EECS 249: Models of Computation
Finite State Machines
Alberto Sangiovanni-Vincentelli
Luciano Lavagno
Roberto Passerone
Marco Sgroi

Design

- From an idea...
- ... build something that performs a certain function
- Never done directly:
  - some aspects are not considered at the beginning of the development
  - the designer wants to explore different possible implementations in order to maximize (or minimize) a cost function
- Models can be used to reason about the properties of an object
Formal Model of Design: why?

- Informal specification leads to:
  - ambiguous specification
  - various stages not logically connected
  - costly redesign
- We need:
  - Formal specification
  - Set of Properties
  - Set of Performance Indices
  - Set of Constraints
  
  ![Example](wait i; emit o
  
  ![Example](AG i → o
  
  ![Example](Time (i,o)
  
  ![Example](Time (i,o) < 10 s

Model Of Computation

- A MoC is a framework in which to express what sequence of actions must be taken to complete a computation
- Examples: Finite State Machine, Turing Machine, differential equation
- Why different models?
  - Different models = different properties
  - Turing-complete models are too powerful!
  - Some problems may be undecidable
- MoC needs to
  - be powerful enough for application domain
  - have appropriate synthesis and validation algorithms
Control versus Data Flow

- Fuzzy distinction, yet useful for:
  - specification (language, model, ...)
  - synthesis (scheduling, optimization, ...)
  - validation (simulation, formal verification, ...)

- Rough classification:
  - control:
    - don't know when data arrive (quick reaction)
    - time of arrival often matters more than value
  - data:
    - data arrive in regular streams (samples)
    - value matters most

Control versus Data Flow

- Specification, synthesis and validation methods emphasize:
  - for control:
    - event/reaction relation
    - response time
    (Real Time scheduling for deadline satisfaction)
    - priority among events and processes
  - for data:
    - functional dependency between input and output
    - memory/time efficiency
    (Dataflow scheduling for efficient pipelining)
    - all events and processes are equal
Telecom/MM applications

- Heterogeneous specifications including
  - data processing
  - control functions
- Data processing, e.g. encryption, error correction...
  - computations done at regular (often short) intervals
  - efficiently specified and synthesized using DataFlow models
- Control functions (data-dependent and real-time)
  - say when and how data computation is done
  - efficiently specified and synthesized using FSM models
- Need a common model to perform global system analysis and optimization

Reactive Real-time Systems

- Reactive Real-Time Systems
  - “React” to external environment
  - Maintain permanent interaction
  - Ideally never terminate
  - timing constraints (real-time)
- As opposed to
  - transformational systems
  - interactive systems
Models Of Computation for reactive systems

- We need to consider essential aspects of reactive systems:
  - time/synchronization
  - concurrency
  - heterogeneity
- Classify models based on:
  - how specify behavior
  - how specify communication
  - implementability
  - composability
  - availability of tools for validation and synthesis

Main MOCs:
- Communicating Finite State Machines
- Dataflow Process Networks
- Petri Nets
- Discrete Event
- (Abstract) Codesign Finite State Machines

Main languages:
- StateCharts
- Esterel
- Dataflow networks
Finite State Machines

- Functional decomposition into states of operation
- Typical domains of application:
  - control functions
  - protocols (telecom, computers, ...)
- Different communication mechanisms:
  - synchronous
    (classical FSMs, Moore ‘64, Kurshan ‘90)
  - asynchronous
    (CCS, Milner ‘80; CSP, Hoare ‘85)

FSM Example

- Informal specification:
  If the driver
  turns on the key, and
  does not fasten the seat belt within 5 seconds
  then an alarm beeps
  for 5 seconds, or
  until the driver fastens the seat belt, or
  until the driver turns off the key
FSM Example

Key: ON $\rightarrow$ Start Timer
- KEY_OFF or BELT_ON $\rightarrow$ END_TIMER_5 $\rightarrow$ ALARM_ON
- END_TIMER_10 or BELT_ON or KEY_OFF $\rightarrow$ ALARM_OFF
- If no condition is satisfied, implicit self-loop in the current state

