Partial Order Reduction for Security Protocols

CONCUR'15

Lucca Hirschi

LSV, ENS Cachan

September 4th 2015

joint work with David Baelde (LSV) *and* Stéphanie Delaune (LSV)

Introduction 1/2

concurrent programs + unsecure network + active attacker
→ (tricky) attacks

⇝ we need formal verification of crypto protocols

Introduction 1/2

concurrent programs + unsecure network + active attacker
$\rightarrow$ (tricky) attacks

$\rightsquigarrow$ we need formal verification of crypto protocols

Our setting

- Applied-π models protocols (π-calculus for crypto);
- Trace equivalence models security properties.

Introduction 1/2

concurrent programs + unsecure network + active attacker
→ (tricky) attacks

⇝ we need **formal verification** of crypto protocols

Our setting

- **Applied-π** models protocols (π-calculus for crypto);
- **Trace equivalence** models security properties.

⇝ **existing algorithms** checking trace equivalence without replication

Introduction 2/2

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

Introduction 2/2

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 this setting
- work for reachability and trace equivalence
- very effective in practice (implem $+$ bench)

Applied-π - Syntax

Terms

$\mathcal{T}$: set of terms + equational theory. *e.g.,* $\mathsf{dec}(\mathsf{enc}(m, k), k) =_{\mathsf{E}} m$.

Applied-π - Syntax

Terms

$\mathcal{T}$: set of terms + equational theory. *e.g.,* $\mathsf{dec}(\mathsf{enc}(m,k),k) =_{\mathsf{E}} m$.

Processes and configurations

$$P, Q ::= 0 \mid (P|Q) \mid \mathsf{in}(c,x).P \mid \mathsf{out}(c,m).P \\ \mid \texttt{if } u = v \texttt{ then } P \texttt{ else } Q \\ \mid !\, \nu \vec{n}.P$$

$$A = (\mathcal{P}; \Phi)$$

- Φ is the set of messages revelead to the network; intuition: intruder's knowledge.

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

Applied-π - Syntax

Terms

$\mathcal{T}$: set of terms + equational theory. *e.g.,* $\mathsf{dec}(\mathsf{enc}(m,k),k) =_{\mathsf{E}} m$.

Processes and configurations

$$
\begin{aligned}
P, Q ::= 0 \mid (P|Q) \mid \mathsf{in}(c,x).P \mid \mathsf{out}(c,m).P \\
\mid \mathtt{if}\ u = v\ \mathtt{then}\ P\ \mathtt{else}\ Q \\
\mid\ !\ \nu \vec{n}.P
\end{aligned}
$$

$A = (\mathcal{P}; \Phi)$

- Φ is the set of messages revelead to the network; intuition: intruder's knowledge.

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

- recipes are terms built using handles

$$e.g., R = \mathsf{dec}(w_1, w_2) \qquad m =_{\mathsf{E}} R\Phi$$

intuition: how the environment builds messages from its knowledge

Applied-π - Semantics - Example

Informal presentation

$$\begin{array}{rcl} \text{Alice} \rightarrow \text{Server} & : & \text{enc}(k, k_{\text{AS}}) \\ \text{Server} \rightarrow \text{Bob} & : & \text{enc}(k, k_{\text{BS}}) \\ \text{Alice} \rightarrow \text{Bob} & : & \text{enc}(m, k) \end{array}$$

Applied-π - Semantics - Example

Informal presentation

$$\begin{aligned} \text{Alice} \to \text{Server} &: \text{enc}(k, k_{\text{AS}}) \\ \text{Server} \to \text{Bob} &: \text{enc}(k, k_{\text{BS}}) \\ \text{Alice} \to \text{Bob} &: \text{enc}(m, k) \end{aligned}$$

Configuration

```
  out(a,enc(k,kas)).out(a,enc(m,k))
| in(s,x). if enc(dec(x,kas),kas) = x
           then out(s,enc(dec(x,kas),kbs))
           else 0
| in(b,x) [...]
```

$$\Phi = \emptyset$$

$$t = \epsilon$$

Let us explore one possible trace.

