CS–172 Computability & Complexity, Spring 2021

Homework 9 Solutions

Note: These solutions are not necessarily model answers. Rather, they are designed to be tutorial in nature, and sometimes contain more explanation than is required for full points. Also, bear in mind that there may be more than one correct solution. The maximum total number of points available is 34.

1. (a) Assuming such a function f exists, we will show that the language $L = \{(x, y) : f^{-1}(x) < y\}$ given in the hint is in NP but not in P. First note that L is in NP, because given (x, y) a nondeterministic TM can simply guess an n-bit integer z (where n is the number of bits in x) and check that $f(z) = x$ and $z < y$. Since f is computable in polynomial time, both the guessing and the checking take polynomial time. [Equivalently we can express L as $L = \{(x, y) : \exists z\, W((x, y), z)\}$, where the certificate $W((x, y), z)$ is true iff z is an integer with the same number of bits as x, y, and also $f(z) = x$ and $z < y$. Clearly W can be tested in polynomial time, so L is in NP.] *8pts*

To see that L is not in P, we show that if it were then we could compute f^{-1} in polynomial time, a contradiction to the given properties of f. Given an n-bit integer x, we can compute $f^{-1}(x)$ by binary search over n-bit integers y using queries of the form $(x, y) \in L$. The number of queries required is at most n, and each query takes polynomial time assuming $L \in \mathrm{P}$. Hence we have our contradiction, so $L \notin \mathrm{P}$.

Some students did not use binary search to compute f^{-1}; this leads to an exponential running time because there are exponentially many n-bit integers to try. Another common mistake was to make unjustified assumptions about the behavior of the TM that computes L. Finally, some students who used the "certificate" approach above were not able to phrase their argument precisely: please review the above version (and also the analogous one in part (b) below.

(b) We show in addition that $L \in$ co-NP, which together with part (a) implies that $\mathrm{NP} \cap \text{co-NP} \neq \mathrm{P}$. Showing that $L \in$ co-NP is equivalent to showing that $\overline{L} \in \mathrm{NP}$. Now we may write $\overline{L} = \{(x, y) : f^{-1}(x) \geq y\} \cup L_{\mathrm{junk}}$, where L_{junk} denotes all strings that are not of the form (x, y). But the first of these languages is in NP by the same argument as we used for L in part (a), and L_{junk} is clearly in P. Hence $\overline{L}$ is in NP. [Equivalently we can express L as $L = \{(x, y) : \forall z\, W'((x, y), z)\}$, where the certificate $W'((x, y), z)$ is true iff z is an integer with the same number of bits as x, y, and also $f(z) \neq x$ or $z < y$. Clearly W' can be tested in polynomial time, so we are in the same situation as in part (a) but with $\exists$ replaced by $\forall$. We saw in class that this is an alternative characterization of co-NP, so $L \in$ co-NP.] *4pts*

2. We first show that the problem is in PSPACE. Without loss of generality, assume that player 2 plays both the first and the last puzzle cards. (If this is not the case, add additional dummy cards with all holes punched.) Let the players' stacks be $\langle c_1^1, c_2^1, \ldots, c_k^1 \rangle$ and $\langle c_1^2, c_2^2, \ldots, c_{k+1}^2 \rangle$. Let H be the set of all currently open hole positions. *4pts*

The following algorithm decides whether player j has a winning strategy, where $j = 1$ or 2.
M= "On input H, $\langle c_1^1, c_2^1, \ldots, c_{k_1}^1 \rangle$ and $\langle c_1^2, c_2^2, \ldots, c_{k_2}^2 \rangle$ (where $k_2 = k_1$ or $k_1 + 1$) and player j:

1. If the stacks of cards $\langle c_1^1, c_2^1, \ldots, c_{k_1}^1 \rangle$ and $\langle c_1^2, c_2^2, \ldots, c_{k_2}^2 \rangle$ are both empty, then *accept* if H is empty and $j = 1$, or if H is non-empty and $j = 2$. *Reject* otherwise.

2. Let H_i^j be the set of holes in H not covered when c_1^j is placed in its i^{th} orientation for $i = 1, 2$. Call M with H_i^j, the remaining cards, and the other player. If either of the calls rejects, then for that move, the other player does not have a winning strategy, so player j has a winning strategy, and hence *accept*. Otherwise, *reject*.

