Partial order reduction for the applied $\pi$-calculus

CHoCoLa

Lucca Hirschi

LSV, ENS Cachan

November 13, 2014

joint work with David Baelde and Stéphanie Delaune

LSV and LSV
unsecure network, active attacker → attacks
~~ we need formal verification of crypto protocols
unsecure network, active attacker $\rightarrow$ attacks
$\leadsto$ we need formal verification of crypto protocols

Our setting
- **Applied-**$\pi$ models protocols;
- **Trace equivalence** models security properties.
unsecure network, active attacker $\rightarrow$ attacks
$\Rightarrow$ we need formal verification of crypto protocols

Our setting
- Applied-$\pi$ models protocols;
- Trace equivalence models security properties.

$\Rightarrow$ several algorithms resolve this problem (Akiss, Apte, Spec)

Issue: Limited practical impact
Too slow. Bottleneck: size of search space (interleavings).
Outline

1. Introduction
2. Model
3. Big Picture
4. Compression
5. Reduction
6. Conclusion
### Applied-$\pi$ - Syntax

#### Terms

$\mathcal{T}$: set of terms + equational theory. *e.g.*, $\text{dec}(\text{enc}(m, k), k) \equiv_{\text{E}} m$. 
Applied-$\pi$ - Syntax

Terms

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

Processes and configurations

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

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

> $\Phi$ is the set of messages revealed to the network; intuition: intruder’s knowledge.

$\Phi = \{ w_0 \rightarrow \text{enc}(m, k); w_1 \rightarrow k \}$

- handle
- out. message
Applied-$\pi$ - Syntax

Terms

$\mathcal{T}$: set of terms + equational theory. e.g., $\text{dec}(\text{enc}(m,k), k) \equiv_m m$.

Processes and configurations

$P, Q ::= 0 \mid (P \mid Q) \mid \text{in}(c,x).P \mid \text{out}(c,m).P \mid \text{if } u = v \text{ then } P \text{ else } Q \mid !P$

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

- $\Phi$ is the set of messages revealed to the network; intuition: intruder’s knowledge.

$$\Phi = \{ \underbrace{w_0} \rightarrow \underbrace{\text{enc}(m,k)}; \underbrace{w_1} \rightarrow k \}$$

- recipes are terms built using only handles

$$e.g., R = \text{dec}(w_0, w_1) \quad m \equiv R\Phi$$

intuition: how the environment builds messages from its knowledge.
Example - Wide Mouth Frog

**Informal presentation**

<table>
<thead>
<tr>
<th>Source</th>
<th>Encrypted Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice → Server</td>
<td>$\text{enc}(k', k_A)$</td>
</tr>
<tr>
<td>Server → Bob</td>
<td>$\text{enc}(k', k_B)$</td>
</tr>
<tr>
<td>Alice → Bob</td>
<td>$\text{enc}(m, k')$</td>
</tr>
</tbody>
</table>
Example - Wide Mouth Frog

Informal presentation

Alice → Server : $\text{enc}(k', k_A)$
Server → Bob : $\text{enc}(k', k_B)$
Alice → Bob : $\text{enc}(m, k')$

Process

\[
\text{out}(a, \text{enc}(k', ka)) . \text{out}(a, \text{enc}(m, k'))
| \text{in}(s, x). \text{if} \ \text{enc}(\text{dec}(x, ka), ka) = x
\quad \text{then} \ \text{out}(s, \text{enc}(\text{dec}(x, ka), kb))
\quad \text{else} \ 0
| \text{in}(b, x) [\ldots]
\]

$\Phi = \emptyset$

\[
t = \epsilon
\]

Let us explore one possible trace.
Example - Wide Mouth Frog

Informal presentation

Alice → Server : $\text{enc}(k', k_A)$
Server → Bob : $\text{enc}(k', k_B)$
Alice → Bob : $\text{enc}(m, k')$

Process

\[
\begin{align*}
\text{out}(a, \text{enc}(k', ka)).\text{out}(a, \text{enc}(m, k')) & \\
\text{in}(s, x). \text{if} \ \text{enc}(\text{dec}(x, ka), ka) = x & \\
\text{then} \ \text{out}(s, \text{enc}(\text{dec}(x, ka), kb)) & \\
\text{else} \ 0 & \\
\text{in}(b, x) & [\ldots] \\
\end{align*}
\]

$\Phi = \{w_0 \mapsto \text{enc}(k', ka)\}$

\[
t = \text{out}(a, w_0)
\]
Example - Wide Mouth Frog

Informal presentation

Alice $\rightarrow$ Server : $\text{enc}(k', k_A)$
Server $\rightarrow$ Bob : $\text{enc}(k', k_B)$
Alice $\rightarrow$ Bob : $\text{enc}(m, k')$

Process

\[
\begin{align*}
\text{out}(a, \text{enc}(k', ka)) & . \text{out}(a, \text{enc}(m, k')) \\
| \text{in}(s, x) & . \text{if} \ \text{enc}(\text{dec}(x, ka), ka) = x \\
& \text{then} \ \text{out}(s, \text{enc}(\text{dec}(x, ka), kb)) \\
& \text{else} \ 0 \\
| \text{in}(b, x) & \ldots
\end{align*}
\]

$\Phi = \{w_0 \mapsto \text{enc}(k', ka)\}$

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

$w_0$ is one possible recipe using $\Phi$
no other for then branch since the attacker does not know $k_A$
Example - Wide Mouth Frog

Informal presentation

Alice → Server : $\text{enc}(k', k_A)$
Server → Bob : $\text{enc}(k', k_B)$
Alice → Bob : $\text{enc}(m, k')$

