Computability and Complexity

Assignment 5

Ari Feiglin

Exercise 5.1:

We define the following complexity classes

$$\mathbf{PSPACE} = \bigcup_{c>0} \mathsf{DSPACE}(n^c), \qquad \mathbf{NPSPACE} = \bigcup_{c>0} \mathsf{NSPACE}(n^c)$$

Prove, disprove, or show the equivalence between the following statement and an open problem:

$$\mathbf{P}^{\mathbf{PSPACE}} = \mathbf{NP}^{\mathbf{NPSPACE}}$$

We will prove this. Recall that by Savitch's theorem:

$$\mathsf{NSPACE}(O(n^c)) \subseteq \mathsf{DSPACE}\big(O(n^{2c})\big)$$

Which means that $\mathbf{NPSPACE} \subseteq \mathbf{PSPACE}$. Since the inclusion in the other direction is trivial ($\mathsf{DSPACE}(n^c) \subseteq \mathsf{NSPACE}(n^c)$), we have that

$$\mathbf{PSPACE} = \mathbf{NPSPACE}$$

We now claim

$$\mathbf{PSPACE} = \mathbf{P}^{\mathbf{PSPACE}} = \mathbf{NP}^{\mathbf{NPSPACE}}$$

Obviously we have the sequence of inclusions

$$\mathbf{PSPACE} \subseteq \mathbf{P}^{\mathbf{PSPACE}} \subseteq \mathbf{NP}^{\mathbf{NPSPACE}}$$

so we will show that $\mathbf{NP}^{\mathbf{NPSPACE}} \subseteq \mathbf{PSPACE}$. Since $\mathbf{PSPACE} = \mathbf{NPSPACE}$, this is equivalent to $\mathbf{NP}^{\mathbf{PSPACE}} \subseteq \mathbf{NPSPACE}$. Suppose $S \in \mathbf{NP}^{\mathbf{PSPACE}}$, so there exists a problem $S' \in \mathbf{PSPACE}$ and a non-deterministic oracle machine $N^{S'}$ which solves S in polynomial time. Since $S' \in \mathbf{PSPACE}$, there exists a deterministic algorithm M' which solves S' in polynomial space. We define the non-deterministic algorithm M to run $N^{S'}$ but instead of asking queries of the form $q \overset{?}{\in} S'$, it runs $M'(q)$ and checks if the return value is 1. M' solves S', so such a query is equivalent, and so M accepts x if and only if $N^{S'}$ does (meaning M can return one on x if and only if $N^{S'}$ can), so M solves S' non-deterministically. Since M' solves S' in polynomial space, the space required by each query is polynomial (in the length of the query).

Notice that the space required by running $M'(q)$ for a query $q \overset{?}{\in} S'$ is polynomial in $|q|$ since M' is a deterministic polynomial-space machine. Since $N^{S'}(x)$ is polynomial-time (and thus polynomial-space), it can only use polynomial space to store its queries, and so q must have a length bound polynomially by $|x|$, meaning the space required by $M'(q)$ is polynomial in $|x|$. We can reuse the space used by a query since once the query is finished, the space it utilized for its work is not needed, and so we can ensure that M runs in polynomial space. Essentially M runs equivalently to $N^{S'}$, except it needs an extra polynomial amount of space to simulate M' on queries, and this means M runs in polynomial space.

Thus M is a non-deterministic polynomial-space machine which solves S, so $S \in \mathbf{NPSPACE} = \mathbf{PSPACE}$. This means that $\mathbf{NP}^{\mathbf{NPSPACE}} \subseteq \mathbf{PSPACE} \subseteq \mathbf{P}^{\mathbf{PSPACE}}$, so we have shown

$$\mathbf{PSPACE} = \mathbf{P}^{\mathbf{PSPACE}} = \mathbf{NP}^{\mathbf{NPSPACE}}$$

Exercise 5.2:

A **connected component** of an undirected graph $G = (V, E)$ is a set of vertices $S \subseteq V$ such that for every two vertices v, u in S, there exists a path from v to u and for every two vertices $v \in S$ and $u \notin S$, there is no path from v to u. Show that the following problem is in **NL**

$$\mathsf{2Components} = \left\{ G \;\middle|\; \begin{array}{l} G \text{ is an undirected graph whose set of vertices can be partitioned into exactly two} \\ \text{connected components} \end{array} \right\}$$

The idea for the non-deterministic log-space algorithm is as follows: find two vertices which are not connected, and verify that all other vertices are connected to one of them. I will write this in pseudocode, but there are some nuances that require explanation afterward

```
1.  function M(G = (V, E))
2.      v ∈ V   ▷ This need not be non-deterministic
3.      u ← ∅
4.      for (w ∈ V)
5.          if (v and w are disconnected)  u ← w
6.      end for
7.      if (u = ∅)  return 0
8.      for (w ∈ V)
9.          if (v and w are connected)  continue
10.         if (u and w are connected)  continue
11.         return 0
12.     end for
13.     return 1
14. end function
```

Now, how do we check if two vertices are connected or disconnected?

(1) To check if two vertices v and w are connected, we know that the problem $\mathsf{st\text{-}conn} \in \mathbf{NL}$, and so we can simply employ the algorithm used by $\mathsf{st\text{-}conn}$ to check if v and w are connected. In other words, we are checking if $(G, v, w) \in \mathsf{st\text{-}conn}$, and since $\mathsf{st\text{-}conn} \in \mathbf{NL}$ this can be done non-deterministically. Notice that since this algorithm is non-deterministic, we cannot get false positives; ie. if v and w are disconnected, this will always return 0. If v and w are connected, this can still return 0 (but there has to be a sequence of decisions that can be made to return 1).

(2) To check if two vertices v and w are disconnected, this is essentially asking if $(G, v, w) \in \mathsf{st\text{-}conn}^c$. Now, recall that $\mathbf{NL} = \mathsf{co}\mathbf{NL}$ so $\mathsf{st\text{-}conn}^c \in \mathbf{NL}$, meaning we can employ a non-deterministic log-space algorithm to check if v and w are disconnected. This again, as a non-deterministic algorithm, cannot return false positives.

Note:

$\mathsf{st\text{-}conn}$ is defined for directed graphs, but the same algorithm which shows that $\mathsf{st\text{-}conn} \in \mathbf{NL}$ shows that its variant for undirected graphs is in **NL** as well. The algorithm just randomly attempts to build a path from s to t by non-deterministically choosing the next vertex in the path at most $|V|$ times. It starts at the vertex s, and verifies that the current vertex is connected to the next. If at any point it reaches t, it returns one. If after choosing $|V|$ vertices it hasn't reached t, it returns zero.

So this explains how lines 5, 9, and 10 function. This algorithm requires space to store three vertices: v, u, and w, as well as the space to check if vertices are connected or disconnected. As explained above, checking if two vertices are connected or disconnected requires log-space, and we can reuse the space required by these queries. So all in all this algorithm has logarithmic space complexity.

Now we claim that M solves $\mathsf{2Components}$. If $G \in \mathsf{2Components}$, there exist precisely two connected components. The algorithm starts by choosing some $v \in V$, and it is an element of one of these connected components. Since there exists another connected component, there exists some w for which v and u are disconnected. So now as M iterates over V, once it gets to w, since checking if two vertices are disconnected is non-deterministic, there exists some sequence of decisions it can make when it checks that w and v are disconnected in order to get that they are. So it sets u to w, and from here on out u is no longer empty/null, so line 7 does not return zero. And since checking if two vertices are disconnected cannot return a false positive, at the end of that first for loop, u is some vertex which is disconnected from v.

Now, as M once again iterates over $w \in V$ for the second for loop, since G has precisely two connected components, w is connected to either v or u. Since checking if vertices are connected is done non-deterministically, there exists some sequence of choices that can be made for each $w \in V$ when checking in order for M to affirm that w is connected to v or it is connected to u. So for every $w \in V$ there is a sequence of choices which can be made in order for either line 9 or 10 to continue to the next iteration of the for loop, meaning the algorithm will return 1.

