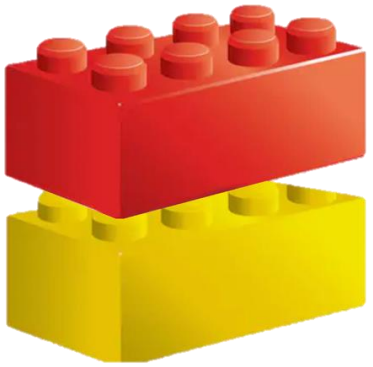
Composition



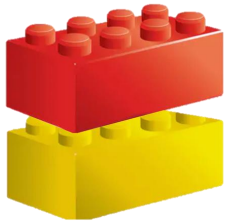
Sayan Mitra

Verifying cyberphysical systems

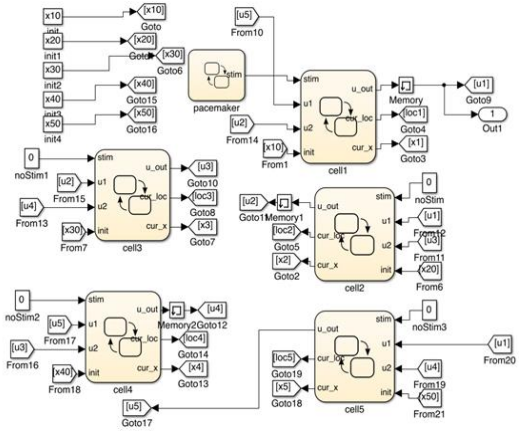
mitras@illinois.edu

- Chapter 5

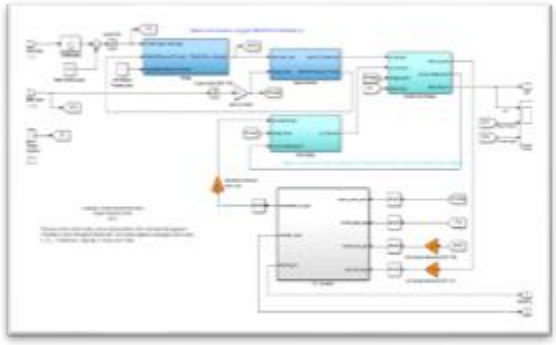
Why compose?



- Complex models are built by putting together **simpler components** or **modules**
- **Enables** portability, reuse, modular reasoning
- **Composition:** operation of *putting together*
- Gives precise definition of module **interfaces**
- What properties are preserved under composition?

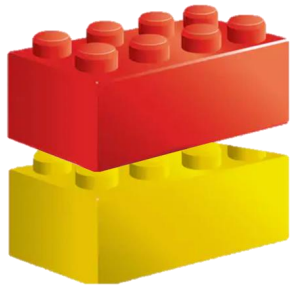


model of a network of oscillators [Huang et. al 14]



Powertrain model from Toyota [Jin et al. 15]

Examples of “composition”



- Addition. A + B = C
- Product. A x B = C
- Concatenation. Append(A,B)
- Function composition. A(B(x)) = C(x)
- Automata (parallel) composition. C = A || B
- Reasoning with composition

$$A \leq B \; implies \; A + C \leq B + C$$

$$A \leq_{sim} B \; implies \; A \,||\, C \leq_{sim} B \,||\, C$$

- Other examples of composition?

- Is your notion of composition is captured by what we will define

Outline

- Composition operation
  - Input/output interfaces
- I/O automata
- hybrid I/O automata
- Examples
- Properties of composition

