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

GdT PPS

Lucca Hirschi

LSV, ENS Cachan & ENS Lyon

June 19, 2014

---

joint work with

David Baelde and Stéphanie Delaune

LSV and LSV

---

LSV

ENS DE LYON
we need formal **verification** of crypto protocols
we need formal verification of crypto protocols

Our setting

- Applied-$\pi$ models protocols;
- Trace equivalence models security properties.
we need formal verification of crypto protocols

Our setting

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

several algorithms resolve this problem (Akiss, Apte, Spec)

Issue: Limited practical impact

Too slow. Main bottleneck: size of search space (interleavings).

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

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

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

### Terms

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

### Processes and configurations

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

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

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

Lucca Hirschi
Example - Wide Mouth Frog

Informal presentation

<table>
<thead>
<tr>
<th>Sender</th>
<th>Recipient</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Server</td>
<td>enc(k', k_A)</td>
</tr>
<tr>
<td>Server</td>
<td>Bob</td>
<td>enc(k', k_B)</td>
</tr>
<tr>
<td>Alice</td>
<td>Bob</td>
<td>enc(m, k')</td>
</tr>
</tbody>
</table>
Example - Wide Mouth Frog

Informal presentation

<table>
<thead>
<tr>
<th></th>
<th>Alice → Server</th>
<th>Server → Bob</th>
<th>Alice → Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>enc(k′, k_A)</td>
<td>enc(k′, k_B)</td>
<td>enc(m, k′)</td>
</tr>
</tbody>
</table>

Process

\[
\begin{align*}
\text{out}(a, \text{enc}(k′, k_A)) \cdot \text{out}(a, \text{enc}(m, k′)) \\
| \text{in}(s, x). \text{if } x = \text{enc}(y, k_A) \text{ then } \text{out}(s, \text{enc}(y, k_B)) \\
| \text{in}(b, x). \text{if } x = \text{enc}(y, k_B) \text{ then } \\
| \quad \text{in}(b, z). \text{if } z = \text{enc}(w, y) \text{ then } Pb′ \\
\phi = \emptyset
\end{align*}
\]

\[t = \epsilon\]

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

Informal presentation

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

Process

\[
\text{out}(a, \text{enc}(k', ka)) . \text{out}(a, \text{enc}(m, k'))
\]

\[
| \text{in}(s, x). \text{if } x = \text{enc}(y, ka) \text{ then } \text{out}(s, \text{enc}(y, kb)) \\
| \text{in}(b, x). \text{if } x = \text{enc}(y, kb) \text{ then } \\
\quad \text{in}(b, z). \text{if } z = \text{enc}(w, y) \text{ then } Pb'
\]

\[
\Phi = \{ w_0 \triangleright \text{enc}(k', ka) \}
\]

\[
t = \text{out}(a, w_0)
\]
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', k_A)) . \text{out}(a, \text{enc}(m, k'))
| \text{in}(s, x) . \text{if } x = \text{enc}(y, k_A) \text{ then } \text{out}(s, \text{enc}(y, k_B))
| \text{in}(b, x) . \text{if } x = \text{enc}(y, k_B) \text{ then }
| \quad \text{in}(b, z) . \text{if } z = \text{enc}(w, y) \text{ then } P_{b'}

\Phi = \{w_0 \triangleright \text{enc}(k', k_A)\}

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

\(w_0\) is one possible recipe using \(\Phi\)
no other for then branch
Example - Wide Mouth Frog

Informal presentation

\begin{align*}
\text{Alice} & \rightarrow \text{Server} : \text{enc}(k', k_A) \\
\text{Server} & \rightarrow \text{Bob} : \text{enc}(k', k_B) \\
\text{Alice} & \rightarrow \text{Bob} : \text{enc}(m, k')
\end{align*}

Process

\begin{align*}
\text{out}(a, \text{enc}(k', k_A)) & . \text{out}(a, \text{enc}(m, k')) \\
| \text{in}(s, x) . \text{if } x = \text{enc}(y, k_A) \text{ then } & \text{out}(s, \text{enc}(k', k_B)) \\
| \text{in}(b, x) . \text{if } x = \text{enc}(y, k_B) \text{ then } & \text{in}(b, z) . \text{if } z = \text{enc}(w, y) \text{ then } \text{Pb'} \\
\end{align*}

\[ \Phi = \{ w_0 \triangleright \text{enc}(k', k_A); w_1 \triangleright \text{enc}(k', k_B) \} \]

