Control Logic using Finite State Machines

Bilung Lee

Edward A. Lee

Department of EECS, UC Berkeley

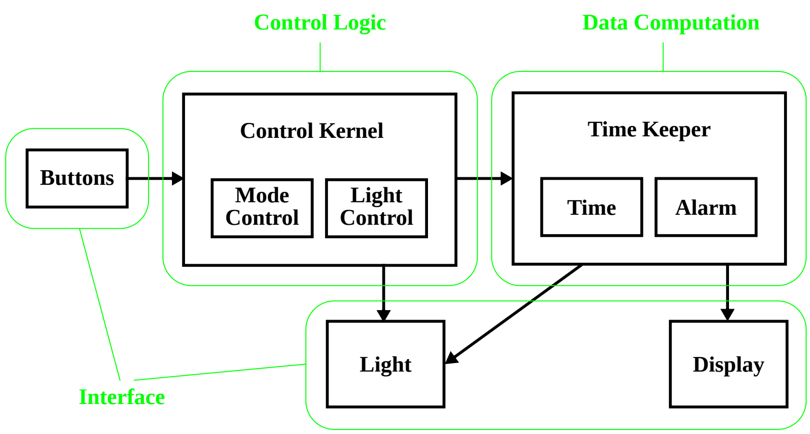
February 19, 1999

Major collaborator: Xiaojun Liu

Problem

- Modern systems tend to include nontrivial control logic

Example: Digital watch



How to describe such a system?

How to Describe the Control Logic?

- **Example: Elevator controller**

Plain English

If the elevator is on the floor 1 and the floor requested is the floor 1, then the elevator remains on the floor 1.
If the elevator is on the floor 1 and the floor requested is the floor 2, then the elevator is raised up 1 floor.
If the elevator is on the floor 1 and the floor requested is the floor 3, then the elevator is raised up 2 floors.

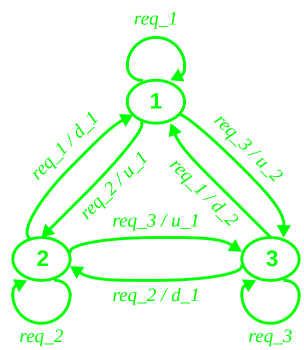
If the elevator is on the floor 2 and the floor requested is the floor 1, then the elevator is lowered down 1 floor.
If the elevator is on the floor 2 and the floor requested is the floor 2, then the elevator remains on the floor 2.
If the elevator is on the floor 2 and the floor requested is the floor 3, then the elevator is raised up 1 floor.

If the elevator is on the floor 3 and the floor requested is the floor 1, then the elevator is lowered down 2 floors.
If the elevator is on the floor 3 and the floor requested is the floor 2, then the elevator is lowered down 1 floor.
If the elevator is on the floor 3 and the floor requested is the floor 3, then the elevator remains on the floor 3.

Imperative programs (e.g. C codes)

```
while (1) {
  switch (cur) {
    case 1:
      if (req_1) {
        u_1=0; d_1=0; u_2=0; d_2=0; cur=1;
      } else if (req_2) {
        u_1=1; d_1=0; u_2=0; d_2=0; cur=2;
      } else if (req_3) {
        u_1=0; d_1=0; u_2=1; d_2=0; cur=3;
      }
      break;
    case 2:
      if (req_1) {
        u_1=0; d_1=1; u_2=0; d_2=0; cur=1;
      } else if (req_2) {
        u_1=0; d_1=0; u_2=0; d_2=0; cur=2;
      } else if (req_3) {
        u_1=1; d_1=0; u_2=0; d_2=0; cur=3;
      }
      break;
    case 3:
      if (req_1) {
        u_1=0; d_1=0; u_2=0; d_2=1; cur=1;
      } else if (req==2) {
        u_1=0; d_1=1; u_2=0; d_2=0; cur=2;
      } else if (req==3) {
        u_1=0; d_1=0; u_2=0; d_2=0; cur=3;
      }
      break;
  }
}
```

