

Lecture 22: Composition of Hybrid Automata

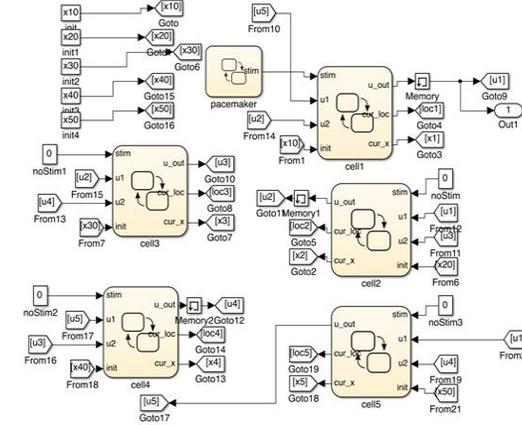
Huan Zhang
huan@huan-zhang.com

Be prepared for your final project presentation!

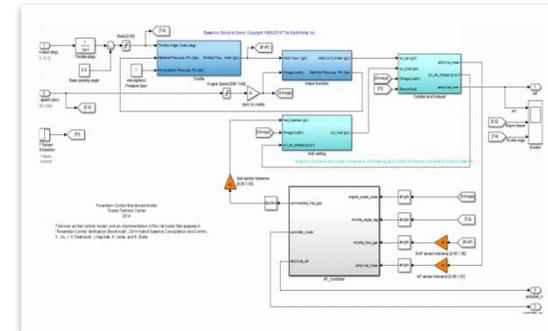
- Final project presentations: Week of **4/29** and **5/1**
- Same presentation schedule as the mid-term presentation:
 - People who presented mid-term on Tuesday will present on Thursday
 - People who presented mid-term on Thursday will present on Tuesday
 - Schedule will be posted on Canvas
- Attendance required, for each lecture, choose 2 projects that you are mostly interested in and submit two **feedback forms** to Canvas
- Feedback forms will be distributed to your peers to help them finalize their report
- No regular class on 5/6 (will become an office hour for final project)
- HW4 will be posted after HW3 due

What is composition?

- Complex models and systems are built by putting together **components** or **modules**
- **Composition** is the mathematical operation of *putting together*
- Leads to 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]

- Give an example of how you've built something more complex from simple components
- Throughout the lecture, think if your notion of composition is captured by what we define

