CSC 202 - Discrete Math for Computer Scientists II

Professor Shai Simonson

**Assignment 1 – Sets, Set Algebra, and Applications to Computer Science**

A. Applications of Sets

Assume a universal set of 8 elements. Given two sets *A* and *B,* each represented by 8 bits, explain how to use bitwise and/or/not operations, to:

1. Extract the rightmost bit of set *A*.
2. Make the odd numbered bits of *A* equal 0. (The convention is to number the bits from right to left as 0 to 7).
3. Make bits 4-6 of *A* equal to 1.
4. Determine if *A* ⊆ *B*.
5. Extract *A* − *B*.

B. More Applications of Sets

The Inclusion/Exclusion Theorem for two sets states that for any two sets, *A* and *B*,

|*A* ∪ *B* | = |*A*| + |*B*| – |*A* ∩ *B*|.

Note that the *converse* of this theorem is, in general, not true! That is, it is possible to have positive integers that satisfy this equation, but *not* have two sets with appropriate cardinalities for each expression. For example, let the values of |*A* ∪ *B*|, |*A*|, |*B*|, |*A* ∩ *B*| be 40, 60, 10, and 30 respectively. Although 40 = 60 + 10 – 30, there is no way to have 30 elements in |*A* ∩ *B*| and only 10 elements in *B*!

The Inclusion/Exclusion Theorem for three sets states that for any three sets, *A*, *B* and *C*,

|*A* ∪ *B* ∪ *C* | = |*A*| + |*B*| + |*C*| – |*A* ∩ *B|* – |*A* ∩ *C*| – |*B* ∩ *C*| + |*A* ∩ *B* ∩ *C*|.

Given a set of 29 students, where:

8 students need housing and financial aid,

12 students need housing, financial aid, and an e-mail account,

17 students need an e-mail account and financial aid,

23 students need housing,

20 students need financial aid,

19 students need e-mail accounts, and

4 students don’t need anything.

1. How many students need both housing and e-mail?
2. Change the numbers above to 57, 8, 3, 21, 21, 32, 31, 8. What is your answer now?
3. Which answer is bogus and why?
4. Prove the general Inclusion-Exclusion Theorem for *n* sets, by induction. To illustrate your general proof, you may assume the theorem works for *n* equals 1 through 4, and prove it for *n* = 5.

C. Induction Proofs

1. Generalize De Morgan’s laws for *n* sets and prove the laws by induction.
2. The power set of *A* is the set of all subsets of *A*. Prove by induction on the size of the set, that the power set *P(A)* has cardinality *2|A*|.

D. Countability Proofs and Functions

1. Prove that the set of integers has the same cardinality as the set of positive integers.
2. Prove that the set of all *quadruples* of positive integers is countable. A quadruple is an ordered 4-tuple (*a,b,c,d*) where the letters are positive integers. Hint: Use the fact that positive rational numbers are countable.
3. Prove that the set of all C++ programs is countable.
4. Section 2.3 – 10, 12, 13, 14, 20.
5. Section 2.5 – 1, 2, 16, 17.

E. Undecidable Problems

Consider any computer program that takes other programs as input, and outputs *yes* or *no* based on some criterion, (a compiler or interpreter, for example). It is possible for such a program to be fed back into itself, and depending on the program, it might either say *yes* (for example, a C++ compiler written in C++) or *no* (for example, a C++ compiler written in Java).

Define the programs that say *no* to themselves as *self-hating* programs. (This is not how we defined self-hating programs in class.) Now, assume that there is a computer program called *Hater* that takes other programs as input and says *yes* to all the *self-hating* programs, and *no* otherwise.

1. Assume that *Hater* never runs forever on an input. Analyze what happens when *Hater* is input to itself. Is such a *Hater* possible?
2. Now consider the possibility that *Hater* might run forever answering neither *yes* nor *no* on some input*,* and reanalyze the paradox of running *Hater* on itself. Is such a *Hater* possible?
3. Finally, redefine *self-hating* to mean that a program does not say yes to itself. (This is the definition we used in class). That is, a program is self-hating if and only if it says no to itself or it loops forever on itself. Once more, consider running *Hater* on itself and determine whether *Hater* is possible.