FSM Definition

- $\text{FSM} = (I, O, S, r, \delta, \lambda)$
- $I = \{\text{KEY_ON}, \text{KEY_OFF}, \text{BELT_ON}, \text{END_TIMER_5}, \text{END_TIMER_10}\}$
- $O = \{\text{START_TIMER}, \text{ALARM_ON}, \text{ALARM_OFF}\}$
- $S = \{\text{OFF}, \text{WAIT}, \text{ALARM}\}$
- $r = \text{OFF}$
- $\delta : 2^I \times S \rightarrow S$
  - e.g. $\delta(\{\text{KEY_OFF}\}, \text{WAIT}) = \text{OFF}$
- $\lambda : 2^I \times S \rightarrow 2^O$
  - e.g. $\lambda(\{\text{KEY_ON}\}, \text{OFF}) = \{\text{START_TIMER}\}$
Non-deterministic FSMs

- $\delta$ and $\lambda$ may be relations instead of functions:
  - $\delta \subseteq 2^I \times S \times S$
    - implicit “and”
  - $\lambda \subseteq 2^I \times S \times 2^O$
    - implicit “or”
- Non-determinism can be used to describe:
  - an unspecified behavior
    (incomplete specification)
  - an unknown behavior
    (environment modeling)

E.g. $\delta\{(\text{KEY\_OFF, END\_TIMER\_5}, \text{WAIT})\} = \{\text{OFF}\}$

NDFSM: incomplete specification

- E.g. error checking first partially specified:

Then completed as even parity:
NDFSM: unknown behavior

- Modeling the environment
- Useful to:
  - optimize (don’t care conditions)
  - verify (exclude impossible cases)
- E.g. driver model:

- Can be refined
  - E.g. introduce timing constraints
    - (minimum reaction time 0.1 s)

NDFSM: time range

- Special case of unspecified/unknown behavior, but so common to deserve special treatment for efficiency
- E.g. undetermined delay between 6 and 10 s
NDFSMs and FSMs

- Formally FSMs and NDFSMs are equivalent (Rabin-Scott construction, Rabin ’59)
- In practice, NDFSMs are often more compact (exponential blowup for determinization)

Finite State Machines

- Advantages:
  - Easy to use (graphical languages)
  - Powerful algorithms for
    - synthesis (SW and HW)
    - verification
- Disadvantages:
  - Sometimes over-specify implementation (sequencing is fully specified)
  - Number of states can be unmanageable
  - Numerical computations cannot be specified compactly (need Extended FSMs)
Modeling Concurrency

- Need to compose parts described by FSMs
- Describe the system using a number of FSMs and interconnect them
- How do the interconnected FSMs talk to each other?

FSM Composition

- Bridle complexity via hierarchy: FSM product yields an FSM
- Fundamental hypothesis:
  - all the FSMs change state together (synchronicity)
- System state = Cartesian product of component states
  (state explosion may be a problem...)
- E.g. seat belt control + timer

```
START_TIMER =>
  1
  SEC => 2
  SEC => 3
  SEC => 4
  SEC => END_5_SEC

START_TIMER =>
  0
  SEC => 9
  SEC => 8
  SEC => 7
  SEC => 6
  SEC => 5
```

FSM Composition

Given
- \( M_1 = (I_{11}, O_{11}, S_{11}, r_{11}, \delta_{11}, \lambda_{11}) \) and
- \( M_2 = (I_{22}, O_{22}, S_{22}, r_{22}, \delta_{22}, \lambda_{22}) \)

Find the composition
- \( M = (I, O, S, r, \delta, \lambda) \)

given a set of constraints of the form:
- \( C = \{ (o, i_{11}, \ldots, i_n) : o \text{ is connected to } i_{11}, \ldots, i_n \} \)
**FSM Composition**

- **Unconditional product** $M' = (I', O', S', r', \delta', \lambda')$
  - $I' = I_1 \cup I_2$
  - $O' = O_1 \cup O_2$
  - $S' = S_1 \times S_2$
  - $r' = r_1 \times r_2$
  - $\delta' = \{(A_1, A_2, s_1, s_2, t_1, t_2) : (A_1, s_1, t_1) \in \delta_1 \text{ and } (A_2, s_2, t_2) \in \delta_2\}$
  - $\lambda' = \{(A_1, A_2, s_1, s_2, B_1, B_2) : (A_1, s_1, B_1) \in \lambda_1 \text{ and } (A_2, s_2, B_2) \in \lambda_2\}$

- **Note:**
  - $A_1 \subseteq I_1, A_2 \subseteq I_2, B_1 \subseteq O_1, B_2 \subseteq O_2$
  - $2^X \cup Y = 2^X \times 2^Y$

