Review and Regular Languages

Exam I

I. Introductory and review information from Chapter 0

II. Problems and Languages

A. Computable problems can be expressed as formal languages
   1. Rules of the problem cast as rules of the languages
   2. Strings of the language are solutions to the problem

B. Much of the study of formal languages pre-dates computers

III. Automata

A. State machines
B. Symbols or characters can be loosely defined
   1. single symbols such as $a$, $b$, 0, 1
   2. Compound symbols (count as one) such as “00”, “01”, “10”, “11”
   3. The empty character is denoted by $\lambda$ or $\varepsilon$
   4. Our book uses $\varepsilon$ for the empty character
   5. The empty character is not the same as the empty string
C. Can be formally defined by a five-tuple
   1. Alphabet of symbols ($\Sigma$)
   2. A set of allowed states ($Q$)
   3. Transition function or table ($\delta$)
   4. Initial state of the machine ($q_0$)
   5. A set of final, or accepting, states ($F$)
D. Machine cycle
   1. Machine reads a single input character (see B.)
   2. Current state and input symbol are input to the Transition table
      a. For each state and input character there must be a defined transition
      b. If there is no defined transition for a state/input pair, machine fails
   3. Machine runs until there is no more input
E. Determinate Finite Automata (DFA)
1. One, and only one, transition defined per state/input pair
2. The empty character is not allowed
3. If the DFA is in a final state when the end of the input is reached, the DFA accepts the input
4. If the DFA is not in a final state when the end of input is reached, the DFA rejects the input

F. Nondeterminate Finite Automata (NFA)
   1. NFAs are a superset of DFAs
   2. NFAs allow multiple transition choices per state/input pair
   3. NFAs allow the empty character to appear between any two characters (including $\varepsilon$)

G. For every NFA there is an equivalent DFA
   1. There is an algorithm to construct a DFA from any NFA
   2. The DFA constructed will accept the exact same strings as the original NFA

H. A DFA can be thought of as an NFA with no $\varepsilon$ transitions and only one defined transition per state/input pair

IV. Regular Expressions
   A. Every DFA or NFA has an equivalent regular expression (RE)
   B. There is an algorithm to convert a DFA or an NFA to an RE
   C. There is an algorithm to convert a regular expression to a DFA

V. Regular Languages
   A. All finite languages are regular
   B. All regular languages are recognized or described by:
      1. DFAs
      2. NFAs
      3. RE

VI. Pumping lemma for regular languages
   A. Proof by contradiction that a language is not regular
   B. Not much use in proving a language is regular
   C. Most often used to prove by contradiction that a language is not regular
   D. Pumping can be thought of as a game between two people

Exam II

VII. Context Free Grammars
   A. Variables denoted by capital letters
   B. Terminals denoted by lowercase letters that are members of $\Sigma$
   C. Rules state how variables may be replace to generate strings in the language
Topics

D. Regular languages and CFGs
   1. All left-handed productions generate regular language
   2. All right-handed productions generate regular language
   3. All regular languages can be generated by a CFG

E. It is possible to design a grammar to generate the strings of a given language

F. Chomsky Normal Form
   1. Recognize the allowed rules
   2. Convert a grammar to CNF
   3. Derivation of a string

G. Ambiguous strings/derivations

H. Relation to syntax rules for computer languages
Topics

Computability Theory

Exam III

VIII. Hilbert’s tenth problem

A. What is it?
B. Led directly to the Church-Turing thesis
C. Simple problems often lead to interesting mathematics!

IX. Computability and Decidability

A. Turing’s work
B. “Holes” in Mathematics

X. Turing Machines

A. Formal definition (mechanical parts)
   1. Infinite input tape
   2. Tape read/write head
      a. Reads one character each cycle
      b. Writes one character each cycle
      c. Moves one character right or left each cycle
   3. Internal states
   4. Internal transition table

B. 7-tuple
   1. $Q$
   2. $\Sigma$
   3. $\Gamma$ (must include the space character)
   4. $\delta$
   5. $q_0$
   6. $q_{accept}$
   7. $q_{reject}$

C. HALTS! When the machine halts
   1. Missing translations (in $\delta$) force the machine to halt instantly. These are treated as “rejects”
   2. $q_{accept}$
   3. $q_{reject}$
   4. There is no guarantee the machine will ever stop

D. Variations of Turing Machines
   1. Multiple input tapes
   2. Nondeterministic TMs
   3. Enumerator
   4. All are equivalent to a normal Turning Machine
Topics

E. Turing recognizable language and machine
F. Turing co-recognizable language and machine
G. Turing decidable language and machine
H. Turing unrecognizable languages
   1. Give a definition
   2. Give an example (HINT: We did one in class.)

