Week 05: Transition Graphs (TGs)

- **Relaxing the restriction on inputs**
  Definition: In standard FAs, each transition is for a single input symbol. By relaxing this, transitions can be labeled with multiple symbols or regular expressions.
  Example: A transition like δ(q, a or b) → q1 allows either ‘a’ or ‘b’ to move to state q1.
- **Transition Graphs (TGs)**
  Definition: A transition graph is a labeled directed graph where nodes represent states and edges represent transitions based on input symbols.
  Example: q0 —a→ q1 —b→ q2 represents a machine that accepts "ab".
- **Generalized Transition Graphs (GTGs)**
  Definition: GTGs allow edges to be labeled with regular expressions instead of single input symbols, representing multiple transitions compactly.
  Example: q0 —(a|b)*→ q1 accepts any string of a’s and b’s.
- **Non-determinism**
  Definition: In non-deterministic finite automata (NFA), a single input symbol may lead to multiple next states or even none, allowing multiple possible computation paths.
  Example: δ(q0, a) = {q1, q2} means input ‘a’ can go to either q1 or q2.

Week 06: NFA and Kleene’s Theorem

- **Turning TGs into RE**
  Definition: This is the process of converting a Transition Graph into an equivalent regular expression that generates the same language.
  Example: q0 —a→ q1 —b→ q2 becomes RE: ab.
- **Converting RE into FA**
  Definition: Every regular expression can be converted into a finite automaton (NFA or DFA) that accepts the same language.
  Example: RE = (a|b)* → FA with loops on both ‘a’ and ‘b’.
- **Conversion between DFA and NFA**
  Definition: Every NFA can be systematically converted into an equivalent DFA using subset construction (power set method).
  Example: An NFA with ε-transitions can be turned into a DFA without ε-moves.
- **Kleene’s Theorem**
  Definition: This theorem states that a language is regular if and only if it can be represented by a regular expression or recognized by a finite automaton.
  Example: The language described by RE (a|b)* is regular and can be accepted by a DFA.

Week 07: Mid Term-I Examination

Week 08: Regular Languages

- **Closure Properties**
  Definition: Regular languages remain regular after applying operations such as union, concatenation, and Kleene star.
  Example: If L1 = {a}, L2 = {b}, then L1 ∪ L2 = {a, b} is also regular.
- **Operations**
  Definition: Common operations on languages include union (L1 ∪ L2), intersection (L1 ∩ L2), complement, etc.
  Example: L1 = {a, b}, L2 = {a, c} → L1 ∩ L2 = {a}.

Week 09: Pumping Lemma and Non-Regular Languages

- **Pumping Lemma**
  Definition: A property that all regular languages satisfy. If a language fails this property, it is not regular.
  Example: $L = \{a^n b^n \mid n \geq 0\}$ cannot be pumped ⇒ not regular.
- **Proving Non-Regular Languages**
  Definition: To prove a language is not regular, assume it is regular and apply the pumping lemma to reach a contradiction.
  Example: $L = \{a^n b^n\}$ fails the pumping lemma ⇒ it's not regular.

Week 10: Context-Free Grammars (CFGs)

- **Syntax as a method of defining languages**
  Definition: CFGs define the structure of strings using a set of rules (productions).
  Example: S → aSb | ε defines a language with equal numbers of a's and b's.
- **Symbolism for generative language**
  Definition: CFGs use non-terminal symbols (like S, A) and terminal symbols (like a, b) to generate strings.
  Example: A → aA | b generates strings of a's followed by one b.
- **Trees**
  Definition: Parse trees visually represent how a string is generated from the start symbol using grammar rules.
  Example: Tree for S → aSb → aaSbb → aabb.
- **Ambiguity**
  Definition: A grammar is ambiguous if a string can have more than one valid parse tree.
  Example: "if-else" statements often cause ambiguity in programming languages.

Week 11: Grammatical Format

- **Regular Grammars**
  Definition: A type of grammar where productions are of the form A → aB or A → a. These generate regular languages.
  Example: S → aS | b.
- **Simplification**
  Definition: Removing useless rules, unreachable symbols, and redundant productions to make a grammar simpler.
  Example: Removing a non-terminal that never appears in any derivation.
- **Chomsky Normal Form (CNF)**
  Definition: A grammar is in CNF if all rules are of the form A → BC or A → a, with special handling for ε.
  Example: S → AB, A → a, B → b.