Finite state machines

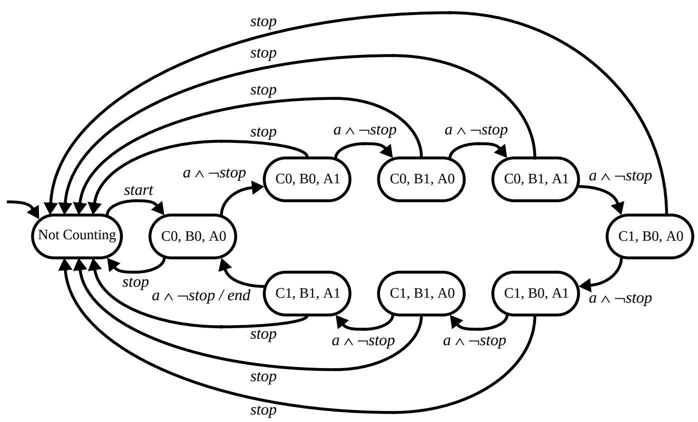


- Visual syntax
- Better analysis

Upgrade to Hierarchical Concurrent FSMs

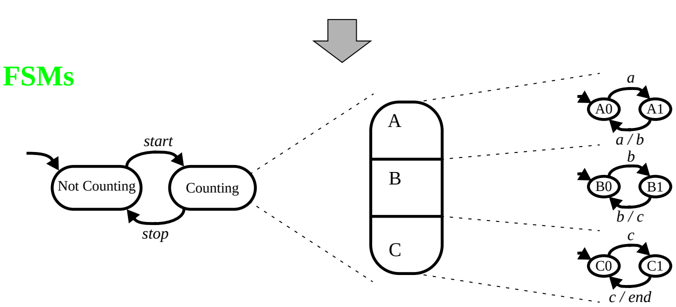
Finite State Machines (FSMs)

- Good for describing sequential control behaviors
- Non-trivial systems generally require a lot of states/transitions



Hierarchical Concurrent FSMs (HCFSMs)

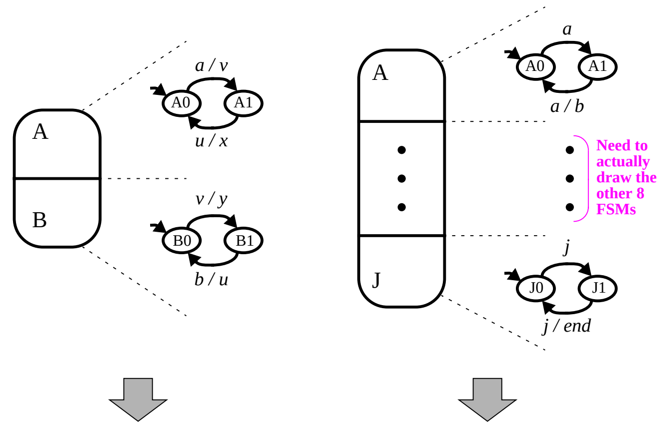
- Hierarchy: A state may be refined into a set of substates
- Concurrency: Multiple simultaneously active and communicating FSMs
- Examples: Statecharts and its (at least 20) variants



Upgrade to Heterogeneous HCFSMs

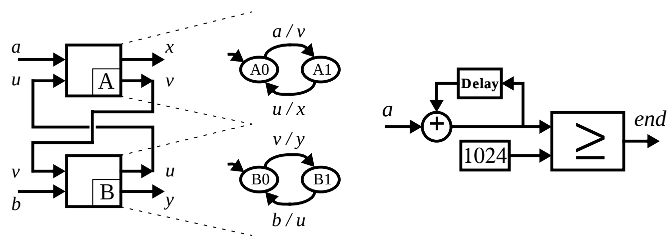
HCFSMs

- Good for describing complex control behaviors
- Most models tightly integrate only one concurrency semantics, for example, Argos = SR + FSM and CFSM = DE + FSM
- Not suitable for specifying computation-oriented tasks (hence, hard to specify a complete design)



Heterogeneous HCFSMs

- Combine FSMs with various concurrency models (hence, enable selection of different concurrency semantics)
- Computation-oriented models, such as dataflow, can be included



Design Methodology

Objective: A system specification scheme capable to

- **Describe both control logic and computation tasks**
- **Specify composite behaviors (concurrency and hierarchy)**
- **Enable the selection of different concurrency semantics**

