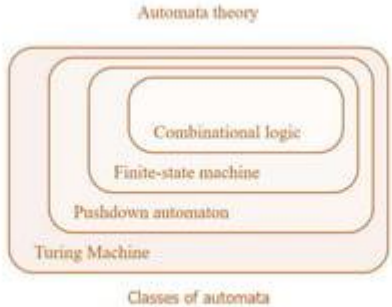
Finite State Machine

- A **finite-state machine** (**FSM**) or **finite-state automaton** (**FSA**, plural: *automata*), **finite automaton**, or simply a **state machine**, is a mathematical model of computation.
- The FSM can change from one state to another in response to some inputs; the change from one state to another is called a *transition*.
- An FSM is defined by a list of its states, its initial state, and the inputs that trigger each transition.
- Finite-state machines are of two types – deterministic finite-state machines and non-deterministic finite-state machines.
- A deterministic finite-state machine can be constructed equivalent to any non-deterministic one.

- The finite-state machine has less computational power than some other models of computation such as the Turing machine.
- The computational power distinction means there are computational tasks that a Turing machine can do but an FSM cannot.
- This is because an FSM's memory is limited by the number of states it has. FSMs are studied in the more general field of automata theory.



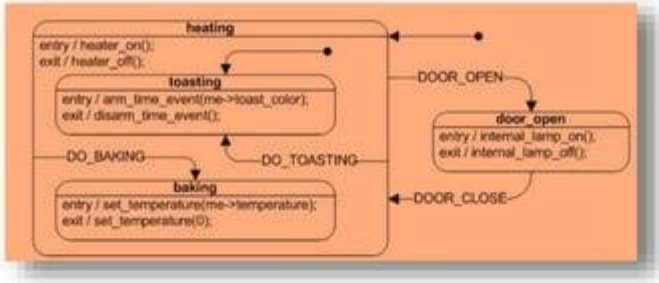
CONCEPTS AND TERMINOLOGY

- A *state* is a description of the status of a system that is waiting to execute a *transition*.

A transition is a set of actions to be executed when a condition is fulfilled or when an event is received.

- For example, when using an audio system to listen to the radio (the system is in the "radio" state), receiving a "next" stimulus results in moving to the next station.
- When the system is in the "CD" state, the "next" stimulus results in moving to the next track. Identical stimuli trigger different actions depending on the current state.
- In some finite-state machine representations, it is also possible to associate actions with a state:
  1. an entry action: performed *when entering* the state, and
  2. an exit action: performed *when exiting* the state.

REPRESENTATIONS



STATE/EVENT TABLE

- Several state-transition table types are used.
- The most common representation is shown below: the combination of current state (e.g. B) and input (e.g. Y) shows the next state (e.g. C).
- The complete action's information is not directly described in the table and can only be added using footnotes.
- An FSM definition including the full actions information is possible using state tables (see also virtual finite-state machine).

State-transition table

Current state Input	State A	State B	State C
Input X	...	...	...
Input Y	...	State C	...
Input Z	...	...	...

UML STATE MACHINES

- The Unified Modeling Language has a notation for describing state machines.
- UML state machines overcome the limitations of traditional finite-state machines while retaining their main benefits.
- UML state machines introduce the new concepts of hierarchically nested states and orthogonal regions, while extending the notion of actions.
- UML state machines have the characteristics of both Mealy machines and Moore machines.
- They support actions that depend on both the state of the system and the triggering event, as in Mealy machines, as well as entry and exit actions, which are associated with states rather than transitions, as in Moore machines.

SDL STATE MACHINES

- The Specification and Description Language is a standard from ITU that includes graphical symbols to describe actions in the transition:
  1. send an event
  2. receive an event
  3. start a timer
  4. cancel a timer
  5. start another concurrent state machine
  6. decision
- SDL embeds basic data types called "Abstract Data Types", an action language, and an execution semantic in order to make the finite-state machine executable.

USAGE

- In addition to their use in modeling reactive systems presented here, finite-state machines are significant in many different areas, including electrical engineering, linguistics, computer science, philosophy, biology, mathematics, video game programming, and logic.
- Finite-state machines are a class of automata studied in automata theory and the theory of computation.
- In computer science, design of hardware digital systems, software engineering, compilers, network protocols, and the study of computation and languages.

MOORE MACHINE

- The FSM uses only entry actions, i.e., output depends only on state. The advantage of the Moore model is a simplification of the behaviour.
- Consider an elevator door. The state machine recognizes two commands: "command_open" and "command_close", which trigger state changes. The entry action (E:) in state "Opening" starts a motor opening the door, the entry action in state "Closing" starts a motor in the other direction closing the door.
- States "Opened" and "Closed" stop the motor when fully opened or closed.
- They signal to the outside world (e.g., to other state machines) the situation: "door is open" or "door is closed".