Process

\[
\begin{align*}
\text{out}(a, \text{enc}(k', ka)) & . \text{out}(a, \text{enc}(m, k')) \\
| \text{in}(s, x) & . \text{if} \ \text{enc}(\text{dec}(x, ka), ka) = x \\
& \text{then} \ \text{out}(s, \text{enc}(k', kb)) \\
& \text{else} \ 0 \\
| \text{in}(b, x) & [...]
\end{align*}
\]

$\Phi = \{w_0 \mapsto \text{enc}(k', ka); w_1 \mapsto \text{enc}(k', kb)\}$

$t = \text{out}(a, w_0).\text{in}(s, w_0).\text{out}(s, w_1)$
Example - Wide Mouth Frog

Informal presentation

Alice → Server : enc(k', k_A)
Server → Bob : enc(k', k_B)
Alice → Bob : enc(m, k')

Process

\[
\text{out}(a, \text{enc}(k', ka)). \text{out}(a, \text{enc}(m, k'))
\]
\[
| \text{in}(s, x). \text{if} \ \text{enc}(\text{dec}(x, ka), ka) = x
\]
\[
\text{then} \ \text{out}(s, \text{enc}(k', kb))
\]
\[
\text{else} \ 0
\]
\[
| \text{in}(b, x) \ [\ldots]
\]

\[
\Phi = \{ w_0 \mapsto \text{enc}(k', ka); w_1 \mapsto \text{enc}(k', kb) \}
\]

\[
t = \text{out}(a, w_0). \text{in}(s, w_0). \text{out}(s, w_1). \text{in}(b, w_1)
\]
Example - Wide Mouth Frog

Informal presentation

Alice → Server : enc(k', k_A)
Server → Bob : enc(k', k_B)
Alice → Bob : enc(m, k')

Process

\[ \Phi = \{ w_0 \mapsto \text{enc}(k', k_a); w_1 \mapsto \text{enc}(k', k_b); w_2 \mapsto \text{enc}(m, k') \} \]

\[ t = \text{out}(a, w_0).\text{in}(s, w_0).\text{out}(s, w_1).\text{in}(b, w_1).\text{out}(a, w_2) \]
Example - Wide Mouth Frog

Informal presentation

Alice → Server : enc(k', k_A)
Server → Bob   : enc(k', k_B)
Alice → Bob    : enc(m, k')

Process

\[ \text{out}(a, \text{enc}(k', k_a)).\text{out}(a, \text{enc}(m, k')) \]
\[ | \text{in}(s, x). \text{if } \text{enc}(\text{dec}(x, k_a), k_a) = x \]
\[ \text{then } \text{out}(s, \text{enc}(k', k_b)) \]
\[ \text{else } 0 \]
\[ | \text{in}(b, x) \quad [...]] \]
\[ \Phi = \{w_0 \mapsto \text{enc}(k', k_a); w_1 \mapsto \text{enc}(k', k_b); w_2 \mapsto \text{enc}(m, k')\} \]
\[ t = \text{out}(a, w_0).\text{in}(s, w_0).\text{out}(s, w_1).\text{in}(b, w_1).\text{out}(a, w_2).\text{in}(b, w_2) \]
### Internal reduction $\rightsquigarrow$:

- $(\text{if } u = v \text{ then } P \text{ else } Q) \rightsquigarrow P \text{ when } u =_E v$;
- $(\text{if } u = v \text{ then } P \text{ else } Q) \rightsquigarrow Q \text{ when } u \neq_E v$;
- $(P \mid Q) \rightsquigarrow (P' \mid Q)$ and $(Q \mid P) \rightsquigarrow (Q \mid P')$ when $P \rightsquigarrow P'$;
- $((P_1 \mid P_2) \mid P_3) \rightsquigarrow (P_1 \mid (P_2 \mid P_3))$; not. $\Pi_{i=1}^3 P_i$
- $(P \mid 0) \rightarrow P$ and $(0 \mid P) \rightsquigarrow P$.
### Applied-$$\pi$$ - Semantics

#### Internal reduction $$\rightsquigarrow$$:

- $$(\text{if } u = v \text{ then } P \text{ else } Q) \rightsquigarrow P$$ when $$u =_E v$$;
- $$(\text{if } u = v \text{ then } P \text{ else } Q) \rightsquigarrow Q$$ when $$u \not= E v$$;
- $$(P \mid Q) \rightsquigarrow (P' \mid Q)$$ and $$(Q \mid P) \rightsquigarrow (Q \mid P')$$ when $$P \rightsquigarrow P'$$;
- $$((P_1 \mid P_2) \mid P_3) \rightsquigarrow (P_1 \mid (P_2 \mid P_3))$$; not. $$\Pi_{i=1}^3 P_i$$
- $$(P \mid 0) \rightarrow P$$ and $$(0 \mid P) \rightsquigarrow P$$.

<table>
<thead>
<tr>
<th>IN</th>
<th>$$({\text{in}(c, x).Q} \cup \mathcal{P}; \Phi)$$</th>
<th>$\xrightarrow{\text{in}(c,M)}$</th>
<th>$$({Q{u/x}} \cup \mathcal{P}; \Phi)$$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>where $$M \in \mathcal{T}(\text{dom}(\Phi))$$ and $$M\Phi =_E u$$</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>$$({\text{out}(c, u).Q} \cup \mathcal{P}; \Phi)$$</td>
<td>$\xrightarrow{\text{out}(c,w)}$</td>
<td>$$({Q} \cup \mathcal{P}; \Phi \cup {w \mapsto u})$$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>where $$w \in \mathcal{W}$$ is fresh</td>
<td></td>
</tr>
<tr>
<td>PAR</td>
<td>$${\Pi_{i=1}^n P_i} \cup \mathcal{P}; \Phi$$</td>
<td>$\xrightarrow{\text{par}}$</td>
<td>$${P_1; \ldots; P_n} \cup \mathcal{P}; \Phi$$</td>
</tr>
<tr>
<td>ZERO</td>
<td>$${0} \cup \mathcal{P}; \Phi$$</td>
<td>$\xrightarrow{\text{zero}}$</td>
<td>$$(\mathcal{P}; \Phi)$$</td>
</tr>
</tbody>
</table>
Trace Equivalence

Properties:
1. Reachability (e.g., secret, authentification) and
2. Trace equivalence (e.g., anonymity, unlikability).

