Partial order reduction for the applied π-calculus

CHoCoLa

Lucca Hirschi

LSV, ENS Cachan

November 13, 2014

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

Introduction

unsecure network, active attacker $\rightarrow$ attacks
$\rightsquigarrow$ we need formal verification of crypto protocols

Introduction

unsecure network, active attacker $\rightarrow$ attacks
$\rightsquigarrow$ we need formal verification of crypto protocols

Our setting

- Applied-π models protocols;
- Trace equivalence models security properties.

Introduction

unsecure network, active attacker $\rightarrow$ attacks
$\rightsquigarrow$ we need formal verification of crypto protocols

Our setting

- Applied-π models protocols;
- Trace equivalence models security properties.

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

Issue: Limited practical impact

Too slow. Bottleneck: size of search space (interleavings).

Outline

Outline

Applied-π - Syntax

Terms

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

Applied-π - Syntax

Terms

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

Processes and configurations

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

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

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

$$\Phi = \{ \underbrace{w_0}_{\text{handle}} \mapsto \underbrace{\mathrm{enc}(m,k)}_{\text{out. message}}; w_1 \mapsto k \}$$

Applied-π - Syntax

Terms

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

Processes and configurations

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

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

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

$$\Phi = \{\underbrace{w_0}_{\text{handle}} \mapsto \underbrace{\mathsf{enc}(m,k)}_{\text{out. message}}; w_1 \mapsto k\}$$

- recipes are terms built using only handles

$$e.g., R = \mathsf{dec}(w_0, w_1) \qquad m =_{\mathsf{E}} R\Phi$$

intuition: how the environment builds messages from its knowledge

Example - Wide Mouth Frog

Informal presentation

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

Example - Wide Mouth Frog

Informal presentation

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

Process

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

$$\Phi = \emptyset$$

$$t = \epsilon$$

Let us explore one possible trace.

Example - Wide Mouth Frog

Informal presentation

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

Process

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

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

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

Example - Wide Mouth Frog

Informal presentation

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

Process

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

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

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

w_0 is one possible recipe using Φ
no other for `then` branch since the attacker does not know k_A

Example - Wide Mouth Frog

Informal presentation

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

Process

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

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

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

Example - Wide Mouth Frog

Informal presentation

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

Process

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

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

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

Example - Wide Mouth Frog

Informal presentation

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

Process

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

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

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

Example - Wide Mouth Frog

Informal presentation

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

Process

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

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

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

Applied-π - Semantics

Internal reduction $\rightsquigarrow$:

- $(\mathtt{if}\ u = v\ \mathtt{then}\ P\ \mathtt{else}\ Q) \rightsquigarrow P$ when $u =_E v$;
- $(\mathtt{if}\ u = v\ \mathtt{then}\ P\ \mathtt{else}\ Q) \rightsquigarrow Q$ 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 $(0 \mid P) \rightsquigarrow P$.

Applied-π - Semantics

Internal reduction $\rightsquigarrow$:

- $(\mathtt{if}\ u = v\ \mathtt{then}\ P\ \mathtt{else}\ Q) \rightsquigarrow P$ when $u =_E v$;
- $(\mathtt{if}\ u = v\ \mathtt{then}\ P\ \mathtt{else}\ Q) \rightsquigarrow Q$ 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. $\Pi_{i=1}^{3} P_i$
- $(P \mid 0) \to P$ and $(0 \mid P) \rightsquigarrow P$.

