CSCI2243-Final Review

This is an outline listing the important course topics and questions that you ought to be able to know how to answer. The questions are not necessarily the kinds you will see on the exam, since in many cases they are a little too involved to make good in-class exam problems.

1 Propositional Logic

1.1 Skills

Take a propositional formula made with any of the connectives $\vee$, $\wedge$, $\neg$, $\rightarrow$, $\leftrightarrow$, $\oplus$ and evaluate it on any assignment of truth values to the variables. (That is, you should be able to construct a truth table.)

Identify when a formula is a tautology, a contradiction, satisfiable. Identify when two formulas are equivalent.

Use identities to simplify formulas and to rewrite them in special form, either using a restricted set of connectives, or obeying some other restriction, for example, disjunctive normal form.

Realize a propositional formula as a boolean circuit, and conversely.

Model a situation described by a word problem as the problem of finding satisfying assignments for a propositional formula, and solving the problem in this form.

1.2 Sample Problems

1. You should be able to solve all the unstarred problems in Chapters 1 and 2.
2. Design a boolean circuit that takes four input bits a_0, a_1, b_0, b_1 and has a single output bit c. c should be equal to 1 if and only $(a_1a_0)_2 > (b_1b_0)_2$, that is, the circuit compares two 2-bit integers in binary. Design an equivalent circuit using

only two-input NAND gates. Write an equivalent propositional formula (*i.e.,* a formula with variables a_0, a_1, b_0, b_1 that says $(a_1a_0)_2 > (b_1b_0)_2$) in CNF.

3. Show that
$$((p \vee q) \wedge (r \vee \neg q)) \rightarrow (p \vee r)$$
is a tautology. You should be able to do this in three different ways: Using a truth table; 'talking it out'; employing identities to show that it is equivalent to **T.**

2 Sets and Functions

2.1 Skills

Understand the use of meaning and use of these notations: $a \in A$, $\emptyset$, $A \cup B$, $A \cap B$, $A - B$ (equivalently $A \backslash B$), $A \subseteq B$, $A \subsetneq B$, $A \not\subseteq B$, $\mathcal{P}(B)$, $A \times B$, $\{x : \ldots\}$, $\bigcup_{j \in J} A_j$, $\bigcap_{j \in J} A_j$, $|A|$.

Understand the meaning of notations for special sets: $\mathbf{N}$, $\mathbf{Z}^+$, *etc.*

Be able to determine the truth of simple set identities through Venn diagrams and translation to formulas of Propositional Calculus.

Understand the use and meaning of these terms and notations: $f : A \rightarrow B$, domain, range, codomain, one-to-one, onto, $f \circ g$.

Understand how to translate between complex structures defined using sets and functions, and the intuitive (sometimes graphically depicted) meanings of these structures. Examples that you've seen include undirected graphs, directed graphs, and the definitions of DFA and Turing machine.

Basics concerning cardinality of finite sets: What is $|A \times B|$? $|\mathcal{P}(A)|$? This includes some basic analysis of counting problems using binomial coefficients.

Basics concerning countable and uncountable sets. Identify in some simple examples whether a given set is countable.

2.2 Sample Problems

1. All the unstarred problems in Chapter 3.

2. This refers to the formal definition of directed graph given in the notes. The *out-degree of a vertex v* in a directed graph is, informally, the number of arrows leaving v. Give a formal definition of the out-degree of v using only the symbols v,

E, V (where E and V are the edge and vertex sets of the graph) and set-theoretic notations (*i.e.,* don't use any English.)

3. Give a formula for the number of functions $f : X \to Y$, given $|X|$ and $|Y|$ (assuming X and Y are finite sets).

4. The definition of directed graph defines the edge set E as a subset of $V \times V$. But we can also define the edge relation as a function $f : V \to \mathcal{P}(V)$, where $f(v)$ denotes the set of vertices to which there is an arrow from v. This suggests that the number of such functions is the same as the cardinality of the set of subsets of $V \times V$. Prove this, using your answer to the preceding problem. (As a sort of companion problem to this, describe explicitly, in both directions, the one-to-one correspondence between subsets of $V \times V$ and functions from V to $\mathcal{P}(V)$.

5. For each of the following infinite sets, tell whether it is countable or uncountable (and justify—see example 21 and problem 48 of Section 2.4, along with the posted notes).

