Describing Secure Interfaces with Interface Automata

Matias Lee Pedro R. D'Argenio [1]

FaMAF - UNC
[1] Also affiliated to CONICET

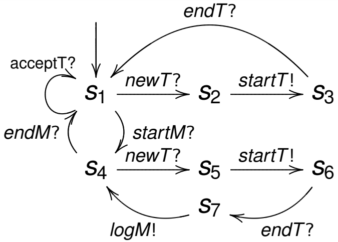
Workshop ReSeCo

Outline

Outline

1. Interfaces Structure for Security
   - Interfaces Automata and Interface Structure For Security
   - Composition
   - Bisimulation-based (Strong) Non-deterministic Non-interference
2. Deriving secure ISS
   - Checking BSNNI
   - Synthesizing Secure ISS
   - The algorithm in the Initial Example
3. Preserving BSNNI after Composition
   - Preserving BSNNI after Composition

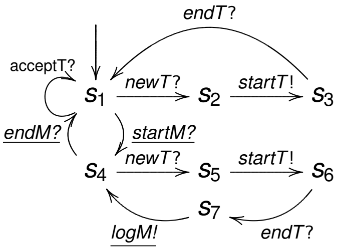
An Interface Automata (IA):



Definition

An *Interface Automaton* (IA) is a tuple $S = \langle Q, q^0, A^I, A^O, A^H, \rightarrow \rangle$ where: (i) Q is a set of *states* with $q^0 \in Q$ being the *initial state*; (ii) A^I, A^O, and A^H are the (pairwise disjoint) sets of *input*, *output*, and *hidden actions*, respectively, with $A = A^I \cup A^O \cup A^H$; and ($iii$) $\rightarrow \subseteq Q \times A \times Q$ is the *transition relation* and we require that it is *input deterministic*.

A Interface Structure for Security (ISS)



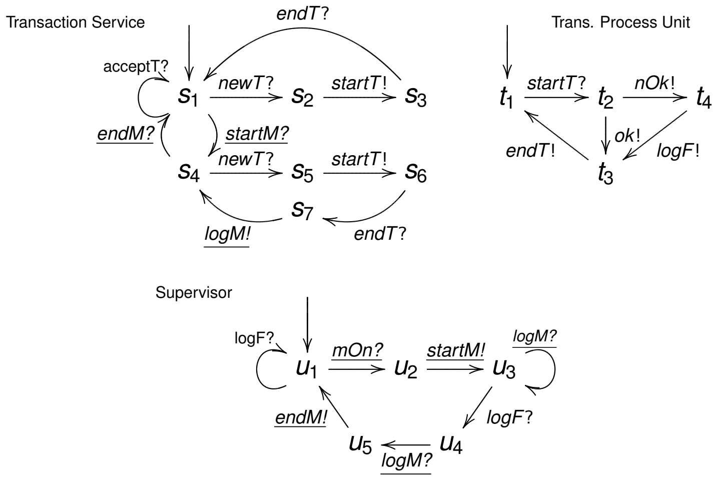
Definition

An *Interface Structure for Security (ISS)* is a tuple $\langle S, A^h, A^l \rangle$ where S is an IA and A^h and A^l are disjoint sets of actions s.t. $A^h \cup A^l = A^O \cup A^I$.

Why IA and ISS?

- Component Based Development and Design has become main approach for software development. Example: *web services*.
- Then, we need good interface description that allows to analyze interaction between components. This way, we can predict if the composed system can satisfy our requirements.
- IA captures temporal aspects of the component interface. In this framework the requirement is that the communication is properly carried out by the interfaces.
- ISS inherits the properties of IA and allows us to study properties related with secure data flow.

Example:


Transaction Service
endT?
acceptT?
newT?
startT!
endM?
startM?
newT?
startT!
logM!
endT?
Trans. Process Unit
startT?
nOk!
ok!
endT!
logF!
Supervisor
logF?
mOn?
startM!
logM?
endM!
logF?
logM?

Outline

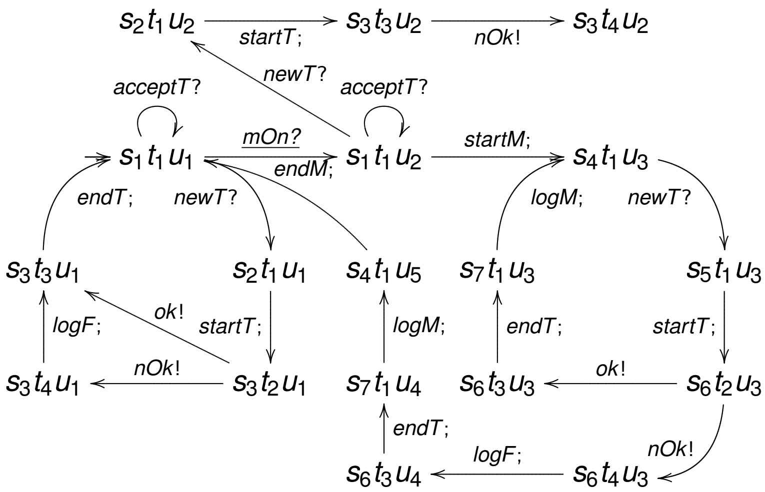
Composition

The product of two composable IA S and T is defined pretty much as CSP parallel composition:

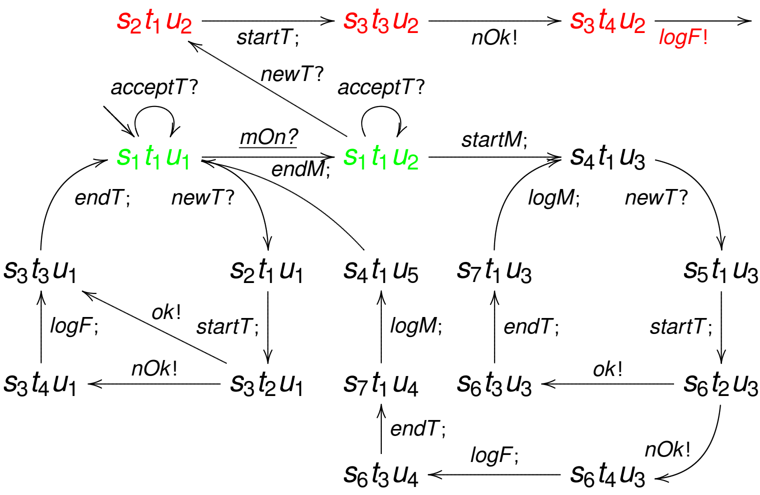
- the state space of the product is the product of the set of states of the components,
- shared actions can only synchronize, i.e., both component should perform a transition with the same synchronizing label (one input, and the other output), and
- transitions with non-shared actions are interleaved.

Besides, shared actions are hidden in the product.

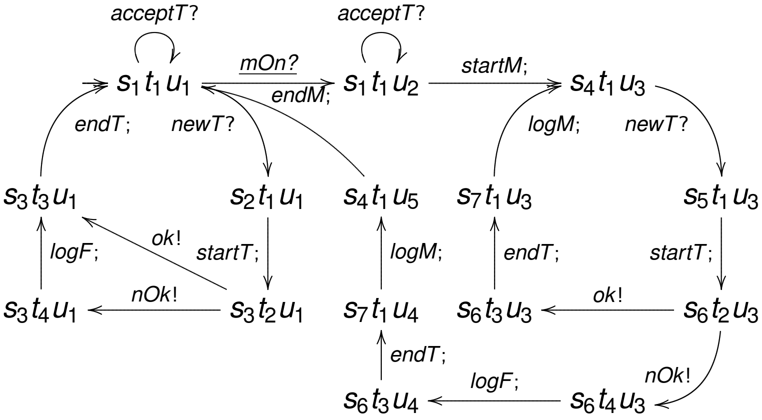
First Step: Product



Error, Incomp. and Compatible states - Compatibles IA



2nd Step: Avoid to reach incompatibles states



Outline

1. Interfaces Structure for Security
   - Interfaces Automata and Interface Structure For Security
   - Composition
   - Bisimulation-based (Strong) Non-deterministic Non-interference
2. Deriving secure ISS
   - Checking BSNNI
   - Synthesizing Secure ISS
   - The algorithm in the Initial Example
3. Preserving BSNNI after Composition
   - Preserving BSNNI after Composition

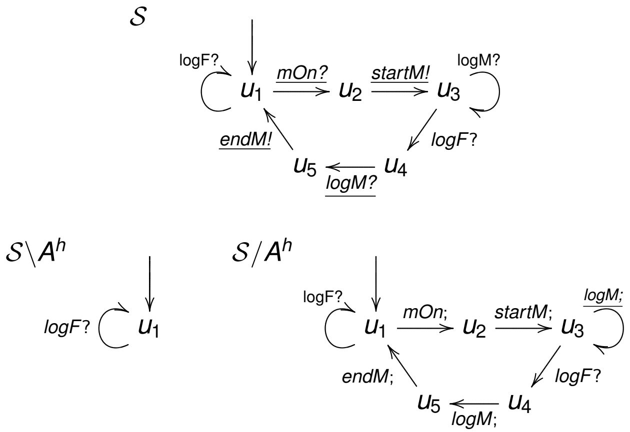
BSNNI and BNNI

- $\mathcal{S} \approx \mathcal{S}'$ represents there is weak bisimulation between $\mathcal{S}$ and $\mathcal{S}'$.
- $\mathcal{S}/X$ represents the hiding of actions X in $\mathcal{S}$
- $\mathcal{S}\backslash X$ represents the restriction of actions X in $\mathcal{S}$

Definition

Let $\mathcal{S}$ be an ISS.
(i) $\mathcal{S}$ is *bisimulation-based strong non-deterministic non-interference (BSNNI)* if $\mathcal{S}\backslash A^h \approx \mathcal{S}/A^h$.
(ii) $\mathcal{S}$ is *bisimulation-based non-deterministic non-interference (BNNI)* if $\mathcal{S}\backslash A^{I,h}/A^{O,h} \approx \mathcal{S}/A^h$.

Example: $\mathcal{S}$ is BSNNI



BSNNI and Composition

All the ISS presented in the example are BSNNI but...