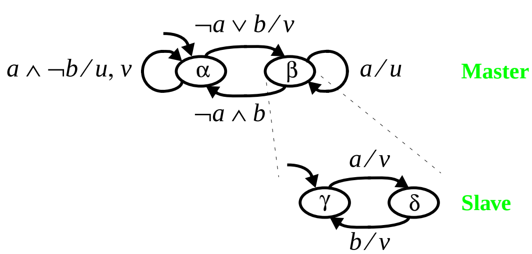
(No existing schemes support all of the above three)

Heterogeneous approach:

- **Allow hierarchy and heterogeneity in the FSM**
- **Let the FSM be hierarchically combined with other existing concurrency models**
- **Choose the most appropriate model for the problem at hand**

Hierarchy in FSMs

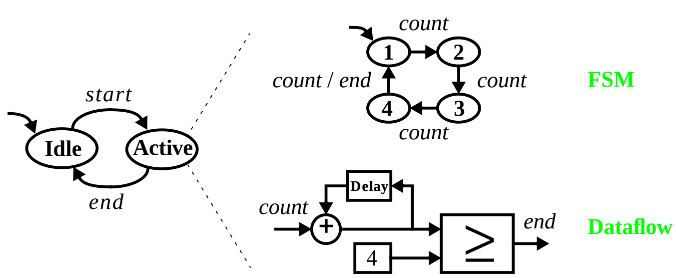
- **A state of an FSM may be refined into another FSM**



- **Inputs/outputs of the slave are a subset of those of its master**
- **The slave reacts first, and then its master reacts**
- **Strength**
  - **Reduce the number of transitions**

Heterogeneity in FSMs

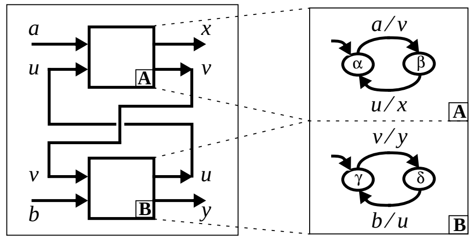
- **The slave inside a state of the FSM need not be an FSM**



- **Key Principle**
  - **The slave must have a well-defined determinate and finite operation, called a *step* of the slave**
- **The slave is invoked first, and then its master reacts**
- **Strength**
  - **Appropriate models can be included for different situations (e.g. dataflow for computation-intensive tasks)**

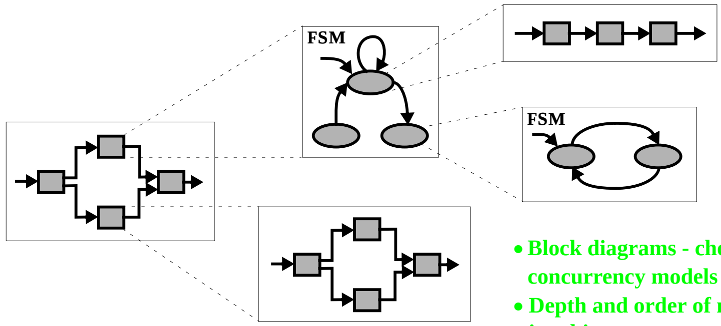
Heterogeneity in FSMs (continued)

- **FSMs may be used inside modules of other model**



- **Key Principle**
  - **The model must provide a way to determine the inputs for each module and when the module should react**
- **These FSMs are concurrent FSMs when the model contains concurrency semantics**
- **Strengths**
  - **Concurrency is naturally included**
  - **Reduce the number of states**

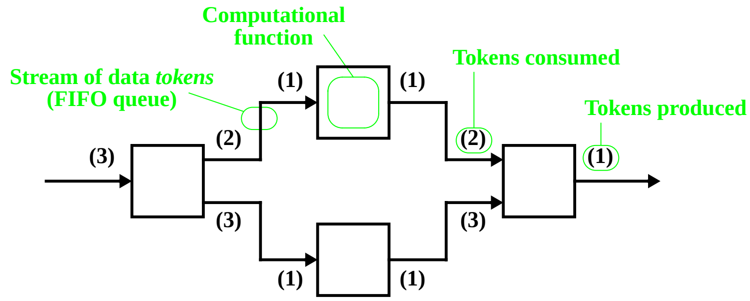
Mixing FSMs with Concurrency Models



- **Block diagrams - choices of concurrency models**
- **Depth and order of nesting is arbitrary**

