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 Delaune LSV

ENS DE LYON

Introduction

Cryptography

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

$\leadsto$ we need formal verification of crypto protocols

Introduction

Cryptography

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

$\rightsquigarrow$ 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.

Introduction

Cryptography

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

$\rightsquigarrow$ 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, ...)

Introduction

Cryptography

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

$\rightsquigarrow$ 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.

$\rightsquigarrow$ 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, ...)

- decidable if we consider a bounded nb. of sessions

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

$\rightsquigarrow$ 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).

$\rightsquigarrow$ 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.
In Martín Abadi and Steve Kremer, editors, *Proceedings of the 3rd International Conference on Principles of Security and Trust (POST'14)*, Lecture Notes in Computer Science, Grenoble, France, April 2014. Springer.
To appear.

Applied-π

Terms

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

Simple Processes

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

Applied-π

Terms

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

Simple Processes

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

Applied-π

Terms

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

Simple Processes

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

Semantics

$$(\{\mathtt{out}(c,m).P\} \uplus \mathcal{P}; \Phi) \xrightarrow{\nu w.\mathtt{out}(c,w)} (\{P\} \uplus \mathcal{P}; \Phi \cup \{w \triangleright m\})$$
if $T \ \wedge\ w$ fresh in Φ

$$(\{\mathtt{in}(c,x).P\} \uplus \mathcal{P}; \Phi) \xrightarrow{\mathtt{in}(c,t)} (\{P[x \mapsto u]\} \cup \mathcal{P}; \Phi)$$
if $t\Phi = u \ \wedge\ fv(t) \subseteq \mathrm{dom}(\Phi)$

Example

Wide Mouth Frog

$$\begin{array}{rcl} \text{Alice} \rightarrow \text{Serveur} & : & \text{enc}(k', k_A) \\ \text{Serveur} \rightarrow \text{Bob} & : & \text{enc}(k', k_B) \\ \text{Alice} \rightarrow \text{Bob} & : & \text{enc}(m, k') \end{array}$$

Example

Wide Mouth Frog

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

```
  out(a,enc(k',ka)).out(a,enc(m,k'))
| in(s,x). if x = enc(y,ka) then out(s,enc(y,kb))
| in(b,x). if x = enc(y,kb) then
    b(z). if z = enc(w,y) then  ...
```

$$\Phi = \varnothing$$

Example

Wide Mouth Frog

$$\begin{array}{rcl} \text{Alice} \rightarrow \text{Serveur} & : & \text{enc}(k', k_A) \\ \text{Serveur} \rightarrow \text{Bob} & : & \text{enc}(k', k_B) \\ \text{Alice} \rightarrow \text{Bob} & : & \text{enc}(m, k') \end{array}$$

```
  out(a,enc(k',ka)).out(a,enc(m,k'))
| in(s,x). if x = enc(y,ka) then out(s,enc(y,kb))
| in(b,x). if x = enc(y,kb) then
    b(z). if z = enc(w,y) then  ...
```

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

Example

Wide Mouth Frog

$$\begin{array}{rcl} \text{Alice} \rightarrow \text{Serveur} & : & \text{enc}(k', k_A) \\ \text{Serveur} \rightarrow \text{Bob} & : & \text{enc}(k', k_B) \\ \text{Alice} \rightarrow \text{Bob} & : & \text{enc}(m, k') \end{array}$$

```
  out(a,enc(k',ka)).out(a,enc(m,k'))
| in(s,x). if x = enc(y,ka) then out(s,enc(y,kb))
| in(b,x). if x = enc(y,kb) then
    b(z). if z = enc(w,y) then  ...
```

$$\Phi = \{enc(k', ka); enc(k', kb)\}$$

Example

Wide Mouth Frog

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

```
  out(a,enc(k',ka)).out(a,enc(m,k'))
| in(s,x). if x = enc(y,ka) then out(s,enc(y,kb))
| in(b,x). if x = enc(y,kb) then
    b(z). if z = enc(w,y) then  ...
```

