**Finite Automata**

* **Deterministic finite automata (DFA)**
  + Comprised of
    - A finite set of states – including one start state and at least one final states
    - A finite set of input symbols – the alphabet
    - A finite set of transitions from one state to another based on input (the label)
  + Can reject or accept input
  + Deterministic – each edge leaving a state has a unique label
  + No explicit error state – if there is no edge leaving a state that has label matches the input, then it’s an error
  + Ex. M:
    - → S → (b) b → (e) be → (q) beq^
      * b → (n) bn → (e) bne^
  + Formal definition – a DFA is a 5-tuple (∑, Q, q0, A, δ) where
    - ∑ = finite alphabet
      * e.g. ∑ = {b, e, n, q}
    - Q = finite set of states
      * e.g. Q = {S, b, be, bn, beq, bne}
    - q0 = start state
      * e.g. q0 = {S}
    - A = set of accepting states
      * The language accepted by the DFA M is L(M)
      * e.g. A = {beq, bne} = L(M)
    - δ = transition function that maps (state, symbol) pairs → a state
      * e.g. δ(S, b) = b; δ(b, e) = be; δ(b, n) = bn; δ(be, q) = beq; etc.
    - E.g. L(M) = {bne, beq} = A
  + Pseudocode:
    - Input: c1, c2, … cn
    - State ← q0

For each ci in input

State ← δ(state, ci)

Return whether state ∈ A

* + δ(state, c) can be implemented as a table
  + Blank means error

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **δ** | **b** | **e** | **n** | **q** |
| **S** | b |  |  |  |
| **b** |  | be | bn |  |
| **bn** |  | bne |  |  |
| **bne** |  |  |  |  |
| **be** |  |  |  | beq |
| **beq** |  |  |  |  |

* **Non-deterministic finite automata (NFA)**
  + Two or more edges leaving the same state can have the same label
  + A language is accepted if at least one path leads to a final state
  + Can be converted to an equivalent DFA
  + Can be in multiple states at the same time
  + 2^Q = power set of Q – all possible subset of Q
    - |2^Q| = 2^|Q|
    - e.g. Q = {a, b, c}; 2^Q = {{}, {a}, {b}, {c}, {ab}, {ac}, {bc}, {abc}}
  + δ maps (state, symbol) → a power set of states
  + Pseudocode:
    - Input c1, c2, … cn
    - States ← {q0}

For each ci in input

S’ = {}

For each s in states

S’ = S’ ∪ δ(s, ci)

States ← S’

Return whether states ∩ A is (not) empty

* Transducer – used to transform input
  + For each transition, read an input character and also output a character
* ε-NFA – allows transitions between states without any input
* **Converting RE to ε-NFA**
  + If RE = ∅ then ε-NFA has no accepting state
  + If RE = ε then ε-NFA only accepts ε
  + If RE = a where a ∈ ∑ then ε-NFA has accepting state that can be reached with transition ‘a’
  + If RE is in the form E1E2 (concatenation)
    - Convert accepting states of E1 into non-accepting
    - Link with start state of E2 via ε-transitions
    - i.e. expressions & automata occur in sequence
  + If RE is in the form E1|E2 (union)
    - Create new start state and link to start states of E1 and E2 via ε-transitions
    - i.e. expressions & automata occur in parallel
  + If RE is in the form E\* (repetition)
    - Link accepting states of E to its start state via ε-transitions
    - i.e. expressions & automata occur in cycle
* **Maximal Munch scanning**
  + Input = c0c1c2 … ck-1
  + Check next state based on ci
  + Keep going until error state is reached – look back at previous state
    - If it was not a final state → error
    - If it was whitespace → ignore
    - If it was an accepting state → output token
  + Then return to start state (scan next token)
* Implementing a scanner:
  + Describe each set of tokens with a regular expression
  + Create an NFA for each regular expression
  + Combine NFAs into a large one using ε-transitions
  + Convert NFA into DFA