ITS64304 Theory of Computation

School of Computer Science
Taylor's University Lakeside Campus

Lecture 2: Finite State Machines

- Deterministic Finite Automata (DFA)
- Nondeterministic Finite Automata (NFA)

Dr Raja..

Learning outcomes

At the end of this topic students should be able to:

- Identify a formal language for a given DFA or NFA, and PDA
- Design a DFA, NFA and PDA for a given formal language*

* Course Learning Outcome 2

Automata

Automata (such as Finite State Machines) are tools for reasoning about computation

Finite State Machines or Finite automaton*

- Our model of computation has two parts:
  - input and output language
  - Processor
- Examples:
  - Automatic door
  - Coke machine (input money, output drinks + change if right coins inserted)
  - Hardware adder (input bits, output bits representing sum of input)
  - Credit card checker (input card number + expiry date, output yes or no)
  - Compiler (input program, output yes/no answer + compiled code)
  - Operating System (input commands, output could be anything!)

***machine designed to respond to encoded instructions**

Automata

FSAs are a simple method for specifying a process

- have a fixed and very limited amount of memory
- driven by input and halt when the input finishes
- have a current state
- driven by the input and current state to a new state
- produce simple output, often accept/reject

This model is simple, but useful

- applications include lexical analysis in compilers and string search algorithms, automatic door and etc.

Automata

Consider a (simple) Coke machine:

- accepts 20c (t), 50c (f) and $1.00 (d) coins
- if given $1.00 or more it delivers a coke (cheap!)

- States: Fed 0, 20, 40, 50, . . . 100+
- Initial state: Fed 0
- Transitions: e.g. fed 20, insert 20, new state is fed 40
- Final state: Fed 100+

Automata

Symbols

start state

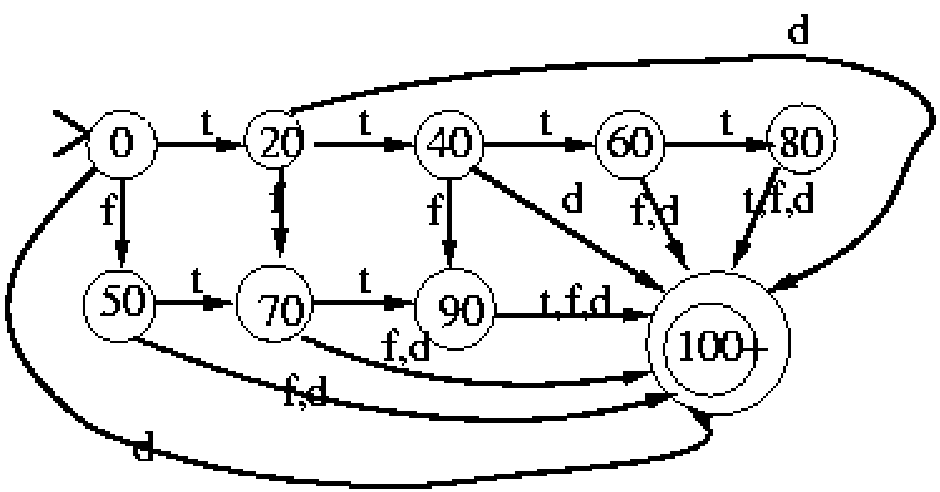
end state

intermediate state

transition

Transition table

	t	f	d
0	20	50	100
20	40	70	100
40	60	90	100
50	70	100	100
60	80	100	100
70	90	100	100
80	100	100	100
90	100	100	100



Finite State Machine

- A finite state machine/ finite automaton has:
  - Finite Set of states
    - A distinguished initial state
    - A number of final states
  - Input is a string
  - String is processed strictly one symbol at a time, left to right
  - Output is either 'yes' or 'no'
  - The automaton reads one symbol and then enters a new state that depends only on the current state and the input symbol. (no choice)
      - **Deterministic finite automaton**
- An input string is accepted if the machine is in a final state when the input finishes; otherwise it is rejected
- The language accepted by the machine is the set of strings it accepts

