HW3 Solution Set
CPSC 468/568: Computational Complexity
(Spring 2015)

1 Problem 4.10

> Show that in every finite two-person game with perfect information (by *finite* we mean that there is an a priori upper bound n on the number of moves after which the game is over and one of the two players is declared the victor – there are no draws) one of the two players has a winning strategy.

Represent the game as an instance of QBF with n variables. As described in Example 4.15 in the textbook, player 1 has a winning strategy iff $Q = \exists x_1 \forall x_2 \exists x_3 ... (\exists/\forall) x_n \phi(x_1, ..., x_n)$ is true, for some ϕ which describes what it means for player 1 to win.

We will show that if player 1 doesn't have a winning strategy, then player 2 must have one. Suppose player 1 doesn't have a winning strategy. Then Q is false, and so $\neg Q$ is true. $\neg Q = \forall x_1 \exists x_2 ... (\forall/\exists) x_n \neg \phi(x_1, ..., x_n)$. Since the existential and universal quantifiers have swapped positions, this formula corresponds to a strategy for player 2. Also, notice that $\phi(x_1, ..., x_n)$ in Q encoded whether player 1 wins; thus $\neg\phi(x_1, ..., x_n)$ encodes player 1 not winning, which is the same as player 2 winning since there are no ties and the game is finite. Thus $\neg Q$ precisely encodes whether player 2 has a winning strategy, and since $\neg Q$ is true, player 2 does indeed have one.

2 Problem 5.7

> Show that **APSPACE = EXP**

($\Rightarrow$) Suppose $L \in APSPACE$, and let M be a poly-space ATM deciding L. We build a TM M' that decides L by computing the configuration graph of M on a given input x, and then marking nodes with "ACCEPT" as specified in the definition of an ATM to decide whether M accepts x. Since M uses polynomial space, the size of its configuration graph is at most singly exponential, and so M' runs in singly exponential time. Thus $L \in EXP$.

($\Leftarrow$) Suppose $L \in EXP$, and let M be a two-tape exp-time TM deciding L. We will make use of the locality of M's computation to shrink our space usage

to polynomial. Given an input x, let $H(i, j, a, q)$ be a predicate that's true iff after M runs on x for i steps, its tape head (more specifically, work/output tape head) is at position j, the character a is on the tape at position j, and M is in state q. Let $N(i, j, a, q)$ be true iff after M runs on x for i steps, character a is on the tape at position j, M is in state q, and the tape head is *not* at position j. Note that since M runs in time $O(2^{poly(n)})$, the step i and cell position j can be written in $O(poly(n))$ space. Thus these predicates only take up polynomial space. Now we come up with recursive formulas to compute H and N. Notice that if $H(i, j, a, q)$ holds then at step $i-1$, either the tape head was still at cell j and didn't move, or the tape head was at cell $j+1$ and moved left, or it was at cell $i-1$ and moved right. Similarly, if $N(i, j, a, q)$ holds then at step $i-1$, either the tape head was at cell j and moved away, or it was at cell $j+1$ and didn't move left, or it was at cell $j-1$ and didn't move right, or it wasn't at any of cells $j-1$, j, $j+1$. This reasoning leads to the following formulas for H and N when $i > 1$:

$$
\begin{aligned}
H(i,j,a,q) := \exists b, q'.&\\
&(\delta(q',b) = (q,a,S) \wedge H(i-1,j,b,q'))\\
\vee&(\delta(q',b) = (q,\_,L) \wedge H(i-1,j+1,b,q') \wedge N(i-1,j,a,q'))\\
\vee&(\delta(q',b) = (q,\_,R) \wedge H(i-1,j-1,b,q') \wedge N(i-1,j,a,q'))\\
N(i,j,a,q) := \exists b, c, q'.&\\
&(\delta(q',b) = (q,a,R|L) \wedge H(i-1,j,b,q'))\\
\vee&(\delta(q',b) = (q,\_,R|S) \wedge H(i-1,j+1,b,q') \wedge N(i-1,j,a,q'))\\
\vee&(\delta(q',b) = (q,\_,L|S) \wedge H(i-1,j-1,b,q') \wedge N(i-1,j,a,q'))\\
\vee&(N(i-1,j-1,b,q') \wedge N(i-1,j,a,q') \wedge N(i-1,j+1,c,q'))
\end{aligned}
$$

We leave it as a simple but tedious exercise for the reader to formally prove the validity of these formulas. Also note that we technically should be keeping track of the input tape head as well and have input bits as part of our transitions, but this would make the formula somewhat unreadable and is completely trivial to do, so we're omitting it. Now we can use the above formulas to obtain a poly-space ATM M' deciding $L(M)$ as follows: on input x accept iff $H(2^{p(n)}, 1, 1, q_{halt})$ holds. To check whether it holds, recurse on the formulas above, using $\exists$ states to choose b, c, q', and the disjunct to recurse on, and using $\forall$ states to simultaneously recurse on all the conjuncts in a particular disjunct. Once we reach $i = 1$, we just look at the definition of M to decide what to return. We can reuse all space with each recursive call since we don't need any kind of stack, so this algorithm runs in poly-space. Thus $L \in APSPACE$.

