Theory of Computation - Comprehensive Study Notes

1. Automata Theory and Regular Languages

1.1 Finite Automata (FA)

- **Definition**: Simplest model of computation that reads input step-by-step
- **Key features**:
  - Always in one or more of a finite number of states
  - Reads input string symbol-by-symbol
  - Changes state based solely on current state and input symbol
  - No internal memory beyond current state
  - Only accepts/rejects input strings (no output)

1.2 Deterministic Finite Automata (DFA)

- **Formal definition**: 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where:
  - Q: finite set of states
  - Σ: finite alphabet (input symbols)
  - $\delta: Q \times \Sigma \to Q$ (transition function)
  - q_0: start state
  - F: set of accepting states
- **Key restrictions**:
  - Always in exactly one state
  - Each state has exactly one transition for each input symbol
- **Language accepted by DFA**: $L(M) = \{w \mid \hat{\delta}(q_0, w) \in F\}$
- **Regular language**: A language accepted by some DFA

1.3 Non-Deterministic Finite Automata (NFA)

- **Formal definition**: 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where:
  - $\delta: Q \times \Sigma \to P(Q)$ (transition function returns a set of states)
- **Key features**:
  - Can be in multiple states simultaneously
  - Some transitions can be missing
  - String accepted if ANY path leads to an accepting state
- **Transition function for strings**:
  - $\tilde{\delta}(q, \varepsilon) = \{q\}$ for every state q
  - $\tilde{\delta}(q, xa) = \cup_{i=1}^{k} \delta(q_i, a)$ where $\tilde{\delta}(q, x) = \{q_1, q_2, ..., q_k\}$
- **Language accepted by NFA**: $L(M) = \{w \mid \tilde{\delta}(q_0, w) \cap F \neq \emptyset\}$

1.4 NFAs with ε-transitions

- Extends NFA with spontaneous transitions (without reading input)
- **Transition function**: $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \to P(Q)$
- **ε-closure**: Set of all states reachable from q using only ε transitions
- **Transition function extension**:
  - $\tilde{\delta}(q, \varepsilon) = \varepsilon\text{-CLOSURE}(q)$
  - $\tilde{\delta}(q, xa) = \varepsilon\text{-CLOSURE}(\cup_{i=1}^{m} \delta(p_i, a))$ where $\tilde{\delta}(q, x) = \{p_1, p_2, ..., p_m\}$

1.5 Equivalence of FA Types

- **Theorem**: DFAs, NFAs, and NFAs with ε-transitions all recognize exactly the same class of languages (regular languages)
- **Subset Construction** (NFA to DFA conversion):
  - States of DFA = Power set of NFA states
  - Transition function: δ(r, a) = ∪(i=1 to n) δ'(p'i, a) where r = [p'1, p'2, ..., p'n]
  - Accepting states: Any set containing at least one accepting state of the NFA
  - Can have up to 2^n states if NFA has n states

1.6 Regular Expressions

- **Formal definition**:
  1. ∅ is a regular expression with L(∅) = ∅
  2. ε is a regular expression with L(ε) = {ε}
  3. For any a ∈ Σ, a is a regular expression with L(a) = {a}
  4. If r and s are regular expressions, then:
     - r+s is a regular expression with L(r+s) = L(r) ∪ L(s) (union)
     - rs is a regular expression with L(rs) = L(r)L(s) (concatenation)
     - r* is a regular expression with L(r*) = L(r)* (Kleene star)
- **Precedence rules**: * (highest), concatenation, + (lowest)
- **Equivalence**: Regular expressions describe exactly the same languages as finite automata

1.7 Pumping Lemma for Regular Languages

- **Purpose**: Tool to prove languages are NOT regular
- **Statement**: For any regular language L, there exists a positive integer n such that any string w ∈ L with |w| ≥ n can be written as w = xyz where:
  1. |xy| ≤ n
  2. |y| > 0
  3. For all i ≥ 0, xy^iz ∈ L
- **Usage pattern** (proof by contradiction):
  1. Assume L is regular, so pumping lemma applies
  2. Take arbitrary pumping length n
  3. Find a string w ∈ L with |w| ≥ n
  4. Consider all possible divisions w = xyz satisfying conditions 1 and 2
  5. Show there exists an i where xy^iz ∉ L
  6. Conclude L is not regular
- **Example**: L = {0^n1^n | n ≥ 0} is not regular
  - Take w = 0^n1^n (length 2n)
  - For any division with |xy| ≤ n, y must contain only 0's
  - For i = 2, xy^2z has more 0's than 1's, so xy^2z ∉ L
  - Therefore, L is not regular