- **Strength**
  - **Heterogeneity, hierarchy, concurrency and FSMs are all included**
- **Current focus: Interaction of FSMs with**
  - **Synchronous Dataflow (SDF)**
  - **Discrete Events (DE)**
  - **Synchronous/Reactive Model (SR)**

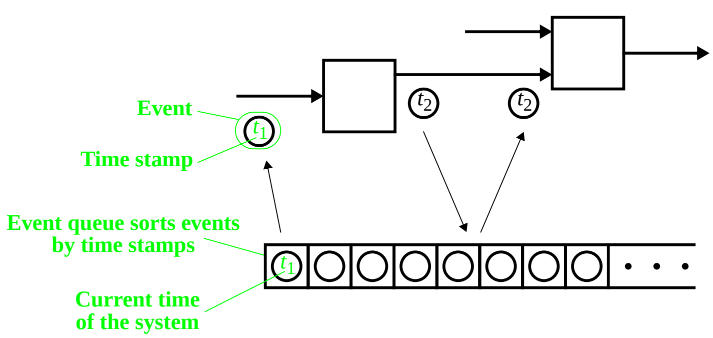
Synchronous Dataflow (SDF)



To interact with the FSM

- Event encoding
  - Absent/Present event in FSM ↔ 0/1 valued token in SDF
- FSM inside SDF
  - One block firing in SDF → One reaction of FSM
- SDF inside FSM
  - One slave step in FSM → One iteration of SDF

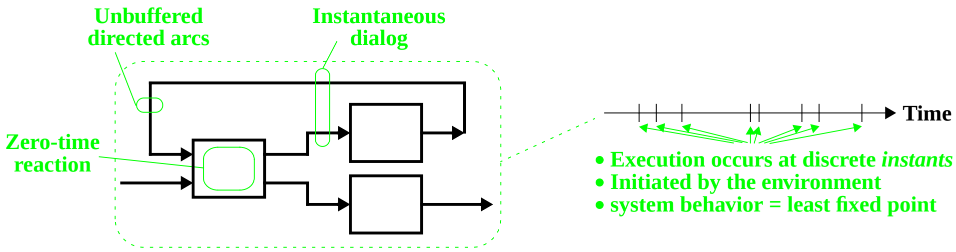
Discrete Events (DE)



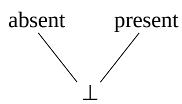
To interact with the FSM

- Events passed through FSM have the same time stamps
- FSM inside DE
  - One block firing in DE → One reaction of FSM
- DE inside FSM
  - One slave step in FSM → Simulation of DE up to time stamp of input

Synchronous/Reactive Model (SR)



