Chap 1.1 – Finite Automata

- **Automata**: (Greek for “self-acting”) Device that
  - Performs its actions at (usually fixed) periodic intervals (Called a **Clock**)
  - With the change to the next interval called a **tick**
  - Accepts strings of input data one per tick
  - Optionally generates an output one per tick
  - Can be associated with either state or edge
  - Carries over memory of the **state** of its computation from tick to tick
  - Follows a stored set of **transition rules** that determines for each input & current state:
    - what is new state, what is output

- **State**:
  - Dictionary: “particular condition that something is in at a specific time”
  - For automata: Sum total of all information about computation that may affect what it does next
    - Corresponds to “memory”
  - Example: p. 32 – automatic door opener
Finite Automata (FA) a.k.a Finite State Machine

- Number of different states that system can be in is fixed
- Equivalent to a finite (and small) amount of memory
- Transition rules can only specify from one of these states to another
- For now only one kind of output: “Yes” or “No”
  - Alternatively “Accept” or “Reject”

State Diagram: Graph representation of a FA

- One “labelled vertex” per state
  - Label is name of state
- “Labelled Edge” represents a transition rule
  - Source vertex is state FA is in before a tick
  - Edge label is symbol that was on input
  - Target vertex is state the FA goes into next
- If multiple transition rules go between same 2 states
  - Draw just one edge
  - With label = concatenation of all symbols from rules

Start State: state FA is to be in when it is turned on
- Specified by an edge with no source

Accepting State: when entered, outputs “yes”
- Double circle around state

FA “accepts” or “rejects only when last input processed
• **Deterministic Finite Automata (DFA):** *Exactly one* transition rule defined for each combination of state and input

• **Nondeterministic Finite Automata (NDFA):** *(next class)*
  • *More than 1 rule* possible per state & input
  • But only one taken at a time
    • Which will be discussed later

• **P. 33: Transition table** D:
  • 1 column for each possible input symbol
  • 1 row for each possible state
  • Contents of a cell of D: next state

• DFA Examples:
  • *(p. 32-33)* has transition table
  • *(p. 32)* has state diagram with start and accepting states
  • *(p. 36)* Ex. 1.6 M₁: *(Figs. 1.4 & 1.6)* accepts any string with an even number of 0’s after the last 1 *(where no 0s is an even number)*


• **P. 35. Formal Definition of a FA M** is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\)
  • \(Q\): finite set of states
  • \(\Sigma\): finite set of symbols called **alphabet**
  • \(\delta\): \(Q \times \Sigma \rightarrow Q\) called **transition function**
    • domain is pair of current_state and Current_input
    • range is from \(Q\) (new_state)
  • \(q_0 \in Q\) designated as **start state**
  • \(F \subseteq Q\) is set of accepting states

• **P. 40 Formal Definition of a Computation**:
  • Given “machine” \(M = (Q, \Sigma, \delta, q_0, F)\)
  • And \(w = w_1w_2 \ldots w_n\) a string from \(\Sigma\)
  • **M accepts \(w\)** if \(w\) causes a sequence of \(n+1\) states \(r_0, r_1, \ldots\)
    \(r_i, r_{i+1}, \ldots r_n\)
  • \(r_0 = q_0\),
  • \(\delta(r_i, w_{i+1}) = r_{i+1}\) for \(i = 0\) to \(n-1\)
  • \(r_n \in F\) (key – in an accepting state after last input)

• **M recognizes** language \(A\) if
  • \(A\) is a language over \(\Sigma\) (i.e. \(A\) is a subset of \(\Sigma^*\))
  • For all strings \(w\) in \(A\), \(M\) accepts \(w\)
  • For all strings \(w\) **not** in \(A\), \(M\) **does not accept** \(w\)
• Examples of machines that recognize languages
  • (p. 36) Ex. 1.7 M₂: end in “1”
  • (p. 38) Ex 1.9 M₃: either empty or end with a “0”
  • (p. 38) Ex 1.11 M₄: start or end with “a”, or “b”
  • (p. 39) Ex 1.13 M₅: sum of inputs after a reset = 0 mod 3
  • (p. 40) Ex 1.15 M₆: sum of inputs after a reset = 0 mod i
• (p. 41-43) – tips for designing FAs