(bisimulation: too strong)
Trace Equivalence

Properties:

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

(bisimulation: too strong)

Trace equivalence

\[ A \approx B \iff \forall A \xrightarrow{t} A', \exists B \xrightarrow{t'} B' \text{ such that } \text{obs}(t) = \text{obs}(t') \text{ and } \Phi_{A'} \sim \Phi_{B'} \text{ (and conversely)} \]
Trace Equivalence

Properties:

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

(bisimulation: too strong)

Trace equivalence

\[ A \approx B \iff \forall A \xrightarrow{t} A', \exists B \xrightarrow{t'} B' \text{ such that } \mathsf{obs}(t) = \mathsf{obs}(t') \text{ and } \Phi_{A'} \sim \Phi_{B'} \text{ (and conversely)} \]

\[ \Phi \sim \Phi' \iff (\forall M, N, M\Phi = N\Phi \iff M\Phi' = N\Phi') \]
Trace Equivalence

Properties:
1. Reachability (e.g., secret, authentication) and
2. Trace equivalence (e.g., anonymity, unlikability).

(bisimulation: too strong)

Trace equivalence

\[ A \approx B \iff \forall A \xrightarrow{t} A', \exists B \xrightarrow{t'} B' \text{ such that } \text{obs}(t) = \text{obs}(t') \text{ and } \Phi_{A'} \sim \Phi_{B'} (\text{and conversely}) \]

\[ \Phi \sim \Phi' \iff (\forall M, N, M\Phi = N\Phi \iff M\Phi' = N\Phi') \]

Example: unlinkability of WMF

\[
\begin{align*}
\text{Alice} & \rightarrow S \rightarrow \text{Bob} \\
\{P_a; P_s; P_b\} & \cup \{P_a; P_s; P_b\} \cup \epsilon
\end{align*}
\]

\[
\begin{align*}
\text{Alice} & \rightarrow S \rightarrow \text{Bob} \\
\{P_a; P_s; P_b\} & \cup \{P'_a; P'_s; P_c\} \cup \epsilon
\end{align*}
\]

Broken: reusing 1st session on the left, impossible on the right
Outline

1. Introduction
2. Model
3. Big Picture
4. Compression
5. Reduction
6. Conclusion
**Motivation:** Improve algorithms checking trace equivalence

**How:** Remove redundant interleavings via a reduced semantics

\[ \approx \overset{\text{Compression}}{\Rightarrow} \approx_c \overset{\text{Reduction}}{\Rightarrow} \approx_r \]

\[ \rightarrow_r \text{ does not explore all behaviours but sufficiently to ensure } \approx = \approx_r \]
Big Picture

\[
\begin{array}{ccc}
\rightarrow & \text{Compression} & \Rightarrow \\
\approx & \rightarrow c & \Rightarrow \\
\approx c & \rightarrow r & \approx r
\end{array}
\]

\textbf{Required properties}

\(\rightarrow_r\) is such that:

- reachability properties coincide on \(\rightarrow_r\) and \(\rightarrow\);
- for \textbf{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-determinism

$A$ is action-deterministic if $\forall A \xrightarrow{t} (P; \Phi), \forall P, Q \in \mathcal{P}, P$ and $Q$ cannot perform an observable action of the same nature on the same channel.

Makes sense in security (e.g., IP of agents)
Outline

1. Introduction
2. Model
3. Big Picture
4. Compression
5. Reduction
6. Conclusion
Intuitions

The Idea

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

Polarities of processes (similar to focusing of LL):

- **negative**: `\text{out}(\cdot). P, \Pi P_i, 0`
  
  Bring new **data** or **choices**, execution independent of the context
Intuitions

The Idea

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

Polarities of processes (similar to focusing of LL):

- **negative**: `out().P, \Pi P_i, 0`
  - Bring new *data* or *choices*, execution independent of the context

- **positive**: `in().P`
  - Execution *depends* on the context
Intuitions

The Idea

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

**Polarities** of processes (similar to focusing of LL):

- **negative**: `out().P, \Pi P_i, 0`
  Bring new **data** or **choices**, execution independent of the context
  \[\leadsto\] to be performed as soon as possible in a given order

- **positive**: `in().P`
  Execution **depends** on the context
Intuitions

The Idea

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

Polarities of processes (similar to focusing of LL):

- **negative**: `out().P, !P_i, 0`
  - Bring new **data** or **choices**, execution independent of the context
  - `⇝` to be performed as soon as possible in a given order

- **positive**: `in().P`
  - Execution **depends** on the context
  - `⇝` can be performed only if no **negative**
Intuitions

The Idea

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

Polarities of processes (similar to focusing of LL):

- **negative**: \texttt{out().P}, \texttt{ΠP_i}, 0
  Bring new data or choices, execution independent of the context
  \(\Rightarrow\) to be performed as soon as possible in a given order

- **positive**: \texttt{in().P}
  Execution depends on the context
  \(\Rightarrow\) can be performed only if no negative
  \(\Rightarrow\) we make a choice that we must maintain while it is positive
  \(\Rightarrow\) the chosen one is under focus, released when negative
Intuitions

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

Polarities of processes (similar to focusing of LL):

- **negative**: $\text{out}().P, \Pi P_i, 0$
  Bring new **data** or **choices**, execution independent of the context
  $\rightsquigarrow$ to be performed as soon as possible in a given order

- **positive**: $\text{in}().P$
  Execution **depends** on the context
  $\rightsquigarrow$ can be performed only if no **negative**
  $\rightsquigarrow$ we make a choice that we must maintain while it is **positive**
  $\rightsquigarrow$ the chosen one is **under focus**, released when **negative**
Compressed semantics - Definitions

\( \mathcal{P} \) is **initial** if \( \forall P \in \mathcal{P}, P \) is **positive**.