$$\Phi = \{enc(k', ka); enc(k', kb)\}$$

Example

Wide Mouth Frog

$$\begin{array}{rcl} \text{Alice} \rightarrow \text{Serveur} & : & \text{enc}(k', k_A) \\ \text{Serveur} \rightarrow \text{Bob} & : & \text{enc}(k', k_B) \\ \text{Alice} \rightarrow \text{Bob} & : & \text{enc}(m, k') \end{array}$$

```
  out(a,enc(k',ka)).out(a,enc(m,k'))
| in(s,x). if x = enc(y,ka) then out(s,enc(y,kb))
| in(b,x). if x = enc(y,kb) then
    b(z). if z = enc(w,y) then ...
```

- $P = (\mathcal{P}; \Phi)$;
- Φ: attacker's knowledge.

Example

Wide Mouth Frog

$$\begin{array}{rcl} \text{Alice} \to \text{Serveur} & : & \text{enc}(k', k_A) \\ \text{Serveur} \to \text{Bob} & : & \text{enc}(k', k_B) \\ \text{Alice} \to \text{Bob} & : & \text{enc}(m, k') \end{array}$$

```
    out(a,enc(k',ka)).out(a,enc(m,k'))
  | in(s,x). if x = enc(y,ka) then out(s,enc(y,kb))
  | in(b,x). if x = enc(y,kb) then
      b(z). if z = enc(w,y) then ...
```

- $P = (\mathcal{P}; \Phi)$;
- Φ: attacker's knowledge.

Properties:

1. Reachability (secret, authentification) 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

- $\Phi \sim \Phi' \iff (\forall M, N,\ M\Phi = N\Phi \iff M\Phi' = N\Phi')$
- $A \approx B \iff \forall A \xrightarrow{s} A',\ \exists B',\ B \xrightarrow{s} B' \wedge \Phi_{A'} \sim \Phi_{B'}$ and conversely.

Trace equivalence allows to model anonymity, unlinkability, etc.

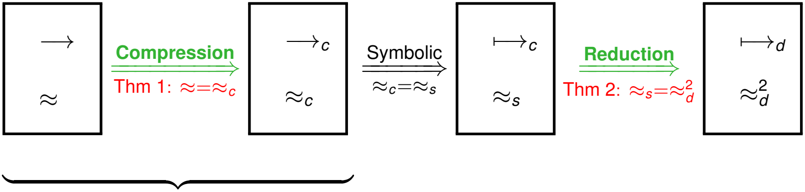
Big Picture

Goal

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

$$\boxed{\begin{matrix}\longrightarrow \\ \approx\end{matrix}} \xRightarrow[\text{Thm 1: } \approx = \approx_c]{\textbf{Compression}} \boxed{\begin{matrix}\longrightarrow_c \\ \approx_c\end{matrix}} \xRightarrow[\approx_c = \approx_s]{\text{Symbolic}} \boxed{\begin{matrix}\longmapsto_c \\ \approx_s\end{matrix}} \xRightarrow[\text{Thm 2: } \approx_s = \approx_d^2]{\textbf{Reduction}} \boxed{\begin{matrix}\longmapsto_d \\ \approx_d^2\end{matrix}}$$

