CIS 4930/6930: Principles of Cyber-Physical Systems
Chapter 5: Composition of State Machines

Hao Zheng

Department of Computer Science and Engineering
University of South Florida
State machines are useful for modeling system behaviors.

How to represent a system for systematic analysis?

Complete systems though often have a very large state space.

Can represent complicated system as composition of simpler systems.
  - Modular approaches are always needed to handle large complex problems.

Care must be taken though as the same syntax (model notation) often has different semantics (meaning).
Actor Model and Extended SM Notation

- \( i_1 \) to \( o_1 \)
- \( i_n \) to \( o_m \)
- Variable declaration(s)
- Input declaration(s)
- Output declaration(s)

Guard / output action
Set action

Initial set action
Guard / output action
Set action

State 1
State 2
Overview

- Side-by-side **synchronous** composition (simultaneous reactions).
- Side-by-side **asynchronous** composition (independent reactions).
- Communication through shared variables.
- Cascade (serial) composition.
- General composition that combines side-by-side and cascade.
- Hierarchical state machines.
Side-by-side Composition

\[ A \]
\[ i_1 \rightarrow \quad o_1 \]

\[ B \]
\[ i_2 \rightarrow \quad o_2 \]

\[ C \]
\[ i_1 \rightarrow \quad o_1 \]
\[ i_2 \rightarrow \quad o_2 \]
**Side-by-side Composition Example**

**outputs:** $a, b : \text{pure}$

**output:** $a : \text{pure}$

```
  s1 --true / a-- s2
   \         /\       /
    \     /  \     /
     \   /    \   /
      \ /     \ /
       \      \
        \     A
```

**output:** $b : \text{pure}$

```
  s3 --true / b-- s4
   \         /\       /
    \     /  \     /
     \   /    \   /
      \ /     \ /
       \      \
        \     B
```

**C**
A = (States_A, Inputs_A, Outputs_A, update_A, initialState_A)
B = (States_B, Inputs_B, Outputs_B, update_B, initialState_B)

The synchronous side-by-side composition C is given by:

\[
\begin{align*}
\text{States}_C &= \text{States}_A \times \text{States}_B \\
\text{Inputs}_C &= \text{Inputs}_A \times \text{Inputs}_B \\
\text{Outputs}_C &= \text{Outputs}_A \times \text{Outputs}_B \\
\text{initialState}_C &= (\text{initialState}_A, \text{initialState}_B) \\
\text{update}_C\left((s_A, s_B), (i_A, i_B)\right) &= ((s'_A, s'_B), (o_A, o_B)) \\
\text{where} \\
(s'_A, o_A) &= \text{update}_A(s_A, i_A) \\
(s'_B, o_B) &= \text{update}_B(s_B, i_B)
\end{align*}
\]

for all \( s_A \in \text{States}_A, s_B \in \text{States}_B \), \( i_A \in \text{Inputs}_A \), and \( i_B \in \text{Inputs}_B \).
Synchronous Side-by-side Composition

**outputs:** $a, b : \text{pure}$

**output:** $a : \text{pure}$

- $s_1 \rightarrow \text{true / a} \rightarrow s_2$
- $A$

**output:** $b : \text{pure}$

- $s_3 \rightarrow \text{true /} \rightarrow s_4$
- $B$

**outputs:** $a, b : \text{pure}$

- $(s_1, s_3) \rightarrow \text{true / a} \rightarrow (s_2, s_4)$
- $C$

- $(s_1, s_4) \rightarrow \text{true / a, b} \rightarrow (s_2, s_3)$
- $true / b$
- $C$
Asynchronous Side-by-side Composition

- **Semantics 1**: a reaction of $C$ is a reaction of one of $A$ or $B$, where the choice is nondeterministic (*interleaving semantics*).

  $$\text{update}_C((s_A, s_B), (i_A, i_B)) = ((s'_A, s'_B), (o'_A, o'_B))$$

  where either

  $$(s'_A, o'_A) = \text{update}_A(s_A, i_A) \text{ and } s'_B = s_B \text{ and } o'_B = \text{absent}$$

  or

  $$(s'_B, o'_B) = \text{update}_B(s_B, i_B) \text{ and } s'_A = s_A \text{ and } o'_A = \text{absent}$$

  for all $s_A \in \text{States}_A$, $s_B \in \text{States}_B$, $i_A \in \text{Inputs}_A$, and $i_B \in \text{Inputs}_B$.

- **Semantics 2**: a reaction of $C$ is a reaction of $A$, $B$, or both $A$ and $B$, where the choice is nondeterministic.
Asynchronous Side-by-side Composition

outputs: $a, b : \text{pure}$

output: $a: \text{pure}$

outputs: $a, b : \text{pure}$

output: $b: \text{pure}$
Shared Variables

- Extended state machines have variables that are read/written by transitions.
- These can be shared when composing state machines.
- Useful when modeling interrupts and threads.
- Ensuring correct semantics though can be challenging.
**Shared Task Queue Example**

