Outline

1. Finite State Machines (FSMs)
   - Introduction
   - Moore and Mealy machines
   - State Diagrams
   - Example
   - Mealy machines
   - Exercise

2. Non deterministic FSMs
   - Non determinism
   - Exercise

3. Hierarchical Finite State Machines
   - Problems with FSMs
   - H-FSM specification

4. The Elevator Example
   - Simple FSM
   - Improved design
State machines are basic building blocks for computing theory.

- very important in theoretical computer science
- many applications in practical systems
- There are many slightly different definitions, depending on the application area
- A state machine is a Discrete Event Discrete State system
  - transitions from one state to another only happen on specific events
  - events do not need to occur at specific times
  - we only need a temporal order between events (events occur one after the other), not the exact time at which they occur
Definition

A deterministic finite state machine (DFSM) is a 5-tuple:

- **S** (finite) set of states
- **I** set of possible input symbols (also called input alphabet)
- **s₀** initial state
- **φ** transitions: a function from (state,input) to a new state
  \[ \phi : S \times I \rightarrow S \]
- **ω** output function (see later)

An event is a new input symbol presented to the machine.

- In response, the machine will react by updating its state and possibly producing an output. This reaction is instantaneous (synchronous assumption).
Outline

1. Finite State Machines (FSMs)
   - Introduction
   - Moore and Mealy machines
   - State Diagrams
   - Example
   - Mealy machines
   - Exercise

2. Non deterministic FSMs
   - Non determinism
   - Exercise

3. Hierarchical Finite State Machines
   - Problems with FSMs
   - H-FSM specification

4. The Elevator Example
   - Simple FSM
   - Improved design
Output function

Two types of machines:

**Moore** output only depends on state:

\[ \omega_{mr} : S \rightarrow \Omega \]

Where \( \Omega \) is the set of output symbols. In this case, the output only depends on the state, and it is produced upon entrance on a new state.

**Mealy** output depends on state and input:

\[ \omega_{ml} : S \times I \rightarrow \Omega \]

In this case, the output is produced upon occurrence of a certain transaction.
Moore machines

- Moore machines are the simplest ones
- If $\Omega = \{\text{yes, no}\}$, the machine is a recognizer
- A recognizer is able to accept or reject sequences of input symbols
- The set of sequences accepted by a recognizer is a regular language
Outline

1 Finite State Machines (FSMs)
   - Introduction
   - Moore and Mealy machines
   - State Diagrams
     - Example
     - Mealy machines
     - Exercise

2 Non deterministic FSMs
   - Non determinism
   - Exercise

3 Hierarchical Finite State Machines
   - Problems with FSMs
   - H-FSM specification

4 The Elevator Example
   - Simple FSM
   - Improved design
State diagrams

- FSM can be represented by State Diagrams

- final states are identified by a double circle
Outline

1. Finite State Machines (FSMs)
   - Introduction
   - Moore and Mealy machines
   - State Diagrams
   - Example
   - Mealy machines
   - Exercise

2. Non deterministic FSMs
   - Non determinism
   - Exercise

3. Hierarchical Finite State Machines
   - Problems with FSMs
   - H-FSM specification

4. The Elevator Example
   - Simple FSM
   - Improved design
In this example $l = \{a, b\}$. The following state machine recognizes string $aba$.
Example: recognizer II

- Recognize string $a^n b^m$ with $n$ even and $m$ odd (i.e. $aabbb$, $b$, $aab$ are all legal sequences, while $a$, $aabb$, are non legal)

- $S_4$ is an error state. It is not possible to go out from an error state (for every input, no transaction out of the state)

- $S_2$ is an accepting state, however we do not know the length of the input string, so it is possible to exit from the accepting state if the input continues

- If we want to present a new string we have to reset the machine to its initial state
Non regular language

- FSM are not so powerful. They can only recognize simple languages.
- Example:
  - strings of the form $a^n b^n$ for all $n \geq 0$ cannot be recognized by a FSM (because they only have a finite number of states).
  - they could if we put a limit on $n$. For example, $0 \leq n \leq 10$. 
Outline