MEALY MACHINE

- The FSM also uses input actions, i.e., output depends on input and state. The use of a Mealy FSM leads often to a reduction of the number of states.
- The example in figure 9.1.1.79.1.1.7 shows a Mealy FSM implementing the same behaviour as in the Moore example (the behaviour depends on the implemented FSM execution model and will work, e.g., for virtual FSM but not for event-driven FSM).
- There are two input actions (I:): "start motor to close the door if command_close arrives" and "start motor in the other direction to open the door if command_open arrives".
- The "opening" and "closing" intermediate states are not shown.

SEQUENCERS

- **Sequencers** (also called **generators**) are a subclass of acceptors and transducers that have a single-letter input alphabet.
- They produce only one sequence which can be seen as an output sequence of acceptor or transducer outputs.

DETERMINISM

- A further distinction is between **deterministic** (DFA) and **non-deterministic** (NFA, GNFA) automata. In a deterministic automaton, every state has exactly one transition for each possible input.
- In a non-deterministic automaton, an input can lead to one, more than one, or no transition for a given state.

- The powerset construction algorithm can transform any nondeterministic automaton into a (usually more complex) deterministic automaton with identical functionality.
- A finite-state machine with only one state is called a "combinatorial FSM".
- It only allows actions upon transition *into* a state.
- This concept is useful in cases where a number of finite-state machines are required to work together, and when it is convenient to consider a purely combinatorial part as a form of FSM to suit the design tools.

ALTERNATIVE SEMANTICS

- There are other sets of semantics available to represent state machines.
- For example, there are tools for modeling and designing logic for embedded controllers.
- They combine hierarchical state machines (which usually have more than one current state), flow graphs, and truth tables into one language, resulting in a different formalism and set of semantics.
- These charts, like Harel's original state machines, support hierarchically nested states, orthogonal regions, state actions, and transition actions.

MATHEMATICAL MODEL

- In accordance with the general classification, the following formal definitions are found.
- A *deterministic finite-state machine* or *deterministic finite-state acceptor* is a quintuple $(\Sigma,S,s0,\delta,F)(\Sigma,S,s0,\delta,F)$, where:
- Σ is the input alphabet (a finite non-empty set of symbols);
- S is a finite non-empty set of states;
- s0 is an initial state, an element of SS;
- $\delta\delta$ is the state-transition function: $\delta:S\times\Sigma\rightarrow S\delta:S\times\Sigma\rightarrow S$ (in a nondeterministic finite automaton it would be $\delta:S\times\Sigma\rightarrow P(S)\delta:S\times\Sigma\rightarrow P(S)$, i.e. $\delta\delta$ would return a set of states);
- FF is the set of final states, a (possibly empty) subset of S.

- If an FSM MM is in a state ss, the next symbol is xx and $\delta(s,x)\delta(s,x)$ is not defined, then MM can announce an error (i.e. reject the input).
- This is useful in definitions of general state machines, but less useful when transforming the machine.
- Some algorithms in their default form may require total functions.
- A finite-state machine has the same computational power as a Turing machine that is restricted such that its head may only perform "read" operations, and always has to move from left to right.
- That is, each formal language accepted by a finite-state machine is accepted by such a kind of restricted Turing machine, and vice versa.

A *finite-state transducer* is a sextuple $(\Sigma,\Gamma,S,s0,\delta,\omega)(\Sigma,\Gamma,S,s0,\delta,\omega)$, where:

- $\Sigma\Sigma$ is the input alphabet (a finite non-empty set of symbols);
- $\Gamma\Gamma$ is the output alphabet (a finite non-empty set of symbols);
- SS is a finite non-empty set of states;
- $s0s0$ is the initial state, an element of SS;
- $\delta\delta$ is the state-transition function: $\delta:S\times\Sigma\rightarrow S\delta:S\times\Sigma\rightarrow S$;
- $\omega\omega$ is the output function.

- If the output function depends on the state and input symbol ($\omega:S\times\Sigma\rightarrow\Gamma\omega:S\times\Sigma\rightarrow\Gamma$) that definition corresponds to the *Mealy model*, and can be modelled as a Mealy machine.
- If the output function depends only on the state ($\omega:S\rightarrow\Gamma\omega:S\rightarrow\Gamma$) that definition corresponds to the *Moore model*, and can be modelled as a Moore machine. A finite-state machine with no output function at all is known as a semiautomaton or transition system.
- If we disregard the first output symbol of a Moore machine, $\omega(s0)\omega(s0)$, then it can be readily converted to an output-equivalent Mealy machine by setting the output function of every Mealy transition (i.e. labeling every edge) with the output symbol given of the destination Moore state.

SOFTWARE APPLICATIONS

The following concepts are commonly used to build software applications with finite-state machines:

- Automata-based programming
- Event-driven finite-state machine
- Virtual finite-state machine
- State design pattern

FINITE-STATE MACHINES AND COMPILERS

- Finite automata are often used in the frontend of programming language compilers. Such a frontend may comprise several finite-state machines that implement a lexical analyzer and a parser.
- Starting from a sequence of characters, the lexical analyzer builds a sequence of language tokens (such as reserved words, literals, and identifiers) from which the parser builds a syntax tree.
- The lexical analyzer and the parser handle the regular and context-free parts of the programming language's grammar.