To decide whether player 2 has a winning strategy, we call M with H, $\langle c_1^1, c_2^1, \ldots, c_k^1 \rangle$ and $\langle c_1^2, c_2^2, \ldots, c_{k+1}^2 \rangle$, and player 2. Since this algorithm basically searches the game tree, it decides the puzzle game. Also, note that the number of levels of recursion is $2k + 1$, and at each level we only need to store some subset of H and the orientation of one card. Thus the algorithm above is a PSPACE algorithm.

To show that the game is PSPACE-hard, we show that FORMULA-GAME is polynomial time reducible to PUZZLE. Let ϕ be an instance of FORMULA-GAME. We assume that the quantifiers in ϕ begin with a $\exists$, end in a $\forall$, and that they strictly alternate between $\exists$ and $\forall$. (If this is not originally the case, we can modify the instance by adding extra quantifiers that bind fresh new "dummy" variables). Thus ϕ is of the form: *8pts*

$$\phi \quad = \quad \exists x_1 \forall x_2 \exists x_3 \ldots \forall x_{2k} [\psi]$$

We assume also that ψ is in CNF.

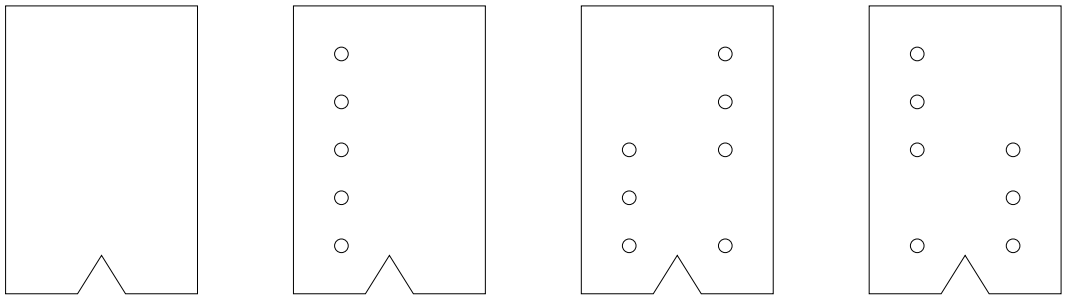

Figure 1: *Reduction to* PUZZLE. *The leftmost picture shows the shape of the box (slightly different from Sipser, but achieves the same effect that each card fits in one of two ways); the second picture shows card c'; the third and fourth show the two sides of the card c_{x_1} for variable x_1, assuming that x_1 occurs in the first two clauses and $\overline{x_1}$ occurs in the fourth clause.*

Now create a card for each quantifier/variable, and for each clause in the formula ψ create a pair of hole positions, one in each of the two columns. We punch holes in the left column of the card in every position which corresponds to a clause that *does not* contain that card's variable, and in the right column for every clause which does not contain the complement of that card's variable (see figure). The first player gets all the odd numbered cards (in the order $c_{x_1}, c_{x_3}, \ldots$) and the second player gets all the even numbered cards $c_{x_2}, c_{x_4}, \ldots, c_{x_{2k}}$. In addition, player 2's first card is a special card c' which has all the holes in the left column punched out, and the holes in the right column not punched out. It is easy to see that this reduction can be done in polynomial time.

Now we need to show that the reduction is correct. Because of symmetry, there is no loss of generality in assuming that in her first move, player 2 puts the card c' face up. Thus all the right column holes are covered. Now player 1 puts card c_{x_1} in one of its two possible configurations. The two configurations correspond to player E's two possible choices in the formula game: putting c_{x_1} face up corresponds to setting x_1 to TRUE, and putting it face down corresponds to setting x_1 to FALSE. Note that for any clause that contains x_1 (and is thus satisfied by setting x_1 to TRUE), the left column hole corresponding to it is blocked when card c_{x_1} is put face up. Similarly any clause containing $\overline{x_1}$ has its left hole covered if c_{x_1} is put face down.

It is thus clear that a player's placing a card c_{x_i} in one of its two configurations corresponds exactly to her selecting the value of variable x_i. After all the cards have been placed, the players have effectively chosen an assignment to variables $x_1, x_2, \ldots x_{2k}$. A hole in the left column is blocked only if the corresponding clause is satisfied by this assignment. Now player 1 wins if all the holes are blocked, i.e., if all the clauses are satisfied by the corresponding assignment, i.e., if the assignment satisfies ψ. Thus player 1 has a winning strategy if and only if player E has a winning strategy for ϕ.