Applied-π - Semantics - Example

Informal presentation

$$\begin{aligned} \text{Alice} \rightarrow \text{Server} &: \text{enc}(k, k_{\text{AS}}) \\ \text{Server} \rightarrow \text{Bob} &: \text{enc}(k, k_{\text{BS}}) \\ \text{Alice} \rightarrow \text{Bob} &: \text{enc}(m, k) \end{aligned}$$

Configuration

```
  out(a,enc(k,kas)).out(a,enc(m,k))
| in(s,x). if enc(dec(x,kas),kas) = x
           then out(s,enc(dec(x,kas),kbs))
           else 0
| in(b,x) [...]
```

$$\Phi = \{w_0 \mapsto enc(k, k_{\text{as}})\}$$

$$t = \text{out}(a, w_0)$$

Applied-π - Semantics - Example

Informal presentation

$$\begin{array}{rcl} \text{Alice} \rightarrow \text{Server} & : & \text{enc}(k, k_{\text{AS}}) \\ \text{Server} \rightarrow \text{Bob} & : & \text{enc}(k, k_{\text{BS}}) \\ \text{Alice} \rightarrow \text{Bob} & : & \text{enc}(m, k) \end{array}$$

Configuration

```
  out(a,enc(k,kas)).out(a,enc(m,k))
| in(s,x). if enc(dec(x,kas),kas) = x
           then out(s,enc(dec(x,kas),kbs))
           else 0
| in(b,x) [...]
```

$$\Phi = \{w_0 \mapsto enc(k, k_{\text{as}})\}$$

$$t = \text{out}(a, w_0).\text{in}(s, w_0)$$

w_0 is one possible recipe using Φ

Applied-π - Semantics - Example

Informal presentation

$$\begin{aligned} \text{Alice} \rightarrow \text{Server} &: \text{enc}(k, k_{\text{AS}}) \\ \text{Server} \rightarrow \text{Bob} &: \text{enc}(k, k_{\text{BS}}) \\ \text{Alice} \rightarrow \text{Bob} &: \text{enc}(m, k) \end{aligned}$$

Configuration

```
  out(a,enc(k,kas)).out(a,enc(m,k))
| in(s,x). if enc(dec(x,kas),kas) = x
           then out(s,enc(k,kbs))
           else 0
| in(b,x) [...]
```

$$\Phi = \{w_0 \mapsto enc(k, k_{\text{as}}); w_1 \mapsto enc(k, k_{\text{bs}})\}$$

$$t = \text{out}(a, w_0).\text{in}(s, w_0).\text{out}(s, w_1)$$

Applied-π - Semantics - Example

Informal presentation

$$\begin{array}{rcl} \text{Alice} \rightarrow \text{Server} & : & \text{enc}(k, k_{\mathrm{AS}}) \\ \text{Server} \rightarrow \text{Bob} & : & \text{enc}(k, k_{\mathrm{BS}}) \\ \text{Alice} \rightarrow \text{Bob} & : & \text{enc}(m, k) \end{array}$$

Configuration

```
  out(a,enc(k,kas)).out(a,enc(m,k))
| in(s,x). if enc(dec(x,kas),kas) = x
           then out(s,enc(k,kbs))
           else 0
| in(b,x) [...]
```

$$\Phi = \{w_0 \mapsto enc(k, k_{\mathrm{as}}); w_1 \mapsto enc(k, k_{\mathrm{bs}})\}$$

$$t = \text{out}(a, w_0).\text{in}(s, w_0).\text{out}(s, w_1).\text{in}(b, w_1)$$

Applied-π - Semantics - Example

Informal presentation

$$\begin{array}{rcl} \text{Alice} \rightarrow \text{Server} & : & \text{enc}(k, k_{AS}) \\ \text{Server} \rightarrow \text{Bob} & : & \text{enc}(k, k_{BS}) \\ \text{Alice} \rightarrow \text{Bob} & : & \text{enc}(m, k) \end{array}$$