Semantics:
Compressed semantics - Definitions

\( \mathcal{P} \) is **initial** if \( \forall P \in \mathcal{P} \), \( P \) is **positive**.

Semantics:

\[
\begin{align*}
\text{START/IN} & \quad (\mathcal{P} \uplus \{P\}; \emptyset; \Phi) \xrightarrow{\text{foc}(\text{in}(c,M))} c (\mathcal{P}; P'; \Phi) \\
\text{POS/IN} & \quad (P; \Phi) \xrightarrow{\text{in}(c,M)} (P'; \Phi)
\end{align*}
\]
Compressed semantics - Definitions

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

Semantics:

- **START/IN**
  
  $\mathcal{P}$ is initial  
  \[
  (\mathcal{P} \cup \{P\}; \emptyset; \Phi) \xrightarrow{\text{foc}(\text{in}(c, M))} c (\mathcal{P}; P'; \Phi)
  \]

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

- **RELEASE**
  
  $(\mathcal{P}; P; \Phi) \xrightarrow{\text{rel}} c (\mathcal{P} \cup \{P\}; \emptyset; \Phi)$

- **NEG/\alpha**
  
  $(\{P\}; \Phi) \xrightarrow{\alpha} (\mathcal{P}'; \Phi')$

  \[
  (\mathcal{P} \cup \{P\}; \emptyset; \Phi) \xrightarrow{\alpha} c (\mathcal{P} \cup \mathcal{P}'; \emptyset; \Phi')
  \]

  $\alpha \in \{\text{par}, \text{zero}, \text{out}(\_, \_)\}$
Results - Reachability

Translations:

\[
[(\mathcal{P}; \Phi)] = (\mathcal{P}; \emptyset; \Phi), \quad [(\mathcal{P}; \emptyset; \Phi)] = (\mathcal{P}; \Phi), \quad [(\mathcal{P}; P; \Phi)] = (\mathcal{P} \uplus \{P\}; \Phi).
\]
Results - Reachability

Translations:

\[(\mathcal{P}; \Phi) = (\mathcal{P}; \emptyset; \Phi), \quad (\mathcal{P}; \emptyset; \Phi) = (\mathcal{P}; \Phi), \quad (\mathcal{P}; \mathcal{P}; \Phi) = (\mathcal{P} \uplus \{P\}; \Phi).\]

\[\lfloor \epsilon \rfloor = \epsilon, \quad \lfloor \text{foc}(\alpha).t \rfloor = \alpha.\lfloor t \rfloor, \quad \lfloor \text{rel}.t \rfloor = \lfloor t \rfloor, \quad \text{and} \]
\[\lfloor \alpha.t \rfloor = \alpha.\lfloor t \rfloor \text{ for any other } \alpha.\]
Results - Reachability

Translations:

\[
\begin{align*}
\lceil (\mathcal{P}; \Phi) \rceil &= (\mathcal{P}; \emptyset; \Phi), \\
\lfloor (\mathcal{P}; \emptyset; \Phi) \rfloor &= (\mathcal{P}; \emptyset), \\
\lfloor (\mathcal{P}; \mathcal{P}; \Phi) \rfloor &= (\mathcal{P} \cup \{ \mathcal{P} \}; \Phi).
\end{align*}
\]

\[
\begin{align*}
\lfloor \epsilon \rfloor &= \epsilon, \\
\lfloor \text{foc}(\alpha).t \rfloor &= \alpha.[t], \\
\lfloor \text{rel}.t \rfloor &= [t], \text{ and} \\
\lfloor \alpha.t \rfloor &= \alpha.[t] \text{ for any other } \alpha.
\end{align*}
\]

Lemma: soundness for reachability

Let \( A, A', \) and \( t \) be such that \( A \xrightarrow{t} A' \). We have that \( \lfloor A \rfloor \xrightarrow{[t]} \lfloor A' \rfloor \).

Easy.

Lemma: completeness for reachability

Let \( A, A' \), and \( t \) be such that \( A \xrightarrow{t} A' \) is complete. There exists a trace \( t_c \) such that \( \lfloor t_c \rfloor \) is a permutation of \( t \) and \( \lceil A \rceil \xrightarrow{t_c} \lfloor A' \rfloor \).

Sequential dependencies

We need to formalize sequential dependencies.

- add syntactical info. on processes and produced actions
- \textit{labels}: list of integers;
- denote the position of the current action in “the tree of parallel compositions”

Example

Labelled configuration:

\[
A = ([\text{in}(c, x). (\text{in}(c, y). \text{out}(c, x). y). 0 | \text{in}(d, y). \text{out}(d, y_c). 0] \uparrow) ; \emptyset
\]

Labelled trace:

\[
t = [\text{in}(c, x)] \uparrow
\]
Sequential dependencies

We need to formalize **sequential dependencies**.

- add syntactical info. on process and produced actions
- **labels**: list of integers;
- denote the position of the current action in “the tree of parallel compositions”

**Example**

Labelled configuration:

\[
A = (\{[\text{in}(c, x)]^1.([\text{in}(c, y).\text{out}(c, x_y).0 | \text{in}(d, y).\text{out}(d, y_c).0]^1}\}; \emptyset)
\]

Labelled trace:

\[
t = [\text{in}(c, x)]^1[\text{par}]^1
\]
Sequential dependencies

We need to formalize sequential dependencies.