1. Finite State Machines (FSMs)
   - Introduction
   - Moore and Mealy machines
   - State Diagrams
   - Example
   - Mealy machines
   - Exercise

2. Non deterministic FSMs
   - Non determinism
   - Exercise

3. Hierarchical Finite State Machines
   - Problems with FSMs
   - H-FSM specification

4. The Elevator Example
   - Simple FSM
   - Improved design
Mealy machines

- In Mealy machines, output is related to both state and input.
- In practice, output can be associated to a transition.
- Given the synchronous assumption, the Moore’s model is equivalent to the Mealy’s model: for every Moore machine, it is possible to derive an equivalent Mealy machine, and viceversa.
Example: parity check

- In this example, we have a Mealy machine that
  - outputs 1 if the number of symbols 1 in input so far is odd;
  - it outputs 0 otherwise.

![Mealy machine diagram]

- Usually, Mealy machines have a more compact representation than Moore machines (i.e. they perform the same task with a number of states that is no less than the equivalent Moore machine).
A FSM can be represented through a table.
The table shown below corresponds to the parity-check Mealy FSM shown just before.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_0$</td>
<td>$S_0 / 0$</td>
<td>$S_1 / 1$</td>
</tr>
<tr>
<td>$S_1$</td>
<td>$S_1 / 1$</td>
<td>$S_0 / 0$</td>
</tr>
</tbody>
</table>
Stuttering symbol

- Input and output alphabets include the absent symbol $\epsilon$
- It correspond to a null input or output
- When the input is absent, the state remains the same, and the output is absent
- Any sequence of inputs can be interleaved or extended with an arbitrary number of absent symbols without changing the behavior of the machine
- the absent symbol is also called the stuttering symbol
Abbreviations

- If no guard is specified for a transition, the transition is taken for every possible input (except the absent symbol $\epsilon$)
- If no output is specified for a transition, the output is $\epsilon$
- Given a state $S_0$, if a symbol $\alpha$ is not used as guard of any transition going out of $S_0$, then an implicit transition from $S_0$ to itself is defined with $\alpha$ as guard and $\epsilon$ as output
Exercise

Draw the state diagram of a FSM with $I = \{0, 1\}$, $\Omega = \{0, 1\}$, with the following specification:

- let $x(k)$ be the sequence of inputs
- the output $\omega(k) = 1$ iff $x(k-2) = x(k-1) = x(k) = 1$
Solution

- three states: S0 is the initial state, S1 if last input was 1, S2 if last two inputs were 1
Outline

1. Finite State Machines (FSMs)
   - Introduction
   - Moore and Mealy machines
   - State Diagrams
   - Example
   - Mealy machines
   - Exercise

2. Non deterministic FSMs
   - Non determinism
   - Exercise

3. Hierarchical Finite State Machines
   - Problems with FSMs
   - H-FSM specification

4. The Elevator Example
   - Simple FSM
   - Improved design
Deterministic machines

Transitions are associated with
- a source state
- a guard (i.e., an input value)
- a destination state
- a output

In deterministic FSM, a transition is uniquely identified by the first two.

In other words, given a source state and an input, the destination and the output are uniquely defined.
Non deterministic FSMs

- A non deterministic finite state machine is identified by a 5-tuple:
  - $I$ set of input symbols
  - $\Omega$ set of output symbols
  - $S$ set of states
  - $S_0$ set of initial states
  - $\phi$ transition function:

  $$\phi : S \times I \rightarrow (S \times \Omega)^*$$

  where $S^*$ denotes the power set of $S$, i.e. the set of all possible subsets of $S$.

- In other words, given a state and an input, the transition returns a set of possible pairs (new state, output).
Non determinism

- Non determinism is used in many cases:
  - to model randomness
  - to build more compact automata
Non determinism

- Non determinism is used in many cases:
  - to model *randomness*
  - to build *more compact* automata
- Randomness is when there is more than one possible behaviour and the system follows one specific behavior at random
Non determinism

- Non determinism is used in many cases:
  - to model randomness
  - to build more compact automata
- Randomness is when there is more than one possible behaviour and the system follows one specific behavior at random
- Randomness has nothing to do with probability! we do not know the probability of occurrence of every behavior, we only know that they are possible
Non determinism