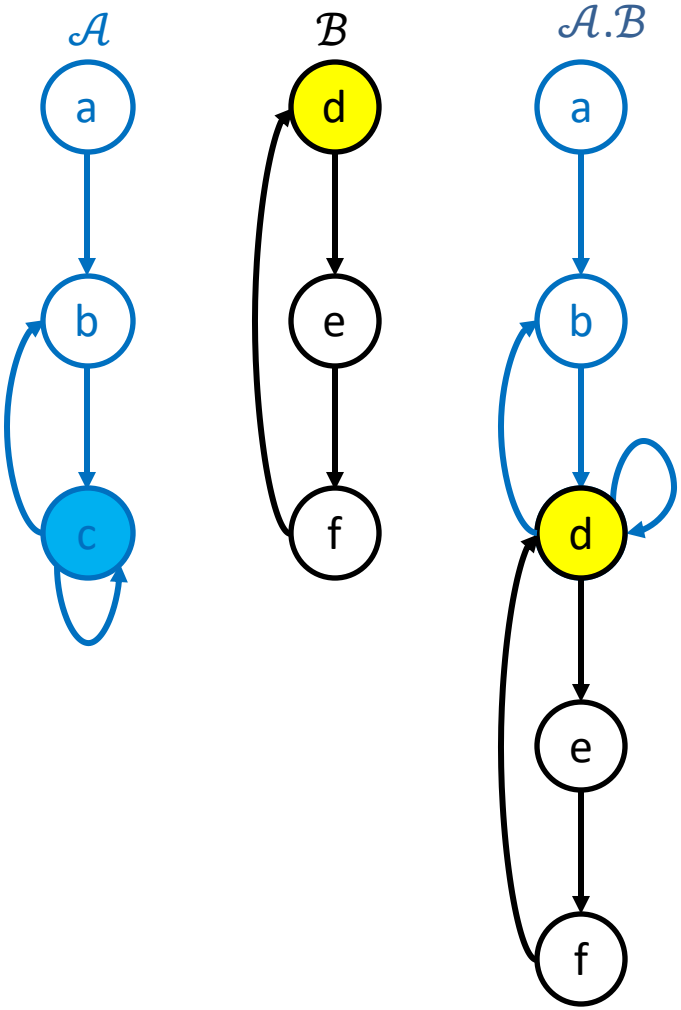
Composition of automata

Recall $\mathcal{A} = \langle X, \Theta, A, \boldsymbol{D} \rangle$

$\mathcal{A} = \mathcal{A}_1 || \mathcal{A}_2$

- $\mathcal{A}_1, \mathcal{A}_2$ are the *component automata* and
- $\mathcal{A}$ is the *composed* automaton
- $||$ symbol for the composition operator

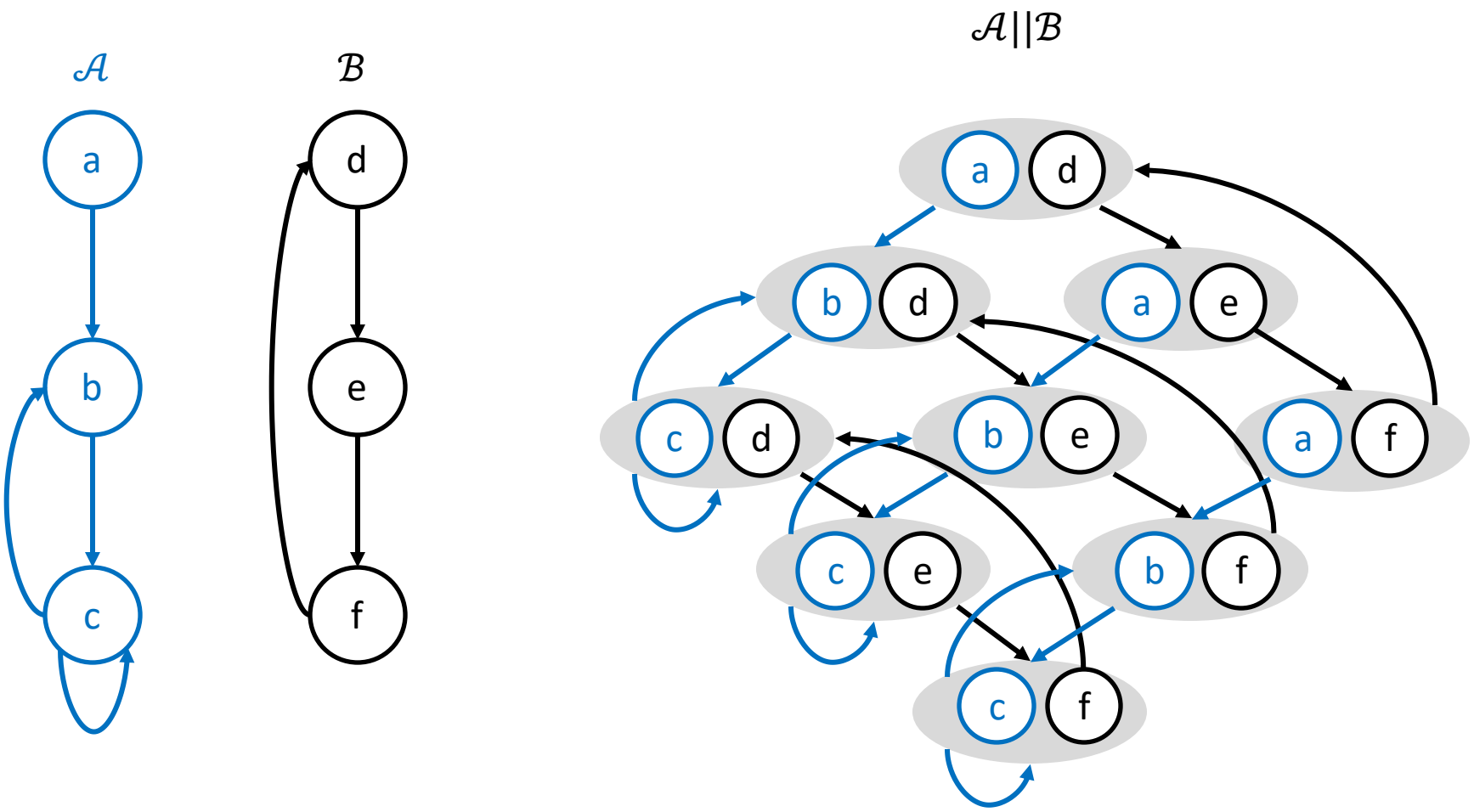
Sequential composition



In general, every final state has to be merged with every start state.

