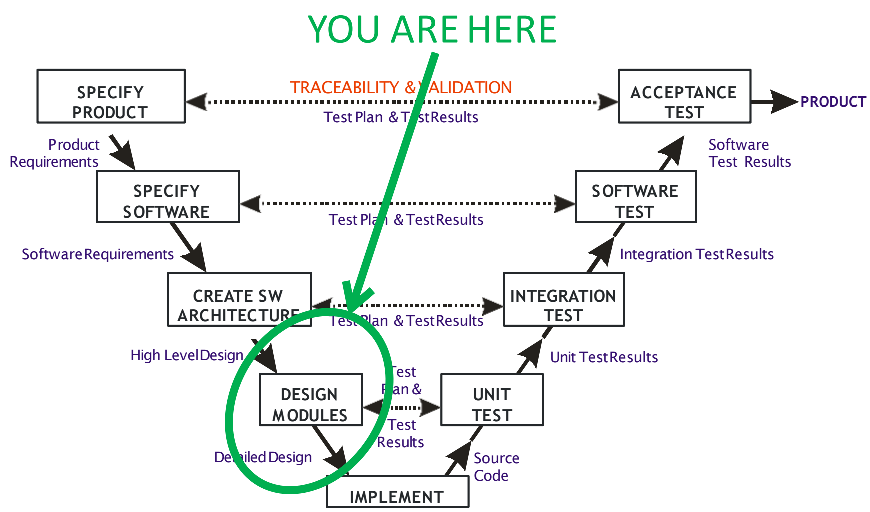
Lecture 8: Composition of State Machines, StateCharts

Seyed-Hosein Attarzadeh-Niaki

Based on slides by Edward Lee and Peter Marwedel

Review

- FSMs with continuous-time inputs
- State refinement
- Classes of hybrid systems
  - Timed automata
  - Higher-order dynamics
  - Supervisory control



Composition of State Machines

How do we construct complex state machines out of simpler "building blocks"?

Spatial

How do the components communicate between each other?

- Side-by-side composition
- Cascade composition
- Feedback composition

Temporal

When do the components execute, relative to each other?

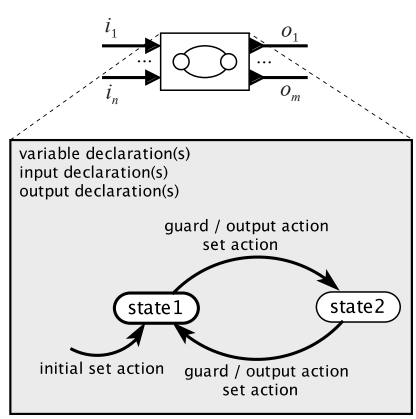
- Sequential
- Concurrent
  - Asynchronous
  - Synchronous

Hybrid Systems Provide *Sequential* Composition Modal models: Sequencing between modes

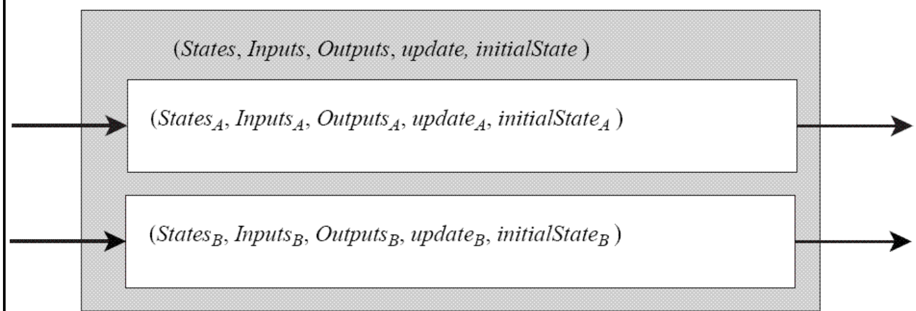
https://www.youtube.com/watch?v=iD3QgGpzzIM [Tomlin et al.]

Requirement for Concurrent Composition: An Interface.

