A reduced semantics for deciding trace equivalence using constraint systems

CEA - Seminar

Lucca Hirschi

LSV, ENS Cachan & ENS Lyon

April 1, 2014

joint work with

David Baelde LSV

and

Stéphanie Delaume LSV

LSV
## Cryptography

We need **secure** cryptography to protect our data, set up trustworthy communication channels, preserve our anonymity, etc.

.quiz {text-align: justify}

$$\leadsto$$ we need formal **verification** of crypto protocols
Cryptography

We need **secure** cryptography to protect our data, set up trustworthy communication channels, preserve our anonymity, etc.

\[\Rightarrow\] we need formal **verification** of crypto protocols

Our setting

Prove automatically security properties of cryptographic protocols using formal methods.

- **Applied-\(\pi\)** models protocols. **Dolev-Yao model**: We make strong assumptions over the cryptographic **primitives** but we model an active attacker.
Cryptography

We need secure cryptography to protect our data, set up trustworthy communication channels, preserve our anonymity, etc.

We need formal verification of crypto protocols

Our setting

Prove automatically security properties of cryptographic protocols using formal methods.

- Applied-$\pi$ models protocols. Dolev-Yao model: We make strong assumptions over the cryptographic primitives but we model an active attacker.

- Trace equivalence models security properties (e.g., strong secrecy, unlinkability, anonymity, ...)

Lucca Hirschi
Introduction

Cryptography

We need secure cryptography to protect our data, set up trustworthy communication channels, preserve our anonymity, etc.

we need formal verification of crypto protocols

Our setting

Prove automatically security properties of cryptographic protocols using formal methods.

- Applied-π models protocols. Dolev-Yao model: We make strong assumptions over the cryptographic primitives but we model an active attacker.

- Trace equivalence models security properties (e.g., strong secrecy, unlinkability, anonymity, ...)

- undecidable in general
Introduction

Cryptography

We need secure cryptography to protect our data, set up trustworthy communication channels, preserve our anonymity, etc.

\[\Rightarrow\] we need formal verification of crypto protocols

Our setting

Prove automatically security properties of cryptographic protocols using formal methods.

- **Applied-\(\pi\)** models protocols. **Dolev-Yao model:** We make strong assumptions over the cryptographic primitives but we model an active attacker.

- **Trace equivalence** models security properties (**e.g.**, strong secrecy, unlinkability, anonymity, ...)

\[\Rightarrow\] decidable if we consider a bounded nb. of sessions

\[\Rightarrow\] several algorithms resolve this problem (Akiss, Apte, Spec)
Several algorithms (Akiss, Apte, Spec) compute trace equivalence of protocols (bounded nb. of sessions).

**Issue: Limited practical impact**

Too slow. Main bottleneck: size of search space (interleavings).
several algorithms (Akiss, Apte, Spec) compute trace equivalence of protocols (bounded nb. of sessions).

Issue: Limited practical impact

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

Our Contribution

Reduce search space of equivalence checking using POR ideas by eliminating a lot of redundancies (for simple processes).

David Baelde, Stéphanie Delaune, and Lucca Hirschi.

A reduced semantics for deciding trace equivalence using constraint systems.


To appear.
**Terms**

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

**Simple Processes**

- $P_c ::= 0 \mid \text{in}(c, x) \mid \text{out}(c, m).P_c \mid \text{if } T \text{ then } P_c \text{ else } P_c$
- $P_s ::= P_{c_1} \mid P_{c_2} \mid \ldots P_{c_n}$ \hspace{1em} $c_i \neq c_j$
Terms

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

Simple Processes

- \( P_c ::= 0 | \text{in}(c, x) | \text{out}(c, m).P_c | \text{if \ T \ then \ P_c \ else \ P_c} \)
- \( P_s ::= P_{c_1} | P_{c_2} | \ldots | P_{c_n} \quad c_i \neq c_j \)
- Process: \( (P_s; \Phi) \) (\( \Phi \) set of messages revealed to the intruder).
Applied-$\pi$

Terms

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

Simple Processes

- $P_c ::= 0 \mid \text{in}(c, x) \mid \text{out}(c, m).P_c \mid \text{if } T \text{ then } P_c \text{ else } P_c$
- $P_s ::= P_{c_1} \mid P_{c_2} \mid \ldots \mid P_{c_n} \quad c_i \neq c_j$
- Process: $(P_s; \Phi)$ ($\Phi$ set of messages revealed to the intruder).