IN	$(\{\mathtt{in}(c,x).Q\} \uplus \mathcal{P}; \Phi)$	$\xrightarrow{\mathtt{in}(c,M)}$	$(\{Q\{u/x\}\} \uplus \mathcal{P}; \Phi)$ where $M \in \mathcal{T}(\mathrm{dom}(\Phi))$ and $M\Phi =_E u$
OUT	$(\{\mathtt{out}(c,u).Q\} \uplus \mathcal{P}; \Phi)$	$\xrightarrow{\mathtt{out}(c,w)}$	$(\{Q\} \uplus \mathcal{P}; \Phi \cup \{w \mapsto u\})$ where $w \in \mathcal{W}$ is fresh
PAR	$(\{\Pi_{i=1}^{n} P_i\} \uplus \mathcal{P}; \Phi)$	$\xrightarrow{\mathtt{par}}$	$(\{P_1; \ldots; P_n\} \uplus \mathcal{P}; \Phi)$
ZERO	$(\{0\} \uplus \mathcal{P}; \Phi)$	$\xrightarrow{\mathtt{zero}}$	$(\mathcal{P}; \Phi)$

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

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

Trace Equivalence

Properties:

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

(bisimulation: too strong)

Trace equivalence

- $A \approx B \iff \forall A \xrightarrow{t} A',\ \exists B \xrightarrow{t'} B'$ such that $\mathrm{obs}(t) = \mathrm{obs}(t')$ and $\Phi_{A'} \sim \Phi_{B'}$ (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, authentification) and
2. Trace equivalence (*e.g.,* anonymity, unlikability).

(bisimulation: too strong)

Trace equivalence

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

Example: unlinkability of WMF

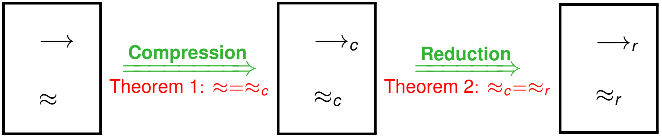
$$(\overbrace{\{P_a; P_s; P_b\}}^{Alice \to S \to Bob} \cup \overbrace{\{P_a; P_s; P_b\}}^{Alice \to S \to Bob}; \epsilon) \overset{?}{\approx} (\overbrace{\{P_a; P_s; P_b\}}^{Alice \to S \to Bob} \cup \overbrace{\{P'_a; P'_s; P_c\}}^{Alice \to S \to Charlie}; \epsilon)$$

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

Outline

Big Picture

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



$\rightarrow_r$ does not explore all behaviours but sufficiently to ensure $\approx = \approx_r$

Big Picture

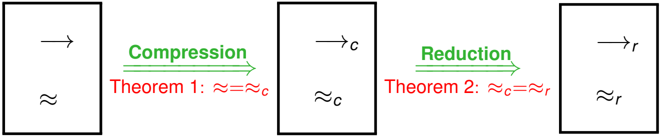
$\boxed{\begin{array}{c}\longrightarrow \\ \approx\end{array}} \xRightarrow[\text{Theorem 1: } \approx = \approx_c]{\text{Compression}} \boxed{\begin{array}{c}\longrightarrow_c \\ \approx_c\end{array}} \xRightarrow[\text{Theorem 2: } \approx_c = \approx_r]{\text{Reduction}} \boxed{\begin{array}{c}\longrightarrow_r \\ \approx_r\end{array}}$

Required properties

$\rightarrow_r$ is such that:

- reachability properties coincide on $\rightarrow_r$ and $\rightarrow$;
- for **action-determinate** processese, 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** processese, trace-equivalence coincides on $\rightarrow_r$ and $\rightarrow$.

Action-determinsm

A is *action-deterministic* if $\forall A \xrightarrow{t} (\mathcal{P}; \Phi),\ \forall P, Q \in \mathcal{P}$, 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

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

Polarities of processes (similar to focusing of LL):

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

  $\rightsquigarrow$ to be performed as soon as possible in a given order
- *positive*: $\mathtt{in}().P$
  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*

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

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

POS/IN
$$\frac{(P; \Phi) \xrightarrow{\mathtt{in}(c,M)} (P'; \Phi)}{(\mathcal{P}; P; \Phi) \xrightarrow{\mathtt{in}(c,M)}_c (\mathcal{P} \uplus \{P'\}; \Phi;)}$$

Compressed semantics - Definitions

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

Semantics:

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

POS/IN
$$\frac{(P;\Phi) \xrightarrow{\mathtt{in}(c,M)} (P';\Phi)}{(\mathcal{P};P;\Phi) \xrightarrow{\mathtt{in}(c,M)}_c (\mathcal{P} \uplus \{P'\};\Phi;)}$$

