Lecture 22: Composition of Hybrid Automata

Huan Zhang
huan@huan-zhang.com

Slides adapted from Prof. Sayan Mitra’s slides in Fall 2021
Homework and final presentations

HW 3 due **4/27**

HW 4 due **5/11**

Final project presentation slides due **4/30, 8 am** (hard deadline, since presentations will start at 11 am)

Final project presentations:
- **Tuesday 11 am - 12:20 pm**, ECEB 3015 (lecture time)
- **Friday 2 pm - 3:30 pm**, ECEB 2015

Schedule will be announced by the end of this week. If you cannot present on Friday, please let me and the TA Sanil (schawla7@illinois.edu) know by Thursday (4/25)

Final project report due: **5/11**
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 $\mathcal{A} = \langle X, \Theta, A, D \rangle$
• $\mathcal{A} = \mathcal{A}_1 \parallel \mathcal{A}_2$
  • $\mathcal{A}_1$, $\mathcal{A}_2$ are the component automata and
  • $\mathcal{A}$ is the composed automaton
  • $\parallel$ symbol for the composition operator
Composition: asynchronous modules

\[ A \]
- a
- b
- c

\[ B \]
- d
- e
- f

\[ A || B \]
- a \rightarrow d
- b \rightarrow d
- b \rightarrow e
- b \rightarrow f
- c \rightarrow d
- c \rightarrow e
- c \rightarrow f
- d \rightarrow a
- d \rightarrow e
- d \rightarrow f
- e \rightarrow a
- e \rightarrow b
- e \rightarrow f
- f \rightarrow a
- f \rightarrow b
- f \rightarrow c
composition: modules synchronize
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 $\mathcal{A} = \langle X, \emptyset, A, D \rangle$ 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 \texttt{brakeOn} controlled by $A_1$ and listend-to or read by $A_2$
- Input enabling ensures that when $A_1$ and $A_2$ are composed then $A_2$ can react to \texttt{brakeOn}

\textbf{Definition}. An \textit{input/output automaton} is a tuple $A = \langle X, \Theta, A, D \rangle$ 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
  - $D \subseteq val(X) \times A \times val(X)$ is the set of transitions and $A$ satisfies the input enabling condition (E1):

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

\textbf{E1} ensures that the transition is well defined for every input action at any state
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 $\mathcal{A} = (X, \Theta, A, D)$

- $X = X_1 \cup X_2$
- $\Theta = \{ x \in \text{val}(X) | \forall i \in \{1,2\}: x_i \in \Theta_i \}$
- $H = H_1 \cup H_2$
- $O = O_1 \cup O_2$
- $I = I_1 \cup I_2 \setminus O$
- $(x, a, x') \in D$ iff for $i \in \{1,2\}$
  - $a \in A_i$ and $(x_i, a, x'[X_i]) \in D_i$
  - $a \notin A_i \ x_i = x_i$
**Theorem.** The class of IO-automata is closed under composition. If $\mathcal{A}_1$ and $\mathcal{A}_2$ are compatible I/O automata then $\mathcal{A} = \mathcal{A}_1 \mid \mathcal{A}_2$ 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 $E1$. Consider any state $x \in val(X_1 \cup X_2)$ and any input action $a \in I_1 \cup I_2 \setminus O$ such that $a$ is enabled in $x$.

- Suppose, w.l.o.g. $a \in I_1$

- We know by $E1$ of $\mathcal{A}_1$ that there exists $x'_1 \in val(X_1)$ 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)
prefailed

input fail
prefalse

efffailed := T

Loc 1

send(m)

~failed

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

**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 := T
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 \( \mathcal{A} = \langle V, \Theta, A, D, T \rangle \)

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 \texttt{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 \textit{hybrid input/output automaton} is a tuple \(\mathcal{A} = \langle V, \Theta, A, \mathcal{D}, T \rangle\) 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
- \(T\) is a set of trajectories for \(V\) closed under prefix, suffix, and concatenation.

**E1.** For each \(x \in \text{val}(X),\ \alpha \in I\) there exists \(x' \in \text{val}(X)\) such that \(x \xrightarrow{\alpha} x'\).

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

i.e, \(\exists \tau \in T\) with \(\tau.f\ \text{state} = x\) such that \(\tau \downarrow U\) is a prefix of \(\eta\), and either (a) \(\tau \downarrow U = \eta\) or (b) \(\tau\) is closed and some \(\alpha \in H \cup O\) is enabled at \(\tau.l\text{state}\). \text{(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 $\mathcal{A} = (V, \Theta, A, D, \mathcal{T})$.