- add syntactical info. on process and produced actions
- *labels*: list of integers;
- denote the position of the current action in “the tree of parallel compositions”

Example

Labelled configuration:

\[ A = (\text{in}(c, y)\cdot \text{out}(c, x_y).0)^{1.1}; (\text{in}(d, y)\cdot \text{out}(d, y_c).0)^{1.2}; \emptyset) \]

Labelled trace:

\[ t = \text{in}(c, x)^1 \par^1 \text{in}(c, y)^1 \text{out}(c, w_0)^1 \text{zero}^1 \text{in}(d, y)^1 \text{out}(d, w_1)^1 \text{zero}^1 \]
Swapping actions

**Definition**

$[\alpha]^\ell$ and $[\beta]^\ell'$ are **sequentially dependent** if $\ell$ is a prefix of $\ell'$ (or the converse).

**Definition**

$[\alpha]^\ell$ and $[\beta]^\ell'$ are **recipe dependent** if $\{\alpha; \beta\} = \{\text{in}(c, M); \text{out}(d, w)\}$ with $w \in \text{fv}(M)$.

We note $[\alpha]^\ell \parallel [\beta]^\ell'$ when they are recipe and sequentially independent.

**Swapping Lemma**

Consider a labelled configuration $A$ and two actions $[\alpha]^\ell \parallel [\beta]^\ell'$. We have that

$$A \xrightarrow{[\alpha]^\ell[\beta]^\ell'} A' \iff A \xrightarrow{[\beta]^\ell'[\alpha]^\ell} A'$$
Proof sketch of completeness

Let $A$, $A'$, and $t$ be such that $A \xrightarrow{t} A'$ is complete. There exists a trace $t_c$ such that $\lfloor t_c \rfloor$ is a permutation of $t$ and $\lceil A \rceil \xrightarrow{t_c} \lceil A' \rceil$.

Using the swapping Lemma we translate iteratively

$A = (\mathcal{P}; \Phi_0) \xrightarrow{\text{tr}} (\emptyset; \Phi)$ into

$$\lceil A \rceil \xrightarrow{\text{tr pos}.\text{rel}.\text{tr pos}.\text{rel}.\text{tr pos} \ldots} c (\emptyset; \emptyset; \Phi)$$

$\leadsto$ Induction on the length of the derivation.
Proof sketch of completeness

Let $A$, $A'$, and $t$ be such that $A \xrightarrow{t} A'$ is complete. There exists a trace $t_c$ such that $\lfloor t_c \rfloor$ is a permutation of $t$ and $\lceil A \rceil \xrightarrow{t_c} c \lceil A' \rceil$.

Using the swapping Lemma we translate iteratively $A = (\mathcal{P}; \Phi_0) \xrightarrow{\text{tr}} (\emptyset; \Phi)$ into

$$[A] \xrightarrow{\ldots \text{tr}_{\text{pos}} \cdot \text{rel} \cdot \text{tr}_{\text{pos}} \cdot \text{rel} \cdot \text{tr}_{\text{pos}} \cdot \text{rel} \cdot \text{tr}_{\text{pos}} \ldots} c (\emptyset; \emptyset; \Phi)$$

$\leadsto$ Induction on the length of the derivation.

- If there is a negative $P \in \mathcal{P}$. This $P$ performs $\alpha_P$ at some point.

$$A \xrightarrow{\text{tr}_1 \cdot \alpha_P \cdot \text{tr}_2} (\emptyset; \Phi) \leadsto A \xrightarrow{\alpha_P} A_1 \xrightarrow{\text{tr}_1 \cdot \text{tr}_2} (\emptyset; \Phi)$$
Proof sketch of completeness

Let $A$, $A'$, and $t$ be such that $A \xrightarrow{t} A'$ is complete. There exists a trace $t_c$ such that $\lfloor t_c \rfloor$ is a permutation of $t$ and $\lceil A \rceil \xrightarrow{t_c} \lceil A' \rceil$.

Using the swapping Lemma we translate iteratively $A = (\mathcal{P}; \Phi_0) \xrightarrow{\text{tr}} (\emptyset; \Phi)$ into

$$\lceil A \rceil \xrightarrow{\text{tr}_{\text{pos}} \text{rel} \text{tr}_{\text{pos}} \text{rel} \text{tr}_{\text{pos}} \text{rel} \text{tr}_{\text{pos}} \cdots} \xrightarrow{c} (\emptyset; \emptyset; \Phi)$$

$\rightsquigarrow$ Induction on the length of the derivation.

- If there is a negative $P \in \mathcal{P}$. This $P$ performs $\alpha_P$ at some point.

$$A \xrightarrow{\text{tr}_1 \cdot \alpha_P \cdot \text{tr}_2} (\emptyset; \Phi) \rightsquigarrow A \xrightarrow{\alpha_P} A_1 \xrightarrow{\text{tr}_1 \cdot \text{tr}_2} (\emptyset; \Phi)$$

$$\rightsquigarrow A \xrightarrow{\alpha_P} A_1, \ \lceil A_1 \rceil \xrightarrow{\text{tr}'} (\emptyset; \Phi)$$

$$\rightsquigarrow \lceil A \rceil \xrightarrow{\alpha_P} \lceil A \rceil \xrightarrow{\text{tr}'} (\emptyset; \Phi)$$
Proof sketch of completeness

Let \( A, A', \) and \( t \) be such that \( A \xrightarrow{t} A' \) is complete. There exists a trace \( t_c \) such that \( \lfloor t_c \rfloor \) is a permutation of \( t \) and \( \lceil A \rceil \xrightarrow{t_c} \lceil A' \rceil \).

