**Computer Science 81, Fall 2007**
Assignment 5
Due Tue. Oct. 9

1. [50 points] Show, by working out an example that obviously generalizes, that there is no algorithm for determining whether a set of predicate logic formulas is unsatisfiable.

   Suggestion: Start with the result that there is no algorithm for determining whether an arbitrary Turing machine halts on a given input. Show that for any Turing machine M and initial tape x, there is a set of clauses such that the set of clauses is unsatisfiable iff M halts on x.

   This could be established by using one 1-ary function symbol for each distinct tape symbol. The clauses would define the possible moves of the Turing machine. (Note that you will need to represent blank tape and tape expansion.) This is similar to the pegs puzzle example given in class. Clauses for it can be found on the course web page as Pegs Puzzle.

   Demonstrate your technique on a specific M and x by giving the corresponding clauses to Otter-λ and have it check unsatisfiability. Do this for both halting and non-halting cases. The examples to use are given by the two TM tables below. The tape alphabet is {**a**, **x**, **b**} where **b** represents blank. It is assumed the tape has some number of contiguous **a**'s and that the head is positioned at the rightmost **a**, if there is any. Program **double** is supposed to double the number of contiguous **a**'s, which it does by adding an **x** at the right end for each **a**, after replacing that **a** with an **x**. Once each **a** has produced two **x**'s, it converts all **x**'s back to **a**'s. If the head starts over a **b**, that indicates that there are no **a**'s to double. State s is the start state.

   Program **damaged** is a damaged version of **double**, which doesn't converge. In it, the symbols shown in parentheses in the table replace the other symbols.

| double | | | | |
|---|---|---|---|---|
| Current state | Symbol read | Symbol written | Head moves | Next state |
| g (start) | b | b | left | t |
| g | a | c | right | h |
| t | b | b | right | j |
| t | a | c | right | h |
| t | c | c | left | t |
| h | c | c | right | h |
| h | b | c (b) | left | t |
| j | c | a | right | j |
| j | b | b | left | e (g) |

Show your work running on input **baaaa**. Submit a description in prose of the entire approach, not just the resulting clauses.

2. [50 points]

Otter is not the best tool for doing inductive proofs such as the ones you did in assignment 4 for natural numbers. (ACL2 is a tool that can do this, but it works in a totally different way.) For one reason, Otter does not know how to discover instances of the induction rule that is usually needed in such a proof. However, we can give the various pieces of such a proof to Otter and expect that it will produce a proof for each piece. For example, we could give the basis of an inductive proof and the induction step as separate theorems to be proved. In effect, Otter is doing the laborious parts of the proof, while leaving the creative part up to us.

By providing appropriate axioms and lemmas, see if you can get Otter to prove the commutative law of addition for the natural numbers, as you proved in assignment 4, problem 1. Use Otter's built-in version of equality. You may also use its version of quantified formulas, or you can convert to clausal form by hand.