CS 21 Decidability and Tractability **Winter 2008**

Final Exam Solutions

Posted: March 26

1. (a) First, the problem is in PSPACE, for the usual reason for 2-player games. Given M, B, we are asking whether $\exists a_1 \forall a_2 \exists a_3 \forall a_4 \cdots Q a_n$ (where each $a_i \in \{1, 2, \cdots, n\}$ and Q is $\exists$ if n is odd and $\forall$ if n is even) such that $M[1, a_1] + M[2, a_2] + M[3, a_3] + \cdots + M[n, a_n] = B$. In particular, we can devise a recursive algorithm (for the slightly more general problem in which the leading quantifier may be either $\exists$ or $\forall$, and M may have fewer than n rows). The algorithm operates as follows: if the leading quantifier is $\exists$, we recursively check (using n recursive calls) whether for there exists $a \in \{1, 2, \ldots, n\}$, for which

$$\forall a_2 \exists a_3 \forall a_4 \cdots Q a_n \sum_{i=2}^{n} M[i, a_i] = B - M[1, a];$$

if the leading quantifier is $\forall$, we recursively check (again using n recursive calls) whether for all $a = 1, 2, \ldots, n$,

$$\exists a_2 \exists a_3 \forall a_4 \cdots Q a_n \sum_{i=2}^{n} M[i, a_i] = B - M[1, a].$$

The base case just involves comparing two integers. This leads to a recursive algorithm with recursion depth n and $\mathrm{poly}(|\langle M, B\rangle|)$ bits of state at each level, for a total space usage of $\mathrm{poly}(|\langle M, B\rangle|)$.

(b) We reduce from QSAT and we refer to the reduction for SUBSET SUM from Lecture 21. An instance of QSAT is a 3-CNF formula $\phi(x_1, x_2, \ldots, x_n)$, with m clauses, and we are asking whether

$$\exists x_1 \forall x_2 \exists x_3 \forall x_4 \cdots Q x_n \phi(x_1, x_2, \ldots, x_n) = 1.$$

We assume without loss of generality that n is even (we can add a dummy variable if necessary). We now describe the reduction.

In the first row of our matrix M we place two values (repeated as necessary to fill out the whole row), x_1^{true} and x_1^{false} from the SUBSET SUM reduction applied to ϕ. In the next row we place the two values, x_2^{true} and x_2^{false}. We continue in this fashion until we have populated the first n rows of matrix M, with the i-th row containing the values x_i^{true} and x_i^{false}.

The next $2m$ rows are: a row with the three values 0, $FILL1_1$ and twice $FILL1_1$ (again referring to the SUBSET SUM reduction), and a row consisting of all zeros, a row with the three values 0, $FILL1_2$ and twice $FILL1_2$, and a row with all zeros, and so on up to a row with the three values 0, $FILL1_m$ and twice $FILL1_m$ followed by a row with all zeros. The "target" value B we produce in the reduction is the same one produced in the SUBSET SUM reduction applied to ϕ.

Clearly this reduction runs in polynomial time. We argue that YES maps to YES. In the two-player game interpretation of QSAT (in which the players alternately assign truth values to the variables $x_1, x_2, \ldots, x_n$ with player 1 trying to end with a satisfying assignment), a YES instance ϕ implies that there is a win for player 1. In our matrix M, the first n rows exactly correspond to the players alternately choosing truth values for $x_1, x_2, \ldots, x_n$, and so there is a strategy for player one that ends this part of the game with a satisfying truth assignment selected. Now, in the remaining $2m$ rows, notice that player 1 is able to pick the required "filler" values (a 0, 1, or 2 in the required digit) for each of the m clauses, one at a time, to make the sum exactly B (player 2 always gets rows of all zeros, so he can't influence the sum in this phase). Thus there is a win for player 1.

Now we argue that NO maps to NO. As before in the matrix M, after the first n rows, the players have alternately selected truth values for $x_1, x_2, \ldots, x_n$, and since ϕ is a NO instance, player 2 will be able to ensure that the resulting assignment is not a satisfying one. In particular there will be some clause whose corresponding digit has 0 in the sum so far (since all of its literals are false under the selected assignment), which means that no matter what values player 1 selects in the second phase the sum of B will not be reached. Thus there is not a win for player 1.

2. (a) This problem is in P. Notice that a graph with maximum degree 100 cannot have any clique on more than 101 nodes. Therefore, we can find the maximum clique size in time n^{101} by enumerating all candidate cliques of up to 101 nodes, and checking each one. We then accept iff k is at most this maximum clique size.

   (b) HITTING SET-2 is NP-complete. It is clearly in NP because given a purported hitting set $H \subseteq U$ it is easy to check in polynomial time that $|H| \leq k$ and that each S_i has non-empty intersection with H.

   To show it is NP-hard, we reduce from VERTEX COVER. Given an instance of vertex cover $\langle G = (V, E), k \rangle$, we produce the following instance of HITTING SET-2: U is the set of vertices V, and we have a set $S_{(u,v)} = \{u, v\}$ in our collection $\mathcal{C}$ for each edge $(u, v) \in E$. Note that as required, all of the sets in $\mathcal{C}$ have size at most two. We set the bound k in the instance of HITTING SET-2 to be the same k as in the instance of vertex cover.