   - The set of rational numbers
   - The set of real numbers
   - The set of irrational numbers
   - The set of square roots of positive rational numbers
   - The set of languages $L \subseteq \{a, b\}^*$
   - The set of regular languages $L \subseteq \{a, b\}^*$ (and, while we're at it, the set of Turing-decidable languages and the set of Turing-recognizable languages).

3 Predicate Logic and Relations

3.1 Skills

Understand the language of predicate logic: quantifiers, relation symbols, function symbols, free and bound variables, terms.

Be able to recognize some simple instances of equivalence. For instance

$$\forall x \phi(x) \wedge \forall x \psi(x)$$

is equivalent to

$$\forall x (\phi(x) \wedge \psi(x))$$

but the corresponding statement with the existential quantifier is not true (why not?) As another example, $\neg \forall x \phi(x)$ is equivalent to $\exists x \neg \phi(x)$. (By the way, the general problem of determining whether two formulas of predicate logic are equivalent is undecidable.)

Be able to translate between informal English statements and the equivalent formal statement of predicate logic. Understand that the truth of a statement is dependent on the domain in which we interpret the symbols.

Understand terminology for various properties of binary relations: reflexive, symmetric, transitive, antyisymmetric, equivalence relation, preorder, partial order, total order. Be able to identify in simple cases which of these properties holds, given a description of the relation.

3.2 Sample Problems

1. Any of the problems in Chapter 4.
2. Show that if ϕ and ψ are formulas of predicate logic with one free variable then
$$\exists x\phi(x) \land \exists x\psi(x)$$
is not in general equivalent to
$$\exists x(\phi(x) \land \psi(x)).$$
To do this, give a single counterexample: That is, give an example specific formulas ϕ and ψ over a specific domain for which one of the above sentences is true and the other false. Does one of these sentences always imply the other?
3. Model the hierarchical structure of an organization by having variables represent people in the organization, and letting $R(x, y)$ denote the relation, 'x reports to y', or equivalently 'y supervises x'. The languages also has constants like 'Fred', 'Tanya', 'Roxanne' representing specific people. Write sentences of predicate logic that express the following:
   - There is exactly one person in the organization who reports to nobody.
   - There are at least three people who report to Tanya
   - Everyone whom Roxanne supervises, supervises someone.
4. Write formulas of predicate logic over the base 0,1,+, $\times$ that express the following
   - $x|y$

- x is prime (you can use '|' in the formula, essentially appealing to the solution to the previous question, but it should be possible to expand this and get a formula that only uses the original base of symbols).
- there are infinitely many primes (you can use 'is prime' in the formula, again appealing to the solution of the preceding problem).
- there are infinitely many pairs of twin primes (see Example 9 of Section 3.5).

4 Number Theory

4.1 Skills

Understand the division 'algorithm', the use of the mod symbol in both its forms, *e.g.,*

$$7 \bmod 3 = 1$$

$$7 \equiv -5 \pmod 3,$$

and the use of the symbol '|' for divides. Understand that congruence is compatible with addition and multiplication.

Understand how positional number systems work, and how to translate between bases; especially between decimal and binary.

Understand what prime numbers and composite numbers are, and what the Fundamental Theorem of Arithmetic is.

Understand Euclid's Algorithm in both its simple form (for computing greatest common divisors) and its extended form (for finding solutions a, b to

$$am + bn = \gcd(m, n)$$

Understand how to rapidly compute (with a computer!) $a^b \bmod c$ even when a, b, c are integers that are too large for the computer to count up to. (For instance, integers several hundred digits long.) Understand how to carry out this computation rapidly by hand when c is small.

4.2 Sample Problems

1. Write down a string of bits, view it as the binary representation of an integer, and convert it to decimal.

2. Write down a string of decimal digits, view it as the decimal representation of an integer, and convert it to binary.

3. Write down a reasonably long string of decimal digits (at least 8 digits) and determine the remainder it leaves upon division by 7, and also upon division by 13. You should be able to accomplish this with a very small amount of computation and never have to do arithmetic with numbers larger than about 20 or 30.

4. Find an integer $0 \leq x < 50$ such that $6x \equiv 1 \pmod{35}$. How do you know, even before any computation is attempted, that there is a solution to this problem? How many different solutions are there?

5. Compute $6^{432} \bmod 8$ without a calculator. Computer $6^{43256789} \bmod 7$ in your head.

5 Proofs, including Proofs by Induction

5.1 Skills

You should understand and be able to reproduce several different kinds of arguments: direct proofs, proofs by contradiction, situations where a single example suffices to prove something, proofs by induction, and proofs by strong induction. You should also understand recursively defined sets, functions and sequences.