Alternate Solution: It has been proved that finding a winning strategy for the game of checkers on an $n \times n$ board is EXP-complete (see "N by N

Checkers is Exptime Complete" by J. M. Robson at `http://epubs.siam.org/doi/abs/10.1137/0213018`). Thus if we can show that CHECKERS $\in APSPACE$, then we immediately get $EXP \subseteq APSPACE$. It's actually quite easy to see that CHECKERS $\in APSPACE$; player 1 gets $\exists$ states and player 2 gets $\forall$ states. It's pretty clear that we can encode the current board state and legal moves in polynomial space, so we can indeed decide this problem with a poly-space ATM. Note that we can't say CHECKERS $\in PSPACE$ by the same logic because the number of moves (i.e. quantifier alternations) can be exponential.

3 Problem 5.9

> The class DP is defined as the set of languages L for which there are two languages $L_1 \in NP$, $L_2 \in coNP$ such that $L = L_1 \cap L_2$. (Do not confuse DP with $NP \cap coNP$, which may seem superficially similar.) Show that
>
> (a) EXACT INDSET $\in \Pi_2^p$
>
> (b) EXACT INDSET $\in DP$.
>
> (c) Every language in DP is polynomial-time reducible to EXACT INDSET.

a) $(G,k) \in EI \iff \forall S \subseteq V$, $\exists S' \subseteq V$ s.t. (S' is an ind set, $|S'| = k$, and (S is an ind set $\Rightarrow |S| \leq |S'|$)). Thus $EI \in \Pi_2^p$.

b)

$$(G,k) \in EI \iff (G,k) \in INDSET \wedge (G,k+1) \notin INDSET$$

Letting $L = \{(G,k) | G$ doesn't have an ind set of size $k+1\}$, we see from the above that $EI = INDSET \cap L$. It's clear that $INDSET \in NP$ and $L \in coNP$; hence $EI \in DP$.

c) Pick any $L \in DP \Rightarrow L = L_1 \cap L_2$, where $L_1 \in NP, L_2 \in coNP$. Let ϕ_1 be the formula obtained from reducing L_1 to $3SAT$, and ϕ_2 the formula obtained from reducing L_2 to $\overline{3SAT}$. Now consider the reduction from $3SAT$ to $INDSET$ described in Figure 2.5 on pg 52 of the textbook. Augment the reduction by adding $k-1$ nodes and add edges from each of them to every other node in the graph. Then the reduction will always give a maximum ind set of size k if the formula of k clauses is satisfiable, and a maximum ind set of size $k-1$ if it's not. Let (G_1, k_1) and (G_2, k_2) be the results of

applying this reduction to ϕ_1 and ϕ_2, respectively. If k_1 and k_2 are equal, then duplicate an arbitrary clause in ϕ_1 and recompute the reduction — this will result in different values for k_1 and k_2 (we will use the fact that $k_1 \neq k_2$ later).

Thus we have: $x \in L_1 \iff \phi_1 \in 3SAT \iff (G_1, k_1) \in EI$ and $x \in L_2 \iff \phi_2 \notin 3SAT \iff (G_2, k_2) \notin INDSET \iff (G_2, k_2 - 1) \in EI$. Now define $G = G_1 \times G_2$, where $V(G) = V(G_1) \times V(G_2)$, and there's an edge in G from (u_1, u_2) to (v_1, v_2) iff there's an edge from u_1 to v_1 in G_1 or an edge from u_2 to v_2 in G_2. It's not hard to see that, for any a and b, $(G_1, a) \in EI \wedge (G_2, b) \in EI \implies (G, ab) \in EI$.

We wish to prove that $x \in L \iff (G, k_1(k_2 - 1)) \in EI$, showing that L does indeed reduce to EI. If $x \in L$, then $x \in L_1$ and $x \in L_2$, and so $(G_1, k_1) \in EI$ and $(G_2, k_2 - 1) \in EI$, implying that $(G, k_1(k_2 - 1)) \in EI$ as desired. On the other hand, if $x \notin L$, then $x \notin L_1$ or $x \notin L_2$. This gives us three possibilities:

1. $x \notin L_1 \wedge x \in L_2$: then $(G_1, k_1 - 1) \in EI$ and $(G_2, k_2 - 1) \in EI$, which means $(G, (k_1 - 1)(k_2 - 1)) \in EI$. Thus $(G, k_1(k_2 - 1)) \notin EI$, as desired.

2. $x \in L_1 \wedge x \notin L_2$: then $(G_1, k_1) \in EI$ and $(G_2, k_2) \in EI$, which means $(G, k_1 k_2) \in EI$. Thus $(G, k_1(k_2 - 1)) \notin EI$, as desired.