So if $G \in \mathsf{2Components}$, there exists a sequence of choices which M can make for it to return 1. In other words, if $G \in \mathsf{2Components}$ then M accepts G.

Now if $G \notin \mathsf{2Components}$, there are two reasons for this:

(1) G has only one connected component, meaning all vertices in G are connected to one another. In this case, since line 5 cannot have false positives and every $w \in V$ is connected to v, u is never altered. So at line 7, $u = \varnothing$ and so M returns zero.

(2) If G has more than two connected components. In this case M can make choices so that $u = \varnothing$ so at line 7 it returns zero, but it can also make choices and find a u which is disconnected from v. But since there exists more than two connected components, there exists a w which is disconnected from both v and u. So once the second for loop iterates over w, both line 9 and 10 will fail (it won't enter the if block) since checking if vertices are disconnected cannot give false positives, and so M will return zero. So if G has more than two connected components, no matter what choices M makes it will return zero.

So if $G \notin \mathsf{2Components}$, no matter what choices M makes, $M(G)$ will return zero as required. So $G \in \mathsf{2Components}$ if and only if M accepts G. Therefore M is a non-deterministic log-space algorithm which solves $\mathsf{2Components}$, meaning $\mathsf{2Components} \in \mathbf{NL}$ as required.

Exercise 5.3:

For each of the following statements, either prove it, disprove it, or show it implies an answer to an open question.

(1) $\mathbf{BPP}_{1/2} = \mathbf{BPP}$

(2) For every polynomial $p \geq 6$, $\mathbf{BPP}_{1/2+1/p} = \mathbf{BPP}$

(3) For every polynomial $p \geq 2$, $\mathbf{BPP}_{1/2+2^{-p}} = \mathbf{BPP}$

(1) This is false. Let us define the algorithm $N(x)$ which randomly decides between a value of 0 or 1, and returns that value. For any decision problem S, and for every string x, the probability that $N(x)$ returns the correct answer is $\frac{1}{2}$. Therefore $S \in \mathbf{BPP}_{1/2}$ meaning $\mathbf{BPP}_{1/2} = \mathcal{P}(\{0,1\}^*)$.

In particular, $\mathbf{BPP}_{1/2}$ contains undecidable decision problems. But we know $\mathbf{BPP} \subseteq \Sigma_2$ which contains only decidable decision problems, and so $\mathbf{BPP} \neq \mathbf{BPP}_{1/2}$.

(2) This is true. Since $\frac{2}{3} \geq \frac{1}{2} + \frac{1}{p(n)}$, we have that

$$\mathbf{BPP} = \mathbf{BPP}_{2/3} \subseteq \mathbf{BPP}_{1/2+1/p}$$

Now we will show inclusion in the other direction. Suppose $S \in \mathbf{BPP}_{1/2+1/p}$ so there exists a probabilistic algorithm M which returns the correct answer with a probability of no less than $\frac{1}{2} + \frac{1}{p(n)}$. Let $k(n)$ denote some function for now, and we will amplify M like we did in lecture:

function $M'(x)$
    $c_0, c_1 \leftarrow 0$
    repeat $k(|x|)$ **times**
        if $(M(x) = 1)$ $c_1 \leftarrow c_1 + 1$
        else $c_0 \leftarrow c_0 + 1$
    end repeat
    return 1 **if** $c_1 > c_0$, **else** 0
end function

So now we will find a suitable function $k(n)$ which will ensure that M' runs in polynomial time and gives the correct answer with a probability of no less than $\frac{2}{3}$. Let ω_i indicate that $M(x)$ gave the wrong answer on the ith iteration. Then $M'(x)$ will return the wrong answer only if $\sum_{i=1}^{k} \omega_i > \frac{1}{2}k$ (we can assume k is odd), as this would mean that the wrong counter c_i is larger than the other counter. Eg. if $x \in S$ then $M(x)$ will return 0 if and only if $c_0 > c_1$, meaning $M(x)$ gave the wrong answer (and so incremented c_0 instead of c_1) more times than it gave the correct answer (which increments c_1), which is if and only if it gave the wrong answer more than half the time.