Week 12: Pushdown Automata (PDA)

- **A new format for FAs**
  Definition: PDAs are like finite automata but include a stack for memory, allowing them to recognize context-free languages.
  Example: PDA that accepts L = $\{a^n b^n\}$ by pushing a's and popping for b's.
- **Adding a Pushdown Stack**
  Definition: The stack allows the PDA to remember previous inputs, especially useful for matching nested patterns.
  Example: Push 'a' for every input a, and pop on each b.
- **Defining PDAs**
  Definition: A PDA is defined by states, input symbols, stack symbols, transition rules, a start state, and accepting conditions.
  Example: δ(q, a, Z) = (q, aZ) means push 'a' on top of stack symbol Z.

Week 13: PDA, CFG, and NFA

- **CFG and PDA**
  Definition: Every context-free grammar has an equivalent PDA, and every PDA has an equivalent CFG.
  Example: Grammar S → aSb | ε corresponds to a PDA with stack matching.
- **Conversion between CFG and NPDA**
  Definition: There are standard algorithms to convert CFGs to NPDA and vice versa, preserving language equivalence.
  Example: PDA simulates leftmost derivation of CFG using its stack.

Week 14: Mid Term-II Examination

Week 15: Context-Free Languages (CFLs)

- **Closure Properties of CFLs**
  Definition: CFLs are closed under union, concatenation, and star, but not under intersection or complement.
  Example: If L1 = $\{a^n b^n\}$, L2 = $\{b^n c^n\}$, then L1 ∪ L2 is CFL, but L1 ∩ L2 may not be.
- **Mixing CFG and Regular Languages**
  Definition: The intersection of a context-free language with a regular language is always a context-free language.
  Example: L1 (CFG) = $\{a^n b^n\}$, L2 (DFA) = all strings ending in b → L1 ∩ L2 is CFL.

Guess paper:

Q1: Finite Automata & Transition Graphs (7 marks)

a) Define Finite Automaton and explain its components. (3 marks)
b) Draw a transition graph for the regular expression `(a|b)*abb`. (4 marks)

Q2: Deterministic vs Non-Deterministic FA (6 marks)

a) Differentiate between DFA and NFA with examples. (3 marks)
b) Convert the following NFA to an equivalent DFA:

- States: {q0, q1}
- Input: {a}
- Transitions: $\delta(q0, a) = \{q0, q1\}$, $\delta(q1, a) = \{q1\}$
- Start state: q0
- Final state: {q1} (3 marks)

Q3: Regular Expressions and Kleene's Theorem (7 marks)

a) State Kleene's Theorem. Why is it important in the theory of computation? (3 marks)
b) Convert the following DFA into a regular expression:

- States: q0 → q1 → q2 (Final)
- Transitions: a from q0 to q1, b from q1 to q2 (4 marks)

Q4: Pumping Lemma (6 marks)

Use the Pumping Lemma to prove that the language $\mathbf{L = \{a^n b^n \mid n \geq 0\}}$ is not regular.

Q5: Context-Free Grammar (CFG) (6 marks)

a) Define CFG. Write a CFG that generates all palindromes over the alphabet {a, b}. (3 marks)
b) Draw the parse tree for the string "abba" using your CFG. (3 marks)

Q6: Pushdown Automata (PDA) (6 marks)

a) Define PDA and explain the role of the stack. (2 marks)
b) Design a PDA that accepts the language **L = {$a^n b^n$ | n ≥ 1}**. Show its transition diagram. (4 marks)

Q7: Closure Properties of CFL and Regular Languages (6 marks)

a) List any three closure properties of regular languages with examples. (3 marks)
b) Explain how the intersection of a CFG and regular language remains context-free with an example. (3 marks)

Answers

Q1: Finite Automata & Transition Graphs (7 marks)

a) Define Finite Automaton and explain its components. (3 marks)
A **Finite Automaton (FA)** is a theoretical machine used to recognize regular languages. It consists of:

- **Q**: A finite set of states
- **Σ (Sigma)**: A finite set of input symbols (alphabet)
- **δ**: A transition function (Q × Σ → Q)
- **q_0**: Start state (q_0 ∈ Q)
- **F**: Set of accepting/final states (F ⊆ Q)

b) Draw a transition graph for the regular expression (a|b)*abb. (4 marks)

```
plaintext
CopyEdit
→ (q0) --a,b--> (q0)
(q0) --a--> (q1)
(q1) --b--> (q2)
(q2) --b--> (q3)*
```

This FA accepts strings ending in `abb` over alphabet {a, b}.

