CSE596, Fall 2012 **Problem Set 9 Answer Key** **Due Fri. 12/7**

(1) Show that the following decision problem is NP-complete.

INSTANCE: A collection $S_1, ..., S_r$ of subsets of a finite set U.
QUESTION: Can U be partitioned into two subsets V and W such that every set S_i contains at least one element of V and one element of W?

Hint: Take U to be the set of literals and their negations, plus a special "falseness element e." Use a reduction from $3SAT$, and take the interpretation that whichever of V or W gets e equals "false."

Answer: In cases where the answer is "yes," we need only guess V (a subset of U) and then the rest can be verified in polynomial time, so the language L of this problem belongs to NP. We show $3SAT \leq_m^p L$. Given a 3CNF formula ϕ with n variables $x_1, \ldots, x_n$ and m clauses, define:

- $U := \{ x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_n, \bar{x}_n, e \}$.

Here e is a special "falseness element," and its use is the main trick of the solution. We will interpret whichever partition gets e as the subset of literals made `false`, and the other subset `true`. To enforce a correspondence between "legal" partitions and truth assignments, we must ensure that a literal and its negation always belong to different partitions. This is done by the following subsets:

- $S_1 := \{ x_1, \bar{x}_1 \}, \ldots, S_n := \{ x_n, \bar{x}_n \}$.

Finally, we define the remaining sets so that choosing an element from them connotes satisfaction. Here and for the next problem we make the convention that u, v, w stand for literals, i.e. positively or negatively signed x_i for some i.

- For each clause $C_j = (u \vee v \vee w)$, $S_{n+j} := \{ u, v, w, e \}$. This gives $r = n + m$.

(If you wish, you could think of a "4CNF" clause $(u \vee v \vee w \vee e)$ being involved as a midway point, as on problem 2.) Clearly the corresponding insatnce of our problem is constructed in $\tilde{O}(n+m)$ time, where the "$\tilde{O}$" is modern notation for "ignore factors of $\log(n)$ or $\log(m)$," such as the labels i of variables and j of clauses. Anyway it's polynomial time.

If ϕ is satisfiable, then let V be the set of n literals made true in some satisfying assignment, and give e to W. Then V and W each contain one element from each S_i: within each clause set S_{n+j}, V has a satisfied literal while W has e. Conversely, suppose V and W partition U and each contains at least one member of each S_i. One of V and W must contain e—let's say without loss of generality that it's W. Then $|V| = n$, because V and W each take one element from S_1 through S_n, and if we call the literals in V `true` and those in W `false`, then we have a legal truth assignment to the variables. This assignment satisfies ϕ because V contains at least one element from each clause set $S_{n+1}, \ldots, S_{n+m}$.

Footnotes. If we dispense with the extra element e, then the remaining construction has the property that a legal partition exists precisely when there is a truth assignment that makes at least one literal in each clause true, and at least one in each clause false. The set of CNF formulas with such a satisfying assignment is the language of a problem called "Not All Equal (3)SAT." This is not the same problem as $3SAT$—its language is a proper subset—but the above does show that NAE-$3SAT \leq_m^p L$. In lecture and/or stray remarks I have touched on the fact that NAE-$3SAT$ is NP-complete, but it really should be justified. Recall from lecture that the proof of Cook's Theorem via circuits has the property that at most two literals in any clause can be satisfied in a satisfying assignment to the whole formula. The clauses were: singletons for the bits of "x" and the output wire, and triples $(u \vee w) \wedge (v \vee w) \wedge (\bar{u} \vee \bar{v} \vee \bar{w})$ for each (output wire w of each) NAND gate.

One would like to pad the smaller clauses with literals c and d that must be made `false`, but one can't directly use the formula from problem (1) because it doesn't have the "Not All Equal" property. Indeed, there is *no* way to force a literal to a given value via the NAE property, because the complementary truth assignment has the same property! However, the simple trick is to add a new literal e to all of the size-1 and size-2 clauses in the Cook-Levin proof. Then given an NAE assignment to the resulting new clauses, either the assignment or its complement makes $e =$ `false`, and so it induces a satisfying assignment of the original clauses. This actually shows that "NAE-3SAT" is NP-complete, whereas the reduction in my original answer implicitly reduced 3SAT to "NAE-4SAT."

