CSEP590 – Problem Set 1
Summer 2003
Due: Saturday, July 5 by midnight.
Directions: Answer each of the following questions. Email your solutions in doc, pdf, or html format to evan@cs.washington.edu

1. Translate the following English sentences into propositional formulas. As a first step, clearly define your set of atomic propositions (for example, atomic proposition p might be defined as "I am a student")

a. It is below freezing and snowing.
b. It is below freezing but not snowing.
c. It is either snowing or below freezing (or both).
d. It is either below freezing or it is snowing, but it is not snowing if it is below freezing.
e. It is not snowing if and only if it is below freezing or it is too warm.

2. A collection of logical connectives C is called *functionally complete* if every propositional formula is logically equivalent to a propositional formula involving only the collection C of connectives.

a. Prove that the set $\{\neg, \wedge\}$ of logical connectives form a functionally complete collection.
b. Recall the NAND connective defined in class, namely that if x and y are two propositional formulas, then x NAND y $\Leftrightarrow \neg(x \wedge y)$. We'll denote the NAND connective by the symbol '|'. Prove that the set {|} (ie, the set containing only the NAND connective) is a functionally complete collection.

3. In class we discussed a set of proof rules for performing natural deduction on propositions. Using the same style as shown in class (in particular, be sure to number and justify each line of the proof), prove the following sequent (where α denotes "entails"):

$$p \rightarrow q,\ r \rightarrow s \ \alpha \ \ p \vee r \rightarrow q \vee s$$

4. This problem will exercise the technique of proof by mathematical induction. Below is a brief synopsis of the technique for purposes of review. Many theorems state that P(n) is true for all positive integers n, where P(n) is a propositional function, such as the statement $P(n) = 1+2+\ldots+n = n(n+1)/2$ as we saw in class. Mathematical induction is a technique for proving theorems of this flavor (we saw a proof of the previous P(n) in class). In other words, mathematical induction is used to prove $\forall n\ P(n)$ where n ranges over the set of positive integers. The proof technique consists of 2 steps:

Basis step: prove that P(1) is true.

Inductive step: Assume that P(n) is true for some arbitrary $n > 1$. Prove that $P(n) \rightarrow P(n+1)$ is true.

Armed with this technique, consider the following. The *harmonic numbers* H_k, k=1,2,3,… are defined by

$H_k = 1 + 1/2 + 1/3 + \ldots + 1/k.$

For example, $H_4 = 1 + 1/2 + 1/3 + 1/4 = 25/12$.

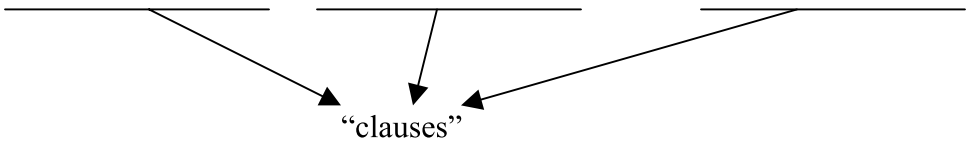
Use mathematical induction to prove the following inequality:

$H_{2^n} \geq 1 + n/2.$

(Hint: This problem should take a bit of thinking, but really boils down to careful manipulation of the math. Let P(n) be the proposition that $H_{2^n} \geq 1 + n/2$. So in the basis step, you prove that P(0) holds where P(0) is of course just $H_{2^0} = H_1$)

5. This problem should hopefully be more of a challenge, but I think it will help you to gain a deeper understanding of an important problem in computer science, and one with many applications to model checking that we'll see later in the course. Recall our discussion in class about propositional formulas in conjunctive normal form (CNF). Remember that such a formula F is defined as:

$$F = (A_{11} \vee A_{12} \vee \ldots \vee A_{1n(1)}) \wedge (A_{21} \vee A_{22} \vee \ldots \vee A_{2n(2)}) \wedge \ldots \wedge (A_{m1} \vee A_{m2} \vee \ldots \vee A_{mn(k)})$$



where A_{ij} are literals (either an atomic proposition or its negation), and a set of disjunction of literals (denoted by the underlined sections of F above) are called clauses. Note that F can contain clauses of arbitrary length, were the length of a clause refers to the number of literals appearing within the clause. Literals and their negations may appear more than once in F and more than once in any individual clause. The general problem of determining whether F is satisfiable is known as k-SAT, meaning that clauses in F have length $\leq$ k for some value of k. We discussed how efficiently determining whether such formulas are satisfiable is a major open question in computer science (and indeed mathematics). When I say "efficient" I refer to an algorithm that runs asymptotically better than exponential time in the worst case (indeed, an algorithm that runs in polynomial time, even in the worst case).

So what if we restricted the structure of F? Would this make the problem any easier to solve? In part A of this question, you will discover a partial answer.

a. Suppose that we restrict our CNF formulas so that k = 3. That is, we consider the problem 3-SAT where each clause in F has length *exactly* 3 (strictly speaking, clauses could have length 1 or 2, but we won't deal with that here. In fact, length 2 or length 1 clauses don't add any difficulty to the problem). The question is, does there exist an efficient algorithm for determining the satisfiability of 3-SAT formulas? You'll show that in fact 3-SAT is just as hard to solve as k-SAT, meaning that there only exists an efficient algorithm for 3-SAT if there exists one for k-SAT. To show this, your task is as follows: show that given an arbitrary formula F for k-SAT, you can efficiently convert

F into a formula F' for 3-SAT, *such that F' is satisfiable if and only if F is satisfiable*. This "reduction" will prove the following, namely that if I did have an efficient algorithm for 3-SAT, then I could take my instance F of k-SAT, efficiently convert it into a 3-SAT instance F', and use my efficient 3-SAT algorithm to solve F'. Since F' is satisfiable if and only if F is satisfiable, this means that I've also solved my original instance F of k-SAT! Thus, the two problems must be equally difficult!

b. Now let's consider CNF formulas so that k=2. Is this still a "hard" problem? Argue why instances of 2-SAT can be solved efficiently. What about its structure makes it an easier problem? Give an efficient algorithm for determining whether a formula F for 2-SAT is satisfiable.