... the composed system is not! :'(

BSNNI and Composition

All the ISS presented in the example are BSNNI but...
... the composed system is not! :'(

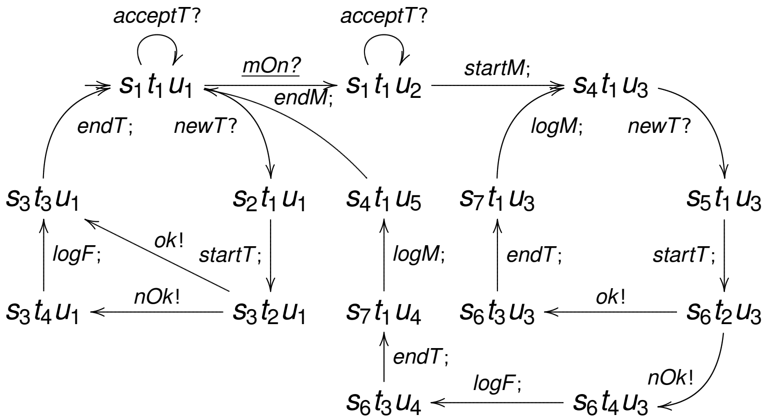
Outline

Checking Bisimulation

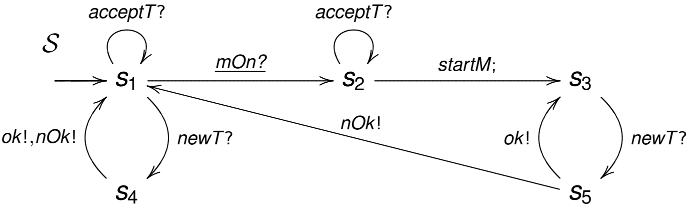
Our algorithm is a variation of Fernandez and Mounier to check bisimulation *on the fly*. Roughly, our algorithm works as follows:

- IA are saturated adding all weak transitions
- a full synchronous product is constructed where transitions synchronize whenever they have the same label;
- whenever there is a mismatching transition, a new transition is added on the product leading to a special *fail* state;
- if reaching a fail state is inevitable (we later define this properly) the IA are not bisimilar; if there is always a way to avoid reaching a fail state, the IA are bisimilar.

Original Composed System

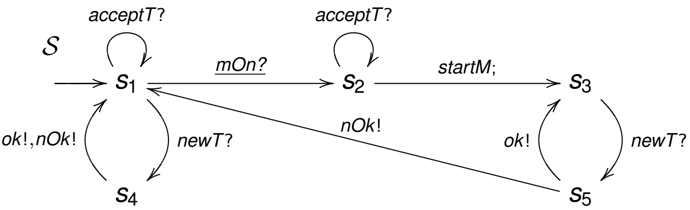


Simplified Composed System

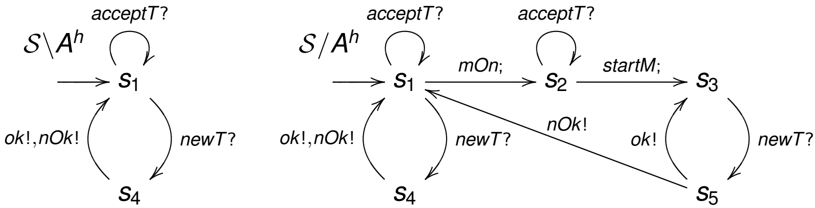


and we want to check bisimulation between:

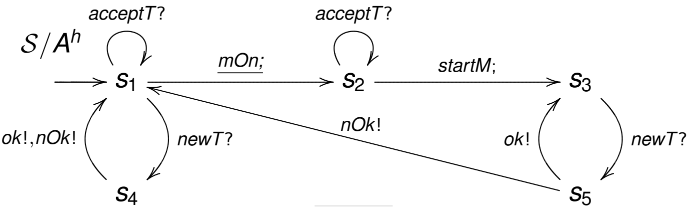
Simplified Composed System



and we want to check bisimulation between:

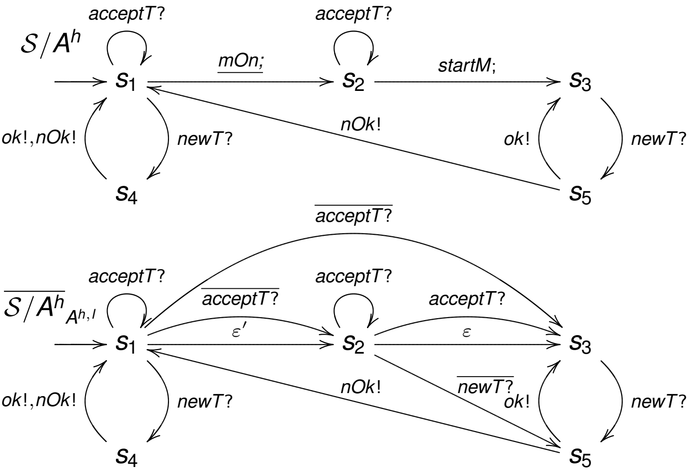


Saturation marking set B. $B = \{\underline{mOn?}\}$



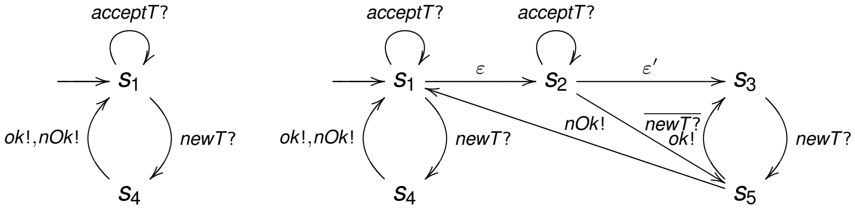
Note: We will omit the action added by the saturation process that are not necessary.