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

Informal presentation

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

Process

\[
\begin{align*}
out(a, \text{enc}(k', ka)) & . out(a, \text{enc}(m, k')) \\
| in(s, x). if \ x = \text{enc}(y, ka) & \ then \ out(s, \text{enc}(k', kb)) \\
| in(b, x). if \ x = \text{enc}(y, kb) & \ then \\
\quad in(b, z). if \ z = \text{enc}(w, y) & \ then \ Pb'
\end{align*}
\]

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

\(t = out(a, w_0). in(s, w_0). out(s, w_1). in(b, w_1)\)
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

$$\text{out}(a, \text{enc}(k', ka)). \text{out}(a, \text{enc}(m, k'))$$

$$| \text{in}(s, x). \text{if } x = \text{enc}(y, ka) \text{ then } \text{out}(s, \text{enc}(k', kb))$$

$$| \text{in}(b, x). \text{if } x = \text{enc}(y, kb) \text{ then }$$

$$\text{in}(b, z). \text{if } z = \text{enc}(w, k') \text{ then } Pb'$$

$$\Phi = \{ w_0 \triangleright \text{enc}(k', ka); w_1 \triangleright \text{enc}(k', kb); w_2 \triangleright \text{enc}(m, k') \}$$

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

Informal presentation

<table>
<thead>
<tr>
<th>Sender</th>
<th>Message</th>
<th>Encryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>$\rightarrow$ Server</td>
<td>$\text{enc}(k', k_A)$</td>
</tr>
<tr>
<td>Server</td>
<td>$\rightarrow$ Bob</td>
<td>$\text{enc}(k', k_B)$</td>
</tr>
<tr>
<td>Alice</td>
<td>$\rightarrow$ Bob</td>
<td>$\text{enc}(m, k')$</td>
</tr>
</tbody>
</table>

Process

$\text{out}(a, \text{enc}(k', ka)).\text{out}(a, \text{enc}(m, k'))$

\[
| \text{in}(s, x). \text{if } x = \text{enc}(y, ka) \text{ then } \text{out}(s, \text{enc}(k', kb)) |
| \text{in}(b, x). \text{if } x = \text{enc}(y, kb) \text{ then } |
| \text{in}(b, z). \text{if } z = \text{enc}(w, k') \text{ then } P_b' |
\]

$\Phi = \{w_0 \triangleright \text{enc}(k', ka); w_1 \triangleright \text{enc}(k', kb); w_2 \triangleright \text{enc}(m, k')\}$

\[
t = \text{out}(a, w_0).\text{in}(s, w_0).\text{out}(s, w_1).\text{in}(b, w_1).\text{out}(a, w_2).\text{in}(b, w_2)
\]
Applied-\(\pi\) - Semantics

**Internal reduction \(\leadsto\):**

- \((\text{if } u = v \text{ then } P \text{ else } Q) \leadsto P \text{ when } u \equiv_E v;\)
- \((\text{if } u = v \text{ then } P \text{ else } Q) \leadsto Q \text{ when } u \not\equiv_E v;\)
- \((P | Q) \leadsto (P' | Q) \text{ and } (Q | P) \leadsto (Q | P') \text{ when } P \leadsto P';\)
- \(((P_1 | P_2) | P_3) \leadsto (P_1 | (P_2 | P_3));\)
- \((P | 0) \rightarrow P \text{ and } (0 | P) \leadsto P.\)
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 $u \neq_E v$;
- $(P \mid Q) \rightsquigarrow (P' \mid Q)$ 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 $(0 \mid P) \rightsquigarrow P$.

