

Embedded System Design and Modeling

VI. Composition of State Machines

Composition of state machines

- ▶ How to combine multiple smaller FSMs into a bigger one?
- ▶ What problems arise?
- ▶ Two types of compositions:
 1. **Spatial** composition: how are the components connected?
 2. **Temporal** composition: how do the components react in time?

Spatial composition

Spatial composition = how are two components connected, how does the information flow between the components

- ▶ **Side-by-side** composition = no common inputs/outputs, no shared data
- ▶ **Cascade** composition = Outputs of one FSM are inputs to another one
- ▶ **Feedback** composition = (Some) outputs of a FSM are inputs to the same FSM, or to some other component which is in front

Side-by-side composition

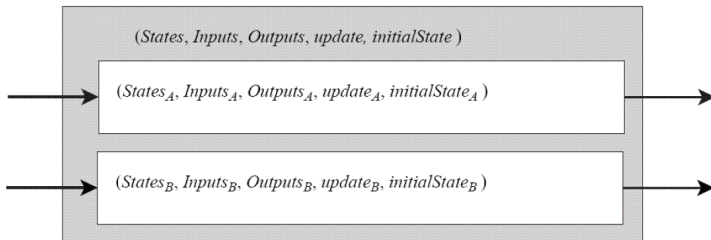


Figure 1: Side-by-side composition

Cascade composition

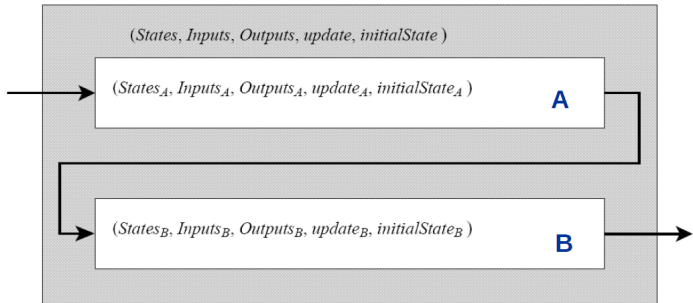


Figure 2: Cascade composition

- Outputs of FSM A are inputs to FSM B

Feedback composition

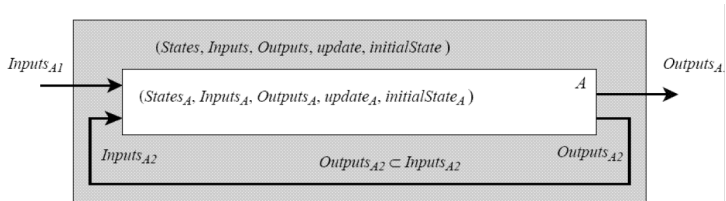


Figure 3: Feedback composition

- Some outputs of the FSM are coming back as inputs

Temporal composition

Temporal composition = when do two components react?

- ▶ **Sequential vs Parallel** composition:
 - ▶ **Sequential** = the two FSM do not work at the same time
 - ▶ **Parallel** = the two FSM work at the same time
- ▶ **Asynchronous vs Synchronous** composition = only for parallel composition
 - ▶ **Synchronous** = transitions are taken at the same time in both FSMs
 - ▶ **Asynchronous** = transitions are taken at independent times in the FSMs

Sequential composition

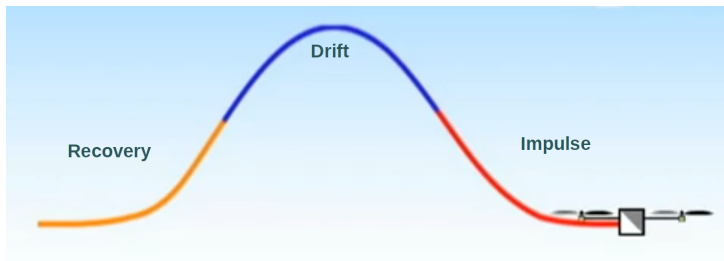


Figure 4: Example of Sequential composition

- ▶ 13579/:<https://www.youtube.com/watch?vD3QgGpzzIM>
- ▶ The drone has three modes of operation, working **in sequence**