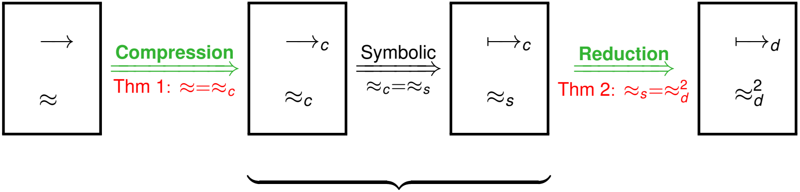
Big Picture



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

$$\boxed{\begin{matrix}\longrightarrow \\ \approx\end{matrix}} \xrightarrow[\text{Thm 1: } \approx=\approx_c]{\textbf{Compression}} \boxed{\begin{matrix}\longrightarrow_c \\ \approx_c\end{matrix}} \xrightarrow[\approx_c=\approx_s]{\text{Symbolic}} \boxed{\begin{matrix}\longmapsto_c \\ \approx_s\end{matrix}} \xrightarrow[\text{Thm 2: } \approx_s=\approx_d^2]{\textbf{Reduction}} \boxed{\begin{matrix}\longmapsto_d \\ \approx_d^2\end{matrix}}$$

Analyze dependencies:

- force one order for independent (parallel) actions
- analyze dependencies "on the fly"
- $\longmapsto_d$ explores even less traces
- Theorem 2: $\approx_s=\approx_d^2$

Outline

Outline

Compression

- Reachability: 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*

$$\mathtt{in}(c,\_)\ldots\mathtt{in}(c,\_).\mathtt{out}(c,\_)\ldots\mathtt{out}(c,\_)$$

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 can not perform an output any more.

Example

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

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

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

  and

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

- 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
  $\rightsquigarrow$ reduced semantics

Outline

Symbolic calculus - 1

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

Symbolic calculus - 1

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

System of Constraints

- Constraints: $D \vdash^?_X x \qquad u =^? v \qquad u \neq^? v$
- System of constraints: $(\Phi, \mathcal{S})$.

Symbolic calculus - 1

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

System of Constraints

- Constraints: $D \vdash^?_X x \qquad u =^? v \qquad u \neq^? v$
- System of constraints: $(\Phi, \mathcal{S})$.

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

leads to

$$\mathcal{S} = \{\{w\} \vdash^?_X x, \{w,w'\} \vdash^?_Y y\}$$
$$\Phi = \{w \rhd k; w' \rhd \langle k,x\rangle\}$$

Symbolic calculus - 1

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

System of Constraints

- Constraints: $D \vdash^?_X x \qquad u =^? v \qquad u \neq^? v$
- System of constraints: $(\Phi, \mathcal{S})$.

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

leads to

$$\mathcal{S} = \{\{w\} \vdash^?_X x, \{w, w'\} \vdash^?_Y y\}$$
$$\Phi = \{w \triangleright k; w' \triangleright \langle k, x\rangle\}$$

Symbolic process

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

Symbolic Calculus - 2

Semantics

$$(\{\mathtt{out}(c,m).P\} \uplus \mathcal{P}; \Phi; \mathcal{S}) \xrightarrow{\nu w.\mathtt{out}(c,X)} (\{P\} \uplus \mathcal{P}; \Phi \cup \{w \triangleright m\}; \mathcal{S})$$

if w fresh in ϕ

$$(\{\mathtt{in}(c,x).P\} \uplus \mathcal{P}; \Phi; \mathcal{S}) \xrightarrow{\mathtt{in}(c,X)} (\mathcal{P}; \Phi; \mathcal{S} \cup \{\mathrm{dom}(\phi) \vdash^?_X x\})$$

if X fresh in $\mathcal{S}$

Symbolic Calculus - 2

Semantics

$$(\{\mathtt{out}(c,m).P\} \uplus \mathcal{P}; \Phi; \mathcal{S}) \xrightarrow{\nu w.\mathtt{out}(c,X)} (\{P\} \uplus \mathcal{P}; \Phi \cup \{w \rhd m\}; \mathcal{S})$$
if w fresh in ϕ

$$(\{\mathtt{in}(c,x).P\} \uplus \mathcal{P}; \Phi; \mathcal{S}) \xrightarrow{\mathtt{in}(c,X)} (\mathcal{P}; \Phi; \mathcal{S} \cup \{\mathrm{dom}(\phi) \vdash^?_X x\})$$
if X fresh in $\mathcal{S}$

Symbolic equivalence

$A \approx_s B \iff \forall A \overset{s}{\mapsto} A' \; \forall \Theta \in \mathcal{S}\mathrm{ol}(\Phi_{A'}, \mathcal{D}_{A'}),\ \exists B' \; B \overset{s}{\mapsto} B', \Theta \in \mathcal{S}\mathrm{ol}(\Phi_{B'}, \mathcal{D}_{B'})$ and $\Phi_{A'} \sim \Phi_{B'}$ and conversely.