When showing membership in PSPACE*, some students gave the correct recursion above, but didn't specify when to accept or reject. Points were deducted for this. For hardness, there were a number of incorrect reductions; if this applied to you, please review the comments on your submission and the correct reduction above.*

3. (a) To show that *IPA*$_{\text{TM}}$ is PSPACE-complete, we need to show that it is in PSPACE and is PSPACE-hard. *7pts*

We construct a machine to decide *IPA*$_{\text{TM}}$ as follows: on input $\langle M, w\rangle$ it places a marker at the $(|w| + 2)$th tape square and proceeds to simulate M on w. It also keeps a counter of the number of steps simulated so far. If M ever rejects or strays past the marker, it rejects. Otherwise, if M accepts, it accepts. Finally, since there are only $2^{O(|w|)}$ distinct configurations that M could be in while staying in the first $|w| + 1$ tape squares, if M has neither accepted not rejected after that many steps, then the *IPA*$_{\text{TM}}$ machine rejects, since by the pigeonhole principle M must have reached the same configuration twice and hence will loop forever.

The above machine requires $O(|w|)$ space to simulate M, and an additional $\log 2^{O(|w|)} = O(|w|)$ space to keep the step counter; hence it is in PSPACE.

Some students forgot the step counter, and/or didn't specify or justify what the TM should do if looping is detected.

To show that *IPA*$_{\text{TM}}$ is PSPACE-hard, we provide a method for reducing an arbitrary PSPACE problem to it. Let $L \in$ PSPACE. Then there is a machine M that decides L and is cn^k space-bounded for some fixed c and k. The reduction from L to *IPA*$_{\text{TM}}$ takes an input w for L of length n, and constructs a "padded" string w' of total length cn^k, consisting of w followed by $cn^k - n$ special characters (say '#'). The output from the reduction consists of the pair $\langle M', w'\rangle$, where M' is the TM M modified slightly so that it treats '#' characters as blanks. (We cannot use actual blank characters since they are not allowed to be part of the input.) This reduction takes polynomial time, since outputting $cn^k - n$ "blanks" is clearly polynomial, and outputting the description of M' takes constant time as it is independent of the input w. (Note that we regard M as fixed in this reduction.)

If M accepts w, then it does so using at most cn^k tape squares; by our construction, this is is the length of the new input, and so $\langle M', w'\rangle \in$ *IPA*$_{\text{TM}}$. Conversely, if $\langle M', w'\rangle \in$ *IPA*$_{\text{TM}}$ then certainly M must accept w.

Therefore this reduction is correct. Since any PSPACE problem can be reduced to *IPA*$_{\text{TM}}$, it is PSPACE-hard and hence PSPACE-complete.

[Note: this is in some sense an existential proof: for any given PSPACE machine and input, there is some space bound cn^k; you may not happen to know what it is. For a more concrete proof, you can reduce directly from a PSPACE-complete problem that you know, such as TQBF. In that case you can output a description of a specific machine that decides TQBF, such as the one on Sipser p. 340, and from that compute the exact amount of padding you need – the size of a stack frame times the number of variables in the input – and pad the input accordingly.]

(b) We are asked to show that the language *3pts*

$$A_{\text{LBA}} = \{\langle M, w\rangle : \text{the LBA } M \text{ accepts } w\}$$

is PSPACE-complete. Again, this means we have to show that it is both in PSPACE and PSPACE-hard.

The fact that it is PSPACE-hard follows immediately from the same reduction as in part (a), since an in-place TM is a special case of an LBA.

To show that A_{LBA} is in PSPACE, we simply note that it is clearly in $\text{NSPACE}(n)$ as a nondeterministic TM can simply simulate the LBA directly in the same amount of space. [Note that we are assuming here that there is some mechanism in the description of an LBA that explicitly prevents it from using more than its allowed amount of space. If we simply coded an LBA in the same way as a general TM, we would have to check that M is in fact an LBA, which is easily seen to be an undecidable problem!] Now by Savitch's theorem we conclude that $A_{\text{LBA}} \in \text{SPACE}(n^2)$, and hence clearly in PSPACE.

Some students didn't use Savitch's Theorem to convert from nondeterministic to deterministic space; those students seemed to overlook the fact that the LBA is nondeterministic.