(2) Tied s-t Path. You are given a directed acyclic graph $G = (V, E)$ in which each node has one "left" out-arc and one "right" out-arc, with a distinguished source node s and sink node t. You are also given a list of "ties" (u, v) which say that if you take the left [right] edge out of u, then you must also take the left [right] edge out of v. Is there a path from s to t subject to the ties? Show that this decision problem is NP-complete. (30 pts.)

[**18 pts. extra credit** for doing this with the extra condition that no node is tied more than once, i.e., the ties are disjoint pairs of nodes—you may wish to lean on special properties of the 3CNF formulas that translate circuits in my Cook-Levin theorem proof.]

Answer: First, the comparatively easy solution without the extra condition. "In NP" is immediate either way, since the path has smaller size than the graph and it is easy to check for each tie that the path takes the same left/right turn at the two nodes in the tie. Given a 3CNF formula ϕ with n variables and m clauses, we design the DAG G_ϕ to be a chain of m "clause gadgets," with some nodes labeled by variable names.

Each clause gadget C_j has an entry node s_j, an exit node t_j which can be identified with the entry node s_{j+1} for the next clause, and two other nodes. The entry node s_1 for the first gadget is "s," and "t" is taken to be t_m. Finally, G has one more sink node r for "reject."

Within each clause gadget, s_j is labeled by one of the three variables appearing in the clause, and the two other nodes besides t_j—call them u_j and v_j—by the other two variables. All nodes labeled by a variable x_i in the whole graph are tied to each other (or equivalently, successive pairs of them are tied). Let us designate the left arc out of any node labeled x_i as standing for the assignment $x_i = 0$, and the right arc for $x_i = 1$. If s_j is labeled x_i and x_i occurs positively in C_j, then the right arc goes to t_j—signifying that the clause has already been satisfied—while the left arc goes to u_j to poll the second member of the clause. If x_i occurs negatively (i.e., as $\bar{x}_i$), then the left arc goes to t_j while the right arc goes to u_j. Node u_j is coded similarly with the satisfying arc going to t_j and the unsatisfying arc going to v_j. At v_j, however, the unsatisfying arc goes to r. This finishes the description of G. The function $f(\phi) = G$ is computed in one pass through the clauses of ϕ, hence clearly in polynomial time.

Since every node of G other than t and r (recall t_j is the same as s_{j+1} for $j < m$) is labeled by a variable, and all occurrences of a variable are tied, maximal paths in G that respect the ties are in 1-1 correspondence with assignments $a \in \{ 0, 1 \}^n$. If a satisfies ϕ, then the corresponding path goes to t_j in every clause gadget and so ends up at t. Conversely, if a path goes from s to t then it must take a satisfying arc in each clause, which is possible only if the clause is satisfied by the corresponding assignment. Thus the reduction f is correct.

Now to comply with the extra(-credit) condition on ties, we replace each node labeled x_i in a clause gadget by a "sub-gadget." Each sub-gadget has two "exit nodes" (in parallel), and ones for the second and later occurrences of x_i or $\bar{x}_i$ in ϕ have two "entry nodes" (in sequence).

The first time x_i occurs, the 0-arc goes to an exit node labeled y_{i1}, and the 1-arc to one labeled z_{i1}. The 1-arc out of y_{i1} and the 0-arc out of z_{i1} then go to r, since they represent internal contradictions.

The 0-arc out of y_{i1} and the 1-arc out of z_{i1} go to the (first entry nodes of the subgadgets for the) same places the 0-arc and 1-arc out of x_i went in the original G described above. Note that these destinations still depend only on whether x_i occurs positively in the clause or as $\bar{x}_i$.

The next time x_i occurs, the first entry node is also labeled y_{i1} and is tied to the previous y_{i1}. The 1-arc out of this does not go to r. Instead it means $x_i = 1$, since the only way it can be legally taken in a path is for the earlier part of the path to have gone through z_{i1}. It goes to the new exit node z_{i2}. The 0-arc out of this y_{i1} does not go immediately to y_{i2}, but instead to the second entry node, which is labeled z_{i1} and tied to the previous z_{i1}. The 1-arc out of this node goes to r, while the 0-arc continues to y_{i2}. The meaning is this: If the earlier part of the path chose $x_i = 0$ then it went through the exit node y_{i1}, and so must have taken the 0-arc out of the entry node y_{i1}. It may then legally take the 1-arc out of the entry node z_{i1} here, since that choice is not tied. If the earlier part chose $x_i = 1$, then it is not tied at the entry node $yi1$ here, but choosing the 0-arc out of $yi1$ leads to the z_{i1} entry node where the tie to the earlier z_{i1} forces it to oblivion at r. Hence to survive, the path must exploit the fact that it's not tied to y_{i1} by taking the 1-arc there.