- Actor Model for State Machines
- Expose inputs and outputs



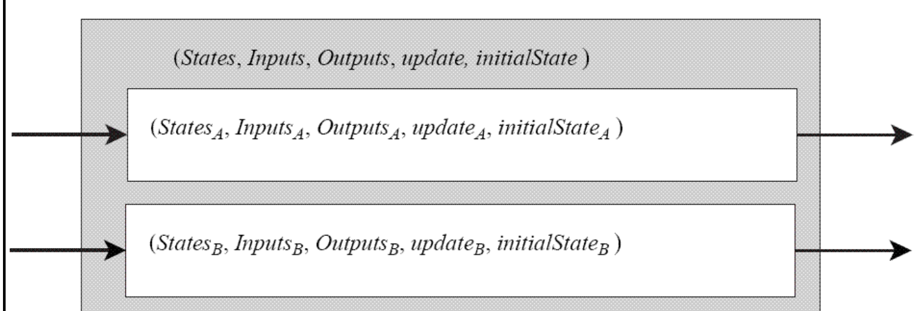
Side-by-Side Composition



A key question: When do these machines react?

How the reactions of composed machines is coordinated is called a “Model of Computation” (MoC).

Side-by-Side, Parallel Composition



When do these machines react? Two of many possibilities:

- Together, in *lock step* (synchronous, concurrent composition)
- *Independently* (asynchronous, concurrent composition)
  - Semantic 1: a reaction of C is a reaction of A or B (interleaving)
  - Semantic 2: a reaction of C is a reaction of A, B, or both

Synchronous Composition

$C = A \times B = (States_C, Inputs_C, Outputs_C, update_C, initialState_C)$

$$States_C = States_A \times States_B$$
$$Inputs_C = Inputs_A \times Inputs_B$$
$$Outputs_C = Outputs_A \times Outputs_B$$
$$initialState_C = (initialState_A, initialState_B)$$
$$update_C((s_A, s_B), (i_A, i_B)) = ((s'_A, s'_B), (o_A, o_B))$$

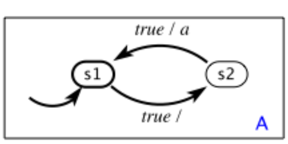
Where:

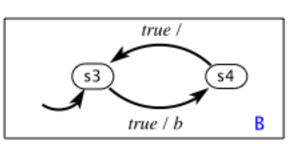
$$(s'_A, o_A) = update_A(s_A, i_A)$$
$$(s'_B, o_B) = update_B(s_B, i_B)$$

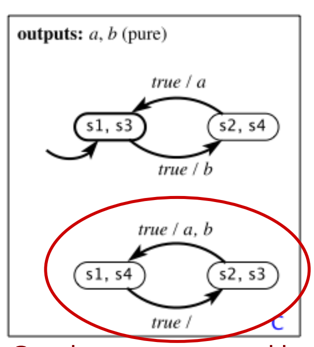
Synchronous Composition

$S_C \subseteq S_A \times S_B$

outputs: *a*, *b* (pure)







Synchronous composition

Note that these two states are not reachable.

Asynchronous Composition (Interleaving Semantics)

$C = A \times B = (States_C\ , Inputs_C\ , Outputs_C\ , update_C\ , initialState_C\ )$

$States_C = States_A \times States_B$

$Inputs_C = Inputs_A \times Inputs_B$

$Outputs_C = Outputs_A \times Outputs_B$

$initialState_C = (initialState_A, initialState_B)$

$update_C((s_A,s_B),(i_A,i_B)) = ((s'_A,s'_B),(o'_A,o'_B))$

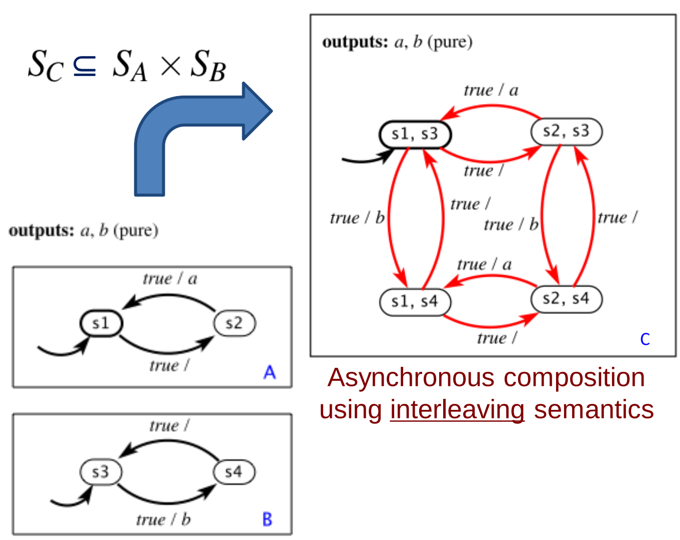
Where:

$(s'_A,\ o'_A) = update_A(s_A,\ i_A)$ and $s'_B = s_B$ and $o'_B =$ absent

$(s'_B,o'_B) = update_B(s_B,i_B)$ and $s'_A = s_B$ and $o'_A =$ absent

Asynchronous Composition

$S_C \subseteq S_A \times S_B$

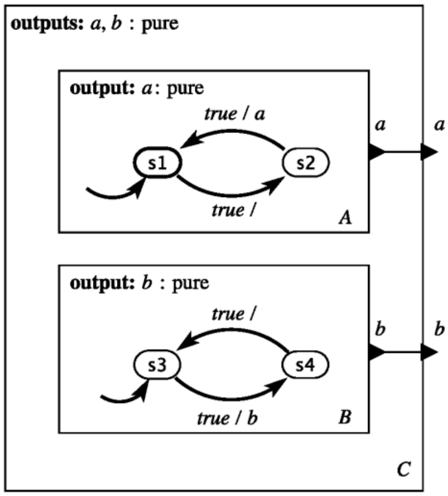


Asynchronous composition using interleaving semantics

Note that now all states are reachable.

Syntax vs. Semantics

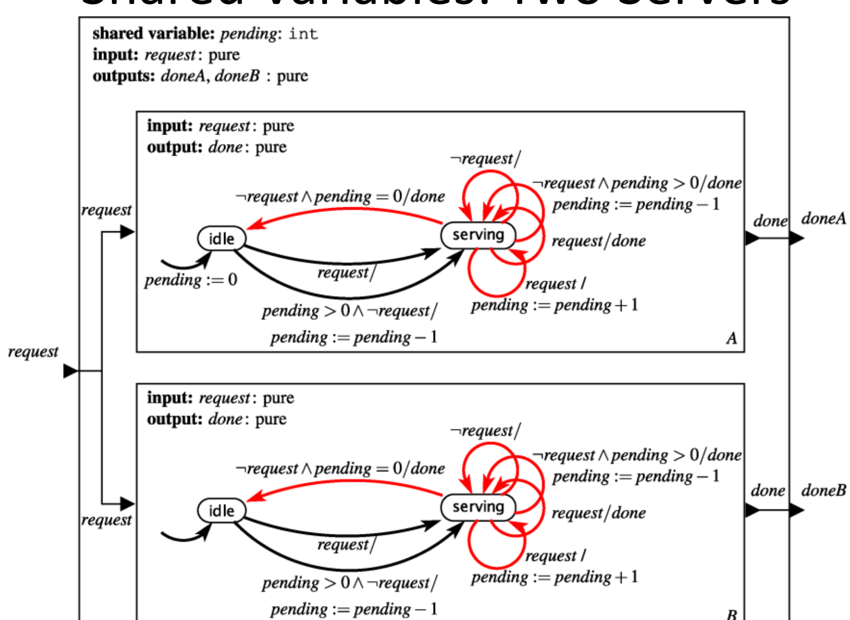
The answers to these questions defines the MoC being used.



Synchronous or Asynchronous composition?

If asynchronous, does it allow simultaneous transitions in A & B? How to choose whether A or B reacts when C reacts?

