

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)

ASV's Slide

1

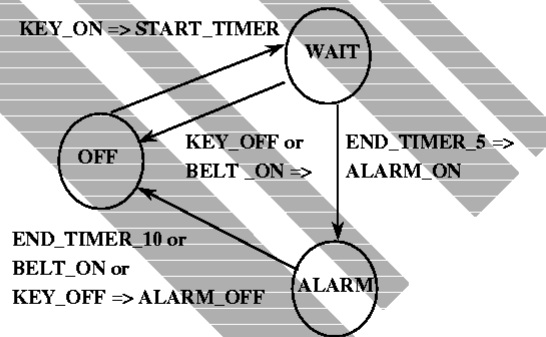
FSM Example

- 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*

ASV's Slide

2

FSM Example



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

ASV's Slide

3

FSM Definition

- ◆ $FSM = (I, O, S, r, \delta, \lambda)$
- ◆ $I = \{ KEY_ON, KEY_OFF, BELT_ON, END_TIMER_5, END_TIMER_10 \}$
- ◆ $O = \{ START_TIMER, ALARM_ON, ALARM_OFF \}$
- ◆ $S = \{ OFF, WAIT, ALARM \}$
- ◆ $r = OFF$
- ◆ $\delta : 2^I \times S \rightarrow S$
 e.g. $\delta(\{ KEY_OFF \}, WAIT) = OFF$
Set of all subsets of I (implicit "and")
All other inputs are implicitly absent
- ◆ $\lambda : 2^I \times S \rightarrow 2^O$
 e.g. $\lambda(\{ KEY_ON \}, OFF) = \{ START_TIMER \}$
note: self-loop not implied in the function

ASV's Slide

4

Non-deterministic FSMs

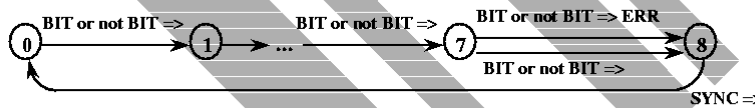
- δ and λ may be *relations* instead of *functions*:
 - ◆ $\delta \subseteq 2^I \times S \times S$
 - implicit "and" (pointing to the set notation)
 - implicit "or" (pointing to the set notation)
 - e.g. $\delta(\{\text{KEY_OFF}, \text{END_TIMER_5}\}, \text{WAIT}) = \{\{\text{OFF}\}, \{\text{ALARM}\}\}$
 - ◆ $\lambda \subseteq 2^I \times S \times 2^O$
- Non-determinism can be used to describe:
 - ◆ an unspecified behavior (incomplete specification)
 - ◆ an unknown behavior (environment modeling)

ASV's Slide

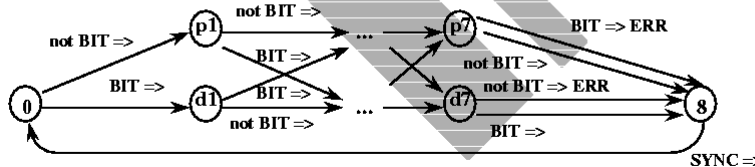
5

NDFSM: incomplete specification

- E.g. error checking first partially specified:



- Then completed as *even parity*:

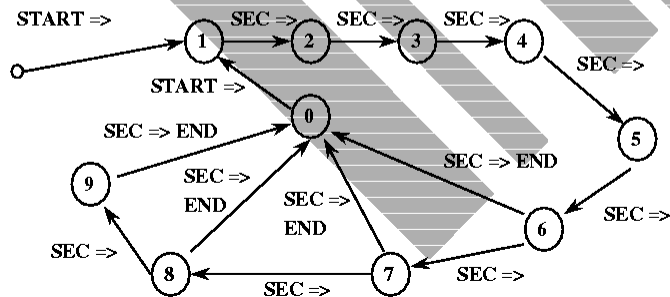


ASV's Slide

6

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



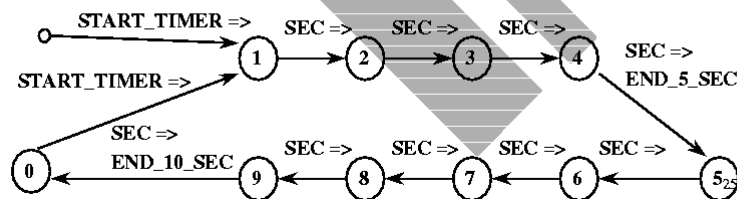
22

ASV's Slide

7

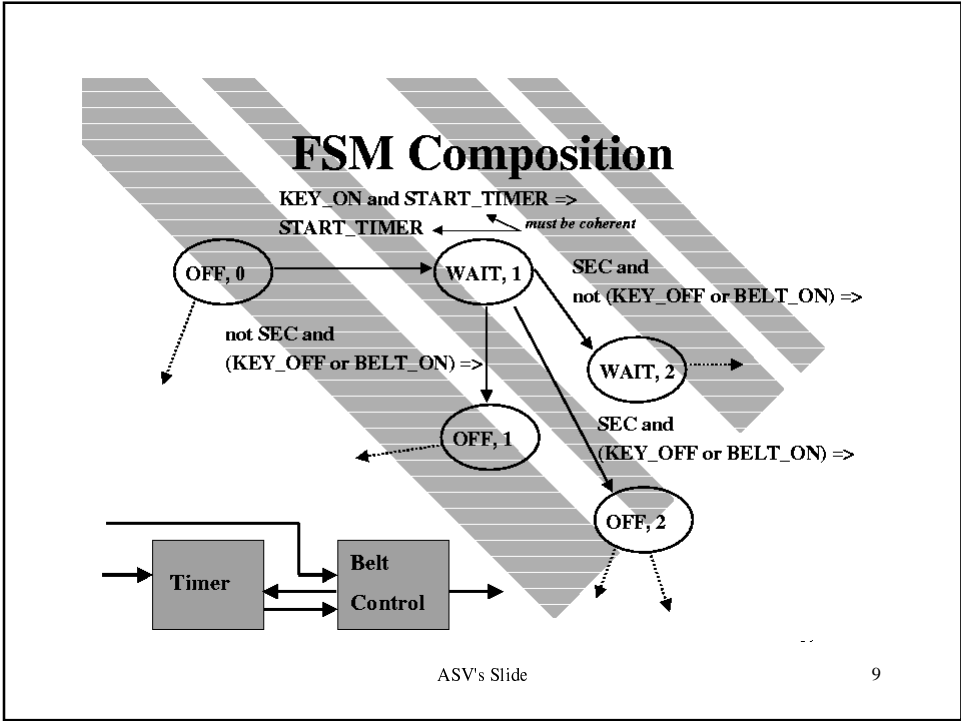
FSM Composition

- 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



ASV's Slide

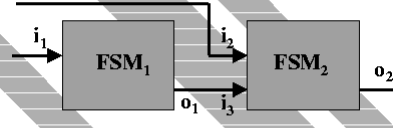
8



- ## 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:**
 - ◆ $C = \{ (o, i_1, \dots, i_n) : o \text{ is connected to } i_1, \dots, i_n \}$
- ASV's Slide 10

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)
- δ and λ 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:
 - ◆ $\delta(\{i_1, i_2, i_3\}, (s_1, s_2)) = (t_1, t_2)$
 - ◆ $\lambda(\{i_1, i_2, i_3\}, (s_1, s_2)) = \{o_1, o_2\}$
 i.e. i_3 is in input pattern iff o_2 is in output pattern

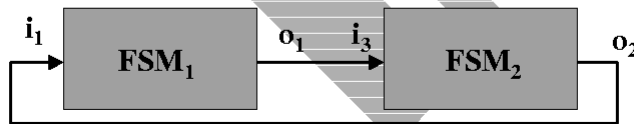


ASV's Slide

11

FSM Composition

- **Problem: what if there is a cycle?**
 - ◆ Moore machine: δ depends on input and state, λ only on state
 ☞ composition is always *well-defined*
 - ◆ Mealy machine: δ and λ depend on input and state
 ☞ composition may be *undefined*
 - ◆ what if $\lambda_1(\{i_1\}, s_1) = \{o_1\}$ but $o_2 \notin \lambda_2(\{i_3\}, s_2)$?
 Is o_1 output or not?



- ◆ Causality analysis in Mealy FSMs (Berry '98)

ASV's Slide

12

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

ASV's Slide

13

Moore vs. Mealy

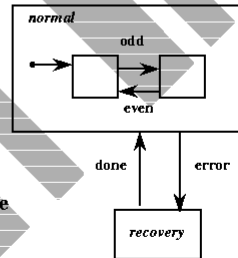
- Mealy machines:
 - ◆ reactive
(0 response time)
 - ◆ hard to *compose*
(problem with combinational cycles)
 - Esterel compilation algorithm
 - ◆ problematic for implementation
 - software must be “fast enough”
(synchronous hypothesis)
 - may be needed in hardware, for speed

ASV's Slide

14

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



ASV's Slide

15

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

ASV's Slide

16

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!!!!

ASV's Slide

17

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**

ASV's Slide

18

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)