\[
\begin{align*}
\text{IN} & \quad (\{\text{in}(c, x).Q\} \cup P; \Phi) \xrightarrow{\text{in}(c, M)} (\{Q[u/x]\} \cup P; \Phi) \\
& \qquad \text{where } M \in \mathcal{T}(\text{dom}(\Phi)) \text{ and } M\Phi =_E u \\
\text{OUT} & \quad (\{\text{out}(c, u).Q\} \cup P; \Phi) \xrightarrow{\text{out}(c, w)} (\{Q\} \cup P; \Phi \cup \{w \mapsto u\}) \\
& \qquad \text{where } w \in \mathcal{W} \text{ is fresh} \\
\text{PAR} & \quad (\{\Pi_{i=1}^n P_i\} \cup P; \Phi) \xrightarrow{\text{par}} (\{P_1; \ldots; P_n\} \cup P; \Phi) \\
\text{ZERO} & \quad (\{0\} \cup P; \Phi) \xrightarrow{\text{zero}} (P; \Phi)
\end{align*}
\]
Trace Equivalence

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

\[ A \approx B \iff \forall A t^\rightarrow A', \exists B t'^\rightarrow B' \text{ such that } \text{obs}(t) = \text{obs}(t') \text{ and } \Phi A' \sim \Phi B' \]

Example: unlinkability
\[
\begin{align*}
\text{Alice} \rightarrow S \rightarrow Bob & \approx (\text{Alice} \rightarrow S \rightarrow Bob \cup \text{Alice} \rightarrow S \rightarrow \text{Charlie}) \\
\end{align*}
\]
Trace Equivalence

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

Trace equivalence

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

and conversely. indistinguishable sets of msgs.
Trace Equivalence

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

Trace equivalence

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

  indistinguishable sets of msgs.

- $\Phi \sim \Phi' \iff (\forall M, N, M\Phi = N\Phi \iff M\Phi' = N\Phi')$
Trace Equivalence

Properties:

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

Trace equivalence

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

(indistinguishable sets of msgs. and conversely)

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

Example: unlikability

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

- **Motivation**: Improve algorithms checking trace equivalence
- **How**: Dramatically decrease the number of interleavings to consider via a reduced semantics

David Baelde, Stéphanie Delaune, and Lucca Hirschi.
A reduced semantics for deciding trace equivalence using constraint systems.

\[ \Rightarrow \text{Compression} \]
\[ \Rightarrow \text{Reduction} \]

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

\[ \rightarrow \approx \quad \text{Compression} \quad \rightarrow c \approx c \quad \text{Reduction} \quad \rightarrow r \approx r \]

Theorem 1: \( \approx = \approx_c \)

Theorem 2: \( \approx_c = \approx_r \)

Required properties

\( \rightarrow_r \) is such that:

- reachability properties coincide on \( \rightarrow_r \) and \( \rightarrow \);
- for action-determinate processes, trace-equivalence coincide on \( \rightarrow_r \) and \( \rightarrow \).
Big Picture

\[\vdash \approx \Rightarrow \text{Theorem 1: } \approx_c \approx \]

\[\vdash \approx_r \Rightarrow \text{Theorem 2: } \approx_c \approx_r \]

Required properties

\[\vdash_r \text{ is such that:} \]

- reachability properties coincide on \(\vdash_r\) and \(\vdash\);
- for action-determinate processes, trace-equivalence coincide on \(\vdash_r\) and \(\vdash\).

Action-determinism

\(A\) is action-deterministic if \(\forall A \vdash (P; \Phi), \forall P, Q \in \mathcal{P} \text{ (different occurence)}, P \text{ and } Q \text{ cannot perform a same observable action.}\)

Make 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 the available actions.

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

- **negative:** `out() . P, Π P_i`
  
  Bring new **data** or **choices**, execution does **not depend** on the context
Intuitions

The Idea

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

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

- **negative**: $\text{out}().P$, $\Pi P_i$
  - Bring new **data** or **choices**, execution does **not depend** on the context
- **positive**: $\text{in}().P$
  - Execution **depends** on the context
Intuitions

The Idea

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

Polarities of processes (similar to focusing):

- **negative**: `out().P, ΠP_i`
  - Bring new **data** or **choices**, execution does **not depend** on the context
  - `⇝` 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 the available actions.

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

- **negative**: $\mathsf{out}().P$, $\prod P_i$
  - Bring new **data** or **choices**, execution does **not depend** on the context
  - $\rightsquigarrow$ to be performed as soon as possible in a given order

- **positive**: $\mathsf{in}().P$
  - Execution **depends** on the context
  - $\rightsquigarrow$ 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 the available actions.