Parallel composition

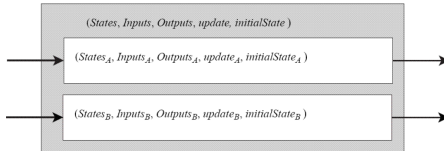


Figure 5: Side-by-side composition

- ▶ The two FSMs form an **equivalent model**
- ▶ When do the transitions in these FSM take place?
 - ▶ Synchronous: simultaneously
 - ▶ Asynchronous: independently

Synchronous composition

- ▶ Consider the two FSM on the left (A and B)
- ▶ The equivalent model is on the right

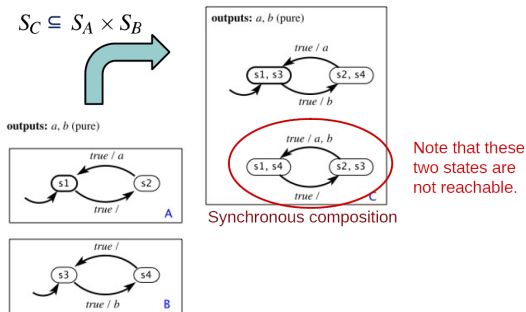


Figure 6: Synchronous composition

Synchronous composition

Redraw here

Synchronous composition

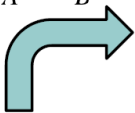
- ▶ In the equivalent model:
 - ▶ States = combination of states of the two FSMs
 - ▶ Transition = transition in FSM A and FSM B, happening simultaneously.
 - ▶ There might exist unreachable states in the equivalent model (states that will never be reached)

Asynchronous composition

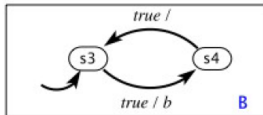
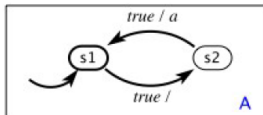
Draw here

Asynchronous composition

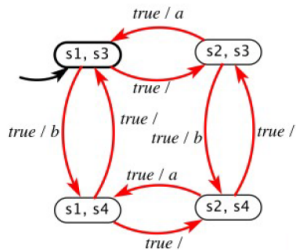
$$S_C \subseteq S_A \times S_B$$



outputs: a, b (pure)



outputs: a, b (pure)



Note that now
all states are
reachable.

Asynchronous composition
using interleaving semantics

Figure 7: Asynchronous composition

Asynchronous composition

- ▶ In the equivalent model:
 - ▶ States = combination of states of the two FSMs
 - ▶ Transitions in the two FSMs can take place at irregular and independent (not synchronized) times
 - ▶ All states are reachable
 - ▶ because one model can be much faster than the other

Asynchronous composition

Flavors of asynchronous composition

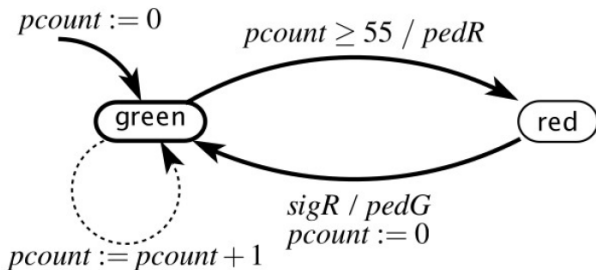
- ▶ How are simultaneous transitions handled?
- ▶ **Interleaving** semantics:
 - ▶ simultaneous transition in models A and B is not allowed (we may have either a transition in model A, or a transition in B)
 - ▶ i.e. transition from A takes place first, then transition from B takes place after a non-zero time delay (or vice-versa)
- ▶ **Simultaneous** semantics:
 - ▶ simultaneous transition in models A and B is allowed
 - ▶ for example, we may have either
 - ▶ transition only in model A
 - ▶ transition only in model B
 - ▶ Simultaneous transition in models A and B

Example

variable: $pcount: \{0, \dots, 55\}$

input: $sigR$: pure

outputs: $pedG, pedR$: pure



This light stays green for 55 seconds, then goes red.
Upon receiving a $sigR$ input, it repeats the cycle.