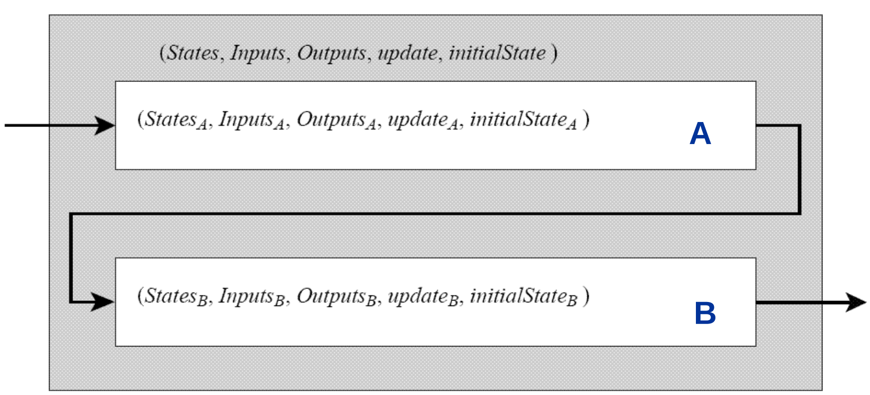
Shared Variables: Two Servers



Subtleties with Shared Variables

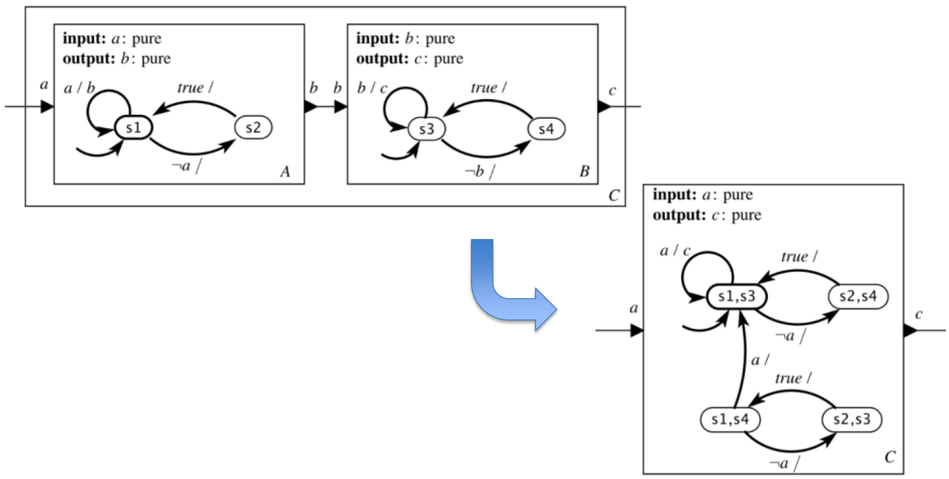
- Interleaving semantics
  - Atomic access to shared variables
  - Missing inputs in case of independent input ports
  - Might not make good use of idle machines
- Synchronous composition
  - Read (by a guard) and write a variable simultaneously
  - Synchronous interleaving semantics
    - Non-determinism
    - Fixed order (priority, etc.)

Cascade Composition (Serial Composition)



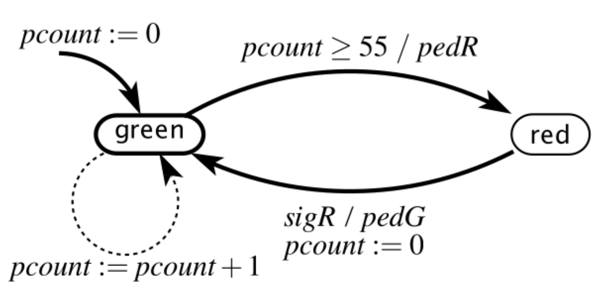
Output port(s) of A connected to input port(s) of B
- Synchronous composition: A and B react in order (but in zero time)
- Asynchronous composition: Needs buffering

Example



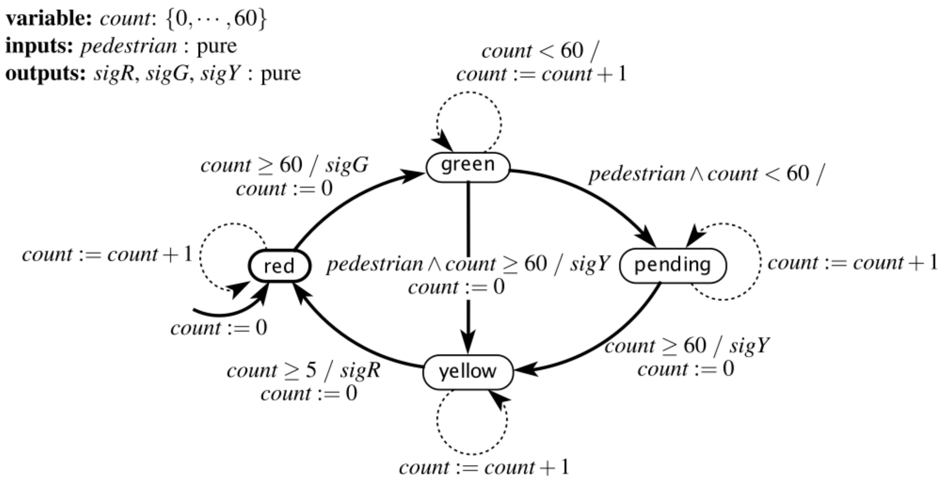
Example:
Time-Triggered Pedestrian Light

variable: *pcount*: $\{0, \cdots, 55\}$
input: *sigR*: pure
outputs: *pedG*, *pedR*: pure