Polarities of processes (similar to focusing):

- **negative**: `out().P, ΠP_i`
  - Bring new **data** or **choices**, execution does **not depend** on the context
  - \(\rightsquigarrow\) to be performed as soon as possible in a given order

- **positive**: `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**
Intuitions

The Idea

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

Polarities of processes (similar to focusing):

- **negative**: `out().P, \Pi P_i`
  Bring new **data** or **choices**, execution does **not depend** on 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**
  ⇝ we make a choice that we must maintain while it is **positive**

Case 0: ends a **positive** phase.
Compressed semantics - Definitions

\((\mathcal{P}; \Phi)\) is **initial** if \(\forall P \in \mathcal{P}, P\) is **positive**.
Compressed semantics - Definitions

\[(\mathcal{P}; \Phi) \text{ is initial if } \forall P \in \mathcal{P}, \ P \text{ is positive.}\]

\[
\frac{\mathcal{P} \text{ is initial}}{(\mathcal{P}; \Phi) \xrightarrow{\text{in}(c, M)} (P'; \Phi)}
\]

\[
\frac{\mathcal{P} \text{ is initial}}{(\mathcal{P} \cup \{P\}; \emptyset; \Phi) \xrightarrow{\text{foc(in}(c, M))} c (\mathcal{P}; P'; \Phi)}
\]

\[
\frac{\mathcal{P} \text{ is initial}}{(P; \Phi) \xrightarrow{\text{in}(c, M)} (P'; \Phi)}
\]

\[
\frac{\mathcal{P} \text{ is initial}}{(\mathcal{P}; P; \Phi) \xrightarrow{\text{in}(c, M)} c (\mathcal{P}; P'; \Phi)}
\]
Compressed semantics - Definitions

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

\[
\begin{align*}
\mathcal{P} \text{ is initial} & \quad (P; \Phi) \xrightarrow{\text{in}(c,M)} (P'; \Phi) \\
\text{START/IN} \quad (\mathcal{P} \uplus \{P\}; \emptyset; \Phi) & \xrightarrow{\text{foc(in}(c,M))} (\mathcal{P} \uplus \{P\}; \emptyset; \Phi) \\
\text{POS/IN} \quad (\mathcal{P}; P; \Phi) & \xrightarrow{\text{in}(c,M)} (\mathcal{P}; P'; \Phi) \\
\text{RELEASE} \quad (\mathcal{P}; P; \Phi) & \xrightarrow{\text{rel}} (\mathcal{P} \uplus \{P\}; \emptyset; \Phi) \\
\text{NEG/\alpha} \quad (\mathcal{P} \uplus \{P\}; \emptyset; \Phi) & \xrightarrow{\alpha} (\mathcal{P} \uplus \mathcal{P}'; \emptyset; \Phi') \\
\end{align*}
\]

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

Translations:

\[
\begin{align*}
[(\mathcal{P}; \Phi)] &= (\mathcal{P}; \emptyset; \Phi), \\
[(\mathcal{P}; \emptyset; \Phi)] &= (\mathcal{P}; \Phi), \\
[(\mathcal{P}; P; \Phi)] &= (\mathcal{P} \cup \{P\}; \Phi).
\end{align*}
\]
Results - Reachability

Translations:

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

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

Translations:

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

\[ [\epsilon] = \epsilon, \quad [\text{foc}(\alpha).t] = \alpha.[t], \quad [\text{rel}.t] = [t], \quad \text{et} \]
\[ [\alpha.t] = \alpha.[t] \text{ for any other } \alpha. \]

Lemma: soundness for reachability

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

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 \( [t_c] \) is a permutation of \( t \) and \( [A] \xrightarrow{t_c}_c [A'] \).
Compressed trace equivalence

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

\[ A \approx_c B \text{ if for any } A \xrightarrow{t} (P; \emptyset; \Phi) = A' \text{ there is } B \xrightarrow{t} (P'; \emptyset; \Phi') = B' \text{ such that } \Phi \sim \Phi' \text{ and } \text{enable}(A') = \text{enable}(B') \text{ (and the conv.).} \]

Theorem: Soundness for equivalence

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

Ingredients:

- par and zero actions can be observed without modifying trace equivalence;
- tests on \( \text{enable}() \) sets do not bring any new information;
- if \( A \approx B \) then + and − phases of \( A \) and \( B \) are sync.
Theorem: Completeness for equivalence

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

Ingredients:
- “complete” witnesses of non-equivalence are sufficient;
- if \( \lfloor A \rfloor \approx_c \lfloor B \rfloor \) then the same swaps can be done for \( A \) and \( B \).
Outline

1 Introduction
2 Model
3 Big Picture
4 Compression
5 Reduction
6 Conclusion
By building upon $\rightarrow_c, \approx_c$:

- because of parallel composition we still need to make choices (which + process?)
- some of them are redundant.
Intuitions

By building upon $\rightarrow_c, \approx_c$:

- because of parallel composition we still need to make **choices** (which + process?)
- some of them are **redundant**.

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

By building upon $\rightarrow_c, \approx_c$:

- because of parallel composition we still need to make choices (which + process?)
- some of them are redundant.

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

Dependency constraint: $X_1$ must depend on $w_2$. 
More redundancies

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

\[ P = IO(a)\cdot IO(b)\cdot IO(c) \text{ where } IO(l) = \text{in}(c_i, X_i)\cdot \text{out}(c_i, w_i) \]

\[ w_c \bowtie X_b \]
More redundancies

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

\[ P = IO(a)|IO(b)|IO(c) \text{ where } IO(l) = in(c_l, X_l).out(c_l, w_l) \]
Ingredients of the reduced semantics

How to detect and remove those redundancies?

- Compressed semantics produces \textit{blocks} of actions of the form:

\[ b = \text{foc}(a).t_{\text{in}}.\text{rel}.t_{\text{out}} \]

- We assume an arbitrary order over blocks: priority order

- The trace \( A \xrightarrow{b_1.b_2} c A' \) is \textit{redundant} (and so is not explored) if
  \begin{enumerate}
  \item \( b_2 \) has priority over \( b_1 \);
  \item \( b_1 \) and \( b_2 \) are \textit{sequentially independant}.
  \item \( b_1 \) and \( b_2 \) have no \textit{data dependency}.
  \end{enumerate}
Analyzing sequentiality dependencies

- We label all syntactical action by a list denoting their position in the “tree of parallel composition”;
- We are able to analyze seq. dependencies of two blocks by comparing their labels.
Analyzing sequentiality dependencies

- We label all syntactical action by a list denoting their position in the “tree of parallel composition”;
- We are able to analyze seq. dependencies of two blocks by comparing their labels.

Example

Processus:
\[\text{in}(c, x)^1 \cdot (\text{in}(c, y)^1)^1 \cdot \text{out}(c, x_y)^1^1 \cdot 0^1^1 | \text{in}(d, y)^1^2 \cdot \text{out}(d, y_c)^1^2 \cdot 0^1^2\]
Analyzing sequentiality dependencies

- We label all syntactical action by a list denoting their position in the “tree of parallel composition”;
- We are able to analyze seq. dependencies of two blocks by comparing their labels.

**Example**

**Processus:**
\[
\text{in}(c, x)^1. (\text{in}(c, y)^{1.1}. \text{out}(c, x_y)^{1.1}.0^{1.1} | \text{in}(d, y)^{1.2}. \text{out}(d, y_c)^{1.2}.0^{1.2})
\]

**Trace (where } b_1 = \text{foc} (\text{in}(c, x))^1. \text{rel.par}:**

\[
\begin{align*}
t &= b_1. \text{foc}(\text{in}(c, y))^{1.1}. \text{rel}. \text{out}(c, w_0). \text{zero} \quad (b_2) \\
&= \underbrace{\text{foc}(\text{in}(d, y))^{1.2}. \text{rel}. \text{out}(d, w_1). \text{zero} \quad (b_3)}_{b_2} \\
\end{align*}
\]

We have that } b_2 \parallel^s b_3 \text{ but not } b_1 \parallel^s b_i.
Analyzing sequentiality dependencies

- We label all syntactical action by a list denoting their position in the “tree of parallel composition”;
- We are able to analyze seq. dependencies of two blocks by comparing their labels.