Semantics

$\left(\{\text{out}(c, m).P\} \cup P; \Phi\right) \xrightarrow{\nu w.\text{out}(c, w)} \left(\{P\} \cup P; \Phi \cup \{w \triangleright m\}\right)$
if $T \land w$ fresh in $\Phi$

$\left(\{\text{in}(c, x).P\} \cup P; \Phi\right) \xrightarrow{\text{in}(c, t)} \left(\{P[x \leftarrow u]\} \cup P; \Phi\right)$
if $t\Phi = u \land \text{fv}(t) \subseteq \text{dom}(\Phi)$
Example

Wide Mouth Frog

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

Wide Mouth Frog

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

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

$\Phi = \emptyset$
Example

Wide Mouth Frog

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

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

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

Wide Mouth Frog

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

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

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

Wide Mouth Frog

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

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

Wide Mouth Frog

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

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

\[P = (\mathcal{P}; \Phi);\]

\[\Phi: \text{attacker's knowledge}.\]
Example

Wide Mouth Frog

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

\[
\text{out}(a, \text{enc}(k', k_A)) \cdot \text{out}(a, \text{enc}(m, k'))
\]
\[
| \text{in}(s, x). \text{if } x = \text{enc}(y, k_A) \text{ then } \text{out}(s, \text{enc}(y, k_B))
\]
\[
| \text{in}(b, x). \text{if } x = \text{enc}(y, k_B) \text{ then }
\]
\[
\text{b}(z). \text{if } z = \text{enc}(w, y) \text{ then } 
\]
\[
\ldots
\]

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

$\Phi$: attacker’s knowledge.

Properties:

1. Reachability (secret, authentication) and
2. Equivalence (anonymity, unlikability).
Trace Equivalence

Trace equivalence

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

Trace equivalence

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

\[ A \approx B \iff \forall A \xrightarrow{s} A', \exists B', B \xrightarrow{s} B' \land \Phi_{A'} \sim \Phi_{B'} \text{ and conversely.} \]

Trace equivalence allows to model anonymity, unlinkability, etc.
Goal

▷ **Motivation:** Improve algorithms checking trace equivalence for simple processes

▷ **How:** Dramatically decrease the number of interleavings to consider via a reduced semantics
Big Picture

Grouping actions:

- generalization of the idea "force to perform output as soon as possible"
- $\rightarrow_c$ only explores specific traces
- Theorem 1: $\approx = \approx_c$
Big Picture

Symbolic semantics:
- classic step adapted for $\rightarrow_c$
Big Picture

\[ \rightarrow \quad \text{Compressed semantics} \quad \rightarrow \quad \text{Symbolic semantics} \quad \rightarrow \quad \text{Reduction} \]

\[ \approx = \approx_c \quad \approx_c \quad \approx_s \quad \approx_{s} = \approx_{d}^{2} \]

Analyze dependencies:

- force one order for independent (parallel) actions
- analyze dependencies "on the fly"
- \( \rightarrow_{d} \) explores even less traces
- Theorem 2: \( \approx_s = \approx_{d}^{2} \)
Outline

1. Introduction
2. Compressed semantics
3. Reduced semantics
4. Conclusion
Compression

- Reachability: \textbf{force} output actions to be performed first
Compression

- Reachability: force output actions to be performed first
- Equivalence: not that simple
  - order of actions matters (observable)
  - we consider two processes (symmetry)
Compression

- Reachability: force output actions to be performed first
- Equivalence: not that simple
  - order of actions matters (observable)
  - we consider two processes (symmetry)

Grouping actions into *blocks*

\[
in(c, \_ \ldots \in(c, \_). out(c, \_ \ldots \out(c, \_)
\]
via a focused semantics \( \rightarrow_c \).
Compression - Example

Basic rules of $\rightarrow_c$:

- choose a basic process $P_i \in \mathcal{P}$, it is now under focus;
- **focus**: only $P_i$ can perform actions
- $P_i$ can release the focus only if:
  - it has performed a block IO (> 1 input, > 1 output) and
  - it can not perform an output any more.
Compression - Example

Basic rules of $\rightarrow_c$:

- choose a basic process $P_i \in \mathcal{P}$, it is now **under focus**;
- **focus**: only $P_i$ can perform actions
- $P_i$ can release the focus only if:
  - it has performed a block IO ($> 1$ input, $> 1$ output) and
  - it cannot perform an output any more.

