CSC 363 — Solutions to Test #2, Winter 2010

1) [30 marks] Recall that a clique in an undirected graph is a set of nodes in which every pair of nodes is connected by an edge. The textbook defined the language CLIQUE as follows:

$$\text{CLIQUE} \;=\; \{\, \langle G, k\rangle \mid G \text{ is an undirected graph that contains a clique with } k \text{ nodes} \,\}$$

The textbook proves that CLIQUE is NP-complete. Define the language TWO-CLIQUES as:

$$\text{TWO-CLIQUES} \;=\; \{\, \langle G, k\rangle \mid G \text{ is an undirected graph that contains two disjoint cliques of size } k \,\}$$

Prove that TWO-CLIQUES is NP-complete. Remember: You need to show *two* things to show that a language is NP-complete.

We first need to show that TWO-CLIQUES is in NP. A polynomial time verifier for TWO-CLIQUES takes $\langle w, c\rangle$ as input. It rejects if w does not have the form $\langle G, k\rangle$. Otherwise, we can have it accept if c is an encoding of two sets of nodes in G, the sets are both of size k, the two sets are disjoint, and both sets define cliques in G (ie, for both sets, there is an edge connecting every pair of nodes in the set), and otherwise it rejects. This can all be done easily in polynomial time.

Alternatively, one could show that TWO-CLIQUES is in NP by describing a nondeterministic Turing Machine that decides it and that runs in polynomial time.

Next, we need to show that every language in NP can be reduced in polynomial time to TWO-CLIQUES. We do this by showing that CLIQUE, which is known to be NP-complete, can be reduced to TWO-CLIQUES, so all languages in NP can be reduced to TWO-CLIQUES using a reduction via CLIQUE. We could reduce CLIQUE to TWO-CLIQUES in several ways; here is one.

The reduction maps $\langle G, k\rangle$ to $\langle G', k\rangle$, where G' is the graph consisting of all the nodes and edges of G, along with another k nodes that are connected to each other, but not to any of the nodes from G. This is easy to do in polynomial time.

We need to show that $\langle G, k\rangle$ is in CLIQUE iff $\langle G', k\rangle$ is in TWO-CLIQUES. For the forward direction, if $\langle G, k\rangle$ is in CLIQUE, then there is a subset of k nodes of G that is a clique, in which case the same subset of G' is a clique, and the k nodes in G' that are not in G also form a clique, so there are two disjoint cliques of size k, and hence $\langle G', k\rangle$ is in TWO-CLIQUES. In the other direction, if $\langle G', k\rangle$ is in TWO-CLIQUE, there are two disjoint cliques of size k in G'. These two cliques can't both be in the k nodes of G' that aren't in G, and can't be partly in G and partly out (since there are no edges between these parts), so there must be a clique of size k in G, and hence $\langle G, k\rangle$ is in CLIQUE.

2) [45 marks total] Part of the proof in the textbook that SAT is NP-complete shows that for any language, A, in NP, which is decided by a nondeterministic Turing Machine, N, that runs in polynomial time, there is a function that maps a string w to a string $\langle \phi\rangle$ that is an encoding of a Boolean formula, ϕ, that is satisfiable iff N accepts w.

The proof shows that there is an algorithm to do this reduction in polynomial time, for some fixed nondeterministic Turing Machine, N, which runs in some polynomial time bound — say $n^k + 2$, for some k, where n is the length of the input. The algorithm takes the string w as input and outputs $\langle \phi\rangle$. The formula ϕ that it creates has variables that describe the "tableau" for a computation of N on input w that halts within $n^k + 2$ steps (we'll let this tableau be $n^k + 3$ by $n^k + 5$ in size). The rows of the tableau are successive configurations of N, bounded by "#" symbols. The variable $x_{i,j,s}$ is 1 iff cell (i, j) of the tableau contains symbol s, where $s \in Q \cup \Gamma \cup \{\#\}$.

Recall that the formula ϕ has the form

$$\phi \;=\; \phi_{\text{cell}} \;\wedge\; \phi_{\text{start}} \;\wedge\; \phi_{\text{move}} \;\wedge\; \phi_{\text{accept}}$$

where ϕ_{cell} enforces that the variables describe a tableau with exactly one symbol in each cell, ϕ_{start} enforces that the first configuration is the correct start configuration for input w, ϕ_{move} enforces that each configure is followed by a possible successor configuration (same as the previous one if the machine has halted), and ϕ_{accept} enforces that the tableau contains an accepting configuration.

Suppose that the input alphabet of machine N is $\Sigma = \{0, 1\}$, the tape alphabet is $\Gamma = \{0, 1, \sqcup\}$, the state space is $Q = \{q_0, q_1, q_{\text{accept}}, q_{\text{reject}}\}$, the start state is q_0, and the transition function, $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$, is as follows:

$$\delta(q_0, 0) = \{(q_1, 1, L), (q_1, 0, R)\}, \quad \delta(q_0, 1) = \{(q_1, 1, L)\}, \quad \delta(q_0, \sqcup) = \{(q_{\text{reject}}, \sqcup, L)\}$$

$$\delta(q_1, 1) = \{(q_1, 1, R)\}, \quad \delta(q_1, 0) = \{(q_{\text{reject}}, 0, R)\}, \quad \delta(q_1, \sqcup) = \{(q_{\text{accept}}, \sqcup, L)\}$$

For all the questions below, suppose that the input is $w = 011$, so that $n = 3$, and that $k = 1$, so the tableau has 6 rows and 8 columns.

a) [12 marks] Fill in the two tableaus below to represent two different accepting computations on this input.

