Introduction to Automata Theory

Unit (1)

- **Automaton:** an abstract computing device. (**Note:** A "device" need not even be a physical hardware!)
- **Alphabet:** is a set of symbols.
- **Sentences:** are string of symbols.
- **Language:** is a set of sentences.
- **Language:** is a collection of sentences of finite length all constructed from a finite alphabet of symbols.
- **Grammar:** is a finite list of rules defining a language.
- **Grammar:** can be regarded as a device that enumerates the sentences of a language" - nothing more, nothing less.
- ✓ An alphabet is a finite, non-empty set of symbols.
- ✓ We use the symbol ∑ (sigma) to denote an alphabet.
- ✓ A string or word is a finite sequence of symbols chosen from ∑.
- ✓ Empty string is ε (or "epsilon").
- ✓ L is a said to be a language over alphabet ∑, only if $L \subseteq \Sigma^*$. (This is because Σ^* is the set of all strings (of all possible length including 0) over the given alphabet ∑).
- **Ø:** denotes the Empty language.
- ✓ Let L = {ε} = Is L=Ø (false).
- ✓ Given a string $w \in \Sigma^*$ and a language L over ∑, decide whether or not $w \in L$.

Unit (2)

Applications for Finite Automata:

1. Software for designing and checking the behavior of digital circuits.
2. Lexical analyzer of a typical compiler.
3. Software for scanning large bodies of text (e.g., web pages) for pattern finding.
4. Software for verifying systems of all types that have a finite number of states (e.g., stock market transaction, communication/network protocol).

- ✓ "If $y \geq 4$, then $2^y \geq y^2$."

- **Theorem:** A major result.
- **Lemma:** An intermediate result that we show to prove a larger result.
- **Corollary:** A result that follows from an already proven result.
- **Theorem:** The height of an n-node binary tree is at least floor (lg n).
- **Lemma:** Level i of a perfect binary tree has 2^i nodes.
- **Corollary:** A perfect binary tree of height h has $2^{h+1}-1$ nodes.

Section (1)

- **Automata:** Plural of automaton.
- **Automata:** Self-operating machine.
- **Automata:** Something that works automatically.
- ✓ **The theory provides principles and concepts of the compiler & programming languages and digital design.**
- **Compiler:** is a program takes a program written in a source language and translates it into an equivalent program in a target language.
- **Power set:** set of all subsets. Or $|P(A)| = 2^{|A|}$
- ✓ **AxB ≠ BxA.**

Unit (3)

- ✓ **Informally, a state diagram that comprehensively captures all possible states and transitions that a machine can take while responding to a stream or sequence of input symbols.**

Type of Finite Automaton (FA):

1. **Deterministic Finite Automata (DFA):**

The machine can exist in only one state at any given time

2. **Non-deterministic Finite Automata (NFA):**

The machine can exist in multiple states at the same time

A Deterministic Finite Automaton (DFA) consists of:

- **Q ==> a finite set of states**
- **∑ ==> a finite set of input symbols (alphabet)**

- q_0 ==> a start state
- F ==> set of accepting states
- δ ==> a transition function, which is a mapping between Q x ∑ ==> Q

A DFA is defined by the 5-tuple:

- {Q, ∑, q_0,F, δ }

✓ Let L(A) be a language recognized by a DFA A. Then L(A) is called a "Regular Language".

✓ The states of a DFA are represented as a circle.

✓ Accepting states are drawn with two circles.

✓ δ (q,w) = destination state from state q on input string w .

✓ δ (q,wa) = δ (δ(q,w), a).

✓ A DFA A accepts string w if there is a path from q_0 to an accepting (or final) state that is labeled by w.

✓ i.e., L(A) = { w | δ(q_0,w) ∈ F }.

✓ I.e., L(A) = all strings that lead to an accepting state from q_0.

A Non-deterministic Finite Automaton (NFA) consists of:

- Q ==> a finite set of states
- ∑ ==> a finite set of input symbols (alphabet)
- q_0 ==> a start state
- F ==> set of accepting states
- δ ==> a transition function, which is a mapping between Q x ∑ ==> subset of Q

An NFA is also defined by the 5-tuple:

- {Q, ∑ , q_0,F, δ }

✓ An NFA accepts w if there exists at least one path from the start state to an accepting (or final) state that is labeled by w.

✓ **$L(N) = \{ w \mid \delta(q_0,w) \cap F \neq \Phi \}$.**

DFA	NFA
1. All transitions are deterministic • Each transition leads to exactly one state 2. For each state, transition on all possible symbols (alphabet) should be defined 3. Accepts input if the last state visited is in F 4. Sometimes harder to construct because of the number of states 5. Practical implementation is feasible	1. Some transitions could be non-deterministic • A transition could lead to a subset of states 2. Not all symbol transitions need to be defined explicitly (if undefined will go to an error state – this is just a design convenience, not to be confused with “non-determinism”) 3. Accepts input if *one of* the last states is in F 4. Generally easier than a DFA to construct 5. Practical implementations limited but emerging (e.g., Micron automata processor)

(THEOREM)

✓ **A language L is accepted by a DFA *if and only if* it is accepted by an NFA.**

(PROOF)

If part:

✓ **Prove by showing every NFA can be converted to an equivalent DFA (in the next few slides…)**

Only-if part is trivial:

✓ **Every DFA is a special case of an NFA where each state has exactly one transition for every input symbol. Therefore, if L is accepted by a DFA, it is accepted by a corresponding NFA.**