Finally, the new exit nodes y_{i2} and z_{i2} are coded similarly to the first exit nodes y_{i1} and z_{i1}. If there is a next occurrence of x_i, they are tied to the entry nodes for it. This completes the description of the modified graph G', and the reduction f' that computes it is clearly still linear-time computable. And G' abides by the extra condition on ties. Note that we kept the nodes of G labeled by the first occurrence of each variable but replaced the others by the entry nodes of sub-gadgets. The correctness of f' follows by the correctness of f and the argument of the previous paragraph. (This problem was posed to me by German researcher Thomas Thierauf in 1998. I solved it within a week, and then found that another German named Detlef Sesse had solved it in a paper earlier that year.)

(3) Text, "Homework 7.21": Prove that if some language that is PSPACE-complete under $\leq^p_m$ belongs to NP, then PSPACE = NP. *Also answer:* what happens if the language is complete under polynomial-time Turing reductions ($\leq^p_T$) instead? *Then* extend your answers to say what happens for the higher levels Σ^p_k of the polynomial hierarchy for $k \geq 2$, noting that $\Sigma^p_1 = \mathsf{NP}$. (9 + 6 + 6 = 21 pts.)

Answer: Suppose B is PSPACE-complete under $\leq^p_m$ and belongs to NP. Let any language A in PSPACE be given. By completeness, $A \leq^p_m B$. By the downward closure of NP under $\leq^p_m$, $A \in \mathsf{NP}$. Since A is an arbitrary member of PSPACE, $\mathsf{PSPACE} \subseteq \mathsf{NP}$, and since we already know $\mathsf{NP} \subseteq \mathsf{PSPACE}$, the classes would be equal. (Indeed, since PSPACE is closed under complements, this would make co-NP equal to NP too, so we would get $\mathsf{PSPACE} = \mathsf{NP} \cap \mathsf{co\text{-}NP}$.)

Under $\leq^p_T$, all we know is that $A \leq^p_T B$ with $B \in \mathsf{NP}$ places A into P^{NP}. Since the latter class (indeed the whole polynomial hierarchy) is contained in PSPACE, we get $\mathsf{PSPACE} = \mathsf{P}^{\mathsf{NP}}$.

For $k > 1$ the analogous things happen: A many-one complete set for PSPACE belonging to Σ^p_k would "collapse" PSPACE down to Σ^p_k—and all higher levels of the polynomial hierarchy would "collapse" down to the same class. Indeed, we get collapse down to $\Sigma^p_k \cap \Pi^p_k$. But for poly-time Turing reductions we only get collapse down to $\mathsf{P}^{\Sigma^p_k}$. (Which incidentally is contained in $\Sigma^p_{k+1} \cap \Pi^p_{k+1}$, but unlike the arithmetical hierarchy is not necessarily equal to it, just as $\mathsf{NP} \cap \mathsf{co\text{-}NP}$ might not equal P.)

(4) Define $2SAT$ to be the language of satisfiable 2CNF formulas, i.e. satisfiable Boolean formulas in conjunctive normal form with at most *two* literals per clause.

(a) Show that $2SAT$ belongs to P (15 pts.).

(b) Show that $2SAT$ is NL-hard under logspace many-one reductions ($\leq^{log}_m$). (Hint: give a log-space reduction from the *complement* of GAP to $2SAT$, and then argue on the basis of NL being closed under complementation.)

Answer. (a) The key trick is to regard a two-literal clause $(u \vee v)$ as the implication $\bar{u} \longrightarrow v$ together with its contrapositive, $\bar{v} \longrightarrow u$. (Recall that u can be a negative literal $\bar{x}_i$, and then $\bar{u} = x_i$.) Now these implications form themselves naturally into a directed graph $G = (V, E)$ with $V = \{ x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n \}$ (same as U in the last problem!) and the implications as edges. We claim that a 2CNF formula ϕ is satisfiable iff the corresponding graph G_ϕ has no *cycle* that includes some node u together with its negation $\bar{u}$.