**Example**

Processus:
\[
\text{in}(c, x)^1. (\text{in}(c, y)^1.1. \text{out}(c, x_y)^1.1.0^1.1 \mid \text{in}(d, y)^1.2. \text{out}(d, y_c)^1.2.0^1.2) \\
\]

Trace (where \( b_1 = \text{foc}(\text{in}(c, x))^1.\text{rel.par} \)):
\[
t = b_1. \underbrace{\text{foc}(\text{in}(c, y))^1.1.\text{rel.out}(c, w_0).\text{zero}}_{b_2}. \\
= \underbrace{\text{foc}(\text{in}(d, y))^1.2.\text{rel.out}(d, w_1).\text{zero}}_{b_3}
\]

We have that \( b_2 \parallel^S b_3 \) but not \( b_1 \parallel^S b_i \).

The general criterion: \( b_1 \parallel^S b_2 \) if \( l_1 \) is not sub-list of \( l_2 \) (or the conv.).
Data dependency

\[ b_1 \vdash^d b_2 \text{ if } \text{fv}(\text{in}(b_1)) \cap \text{out}(b_2) = \emptyset \text{ and the conv.} \]
Data dependency

\[ b_1 \parallel^d b_2 \text{ if } \text{fv}(\text{in}(b_1)) \cap \text{out}(b_2) = \emptyset \text{ and the conv.} \]

Reduced semantics

\[ A \xrightarrow{\epsilon_r} A \text{ and} \]
\[ A \xrightarrow{t_0} (P'; \Phi') (P'; \emptyset; \Phi') \xrightarrow{b_0} (P''; \emptyset; \Phi'') (t_0, \Phi') \xrightarrow{b_0} \]
\[ \text{BLOCK} \]

Availability

A block \( b \) is available after \((t, \Phi')\), denoted \((t, \Phi') \xrightarrow{b_r} \), if:

- either \( t = \epsilon \)
- or \( t = t_0 . b_0 \) with \( \neg b_0 \parallel b \)
- or \( t = t_0 . b_0 \) with \( b_0 \parallel b \), \( b \) has priority over \( b_0 \) and \((t_0, \Phi') \xrightarrow{b} \).

Lucca Hirschi
Data dependency

\[ b_1 \parallel^d b_2 \text{ if } \text{fv}(\text{in}(b_1)) \cap \text{out}(b_2) = \emptyset \text{ and the conv.} \]

Reduced semantics

\[ A \xrightarrow{\epsilon} r A \text{ and } \]

\[ A \xrightarrow{t_0} r (\mathcal{P}'; \Phi') \]
\[ (\mathcal{P}'; \emptyset; \Phi') \xrightarrow{b_0} c (\mathcal{P}'''; \emptyset; \Phi''') \]
\[ (t_0, \Phi') \xrightarrow{b_0} r \]

BLOCK

\[ A \xrightarrow{t_0.b_0} r (\mathcal{P}'''; \Phi''') \]

Availability

A block \( b \) is available after \((t, \Phi')\), denoted \((t, \Phi') \xrightarrow{b} r\), if for any block \( b' \) such that \((b' =_E b)\Phi' :\)

- either \( t = \epsilon \)
- or \( t = t_0.b_0 \) with \( \neg b_0 \parallel b' \)
- or \( t = t_0.b_0 \) with \( b_0 \parallel b' \), \( b' \) has priority over \( b_0 \) and \((t_0, \Phi') \xrightarrow{b'} r\).
Results - Reachability

The relation $\equiv_\Phi$ is the smallest equivalence relation over plausible traces such that:

1. $t.b_1.b_2.t' \equiv_\Phi t.b_2.b_1.t'$ when $b_1 \parallel b_2$;
2. $t.b_1.t' \equiv_\Phi t.b_2.t'$ when $(b_1 =_E b_2)\Phi$.

Lemma: swaps of blocks

If $A = (P; \emptyset; \Phi) \xrightarrow{t} c (P'; \emptyset; \Phi') = A'$ where $A, A'$ are initial then for all $t' \equiv_\Phi t$, we have that $A \xrightarrow{t'\rightarrow_c} A'$.