Configuration

```
  out(a,enc(k,kas)).out(a,enc(m,k))
| in(s,x). if enc(dec(x,kas),kas) = x
           then out(s,enc(k,kbs))
           else 0
| in(b,x) [...]
```

$$\Phi = \{w_0 \mapsto enc(k, k_{as}); w_1 \mapsto enc(k, k_{bs}); w_2 \mapsto enc(m, k)\}$$

$$t = \text{out}(a, w_0).\text{in}(s, w_0).\text{out}(s, w_1).\text{in}(b, w_1).\text{out}(a, w_2)$$

Applied-π - Semantics - Example

Informal presentation

$$\begin{array}{rcl} \text{Alice} \rightarrow \text{Server} & : & \text{enc}(k, k_{\text{AS}}) \\ \text{Server} \rightarrow \text{Bob} & : & \text{enc}(k, k_{\text{BS}}) \\ \text{Alice} \rightarrow \text{Bob} & : & \text{enc}(m, k) \end{array}$$

Configuration

```
  out(a,enc(k,kas)).out(a,enc(m,k))
| in(s,x). if enc(dec(x,kas),kas) = x
           then out(s,enc(k,kbs))
           else 0
| in(b,x)  [...]
```

$$\Phi = \{w_0 \mapsto enc(k, k_{\text{as}}); w_1 \mapsto enc(k, k_{\text{bs}}); w_2 \mapsto enc(m, k)\}$$

$$t = \text{out}(a, w_0).\text{in}(s, w_0).\text{out}(s, w_1).\text{in}(b, w_1).\text{out}(a, w_2).\text{in}(b, w_2)$$

Security Properties

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

Security Properties

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

Trace equivalence

- $A \approx B$: $\forall A \xrightarrow{t} A' \; \exists B \xrightarrow{t} B'$ such that $\Phi_{A'} \sim \Phi_{B'}$ (and conversely)

Security Properties

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

Trace equivalence

- $A \approx B : \ \forall A \xrightarrow{t} A' \ \exists B \xrightarrow{t} B'$ such that $\Phi_{A'} \sim \Phi_{B'}$ (and conversely)
- $\Phi \sim \Phi' : \ (\forall M, N, \ M\Phi = N\Phi \iff M\Phi' = N\Phi')$

(bisimulation: too strong)

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

Redundancies

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

Two types of redundancies:

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

Redundancies

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

Two types of redundancies:

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

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

Redundancies

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

Two types of redundancies:

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

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

Redundancies

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

Two types of redundancies:

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

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

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

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

Redundancies

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

Two types of redundancies:

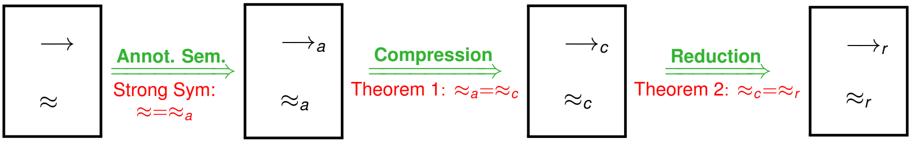
1. $\mathsf{in}(c_1, x) \mid \mathsf{out}(c_2, m) \rightsquigarrow$ $\mathsf{tr}_1 = \mathsf{out}(c_2, w).\mathsf{in}(c_1, M)$
   ~~$\mathsf{tr}_2 = \mathsf{in}(c_1, M).\mathsf{out}(c_2, w)$~~

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

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

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

Big Picture

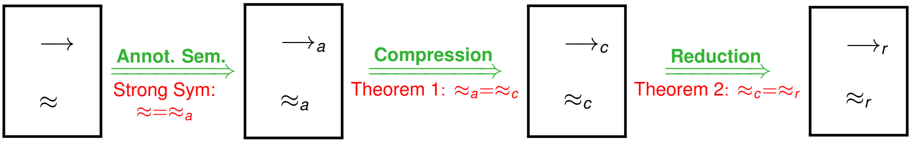


Required properties

$\rightarrow_r$ 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



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-determinsm

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

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

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

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

Compressed interleavings:

$t =$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.