**shared variable**: pending: int
**input**: request: pure
**outputs**: doneA, doneB : pure

**A**

- **input**: request: pure
- **output**: done: pure

- request

- idle
- pending := 0
- pending > 0 ∧ ¬request/
- pending := pending + 1

- request /

- request ∧ pending = 0 / done

- request ∧ pending > 0 / done

- request / done

- pending := pending - 1

- done
- doneA

**B**

- **input**: request: pure
- **output**: done: pure

- request

- idle
- pending := 0
- pending > 0 ∧ ¬request/
- pending := pending - 1

- request /

- request ∧ pending = 0 / done

- request ∧ pending > 0 / done

- request / done

- pending := pending + 1

- done
- doneA

**C**

- **input**: request: pure
- **output**: done: pure

- request

- idle
- pending := 0
- pending > 0 ∧ ¬request/
- pending := pending - 1

- request /

- request ∧ pending = 0 / done

- request ∧ pending > 0 / done

- request / done

- pending := pending + 1
Semantic Subtleties

- Interleaving semantics makes accesses to the shared variable atomic.
  - Tricky to satisfy in practice.
- What if both machines react or machines use synchronous semantics?
  - Leads to non-deterministic outputs.
Cascade Composition

\[ \begin{array}{c}
  i_1 & \rightarrow & A & \rightarrow & o_1 & \rightarrow & i_2 & \rightarrow & B & \rightarrow & o_2 \\
\end{array} \]
Cascade Composition Example

**Input:** $a$: pure
**Output:** $b$: pure

**Input:** $b$: pure
**Output:** $c$: pure

Diagram:

- **A**
  - Initial state: $s1$
  - Transition: $a \rightarrow s2$
  - Transition: $\neg a \rightarrow s1$
  - Transition: $true \rightarrow s2$

- **B**
  - Initial state: $s3$
  - Transition: $b \rightarrow s4$
  - Transition: $\neg b \rightarrow s3$
  - Transition: $true \rightarrow s4$

- **C**
  - Transition: $b \rightarrow c$
  - Transition: $b \rightarrow c$

- **Paths**:
  - Path from $s1$ to $s2$ via $a$
  - Path from $s3$ to $s4$ via $b$

Synchronous Cascade Composition Example

**input:** $a$: pure

**output:** $c$: pure

![Diagram](attachment:diagram.png)
variable: $pcount$: $\{0, \cdots, 55\}$
input: $sigR$: pure
outputs: $pedG, pedR$: pure

\[
\begin{align*}
pcount & := 0 \\
\text{if } pcount \geq 55 \text{ then } pedR \\
\text{else if } sigR \text{ then } pedG \\
\text{else } pcount & := pcount + 1 \\
pcount & := 0
\end{align*}
\]
variable: count: \{0, \cdots, 60\}

inputs: pedestrian : pure

outputs: sigR, sigG, sigY : pure

count < 60 / 
    count := count + 1

count \geq 60 / sigG
    count := 0

count := count + 1

pedestrian \land count < 60 / 
    count := count + 1

count \geq 60 / sigY
    count := 0

count := count + 1

pedestrian \land count \geq 60 / sigY
    count := 0

count := count + 1

count \geq 5 / sigR
    count := 0

count := count + 1

count := count + 1
variables: count: \{0, \cdots, 60\}, pcount: \{0, \cdots, 55\}
input: pedestrian: pure

count < 60 /
count := count + 1

count ≥ 60 / sigG
count := 0

count := count + 1

count ≥ 60 / sigY
count := 0

pedestrian ∨ count ≥ 60 / sigY

pedestrian ∧ count < 60 /

count := count + 1

count ≥ 60 / sigY
count := 0

pcount ≥ 55 / pedR

pedestrian ∧ count ≥ 60 / sigY

yellow, red

count := count + 1

count ≥ 5 / sigR, pedG
count := 0

pcount := 0

red, green

count := 0

pcount := 0

count := count + 1

pcount := pcount + 1
Arbitrary Interconnections of State Machines
Hierarchical FSM

A \quad \quad \quad B

\[ g_1 / a_1 \]

\[ g_2 / a_2 \]

C \quad \quad \quad D

\[ g_3 / a_3 \]

\[ g_4 / a_4 \]
Semantics of a Hierarchical FSM

A

\[ g_1 \land g_4 / a_4; a_1 \]

\[ g_1 \land \neg g_4 / a_1 \]

C

\[ \neg g_1 \land g_3 / a_3 \]

\[ \neg g_1 \land g_4 / a_4 \]

D

\[ g_1 \land \neg g_3 / a_1 \]

\[ g_1 \land g_3 / a_3; a_1 \]

\[ g_2 / a_2 \]
Preemptive Transition Example

$g_1 / a_1$

$g_2 / a_2$

$g_3 / a_3$

$g_4 / a_4$
Semantics of a Preemptive Transition

A \xrightarrow{g_1 / a_1} \xrightarrow{g_2 / a_2} C \xrightarrow{\neg g_1 \land g_3 / a_3} \xrightarrow{\neg g_1 \land g_4 / a_4} D \xrightarrow{g_1 / a_1}
History Transition Example

A \xrightarrow{g_1/a_1} B
A \xrightarrow{g_2/a_2} H

C \xrightarrow{g_3/a_3} D
C \xrightarrow{g_4/a_4} D
Semantics of a History Transition

\[ g_1 \land \neg g_4 / a_1 \]
\[ g_2 / a_2 \]
\[ \neg g_1 \land g_4 / a_4 \]
\[ g_1 \land \neg g_3 / a_1 \]
\[ g_1 \land g_3 / a_3 ; a_1 \]
\[ \neg g_1 \land g_3 / a_3 \]
\[ g_1 \land g_4 / a_4 ; a_1 \]
Concluding Remarks

• Any well-engineered system is a composition of simpler components.
• Considered concurrent composition and hierarchical composition.
• For concurrent composition, introduced both synchronous and asynchronous composition.
• Several possible semantics for asynchronous composition.
• Hierarchical models similar to Statecharts introduced by Harel (1987).