3. $x \notin L_1 \wedge x \notin L_2$: then $(G_1, k_1 - 1) \in EI$ and $(G_2, k_2) \in EI$, which means $(G, k_2(k_1 - 1)) \in EI$. Note that $k_2(k_1 - 1) = k_1(k_2 - 1) \iff k_1 = k_2$. Since we required earlier that $k_1 \neq k_2$, we see that $(G, k_1(k_2 - 1)) \notin EI$, as desired.

4 Problem 6.3

Describe a *decidable* language in $P_{/poly}$ that is not in P.

Define B and U_B as in the proof of the Baker-Gill-Solovay theorem. We know from Problem 3.3 on the previous problem set that we can make B decidable in exponential time. Thus U_B can also be decidable, since we can just check whether each string of a given length is in B. Furthermore, we proved in the course of the theorem that $U_B \notin P^B$, which clearly implies that $U_B \notin P$. Finally, U_B is a unary language, so it's trivially in $P_{/poly}$.

5 Problem 6.8

A language $L \subseteq \{0,1\}^*$ is sparse if there is a polynomial p such that $|L \cap \{0,1\}^n| \leq p(n)$ for every $n \in \mathbb{N}$. Show that every sparse language is in $P_{/poly}$.

Let $L^{=n}$ be the subset of L containing strings of length n. Since L is sparse, there are at most polynomially many strings of length n, *i.e.*, $|L^{=n}| \leq p(n)$. Given all the strings in $L^{=n}$ as advice (see section 6.3), we can decide in polynomial time whether an input of length n is in L, by simply looking through the advice. Now, since the advice for each n is polynomial in size (again, since L is sparse), $L \in P_{/poly}$.

6 Problem 6.9

(Mahaney's Theorem) Show that if a sparse language is NP-complete, then $P = NP$. (This is a strengthening of Exercise 2.30 of Chapter 2.)

We will follow the proof presented in the lecture notes at:
`http://www.cs.umd.edu/~jkatz/complexity/lecture6.pdf`
Consider the language $LSAT$ of pairs (ϕ, x), where x is an assignment of boolean values to all variables in the CNF ϕ, and ϕ has some satisfying assignment which is lexicographically at most x. Obviously, $LSAT$ is NP-complete since $(\phi, 1^n) \in LSAT \iff \phi \in SAT$. Let S be an NP-complete sparse language, and let f be a poly-time reduction from $LSAT$ to S. We will show that $P = NP$ by giving a poly-time algorithm for solving SAT.

Given an input ϕ with n variables, let $p(n)$ be the polynomial upper bound on the length of the reduction f applied to any $LSAT$ instance involving ϕ. Let $q(n)$ be the number of strings in S that have length at most $p(n)$. Since S is sparse and p is a polynomial, q is also a polynomial.

The algorithm for solving SAT proceeds by maintaining a set of partial assignments to variables of ϕ. At step i, the set will contain assignments to the first i variables of ϕ. We will run the algorithm for n steps, and we will prove that, if ϕ is satisfiable, then the lexicographically smallest satisfying assignment of ϕ is always reachable from one of the partial assignments at every step. At step i of the algorithm, our set is initialized to the set from step $i-1$, but with two copies of each partial assignment — one copy assigns variable i to 0, and the other assigns it to 1. We then prune this set, removing elements until the size falls below $q(n)$. It is then clear that the algorithm runs in polynomial time, as q is a polynomial.

Pruning a set of partial assignments V at step i is done as follows: first compute $Z_V = \{f(\phi, v1^{n-i}) \mid v \in V\}$. Then repeatedly apply the following rules:

1. If $f(\phi, v_1 1^{n-i}) = f(\phi, v_2 1^{n-i})$, then remove the lexicographically larger of v_1 and v_2 from V.

2. If Z_V contains more than $q(n)$ distinct values, remove the lexicographically smallest member of V.

We now just need to check that these two prunings are safe with respect to the property that, after pruning, V still contains a prefix partial assignment of the lexicographically smallest satisfying assignment of ϕ. The first pruning is safe because the reduction f maps v_1 and v_2 to the same string. This means that either there is no satisfying assignment of ϕ lexicographically smaller than either $v_1 1^{n-i}$ or $v_2 1^{n-i}$, or there is a satisfying assignment which is smaller than both. Thus removing the larger of the two cannot possibly remove the prefix of the lexicographically *smallest* satisfying assignment of ϕ.

The second pruning is safe because, if Z_V contains more than $q(n)$ distinct strings, then at least one of those strings is not in S (as S contains only $q(n)$ strings total). But note that if the lexicographically smallest v in V mapped to a string in S, then so would all the other elements of V! Thus the lexicographically smallest v in V cannot map to a string in S, and so it's safe to remove it from V.