XI. Decidability

A. Finite languages
B. Regular languages
C. DFA
D. NFA
E. RE
F. CFG
G. Turning Decidable
H. Turning Recognizable
I. Turning Co-Recognizable
J. Are there any languages that do not fit in this hierarchy?
K. Know the relationships between these languages; i.e., which are subsets of which

XII. The halting problem

A. Why do you care?
B. Prove there is no way to build a TM to solve the halting problem.
C. Proves there are problems that are not computable

XIII. Sets and countability

A. Definition of set cardinality
B. Definition of mapping functions for sets A and B (f() over A \rightarrow B)
   1. Injective or “one-to-one”
   2. Surjective or “onto”
   3. Bijective or “one-to-one and onto”
C. Definition of countable
   1. All finite sets are countable
   2. Some infinite sets are “countably infinite”
   3. Know at least one example of countably infinite set and one of uncountable infinite set.
D. Cantor’s diagonal countable
Topics

Complexity Theory

XIV. Classification of problems

A. \textbf{P}
   1. Easy problems
   2. Known polynomial time algorithms for a deterministic TM
   3. Definition of Class \textbf{P}
   4. Know some P problems

B. \textbf{NP}
   1. Solvable by a non-deterministic TM in polynomial time
   2. Solution can be verified by a deterministic TM in polynomial time
   3. Definition of Class \textbf{NP}
   4. Know some NP problems
      a. Class \textbf{P} is a subset (proper or improper) of \textbf{NP}
      b. Remember: You can convert an NFA to a DFA

C. \textbf{NP-hard}
   1. Hard problems
   2. No known polynomial time algorithms for a deterministic TM
   3. Often optimization problems
   4. Intractable
   5. May have a solution, but the time complexity is extreme.
   6. Solution can be verified in polynomial time.
   7. Usually and approximate solution can be found in polynomial time.

D. \textbf{NP-complete} or \textbf{NPC}
   1. Must be \textbf{NP-hard}.
   2. The hardest problems there are.
   3. Know the definition.
   4. Most often cast as decision problems.
   5. Many have only brute force solutions that are typically $O(2^n)$.
   6. If one can be solved in polynomial time, all can be solved in polynomial time.

XV. More on \textbf{P}, \textbf{NP}, and \textbf{NPC}

A. The question in computer science: Is \textbf{P} = \textbf{NP}?
   1. Be able to explain what this means
   2. Be able to give/recognize a problem this would create or solve.

B. Know some \textbf{NPC} problems

C. Karp’s list of 21 \textbf{NPC} problems helps

D. Proof that a problem, say \textbf{L} is \textbf{NPC}
   1. Show it is \textbf{NP-hard}
   2. Show that some known \textbf{NPC} problem can be reduced to a special case of \textbf{L}
a. Given SAT is **NPC**
b. Show SAT $\geq_p L$ (SAT can be reduced to $L$ in polynomial time.)
3. Don’t forget QED, “…as was to be shown.”, or $\square$

e. Some **NPC** problems:
1. 0/1 knapsack (fractional knapsack is **P**)
2. 0/1 Bin Packing (think of a number of knapsacks all the same capacity...)
3. SAT
   a. This one can be proved to be **NPC** without needing a reduction.
   b. The first problem to be shown to be **NPC**
4. 3-SAT (2-SAT is **P**)
5. Set cover
6. Vertex cover
7. Hamiltonian Tour and/or Traveling Salesman Problem (Euler Tour is **P**)
8. Hilbert’s 10th problem
   a. Given an equation of the form $ax^3 + bx^2y + cxy^2 + dy^3 + e = 0$, does this equation have a pair of integer roots?
   b. The answer is **NPC**
9. Set packing or partitioning
10. Longest acyclic path between two vertices. Shortest path is **P**
11. Graph coloring (beyond the scope of this course, but interesting)
12. Integer Linear Programming.
   a. Generalized version of Hilbert’s 10th Problem
   b. Linear Programming with non-integer solutions is in class **P**
13. Sudoku and Battleship (just for fun)
14. k-center problem (Where do you locate $k$ distribution centers to minimize the delivery time to $x$ stores.)

xvi. Anything else that has come up on a problem, class, or lab since we talked about TM recognizable languages.