2. Turing Machines and Computability

2.1 Basic Turing Machine Model

- **Components**:
  - Finite state control (state set Q)
  - One-way infinite tape divided into cells
  - Tape head scanning one cell at a time
  - Input alphabet Σ and tape alphabet Γ (Σ ⊂ Γ)
  - Blank symbol B ∈ Γ (not in Σ)
- **Formal definition**: 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$ where:
  - Q: finite set of states
  - Σ: input alphabet
  - Γ: tape alphabet (Σ ⊂ Γ)
  - $\delta: Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ (transition function)
  - q_0: start state
  - $B \in \Gamma - \Sigma$: blank symbol
  - $F \subseteq Q$: set of accepting states
- **One move consists of**:
  1. Reading symbol in current cell
  2. Changing state
  3. Overwriting symbol in cell
  4. Moving tape head left or right
- **Computation terminates when**:
  1. Machine enters accepting state (string accepted)
  2. Machine halts in non-accepting state (string rejected)
  3. Machine tries to move left off tape (rejected)
  4. Machine never halts (neither accepted nor rejected)

2.2 TM Variations (All Equivalent in Power)

- **Two-way infinite tape TM**: Tape extends infinitely in both directions
- **Multi-tape TM**: Multiple tapes, each with its own head

2.3 Non-Deterministic Turing Machines (NDTM)

- **Definition**: 7-tuple with $\delta: Q \times \Gamma \rightarrow P(Q \times \Gamma \times \{L,R\})$
- **Two equivalent definitions**:
  1. **Standard**: Multiple possible moves at each step
  2. **Verifier**: Two-stage operation (guessing and checking)
- **Time Complexity**:
  - Time to accept: Minimum time over all accepting computations
  - Time complexity T(n): Maximum over all accepted strings of length n

2.4 Turing Machines as Function Computers

- **Encoding integers**: In unary (n is encoded as 0^n)
- **Encoding tuples**: $(i_1, i_2, \ldots, i_k)$ encoded as $0^{\wedge}i_1 10^{\wedge}i_2 1\ldots10^{\wedge}i_k$
- **Function computation**: $f(i_1,\ldots,i_k) = m$ if TM halts with 0^m on its tape

2.5 Decidability and Recognizability

- **Partially decidable language** (recognizable/recursively enumerable):
  - A language L where a TM exists that accepts all strings in L
  - The TM may not halt for strings not in L
- **Totally decidable language** (decidable/recursive):
  - A language L where a TM exists that accepts all strings in L and rejects all strings not in L
  - The TM always halts
- **Unrecognizable language**:
  - No TM can be constructed to accept it

2.6 Church's Thesis

- Every function that conforms to intuitive notion of "computable" can be computed by a Turing Machine
- Modern computers can't compute anything beyond what Turing Machines can compute

3. Undecidability

3.1 Encoding Turing Machines

- TMs can be encoded as binary strings
- Transitions $\delta(q_i, X_j) = (q_k, X_l, D_m)$ encoded as 0^i10^j10^k10^l10^m
- Multiple transitions separated by 11
- Entire encoding wrapped with 111 at start and end

3.2 Diagonalization Language (Not Partially Decidable)

- Let M_i be the i-th TM (based on standard enumeration)
- Ld = $\{w_i \mid M_i$ does not accept $w_i\}$, where w_i is the i-th binary string
- **Proof by contradiction**:
  - Assume Ld is partially decidable, so some TM M_j accepts it
  - Consider whether $w_j \in$ Ld:
    - If $w_j \in$ Ld, then M_j doesn't accept w_j (definition of Ld)
      - But M_j must accept w_j (since it accepts Ld) - contradiction
    - If $w_j \notin$ Ld, then M_j accepts w_j (definition of Ld)
      - But this means $w_j \in$ Ld - contradiction
  - Therefore Ld cannot be partially decidable

3.3 Universal Turing Machine and Universal Language

- Universal TM U simulates any TM M on input w
- Universal Language Lu = $\{\langle M, w\rangle \mid M$ accepts $w\}$
- **Properties**:
  - Lu is partially decidable (by Universal TM)
  - Lu is not totally decidable (proven by reduction from Ld)