- Variables $V = X \cup Y \cup U$
  - $X = X_1 \cup X_2$, $Y = Y_1 \cup Y_2$, $U = U_1 \cup U_2 \setminus Y$

- $\Theta = \{ x \in \text{val}(X) \mid \forall i \in \{1,2\}: x[X_i \in \Theta_i] \}$

- Actions $A = H \cup O \cup I$
  - $H = H_1 \cup H_2$, $O = O_1 \cup O_2$, $I = I_1 \cup I_2 \setminus O$

- $(x, a, x') \in D$ iff $i \in \{1,2\}$
  - $a \in A_i$ and $(x[X_i, a, x'][X_i]) \in D_i$
  - $a \not\in A_i$ $x[X_i] = x[X_i]$

- $\mathcal{T}$: set of trajectories for $V$
  - $\tau \in \mathcal{T}$ iff $\forall i \in \{1,2\}$, $\tau \downarrow V_i \in \mathcal{T}_i$
\[ x_1 = f(x_1, u_2) \]
Closure under composition?

• Conjecture. The class of HIOA is closed under composition. If $\mathcal{A}_1$ and $\mathcal{A}_2$ are compatible HIOA then $\mathcal{A}_1 \parallel \mathcal{A}_2$ 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 \ m = z \ ~failed
  - eff clock := 0
  - input fail
  - pre true
  - eff failed := T

- **trajectories:**
  - evolve d(clock) = 1, d(z) = f(z)
  - invariant failed \ clock \leq u
**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** $\neg$suspected $\land$ clock = $u$
  - **eff** suspected := true

**trajectories:**
- evolve $d(c) = 1$
- invariant clock $\leq u \lor$ suspected

**Automaton** Channel($b,M$)

- **variables internal** queue: Queue[$M,\text{Reals}$] := {}
clock: Reals := 0
- **signature input** send($m:M$)
- **output** receive($m:M$)

**transitions:**
- **input** send($m$)
  - **pre** true
  - **eff** queue := append($<m,clock+b>$, queue)
- **output** receive($m$)
  - **pre** head(queue)[1]=m $\land$
    head(queue)[2]=clock
  - **eff** queue := queue.tail

**trajectories:**
- evolve $d(c) = 1$
- invariant $\forall<m,d>\in$ queue: $d \geq clock$
Example 3: Oscillator and pulse generator

**Oscillator**

- On
  - \( d(\text{now}) = 1 \)
  - \( u = 1 \)
  - \( \text{inv now} \leq T_{\text{on}} \)

- Off
  - \( d(\text{now}) = 1 \)
  - \( u = 0 \)
  - \( \text{inv now} \leq T_{\text{off}} \)

**pulsGen**

- pre now \( \geq T_{\text{off}} \)
- eff now := 0
- eff now := 0
- pre now \( \geq T_{\text{on}} \)

**Mode**

\[
\begin{align*}
    d(x_1) &= -x_1(x_1^2 + 0.9x_1 + 0.9) - x_2 + u \\
    d(x_2) &= x_1 - 2x_2
\end{align*}
\]
Composed automaton

\[ \text{On,Mode} \]
\[
\begin{align*}
d(\text{now}) &= 1 \\
d(x_1) &= -x_1(x_1^2 + 0.9x_1 + 0.9) - x_2 + 1 \\
d(x_2) &= x_1 - 2x_2 \\
\text{now} &\leq T_{\text{on}}
\end{align*}
\]

\[ \text{Off,Mode} \]
\[
\begin{align*}
d(\text{now}) &= 1 \\
d(x_1) &= -x_1(x_1^2 + 0.9x_1 + 0.9) - x_2 + 0 \\
d(x_2) &= x_1 - 2x_2 \\
\text{now} &\leq T_{\text{off}}
\end{align*}
\]
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 $\alpha = \tau_0a_1\tau_1a_2$ 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 $\alpha$ 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 $\alpha.\text{lstate}[V] = \alpha[(A', V')].\text{lstate}$ for any $A', V'$.
Properties of Compositions

**Proposition.** Let $\mathcal{A} = \mathcal{A}_1 \parallel \mathcal{A}_2$. $\alpha$ 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 $\alpha$ and $i$. We prove this by induction on the length of $\alpha$.

- 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, $\tau \downarrow V_i \in Frag_i$

- $\alpha = \alpha' \ \tau \ [(A_i, V_i)$ 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

- $\alpha = \alpha' \ \tau \ [(A_i, V_i)$ and $a \notin A_i$ and by induction hypothesis $\alpha'[A_i, V_i) \in Exec_i$. Let $\tau'$ 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

• $\alpha$ is an execution iff $\alpha[(A_i, V_i), i \in \{1,2\}]$ are both executions.

• $\alpha$ is time bounded iff $\alpha[(A_i, V_i), i \in \{1,2\}]$ are both time bounded.

• $\alpha$ is admissible (infinite duration) iff $\alpha[(A_i, V_i), i \in \{1,2\}]$ are both admissible.

• $\alpha$ is closed (finite time with final trajectory) iff $\alpha[(A_i, V_i), i \in \{1,2\}]$ are both closed.

• $\alpha$ 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