FSM

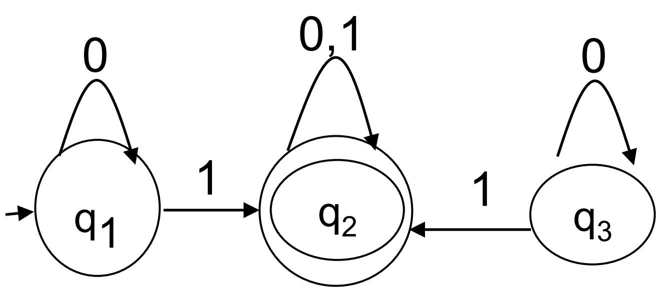
Definition: A **deterministic finite automaton** is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$ where

- Q is a finite set of states
- Σ is an alphabet
- $q_0 \in Q$ is the initial state
- $F \subseteq Q$ is the set of final states
- δ, the transition function maps $Q \times \Sigma \to Q$

FSM Example

- A finite automaton called M_1
  - $M_1 = (Q, \Sigma, \delta, q_1, F)$, where
    - $Q = \{q_1, q_2, q_3\}$
    - $\Sigma = \{0, 1\}$
    - δ is described as
    - q_1 is the start state
    - $F = \{q_2\}$ Final state

	0	1
q_1	q_1	q_2
q_2	q_2	q_2
q_3	q_3	q_2



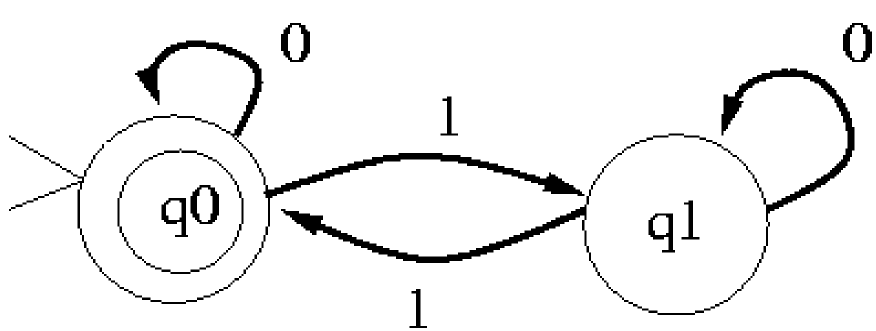
FSM - DFA examples

- Let M be the DFA $(Q,\sum,\delta,s,F)$ where
  - $Q = \{q_0,q_1\}$, $\sum = \{0, 1\}$, $s = q_0$, $F = \{q_0\}$ and δ is

δ	0	1
q_0	q_0	q_1
q_1	q_1	q_0

- L(M) is strings over {0, 1} which contain an even number of 1's

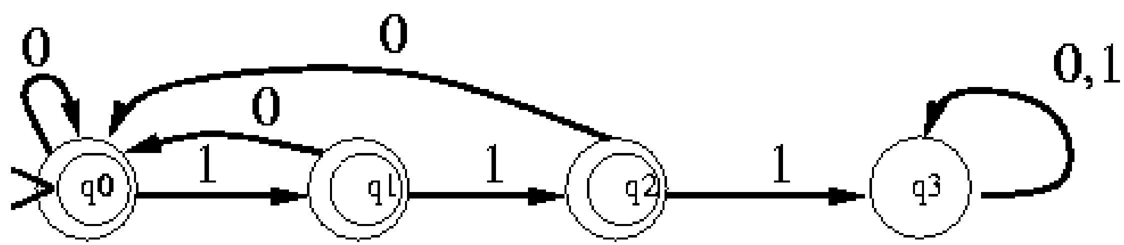
- This is much easier to understand from a state diagram