**Example**

Consider $P = P_1 \parallel P_2$ with $P_i = \text{in}(c_i, x).\text{in}(c_i, y).\text{out}(c_i, \langle x, y \rangle)$.

- Semantics $\rightarrow_c$ explores **only two** interleavings of 6 actions:

  $\text{in}(c_1, x_1).\text{in}(c_1, y_1).\text{out}(c_1, w_1).\text{in}(c_2, x_2).\text{in}(c_2, y_2).\text{out}(c_2, w_2)$

  and

  $\text{in}(c_2, x_2).\text{in}(c_2, y_2).\text{out}(c_2, w_2).\text{in}(c_1, x_1).\text{in}(c_1, y_1).\text{out}(c_1, w_1)$

- **semantics** $\rightarrow$ explores **20** such interleavings.
Compression - Result

The semantics $\rightarrow_c$ induces a **compressed** trace equivalence $\approx_c$

**Theorem 1**

$$A \approx B \iff A \approx_c B$$

**Key ideas**

- **symmetric**: remove same interleavings on both sides
- **completeness**: in any execution, we can swap two actions on different channels (**simple processes**)
Compression - Result

The semantics $\rightarrow_c$ induces a compressed trace equivalence $\approx_c$

Theorem 1

$$A \approx B \iff A \approx_c B$$

Key ideas

- **symmetric**: remove same interleavings on both sides
- **completeness**: in any execution, we can swap two actions on different channels (simple processes)

Benefits

- **first optimization** that decreases (possibly exponentially many) interleavings to consider
- **easy to implement**
- **allow us to reason with** *macro-actions* i.e., blocks $\leadsto$ reduced semantics
Outline

1. Introduction
2. Compressed semantics
3. Reduced semantics
4. Conclusion
Symbolic calculus - 1

Inputs messages: infinitely branching $\leadsto$ symbolic calculus.
Symbolic calculus - 1

Inputs messages: infinitely branching $\rightsquigarrow$ symbolic calculus.

System of Constraints

- Constraints: $D \vdash x \quad u =? v \quad u \neq?v$
- System of constraints: $(\Phi, S)$. 
Symbolic calculus - 1

Inputs messages: infinitely branching $\leadsto$ symbolic calculus.

System of Constraints

- Constraints: $D \vdash^? x \quad u =^? v \quad u \neq^? v$
- System of constraints: $(\Phi, S)$.

$$P = \text{out}(c, k).\text{in}(c, x).\text{out}(c, \langle k, x \rangle).\text{in}(c, y)$$

leads to

$$S = \{\{w\} \vdash^? x, \{w, w'\} \vdash^? y\}$$

$$\Phi = \{w \triangleright k; w' \triangleright \langle k, x \rangle\}$$
Symbolic calculus - 1

Inputs messages: infinitely branching \(\rightsquigarrow\) symbolic calculus.

**System of Constraints**

- Constraints: \(D \vdash_x^? x \quad u =? v \quad u \neq v\)
- System of constraints: \((\Phi, S)\).

\[
P = \text{out}(c, k).\text{in}(c, x).\text{out}(c, \langle k, x \rangle).\text{in}(c, y)
\]

leads to

\[
S = \{\{w\} \vdash_x^? x, \{w, w'\} \vdash_y y\}
\]

\[
\Phi = \{w \triangleright k; w' \triangleright \langle k, x \rangle\}
\]

**Symbolic process**

\((\mathcal{P}; \Phi; S)\)
Semantics

\[
\begin{align*}
\{\text{out}(c, m).P\} \cup P; \Phi; S) & \xrightarrow{\nu w.\text{out}(c, X)} \{P\} \cup P; \Phi \cup \{w \rhd m\}; S) \\
\{\text{in}(c, x).P\} \cup P; \Phi; S) & \xrightarrow{\text{in}(c, X)} (P; \Phi; S \cup \{\text{dom}(\phi) \vdash \exists x\}) \\
\text{if } w \text{ fresh in } \phi \\
\text{if } X \text{ fresh in } S
\end{align*}
\]
Symbolic Calculus - 2