Saturation marking set B. $B = \{\underline{mOn?}\}$



Note: We will omit the action added by the saturation process that are not necessary.

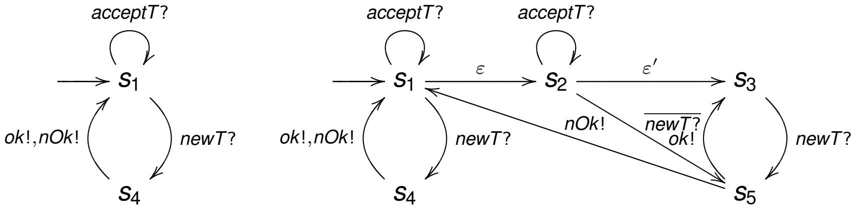
Synchronized Product: $\overline{\mathcal{S}\backslash A^h}_{\emptyset} \times \overline{\mathcal{S}/A^h}_{A^{h,I}}$



We can start with the synchronized product:

S_1, S_1

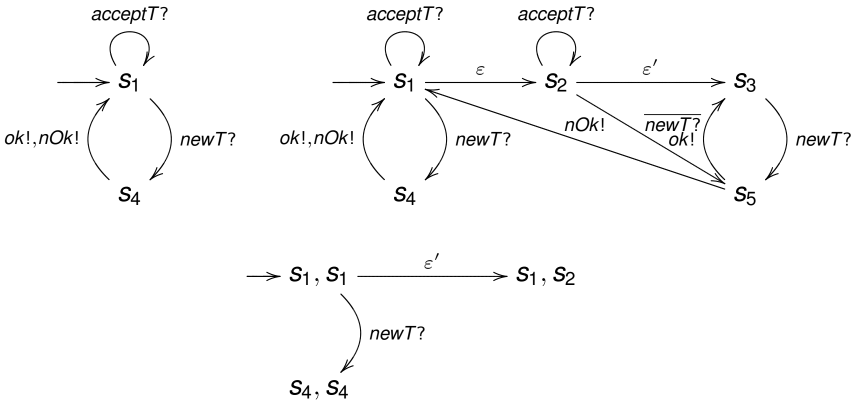
Synchronized Product: $\overline{\mathcal{S}\backslash A^h}_{\emptyset} \times \overline{\mathcal{S}/A^h}_{A^{h,l}}$



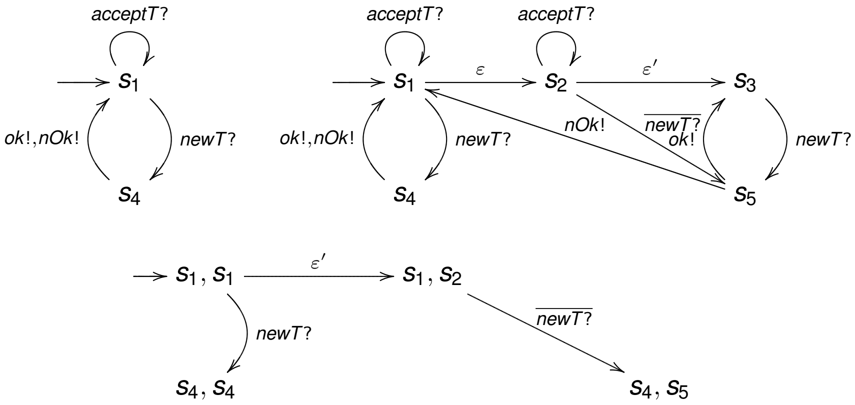
We can start with the synchronized product:

$$s_1, s_1$$

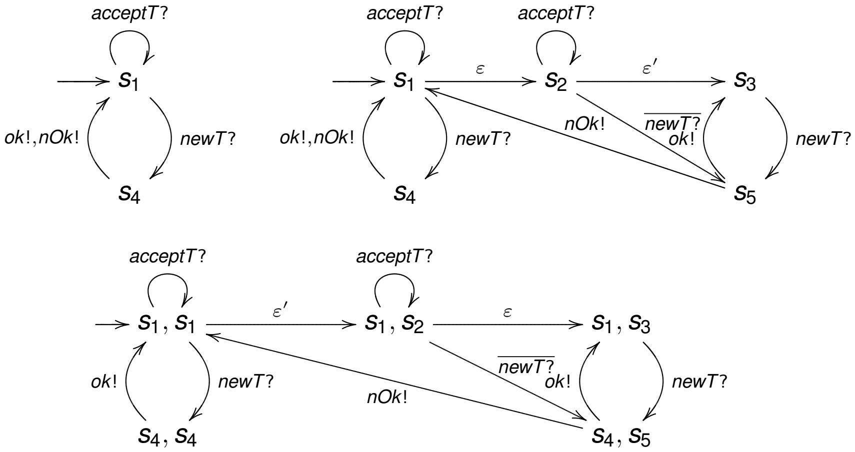
Synchronized Product: $\overline{\mathcal{S} \backslash A^h}_{\emptyset} \times \overline{\mathcal{S} / A^h}_{A^{h,l}}$