- **To interact with the FSM**
- **Support $\perp$ in the FSM**



$\neg$	
0	1
1	0
$\perp$	$\perp$

$\wedge$	0	1	$\perp$
0	0	0	0
1	0	1	$\perp$
$\perp$	0	$\perp$	$\perp$

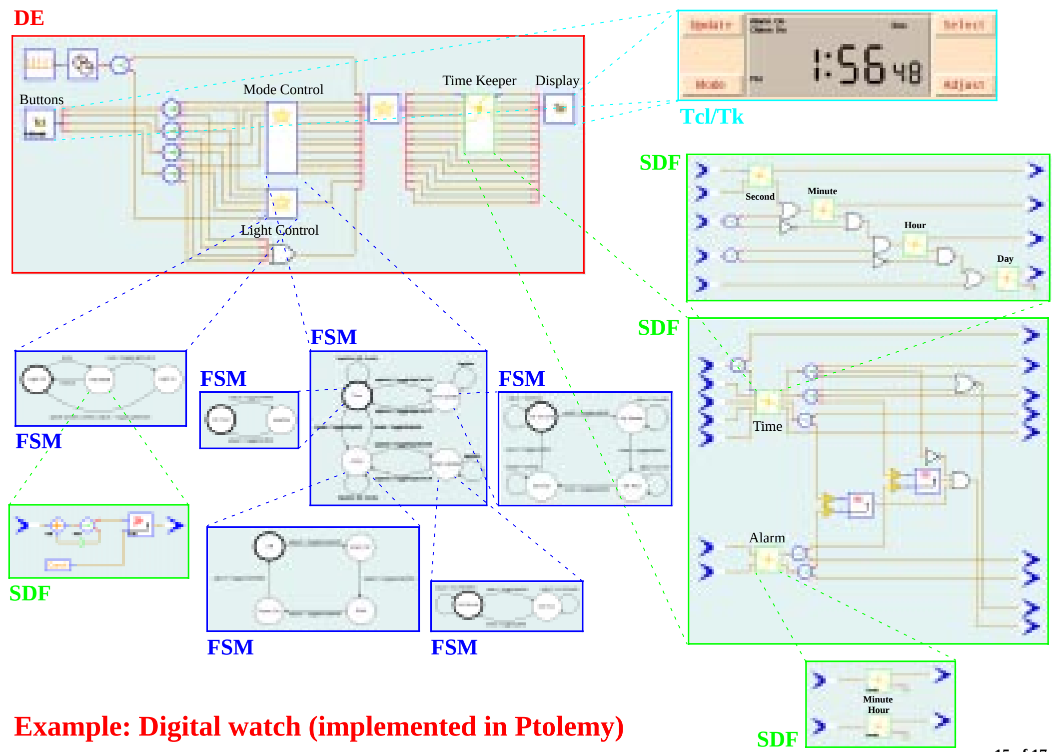
$\vee$	0	1	$\perp$
0	0	1	$\perp$
1	1	1	$\perp$
$\perp$	$\perp$	$\perp$	$\perp$

- **FSM inside SR**
  - **One block firing in SR $\rightarrow$ One reaction of FSM**
- **SR inside FSM**
  - **One slave step in FSM $\rightarrow$ One instant of SR**

Characteristics of Different Models

Model (one step)	Strengths	Weaknesses
Finite State Machines (one reaction)	• Good for sequential control • Can be made deterministic (often is not, however) • Map well to hardware and software	• Computation-intensive systems are hard to specify
Synchronous Dataflow (one iteration)	• Good for signal processing • Loosely synchronized • Deterministic • Map well to hardware and software	• Control-intensive systems are hard to specify
Discrete Events (simulation up to the time stamp of the input)	• Good for asynchronous digital hardware • Globally synchronized • Can be made deterministic (often is not, however)	• Expensive to implement in software • May over-specify systems
Synchronous/ Reactive Model (one instant)	• Good for control-intensive systems • Tightly synchronized • Deterministic • Map well to hardware and software	• Computation-intensive system are over-specified

Capability for a Complete Design



Example: Digital watch (implemented in Ptolemy)

Comparison with Related Work

Specification Schemes	State Transitions	Imperative Constructs	Hierarchy for State Transitions	Concurrency for State Transitions	Concurrency for Imperative Constructs	Choices of Concurrency
SDL	●	○	○	●	○	○
Statecharts	●	○	●	●	○	○
Argos	●	○	●	●	○	○
CFSM	●	○	●	●	○	○
Mini-Statecharts	●	○	●	●	○	⊛
Argos + Lustre	●	●	●	●	●	○
SpecCharts	●	●	●	●	●	○
Stateflow + Simulink	●	●	●	●	●	⊛
This work	●	●	●	●	●	●

● : Fully supported. ⊛ : Partially supported. ○ : Not supported.

Conclusions

- **Heterogeneous combination**
  - **FSMs can be hierarchically combined with multiple concurrency models**
  - **Different models have strengths and weaknesses, and thus are best suitable in certain situations**
  - **Although only three concurrency models are discussed, the combination can be extended for other models, e.g. CSP, CT, etc., as long as we provide their interaction mechanisms**
- **Design framework**
  - **Subsystems can be separately specified and designed**
  - **The simple and determinate mechanisms we provide can be used to combine the subsystems as a whole for validation using simulation**
  - **Example: Digital cellular phone**

Team 1: SDF + FSM for modem, speech coder

Team 2: SR + FSM for user interface controller

Team 3: DE + FSM for communication protocol

Team 4: Combine results for validation