Semantics

\[
\begin{align*}
\{\text{out}(c, m).P\} \cup P; \Phi; S & \xrightarrow{\nu w.\text{out}(c, X)} \{P\} \cup P; \Phi \cup \{w \triangleright m\}; S \\
& \quad \text{if } w \text{ fresh in } \phi \\
\{\text{in}(c, x).P\} \cup P; \Phi; S & \xrightarrow{\text{in}(c, X)} (P; \Phi; S \cup \{\text{dom}(\phi) \vdash X x\}) \\
& \quad \text{if } X \text{ fresh in } S
\end{align*}
\]

Symbolic equivalence

\[A \approx_s B \iff \forall A \xrightarrow{s} A' \forall \Theta \in \text{Sol}(\Phi_{A'}, D_{A'}), \exists B' B \xrightarrow{s} B', \Theta \in \text{Sol}(\Phi_{B'}, D_{B'}) \text{ and } \Phi_{A'} \sim \Phi_{B'} \text{ and conversely.}\]
Symbolic Calculus - 2

Semantics

\[
\begin{align*}
\nu w.\text{out}(c,X) & : (\{\text{out}(c,m).P\} \cup P; \Phi; S) \xrightarrow{\nu w.\text{out}(c,X)} (\{P\} \cup P; \Phi \cup \{w \triangleright m\}; S) \\
& \text{if } w \text{ fresh in } \phi \\
\text{in}(c,X) & : (\{\text{in}(c,x).P\} \cup P; \Phi; S) \xrightarrow{\text{in}(c,X)} (P; \Phi; S \cup \{\text{dom}(\phi) \vdash \not\exists x\}) \\
& \text{if } X \text{ fresh in } S
\end{align*}
\]

Symbolic equivalence

\[
A \approx_s B \iff \forall A \xrightarrow{S} A' \forall \Theta \in \text{Sol}(\Phi_{A'}, D_{A'}), \exists B' B \xrightarrow{S} B', \Theta \in \text{Sol}(\Phi_{B'}, D_{B'}) \text{ and } \Phi_{A'} \sim \Phi_{B'} \text{ and conversely.}
\]

- There already exist several procedures checking equivalence between constraint systems
- **Goal:** starting with $\xrightarrow{c}$ (compressed symbolic semantics), reduces the number of interleavings to explore
\[ P = \text{in}(c_1, x_1).\text{out}(c_1, k_1).P_1 \mid \text{in}(c_2, x_2).\text{out}(c_2, k_2).P_2 \]

Dependency constraint: \( X_1 \) must depend on \( w_2 \).

We can add constraints on the fly thanks to an order \(<\).

▶ symmetry: Eliminate same traces on both sides
▶ Do not remove too much information (intruder can observe the order).

---

Sebastian Mödersheim, Luca Vigano, and David Basin.
Constraint differentiation: Search-space reduction for the constraint-based analysis of security protocols.

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

Dependency constraint: \( X_1 \) must depend on \( w_2 \).
\[ P = \text{in}(c_1, x_1) . \text{out}(c_1, k_1).P_1 \mid \text{in}(c_2, x_2) . \text{out}(c_2, k_2).P_2 \]

Dependency constraint: \( X_1 \) must depend on \( w_2 \).

We can add constraints on the fly thanks to an order \( < \).
\[ P = \text{in}(c_1, x_1) \cdot \text{out}(c_1, k_1) \cdot P_1 \mid \text{in}(c_2, x_2) \cdot \text{out}(c_2, k_2) \cdot P_2 \]

Dependency constraint: \( X_1 \) must depend on \( w_2 \).

We can add constraints on the fly thanks to an order \(<\).

- **symmetry**: Eliminate same traces on both sides
\[ P = \text{in}(c_1, x_1).\text{out}(c_1, k_1).P_1 \mid \text{in}(c_2, x_2).\text{out}(c_2, k_2).P_2 \]

**Dependency constraint:** \( X_1 \) must depend on \( w_2 \).

We can add constraints on the fly thanks to an order \(<\).  