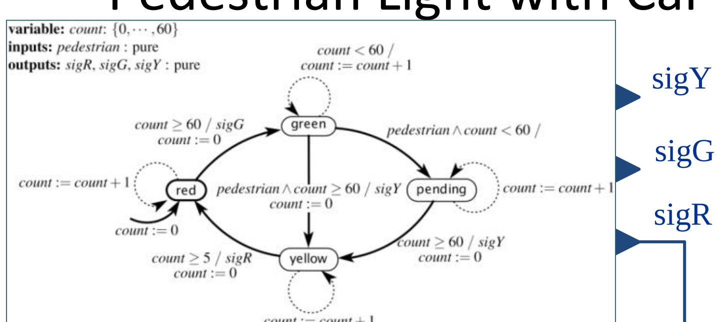


This light stays green for 55 seconds, then goes red.
Upon receiving a sigR input, it repeats the cycle.

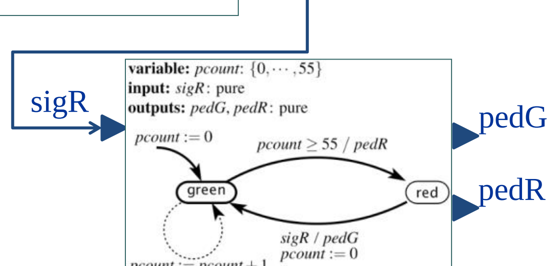
Example: Time-Triggered Car Light



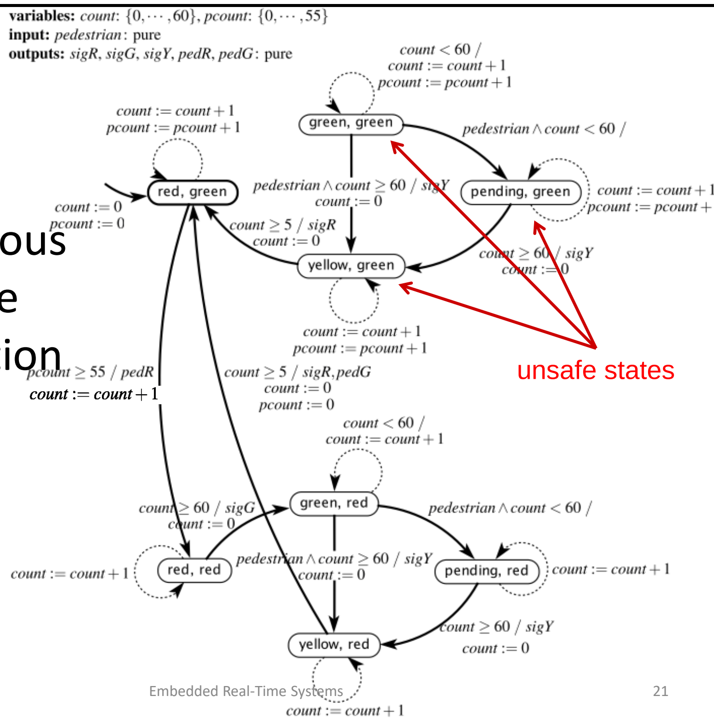
Pedestrian Light with Car Light



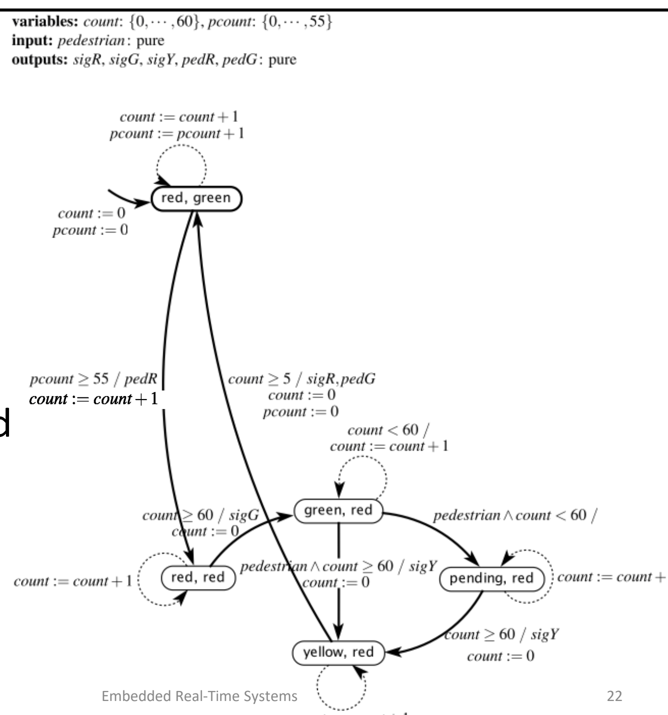
What is the size of the state space of the composite machine?