Symbolic Calculus - 2

Semantics

$$(\{\mathtt{out}(c,m).P\} \uplus \mathcal{P}; \Phi; \mathcal{S}) \xrightarrow{\nu w.\mathtt{out}(c,X)} (\{P\} \uplus \mathcal{P}; \Phi \cup \{w \rhd m\}; \mathcal{S})$$
if w fresh in ϕ

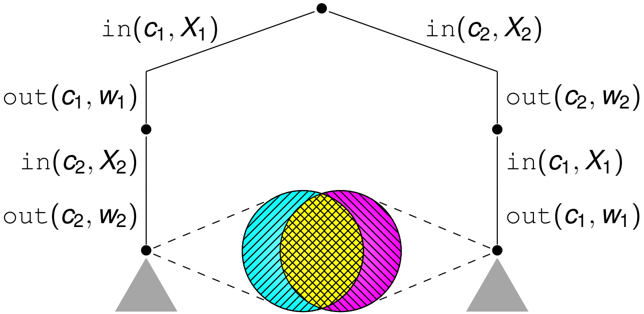
$$(\{\mathtt{in}(c,x).P\} \uplus \mathcal{P}; \Phi; \mathcal{S}) \xrightarrow{\mathtt{in}(c,X)} (\mathcal{P}; \Phi; \mathcal{S} \cup \{\mathrm{dom}(\phi) \vdash^?_X x\})$$
if X fresh in $\mathcal{S}$

Symbolic equivalence

$A \approx_s B \iff \forall A \xrightarrow{s} A' \; \forall \Theta \in \mathcal{S}\mathrm{ol}(\Phi_{A'}, \mathcal{D}_{A'}), \; \exists B' \; B \xrightarrow{s} B', \Theta \in \mathcal{S}\mathrm{ol}(\Phi_{B'}, \mathcal{D}_{B'})$ and $\Phi_{A'} \sim \Phi_{B'}$ and conversely.

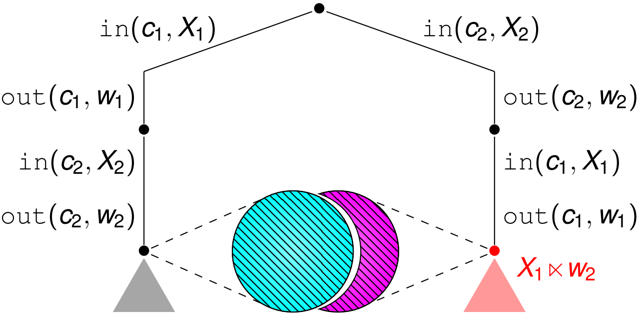
- There already exist several procedures checking equivalence between constraint systems
- Goal: starting with $\mapsto_c$ (compressed symbolic semantics), reduces the number of interleavings to explore

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



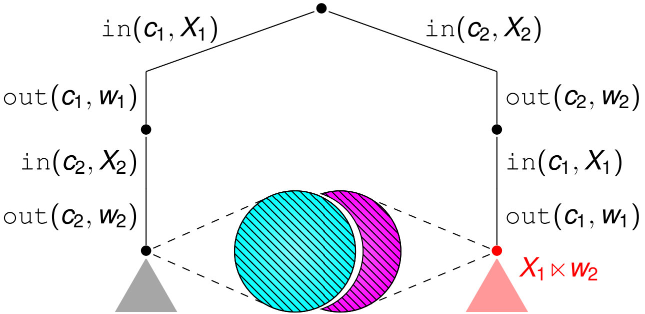
Sebastian Mödersheim, Luca Vigano, and David Basin.
Constraint differentiation: Search-space reduction for the constraint-based analysis of security protocols.
Journal of Computer Security, 18(4):575–618, 2010.