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

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

Results - Reachability

Translations:

$$\lceil(\mathcal{P};\Phi)\rceil = (\mathcal{P};\varnothing;\Phi), \quad \lfloor(\mathcal{P};\varnothing;\Phi)\rfloor = (\mathcal{P};\Phi), \quad \lfloor(\mathcal{P};P;\Phi)\rfloor = (\mathcal{P}\uplus\{P\};\Phi).$$

Results - Reachability

Translations:

$$\lceil(\mathcal{P};\Phi)\rceil = (\mathcal{P};\varnothing;\Phi), \quad \lfloor(\mathcal{P};\varnothing;\Phi)\rfloor = (\mathcal{P};\Phi), \quad \lfloor(\mathcal{P};P;\Phi)\rfloor = (\mathcal{P}\uplus\{P\};\Phi).$$

$\lfloor\epsilon\rfloor = \epsilon$, $\lfloor\mathtt{foc}(\alpha).t\rfloor = \alpha.\lfloor t\rfloor$, $\lfloor\mathtt{rel}.t\rfloor = \lfloor t\rfloor$, and
$\lfloor\alpha.t\rfloor = \alpha.\lfloor t\rfloor$ for any other α.

Results - Reachability

Translations:

$$\lceil(\mathcal{P};\Phi)\rceil = (\mathcal{P};\varnothing;\Phi), \quad \lfloor(\mathcal{P};\varnothing;\Phi)\rfloor = (\mathcal{P};\Phi), \quad \lfloor(\mathcal{P};P;\Phi)\rfloor = (\mathcal{P}\uplus\{P\};\Phi).$$

$\lfloor\epsilon\rfloor = \epsilon$, $\lfloor \mathtt{foc}(\alpha).t\rfloor = \alpha.\lfloor t\rfloor$, $\lfloor \mathtt{rel}.t\rfloor = \lfloor t\rfloor$, and
$\lfloor \alpha.t\rfloor = \alpha.\lfloor t\rfloor$ for any other α.

Lemma: soundness for reachability

Let A, A', and t be such that $A \xrightarrow{t}_c A'$. We have that $\lfloor A\rfloor \xrightarrow{\lfloor t\rfloor} \lfloor A'\rfloor$.

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 $\lceil A\rceil \xrightarrow{t_c}_c \lceil A'\rceil$.

Proof: non-trivial. Adapating the “positive trunk” argument. Involving swaps of actions.

Sequential dependencies

We need to formalize sequential dependencies.

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

Example

Labelled configuration:

$$A = (\{[\mathtt{in}(c, x).\left(\mathtt{in}(c, y).\mathtt{out}(c, x_y).0 \mid \mathtt{in}(d, y).\mathtt{out}(d, y_c).0]^1\right)\}; \emptyset)$$

Labelled trace :
$t = [\mathtt{in}(c, x)]^1$

Sequential dependencies

We need to formalize sequential dependencies.

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

Example

Labelled configuration:

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

Labelled trace :
$t = [\mathtt{in}(c, x)]^1 [\mathtt{par}]^1$

Sequential dependencies

We need to formalize sequential dependencies.

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

Example

Labelled configuration:

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

Labelled trace :

$$t = [\mathtt{in}(c, x)]^{1}[\mathtt{par}]^{1}[\mathtt{in}(c, y)]^{1.1}[\mathtt{out}(c, w_0)]^{1.1}[\mathtt{zero}]^{1.1}$$
$$[\mathtt{in}(d, y)]^{1.2}[\mathtt{out}(d, w_1)]^{1.2}[\mathtt{zero}]^{1.2}$$

Swapping actions

Definition

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

Definition

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

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 $\lceil A \rceil \xrightarrow{t_c}_c \lceil A' \rceil$.