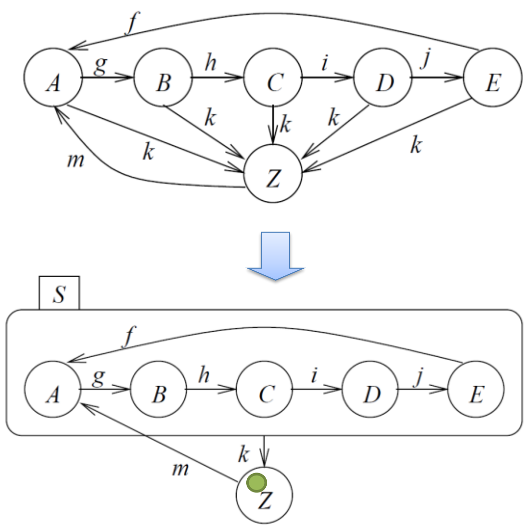
Synchronous cascade composition



Synchronous composition with unreachable states removed

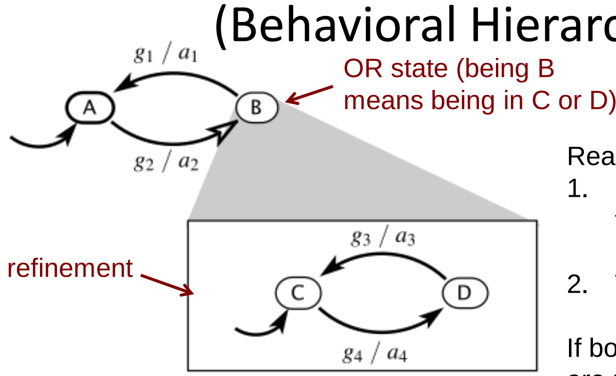


Introducing Hierarchy



FSM will be **in** exactly one of the substates of S if S is **active** (either in A or in B or ..)

Hierarchical State Machines (Behavioral Hierarchy)



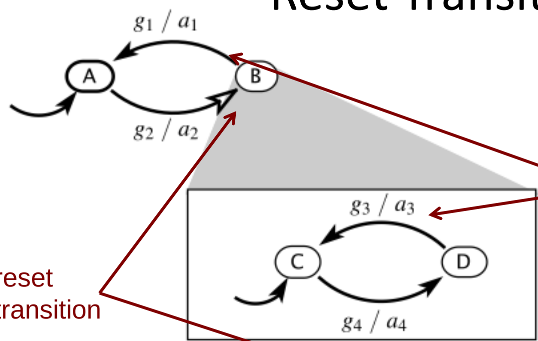
Reaction:

1. First, the refinement of the current state (if any) reacts.
2. Then the top-level machine reacts.

If both produce outputs, they are required to not conflict. The two steps are part of the same reaction.

[Statecharts, David Harel, 1987]

Hierarchical State Machines with Reset Transitions



A reset transition always initializes the refinement of the destination state to its initial state.

simultaneous transitions

Simultaneous transitions can produce multiple outputs. These are required to not conflict.

reset transition

Example trace:

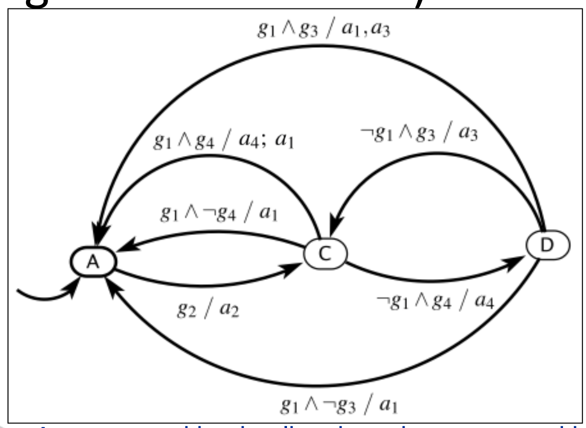
$$A \xrightarrow{g_2/a_2} C \xrightarrow{g_4/a_4} D \xrightarrow{g_1/a_1} A \xrightarrow{g_2/a_2} C \xrightarrow{g_4 \wedge g_1/a_4, a_1} A \cdots$$