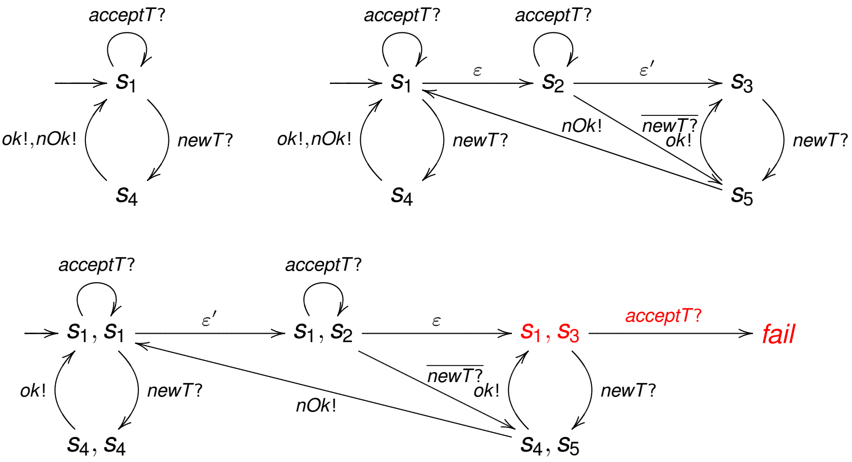
Synchronized Product: $\overline{\mathcal{S} \backslash A^h}_{\emptyset} \times \overline{\mathcal{S} / A^h}_{A^{h,I}}$



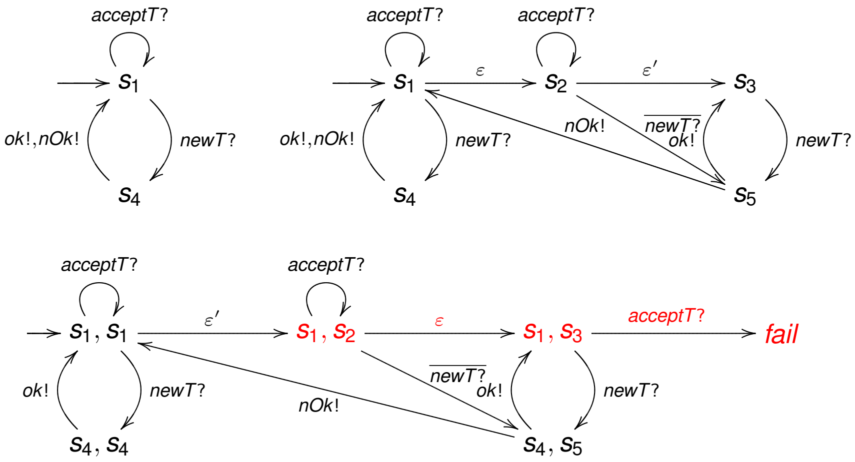
Synchronized Product: $\overline{\mathcal{S}\backslash A^h}_{\emptyset} \times \overline{\mathcal{S}/A^h}_{A^{h,l}}$



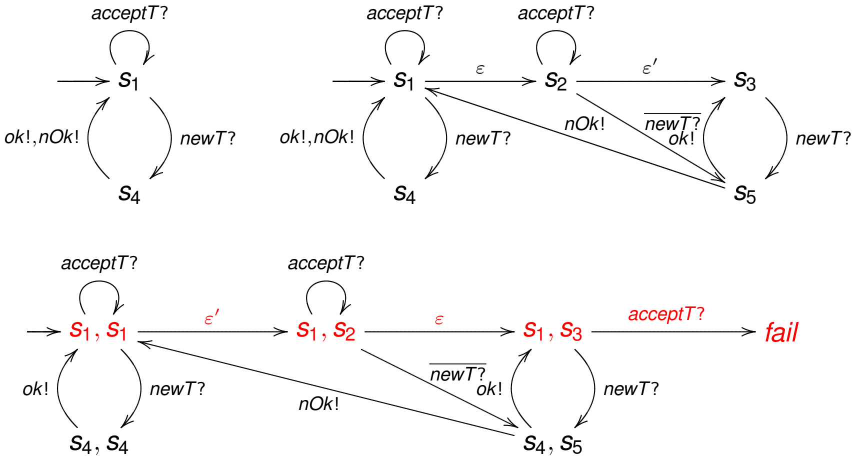
Synchronized Product: $\overline{\mathcal{S}\backslash A^h}_{\emptyset} \times \overline{\mathcal{S}/A^h}_{A^{h,l}}$



Synchronized Product: $\overline{\mathcal{S}\backslash A^h}_{\emptyset} \times \overline{\mathcal{S}/A^h}_{A^{h,l}}$