- symmetry: Eliminate same traces on both sides
- Do not remove too much information (intruder can observe the order).
\[ P = \text{IO}(a) | \text{IO}(b) | \text{IO}(c) \text{ where } \text{IO}(l) = \text{in}(c_l, X_l).\text{out}(c_l, w_l) \]
\[ P = IO(a) | IO(b) | IO(c) \] where \[ IO(l) = in(c_l, X_l).out(c_l, w_l) \]
\[ P = \text{IO}(a) \| \text{IO}(b) \| \text{IO}(c) \text{ where } \text{IO}(l) = \text{in}(c_l, X_l) \cdot \text{out}(c_l, w_l) \]
\[ P = IO(a)\mid IO(b)\mid IO(c) \text{ where } IO(l) = \text{in}(c_l, X_l).\text{out}(c_l, w_l) \]

A block on \( c \) is executed following \( t \), one input of the block must depend on one output of \( \text{dep}(t, c) = \{w_1 \ldots w_n\} \) if

- \( t = t_1.IO(c_1).IO(c_2) \ldots IO(c_n).IO(c) \)
- \( c < c_1 \)
- \( c_1, c_2, \ldots, c_n < c \)
Reduced semantics

Reduced semantics $\mapsto^d$

Compressed, symbolic semantics $+$ dependency constraints built on the fly.
Reduced semantics

Reduced semantics $\rightarrow_d$

Compressed, symbolic semantics + dependency constraints built on the fly.

\[
(P; \Phi; \emptyset) \xrightarrow{\text{tr}}_d (P'; \Phi'; S') \quad (P'; \Phi'; S') \xrightarrow{\text{io}_c(\overrightarrow{X}, \overrightarrow{w})} c (P''; \Phi''; S'')
\]

\[
(P; \Phi; \emptyset) \xrightarrow{\text{tr} \cdot \text{io}_c(\overrightarrow{X}, \overrightarrow{w})} \rightarrow_d (P''; \Phi''; S'' \cup \{ \overrightarrow{X} \times \text{dep} (\text{tr}, c) \})
\]

Two possible semantics for $\text{Sol}(X \times w)$:

- **second order**: $w$ occurs in the recipe $X\Theta$
- **first order**: for all recipe $R$, if $R\Phi = (X\Theta)\Phi$ then $w$ occurs in $R$
Reduced semantics

Reduced semantics $\rightarrow_d$

Compressed, symbolic semantics + dependency constraints built on the fly.

\[
(P; \Phi; \emptyset) \xrightarrow{\text{tr}}_d (P'; \Phi'; S') \quad (P'; \Phi'; S') \xrightarrow{\text{io}_c(X, W)}_c (P''; \Phi''; S'')
\]

\[
(P; \Phi; \emptyset) \xrightarrow{\text{tr} \cdot \text{io}_c(X, W)}_d (P''; \Phi''; S'' \cup \{X \times \text{dep(tr, c)}\})
\]

Two possible semantics for $\text{Sol}(X \times w)$:

- **second** order: $w$ occurs in the recipe $X \Theta$
- **first** order: for all recipe $R$, if $R\Phi = (X \Theta)\Phi$ then $w$ occurs in $R$

**Theorem 2**

\[
\approx^2_d = \approx_s \\
\approx^1_d = \approx_s
\]
Idea of the proof

- \([t]\): set of traces modulo valid permutations;
- \(\text{Min}([t])\): lexicographic minimum of the class.

Lemma 1
If \(P\) has an trace \(t\) then it has all traces of \([t]\).