Using the swapping Lemma we translate iteratively
\[
A = (P; \Phi_0) \xrightarrow{\text{tr}} (\emptyset; \Phi)
\]
into
\[
\lceil A \rceil \xrightarrow{\text{tr}_{\text{pos}} \cdot \text{rel} \cdot \text{tr}_{\text{pos}} \cdot \text{rel} \cdot \text{tr}_{\text{pos}} \cdot \text{rel} \cdot \text{tr}_{\text{pos}} \cdots} c (\emptyset; \emptyset; \Phi)
\]

\( \rightsquigarrow \) Induction on the length of the derivation.

- If there is a negative \( P \in P \). Done.
- Otherwise, \( A \) is initial. Only positive actions leading to a negative process \( P^- \).

\[
A \xrightarrow{\text{tr}_{\text{in}}} (P' \cup \{P^-\}; \Phi) \xrightarrow{\text{tr}_0} (\emptyset; \Phi)
\]

\( \rightsquigarrow \)
\[
A = (\{P\} \cup P_0; \Phi) \xrightarrow{\text{tr}_P} (\{P^-\} \cup P_0; \Phi) \xrightarrow{\text{tr}_{\text{in}}'} (P' \cup \{P^-\}; \Phi) \xrightarrow{\text{tr}_0} (\emptyset; \Phi)
\]

Lucca Hirschi  
CHoCoLa: Partial order reduction for the applied \( \pi \)-calculus
Proof sketch of completeness

Let \( A, A', \) and \( t \) be such that \( A \xrightarrow{t} A' \) is complete. There exists a trace \( t_c \) such that \([t_c]\) is a permutation of \( t \) and \([A] \xrightarrow{t_c} [A']\).

Using the swapping Lemma we translate iteratively

\[
A = (\mathcal{P}; \Phi_0) \xrightarrow{\text{tr}} (\emptyset; \Phi) \text{ into }
\]

\[
[A] \xrightarrow{\cdots \text{tr}_{\text{pos}} \cdot \text{rel} \cdot \text{tr}_{\text{pos}} \cdot \text{rel} \cdot \text{tr}_{\text{pos}} \cdot \text{rel} \cdot \text{tr}_{\text{pos}} \cdots} (\emptyset; \emptyset; \Phi)
\]

\( \rightsquigarrow \) Induction on the length of the derivation.

- If there is a negative \( P \in \mathcal{P} \). Done.
- Otherwise, \( A \) is initial. Only positive actions leading to a negative process \( P^- \).

\[
A \xrightarrow{\text{tr}_{\text{in}}} (\mathcal{P}' \cup \{P^-\}; \Phi) \xrightarrow{\text{tr}_0} (\emptyset; \Phi)
\]

\( \rightsquigarrow \)

\[
A = (\{P\} \cup \mathcal{P}_0; \Phi) \xrightarrow{\text{tr}_P} (\{P^-\} \cup \mathcal{P}_0; \Phi) \xrightarrow{\text{tr}_{\text{in}}'} (\mathcal{P}' \cup \{P^-\}; \Phi) \xrightarrow{\text{tr}_0} (\emptyset; \Phi)
\]

\( \rightsquigarrow \)

\[
(\{P\} \cup \mathcal{P}_0; \Phi) \xrightarrow{\text{tr}_P} (\{P^-\} \cup \mathcal{P}_0; \Phi), \ (\{P^-\} \cup \mathcal{P}_0; \emptyset; \Phi) \xrightarrow{\text{tr}_{\text{in}}'} (\emptyset; \Phi)
\]
Proof sketch of completeness

Let $A$, $A'$, and $t$ be such that $A \xrightarrow{t} A'$ is complete. There exists a trace $t_c$ such that $\lfloor t_c \rfloor$ is a permutation of $t$ and $\llbracket A \rrbracket \xrightarrow{t_c} \llbracket A' \rrbracket$.

Using the swapping Lemma we translate iteratively

$A = (P; \Phi_0) \xrightarrow{\text{tr}} (\emptyset; \Phi)$ into

$$
\llbracket A \rrbracket \xrightarrow{\text{...tr}_\text{pos} \cdot \text{rel} \cdot \text{tr}_\text{pos} \cdot \text{rel} \cdot \text{tr}_\text{pos} \cdot \text{rel} \cdot \text{tr}_\text{pos} \cdot \text{...}} c \ (\emptyset; \emptyset; \Phi)
$$

$\leadsto$ Induction on the length of the derivation.

- If there is a negative $P \in P$. Done.
- Otherwise, $A$ is initial. Only positive actions leading to a negative process $P^−$.

$$
A \xrightarrow{\text{\text{tr}_\text{in}}} (P' \cup \{P^−\}; \Phi) \xrightarrow{\text{tr}_0} (\emptyset; \Phi)
$$

$$
\leadsto A = (\{P\} \cup P_0; \Phi) \xrightarrow{\text{tr}_P} (\{P^−\} \cup P_0; \Phi) \xrightarrow{\text{tr}_\text{in}'} (P' \cup \{P^−\}; \Phi) \xrightarrow{\text{tr}_0} (\emptyset; \Phi)
$$

$$
\leadsto (\{P\} \cup P_0; \Phi) \xrightarrow{\text{tr}_P} (\{P^−\} \cup P_0; \Phi), (\{P^−\} \cup P_0; \emptyset; \Phi) \xrightarrow{\text{tr}_c'} c (\emptyset; \Phi)
$$

$$
\llbracket A \rrbracket \xrightarrow{\text{foc(tr}_P)\cdot \text{rel}} c (\{P^−\} \cup P_0; \emptyset; \Phi) \xrightarrow{\text{tr}_c'} c (\emptyset; \Phi)
$$
Results - Equivalence

Compressed trace equivalence

\[ A \approx_c B \text{ if for any labelled trace } t \text{ and execution } A \xrightarrow{t} (\mathcal{P}; \emptyset; \Phi) \text{ there is } B \xrightarrow{t} (\mathcal{P}'; \emptyset; \Phi') \text{ such that } \Phi \sim \Phi' \text{ (and the converse).} \]
Compressed trace equivalence

\[ A \approx_c B \] if for any labelled trace \( t \) and execution \( A \xrightarrow{t} (P; \emptyset; \Phi) \) there is \( B \xrightarrow{t} (P'; \emptyset; \Phi') \) such that \( \Phi \sim \Phi' \) (and the converse).