- Non determinism is used in many cases:
  - to model randomness
  - to build more compact automata

- Randomness is when there is more than one possible behaviour and the system follows one specific behavior at random

- Randomness has nothing to do with probability! we do not know the probability of occurrence of every behavior, we only know that they are possible

- A more abstract model of a system hides unnecessary details, and it is more compact (less states)
We now build an automata to recognize all input strings (of any length) that end with a 01.
Equivalence between D-FSM and N-FSM

- It is possible to show that Deterministic FSMs (D-FSMs) are equivalent to non deterministic ones (N-FSMs)
- Proof sketch
  - Given a N-FSM $\mathcal{A}$, we build an equivalent D-FSM $\mathcal{B}$ (i.e. that recognizes the same strings recognized by the N-FSM. For every subset of states of the $\mathcal{A}$, we make a state of $\mathcal{B}$. Therefore, the maximum number of states of $\mathcal{B}$ is $2^{|S|}$. The start state of $\mathcal{B}$ is the one corresponding to the $\mathcal{A}$. For every subset of states that are reachable from the start state of state of $\mathcal{A}$ with a certain symbol, we make one transition in $\mathcal{B}$ to the state corresponding to the sub-set. The procedure is iterated until all transitions have been covered.
Outline

1 Finite State Machines (FSMs)
   - Introduction
   - Moore and Mealy machines
   - State Diagrams
   - Example
   - Mealy machines
   - Exercise

2 Non deterministic FSMs
   - Non determinism
   - Exercise

3 Hierarchical Finite State Machines
   - Problems with FSMs
   - H-FSM specification

4 The Elevator Example
   - Simple FSM
   - Improved design
Exercise

As an exercise, build the D-FSM equivalent to the previous example of N-FSM

Figure: The N-FSM
Solution

Figure: The N-FSM

- Initial state: \{S0\}

<table>
<thead>
<tr>
<th>state name</th>
<th>subset</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>{S0}</td>
<td>{S0, S1}</td>
<td>{S0}</td>
</tr>
<tr>
<td>q1</td>
<td>{S0, S1}</td>
<td>{S0, S1}</td>
<td>{S0, S2}</td>
</tr>
<tr>
<td>q2</td>
<td>{S0, S2}</td>
<td>{S0, S1}</td>
<td>{S0}</td>
</tr>
</tbody>
</table>
Solution

Figure: The equivalent D-FSM
Outline

1 Finite State Machines (FSMs)
   - Introduction
   - Moore and Mealy machines
   - State Diagrams
   - Example
   - Mealy machines
   - Exercise

2 Non deterministic FSMs
   - Non determinism
   - Exercise

3 Hierarchical Finite State Machines
   - Problems with FSMs
   - H-FSM specification

4 The Elevator Example
   - Simple FSM
   - Improved design
Problems with FSMs

- FSM are *flat* and *global*
- All states stay on the same level, and a transition can go from one state to another
  - It is not possible to *group* states and transitions
- Replicated transition problem:
Product of two FSM

Another problem is related to the cartesian product of two FSM

Suppose we have two distinct FSMs that we want to combine into a single one

Figure: FSM 1

Figure: FSM 2
The result is a state machine FSM 3 where each state corresponds to a pair of state of the original machine. Also, each transition in FSM 3 corresponds to one transition in either of the two original state machines.
Complexity handling

- All these problems have to do with *complexity* of dealing with states.
- In particular, the latter problem is very important, because we often need to combine different simple state machines.
- However, the resulting diagram (or table specification) can become very large.
- We need a different specification mechanism to deal with such complexity.
- In this course, we will study Statecharts (similar to Matlab StateFlow), first proposed by Harel.
Outline

1. Finite State Machines (FSMs)
   - Introduction
   - Moore and Mealy machines
   - State Diagrams
   - Example
   - Mealy machines
   - Exercise

2. Non deterministic FSMs
   - Non determinism
   - Exercise

3. Hierarchical Finite State Machines
   - Problems with FSMs
   - H-FSM specification

4. The Elevator Example
   - Simple FSM
   - Improved design
In H-FSMs, a state can be final or composite.
State specification