Lemma 2
- If \(P\) has an trace \(t\) then it has a reduced trace \(\text{Min}(t)\);
- \(P\) has no other reduced trace in \([t]\).
Outline

1. Introduction
2. Compressed semantics
3. Reduced semantics
4. Conclusion
Conclusion

- New optimization in two steps:
  - compression
  - reduction
- applied to trace equivalence checking
- potentially exponential speed up
- 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></td>
<td>BAC 1 S./1</td>
<td>4/52/6</td>
<td>13.98</td>
<td>0.02</td>
<td>0.008</td>
</tr>
<tr>
<td></td>
<td>Simple 3 par</td>
<td>3/6/2</td>
<td>0.060</td>
<td>0.004</td>
<td>0.0040</td>
</tr>
<tr>
<td></td>
<td>Simple 5 par</td>
<td>5/10/2</td>
<td>178.8</td>
<td>0.124</td>
<td>0.024</td>
</tr>
<tr>
<td></td>
<td>Simple 7 par</td>
<td>7/14/2</td>
<td>&gt; 163h</td>
<td>8.512</td>
<td>0.196</td>
</tr>
<tr>
<td></td>
<td>Simple 10 par</td>
<td>7/14/2</td>
<td>&gt; 163h</td>
<td>664</td>
<td>1.05</td>
</tr>
<tr>
<td></td>
<td>Complex 4 par</td>
<td>4/10/4</td>
<td>99.87</td>
<td>0.55</td>
<td>0.136</td>
</tr>
<tr>
<td></td>
<td>Complex 7 par</td>
<td>7/16/4</td>
<td>&gt; 163h</td>
<td>198077</td>
<td>363.08</td>
</tr>
<tr>
<td>SPEC</td>
<td>2 par</td>
<td>2/22/10</td>
<td>&gt; 20 hours</td>
<td>13853.04</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>
<tr>
<td></td>
<td>WMF 1S</td>
<td>3/16/3</td>
<td>65.20</td>
<td>8.01</td>
<td>8.09</td>
</tr>
<tr>
<td></td>
<td>WMF 2S</td>
<td>6/24/3</td>
<td>7742.24</td>
<td>3.21</td>
<td>3.30</td>
</tr>
</tbody>
</table>
Future Work

- study constraint solving in more details
- study others redundancies \(\rightsquigarrow\) recognize symmetries ?
- using those optimizations for interactive proofs of trace equivalence
- dealing with ! and nested parallels
- investigate such optimizations without the determinacy assumption
Future Work

- study constraint solving in more details
- study others redundancies \(\sim\) recognize symmetries?
- using those optimizations for interactive proofs of trace equivalence
- dealing with \! and nested parallels
- investigate such optimizations without the determinacy assumption

Any question?
Outline

More compression using focusing
Informal Analogy: Focusing - Compression

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

**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>(\text{in}(c,x).P)</td>
<td>(\exists x.A)</td>
<td>synchronous</td>
</tr>
<tr>
<td>(\text{out}(c,t).P)</td>
<td>(\forall x.A)</td>
<td>asynchronous</td>
</tr>
<tr>
<td>(P_1</td>
<td>P_2)</td>
<td>(P_1 \bowtie P_2)</td>
</tr>
<tr>
<td>(\ddagger a.\overrightarrow{c}P)</td>
<td>(?P)</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>( \text{in}(c, x).P )</td>
<td>( \exists x.A )</td>
<td>synchronous</td>
</tr>
<tr>
<td>( \text{out}(c, t).P )</td>
<td>( \forall x.A )</td>
<td>asynchronous</td>
</tr>
<tr>
<td>( P_1</td>
<td>P_2 )</td>
<td>( P_1 \not&lt; P_2 )</td>
</tr>
<tr>
<td>( !a. \overrightarrow{c} P )</td>
<td>( P )</td>
<td>asynchronous</td>
</tr>
</tbody>
</table>

| compressed execution completeness of \( \approx_c \) | focused derivation completeness of focused proof system |
Focused semantics

Focused execution: alternation of two phases

1. Asynchronous phase:
2. Synchronous phase:
Focused execution: alternation of two phases

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

2. **Synchronous phase:**
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.
More compression using focusing

Results (work in progress)

Even more effective compression handling REPLICATION and nested parallel compositions for determinate processes.

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

Proof of completeness following the informal analogy and:

Dale Miller and Alexis Saurin.
From proofs to focused proofs: a modular proof of focalization in linear logic.
More compression using focusing

Results (work in progress)

Even more effective compression handling **REPLICATION** and nested parallel compositions for determinate processes.

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

Proof of completeness following the informal analogy and:

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

---

Dale Miller and Alexis Saurin.

From proofs to focused proofs: a modular proof of focalization in linear logic.