   Now, suppose there is a vertex cover $V' \subseteq V$ of size at most k. Then we claim that $H = V'$ is a hitting set, since for each set $S_{(u,v)} \in \mathcal{C}$, one or both of u, v must be in V' and hence in H.

   In the other direction, suppose there is a hitting set $H \subseteq U$ of size at most k. By definition, for each edge (u, v), H must include one or both of u, v since it hits set $S_{(u,v)} \in \mathcal{C}$. Thus $V' = H$ is a vertex cover of size at most k.

   We conclude that HITTING SET-2 is NP-complete.

   (c) This problem is NP-complete. It is in NP for the usual reasons. To show it is NP-hard, we reduce from (3,3)-SAT (from Problem Set 5). We use exactly the reduction from 3-SAT in Lecture 19, but since our starting CNF formula has no variable occurring more than 3 times, the degree of the resulting graph will be at most 4, which is certainly less than 100.

3. Let p be the pumping length, and consider the string

$$w = a^p b^p c^{p+p!},$$

with the first $2p$ symbols (the a's and the b's) marked. Ogden's Lemma states that w can be written as $w = uvxyz$, with vy containing at least 1 marked position, and vxy containing at most p marked positions. Since vy must contain at least 1 marked position, it cannot be completely within the c's. If v and y are both within the a's or both within the b's, then clearly pumping produces a string with unequal number of a's and b's, which is not in the language. Similarly, if v or y straddle the boundary between the a's and the b's, then pumping produces a string with a's and b's out of order, which is not in the language.

Otherwise, we must have v contained within the a's and y contained within the b's, and $|v| = |y| = k$ (if they are unequal lengths, then pumping produces a string with unequal numbers of a's and b's). So the string $w' = uv^{i+1}xy^{i+1}z$ is:

$$a^{p+ik}b^{p+ik}c^{p+p!}.$$

Note that the second condition of Ogden's Lemma ensures that $k < p$. So, choosing $i = p!/k$ (which is an integer!), we obtain a string which is not in the language. Thus L cannot have been context free.

4. (a) Decidable. We will reduce this problem to E_{CFG} (emptiness of context-free-grammars), which we saw in lecture was decidable. Given E we first build the DFA that recognizes language A, the complement of $L(E)$. This is possible because regular languages are closed under complement. We also know how to construct a NPDA that recognizes the language $B = L(G) \cap A$ (from PS2 and the solution Sipser problem 2.18(a)). We now check if B is empty. From lecture 11 (and Sipser Thm 4.8) we know that emptiness of CFGs is decidable. Moreover the complementation step and the intersection step are all computable transformations. Finally, note that B is empty iff $L(G) \subseteq L(E)$, so the language CFG-IN-REG is decidable.

   (b) Undecidable. We reduce ALL_{CFG} to REG-IN-CFG. Set $E = \Sigma^*$. Given an instance G of ALL_{CFG}, we produce the pair (E, G). If $G = \Sigma^*$ then clearly $L(E) \subseteq L(G)$; if $G \neq \Sigma^*$ then $L(E) \not\subseteq L(G)$. Therefore we have reduced ALL_{CFG} to REG-IN-CFG, and we know from Lecture 12 (and Sipser Thm 5.13) that ALL_{CFG} is undecidable.

5. (a) This is a special case of part (b) (although if you couldn't get (b), you could try to solve this easier case).

   (b) Let $A = (Q, \Sigma = \{0,1\}, \delta, s, F)$ be a DFA deciding language L. Let k be the length of $y = y_1 y_2 \dots y_k$. We construct a NFA B deciding L_{-y}. Machine B will consist of $k+1$ copies of A. The first and last copies will have all the transitions of A; the others will have no transitions within the states in that copy. B's start state will be the start state of the first copy of A, and its accept states will be the accept states of the k-th copy of A. For every transition in A from state p to state q labelled with y_1, we add an ϵ-transition from state p in the first copy of A to the copy of state q in the second copy of A. In general, for every transition in the A from state p to state q labelled with y_i, we add an ϵ-transition from state p (in copy i) to state q (in copy $i+1$).

For a string $xy_1y_2 \dots y_kz \in L$, we can follow the arcs the machine A would have followed while reading x in the first copy of A, then follow the newly available ϵ-transition to the second copy of A (placing us in a copy of the state we would have been in after reading y_1), then the newly available ϵ-transition to the third copy of A (placing us in a copy of the state we would have been in after further reading y_2), and so on. Finally we arrive at the $(k+1)$-st copy of A, in the state we would have been in after reading $y_1y_2 \dots y_k$. Now we proceed in this copy, reading z, which leads to an accept state, since $xy_1y_2 \dots y_kz \in L$.

If a string w is accepted by machine B, then there must be a computation path from the start state in the first copy of A to an accept state in the final copy of A. Let x be the portion of the string read before departing the first copy of A, and let z be the portion of the string read after entering the last copy of A. Then by construction $xy_1y_2 \dots y_kz$ must have been accepted by A, which completes the proof that B satisfies the requirements.