Synchronized Product: $\overline{\mathcal{S} \backslash A^h}_{\emptyset} \times \overline{\mathcal{S} / A^h}_{A^{h,l}}$



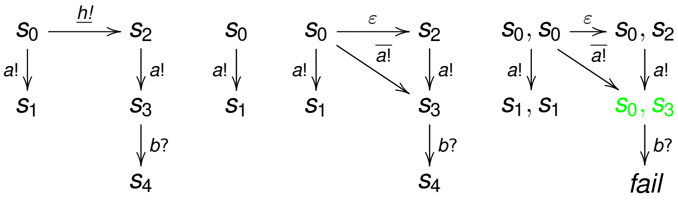
Outline

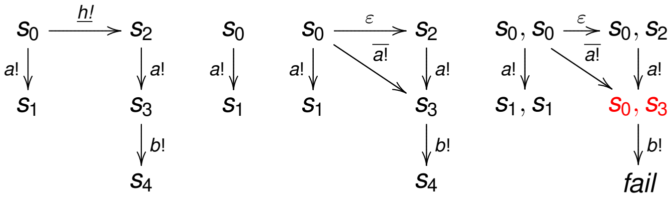
Synthesizing Secure ISS

If the system does not pass a the simulation test, i.e. the initial state of the synchronized product does not contain a pair of bisimilar states, we can divide all the state that does not pass the bisimulation test in 3 disjoint set:

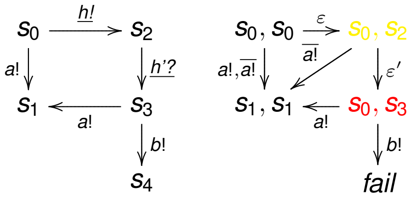
- May State: contains pairs of states of product synchronization such that if some low input transitions are pruned this pairs become bisimilar in the new product.
- Fail State: contains pairs of states that cannot be turned into bisimilar by pruning low input transitions.
- Undetermined state: contains undetermined pair of states. This is consequence that they may become bisimilar if a high input transitions is removed, but remove the transition can create a new problem.

Example of May and Fail states:



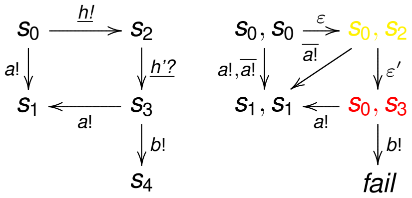


Example 1 of Undetermined states:

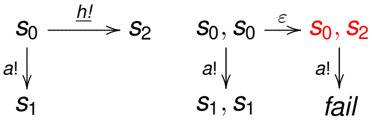


If we remove transition $s_2 \xrightarrow{h'?} s_3$, we obtain the next interface that is not secure:

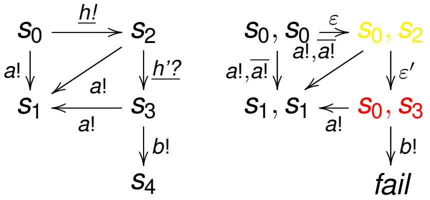
Example 1 of Undetermined states:



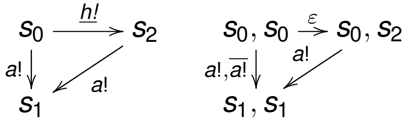
If we remove transition $s_2 \xrightarrow{h'?} s_3$, we obtain the next interface that is not secure:



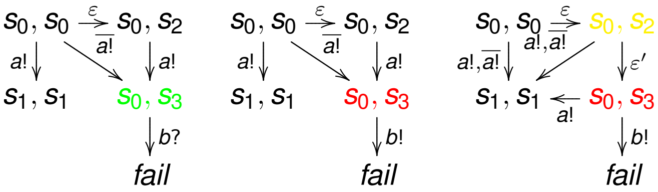
Example 2 of Undetermined states:



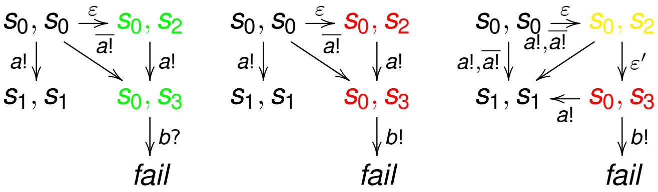
In this case, if we remove transition $s_2 \xrightarrow{h'?} s_3$, we obtain a secure interface:



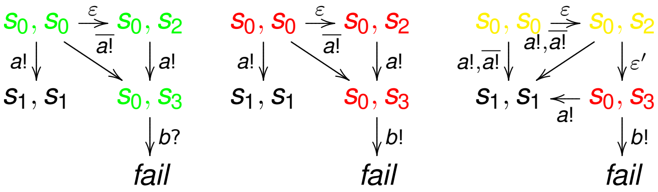
May/Fail/Undetermined ISS



May/Fail/Undetermined ISS



May/Fail/Undetermined ISS



The main results of this work

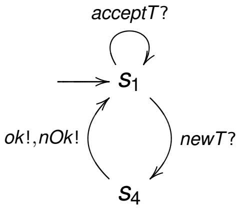
Theorem

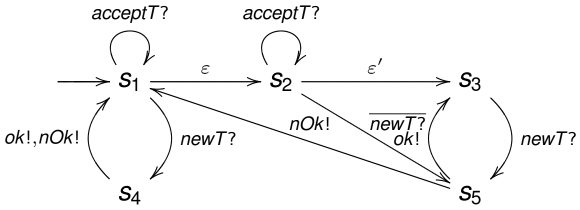
Let $\mathcal{S}$ be an ISS s.t. $P_{\mathcal{S}}$ may pass the bisimulation test. Then there exists a set $\rightarrow_{\chi}$ of low input transitions such that, if $\mathcal{S}'$ is the ISS obtained from $\mathcal{S}$ by removing all transitions in $\rightarrow_{\chi}$, $\mathcal{S}'$ is BSNNI.

- The set $\rightarrow_{\chi}$ is included in a particular input set called rmCandidates($\mathcal{S}$)
- rmCandidates($\mathcal{S}$) is defined using May states definition.
- The proof is constructive and it defines an algorithm.

Outline

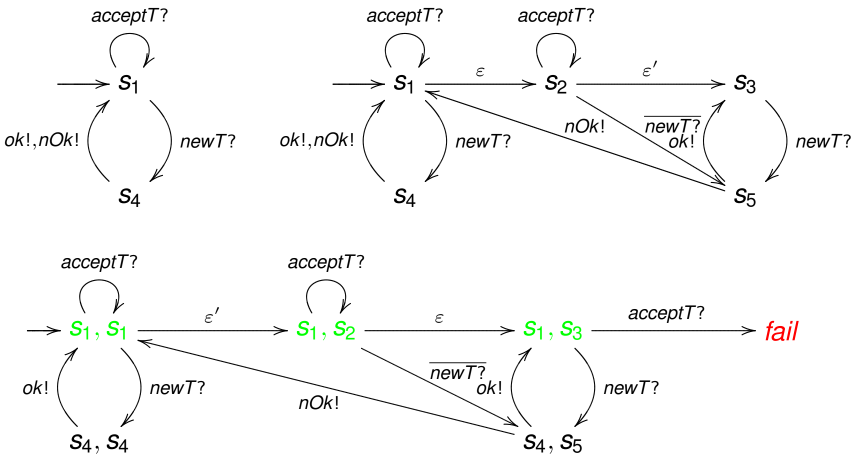
Iteration 1





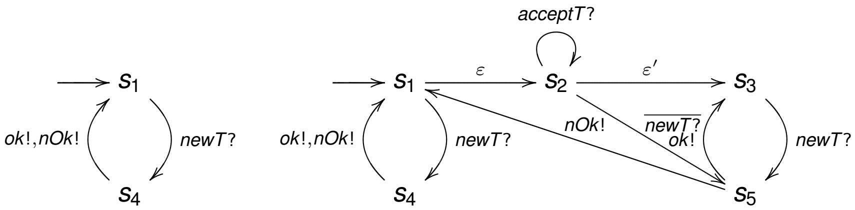
$\mathrm{rmCandidates}(\mathcal{S}) = \{s_1 \xrightarrow{a?} s_1\}$

Iteration 1



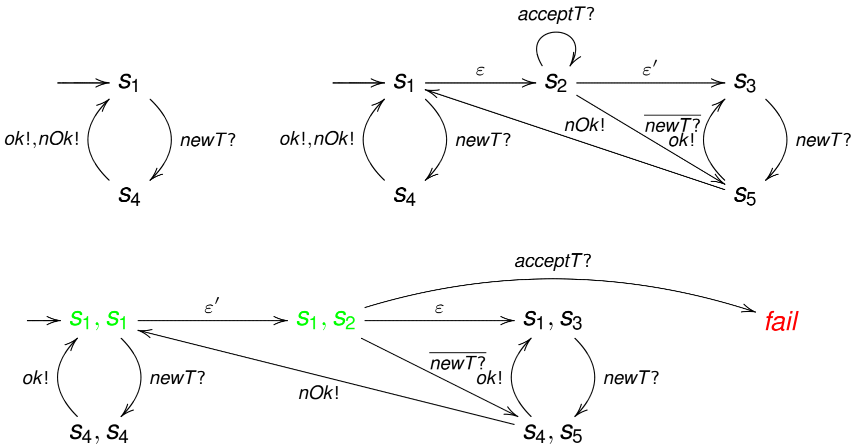
$$\text{rmCandidates}(\mathcal{S}) = \{s_1 \xrightarrow{a?} s_1\}$$