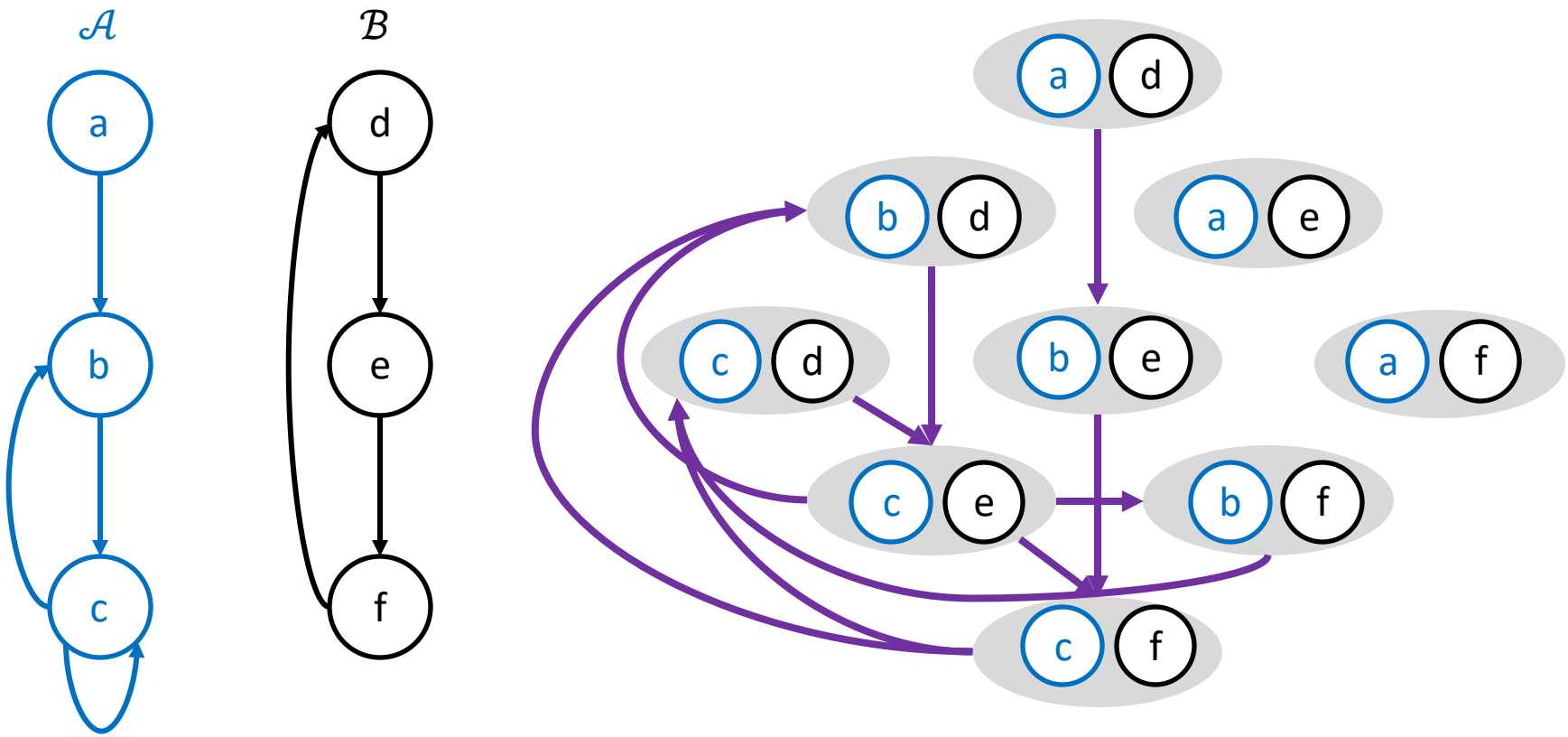
Not particularly interesting

Parallel Composition: asynchronous transitions



$$\mathcal{A}||\mathcal{B} = \langle Q_A \times Q_B, \Theta_A \times \Theta_B, A, \boldsymbol{D} \rangle$$

parallel composition: synchronous transitions



Composition of (discrete) automata

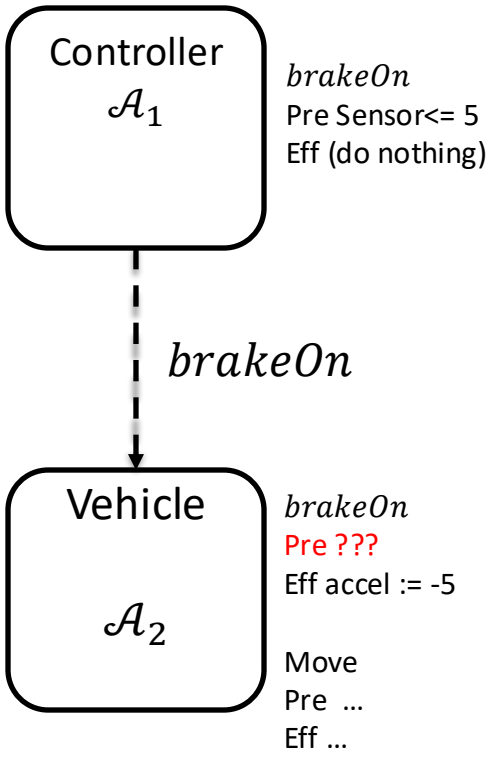
- More generally, some transitions of $\mathcal{A}$ and $\mathcal{B}$ may synchronize, while others may not synchronize
- Further, some transitions may be **controlled** by $\mathcal{A}$ which when occurs **forces** the corresponding transition of $\mathcal{B}$
- Thus, we will partition the set of actions A of $\mathcal{A} = \langle X, \Theta, A, \boldsymbol{D} \rangle$ into
  - H: **internal** (do not synchronize)
  - O: **output** (synchronized and controlled by $\mathcal{A}$)
  - I: **input** (synchronized and controlled by some other automaton)
- $A = H \cup O \cup I$
- This gives rise to **I/O automata** **[Lynch, Tuttle 1996]**