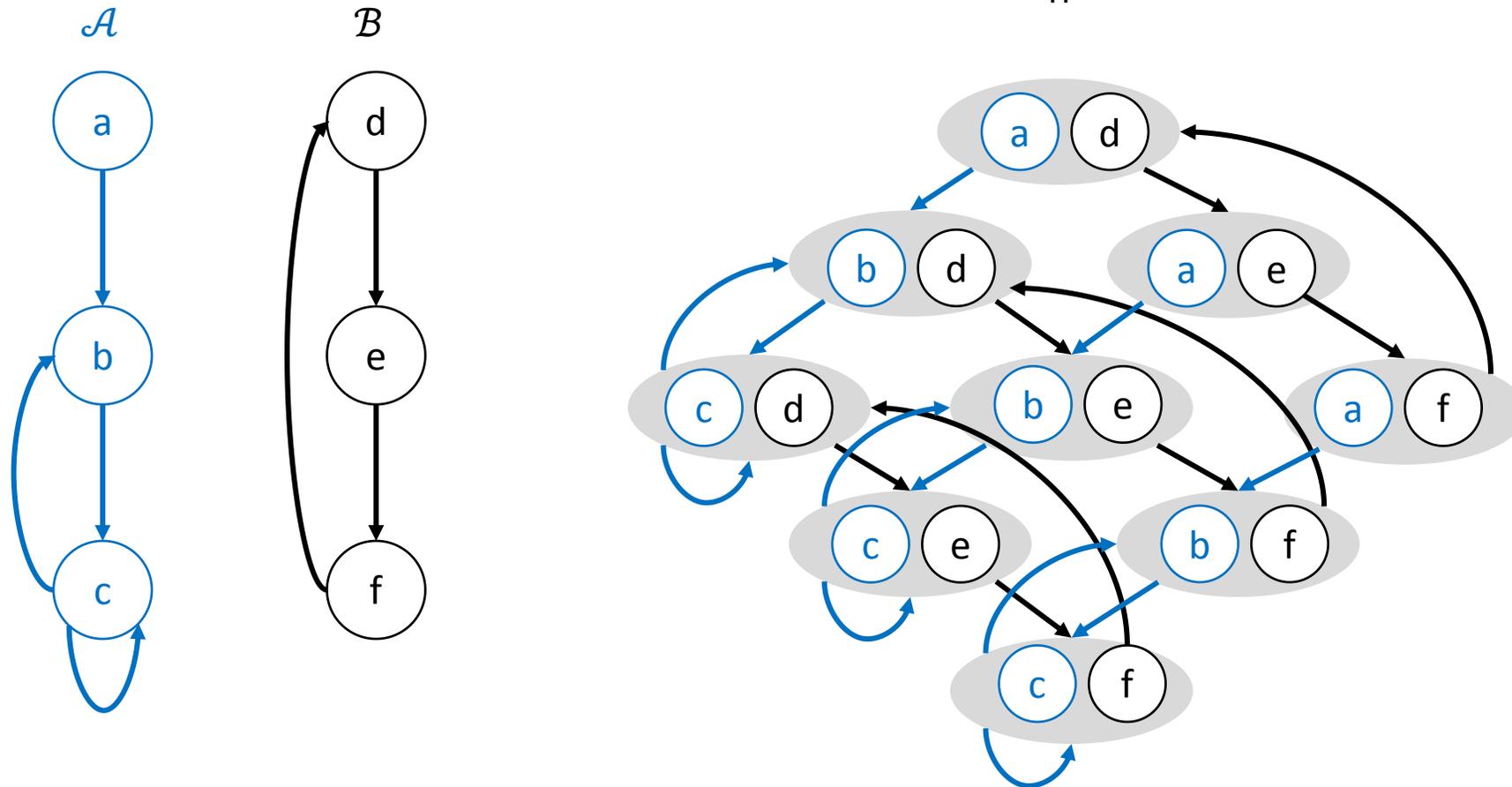
Outline

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

Composition of (discrete) automata

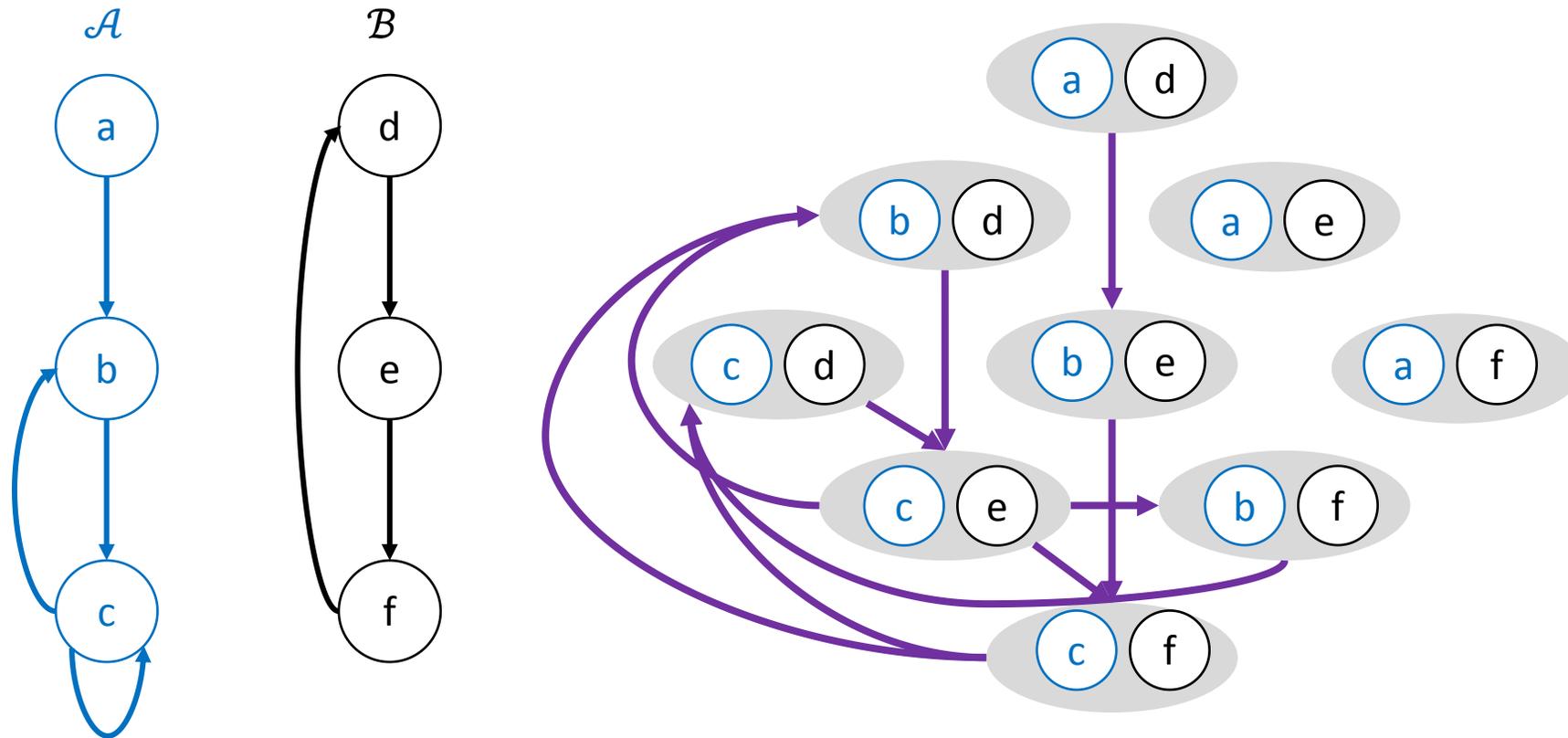
- Complex systems are built by “putting together” simpler subsystems
- Recall $\mathcal{A} = \langle X, \Theta, A, \mathbf{D} \rangle$
- $\mathcal{A} = \mathcal{A}_1 || \mathcal{A}_2$
 - $\mathcal{A}_1, \mathcal{A}_2$ are the *component automata* and
 - \mathcal{A} is the *composed* automaton
 - $||$ symbol for the composition operator

Composition: asynchronous modules



\mathcal{A} and \mathcal{B} each can change its state asynchronously

composition: modules synchronize



\mathcal{A} and \mathcal{B} must change their states at the same time, together

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 $\mathcal{A} = \langle X, \Theta, A, \mathbf{D} \rangle$ 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 listened-to or read by \mathcal{A}_2
- Input enabling ensures that when \mathcal{A}_1 and \mathcal{A}_2 are composed then \mathcal{A}_2 can react to **brakeOn**

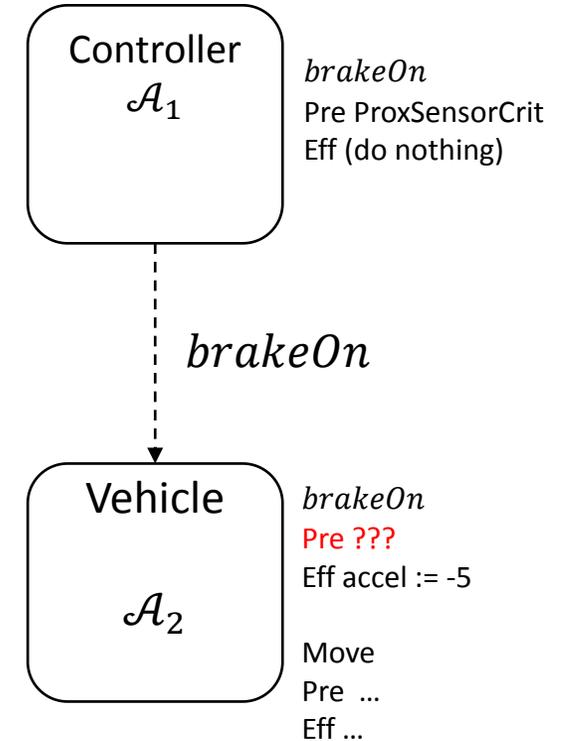
Definition. An **input/output automaton** is a tuple $\mathcal{A} = \langle X, \Theta, A, \mathcal{D} \rangle$ where

