Finite State Machines (FSM)

- Functional decomposition into states of operation
- Inputs and outputs are sequences of events
- 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** => **START_TIMER**
- **WAIT**
- **OFF**
  - **KEY_OFF** or
  - **BELT_ON**
- **END_TIMER_5** or
  - **BELT_ON** or
  - **KEY_OFF** => **ALARM_OFF**
- **ALARM**

If no condition is satisfied, implicit self-loop in the current state

---

**FSM Definition**

- **FSM** = (I, O, S, r, δ, λ)
- **I** = { **KEY_ON**, **KEY_OFF**, **BELT_ON**, **END_TIMER_5**, **END_TIMER_10** }
- **O** = { **START_TIMER**, **ALARM_ON**, **ALARM_OFF** }
- **S** = { **OFF**, **WAIT**, **ALARM** }
- **r** = **OFF**
- **δ** : 2^I × S → S
  - e.g. δ( { **KEY_OFF** }, **WAIT** ) = **OFF**
- **λ** : 2^I × S → 2^O
  - e.g. λ ( { **KEY_ON** }, **OFF** ) = { **START_TIMER** }

*note: self-loop not implied in the function*
Non-deterministic FSMs

- $\delta$ and $\lambda$ may be relations instead of functions:
  - $\delta \subseteq 2^S \times S$ (implied "and")
  - $\lambda \subseteq 2^S \times 2^O$ (implied "or")
  - e.g., $\delta([\text{KEY OFF}, \text{END TIMER 5}], \text{WAIT}) = \{\{\text{OFF}\}, \{\text{ALARM}\}\}$

- Non-determinism can be used to describe:
  - an unspecified behavior (incomplete specification)
  - an unknown behavior (environment modeling)

NDFSM: incomplete specification

- E.g., error checking first partially specified:

- Then completed as even parity:
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

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
FSM Composition

Given
- \( M_1 = (I_1, O_1, S_1, r_1, \delta_1, \lambda_1) \) and
- \( M_2 = (I_2, O_2, S_2, r_2, \delta_2, \lambda_2) \)

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

given a set of constraints of the form:
- \( C = \{ (o, i_1, \ldots, i_n) : o \text{ is connected to } i_1, \ldots, i_n \} \)
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:
  - $\delta_1(\{i_1\}, s_1) = t_1, \quad \lambda_1(\{i_1\}, s_1) = \{o_1\}$
  - $\delta_2(\{i_2, i_3\}, s_2) = t_2, \quad \lambda_2(\{i_2, i_3\}, s_2) = \{o_2\}$

we have:
- $\delta(\{i_1, i_2, i_3\}, (s_1, s_2)) = (t_1, t_2)$
- $\lambda(\{i_1, i_2, i_3\}, (s_1, s_2)) = \{o_1, o_2\}$

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 $\lambda_1(\{i_1\}, s_1) = \{o_1\}$ but $o_1 \in \lambda_2(\{i_3\}, s_2)$?
  - Is $o_1$ output or not?

- 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 vs. Mealy

- Mealy machines:
  - reactive
    (0 response time)
  - hard to compose
    (problem with combinational cycles)
    - Esterel compilation algorithm
  - 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
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
- Extended with arithmetics
- Original StateCharts have problems with *instantaneous reaction* (micro-steps):
  - behavior is implementation-dependent
  - not a truly synchronous language

Synchronous Languages

- Assumptions:
  - the system continuously reacts to internal and external *events* by emitting other events
  - events can occur only at discrete instants
  - zero (negligible) reaction time
- Both control (Esterel) and data flow (Lustre, Signal)
- Very simple syntax and clean semantics
  (based on FSMs)
- Deterministic behavior
- Simulation, software and hardware synthesis, verification
Summary of 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)
  - Numerical computations cannot be specified compactly
    (need extended FSMs)