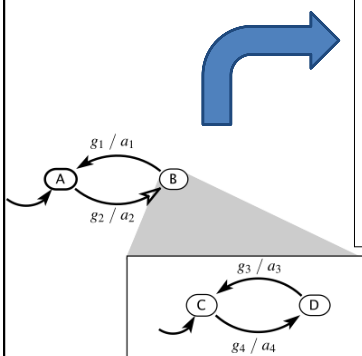
A reset transition implies that when a state with a refinement is left, you can forget the state of the refinement.

Equivalent Flattened State Machine

- Every hierarchical state machine can be transformed into an equivalent "flat" state machine.
- This transformation can cause the state space to blow up substantially.

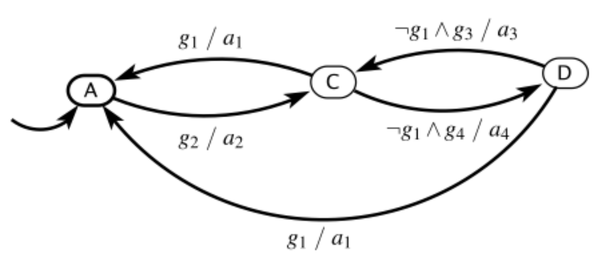
Flattening the state machine (assuming reset transitions):

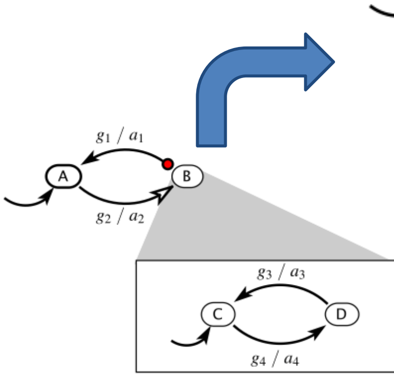




A reset transition implies that when a state with a refinement is left, it is not necessary to remember the state of the refinement. Hence there are fewer states.

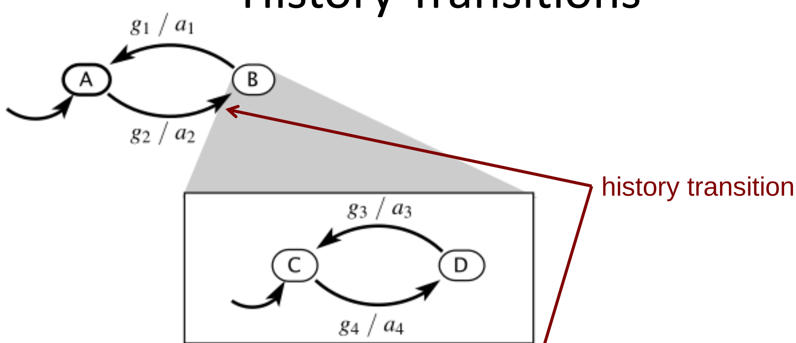
Preemptive Transitions





A preemptive transition specifies that the guard should be evaluated before the current state refinement reacts, and if it is true, then the current state should not react.

Hierarchical State Machines with History Transitions

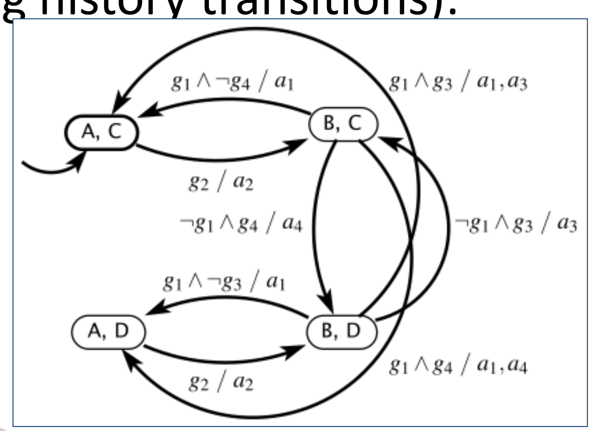


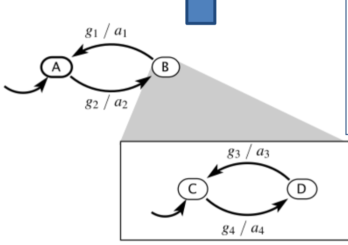
Example trace:

$$A \xrightarrow{g_2/a_2} C \xrightarrow{g_4/a_4} D \xrightarrow{g_1/a_1} A \xrightarrow{g_2/a_2} D \xrightarrow{g_3 \wedge g_1/a_3,a_1} A \cdots$$

A history transition implies that when a state with a refinement is left, it is nonetheless necessary to remember the state of the refinement.