Reactivity: Input enabling

Consider a shared action **brakeOn** controlled by $\mathcal{A}_1$ and listend-to or read by $\mathcal{A}_2$

What happens if $\mathcal{A}_2$ has a state x from which there is no well-defined behavior/transition labeled by **brakeOn?**

Input enabling ensures that when $\mathcal{A}_1$ and $\mathcal{A}_2$ are composed then $\mathcal{A}_2$ can react to **brakeOn**



Definition. An **input/output automaton** is a tuple $\mathcal{A} = \langle X, \Theta, A, \mathcal{D} \rangle$ where

- X is a set of names of variables
- $\Theta \subseteq val(X)$ is the set of initial states
- $A = I \cup O \cup H$ is a set of names of actions
- $\mathcal{D} \subseteq val(X) \times A \times val(X)$ is the set of transitions and $\mathcal{A}$ satisfies the input enabling condition:

E1. For each $\boldsymbol{x} \in val(X), a \in I$ there exists $\boldsymbol{x'} \in val(X)$ such that $\boldsymbol{x} \rightarrow_a \boldsymbol{x'}$

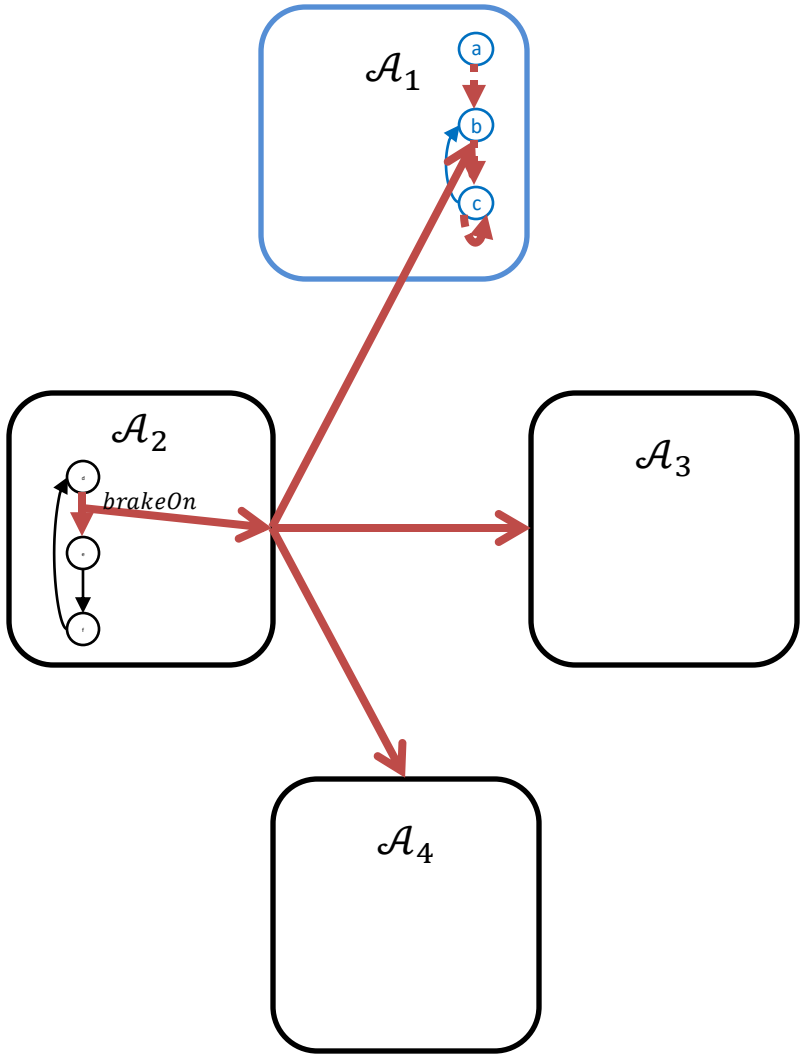
Compatibility IOA

A pair of I/O automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are compatible if

$H_i \cap A_j = \emptyset$ no unintended interactions

$O_i \cap O_j = \emptyset$ no shared control

Extended to collection of automata in the natural way



Composition of I/O automaton

Definition. For compatible automata $\mathcal{A}_1$ and $\mathcal{A}_2$ their composition $\mathcal{A}_1 \,||\, \mathcal{A}_2$ is the structure $\boldsymbol{\mathcal{A}} = (X, \Theta, A, \mathcal{D})$