---

**FSM Composition**

- **Constraint application**
  - $\lambda = \{(A_1, A_2, s_1, s_2, B_1, B_2) \in \lambda' : \text{for all } (o, i_1, \ldots, i_n) \in C \text{ if and only if } i_j \in A_1 \cup A_2 \text{ for all } j\}$

- **The application of the constraint rules out the cases where the connected input and output have different values (present/absent).**
FSM Composition

- $I = I_1 \cup I_2$
- $O = O_1 \cup O_2$
- $S = S_1 \times S_2$
- Assume that $o_1 \in O_1, i_3 \in I_2, o_1 = i_3$ (communication)
- $\delta$ and $\lambda$ are such that, e.g., for each pair:
  - $\delta_1(\{i_1\}, s_1) = t_1$, $\lambda_1(\{i_1\}, s_1) = \{o_1\}$
  - $\delta_2(\{i_2, i_3\}, s_2) = t_2$, $\lambda_2(\{i_2, i_3\}, s_2) = \{o_2\}$
  - we have:
    - $\delta(\{i_1, i_2, i_3\}, s_v, s_2) = (t_1, t_2)$
    - $\lambda(\{i_1, i_2, i_3\}, s_v, s_2) = \{o_1, o_2\}$
  - i.e. $i_3$ is in input pattern iff $o_2$ is in output pattern

Problem: what if there is a cycle?

- Moore machine: $\delta$ depends on input and state, $\lambda$ only on state
  - composition is always well-defined
- Mealy machine: $\delta$ and $\lambda$ depend on input and state
  - composition may be undefined
  - what if $\lambda_1(\{i_2\}, s_2) = \{o_2\}$ but $o_2 \notin \lambda_2(\{i_3\}, s_2)$?

Causality analysis in Mealy FSMs (Berry ’98)
Moore vs. Mealy

- Theoretically, same computational power (almost)
- In practice, different characteristics

Moore machines:
- non-reactive (response delayed by 1 cycle)
- easy to compose (always well-defined)
- good for implementation
  - software is always “slow”
  - hardware is better when I/O is latched

Mealy machines:
- reactive (0 response time)
- hard to compose (problem with combinational cycles)
- problematic for implementation
  - software must be “fast enough” (synchronous hypothesis)
  - may be needed in hardware, for speed
Hierarchical FSM models

- Problem: how to reduce the size of the representation?
- Harel’s classical papers on StateCharts (language) and bounded concurrency (model): 3 orthogonal exponential reductions
- Hierarchy:
  - state a “encloses” an FSM
  - being in a means FSM in a is active
  - states of a are called OR states
  - used to model pre-emption and exceptions
- Concurrency:
  - two or more FSMs are simultaneously active
  - states are called AND states
- Non-determinism:
  - used to abstract behavior

Models Of Computation for reactive systems

- Main MOCs:
  - Communicating Finite State Machines
  - Dataflow Process Networks
  - Petri Nets
  - Discrete Event
  - Codesign Finite State Machines
- Main languages:
  - StateCharts
  - Esterel
  - Dataflow networks
StateCharts

- An extension of conventional FSMs
- Conventional FSMs are inappropriate for the behavioral description of complex control
  - flat and unstructured
  - inherently sequential in nature
- StateCharts supports repeated decomposition of states into sub-states in an AND/OR fashion, combined with a synchronous (instantaneous broadcast) communication mechanism

State Decomposition

- OR-States have sub-states that are related to each other by exclusive-or
- AND-States have orthogonal state components (synchronous FSM composition)
  - AND-decomposition can be carried out on any level of states (more convenient than allowing only one level of communicating FSMs)
- Basic States have no sub-states (bottom of hierarchy)
- Root State: no parent states (top of hierarchy)
To be in state U, the system must be either in state S or in state T.