Figure 8: Composition - Pedestrian Light

Example

variable: $count: \{0, \dots, 60\}$

inputs: $pedestrian : \text{pure}$

outputs: $sigR, sigG, sigY : \text{pure}$

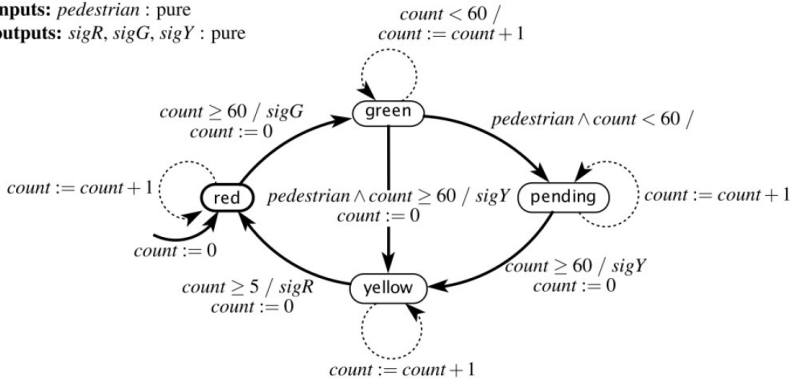
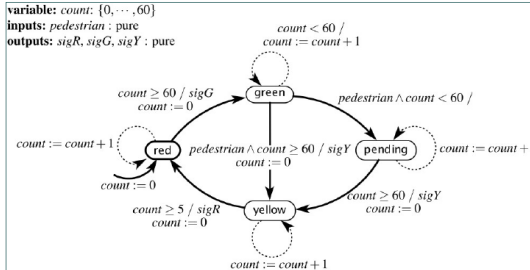


Figure 9: Composition - Car Light

Example

Pedestrian Light with Car Light



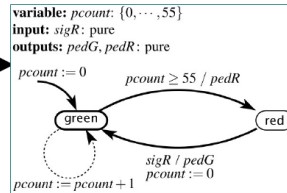
$sigY$

$sigG$

$sigR$

What is the size of
the state space of the
composite machine?