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



Dependency constraint: X_1 must depend on w_2.

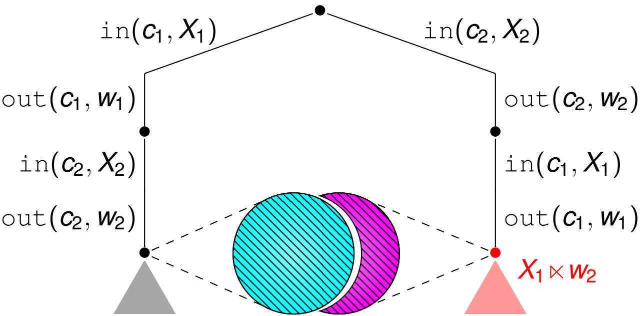
$$P = \mathtt{in}(c_1, x_1).\mathtt{out}(c_1, k_1).P_1 \mid \mathtt{in}(c_2, x_2).\mathtt{out}(c_2, k_2).P_2$$



Dependency constraint: X_1 must depend on w_2.

We can add constraints on the fly thanks to an order $<$.

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

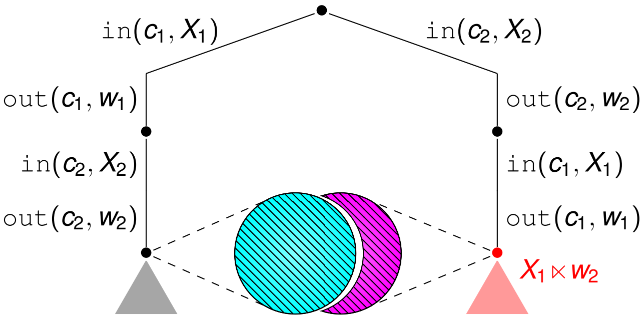


Dependency constraint: X_1 must depend on w_2.

We can add constraints on the fly thanks to an order $<$.

- symmetry: Eliminate same traces on both sides

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

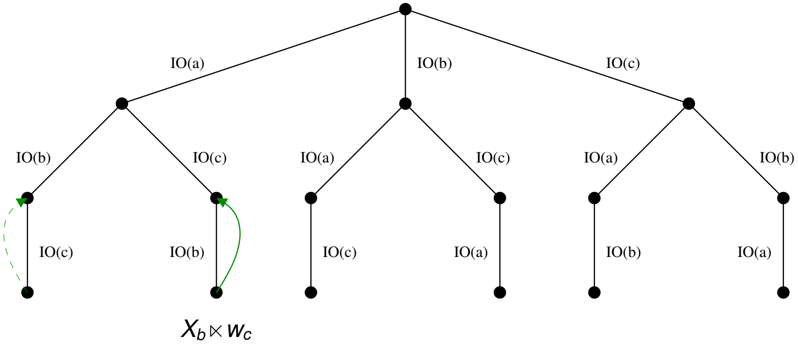


Dependency constraint: X_1 must depend on w_2.

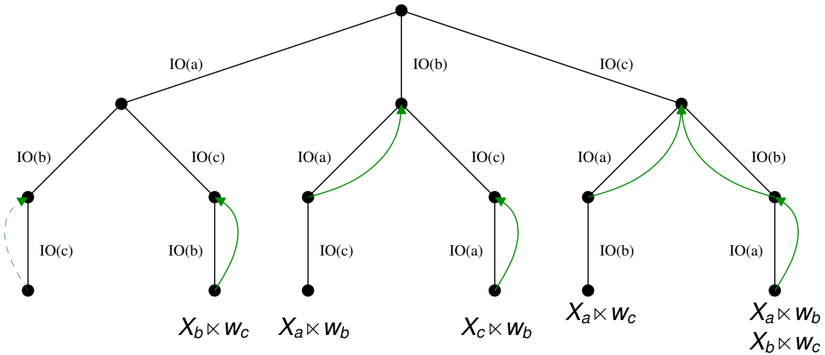
We can add constraints on the fly thanks to an order $<$.