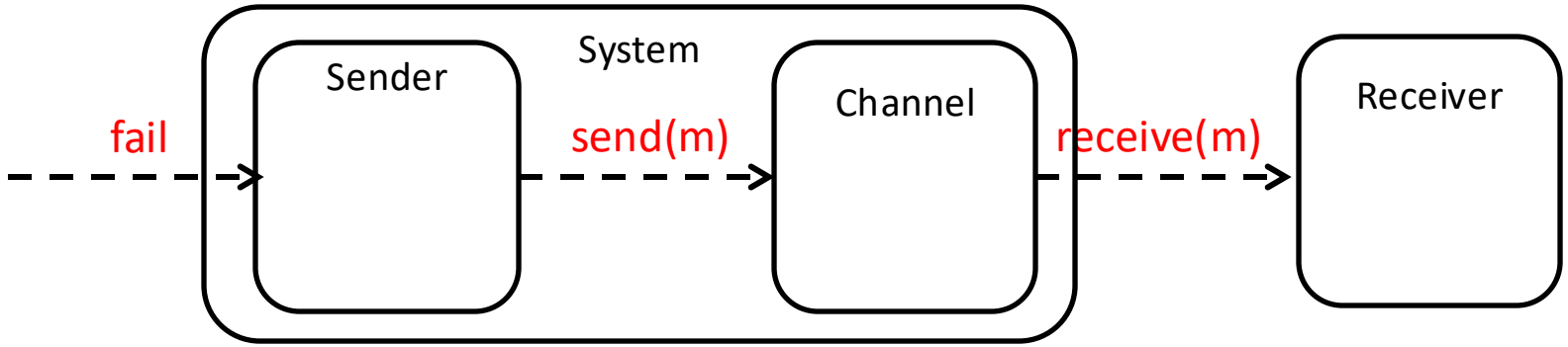
- $X = X_1 \cup X_2$
- $\Theta = \{\, \boldsymbol{x} \in val(X) \mid \forall\, i \in \{1,2\}\colon \boldsymbol{x}.X_{\boldsymbol{i}} \in \Theta_i\}$
- $H = H_1 \cup H_2$
- $O = O_1 \cup O_2$ $\}$ $A = H \cup O \cup I$
- $I = I_1 \cup I_2 \setminus O$
- $(\boldsymbol{x}, a, \boldsymbol{x}') \in \mathcal{D}$ *iff for* $i \in \{1,2\}$
  - $a \in A_i$ *and* $(\boldsymbol{x}.X_i, a, \boldsymbol{x}'.X) \in \mathcal{D}_i$
  - $a \notin A_i$ $\boldsymbol{x}.X_i = \boldsymbol{x}.X_i$

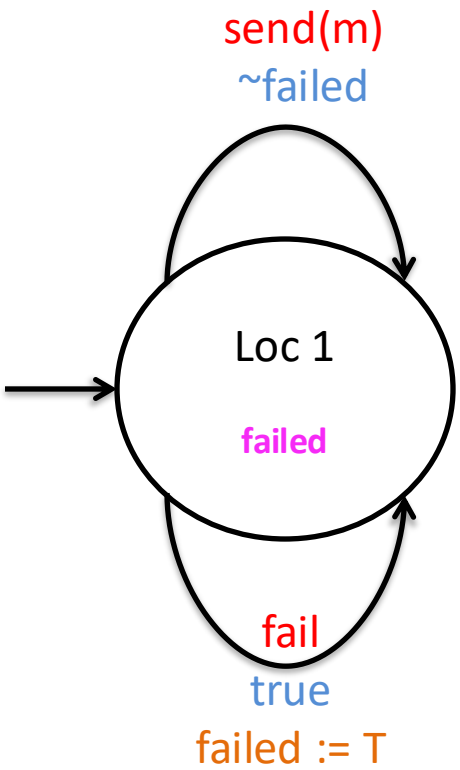
Theorem. The class of IO-automata is closed under composition. If $\mathcal{A}_1$ and $\mathcal{A}_2$ are compatible I/O automata then $\mathcal{A} = \mathcal{A}_1 || \mathcal{A}_2$ is also an I/O automaton.

Proof. Only 2 things to check

- Input, output, and internal actions are disjoint---by construction
- $\mathcal{A}$ satisfies **E1**. Consider any state $\boldsymbol{x} \in val(X_1 \cup X_2)$ and any input action a $\in I_1 \cup I_2 \setminus O$ such that a is enabled in $\boldsymbol{x}$.
- Suppose, w.lo.g. a $\in I_1$
  - We know by **E1** of $\mathcal{A}_1$ that there exists $\boldsymbol{x_1'} \in val(X_1)$ such that $\boldsymbol{x}.X_1 \rightarrow_a \boldsymbol{x_1'}$
  - $a \notin O_2, I_2, H_2$ (by compatibility)
- Therefore, $\boldsymbol{x} \rightarrow_a (\boldsymbol{x_1'}, x.X_2)$ is a valid transition of $\mathcal{A}$ (by definition of composition)

