given a frame Φ, the relation $\equiv_\Phi$ is the smallest equivalence over compressed traces such that:

- $t.b_1.b_2.t' \equiv_\Phi t.b_2.b_1.t'$ when $b_1 \parallel b_2$, and
- $t.b_1.t' \equiv_\Phi t.b_2.t'$ when $(b_1 =_E b_2)\Phi$.

Reduction - Monoid of traces

Definition

Given a frame Φ, the relation $\equiv_\Phi$ is the smallest equivalence over compressed traces such that:

- $t.b_1.b_2.t' \equiv_\Phi t.b_2.b_1.t'$ when $b_1 \parallel b_2$, and
- $t.b_1.t' \equiv_\Phi t.b_2.t'$ when $(b_1 =_\mathsf{E} b_2)\Phi$.

Lemma

If $A \xrightarrow{t}_c A'$. Then $A \xrightarrow{t'}_c A'$ for any $t' \equiv_{\Phi(A')} t$.

Goal: explore one trace per equivalence class.

Reduced semantics

We assume an arbitrary order $\prec$ over blocks priority order.

Semantics (informal)

$$\frac{A \xrightarrow{t}_r A' \quad A' \xrightarrow{b}_c A''}{A \xrightarrow{t.b}_r A'} \quad \text{if } t \ltimes b$$

Informally, $t \ltimes b$ means:

> *there is no way to swap b towards the beginning of t before a block $b_0 \succ b$ (even by modifying recipes)*

Reduced semantics

We assume an arbitrary order $\prec$ over blocks priority order.

Semantics (informal)

$$\frac{A \xrightarrow{t}_r A' \quad A' \xrightarrow{b}_c A''}{A \xrightarrow{t.b}_r A'} \quad \text{if } t \ltimes b$$

Informally, $t \ltimes b$ means:

> *there is no way to swap b towards the beginning of t before a block $b_0 \succ b$ (even by modifying recipes)*

t is Φ-minimal if there is no $t' \equiv_\Phi t$ such that $t' \prec_{\text{lex}} t$

If $A \xrightarrow{t}_c A'$ then t is $\Phi(A')$-minimal if, and only if, $A \xrightarrow{t}_r A'$.

Theorem

$\approx = \approx_r$ for action-deterministic configurations.

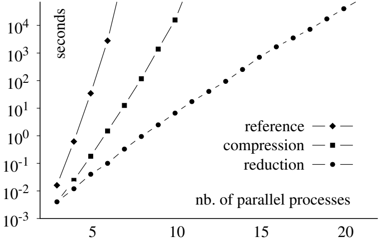
V : Applications

Benchmarks

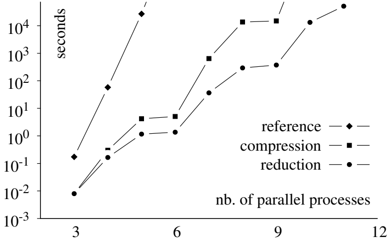
We implemented compression/reduction in APTE by adapting well established techniques based on:

- symbolic semantics (abstract inputs);
- constraint solving procedures.

tr$\ltimes$*b*: a new type of constraints



Toy example



Wide Mouthed Frog

All benchmarks & instructions for reproduction:
`www.lsv.ens-cachan.fr/~hirschi/apte_por`

VI : Conclusion

Conclusion

- New optimizations: compression and reduction;
- applied to trace equivalence checking;
- implementation in APTE.

Conclusion

- New optimizations: compression and reduction;
- applied to trace equivalence checking;
- implementation in APTE.

Future Work

1. drop action-deterministic assumption
2. impact of the choice of $\prec$
3. POR for backward research
4. study others redundancies $\rightsquigarrow$ recognize symmetries ?

Conclusion

- New optimizations: compression and reduction;
- applied to trace equivalence checking;
- implementation in APTE.

Future Work

1. drop action-deterministic assumption
2. impact of the choice of $\prec$
3. POR for backward research
4. study others redundancies $\rightsquigarrow$ recognize symmetries ?

Any question?

Compressed semantics - Definition

$\mathcal{P}$ is **initial** if $\forall P \in \mathcal{P}$, P is *positive*or replicated.

Semantics:

Compressed semantics - Definition

$\mathcal{P}$ is **initial** if $\forall P \in \mathcal{P}$, P is *positive*or replicated.

Semantics:

$$\text{START/IN} \qquad \frac{\mathcal{P} \text{ is initial} \quad (P; \Phi) \xrightarrow{\mathrm{in}(c,M)} (P'; \Phi)}{(\mathcal{P} \uplus \{P\}; \varnothing; \Phi) \xrightarrow{\mathtt{foc}(\mathrm{in}(c,M))}_c (\mathcal{P}; P'; \Phi)}$$

$$\text{POS/IN} \qquad \frac{(P; \Phi) \xrightarrow{\mathrm{in}(c,M)} (P'; \Phi)}{(\mathcal{P}; P; \Phi) \xrightarrow{\mathrm{in}(c,M)}_c (\mathcal{P}; P'; \Phi)}$$

Compressed semantics - Definition

$\mathcal{P}$ is **initial** if $\forall P \in \mathcal{P}$, P is *positive*or replicated.