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

$$\lceil A \rceil \xrightarrow{\ldots\mathrm{tr}_{\mathrm{pos}}.\mathtt{rel}.\mathrm{tr}_{\mathrm{pos}}.\mathrm{tr}_{\mathrm{pos}}.\mathtt{rel}.\mathrm{tr}_{\mathrm{pos}}.\mathrm{tr}_{\mathrm{pos}}.\mathtt{rel}.\mathrm{tr}_{\mathrm{pos}}\ldots}_c (\emptyset; \varnothing; \Phi)$$

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

$$\lceil A \rceil \xrightarrow{\ldots \mathrm{tr}_{\mathrm{pos}}.\mathtt{rel}.\mathrm{tr}_{\mathrm{pos}}.\mathrm{tr}_{\mathrm{pos}}.\mathtt{rel}.\mathrm{tr}_{\mathrm{pos}}.\mathrm{tr}_{\mathrm{pos}}.\mathtt{rel}.\mathrm{tr}_{\mathrm{pos}} \ldots}_c (\emptyset; \varnothing; \Phi)$$

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

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

$$A \xrightarrow{\mathrm{tr}_1.\alpha_P.\mathrm{tr}_2} (\emptyset; \Phi) \quad \rightsquigarrow \quad A \xrightarrow{\alpha_P} A_1 \xrightarrow{\mathrm{tr}_1.\mathrm{tr}_2} (\emptyset; \Phi)$$

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 $\lceil A \rceil \xrightarrow{t_c}_c \lceil A' \rceil$.

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

$$\lceil A \rceil \xrightarrow{\ldots \mathrm{tr_{pos}}.\mathtt{rel}.\mathrm{tr_{pos}}.\mathrm{tr_{pos}}.\mathtt{rel}.\mathrm{tr_{pos}}.\mathrm{tr_{pos}}.\mathtt{rel}.\mathrm{tr_{pos}} \ldots}_c (\emptyset; \varnothing; \Phi)$$

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

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

$$\begin{aligned} A \xrightarrow{\mathrm{tr}_1.\alpha_P.\mathrm{tr}_2} (\emptyset; \Phi) \quad &\rightsquigarrow \quad A \xrightarrow{\alpha_P} A_1 \xrightarrow{\mathrm{tr}_1.\mathrm{tr}_2} (\emptyset; \Phi) \\ &\rightsquigarrow \quad A \xrightarrow{\alpha_P} A_1, \ \lceil A_1 \rceil \xrightarrow{\mathrm{tr}'}_c (\emptyset; \Phi) \\ &\rightsquigarrow \quad \lceil A \rceil \xrightarrow{\alpha_P}_c \lceil A_1 \rceil \xrightarrow{\mathrm{tr}'}_c (\emptyset; \Phi) \end{aligned}$$

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

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

$$\lceil A \rceil \xrightarrow{\dots\mathrm{tr}_{\mathrm{pos}}.\mathtt{rel}.\mathrm{tr}_{\mathrm{pos}}.\mathrm{tr}_{\mathrm{pos}}.\mathtt{rel}.\mathrm{tr}_{\mathrm{pos}}.\mathrm{tr}_{\mathrm{pos}}.\mathtt{rel}.\mathrm{tr}_{\mathrm{pos}}\dots}_c (\emptyset; \varnothing; \Phi)$$

$\rightsquigarrow$ 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^-.

$$A \xrightarrow{\mathrm{tr}_{\mathrm{in}}} (\mathcal{P}' \uplus \{P^-\}; \Phi) \xrightarrow{\mathrm{tr}_0} (\emptyset; \Phi)$$

$$\rightsquigarrow \quad A = (\{P\} \uplus \mathcal{P}_0; \Phi) \xrightarrow{\mathrm{tr}_P} (\{P^-\} \uplus \mathcal{P}_0; \Phi) \xrightarrow{\mathrm{tr}'_{\mathrm{in}}} (\mathcal{P}' \uplus \{P^-\}; \Phi) \xrightarrow{\mathrm{tr}_0} (\emptyset; \Phi)$$

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 $\lceil A \rceil \xrightarrow{t_c}_c \lceil A' \rceil$.