We assume \( A \) and \( B \) to be labelled consistently.

Theorem: Soundness of \( \approx_c \)

Let \( A \) and \( B \) be two initial action-deterministic configurations. If \( A \approx B \) then \( \lceil A \rceil \approx_c \lceil B \rceil \).

Ingredients:
- \( A \approx B \) concides with trace equivalence for labelled trace (including non-observable actions);
- \( + \) and \( - \) phases of \( A \) and \( B \) are sync.
Theorem: Completeness of $\approx_c$

Let $A$ and $B$ be two initial action-deterministic configurations. If $\lceil A \rceil \approx_c \lceil B \rceil$ then $A \approx B$.

- “complete” witnesses of non-equivalence are sufficient;
- undo permutations of $(\xrightarrow{\text{tr}}) \rightsquigarrow (\xrightarrow{\text{tr}_c})$ in $\lceil B \rceil$’s answer
Intuitions

By building upon $\rightarrow_c, \approx_c$:

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

$$b = \text{foc}(a).t^\text{in}.\text{rel}.t^-$$

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

- compressed semantics produces *blocks* of actions of the form:
  
  \[ b = \text{foc}(a).t^\text{in}.\text{rel}.t^- \]

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

\[ P = \text{in}(c_1, x_1).\text{out}(c_1, k_1).P_1 | \text{in}(c_2, x_2).\text{out}(c_2, k_2).P_2 \]
Intuitions

- compressed semantics produces blocks of actions of the form:
  \[ b = \text{foc}(a).t^\text{in}.\text{rel}.t^- \]

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

\[ P = \text{in}(c_1, x_1).\text{out}(c_1, k_1).P_1 \mid \text{in}(c_2, x_2).\text{out}(c_2, k_2).P_2 \]

\[ \text{in}(c_1, X_1) \]
\[ \text{out}(c_1, w_1) \]
\[ \text{in}(c_2, X_2) \]
\[ \text{out}(c_2, w_2) \]
\[ \text{in}(c_2, X_2) \]
\[ \text{out}(c_2, w_2) \]
\[ \text{in}(c_1, X_1) \]
\[ \text{out}(c_1, w_1) \]

\[ X_1 \text{ must depend on } w_2. \]
Intuitions: More redundancies

\[ P = IO(a) | IO(b) | IO(c) \quad \text{where} \quad IO(l) = \text{in}(c_i, X_i).\text{out}(c_i, w_i) \]
Intuitions: More redundancies

\[ P = \text{IO}(a) | \text{IO}(b) | \text{IO}(c) \] where \[ \text{IO}(l) = \text{in}(c_l, X_l) . \text{out}(c_l, \omega_l) \]
Intuitions: More redundancies

\[ P = IO(a) | IO(b) | IO(c) \] where \[ IO(l) = in(c_l, X_l) . out(c_l, w_l) \]
Intuitions: More redundancies

\[ P = IO(a) | IO(b) | IO(c) \] where \( IO(l) = in(c_l, X_l).out(c_l, w_l) \)
Monoid of traces

Definition

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

- $\text{tr}.b_1.b_2.\text{tr}' \equiv_\Phi \text{tr}.b_2.b_1.\text{tr}'$ when $b_1 \parallel b_2$, and
- $\text{tr}.b_1.\text{tr}' \equiv_\Phi \text{tr}.b_2.\text{tr}'$ when $(b_1 =_E b_2)\Phi$. 
Monoid of traces

Definition

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

- $\text{tr}.b_1.b_2.\text{tr}' \equiv_\Phi \text{tr}.b_2.b_1.\text{tr}'$ when $b_1 \parallel b_2$, and
- $\text{tr}.b_1.\text{tr}' \equiv_\Phi \text{tr}.b_2.\text{tr}'$ when $(b_1 =_E b_2)\Phi$.

Lemma

Let $A$ and $A'$ be two initial configurations such that $A \xrightarrow{\text{tr}} A'$. Then $A \xrightarrow{\text{tr}'} A'$ for any $\text{tr}' \equiv_{\Phi(A')} \text{tr}$.

Goal: explore on trace per equivalence class.
Reduced semantics

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

Semantics

\[
\begin{align*}
A & \xrightarrow{\epsilon_r} A \\
A & \xrightarrow{\text{tr}_r} (P; \emptyset; \Phi) (P; \emptyset; \Phi) \xrightarrow{b_c} A' \\
A & \xrightarrow{\text{tr}_r b_r} A'
\end{align*}
\]

if $\text{tr} \not\asymp b'$ for all $b'$

with $(b' =_E b) \Phi$

transparent=0

Availability

A block $b$ is available after $\text{tr}$, denoted $\text{tr} \not\asymp b$, if:

- either $\text{tr} = \epsilon$
- or $\text{tr} = \text{tr}_0 . b_0$ with $\neg (b_0 \parallel b)$
- or $\text{tr} = \text{tr}_0 . b_0$ with $b_0 \parallel b$, $b_0 \prec b$ and $\text{tr}_0 \not\asymp b$. 
Results - Reachability

**Done:** explore on trace per equivalence class.

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

**Lemma:** completeness for reachability

If $A$ and $A' = (P'; \Phi')$ are initial and $[A] \xrightarrow{t} [A']$ then $t$ is $\Phi(A')$-minimal if, and only if, $A \xrightarrow{t_r} A'$.
Results - Reachability

Done: explore on trace per equivalence class.