Example: Sending process and channel





Automaton Sender(u)
variables internal
failed:Boolean := F
output send(m:M)
input fail
transitions:
output send(m)
pre ~failed
eff
input fail
pre true
eff failed := T

FIFO channel & Simple Failure Detector

Automaton Sender(u)
variables internal
failed:Boolean := F
output send(m:M)
input fail
transitions:
output send(m)
pre ~failed
eff
input fail
pre true
eff failed := T

Automaton Channel(M)
variables internal queue: **Queue**[M] := {}
actions input send(m:M)
output receive(m:M)
transitions:
input send(m)
pre true
eff queue := append(m, queue)
output receive(m)
pre head(queue)=m
eff queue := queue.tail

Automaton System(M)
variables queue: **Queue**[M] := {}, failed: Bool
actions input fail
output send(m:M), receive(m:M)
transitions:
output send(m)
pre ~failed
eff queue := append(m, queue)
output receive(m)
pre head(queue)=m
eff queue := queue.tail
input fail
pre true
eff failed := true

Properties of Compositions

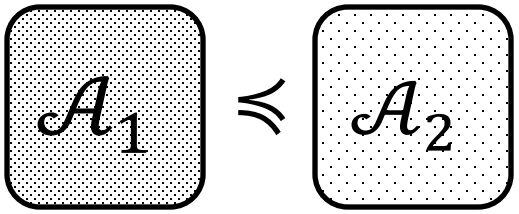
Proposition. Let $\mathcal{A} = \mathcal{A}_1 || \mathcal{A}_2$. α is an execution fragment of $\mathcal{A}$ iff

$\alpha \lceil (A_i, V_i), i \in \{1,2\}$ are both execution fragments of $\mathcal{A}_i$.

Proof of the forward direction. Fix α and i. We prove this by induction on the length of α.

