CS 271 (Spring 2013) — Assignment 4
Due: 02/21/2013

This homework is to be done in groups of 3 students, assigned randomly by the program posted on the course web site. Each group can only turn in one solution, with all students' names listed on it. The group composition is posted at `http://www-bcf.usc.edu/~dkempe/CS271/team-assignments-4.html`.

(1) Read Chapter 1. Seriously, read the whole thing. Preferably twice. If you really understand this, the rest of the course will be a lot easier on you.

(2) Attend a sherpa session this week (meaning: 2/18–21). Preferably, that would be the one you're assigned to, but it doesn't have to be. This week, sherpas will be taking attendance. Your group will receive credit based on how many of you were at your sherpa sessions.

(3) Find at least three famous lines (proverbs, lines from movies, folk wisdoms, ...) which contain phrases that can be naturally translated into quantifiers. (Examples: "Every dog has his day", or "There's no business like show business.") Give those lines and their translation. Discuss any potential ambiguities in the quantifier order (similar to the "There's a soulmate for everyone" example from class).

(4) Solve the following exercises from the textbook

(a) Section 1.4, Exercises 8, 28, 34, 40, 61

(b) Section 1.5, Exercises 2, 6, 16, 30, 36, 44, 48

(c) Section 1.6, Exercises 6, 14, 20, 28, 35

(5) [0 points]
Chocolate Problem (2 chocolate bars): Turing Machines[1] are an extremely stripped down version of a computer. For many decades, they were the most widely used mathematical model of computing, and they are still incredibly useful. Here is how they work:

(a) The memory of a Turing Machine consists of an infinite one-dimensional array; let's call it a. Thus, $a[i]$ holds the content of the i^{th} memory cell, for each $i \in \mathbb{Z}$. For simplicity, we can assume that each memory cell contains just one of three value, so each $a[i] \in \{0, 1, \perp\}$.[2] Think of $\perp$ as indicating that the memory cell is empty.

(b) The processing is done by what's called the "head" of the Turing Machine. You can think of the head as a tiny robot. Here is what the head does. At any point, it sits on one of the memory cells i, and reads the content $a[i]$. It also holds some internal state s, which is an integer between 0 and N, for some number N. (Think of N as the number of lines of code.) Based on $a[i]$ and s, the head decides on the next move, which consists of three parts:

- Overwrite $a[i]$ with a new value, which is 0, 1, or $\perp$.
- Enter a new state $s' \in 0, \ldots, N$.
- Move one step to the right, one step to the left, or stay in the same place.

(c) Which actions are executed is part of a lookup table, which constitutes the "program" of the Turing Machine. You can think of the lookup table as three 2-dimensional arrays. One array $W[c, s] \in \{0, 1, \perp\}$ tells the Turing Machine what to write as a function of the character c it is reading and the state s it is in. The second array $S[c, s] \in \{0, \ldots, N\}$ tells it which new state to enter. And the third array $M[c, s] \in \{-1, 0, 1\}$ tells it which direction to move.

[1] named after Alan Turing whose 100th birthday we were celebrating last year

[2] The general model allows a larger set, so long as it is finite; that doesn't really change anything, though sometimes, it makes "programming" much easier.

(d) The input to the program is written on the tape in binary starting at position 0. All other memory cells are set to $\perp$ initially.

(e) The Turing Machine always starts on cell 0, in state 0.

(f) When the Turing Machine reaches the state N, that's a sign to terminate. Whatever is written on the tape at that point is the output.

It is a bit tedious, but not particularly difficult, to show that every single C program (or any other programming language) can be emulated by a Turing Machine.

Here, instead, you are supposed to encode a Turing Machine into a propositional logic formula. Specifically, let's encode the following simpler task. You want to find out whether your Turing Machine will terminate (i.e., enter state N) in t or fewer steps. Describe how to produce, from the given lookup table and input that's written on the tape, a logic formula with the property that it is true whenever on the given input, the Turing Machine terminates in t or fewer steps, and false whenever the Turing Machine does not terminate in t or fewer steps on the input.

Hint: You will mostly want to think carefully about what propositional variables to use. In particular, how can you encode the memory and the lookup table? Also, think carefully about how you are going to make sure that your formula has only finite length?

It may amuse you also to write a little Turing Machine simulator in C++; that's actually quite easy. Then, you can try to write some simple programs as Turing Machine lookup tables. You'll soon be grateful how much easier by comparison it is to write programs in C++ than in Turing Machines.