If G_ϕ has such a cycle, then $u \Longrightarrow \bar{u}$ follows by a chain of implications that follows the path from u to $\bar{u}$ along the cycle, and $\bar{u} \Longrightarrow u$ follows by the rest of the cycle. This is very much like the contradiction "$d \in D \longrightarrow d \notin D \longrightarrow d \in D \ldots$" at the heart of the proof of "$D \notin \mathsf{RE}$" by diagonalization, and it means that ϕ itself implies a contradiction and must always be false. To show this more concretely, let t be any truth assignment to the variables, and suppose that t makes $u = \texttt{true}$ and $\bar{u} = \texttt{false}$. Somewhere along the path from u to $\bar{u}$ there must be an edge (v, w) that goes from a node v made `true` to a node w made `false`. Then the corresponding implication $v \longrightarrow w$ is false, and so the assignment t fails to satisfy the corresponding clause $(\bar{v} \vee w)$. The case where t makes $u = \texttt{false}$ and $\bar{u} = \texttt{true}$ is handled symmetrically, using the path from $\bar{u}$ back to u.

Conversely, if there is no such cycle, then let us select any node u in G_ϕ. If there is no path from u to $\bar{u}$, then set $u = \texttt{true}$; else set $u = \texttt{false}$. When there is no path from u to $\bar{u}$, the node sets

$$\begin{aligned} A_u &= \{ v : \text{there is a path from } u \text{ to } v \} \qquad \text{and} \\ B_{\bar{u}} &= \{ v : \text{there is a path from } v \text{ to } \bar{u} \} \end{aligned}$$

are disjoint. Set all the literals in A_u `true`, and all those in B_u `false`. If there is any node u' left over, repeat the process, setting $u' =$ `true` or `false` depending on whether there is a path from u' to $\bar{u}'$, and doing likewise to $A_{u'}$ and $B_{\bar{u}'}$ as before. Continue in this way until all nodes have been assigned truth values. Doing so *never* gives an edge going from a `true` node to a `false` one, because all edges into `false` nodes from previous iterations are exhausted in the definition of $B_{\bar{u}}$, while such an edge in a current iteration with u' and its negation $\bar{u}'$ implies a path from u' to $\bar{u}'$. Thus every clause corresponding to an edge in G_ϕ is satisfied by the resulting truth assignment.

Hence we have proved that the following algorithm really does determine whether ϕ is satisfiable: build G_ϕ and test for each literal u whether there is a path to $\bar{u}$ and also a path from $\bar{u}$ to u that would complete a cycle. This means running breadth-first search at most $2n$ times, and gives an $O(n^2)$ (i.e., polynomial) time algorithm.

Technotes. Actually, we have shown that $2SAT$ is in $\mathsf{co\text{-}NL}$, since when ϕ is unsatisfiable, a nondeterministic logspace machine can guess a literal x_i and guess and follow a sequence of clauses that correspond to the edges in a path from x_i to $\bar{x}_i$ and a return path from $\bar{x}_i$ to x_i. Since NL is closed under complements (Immerman-Szelépcsenyi Theorem), we get $2SAT \in \mathsf{NL}$, and from (b) it will follow that $2SAT$ is NL-complete. An alternative algorithm strategy is to combine every pair of clauses of the form $(u \vee v)$ and $(\bar{u} \vee w)$ and obtain the new clause $(v \vee w)$. Adding this so-called *resolvent* clause to ϕ doesn't change whether ϕ is satisfiable, since the new clause is implied by the other two. Keep doing this until you already have all possible resolvent clauses—this must happen within $\binom{2n}{2} \simeq 2n^2$ iterations since that's how many possible binary clauses there are. Then you can show that ϕ is satisfiable iff you never get a pair of clauses $(u \vee u)$ and $(\bar{u} \vee \bar{u})$ from this process. Actually, what this process of *resolution* has done is define the *transitive closure* of the graph G_ϕ above, and clearly G_ϕ has a path from u to $\bar{u}$ iff its transitive closure has $(u, \bar{u})$ as an edge. Hence this argument can proceed along similar lines to the featured one. The reason why this process of resolution doesn't work efficiently for $3SAT$ is that the resolvent of two 3CNF clauses is a 4CNF clause, and the blowup gets worse from there.