Flattening the state machine (assuming history transitions):

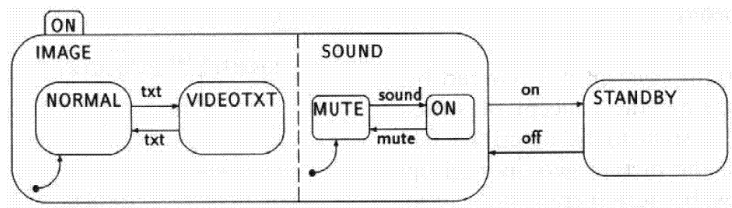




A history transition implies that when a state with a refinement is left, it is nonetheless necessary to remember the state of the refinement. Hence A,C and A,D.

Hierarchical FSMs + Synchronous Composition: StateCharts [Harel 87]

- Modeling with
  – Hierarchy (OR states)
  – Synchronous composition (AND states)
  – Broadcast (for communication)
- Used extensively in practice



Example due to Reinhard von Hanxleden

Summary of Key Concepts

- States can have refinements (other modal models)
  – OR states
  – AND states
- Different types of transitions:
  – History
  – Reset
  – Preemptive

Evaluation of StateCharts

Pros (👍)

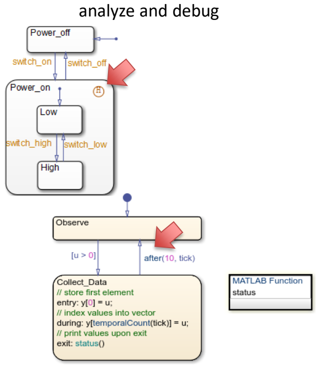
- Hierarchy allows arbitrary nesting of AND- and OR-super states.
- (StateMate-) Semantics defined in a follow-up paper to original paper.
- Large number of commercial simulation tools available (StateMate, StateFlow, BetterState, ...)
- Available "back-ends" translate StateCharts into SW or HW languages, thus enabling software or hardware implementations.

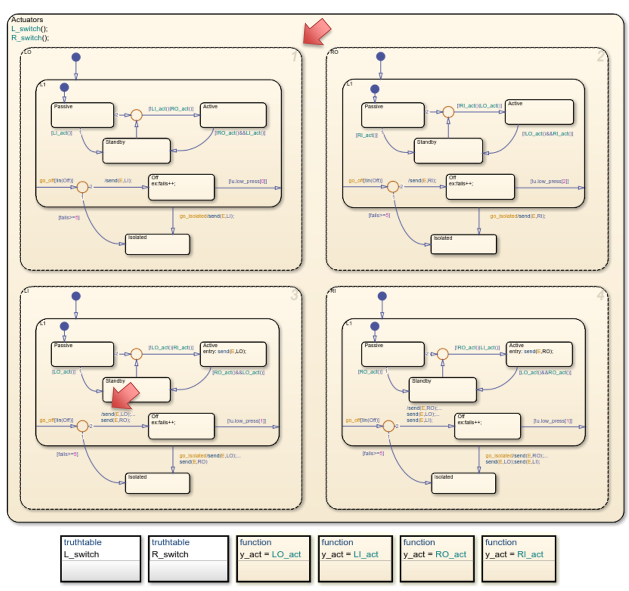
Cons (👎)

- Not useful for distributed applications,
- no program constructs,
- no description of non-functional behavior,
- no object-orientation,
- no description of structural hierarchy,
- generated programs may be inefficient.

StateCharts Modeling in Stateflow

- Design state machines, flow charts, state transition tables, and truth tables
- React to input signals, events, messages, and time-based conditions
- Use graphical animation to analyze and debug





Summary

- Composition enables building complex systems from simpler ones.
  - Synchronous vs. Asynchronous composition
- The emphasis of synchronous composition, in contrast with threads, is on *determinate* and *analyzable* concurrency.
- Hierarchical FSMs enable compact representations of large state machines.
  - Can be converted to flat FSMs with more states