Finite Automata and Regular Languages

FUNDAMENTALS

Alphabet: alphabet is a finite non-empty set of symbols
Σ denotes an alphabet

String:

- a string over an alphabet 'A' is a finite ordered sequence of symbols from 'A'. The length of the string is the number of symbols in string with repetitions counted
- an empty string denoted by 'ε', is the (unique) string of length zero.
- If S and T are sets of strings then ST = {xy | x ϵ S and y ϵ T }
- Given an alphabet A,
    - A^o = { ε}
    - A^{n+1} = A.A^n
    - . . .
    - A^* = U(n = 0 to infinity) A^n

LANGUAGES

- A language 'L' over Σ is any finite or infinite set of strings over Σ.
- The elements in L are strings – finite sequences of symbols
- A language which does not contain any elements is called an 'empty language'
- Empty language resembles empty set => { } = Φ ≠ ε
- A language L over an alphabet A is subset of A^* i.e. L (A^*
- An empty language is a language that does not accept any strings including ε. (->O)
- A language which only accepts ε (->OO (concentric))

OPERATIONS

Operations on strings

1. **Concatenation**: Combines two strings by putting one after the other (a.b)
    - Concatenation of empty string with any other string gives the string itself

2. **Substring:** If 'w' is a string, then 'v' is a substring of 'w' if there exists string x and y such that w = xvy. 'x' is called the prefix and 'y' is called the suffix of w.

3. **Reversal:** if 'w' is a string, then w^R is reversal of string spelled backwards.
    - $x=(x^R)^R$
    - $(xz)^R = z^R.x^R$

4. **Kleen star operation:** Let 'w' be a string w^* is set of strings obtained by applying any number of concatenations of w with itself, including empty string.
    - Example: a^* = {ε, a, aa, aaa, ...}

Operations on Langauges

1. **Union:** Given some alphabet Σ, for any two languages, L_1, L_2 oven Σ, the union L_1 union L_2 of L_1 and L_2 is the language L_1 union L_2 = {w ϵ Σ^* | w ϵ L_1 or w ϵ L_2 }
2. **Intersection:** Given some alphabet Σ, for any two languages, L_1, L_2 oven Σ, the intersection $L_1 \cap L_2$ of L_1 and L_2 is the language $L_1 \cap L_2$ = {w ϵ Σ^* | w ϵ L_1 and w ϵ L_2 }

3. **Difference / Relative Complement:** Given some alphabet Σ, for any two languages, L_1, L_2 oven Σ, the differnece L_1- L_2 of L_1 and L_2 is the language $L_1 - L_2 = \{w \in \Sigma^* \mid w \in L_1$ and w does not belong to $L_2 \}$
4. **Concatenation:** Given some alphabet Σ, for any two languages, L_1, L_2 oven Σ, the concatenation $L_1\ L_2$ of L_1 and L_2 is the language $L_1 L_2 = \{w \in \Sigma^* \mid \exists\ u \in L_1, \exists v \in L_2$ and $w = uv\}$
   - **Properties**
   - $L\emptyset = \emptyset = \emptyset L$
   - $L\{\varepsilon\} = L = \{ \varepsilon\} L$
   - $(L_1 \cup \{\varepsilon\}) L_2 = L_1L_2 \cup L_2$
   - $L_1(L_2 \cup \{\varepsilon\}) = L_1L_2 \cup L_1$
   - $L^nL = LL^n = L^{n+1}$
     - $L_1L_2 \neq L_2L_1$
5. **Kleen * Closure (L^*)**: Given an alphabet Σ, for any language L over Σ, the * closure L^* of L is language, $L^* = U_{n>=0} L^n$
6. **Kleen + Closure (L^+)**: Given an alphabet Σ, for any language L over Σ, the kleen $^+$ closure L^+ of L is language, L+ $= U_{n>=1} L^n$

   Properties
   - $\emptyset^* = \{\varepsilon\}$
   - $L^+ = L^*L$
   - $(L^*)^* = L^*$
   - $L^*L^* = L^*$

FINITE STATE MACHINES

FSM is the simplest computational model of limited memory computers. It is designed to solve decision problems, i.e, to decide whether the given input satisfies certain conditions. The next state and output of FSM is a function of input and the current state.