Q2: Deterministic vs Non-Deterministic FA (6 marks)

a) Differentiate between DFA and NFA with examples. (3 marks)

Feature	DFA	NFA
Transition	Exactly one per input symbol	Can have multiple or zero

Feature	DFA	NFA
Simplicity	Simpler to simulate	Easier to design
ε-moves	Not allowed	Allowed

Example DFA: δ($q0$, a) = $q1$
Example NFA: δ($q0$, a) = {$q1$, $q2$}

b) Convert the NFA to DFA (3 marks)
Given:

- NFA states: {q0, q1}, input: {a}
- δ(q0, a) = {q0, q1}, δ(q1, a) = {q1}

Subset Construction DFA:

DFA State	Transitions (a)	Accepting?
{q0}	{q0, q1}	No
{q0,q1}	{q0, q1}	Yes

Final DFA has:

- States: {q0}, {q0,q1}
- Start: {q0}
- Final: {q0,q1}

Q3: Regular Expressions and Kleene's Theorem (7 marks)

a) State Kleene's Theorem. Why is it important? (3 marks)
Kleene's Theorem: A language is regular if and only if it can be recognized by a finite automaton or described by a regular expression.

Importance: It establishes the equivalence between automata and regular expressions, allowing conversion between them.

b) Convert DFA to RE: (4 marks)
DFA: q0 --a→ q1 --b→ q2 (Final)

This DFA accepts the string "ab", so the equivalent regular expression is:
RE = ab

Q4: Pumping Lemma (6 marks)

To prove **L = {$a^n b^n$ | n ≥ 0}** is not regular:

Assume: L is regular.
By **Pumping Lemma**, there exists a pumping length `p`.
Let w = $a^p b^p$ (|w| ≥ p)

Split: w = xyz such that:

- |xy| ≤ p → y = a^k (only a's)
- |y| > 0
- For all i ≥ 0, $xy^iz \in L$

Take i = 2:
w' = xyyz = $a^{p+k}b^p$ → #a ≠ #b → w' ∉ L
Contradiction ⇒ L is **not regular**

Q5: Context-Free Grammar (CFG) (6 marks)

a) Define CFG & CFG for palindromes. (3 marks)
A **CFG** is a 4-tuple (V, Σ, R, S) where:

- V = variables (non-terminals)
- Σ = terminals
- R = rules (productions)
- S = start symbol

CFG for palindromes over {a, b}:

```less
CopyEdit
S → aSa | bSb | a | b | ε
```

b) Parse tree for "abba": (3 marks)

```less
CopyEdit
        S
      / | \
     a  S  a
       / \
      b   b
```

Q6: Pushdown Automata (PDA) (6 marks)

a) Define PDA & role of stack. (2 marks)
A **PDA** is a 6-tuple (Q, Σ, Γ, δ, q_0, F) that includes a **stack** for memory.

The stack helps match nested or balanced symbols (e.g., $a^n b^n$), allowing recognition of **context-free languages**.

b) PDA for $L = \{a^n b^n \mid n \geq 1\}$ (4 marks)

States: {q0, q1, q2}, Start: q0, Final: q2
Stack symbols: {Z, A}, Initial stack symbol: Z

Transitions:

- $\delta(q0, a, Z) = (q0, AZ)$
- $\delta(q0, a, A) = (q0, AA)$
- $\delta(q0, b, A) = (q1, \varepsilon)$
- $\delta(q1, b, A) = (q1, \varepsilon)$
- $\delta(q1, \varepsilon, Z) = (q2, Z)$

This PDA pushes A for each a, pops A for each b, accepts if stack returns to Z.

Q7: Closure Properties (6 marks)

a) Three closure properties of regular languages (3 marks):

1. **Union**: If L1 and L2 are regular, then $L1 \cup L2$ is regular.
   Example: $L1 = \{a\}$, $L2 = \{b\} \Rightarrow L1 \cup L2 = \{a, b\}$
2. **Concatenation**: L1L2 is regular if L1 and L2 are regular.
   Example: $L1 = \{a\}$, $L2 = \{b\} \Rightarrow L1L2 = \{ab\}$
3. **Kleene Star**: If L is regular, so is L*.
   Example: $L = \{a\} \Rightarrow L^* = \{\varepsilon, a, aa, aaa, ...\}$

b) CFG ∩ Regular is Context-Free (3 marks):
Let L1 be context-free, L2 be regular. Then **$L = L1 \cap L2$** is context-free.

Example:

- $L1 = \{a^n b^n c^n \mid n \geq 1\} \cap L2$ = strings ending in 'c'
- Resulting L contains strings like `aa bb cc`, which is context-free.