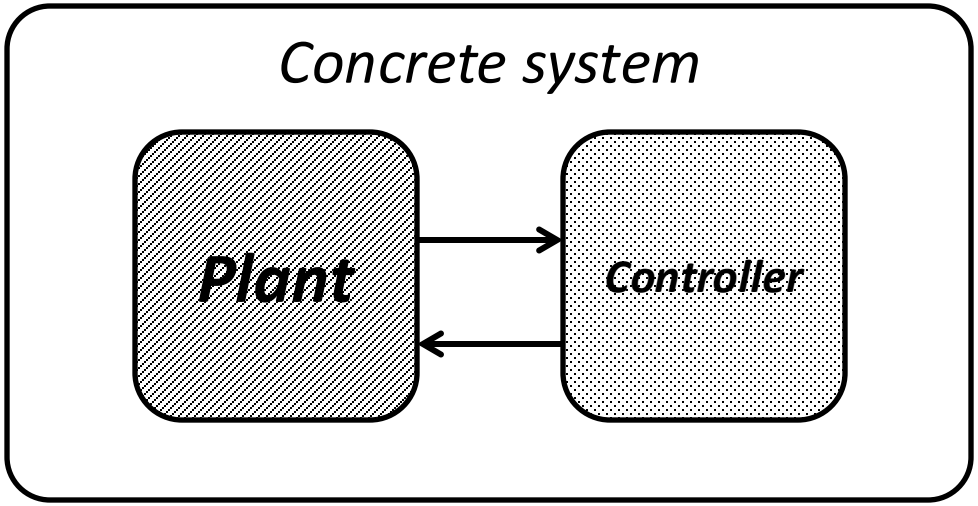
Composition and Abstraction

Abstraction recap

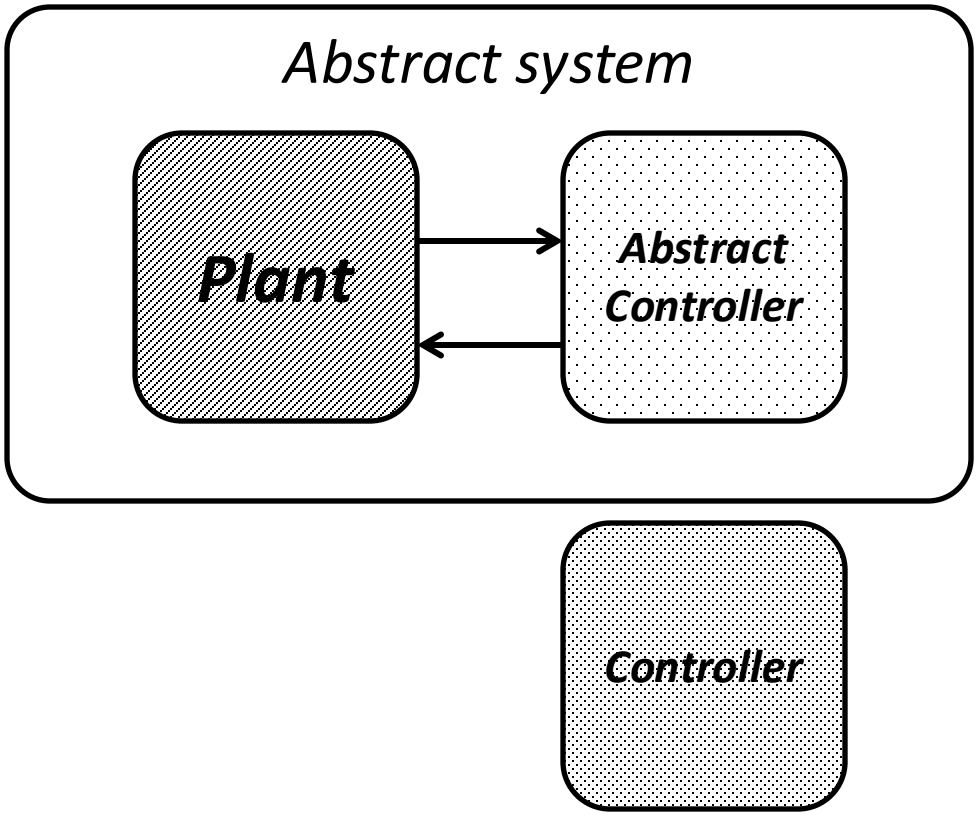


- Definition of abstraction: $\mathcal{A}_1 \preccurlyeq_{\mathrm{T}} \mathcal{A}_2 : \mathrm{Traces}_{\mathcal{A}_1} \subseteq \mathrm{Traces}_{\mathcal{A}_2}$
- $\mathcal{A}_1$ is called a refinement of $\mathcal{A}_2$; $\preccurlyeq$ is sometimes called a refinement or implementation relation
- If $\mathcal{A}_2$ satisfies some property P that is quantified over all traces/executions (e.g., invariants) then $\mathcal{A}_1$ also satisfies P
- Forward and backward simulation relations are sufficient for proving $\mathcal{A}_1 \preccurlyeq_{\mathrm{T}} \mathcal{A}_2$
- Consider the set of all automata $\mathcal{A}$ with the same Input/Output actions
  - We can compare some members of $\mathcal{A}$ because $\preccurlyeq_{\mathrm{T}}$ is transitive.
  - If $\mathcal{A}_1 \preccurlyeq_{\mathrm{T}} \mathcal{A}_2$ and $\mathcal{A}_2 \preccurlyeq_{\mathrm{T}} \mathcal{A}_3$ then $\mathcal{A}_1 \preccurlyeq_{\mathrm{T}} \mathcal{A}_3$
- $(\mathcal{A}, \preccurlyeq_{\mathrm{T}})$ defines a preorder
- CEGAR is a procedure for obtaining a refinement $\mathcal{A}_1$ from $\mathcal{A}_2$ such that some spurious behaviors in $\mathrm{Traces}_{\mathcal{A}_2}$ are eliminated in $\mathrm{Traces}_{\mathcal{A}_1}$

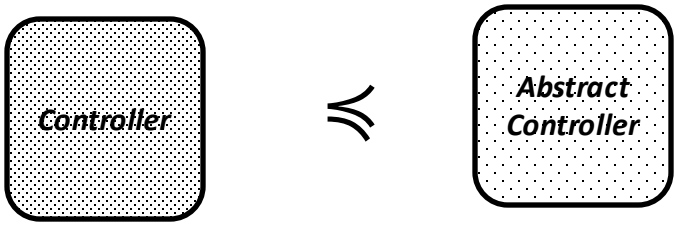
Q. How does composition work with abstraction?
Q. When can we substitute an automaton with its abstraction or its refinement?



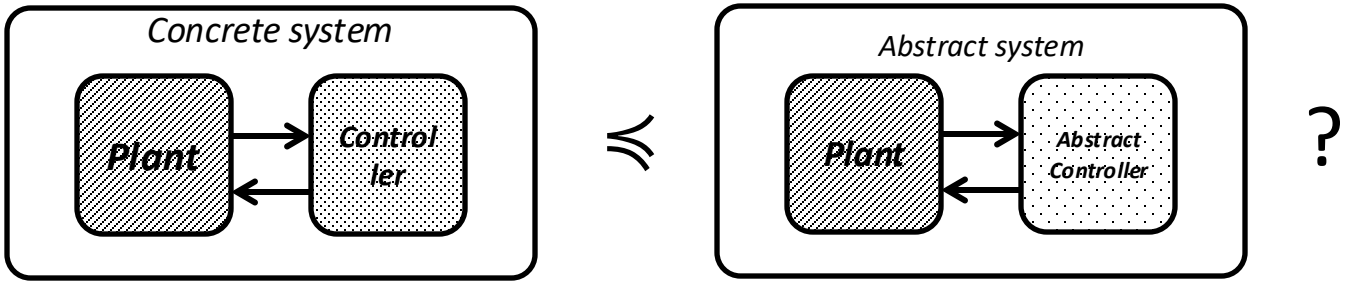
Substituting an automaton with its abstraction



How is the abstract system related to the concrete system?