Iteration 2



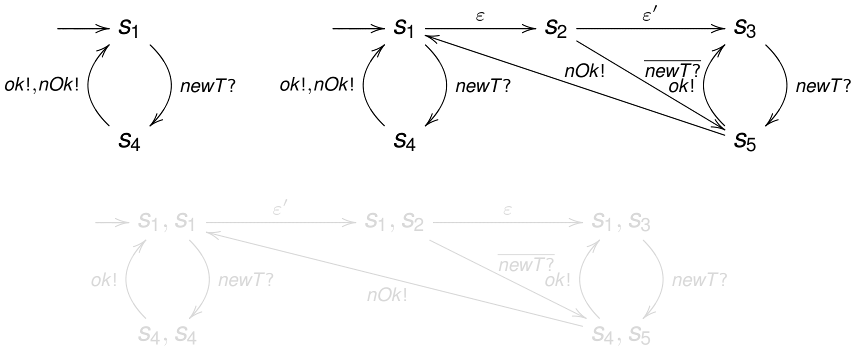
$\text{rmCandidates}(\mathcal{S}) = \{s_2 \xrightarrow{a?} s_2\}$

Iteration 2

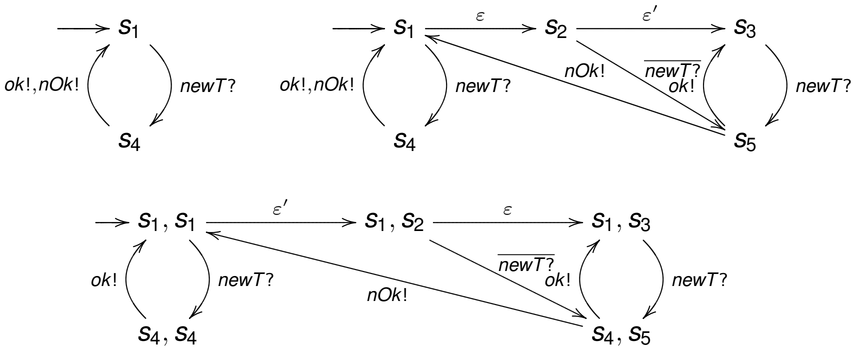


$$\text{rmCandidates}(\mathcal{S}) = \{s_2 \xrightarrow{a?} s_2\}$$

Iteration 3


s_1
s_4
$ok!, nOk!$
$newT?$
s_1
s_2
s_3
s_4
s_5
ε
ε'
$nOk!$
$\overline{newT?}$
$ok!$
$ok!, nOk!$
$newT?$
$newT?$
s_1, s_1
s_1, s_2
s_1, s_3
s_4, s_4
s_4, s_5
ε'
ε
$ok!$
$newT?$
$nOk!$
$\overline{newT?}$
$ok!$
$newT?$

Iteration 3


s_1
ok!,nOk!
newT?
s_4
s_1
ε
s_2
ε′
s_3
ok!,nOk!
newT?
nOk!
$\overline{newT?}$
ok!
newT?
s_4
s_5
s_1, s_1
ε′
s_1, s_2
ε
s_1, s_3
ok!
newT?
$\overline{newT?}$
ok!
newT?
nOk!
s_4, s_4
s_4, s_5

Outline

Lemma

Let $\mathcal{S} = \langle S, A^h_S, A^l_S \rangle$ and $\mathcal{T} = \langle T, A^h_T, A^l_T \rangle$ be two composable ISS.
We define

- $\mathcal{S}' = \langle S, A^h_S - shared(S,T), A^l_S \cup shared(S,T) \rangle$
- $\mathcal{T}' = \langle T, A^h_T - shared(S,T), A^l_T \cup shared(S,T) \rangle$

If $\mathcal{S}'$ and $\mathcal{T}'$ are BSNNI/BNNI and $\mathcal{S} \otimes \mathcal{T}$ has not error states, then $\mathcal{S} \parallel \mathcal{T}$ is BSNNI/BNNI.

Example 1

Two ISS satisfy the hypothesis of the theorem.

$$\begin{array}{ccccc} s_0 & \xrightarrow{H_1?} & s_1 & \xrightarrow{a!} & s_2 \\ {\scriptstyle H_2!}\downarrow & & & & \\ s_3 & \xrightarrow{H_1?} & s_4 & \xrightarrow{a!} & s_5 \end{array} \qquad t_0 \xrightarrow{H_1!} t_1$$

Its synchronized product:

$$\begin{array}{ccccc} s_0,t_0 & \xrightarrow{H_1;} & s_1,t_1 & \xrightarrow{a!} & s_2,t_1 \\ {\scriptstyle H_2!}\downarrow & & & & \\ s_3,t_0 & \xrightarrow{H_1;} & s_4,t_1 & \xrightarrow{a!} & s_5,t_1 \end{array}$$

Example 2

Two ISS do not satisfy the hypothesis of the theorem.

$$
\begin{array}{ccccccc}
s_0 & \xrightarrow{H_1?} & s_1 & \xrightarrow{H_2!} & s_2 & \xrightarrow{H_1?} & s_3 \\
\downarrow a! & & \downarrow a! & & & & \downarrow a! \\
s_4 & \xrightarrow{H_1?} & s_5 & & & & s_6
\end{array}
\qquad
t_0 \xrightarrow{H_1!} t_1
$$

Its synchronized product:

$$
\begin{array}{ccccc}
s_0, t_0 & \xrightarrow{H_1;} & s_1, t_1 & \xrightarrow{H_2!} & s_2, t_1 \\
\downarrow a! & & \downarrow a! & & \\
s_4, t_0 & \xrightarrow{H_1;} & s_5, t_1 & &
\end{array}
$$

The end!

questions?