Using the swapping Lemma we translate iteratively

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

$$\lceil A \rceil \xrightarrow{\ldots \mathrm{tr}_{\mathrm{pos}}.\mathtt{rel}.\mathrm{tr}_{\mathrm{pos}}.\mathrm{tr}_{\mathrm{pos}}.\mathtt{rel}.\mathrm{tr}_{\mathrm{pos}}.\mathrm{tr}_{\mathrm{pos}}.\mathtt{rel}.\mathrm{tr}_{\mathrm{pos}} \ldots}_c (\emptyset; \varnothing; \Phi)$$

$\rightsquigarrow$ 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^-.

$$A \xrightarrow{\mathrm{tr}_{\mathrm{in}}} (\mathcal{P}' \uplus \{P^-\}; \Phi) \xrightarrow{\mathrm{tr}_0} (\emptyset; \Phi)$$

$$\rightsquigarrow \quad A = (\{P\} \uplus \mathcal{P}_0; \Phi) \xrightarrow{\mathrm{tr}_P} (\{P^-\} \uplus \mathcal{P}_0; \Phi) \xrightarrow{\mathrm{tr}'_{\mathrm{in}}} (\mathcal{P}' \uplus \{P^-\}; \Phi) \xrightarrow{\mathrm{tr}_0} (\emptyset; \Phi)$$

$$\rightsquigarrow \quad (\{P\} \uplus \mathcal{P}_0; \Phi) \xrightarrow{\mathrm{tr}_P} (\{P^-\} \uplus \mathcal{P}_0; \Phi), \; (\{P^-\} \uplus \mathcal{P}_0; \varnothing; \Phi) \xrightarrow{\mathrm{tr}'}_c (\emptyset; \Phi)$$

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 $\lceil A \rceil \xrightarrow{t_c}_c \lceil A' \rceil$.

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

$$\lceil A \rceil \xrightarrow{\ldots \mathrm{tr}_{\mathrm{pos}}.\mathtt{rel}.\mathrm{tr}_{\mathrm{pos}}.\mathrm{tr}_{\mathrm{pos}}.\mathtt{rel}.\mathrm{tr}_{\mathrm{pos}}.\mathrm{tr}_{\mathrm{pos}}.\mathtt{rel}.\mathrm{tr}_{\mathrm{pos}} \ldots}_c (\emptyset; \varnothing; \Phi)$$

$\rightsquigarrow$ 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^-.

$$\begin{array}{ll}
& A \xrightarrow{\mathrm{tr}_{\mathrm{in}}} (\mathcal{P}' \uplus \{P^-\}; \Phi) \xrightarrow{\mathrm{tr}_0} (\emptyset; \Phi) \\
\rightsquigarrow & A = (\{P\} \uplus \mathcal{P}_0; \Phi) \xrightarrow{\mathrm{tr}_P} (\{P^-\} \uplus \mathcal{P}_0; \Phi) \xrightarrow{\mathrm{tr}'_{\mathrm{in}}} (\mathcal{P}' \uplus \{P^-\}; \Phi) \xrightarrow{\mathrm{tr}_0} (\emptyset; \Phi) \\
\rightsquigarrow & (\{P\} \uplus \mathcal{P}_0; \Phi) \xrightarrow{\mathrm{tr}_P} (\{P^-\} \uplus \mathcal{P}_0; \Phi),\ (\{P^-\} \uplus \mathcal{P}_0; \varnothing; \Phi) \xrightarrow{\mathrm{tr}'}_c (\emptyset; \Phi) \\
\rightsquigarrow & \lceil A \rceil \xrightarrow{\mathtt{foc}(\mathrm{tr}_P).\mathtt{rel}}_c (\{P^-\} \uplus \mathcal{P}_0; \varnothing; \Phi) \xrightarrow{\mathrm{tr}'}_c (\emptyset; \Phi)
\end{array}$$

Results - Equivalence

Compressed trace equivalence

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

Results - Equivalence

Compressed trace equivalence

$A \approx_c B$ if for any labelled trace t and execution $A \xrightarrow{t}_c (\mathcal{P}; \varnothing; \Phi)$ there is $B \xrightarrow{t}_c (\mathcal{P}'; \varnothing; \Phi')$ 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.