$sigR$



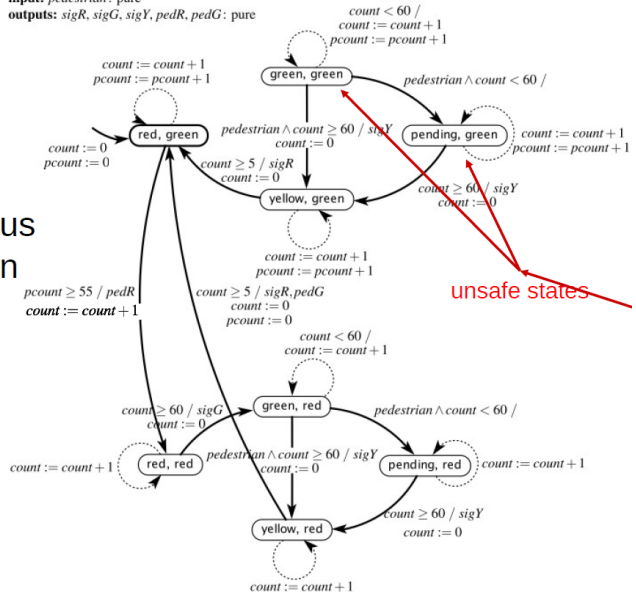
$pedG$

$pedR$

Figure 10: Cascade Composition - Both Lights

Example

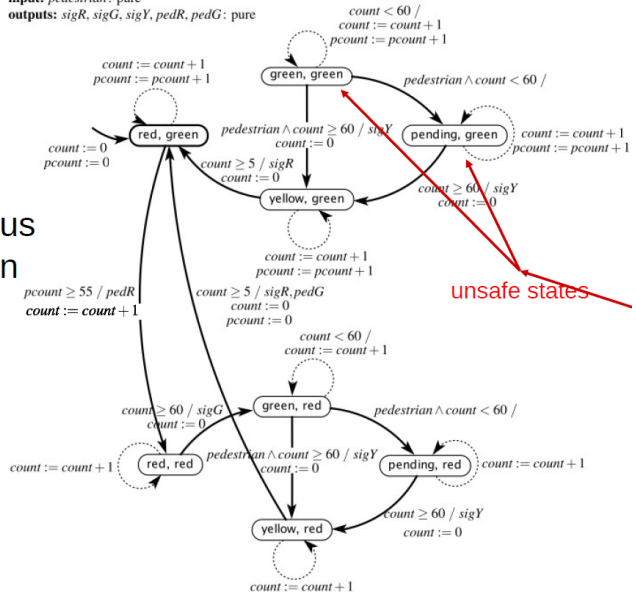
variables: $count: \{0, \dots, 60\}, pcount: \{0, \dots, 55\}$
input: $pedestrian: \text{pure}$
outputs: $sigR, sigG, sigY, pedR, pedG: \text{pure}$



Synchronous
composition

Example

variables: $count: \{0, \dots, 60\}, pcount: \{0, \dots, 55\}$
input: $pedestrian: pure$
outputs: $sigR, sigG, sigY, pedR, pedG: pure$



Synchronous
composition

Models of communications

Models of communications

How do two different processes (e.g. two different FSM's) communicate with one another?

Two models:

- ▶ Shared memory
 - ▶ mutex (lock) for resource control
- ▶ Message passing
 - ▶ blocking (synchronous)
 - ▶ non-blocking (asynchronous)

Shared variables

Other possibilities for model composition:

- ▶ **Shared** variables = variables which can be written / read by both models
 - ▶ Analysis much harder
 - ▶ Potential problems: What happens if both models try to access (read or write) the variable at the same time?
 - ▶ Answer: something bad. Might end up with an incorrect value
 - ▶ Solution: access to shared variable must be via **atomic operations** and guarded with a **mutex**

Shared variables

- ▶ **Atomic** operation = an operation that is indivisible (once it starts, it can't be interrupted until it ends)
- ▶ **Mutex** (or **lock**) = a mechanism for ensuring only one process accesses a given resource (e.g. variable) at one time
 - ▶ A process first **acquires** the mutex, if it is available
 - ▶ Only afterwards it accesses the variable
 - ▶ While the mutex is acquired, no other process can access it
 - ▶ The process **releases** the mutex when it's done with the variable
 - ▶ The code between acquiring and releasing the mutex is known as a **critical section**

Message passing: blocking

- ▶ Message passing: blocking (synchronous)
- ▶ There is a sender process and a receiver process
- ▶ When the sender sends, it **waits** for the receiver to acknowledge that it has received the data
- ▶ When the receiver reads, it **waits** for the data
- ▶ Basically, the earlier one waits for the other one

Message passing: non-blocking

- ▶ Message passing: non-blocking (asynchronous)
- ▶ There is a sender process and a receiver process
- ▶ When the sender sends, it **stores** the data somewhere, and goes on
- ▶ When the receiver reads, it **collects** (if available) the data and goes on
- ▶ Neither process waits
- ▶ Works like **the post office**

Message passing

- ▶ Blocking vs non-blocking:
 - ▶ Non-blocking communication needs a storage mechanism (FIFO, LIFO, Queue, list etc.)
 - ▶ This storage space may overflow => need to have safety mechanisms in place
 - ▶ Blocking communication does not need any special storage space (FIFO, LIFO, Queue, list etc.)
 - ▶ But delays one of the processes until the other one is ready