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

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.

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

Compressed traces:

- $\mathrm{tr}_1 = \mathrm{in}(c_1, M_1).\mathrm{out}(c_1, w_1).\mathrm{in}(c_2, M_2).\mathrm{out}(c_2, w_2)$
- ~~$\mathrm{tr}_2 = \mathrm{in}(c_2, M_2).\mathrm{out}(c_2, w_2).\mathrm{in}(c_1, M_1).\mathrm{out}(c_1, w_1)$~~
  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

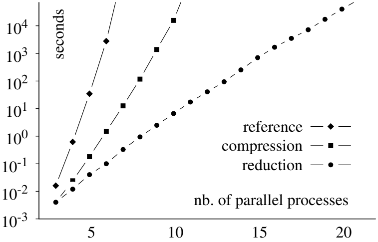
$\approx = \approx_r$ for action-deterministic configurations.

Benchmarks

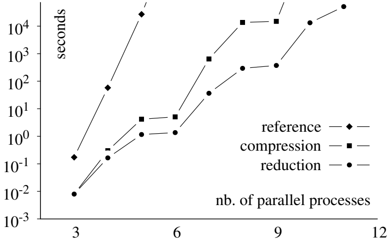
We implemented compression/reduction in APTE
by adapting well established techniques based on:

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

$\mathrm{tr} \ltimes b$: a new type of constraints



Toy example



Wide Mouthed Frog

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

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. 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. 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:

START/IN

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

POS/IN

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

Compressed semantics - Definition

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

Semantics:

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

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
$$\frac{P \textit{ negative}}{(\mathcal{P};P;\Phi) \xrightarrow{\texttt{rel}}_c (\mathcal{P} \uplus \{P\};\varnothing;\Phi)}$$

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

+ Repl/In

Reduced semantics

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

Semantics

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

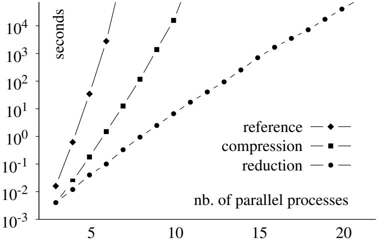
$$\frac{A \xrightarrow{\mathrm{tr}}_r (\mathcal{P};\varnothing;\Phi) \quad (\mathcal{P};\varnothing;\Phi) \xrightarrow{b}_c A'}{A \xrightarrow{\mathrm{tr}.b}_r A'}$$ if $\mathrm{tr} \ltimes b'$ for all b' with $(b' =_E b)\Phi$

Availability

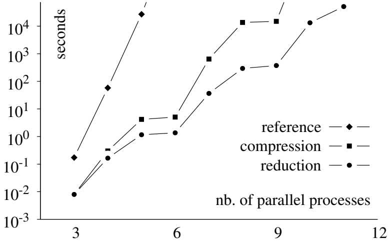
A block b is *available* after tr, denoted $\mathrm{tr} \ltimes b$, if:

- 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

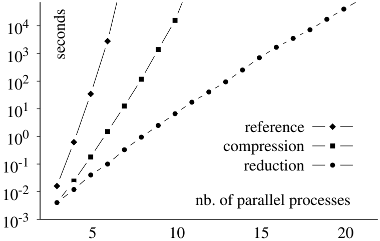


Toy example (Π_i(in.out))

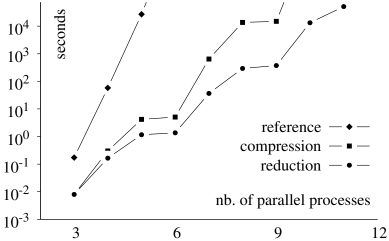


Wide Mouthed Frog

Benchmarks



Toy example (Π_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`