- A state consists of:
  - An *entry* action, executed once when the system enters the state
  - An *exit* action, executed once before leaving the state
  - A *do* action, executed *while* in the state (the semantic is not very clear)

- They are all optional

**Figure:** Entry, exit and do behaviors
Transitions

- A transition can have:
  - A *triggering event*, which activates the transition
  - A *guard*, a boolean expression that *enables* the transition. If not specified, the transition is always enabled
  - An *action* to be performed if the transition is activated and enabled, just after the exit operation of the leaving state, and before the entry operation of the entering state

- Only the triggering event specification is mandatory, the other two are optional

**Figure:** Transition, with event, guard and action specified
Or composition

- A state can be decomposed into substates
- When the machine enters state *Composite*, it goes into state *Comp1*
- Then, if event *e2* it goes in *Comp2*, if event *e3* it goes in *Comp3*, else if event *e4* it exits from *Composite*.

**Figure:** A composite state
History

- When the machine exits from a composite state, normally it forgets in which states it was, and when it enters again, it starts from the starting state.
- To “remember” the state, so that when entering again it will go in the same state it had before exiting, we must use the history symbol.

![Diagram of history example](image)

**Figure:** Example of history
AND decomposition

- A state can be decomposed in orthogonal regions, each one contains a different sub-machine.
- When entering the state, the machine goes into one substate for each sub-machine.

*Figure:* Orthogonal states for a keyboard
Outline

1. Finite State Machines (FSMs)
   - Introduction
   - Moore and Mealy machines
   - State Diagrams
   - Example
   - Mealy machines
   - Exercise

2. Non deterministic FSMs
   - Non determinism
   - Exercise

3. Hierarchical Finite State Machines
   - Problems with FSMs
   - H-FSM specification

4. The Elevator Example
   - Simple FSM
   - Improved design
Elevator

Let’s define an “intelligent” elevator

- For a 5-stores building (ground floor, and four additional floors)
- Users can “reserve” the elevator
- The elevator serves all people in order of reservation

We assume at most one user (or group of users) per each “trip”, and they all need to go to the same floor
Design considerations

- How do you encode at which floor the elevator is?
  1. One different state per each floor
     - Does not scale well; for 100 floors building, we need 100 states!
  2. The floor is encoded as an extended state, i.e. a variable \( cf \)
     - It scales, but more difficult to design
  3. It always depends on what we want to describe!

- Which events do we have?
  - An user press a button to “reserve” the elevator, setting variable \( rf \)
  - An user inside the elevator presses the button to change floor, setting variable \( df \)
First design

```
elevator_machine
<<machine>>

Idle
  - timeout
  - reserve [cf == rf]
    - Move To Reserve
      - motor stop
    - reserve [cf == rf]
      - Ready to Load
        - press button
      - Destination reached
        - motor stop

Destination reached

Move to destination
```
Outline

1. Finite State Machines (FSMs)
   - Introduction
   - Moore and Mealy machines
   - State Diagrams
   - Example
   - Mealy machines
   - Exercise

2. Non deterministic FSMs
   - Non determinism
   - Exercise

3. Hierarchical Finite State Machines
   - Problems with FSMs
   - H-FSM specification

4. The Elevator Example
   - Simple FSM
   - Improved design
Doors

- The previous design does not capture all aspects of our systems
- Let’s start to add details by adding the description of how the doors behave
- Abstraction level
  - The level of details of a design depends on what the designer is more interested in describing with the specification
  - In the previous design, we were not interested in describing all aspects, but only on giving a few high-level details
  - The design can be refined by adding details when needed
The doors submachine

door_machine
<<machine>>

doors_closed

open doors

opening

close_end

closing

close doors

open_end

doors_open

open doors
The elevator, second design
Putting everything together

Global Elevator

Idle
- entry / close doors
  - reserve [c != rf]
  - timeout

Move To Reserve
- motor stop
- go

Ready to Load
- entry / open doors
- motor stop

Destination reached
- entry / open doors
- reserve [cf == rf]
- timeout

Move to destination
- entry / close doors

doors_closed
- open doors
- close_end

opening
- open_end

closing
- close doors

doors_open