- X is a set of names of variables
- $\Theta \subseteq \text{val}(X)$ is the set of initial states
- $A = I \cup O \cup H$ is a set of names of actions
- $\mathcal{D} \subseteq \text{val}(X) \times A \times \text{val}(X)$ is the set of transitions and \mathcal{A} satisfies the **input enabling condition (E1)**:

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

E1 ensures that the transition is well defined for every input action at any state

(input may be received at any state, and the behavior must be defined)



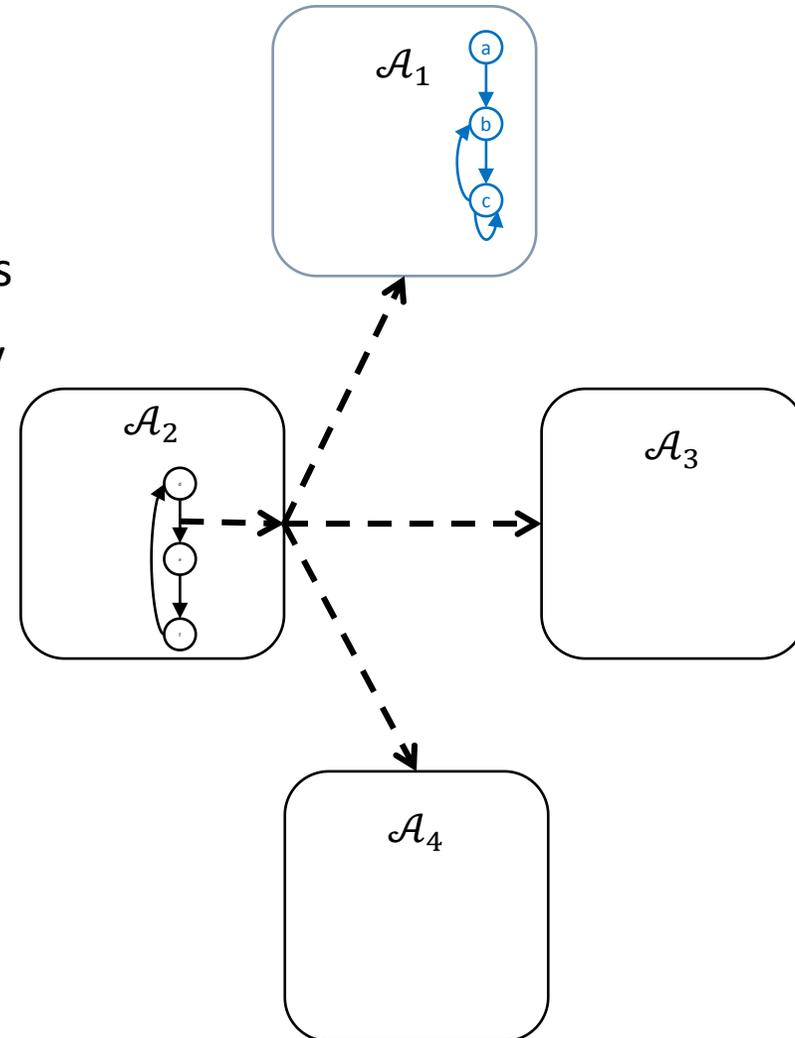
Compatibility IOA

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 duplication of authority

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 $\mathcal{A}_1 \parallel \mathcal{A}_2$ is the structure $\mathcal{A} = (X, \Theta, A, \mathcal{D})$

- $X = X_1 \cup X_2$
 - $\Theta = \{ \mathbf{x} \in \text{val}(X) \mid \forall i \in \{1,2\}: \mathbf{x}[X_i] \in \Theta_i \}$
 - $H = H_1 \cup H_2$
 - $O = O_1 \cup O_2$
 - $I = I_1 \cup I_2 \setminus O$
- $A = H \cup O \cup I$
- Once an input is used by an output, it must be removed to avoid duplication of authority
- $(\mathbf{x}, a, \mathbf{x}') \in \mathcal{D}$ iff for $i \in \{1,2\}$, either:
 - $a \in A_i$ and $(\mathbf{x}[X_i], a, \mathbf{x}'[X_i]) \in \mathcal{D}_i$
 - $a \notin A_i$ and $\mathbf{x}[X_i] = \mathbf{x}'[X_i]$
- Once an input is used by an output, it must be removed to avoid duplication of authority