3.4 Key Properties of Languages

1. If L is totally decidable, $\bar{L}$ is also totally decidable
2. If both L and $\bar{L}$ are partially decidable, then L is totally decidable
3. If L is partially decidable but not totally decidable, then $\bar{L}$ is not partially decidable

3.5 The Halting Problem

- Halting Language = $\{\langle M, w\rangle \mid M$ halts on input $w\}$
- **Properties**:
  - Partially decidable
  - Not totally decidable
  - Its complement is not partially decidable

3.6 Other Undecidable Problems

1. **Post's Correspondence Problem**:
   - Given two sequences of strings ($w_1,...,w_k$) and ($x_1,...,x_k$), is there a sequence of indices where corresponding strings concatenate to form the same string?
   - Partially decidable but not totally decidable
2. **Integer Solutions to Multivariate Polynomials**:
   - Given a polynomial $P(x_1,...,x_r)$ with integer coefficients, determine if there exist integer values for which P=0
   - Partially decidable but not totally decidable

4. Complexity Theory

4.1 Asymptotic Complexity

- **O Notation**: $f(n) = O(g(n))$ if ∃ positive integers c and n_0 such that $f(n) \leq cg(n)$ for all $n \geq n_0$
- **Purpose**: Estimate time complexity focusing on growth rate
- **Examples**:
  - $5n^3 + 2n^2 + 22n + 6 = O(n^3)$
  - $4n^2 + 3n = O(n^2)$
  - $2^n + n^2 = O(2^n)$

4.2 Time and Space Complexity

- **Time complexity**: Number of steps a TM takes to decide an input
- **Space complexity**: Number of cells visited during computation
- For any instance, space required ≤ time required
- Usually expressed as functions of input size n
- Consider worst-case complexity for inputs of size n

4.3 Time Complexity Classes

- **DTIME(t(n))**: Languages decided by TMs with time complexity O(t(n))
- **Practical impact**: For input size n = 100
  - n: 100ns
  - n^2: 10μs
  - n^3: 1ms
  - 2^n: 10^{22}years

4.4 Class P

- **Definition**: Languages decidable in polynomial time
  - P = $\cup_{k=1}$^∞ DTIME(n^k)
- **Significance**: "Easy" or "tractable" problems
- **Robustness**: Class P is invariant under reasonable modifications to the TM model
  - Multi-tape TM with time O(t(n)) can be simulated by single-tape TM in O(t^2(n))

4.5 Class NP

- **Definition**: Languages decidable by NDTM in polynomial time
  - NP = $\cup_{k=1}$^∞ NTIME(n^k)
- **Characterization**: Problems with solutions verifiable in polynomial time
- **Relationship to P**: P ⊆ NP (whether P = NP is a major open question)
- **Theorem**: If L ∈ NP, there exists a deterministic TM that decides L with time complexity 2^p(n)
  - Every NP problem has at least an exponential-time deterministic solution

4.6 NP-Completeness

- **Polynomial reduction**: A $\leq_p$ B means A can be transformed to B in polynomial time
  - If A $\leq_p$ B and B ∈ P, then A ∈ P
  - Transitive: If A $\leq_p$ B and B $\leq_p$ C, then A $\leq_p$ C
- **NP-Complete language**: Language L such that:
  1. L ∈ NP
  2. Every language in NP is polynomial-time reducible to L
- **Significance**: If any NP-Complete problem is in P, then P = NP

4.7 Important NP-Complete Problems

1. **SATISFIABILITY (SAT)**:
   - First proven NP-complete problem (Cook, 1971)
   - Given Boolean formula φ, is there an assignment that makes φ true?
2. **3SAT**:
   - Variant of SAT where formula is in 3-CNF (clauses with exactly 3 literals)
   - Proved NP-complete by reduction from SAT
3. **VERTEX-COVER**:
   - Given graph G and integer k, is there a set of k vertices touching all edges?
   - Proved NP-complete by reduction from 3SAT
4. **HAMPATH**:
   - Given graph G and vertices v_i, v_j, is there a path visiting all vertices exactly once?
   - Example of problem in NP (solution easily verifiable)
5. **CLIQUE**:
   - Given graph G and integer k, does G contain a clique of size k?
   - A clique is a subset of vertices where every two vertices are connected
   - Proved NP-complete by reduction from 3SAT

4.8 Proving NP-Completeness

To prove a language C is NP-Complete:

1. Show C is in NP
2. Show some known NP-Complete problem B is polynomial-time reducible to C ($B \leq_p C$)