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

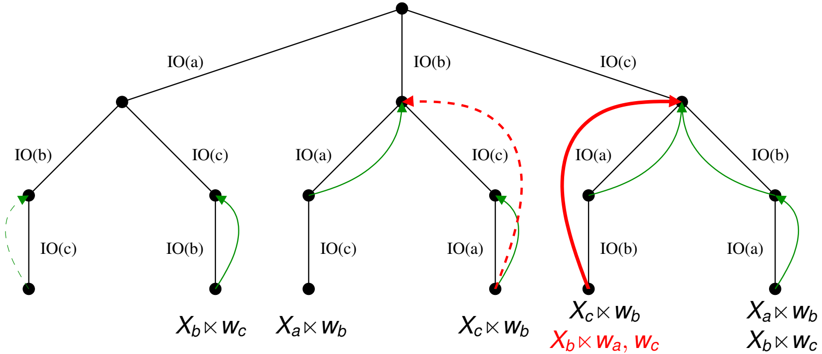
$$P = IO(a)|IO(b)|IO(c) \ \ \text{where} \quad IO(l) = \mathtt{in}(c_l, X_l).\mathtt{out}(c_l, w_l)$$



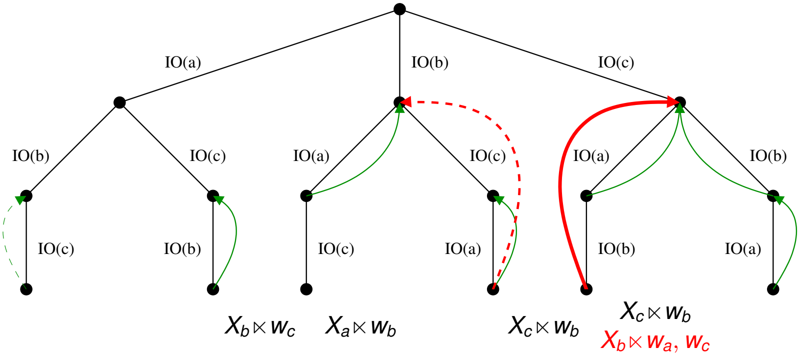
$$P = IO(a)|IO(b)|IO(c) \;\; \text{where} \;\; IO(l) = \mathtt{in}(c_l, X_l).\mathtt{out}(c_l, w_l)$$



$$P = IO(a)|IO(b)|IO(c) \quad \text{where} \quad IO(l) = \mathtt{in}(c_l, X_l).\mathtt{out}(c_l, w_l)$$



$P = IO(a)|IO(b)|IO(c)$ where $IO(l) = \mathtt{in}(c_l, X_l).\mathtt{out}(c_l, w_l)$



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

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

$$\frac{(\mathcal{P};\Phi;\varnothing) \xrightarrow{\mathsf{tr}}_d (\mathcal{P}';\Phi';\mathcal{S}') \quad (\mathcal{P}';\Phi';\mathcal{S}') \xrightarrow{\mathtt{io}_c(\overrightarrow{X},\overrightarrow{w})}_c (\mathcal{P}'';\Phi'';\mathcal{S}'')}{(\mathcal{P};\Phi;\varnothing) \xrightarrow{\mathsf{tr}\cdot\mathtt{io}_c(\overrightarrow{X},\overrightarrow{w})}_d (\mathcal{P}'';\Phi'';\mathcal{S}'' \cup \{\overrightarrow{X} \ltimes \mathrm{dep}\,(\mathsf{tr}, c)\})}$$