$t$ is $\Phi$-minimal if there is no $t' \prec t$ st. $t' \equiv_\Phi t'$

Lemma: completeness for reachability

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

Definition: Reduced trace equivalence

\[ A \approx_r B \text{ if for any } A \xrightarrow{t} (\mathcal{P}; \emptyset; \Phi) = A' \text{ there is } B \xrightarrow{t'} (\mathcal{P}'; \emptyset; \Phi') = B' \text{ such that } t \equiv t', \Phi \sim \Phi' \text{ and } \text{enable}(A') = \text{enable}(B') \text{ (and the conv.).} \]

Definition

\[ t \equiv t' \text{ if } t^* = (t')^* \text{ (without label) a,d } \forall i, j, \ (b_i \parallel^s b_j \iff b_i' \parallel^s b_j'). \]

Theorem

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

\[ A \approx B \text{ if, and only if, } A \approx_r B. \]
Lemma: \( \approx_c \) implies symmetric replies

Let \( A \) and \( B \) be two initial action-deterministic configurations such that \( A \approx_c B \). If \( A \xrightarrow{t}_c A' \) for an initial configuration \( A' \) then there exists \( t' \equiv t \) such that \( B \xrightarrow{t'}_c B' \) for some initial configuration \( B' \).

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 \( t_1 \equiv_{\Phi'} t_2 \).

Lemma

Let \( t_1, t'_1, t_2, t'_2 \) three compressed traces and \( \Phi \) a frame such that \( t_1 \equiv t'_1 \) and \( t_1 \equiv_\Phi t_2 \). Then, there exists a trace \( t'_2 \equiv t_2 \) such that \( t'_1 \equiv_\Phi t'_2 \).
Outline

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

<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.

<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>

Future Work

1. reducing search space:
   - study constraint solving in more details
   - study others redundancies \(\Leftrightarrow\) recognize symmetries?
Conclusion

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

<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>

Future Work

1. reducing search space:
   - study constraint solving in more details
   - study others redundancies ⇔ recognize symmetries?

2. 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.

<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>APTE</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>APTE</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>SPEC</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>

Future Work

1. reducing search space:
   - study constraint solving in more details
   - study others redundancies \(\leadsto\) recognize symmetries?

2. introduce interactivity into the verification process (sub-lemmas, annotations)
Outline

7 Benchmarks

8 More compression using focusing
# Benchmarks

<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><strong>APTE</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PA 1 Sess.</td>
<td>2/9/5</td>
<td>0.164</td>
<td>0.012</td>
<td>0.004</td>
<td></td>
</tr>
<tr>
<td>PA 2 Sess.</td>
<td>4/15/5</td>
<td>&gt; 237h</td>
<td>16.72</td>
<td>11.856</td>
<td></td>
</tr>
<tr>
<td>PA 3 Sess.</td>
<td>6/21/5</td>
<td>&gt; 237h</td>
<td>379696</td>
<td>91266</td>
<td></td>
</tr>
<tr>
<td>BAC 1 S./1</td>
<td>4/52/6</td>
<td>13.98</td>
<td>0.02</td>
<td>0.008</td>
<td></td>
</tr>
<tr>
<td>Simple 3 par</td>
<td>3/6/2</td>
<td>0.060</td>
<td>0.004</td>
<td>0.0040</td>
<td></td>
</tr>
<tr>
<td>Simple 5 par</td>
<td>5/10/2</td>
<td>178.8</td>
<td>0.124</td>
<td>0.024</td>
<td></td>
</tr>
<tr>
<td>Simple 7 par</td>
<td>7/14/2</td>
<td>&gt; 163h</td>
<td>8.512</td>
<td>0.196</td>
<td></td>
</tr>
<tr>
<td>Simple 10 par</td>
<td>7/14/2</td>
<td>&gt; 163h</td>
<td>664</td>
<td>1.05</td>
<td></td>
</tr>
<tr>
<td>Complex 4 par</td>
<td>4/10/4</td>
<td>99.87</td>
<td>0.55</td>
<td>0.136</td>
<td></td>
</tr>
<tr>
<td>Complex 7 par</td>
<td>7/16/4</td>
<td>&gt; 163h</td>
<td>198077</td>
<td>363.08</td>
<td></td>
</tr>
<tr>
<td><strong>SPEC</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 par</td>
<td>2/22/10</td>
<td>&gt; 20 hours</td>
<td>13853.04</td>
<td>122.27</td>
<td></td>
</tr>
<tr>
<td>7 par</td>
<td>7/14/2</td>
<td>&gt; 20 hours</td>
<td>13853</td>
<td>370.65</td>
<td></td>
</tr>
<tr>
<td>WMF 1S</td>
<td>3/16/3</td>
<td>65.20</td>
<td>8.01</td>
<td>8.09</td>
<td></td>
</tr>
<tr>
<td>WMF 2S ⊥</td>
<td>6/24/3</td>
<td>7742.24</td>
<td>3.21</td>
<td>3.30</td>
<td></td>
</tr>
</tbody>
</table>
Outline