5.2 Sample Problems

1. Any problem from Chapter 5 and any unstarred problem from Chapter 6, with the exception of 10, 11 and 13.

2. Prove for all natural numbers n

$$2^n \equiv 1 \pmod{7} \quad \text{if} \quad \text{and} \quad \text{only} \quad \text{if} \quad 3|n.$$

   Remember that when the statement says ‘if and only if’, there are two things to prove.

3. Prove that the n^{th} root of a prime number p must be irrational, as long as $n > 1$. (HINT: You can do this by applying the Fundamental Theorem of Arithmetic: If you assume that the root is rational, you will arrive at two numbers that are equal but that have different numbers of prime factors in their prime factorizations. Make sure your proof is using the assumption that $n > 1$, otherwise you will have proved that p itself is irrational!)

4. Consider the sequence $\{c_n\}$, $n \geq 1$, defined by

$$c_1 = c_2 = c_3 = 1,$$

$$c_{n+3} = c_n + c_{n+1} + c_{n+2}$$

for $n \geq 1$. Write down c_4, c_5, c_6 and c_7. Prove that $c_n < 2^n$ for all n. Prove that $1.7^n < c_n$ for all $n > 20$.

5. Prove that

$$\sum_{i=1}^{n} i2^{i-1} = (n-1)2^n + 1$$

for all $n \geq 1$.

6. Prove that every integer n can be written in a unique fashion as $n = 4d + r$, where r is either -2, -1,0, or 1.

6 State, Machines, Finite Automata and Regular Languages

You do not need to be familiar with the NFA model, nor the algorithms for obtaining an automaton from a regular expression or vice-versa. But you should have a basic understanding of the meaning of DFAs and regular expressions.

6.1 Skills

Understand the difference between a string and a language, between the empty string and the empty set.

Construct DFAs to recognize some simple regular languages, given their descriptions.

Understand the regular operations union, concatenation, and star.

Understand regular expressions. Be able to construct regular expressions for simple regular languages, given their descriptions.

6.2 Sample Problems

1. Exercises 1-4, 6 and 8 of Chapter 8.
2. Design a DFA over the input alphabet $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ that recognizes the set of decimal representations of integers divisible by 3. (Note: Use a special property about the decimal representations of integers divisible by 3. This can be done with 3 states.
3. Design a DFA over the input alphabet $\{a, b\}$ that recognizes the language $\emptyset$.
4. Repeat the preceding problem for the language $\{\epsilon\}$. (These 'stupid' examples are a good way to tell if you understand the definitions.)

   Find a regular expression for the language in Problem 2 above. (This is a little harder, in spite of the small number of states involved, and is probably best done using the algorithm of Section 8.2.9.

7 Turing Machines and Computabiity

7.1 Skills

Simulate the computation of a Turing machine, given its description, on an input string.

Know what the Church-Turing thesis is.

Understand what recognizable, and decidable languages are—not just in terms of formal definitions about Turing machines, but in a way that connects with your own knowledge and intuition about computer programs.

Be familiar with the fact that the halting problem for Turing machines (and for Java programs, C++ programs,....) is undecidable. Use this fact to prove that other problems are undecidable.

7.2 Problems

1. Design a Turing machine with input alphabet $\{a\}$ that accepts a string if and only if its length is a power of 2. (HINT: The machine begins by marking one input symbol, and at each phase, doubles the number of symbols that are marked.)
2. Design a Turing machine with a single halt state (instead of accept and reject states) with input alphabet $\{a, b\}$. When started on input w, the machine erases its tape and writes $abab$ on the tape, halting with the reading head at the first position

in this string. (A machine that does this, replacing its input by a special hard-wired value, is an important ingredient in some reductions of the halting problem to other problems.)

3. Show that the following problem about Turing machines is undecidable: Given a Turing machine $\mathcal{M}$, tell whether $L(\mathcal{M}) = \emptyset$. ($L(\mathcal{M})$ denotes the set of strings accepted by $\mathcal{M}$. As a hint, show that if we had an algorithm for this, we would have one to determine if a given Turing machine accepted a given input–that is, we would have an algorithm for deciding

   $$\{enc(\mathcal{M}\#w : \mathcal{M} \text{ accepts } w\},$$

   whereas we proved that no such algorithm exists. Problem 2 above can help.)

4. What about the same problem for DFAs? Can one decide for a given DFA $\mathcal{A}$ whether $L(\mathcal{A}) = \emptyset$?