Does 

imply



Substitutivity

Theorem 1. Suppose $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{B}$ have the same external interface and $\mathcal{A}_1$, $\mathcal{A}_2$ are compatible with $\mathcal{B}$. If $\mathcal{A}_1 \preccurlyeq \mathcal{A}_2$ then $\mathcal{A}_1 || \mathcal{B} \preccurlyeq \mathcal{A}_2 || \mathcal{B}$

Proof. Follows from the definition of composition and traces. Any enabled transition in $\mathcal{A}_1$ is also an enabled transition in $\mathcal{A}_2$.

Substutivity

Theorem 2. Suppose $\mathcal{A}_1$ $\mathcal{A}_2$ $\mathcal{B}_1$ and $\mathcal{B}_2$ are HAs and $\mathcal{A}_1$ $\mathcal{A}_2$ have the same external actions and $\mathcal{B}_1$ $\mathcal{B}_2$ have the same external actions and $\mathcal{A}_1$ $\mathcal{A}_2$ are compatible with each of $\mathcal{B}_1$ and $\mathcal{B}_2$.

If $\mathcal{A}_1 \preccurlyeq \mathcal{A}_2$ and $\mathcal{B}_1 \preccurlyeq \mathcal{B}_2$ then $\mathcal{A}_1 \,||\, \mathcal{B}_1 \preccurlyeq \mathcal{A}_2 || \mathcal{B}_2$.

- Proof. $\mathcal{A}_1 \,||\, \mathcal{B}_1 \preccurlyeq \mathcal{A}_2 || \mathcal{B}_1$

  $\mathcal{A}_2 || \mathcal{B}_1 \preccurlyeq \mathcal{A}_2 || \mathcal{B}_2$

  By transitivity of $\preccurlyeq$ relation

  $\mathcal{A}_1 \,||\, \mathcal{B}_1 \preccurlyeq \mathcal{A}_2 || \mathcal{B}_2$

A stronger substitutivity result

Theorem 3. $\mathcal{A}_1 \,||\, \mathcal{B}_2 \preccurlyeq \mathcal{A}_2 \,||\, \mathcal{B}_2$ and $\mathcal{B}_1 \preccurlyeq \mathcal{B}_2$ then $\mathcal{A}_1 \,||\, \mathcal{B}_1 \preccurlyeq \mathcal{A}_2 \,||\, \mathcal{B}_2$.

A natural trace theorem for compositions automata

Theorem 5.5 (from Theory of Timed I/O Automata by Lynch et. al.)

Suppose $\mathcal{A} = \mathcal{A}_1 || \mathcal{A}_2$ and let E, E_i be the sets of input/output actions of $\mathcal{A}, \mathcal{A}_i$. Then $\text{Traces}_{\mathcal{A}}$ is exactly the set of E-sequences whose restrictions to E_1 and E_2 are traces of $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively. That is,

$\text{Traces}_{\mathcal{A}}$ = {β | β is an E-sequence and $\beta \lceil E_i \in \text{Traces}_{\mathcal{A}_i}$, i ∈ {1, 2}}.

A stronger substitutivity result

Theorem 3. $\mathcal{A}_1 \,||\, \mathcal{B}_2 \preccurlyeq \mathcal{A}_2 || \mathcal{B}_2$ and $\mathcal{B}_1 \preccurlyeq \mathcal{B}_2$ then $\mathcal{A}_1 \,||\, \mathcal{B}_1 \preccurlyeq \mathcal{A}_2 || \mathcal{B}_2$.

Proof. Let β ∈ $\text{Traces}_{\mathcal{A}_1 \,||\, \mathcal{B}_1}$.
By Theorem 5.5, $\beta\lceil E_{A_1} \in Traces_{\mathcal{A}_1}$ and $\beta\lceil E_{B_1} \in Traces_{\mathcal{B}_1}$.
Since $\mathcal{B}_1 \preccurlyeq \mathcal{B}_2$

$\beta\lceil E_{B_2} \in Traces_{\mathcal{B}_2}$

By Theorem 5.5 , β ∈ $\text{Traces}_{\mathcal{A}_1 \,||\, \mathcal{B}_2}$
Since $\mathcal{A}_1 \,||\, \mathcal{B}_2 \preccurlyeq \mathcal{A}_2 || \mathcal{B}_2$ by assumption, β ∈ $\text{Traces}_{\mathcal{A}_2 \,||\, \mathcal{B}_2}$

properties of executions of composed automata

- α is an *execution* iff $\alpha\lceil(A_i, V_i), i \in \{1,2\}$ are both executions.
- α is *time bounded* iff $\alpha\lceil(A_i, V_i), i \in \{1,2\}$ are both time bounded.
- α is *admissible* iff $\alpha\lceil(A_i, V_i), i \in \{1,2\}$ are both admissible.
- α is *closed* iff $\alpha\lceil(A_i, V_i), i \in \{1,2\}$ are both closed.
- α is *non-Zeno* iff $\alpha\lceil(A_i, V_i), i \in \{1,2\}$ are both time non-Zeno.

Summary

- Composition operation
  - I/O interfaces: actions and variables
  - Reactivity/input enabling
  - (non) Closure under composition
- Properties of executions preserved under composition
- Inductive invariants