Two possible semantics for $\mathsf{Sol}(X \ltimes w)$:

- 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 $\mapsto_d$

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

$$\frac{(\mathcal{P};\Phi;\varnothing) \xmapsto{\mathrm{tr}}_d (\mathcal{P}';\Phi';\mathcal{S}') \quad (\mathcal{P}';\Phi';\mathcal{S}') \xmapsto{\mathtt{io}_c(\vec{X},\vec{w})}_c (\mathcal{P}'';\Phi'';\mathcal{S}'')}{(\mathcal{P};\Phi;\varnothing) \xmapsto{\mathrm{tr}\cdot\mathtt{io}_c(\vec{X},\vec{w})}_d (\mathcal{P}'';\Phi'';\mathcal{S}'' \cup \{\vec{X} \ltimes \mathrm{dep}(\mathrm{tr},c)\})}$$

Two possible semantics for $\mathrm{Sol}(X \ltimes w)$:

- 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

$$\approx_d^2 = \approx_s \qquad \approx_d^1 = \approx_s$$

Idea of the proof

- $[t]$: set of traces modulo valid permutations;
- $\mathrm{Min}([t])$: lexico. 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 $\mathrm{Min}(\mathrm{t})$;
- P has no other reduced trace in $[t]$.

Outline

Conclusion

- New optimization in two steps:
  - compression
  - reduction
- applied to trace equivalence checking
- potentially exponential speed up
- early implementation in SPEC and Apte

Benchmarks

Tool	Protocol	Size	Ref (s)	Comp (s)	Red (s)
APTE	PA 1 Sess.	2/9/5	0.164	0.012	0.004
	PA 2 Sess.	4/15/5	> 237h	16.72	11.856
	PA 3 Sess.	6/21/5	> 237h	379696	91266
	BAC 1 S./1	4/52/6	13.98	0.02	0.008
	Simple 3 par	3/6/2	0.060	0.004	0.0040
	Simple 5 par	5/10/2	178.8	0.124	0.024
	Simple 7 par	7/14/2	> 163h	8.512	0.196
	Simple 10 par	7/14/2	> 163h	664	1.05
	Complex 4 par	4/10/4	99.87	0.55	0.136
	Complex 7 par	7/16/4	> 163h	198077	363.08
SPEC	2 par	2/22/10	$>$ 20 hours	13853.04	122.27
	7 par	7/14/2	$>$ 20 hours	13853	370.65
	WMF 1S	3/16/3	65.20	8.01	8.09
	WMF 2S $\perp$	6/24/3	7742.24	3.21	3.30

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 $\rightsquigarrow$ 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

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

Processes	LL formulae	polarity
$\mathtt{in}(c,x).P$	$\exists x.A$	synchronous
$\mathtt{out}(c,t).P$	$\forall x.A$	asynchronous
$P_1 \mid P_2$	$P_1 \parr P_2$	asynchronous
$!^{a,\vec{c}}\,P$	$?P$	asynchronous

Informal Analogy: Focusing - Compression

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

Processes	LL formulae	polarity
$\mathtt{in}(c, x).P$	$\exists x.A$	synchronous
$\mathtt{out}(c, t).P$	$\forall x.A$	asynchronous
$P_1 \mid P_2$	$P_1 \parr P_2$	asynchronous
$!^{a,\overrightarrow{c}} P$	$?P$	asynchronous

compressed execution	focused derivation
completeness of $\approx_c$	completeness of focused proof system

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 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 *CSL 2007: Computer Science Logic, volume 4646 of LNCS*, pages 405–419. Springer-Verlag, 2007.

Results (work in progress)

Even more effective compression handling 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 *CSL 2007: Computer Science Logic, volume 4646 of LNCS*, pages 405–419. Springer-Verlag, 2007.

- Strictly better: does the same for simple processes.
- Very modular: can be applied to any π-calculus-like.