7 Benchmarks

8 More compression using focusing
Informal Analogy: Focusing - Compression

- **Compression**: complete (wrt. equivalence) reduction of search space
- **Focusing**: complete (wrt. provability) reduction of search space
Informal Analogy: Focusing - Compression

- **Compression**: complete (wrt. equivalence) reduction of search space
- **Focusing**: complete (wrt. provability) reduction of search space

<table>
<thead>
<tr>
<th>Processes</th>
<th>LL formulae</th>
<th>polarity</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in(c, x).P</code></td>
<td><code>∃x.A</code></td>
<td>synchronous</td>
</tr>
<tr>
<td><code>out(c, t).P</code></td>
<td><code>∀x.A</code></td>
<td>asynchronous</td>
</tr>
<tr>
<td>`P_1</td>
<td>P_2`</td>
<td><code>P_1 ≅ P_2</code></td>
</tr>
<tr>
<td><code>!a.→c P</code></td>
<td><code>?P</code></td>
<td>asynchronous</td>
</tr>
</tbody>
</table>
Informal Analogy: Focusing - Compression

- **Compression**: complete (wrt. equivalence) reduction of search space
- **Focusing**: complete (wrt. provability) reduction of search space

<table>
<thead>
<tr>
<th>Processes</th>
<th>LL formulae</th>
<th>polarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>in(c, x).P</td>
<td>∃x.A</td>
<td>synchronous</td>
</tr>
<tr>
<td>out(c, t).P</td>
<td>∀x.A</td>
<td>asynchronous</td>
</tr>
<tr>
<td>P_1</td>
<td>P_2</td>
<td>P_1 ⊯ P_2</td>
</tr>
<tr>
<td>!a. c P</td>
<td>?P</td>
<td>asynchronous</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>compressed execution completeness of ≈_c</th>
<th>focused derivation completeness of focused proof system</th>
</tr>
</thead>
</table>
Focused semantics

Focused execution: alternation of two phases

1. Asynchronous phase:
2. Synchronous phase:
Focused semantics

Focused execution: alternation of two phases

1. *Asynchronous phase:* **When:** $\exists$ output process.
   **What:** Only output actions are available.

2. *Synchronous phase:*
Focused semantics

Focused execution: alternation of two phases

1. **Asynchronous phase:** *When:* \( \exists \) output process.  
   *What:* Only output actions are available.

2. **Synchronous phase:** *When:* all processes start with an input or \(!\).  
   *What:* Choose one input process (or replicate one \(!\)): its is now under focus. Force to perform all its inputs until it reveals an asynchronous action.
Results (work in progress)

Even more effective compression handling \texttt{REPLICATION} and nested parallel compositions for determinate processes.

\[
\text{if } A \xrightarrow{t_r} A_1 \text{ and } A \xrightarrow{t_r} A_2 \text{ then } A_1 = A_2.
\]

Proof of completeness following the informal analogy and:

Dale Miller and Alexis Saurin.  
From proofs to focused proofs: a modular proof of focalization in linear logic.  
In \textit{CSL 2007: Computer Science Logic, volume 4646 of LNCS, pages 405–419.}  
Even more **effective** compression handling **REPLICATION** and nested parallel compositions for determinate processes.

\[
\text{if } A \xrightarrow{t} A_1 \text{ and } A \xrightarrow{t} A_2 \text{ then } A_1 = A_2.
\]

Proof of completeness following the informal analogy and:

- **Strictly better**: does the same for simple processes.
- **Very modular**: can be applied to any \(\pi\)-calculus-like.