FSM - DFA examples

- Consider a machine M which accepts strings over {0, 1} which **do not** contain three consecutive 1's, i.e.
  - $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{0, 1\}$, $s = q_0$,
  - $F = \{q_0, q_1, q_2\}$ and δ is

δ	0	1
q_0	q_0	q_1
q_1	q_0	q_2
q_2	q_0	q_3
q_3	q_3	q_3

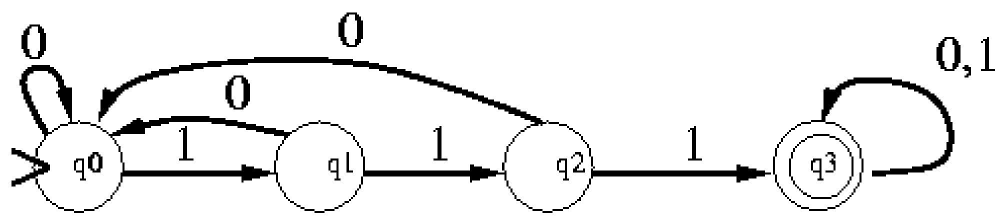


- Note that q_0,q_1 and q_2 are all final states, and that q_3 is a dead state; once computation reaches q_3, it stays there from then on.

FSM - DFA examples

- Here is a machine which accepts strings over {0, 1} which **do contain** three consecutive 1's:
  - $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{0, 1\}$, $s = q_0$,
  - $F = \{q_3\}$ and δ is

δ	0	1
q_0	q_0	q_1
q_1	q_0	q_2
q_2	q_0	q_3
q_3	q_3	q_3



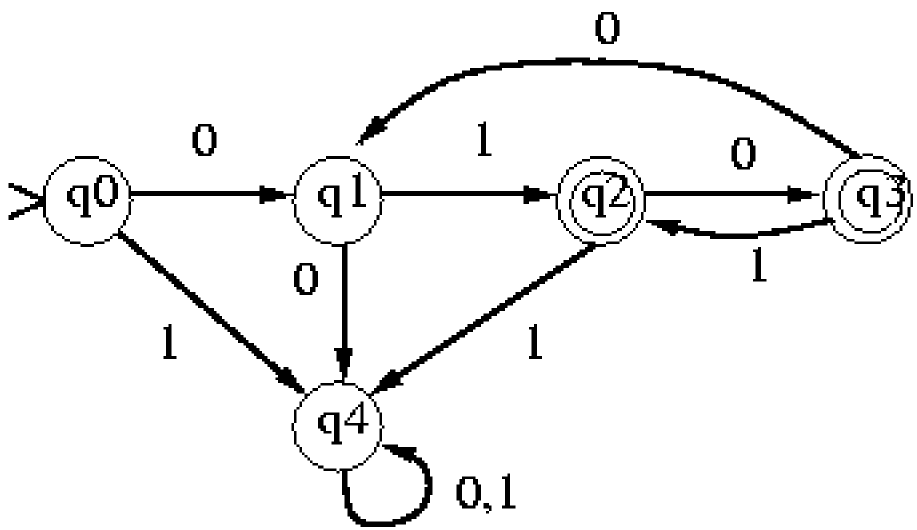
- Hence it is simple to convert a DFA into a DFA for the complement of the language.

Try this….

- Construct DFAs for the following languages over {a, b}:
  - {w | w has at least three *a*’s}
  - {w | w has at least two *b*’s}

Non-determinism

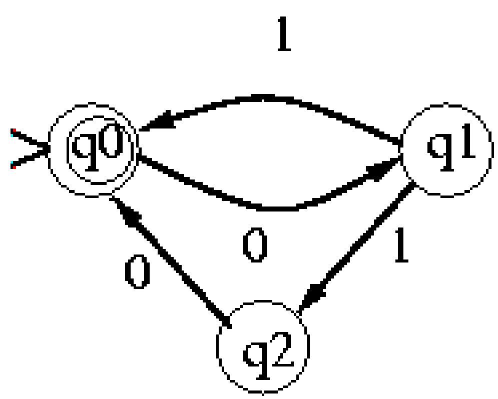
- DFAs are precise, but can be overly “clumsy”. Consider the following DFA for $(01 \cup 010)^*$:



- This is the smallest DFA which accepts this language.