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

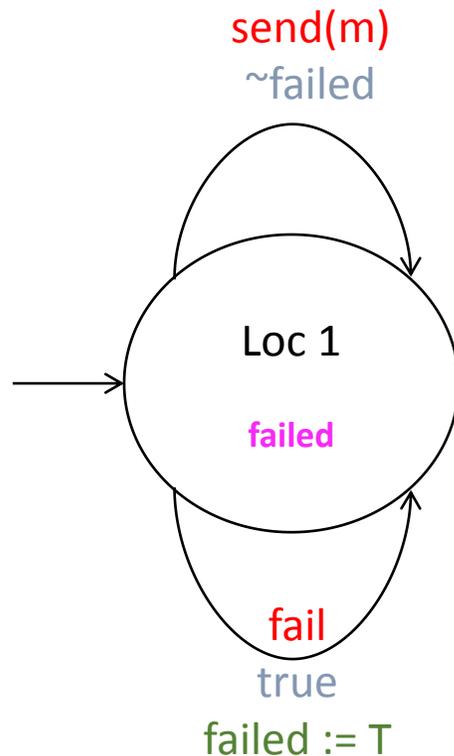
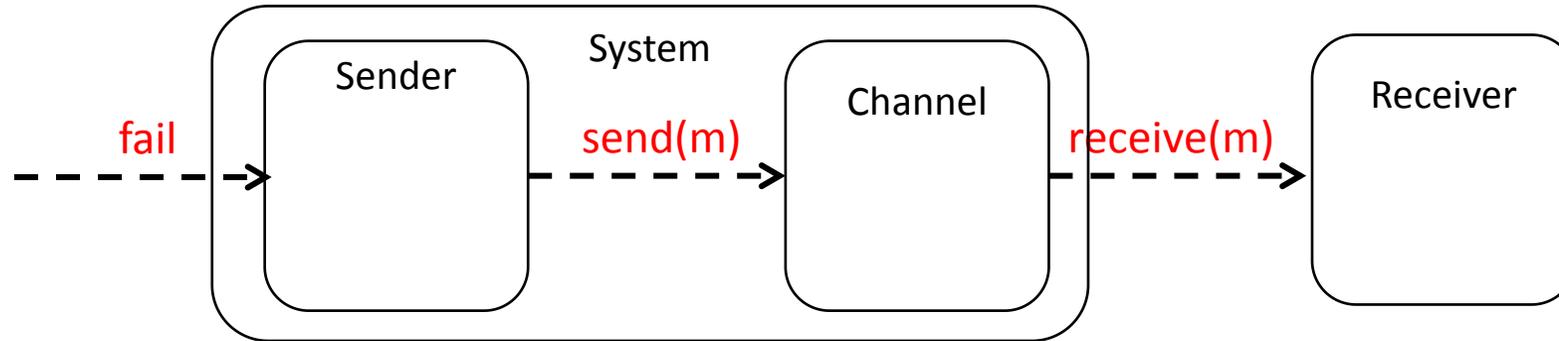
Proof. Only 2 things to check

- (1) Input, output, and internal actions are disjoint---by construction
- (2) \mathcal{A} satisfies **E1**. (ensures that the transition is well defined for every input action at any state)

Consider any state $x \in \text{val}(X_1 \cup X_2)$ and any input action $a \in I_1 \cup I_2 \setminus O$ such that a is enabled in x .

- Suppose, w.l.o.g. $a \in I_1$
- We know by **E1** of \mathcal{A}_1 that there exists $x'_1 \in \text{val}(X_1)$ such that $x[X_1 \rightarrow_a x'_1$
- $a \notin O_2, I_2, H_2$ (by compatibility)
- Therefore, $x \rightarrow_a (x'_1, x[X_2])$ 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

Does this automaton satisfy **input enabling condition (E1)**?

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

composing hybrid systems

Hybrid IO Automaton

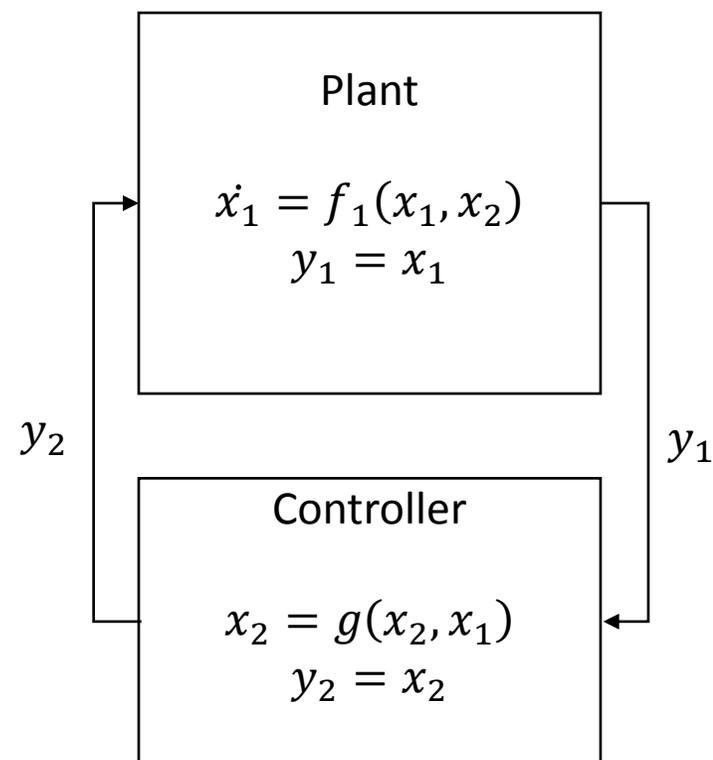
In addition to interaction through shared actions hybrid input/output automata (HIOA) will allow interaction through **shared variables**

Recall a hybrid automaton $\mathcal{A} = \langle V, \Theta, A, \mathbf{D}, \mathbf{T} \rangle$

We will partition the set of variables V of \mathcal{A} into

- X : **internal or state variables** (do not interact)
 - Y : **output** variables
 - U : **input** variables
- $V = X \cup Y \cup U$

This gives rise to **hybrid I/O automata (HIOA)** [Lynch, Segala, Vaandrager 2002]



Reactivity: Input trajectory enabling

Consider a shared variable **throttle** controlled by \mathcal{A}_1 and listened-to or read by \mathcal{A}_2

Input trajectory enabling ensures that when \mathcal{A}_1 and \mathcal{A}_2 are composed then \mathcal{A}_2 can react to any signal generated by \mathcal{A}_1

If the trajectories of \mathcal{A}_2 are defined by ordinary differential equations, then input enabling is guaranteed if \mathcal{A}_1 only generates piece-wise continuous signals (throttle)

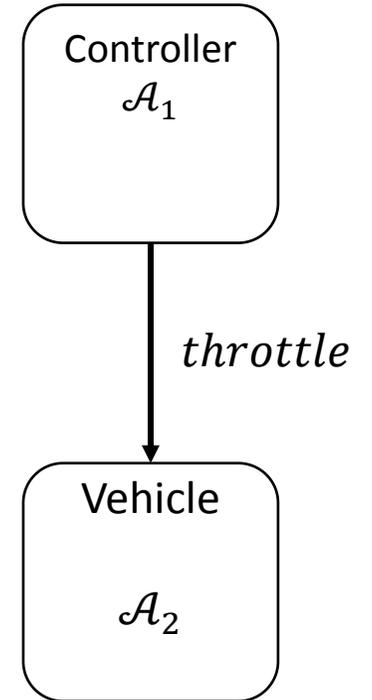
Definition. An **hybrid input/output automaton** is a tuple $\mathcal{A} = \langle V, \Theta, A, \mathcal{D}, \mathcal{T} \rangle$ where