To be in state U, the system must be both in states S and T.
StateCharts Syntax

- The general syntax of an expression labeling a transition in a StateChart is \( e[c]/a \), where:
  - \( e \) is the event that triggers the transition
  - \( c \) is the condition that guards the transition (cannot be taken unless \( c \) is true when \( e \) occurs)
  - \( a \) is the action that is carried out if and when the transition is taken

- For each transition label:
  - event condition and action are optional
  - an event can be the changing of a value
  - standard comparisons are allowed as conditions and assignment statements as actions

StateCharts Actions and Events

- An action \( a \) on the edge leaving a state may also appear as an event triggering a transition going into an orthogonal state:
  - a state transition broadcasts an event visible immediately to all other FSMs, that can make transitions immediately and so on
  - executing the first transition will immediately cause the second transition to be taken simultaneously

- Actions and events may be associated to the execution of orthogonal components: \( \text{start}(A) \), \( \text{stopped}(B) \)
Graphical Hierarchical FSM Languages

- Multitude of commercial and non-commercial variants:
  - StateCharts, UML, StateFlow, ...
- Easy to use for control-dominated systems
- Simulation (animated), SW and HW synthesis
- Original StateCharts have problems with causality loops and instantaneous events:
  - Circular dependencies can lead to paradoxes
  - Behavior is implementation-dependent
  - Not a truly synchronous language
  - Hierarchical states necessary for complex reactive system specification

Synchronous vs. Asynchronous FSMs

- Synchronous (Esterel, StateCharts):
  - Communication by shared variables that are read and written in zero time
  - Communication and computation happens instantaneously at discrete time instants
  - All FSMs make a transition simultaneously (lock-step)
  - May be difficult to implement
    - Multi-rate specifications
    - Distributed/heterogeneous architectures
Synchronous vs. Asynchronous FSMs

- Asynchronous FSMs:
  - free to proceed independently
  - do not execute a transition at the same time (except for CSP rendezvous)
  - may need to share notion of time: synchronization
  - easy to implement

Synchronization

- Base station - Base station
- Base station - Mobile stations
- Base station - Mobile station
A Mobile Station moving across the cell boundary needs to maintain active connections without interruptions or degradations.

Handover
- tight inter-base-station synchronization (in GSM achieved using GPS)
- asynchronous base station operation (UMTS)

Frame Synchronization
- Medium Access Control Layer: TDMA
  - each active connection is assigned a number of time slots (channel)
- A common notion of time is needed
  - each Base Station sends a frame synchronization pilot (FS) at the beginning of every frame to ensure that all Mobile Stations have the same slot counts

```
FS 0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8
```
Bit Synchronization

- Transmitter interleaves the payload data with a pilot sequence known by the receiver.
- Receiver extracts the clock from the pilot sequence and uses it to sample the payload data.

Asynchronous communication

- Blocking vs. non-Blocking
  - Blocking read
    - process cannot test for emptiness of input
    - must wait for input to arrive before proceed
  - Blocking write
    - process must wait for successful write before continue
  - blocking write/blocking read (CSP, CCS)
  - non-blocking write/blocking read (FIFO, CFMS, SDL)
  - non-blocking write/non-blocking read (shared variables)
Asynchronous communication

- Buffers used to adapt when sender and receiver have different rate
  - what size?
- Lossless vs. lossy
  - events/tokens may be lost
  - bounded memory: overflow or overwriting
  - need to block the sender
- Single vs. multiple read
  - result of each write can be read at most once or several times

Communication Mechanisms

- Rendez-Vous (CSP)
  - No space is allocated for the data, processes need to synchronize in some specific points to exchange data
  - Read and write occur simultaneously
- FIFO
  - Bounded (ECFSMs, CFMSs)
  - Unbounded (SDL, ACFSMs, Kahn Process Networks, Petri Nets)
- Shared memory
  - Multiple non-destructive reads are possible
  - Writes delete previously stored data
# Communication models

<table>
<thead>
<tr>
<th></th>
<th>Transmitters</th>
<th>Receivers</th>
<th>Buffer Size</th>
<th>Blocking Reads</th>
<th>Blocking Writes</th>
<th>Single Reads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsynchronized</td>
<td>many</td>
<td>many</td>
<td>one</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Read-Modify-write</td>
<td>many</td>
<td>many</td>
<td>one</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Unbounded FIFO</td>
<td>one</td>
<td>one</td>
<td>unbounded</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Bounded FIFO</td>
<td>one</td>
<td>one</td>
<td>bounded</td>
<td>no</td>
<td>maybe</td>
<td>yes</td>
</tr>
<tr>
<td>Single Rendezvous</td>
<td>one</td>
<td>one</td>
<td>one</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Multiple Rendezvous</td>
<td>many</td>
<td>many</td>
<td>one</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>