Non-determinism

- By not fully specifying δ, and <u>**not** insisting on a unique new state for each old state and character</u> we can simplify a finite automata.



- This introduces non-determinism, i.e. unspecified choices, into the machine.

Non-determinism

- Note that there is no restriction on transitions here
- A string is accepted if there is some way finishing in a final state;
  - not all computational paths will work
- Note that there is no state to enter in the above machine if the first character is 1;
  - here the machine just rejects the string immediately

Non-determinism

Hence the computation cycle is now:

1. (Same) Read current input symbol x. Halt if end-of-input.
2. (NEW) Choose a new state from x and the current state. If there are no such states, halt with failure.
3. (Same) Move the tape head one position to the right.

- Note that step 2 is now nondeterministic, and that the machine can halt with failure before the end of the input is reached.

- Acceptance only requires that there is at least one computation which succeeds.

Nondeterministic Finite Automata (NFA)

Definition: A non-deterministic finite automaton is a quintuple M = (Q, $\sum$, δ, q_0,F) where

- (same) Q is a finite set of states
- (same) $\sum$ is an alphabet
- (same) $q_0 \in$ Q is the initial state
- (same) F $\subseteq$ Q is the set of final states
- (new) δ, the transition relation is a subset of

  Q x ($\sum \cup$ {e}) x Q

- **DFA** specifies exactly one transition for each combination of state and input symbol while **NFA** allows zero, one or more transitions

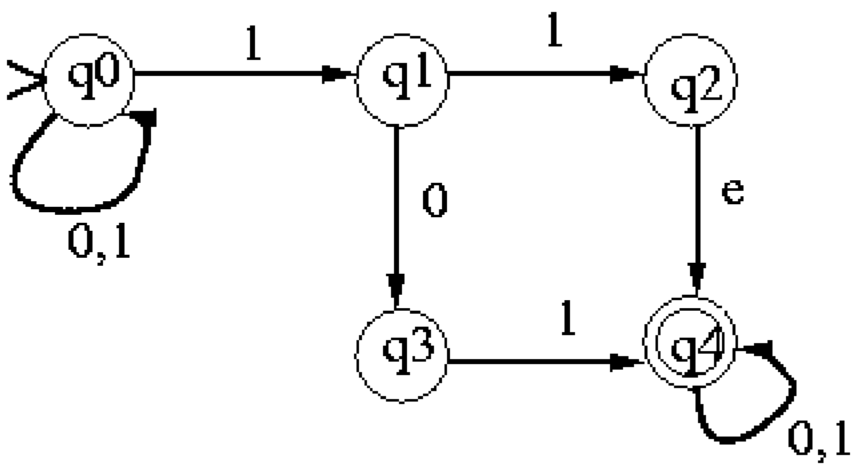
NFA

Configurations are much as before, with two new features:

1. For a transition labeled with e, no input is read
2. For a given configuration (q, w) there can be 0, 1, or several new possible configurations.

- M accepts w if there is at least one computation that succeeds

NFA



- Note that

  $(q_0, 1010101) \vdash M(q_0, 010101)$
  $\vdash M(q_0, 10101)$
  $\vdash M(q_0, 0101)$
  . . .
  $\vdash M(q_0, e)$

(q0, 1010101) |- M(q1, 010101)
|- M(q3, 10101)
|- M(q4, 0101)
. . .
|- M(q4,e)

Read your lesson materials

Look at the Tutorials on Finite Automata, prepare for it…..