- $V = X \cup U \cup Y$ is a set of variables
- $\Theta \subseteq \text{val}(X)$ is the set of initial states
- $A = I \cup O \cup H$ is a set of actions
- $\mathcal{D} \subseteq \text{val}(X) \times A \times \text{val}(X)$ is the set of transitions
- \mathcal{T} is a set of trajectories for V closed under prefix, suffix, and concatenation

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

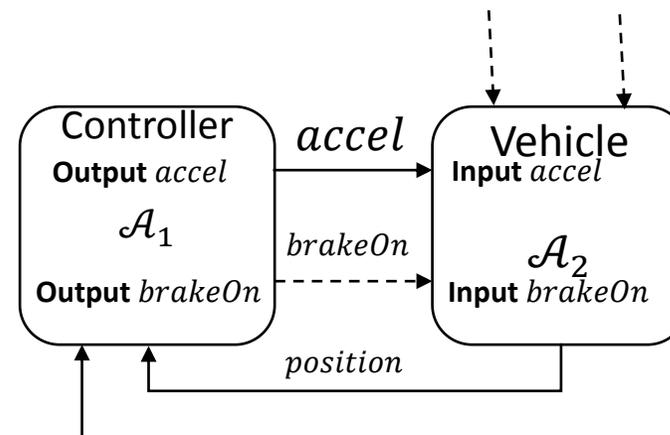
E2. For each $x \in \text{val}(X)$, \mathcal{A} should be able to react to any trajectory η of input variables U .

i.e., $\exists \tau \in \mathcal{T}$ with $\tau.fstate = x$ such that $\tau \downarrow U$ is a prefix of η , and either (a) $\tau \downarrow U = \eta$ or (b) τ is closed and some $a \in H \cup O$ is enabled at $\tau.lstate$. (**the HA cannot restrict its input trajectories**)



Compatibility of hybrid automata

- For the interaction of hybrid automata \mathcal{A}_1 and \mathcal{A}_2 to be well-defined we need to ensure that they have the right *interfaces*
- compatibility conditions



Compatibility HIOA

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

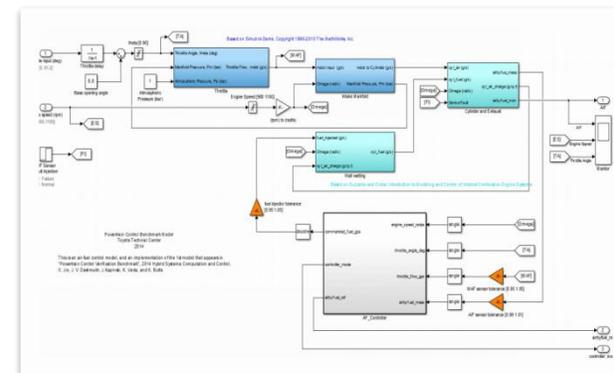
$H_i \cap A_j = \emptyset$ no unintended discrete interactions

$O_i \cap O_j = \emptyset$ no duplication of discrete authority

$X_i \cap V_j = \emptyset$ no unintended continuous interactions

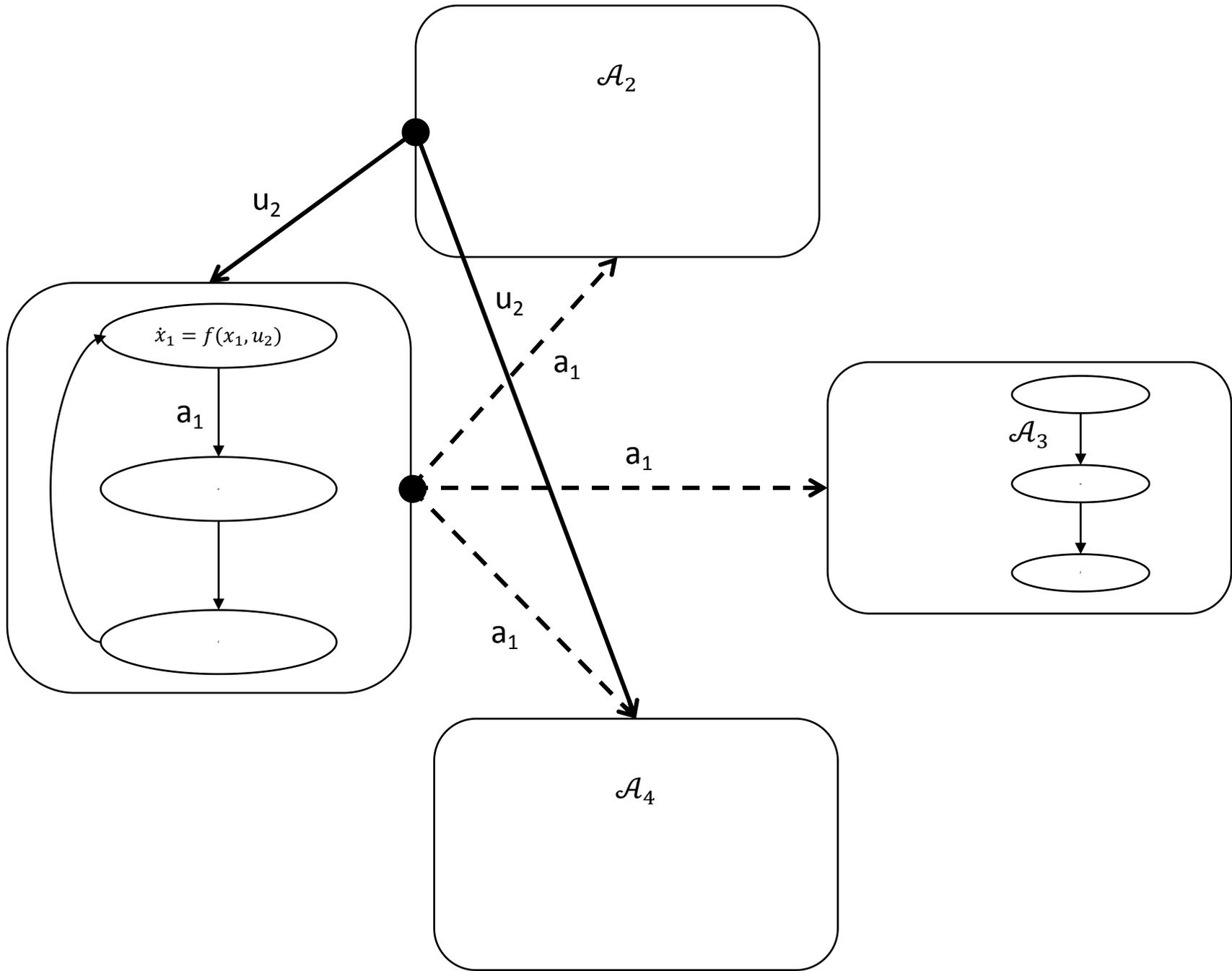
$Y_i \cap Y_j = \emptyset$ no duplication of continuous authority