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

Lemma: completeness for reachability

If \( A \) and \( A' = (P'; \Phi') \) are initial and \( [A] \xrightarrow{t} c [A'] \) then \( t \) is \( \Phi(A') \)-minimal if, and only if, \( A \xrightarrow{t} r A' \).

- reduced semantics explores one trace per equivalence class
- with “swapping lemma” \( \rightsquigarrow \) completeness of reachability for \( \rightarrow_r \)
Results - Equivalence

Definition: Reduced trace equivalence

\[ A \approx_r B \text{ if for any } A \xrightarrow{t} A' \text{ there is } B \xrightarrow{t} B' \text{ such that } \Phi_{A'} \sim \Phi_{B'} \text{ (and the conv.).} \]

Theorem

Let \( A \) and \( B \) be two initial action-deterministic configurations.

\[ A \approx B \text{ if, and only if, } A \approx_r B. \]
**Definition: Reduced trace equivalence**

\[ A \approx_r B \text{ if for any } A \xrightarrow{t} A' \text{ there is } B \xrightarrow{t} B' \text{ such that } \Phi_{A'} \sim \Phi_{B'} \text{ (and the conv.)}. \]

**Theorem**

Let \( A \) and \( B \) be two initial action-deterministic configurations.

\[ A \approx B \text{ if, and only if, } A \approx_r B. \]

- Reachability lemmas +:

**Lemma: Static equivalent frames induce same \( \equiv_\Phi \)**

For any static equivalent frames \( \Phi \sim \Phi' \) and traces \( t_1, t_2 \), we have that \( t_1 \equiv_\Phi t_2 \) if and only if \( t_1 \equiv_{\Phi'} t_2 \).
Outline

1. Introduction
2. Model
3. Big Picture
4. Compression
5. Reduction
6. Conclusion
Implementations

Adapting well established techniques based on:

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

Difficulties:

- decide them exactly is too costly ...
- \( \Rightarrow \) over-approximation;
- in a symmetrical way (otherwise false-attacks).

Results in APTE & SPEC:

<table>
<thead>
<tr>
<th>Tool</th>
<th>Protocol Size</th>
<th>Ref (s)</th>
<th>Comp (s)</th>
<th>Red (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>APTE</td>
<td>PA 1 Sess.</td>
<td>2</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.164</td>
<td>0.012</td>
<td>0.004</td>
</tr>
<tr>
<td></td>
<td>PA 2 Sess.</td>
<td>4</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt; 237h</td>
<td>16.72</td>
<td>11.856</td>
</tr>
<tr>
<td></td>
<td>PA 3 Sess.</td>
<td>6</td>
<td>21</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt; 237h</td>
<td>379696</td>
<td>91266</td>
</tr>
<tr>
<td>SPEC</td>
<td>2 par</td>
<td>2</td>
<td>22</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt; 20 hours</td>
<td>13853</td>
<td>122.27</td>
</tr>
<tr>
<td></td>
<td>7 par</td>
<td>7</td>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt; 20 hours</td>
<td>13853</td>
<td>370.65</td>
</tr>
</tbody>
</table>

Lucca Hirschi

CHoCoLa: Partial order reduction for the applied \( \pi \)-calculus
Implementations

Adapting well established techniques based on:

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

\[ \text{tr} \bowtie b \text{ (availability) as a new type of constraints} \]

Difficulties:

- decide them exactly is too costly ...
Implementations

Adapting well established techniques based on:

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

$\text{tr} \ltimes b$ (availability) as a new type of constraints

Difficulties:

- decide them exactly is too costly ...
- $\leadsto$ over-approximation;
- in a symmetrical way (otherwise false-attacks).
Implementations
Adapting well established techniques based on:
- symbolic semantics (abstract inputs);
- constraint solving procedures.

tr ▷ b (availability) as a new type of constraints

Difficulties:
- decide them exactly is too costly ...
- ⇝ over-approximation;
- in a symmetrical way (otherwise false-attacks).

Results in APTE & SPEC:

<table>
<thead>
<tr>
<th>Tool</th>
<th>Protocol</th>
<th>Size</th>
<th>Ref (s)</th>
<th>Comp (s)</th>
<th>Red (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>APTE</td>
<td>PA 1 Sess.</td>
<td>2/9/5</td>
<td>0.164</td>
<td>0.012</td>
<td>0.004</td>
</tr>
<tr>
<td></td>
<td>PA 2 Sess.</td>
<td>4/15/5</td>
<td>&gt; 237h</td>
<td>16.72</td>
<td>11.856</td>
</tr>
<tr>
<td></td>
<td>PA 3 Sess.</td>
<td>6/21/5</td>
<td>&gt; 237h</td>
<td>379696</td>
<td>91266</td>
</tr>
<tr>
<td>SPEC</td>
<td>2 par</td>
<td>2/22/10</td>
<td>&gt; 20 hours</td>
<td>13853</td>
<td>122.27</td>
</tr>
<tr>
<td></td>
<td>7 par</td>
<td>7/14/2</td>
<td>&gt; 20 hours</td>
<td>13853</td>
<td>370.65</td>
</tr>
</tbody>
</table>
Conclusion

- New **optimizations**: compression and reduction;
- applied to **trace equivalence** checking;
- early implementation in SPEC and Apte.
Conclusion

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

Future Work

1. drop action-deterministic assumption
2. reducing search space:
   - study others redundancies \(\leadsto\) recognize symmetries ?
   - study constraint solving in more details
3. introduce interactivity into the verification process (sub-lemmas, annotations)
Conclusion

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

Future Work

1. drop action-deterministic assumption
2. reducing search space:
   - study others redundancies ↔ recognize symmetries ?
   - study constraint solving in more details
3. introduce interactivity into the verification process (sub-lemmas, annotations)

Any question?