Semantics:

START/IN
$$\frac{\mathcal{P} \text{ is initial} \quad (P;\Phi) \xrightarrow{\text{in}(c,M)} (P';\Phi)}{(\mathcal{P} \uplus \{P\};\varnothing;\Phi) \xrightarrow{\mathtt{foc}(\text{in}(c,M))}_c (\mathcal{P};P';\Phi)}$$

POS/IN
$$\frac{(P;\Phi) \xrightarrow{\text{in}(c,M)} (P';\Phi)}{(\mathcal{P};P;\Phi) \xrightarrow{\text{in}(c,M)}_c (\mathcal{P};P';\Phi)}$$

RELEASE
$$\frac{P \text{ negative}}{(\mathcal{P};P;\Phi) \xrightarrow{\mathtt{rel}}_c (\mathcal{P} \uplus \{P\};\varnothing;\Phi)}$$

NEG/α
$$\frac{(\{P\};\Phi) \xrightarrow{\alpha} (\mathcal{P}';\Phi')}{(\mathcal{P} \uplus \{P\};\varnothing;\Phi) \xrightarrow{\alpha}_c (\mathcal{P} \uplus \mathcal{P}';\varnothing;\Phi')} \quad \alpha \in \{\mathtt{par}, \mathtt{zero}, \text{out}(\_,\_)\}$$

+ Repl/In

Reduced semantics

We assume an arbitrary order $\prec$ over blocks (without recipes/messages): priority order.

Semantics

$$\frac{}{A \xrightarrow{\epsilon}_r A}$$

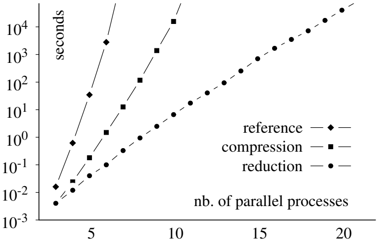
$$\frac{A \xrightarrow{\mathrm{tr}}_r (\mathcal{P}; \varnothing; \Phi) \quad (\mathcal{P}; \varnothing; \Phi) \xrightarrow{b}_c A'}{A \xrightarrow{\mathrm{tr}.b}_r A'} \quad \text{if } \mathrm{tr} \ltimes b' \text{ for all } b' \text{ with } (b' =_E b)\Phi$$

Availability

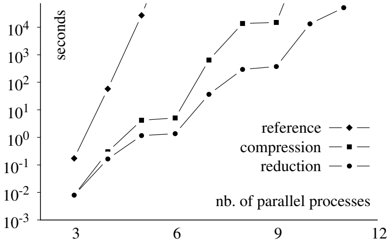
A block b is *available* after tr, denoted $\mathrm{tr} \ltimes b$, if:

- either $\mathrm{tr} = \epsilon$
- or $\mathrm{tr} = \mathrm{tr}_0.b_0$ with $\neg(b_0 \| b)$
- or $\mathrm{tr} = \mathrm{tr}_0.b_0$ with $b_0 \| b$, $b_0 \prec b$ and $\mathrm{tr}_0 \ltimes b$.

Benchmarks

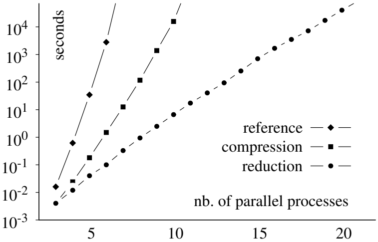


Toy example (Π_i(in.out))

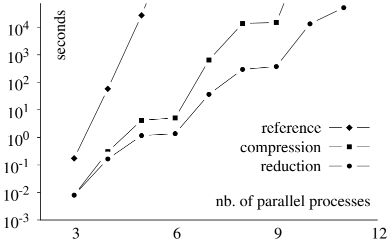


Wide Mouthed Frog

Benchmarks



Toy example ($\Pi_i(in.out)$)



Wide Mouthed Frog

Maximum number of parallel processes verifiable in 20 hours:

Protocol	ref	comp	red
Yahalom (3-party)	4	5	5
Needham Schroeder (3-party)	4	6	7
Private Authentication (2-party)	4	7	7
E-Passport PA (2-party)	4	7	9
Denning-Sacco (3-party)	5	9	10
Wide Mouthed Frog (3-party)	6	12	13

Instructions for reproduction:

`www.lsv.ens-cachan.fr/~hirschi/apte_por`