Extended to collection of automata in the natural way and captures most common notions of composition in, for example, Matlab/Simulink



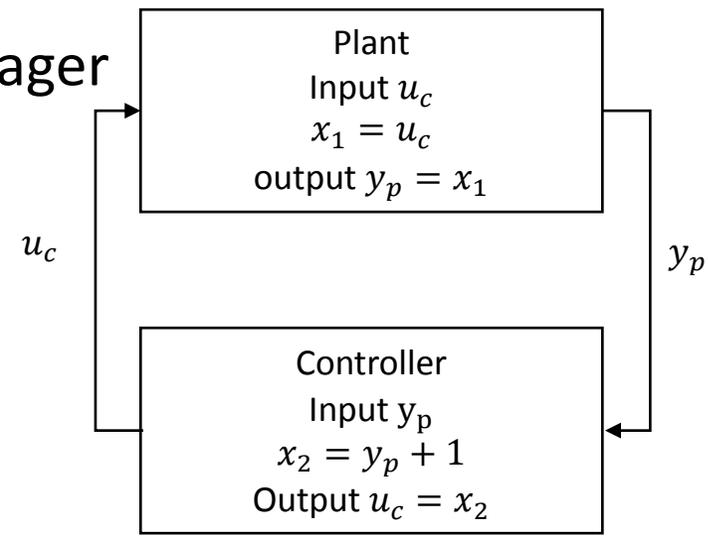
Composition

- For compatible \mathcal{A}_1 and \mathcal{A}_2 their composition $\mathcal{A}_1 \parallel \mathcal{A}_2$ is the structure $\mathcal{A} = (V, \Theta, A, \mathcal{D}, \mathcal{T})$
- Variables $V = X \cup Y \cup U$
 - $X = X_1 \cup X_2, Y = Y_1 \cup Y_2, U = U_1 \cup U_2 \setminus Y$
- $\Theta = \{ \mathbf{x} \in \text{val}(X) \mid \forall i \in \{1,2\}: \mathbf{x}[X_i] \in \Theta_i \}$
- Actions $A = H \cup O \cup I$
 - $H = H_1 \cup H_2, O = O_1 \cup O_2, I = I_1 \cup I_2 \setminus O,$
- $(\mathbf{x}, a, \mathbf{x}') \in \mathcal{D}$ iff for $i \in \{1,2\}$, either
 - $a \in A_i$ and $(\mathbf{x}[X_i], a, \mathbf{x}'[X_i]) \in \mathcal{D}_i$
 - $a \notin A_i$ and $\mathbf{x}[X_i] = \mathbf{x}'[X_i]$
- \mathcal{T} : set of **trajectories** for V
 - $\tau \in \mathcal{T}$ iff $\forall i \in \{1,2\}, \tau \downarrow V_i \in \mathcal{T}_i$

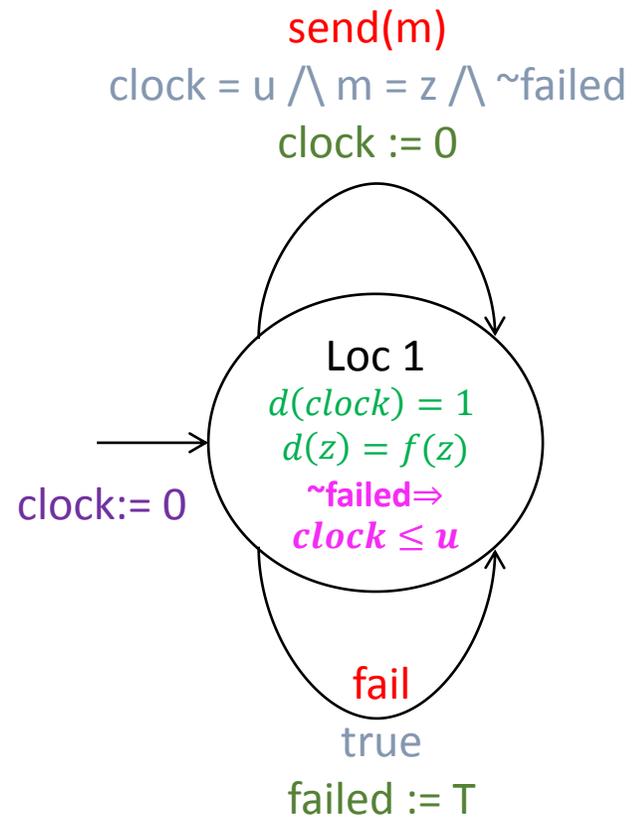


Closure under composition?