Results - Equivalence - Completeness

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 $(\xrightarrow{\mathrm{tr}}) \rightsquigarrow (\xrightarrow{\mathrm{tr}_c}_c)$ in $\lceil B \rceil$'s answer

Outline

Intuitions

By building upon $\rightarrow_c, \approx_c$:

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

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

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

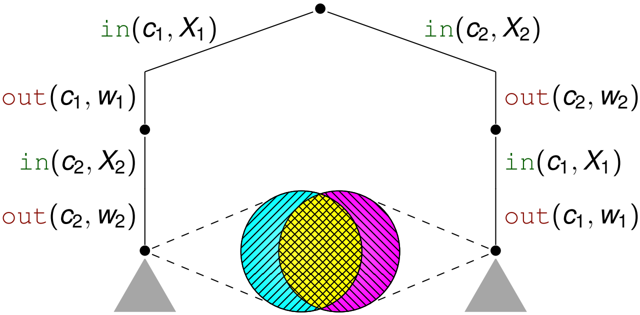
Intuitions

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

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

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

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



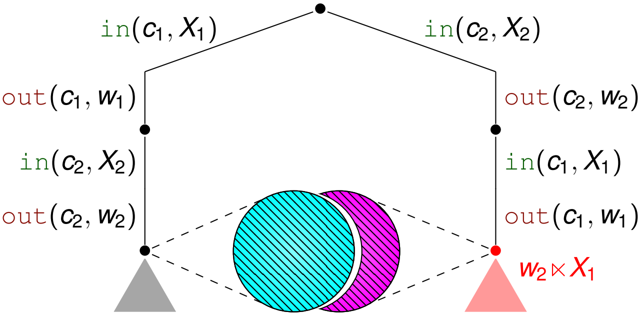
Intuitions

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

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

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

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



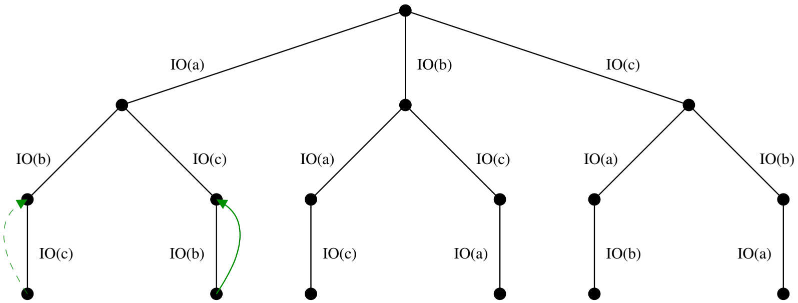
X_1 must depend on w_2.

Intuitions: More redundancies

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

Intuitions: More redundancies

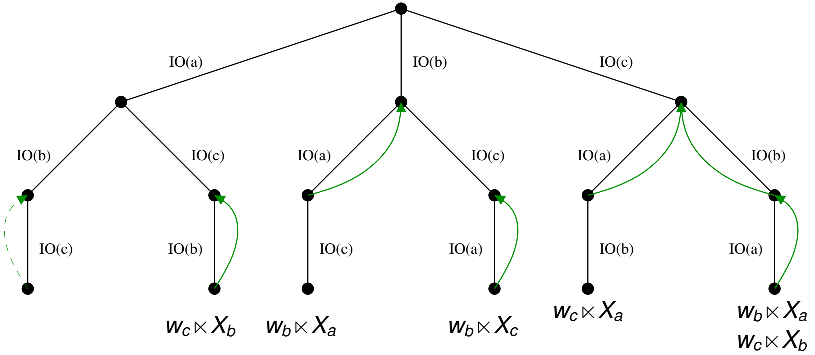
$$P = IO(a)|IO(b)|IO(c) \text{ where } IO(l) = \texttt{in}(c_l, X_l).\texttt{out}(c_l, w_l)$$



