6.080 / 6.089 Great Ideas in Theoretical Computer Science
Spring 2008

6.080/6.089 Problem Set 1

MIT, Spring 2008

Assigned: Feb. 12, 2008
Due: Feb. 28, 2008

1. Express the following sentences in first-order logic. Where appropriate, you can define functions and constants representing the concepts in the sentences.

   Example: "All your base are belong to us" becomes

   $$\forall x \left(\text{Base}\left(x\right) \wedge \text{Owns}\left(\text{You}, x\right)\right) \Rightarrow \text{Owns}\left(\text{Us}, x\right)$$

   (a) We all live in a yellow submarine.

   (b) You can fool some of the people all of the time, and all of the people some of the time, but you can't fool all of the people all of the time.

   (c) Every story has at least two sides.

   (d) There are infinitely many twin primes (i.e., numbers p and $p + 2$ that are both prime). [*Note:* You can assume that natural numbers, addition, and multiplication have already been defined.]

2. Recall that a proof system consists of a set $\mathcal{S}$ of "allowable statements," as well as *inference rules* by which statements in $\mathcal{S}$ can be derived from other statements in $\mathcal{S}$. We say a statement is *satisfied* if the variables in it are fixed in such a way that the statement holds. The proof system is called *complete* if, given a set of statements $S_1, \ldots, S_k$ in $\mathcal{S}$, every statement in $\mathcal{S}$ that's a *consequence* of $S_1, \ldots, S_k$ (meaning: satisfied whenever $S_1, \ldots, S_k$ are satisfied) can be derived from $S_1, \ldots, S_k$ by applying the inference rules.

   Show that a proof system is complete if and only if, starting from any set of statements $S_1, \ldots, S_k$ that's inconsistent (meaning: never simultaneously true), a contradiction can be derived by applying the inference rules. [*Note:* You can assume that, if a statement is in $\mathcal{S}$, then its negation is also in $\mathcal{S}$.]

3. In the resolution proof system, every statement consists of a disjunction (meaning logical OR) of *literals* (meaning Boolean variables or their negations), like so:

   $$(x \vee \neg y \vee z)$$

   The only allowed operation is to *resolve* two statements: that is, find a statement in which a literal x occurs, and another statement in which the literal $\neg x$ occurs, and produce a new statement containing every literal in the original two statements *except* x and $\neg x$. As an example, resolving the statement $(x \vee \neg y \vee z)$ with the statement $(\neg x \vee w)$ produces the statement $(\neg y \vee z \vee w)$. If a statement contains both a literal and its negation, then it's unconditionally true. If a statement doesn't contain *any* literals (i.e., is the "empty statement"), then it's unconditionally false.

   (a) Explain why resolution is a sound proof system. That is, why is the resolution rule a valid logical deduction? Also, if you can derive the empty statement by repeatedly applying the resolution rule, why must your original set of statements have been inconsistent?

(b) Use the resolution rule to show that the following nine statements are inconsistent.

$$\begin{array}{lll} (a \vee b) & (\neg a \vee \neg c) & (\neg b \vee \neg d) \\ (c \vee d) & (\neg a \vee \neg e) & (\neg b \vee \neg f) \\ (e \vee f) & (\neg c \vee \neg e) & (\neg d \vee \neg f) \end{array}$$

(c) Let "restricted resolution" be the variant of resolution where you're only allowed to resolve two statements if one or both of the statements have only one literal. Is this proof system sound? Is it complete?

(d) Let "doofus resolution" be the variant of resolution where you're allowed to simply add or delete any literal from any statement. Is this proof system sound? Is it complete?

(e) *[Challenge Problem]* Show that resolution is a complete proof system.
Hint: Use induction on the number of variables. In other words, first show that resolution is a complete proof system in the special case where all statements involve only one variable, x_1. Then *assume* resolution is a complete proof system when the statements involve $n-1$ variables $x_1, \ldots, x_{n-1}$, and use that assumption to show it's still complete when the statements involve n variables $x_1, \ldots, x_n$.

4. Given a line segment of length 1, show that it's possible, using straightedge and compass alone, to construct a line segment of any given rational length. Your proof can consist of words, a diagram, or both. [*Note:* This contrasts with the fact, for which we sketched a proof in class, that it's *not* possible to construct a line segment of length $\sqrt[3]{2}$.]

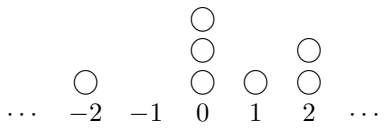
5. Recall that a *finite automaton* is a machine that can only read its input from left to right, that has a finite number of possible states (independent of the length of the input), and that can transition to a new state based on its current state as well as the symbol that it's currently reading.

   (a) Construct a finite automaton that accepts a Boolean string if and only if it contains at least three 1's.

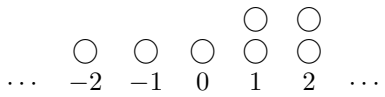
   (b) Prove that there's no finite automaton that accepts a Boolean string if and only if at least half of the bits are 1's.

6. Given a Boolean function f that maps n bits of input to one bit of output, call f *monotone* if changing an input bit from 0 to 1 can only ever change the output bit from 0 to 1 (and never from 1 to 0). Show that any monotone Boolean function can be constructed out of two-bit AND and OR gates.

7. *[Challenge Problem]* Suppose you have a finite number of cannonballs which are arranged along a line in stacks of various heights, like so:



You're allowed to perform the following operation, called "exploding": pick any stack with two or more cannonballs, and move one cannonball to the stack immediately to the left, and one to the stack immediately to the right. For example, starting with the configuration above and exploding the stack labeled 0 produces the following result:



By repeatedly exploding stacks, is it ever possible to return to the configuration that you started with? [*Note:* You can't say you "return to the starting configuration after zero steps"—you have to explode at least one stack!]