- Conjecture. The class of HIOA is closed under composition. If \mathcal{A}_1 and \mathcal{A}_2 are compatible HIOA then $\mathcal{A}_1 || \mathcal{A}_2$ is also a HIOA.
- Can we ensure that input trajectory enabled condition is satisfied in the composed automaton?
- **No, in general (E2 does not always satisfy)**
 - See "Hybrid I/O automata",
by Nancy Lynch, Roberto Segala, Frits Vaandrager



Example 2: Periodically Sending Process



Automaton PeriodicSend(u)

variables internal

clock: Reals := 0, z: Reals, failed: Boolean := F

signature output send(m: Reals)

input fail

transitions:

output send(m)

pre clock = u \wedge m = z \wedge \sim failed

eff clock := 0

input fail

pre true

eff failed := T

trajectories:

evolve $d(\text{clock}) = 1, d(z) = f(z)$

invariant failed \vee clock \leq u

Time bounded channel & Simple Failure Detector

Automaton Timeout(u, M)

variables internal suspected: Boolean := F,
clock: Reals := 0

signature input receive($m:M$)

output timeout

transitions:

input receive(m)

pre true

eff clock := 0; suspected := false;

output timeout

pre \sim suspected \wedge clock = u

eff suspected := true

trajectories:

evolve $d(\text{clock}) = 1$

invariant $\text{clock} \leq u \vee \text{suspected}$

Automaton Channel(b, M)

variables internal queue: Queue[M, Reals] := {}
clock: Reals := 0

signature input send($m:M$)

output receive($m:M$)

transitions:

input send(m)

pre true

eff queue := append($\langle m, \text{clock} + b \rangle$, queue)

output receive(m)

pre head(queue)[1] = $m \wedge$

head(queue)[2] = clock

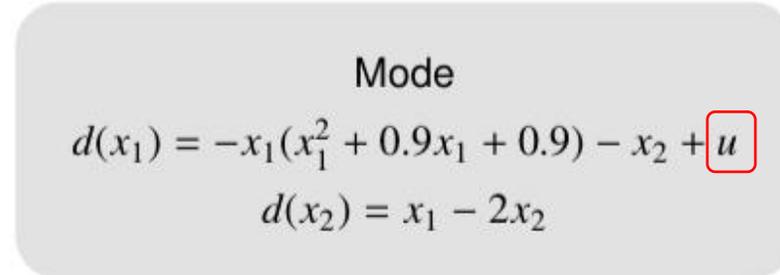
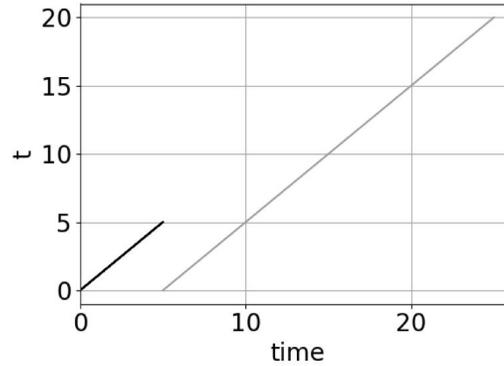
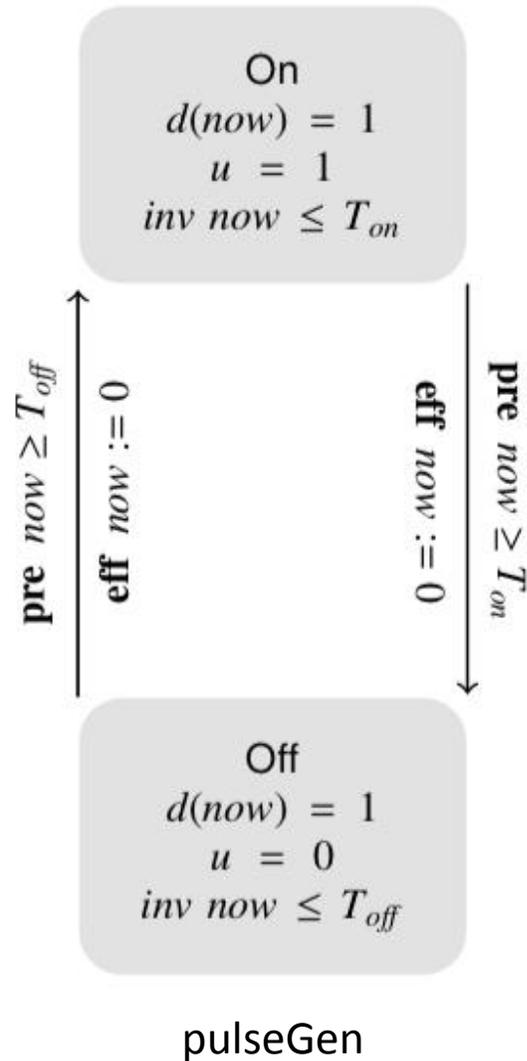
eff queue := queue.tail

trajectories:

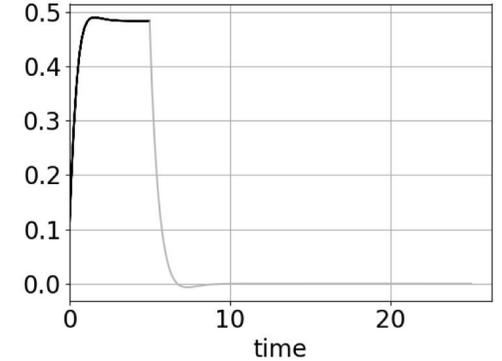
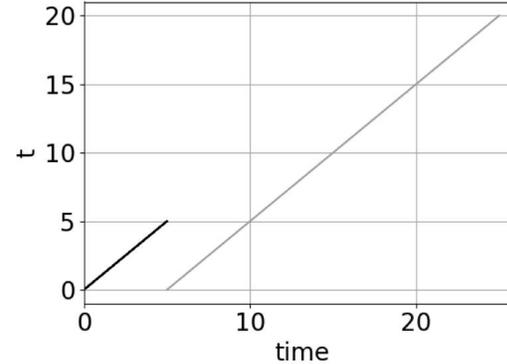
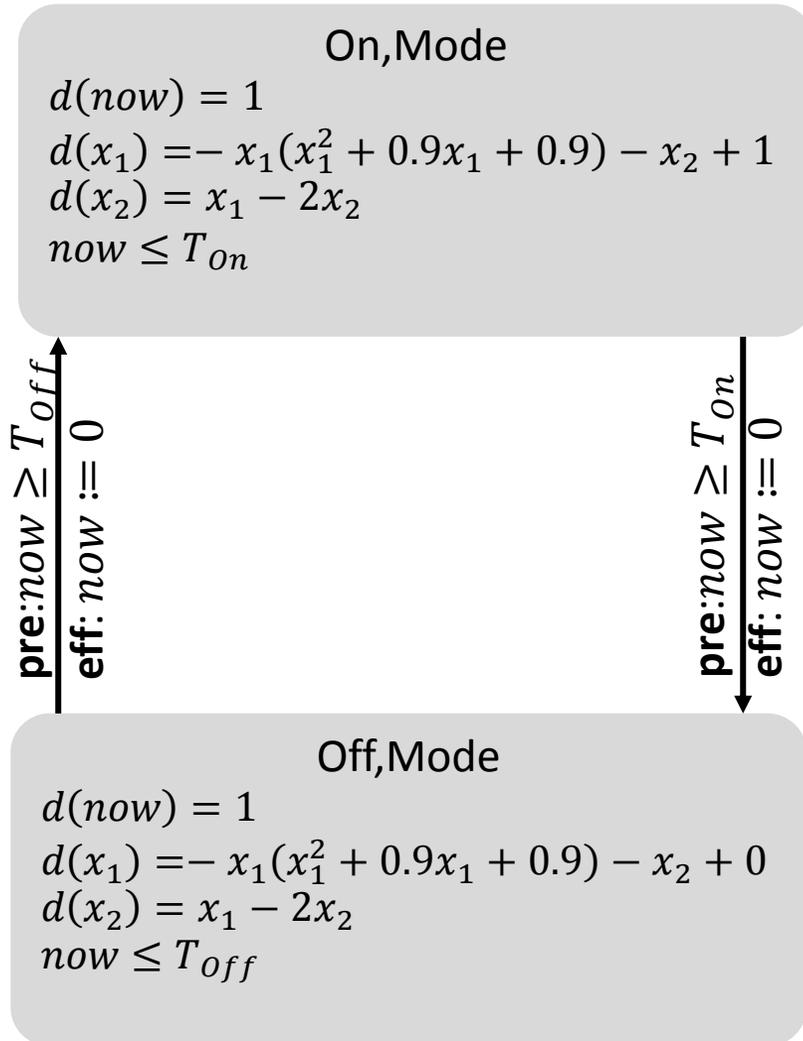
evolve $d(\text{clock}) = 1$

invariant $\forall \langle m, d \rangle \in \text{queue}: d \geq \text{clock}$

Example 3: Oscillator and pulse generator



Composed automaton



Restriction operation on executions

- Sometimes it is useful to restrict our attention to only some subset of variables and actions in an execution
- Recall the **restriction** operations $x[V]$ and $\tau \downarrow V$
- Let $\alpha = \tau_0 a_1 \tau_1 a_2$ be an execution fragment of a hybrid automaton with set of variables V and set of actions A . Let A' be a set of actions and V' be a set of variables.
- **Restriction** of α to (A', V') , written as $\alpha[(A', V')]$ is the sequence defined inductively as:
 - $\alpha[(A', V')] = \tau \downarrow V'$ if $\alpha = \tau$
 - $\alpha a \tau [(A', V')] =$
 - $\alpha [(A', V')] a (\tau \downarrow V')$ if $a \in A'$
 - $\alpha [(A', V')] \text{ concat } (\tau \downarrow V')$ if $a \notin A'$
- From the definition it follows $\alpha.lstate[V'] = \alpha[(A', V').lstate$ for any A', V'

Properties of Compositions

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

$\alpha[(A_i, V_i), i \in \{1,2\}]$ 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 .
- Base case: $\alpha = \tau$. $\alpha[(A_i, V_i)] = \tau \downarrow V_i$ by definition of composition $\tau \downarrow V_i \in T_i$. So, $\tau \downarrow V_i \in Frag_i$
- $\alpha = \alpha' a \tau [(A_i, V_i)]$ and $a \in A_i$ and by induction hypothesis $\alpha'[(A_i, V_i)] \in Frag_i$. Let $\alpha'[(A_i, V_i)].lstate = v$. By the definition of composition: $\tau \downarrow V_i \in T_i$.
 - It remains to show that $v[V_i \rightarrow_a (\tau \downarrow V_i)].fstate$. Since $a \in A_i$, by the definition of composition: $\alpha'[(A_i, V_i)].lstate \rightarrow_a \tau \downarrow V_i.fstate$
- $\alpha = \alpha' a \tau [(A_i, V_i)]$ and $a \notin A_i$ and by induction hypothesis $\alpha'[(A_i, V_i)] \in Exec_i$. Let τ' be the last trajectory in that execution.
 - Since $a \notin A_i$, by the definition of composition: $\tau'.lstate = \tau \downarrow V_i.fstate$. By concatenation closure of T_i , it follows that $\tau' concat \tau \downarrow V_i \in T_i$. Therefore $\alpha[(A_i, V_i)] \in Exec_i$.

properties of executions of composed automata

- α is an *execution* iff $\alpha[(A_i, V_i), i \in \{1,2\}]$ are both executions.
- α is *time bounded* iff $\alpha[(A_i, V_i), i \in \{1,2\}]$ are both time bounded.
- α is *admissible* (infinite duration) iff $\alpha[(A_i, V_i), i \in \{1,2\}]$ are both admissible.
- α is *closed* (finite time with final trajectory in closed domain $[0, T]$) iff $\alpha[(A_i, V_i), i \in \{1,2\}]$ are both closed.
- α is *non-Zeno* iff $\alpha[(A_i, V_i), i \in \{1,2\}]$ 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