$w_c \ltimes X_b$

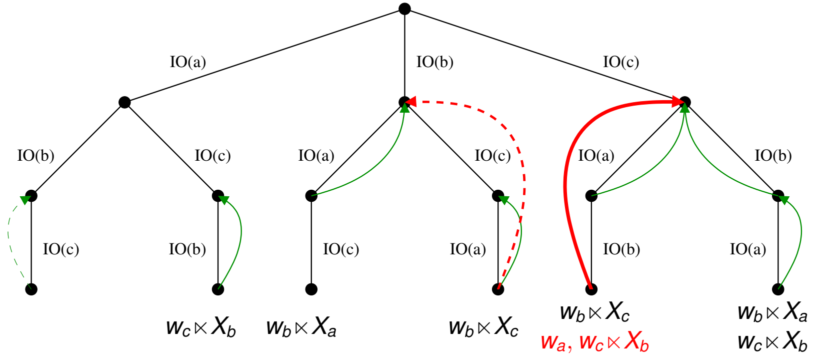
Intuitions: More redundancies

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



Intuitions: More redundancies

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



Monoid of traces

Definition

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

- $\mathrm{tr}.b_1.b_2.\mathrm{tr}' \equiv_\Phi \mathrm{tr}.b_2.b_1.\mathrm{tr}'$ when $b_1 \parallel b_2$, and
- $\mathrm{tr}.b_1.\mathrm{tr}' \equiv_\Phi \mathrm{tr}.b_2.\mathrm{tr}'$ when $(b_1 =_\mathsf{E} b_2)\Phi$.

Monoid of traces

Definition

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

- $\mathrm{tr}.b_1.b_2.\mathrm{tr}' \equiv_\Phi \mathrm{tr}.b_2.b_1.\mathrm{tr}'$ when $b_1 \parallel b_2$, and
- $\mathrm{tr}.b_1.\mathrm{tr}' \equiv_\Phi \mathrm{tr}.b_2.\mathrm{tr}'$ when $(b_1 =_\mathsf{E} b_2)\Phi$.

Lemma

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

Goal: explore on trace per equivalence class.

Reduced semantics

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

Semantics

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

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

transparent=0

Availability

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

- either $\mathrm{tr} = \epsilon$
- or $\mathrm{tr} = \mathrm{tr}_0.b_0$ with $\neg(b_0 \| b)$
- or $\mathrm{tr} = \mathrm{tr}_0.b_0$ with $b_0 \| b$, $b_0 \prec b$ and $\mathrm{tr}_0 \ltimes b$.

Results - Reachability

Done: explore on trace per equivalence class.

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

Lemma: completeness for reachability

If A and $A' = (\mathcal{P}'; \Phi')$ are initial and $\lceil A \rceil \xrightarrow{t}_c \lceil A' \rceil$ 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 is Φ-minimal if there is no $t' \prec_{\text{lex}} t$ such that $t \equiv_\Phi t'$

Lemma: completeness for reachability

If A and $A' = (\mathcal{P}'; \Phi')$ are initial and $\lceil A \rceil \xrightarrow{t}_c \lceil A' \rceil$ 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

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

Theorem

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

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

Results - Equivalence

Definition: Reduced trace equivalence

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

Theorem

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

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

- Reachabilty 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

Implementations

Adapting well established techniques based on:

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

Implementations

Adapting well established techniques based on:

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

$\mathrm{tr} \ltimes b$ (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.

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

Difficulties:

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

Implementations

Adapting well established techniques based on:

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

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

Difficulties:

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

Results in APTE & SPEC:

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
SPEC	2 par	2/22/10	$>$ 20 hours	13853	122.27
	7 par	7/14/2	$>$ 20 hours	13853	370.65

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 $\rightsquigarrow$ 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 $\rightsquigarrow$ recognize symmetries ?
   - study constraint solving in more details
3. introduce interactivity into the verification process (sub-lemmas, annotations)

Any question?