#	q_0	0	1	1	$\sqcup$	$\sqcup$	#
#	q_1	1	1	1	$\sqcup$	$\sqcup$	#
#	1	q_1	1	1	$\sqcup$	$\sqcup$	#
#	1	1	q_1	1	$\sqcup$	$\sqcup$	#
#	1	1	1	q_1	$\sqcup$	$\sqcup$	#
#	1	1	q_{accept}	1	$\sqcup$	$\sqcup$	#

#	q_0	0	1	1	$\sqcup$	$\sqcup$	#
#	0	q_1	1	1	$\sqcup$	$\sqcup$	#
#	0	1	q_1	1	$\sqcup$	$\sqcup$	#
#	0	1	1	q_1	$\sqcup$	$\sqcup$	#
#	0	1	q_{accept}	1	$\sqcup$	$\sqcup$	#
#	0	1	q_{accept}	1	$\sqcup$	$\sqcup$	#

b) [5 marks] How many variables are there in the formula ϕ? Explain.

The tableau has $6 \times 8 = 48$ cells, each of which can contain #, or one of 4 states, or one of 3 symbols, for a total of 8 possibilities. There are therefore $48 \times 8 = 384$ variables, representing the possibilities of each possible symbol being in each cell.
(Slightly different correct answers are also possible, provided they come with explanations indicating a slightly different approach to how ϕ is constructed.)

c) [9 marks] Write down the ϕ_{start} part of ϕ for this input.

This part of ϕ must ensure that the initial configuration has the input string on the tape, the tape head at the left, the state set to q_0, and the edges of the first row of the tableau set to #. This can be done with the following formula:

$$x_{1,1,\#} \wedge x_{1,2,q_0} \wedge x_{1,3,0} \wedge x_{1,4,1} \wedge x_{1,5,1} \wedge x_{1,6,\sqcup} \wedge x_{1,7,\sqcup} \wedge x_{1,8,\#}$$

d) [9 marks] The ϕ_{accept} part of ϕ is a disjunction (or) of literals. Write down three of these literals, and say (and explain) how many literals are in this disjunction.

Here are three:

$$x_{1,2,q_{\text{accept}}},\quad x_{3,3,q_{\text{accept}}},\quad x_{4,7,q_{\text{accept}}}$$

There are a total of $6 \times 8 = 48$ *such literals, one for each cell, though one could omit the ones on the edges that are always set to #, which would leave* $6 \times 6 = 36$ *literals.*

e) [10 marks, +1 for each correct, −1 for each wrong, minimum 0] The ϕ_{move} part of ϕ ensures that every 2×3 "window" of the tableau is legal for the machine N. For each of the following windows, circle "Yes" or "No" to indicate whether it is legal or not (no explanation is required):

#	0	1
#	0	1

Legal? **Yes** No

1	1	q_1
1	1	1

Legal? **Yes** No

q_0	1	1
q_0	1	1

Legal? Yes **No**

0	0	1
0	1	1

Legal? Yes **No**

q_1	0	1
1	0	1

Legal? Yes **No**

#	q_0	1
#	q_0	0

Legal? Yes **No**

q_0	0	1
1	q_0	1

Legal? Yes **No**

q_1	1	1
1	q_1	1

Legal? **Yes** No

1	q_1	$\sqcup$
q_{accept}	1	$\sqcup$

Legal? **Yes** No

#	q_0	$\sqcup$
#	q_0	$\sqcup$

Legal? Yes **No**

3) [25 marks] The class coNP is defined to contain all languages whose complements are in NP — in other words, $L \in \text{coNP}$ iff $\overline{L} \in \text{NP}$. A language L is defined to be coNP-complete if L is in coNP and any other language in coNP is polynomial time reducible to L — in other words, L is coNP-complete iff $L \in \text{coNP}$ and for all $L' \in \text{coNP}$, $L' \leq_P L$.

Prove that $\overline{\text{SAT}}$ is coNP-complete. You may use any parts of the proof that SAT is NP-complete that are useful for proving this.

Since SAT is in NP, $\overline{SAT}$ *is in coNP by definition.*

Let A be any language in coNP. Then $\overline{A}$ is in NP, so the proof that SAT is NP-complete shows that there is a polynomial time reduction, f, from $\overline{A}$ to SAT, such that for all w, $w \in \overline{A}$ iff $f(w) \in SAT$. This implies that $w \notin \overline{A}$ iff $f(w) \notin SAT$, and hence that $w \in A$ iff $f(w) \in \overline{SAT}$. This means that f is also a polynomial time reduction of A to $\overline{SAT}$, and since A was any language in coNP, $\overline{SAT}$ is coNP-complete.