Types of FSM

- Mealy machine
- Moore machine

Finite Automata

- FA is a state machine that comprehensively captures all possible states and transitions that a machine can take while responding to a stream (sequence) of input symbols.
- FA is recognizer of ‘regular languages’

State Machine

- Finite state machines
  - Mealy machine
  - Moore machine
- Finite automata
  - DFA
  - NFA
  - epsilon – NFA

Types of Finite Automata

1. **Deterministic Finite Automata**
   - ○ DFA can exist in only one state at a time
   - ○ DFA is defined by 5-tuple : {Q, Σ, q_0, F, δ}
     - ▪ Q → Finite number of states (elements)
     - ▪ Σ → Finite set of symbols (alphabets)
     - ▪ q_o →Start/Initial state
     - ▪ F → Set of final states
     - ▪ δ → Transition function, which is a mapping between
       - $\delta: Q \times \Sigma \rightarrow Q$
   - ○ How to use DFA

Transition Diagram

State machines are represented by directed graphs called transition (state) diagrams.
Vertices : states
Arcs : transition
Double concentric circles: Final states

Transition Table

Transition function can be represented by tables.
Note: minimum number of states for k-divisibility is k-states

2. Non-deterministic Finite Automata

- The machine can exist in multiple states at the same time
- Each transition function maps to a set of states
- NFA is defined by 5-tuple: {Q, Σ, q_0, F, δ}
  - ▪ Q → Finite number of states (elements)
  - ▪ Σ → Finite set of symbols (alphabets)
  - ▪ q_o →Start/Initial state
  - ▪ F → Set of final states
  - ▪ δ → Transition function, which is a mapping between
    - $\delta: Q \times \Sigma \rightarrow 2^Q$
- How to use NFA

Difference between NFA and DFA

DFA	NFA
1. All transitions are deterministic i.e., each transition leads to exactly one state	1. Transitions could be non-deterministic i.e., a transition could lead to a subset of states
2. For each state, the transition on all possible symbols should be defined	2. For each state, not all symbols necessarily have to be defined
3. Accepts input if the last state is in F	3. Accepts input if one of the last states is in F.
4. Practical implementation is feasible	4. Practical implementation has to be deterministic. It needs conversion from NFA to DFA

Relation between DFA and NFA

1. A language L is accepted by a DFA if and only of it is accepted by NFA
2. DFA is a special case of NFA
3. Every language accepted by NFA is also accepted by a DFA. $D_f = N_f$

NFA WITH Є – MOVES

- Є-transitions in finite automata allows a state to jump to another state without consuming any input symbol

Conversion and equivalence:
Є – NFA → NFA → DFA

NFA without Є – moves:

- Two FA, $N_Є$ and N are said to be equivalent, if $L(N_Є) = L(N)$ i.e., any language described by some $N_Є$ there is a N that accepts the same language.
- For $N_Є$ = {Q, Σ, q_0, F, δ} and N = {Q, Σ`, $q_0$, F`, δ`}, find
- δ`(q, a) = Є – closure (δ (Є-closure(q), a))
- F` = {F U (q_0)}, if Є – closure (q_0) contains a member of F=F, otherwise
- *Note: when transforming Ne to N, only transitions are required to be changes and states remains same.*

CONVERSION OF NFA TO DFA

Let NFA be defined as N = {Q_N, Σ, q_0, F_N, $δ_N$}
The equivalent DFA, D = {Q_D, Σ, q_0, F_D, $δ_D$}

Step 1: $Q_D = 2^{QN}$, i.e., Q_D is set of all subsets of Q, i.e., it is power set of Q_N
Step 2: F_D is set of subsets S of Q_N such that S ∩ F_N ≠ Ø, i.e., F_D is all sets of N's states that include once accepting state of N.
Step 3: For each set, S <= Q_N and for eacg input symbol a in
Σ : $δ_D(S,a) = U_{P ЄS}\ δ_N(P, a)$
That is to compute $δ_D(S, a)$, look at all states P in S, see what states N goes to starting from P on input a, and take union of all those states.

Note: For any NFA, N with 'n' states, the corresponding DFA can have 2^n states.

MINIMIZATION OF DFA

Minimization of DFA means that we are minimizing the representation of a FA to it's least form. Here basically we replace multiple states with a single state without disturbing the representation. In a DFA states p and q are equivalent when:

δ(p, w) Є F => δ(q, w) Є F and δ(p, w) not Є F => δ(q, w) not Є F
if len(w) = 0 and p and q follow the above property: q and p are 0-equivalent
...
if len(w) = n and p and q follow the above property: p and q are n-equivalent

Therefore, instead of having two states, we can combine them into one state and decrese the number of states in the final answer.

We are using Partitioning Method here.

Step1: Identify the start and final state
Step2: If there is any state which is unreachable from the initial state, delete it.
Step3: draw the state transition table
Step4: Find the 0-equivalent sets. Separate non-final states from final states
Step5: Find the 1-equivalent sets. Separate the non equivalent from equivalent
Step5: Keep finding the n-equivalent sets until the next equivalent calculation is identical to the previous one.
Remove the dead states

EQUIVALENCE BETWEEN NFA AND DFA

There is a DFA for any NDA, i.e.,
L(D) = L(N).

Construction:

- In DFA or NFA, whenever an arrow is followed, there is a set of possible states. This set of states is a subset of Q.
- Track the information about subsets of states that can be reached from the initial state after following arrows.
- Consider each subset of states of NFA as a state of DFA nad every subset of states containing a final state as a final state of DFA.