Now, we know that since ω_i is an indicator variable

$$\mathbb{E}[\omega_i] = \mathbb{P}(\omega_i = 1) = \mathbb{P}(M(x) \text{ is wrong}) \leq \frac{1}{2} - \frac{1}{p(n)}$$

This means that

$$\mu = \mathbb{E}\left[\frac{1}{k}\sum i = 1^k \omega_i\right] = \frac{1}{k}\sum_{i=1}^{k}\mathbb{E}[\omega_i] \leq \frac{1}{k}\cdot k \cdot \left(\frac{1}{2} - \frac{1}{p(n)}\right) = \frac{1}{2} - \frac{1}{p(n)}$$

Now, we have that

$$\mathbb{P}(M'(x) \text{ is wrong}) = \mathbb{P}\left(\frac{1}{k}\sum_{i=1}^{k}\omega_i > \frac{1}{2}\right) = \mathbb{P}\left(\frac{1}{k}\sum_{i=1}^{k}\omega_i > \mu + \frac{1}{p(n)}\right)$$

By Chernoff's inequality this is bound by

$$\leq e^{-\frac{2k}{p^2}}$$

Now we want $M'(x)$ to be wrong with a probability of $\leq \frac{1}{3}$, so we can require $e^{-\frac{2k}{p^2}} = 3^{-1}$, so we can set

$$k(n) = \frac{\ln 3}{2}p(n)^2$$

and we get the desired probability.

Since $k(n)$ is a polynomial, $M'(x)$ runs $M(x)$ a polynomial number of times, and since $M(x)$ is polynomial-time, this means that $M'(x)$ is also polynomial-time. Therefore M' is a polynomial-time probabilistic algorithm which gives solves S, giving the correct answer with a probability of $\geq \frac{2}{3}$, meaning $S \in \mathbf{BPP}$. So we have that $\mathbf{BPP}_{1/2+1/p} \subseteq \mathbf{BPP}$ and therefore $\mathbf{BPP}_{1/2+1/p} = \mathbf{BPP}$ as required.

(3) We will show that this implies $\mathbf{NP} \subseteq \mathbf{BPP}$ (and so $\mathbf{NP} = \mathbf{RP}$). Firstly, $\mathbf{BPP}$ is closed under Karp reductions. This is quite simple: suppose $S \in \mathbf{BPP}$ and that there exists a Karp reduction f from some decision problem S' to S. Since $S \in \mathbf{BPP}$, there exists a probabilistic polynomial-time algorithm M for which for every x, $M(x)$ is correct with a probability of $\geq \frac{2}{3}$. We define the probabilistic machine M' such that for every x, $M'(x) = M(f(x))$. Since computing f takes polynomial time, and M is polynomial-time, so $M'(x)$ takes polynomial time in $|f(x)|$ which is bound by a polynomial of $|x|$, meaning M' takes polynomial time. And since $x \in S'$ if and only if $f(x) \in S$, we have

$$\mathbb{P}(M'(x) \text{ is correct}) = \mathbb{P}(M(x) \text{ is correct}) \geq \frac{2}{3}$$

and so $S' \in \mathbf{BPP}$.

So we will show that $\mathsf{SAT} \in \mathbf{BPP}$, which means that since $\mathbf{BPP}$ is closed under Karp reductions and SAT is $\mathbf{NP}$-complete, $\mathbf{NP} \subseteq \mathbf{BPP}$. Suppose the number of variables in a formula φ is n. Let us define the algorithm

```
1.  function M(φ)
2.      choose a boolean vector of length n
3.      if (τ satisfies φ)  return 1
4.      choose x ∈ {0,1}
5.      if (x = 0)  return 0
6.      repeat n times
7.          choose x ∈ {0,1}
8.          if (x = 1)  return 1
9.      end repeat
10.     return 0
11. end function
```

We will show that $M(\varphi)$ has a probability of being correct of $\geq \frac{1}{2} + \frac{1}{2^{p(n)}}$. If $\varphi \in \mathsf{SAT}$, then worst case there is only one boolean vector of length n which satisfies φ. The probability of choosing that boolean vector is $\frac{1}{2^n}$ since there are 2^n boolean vectors of length n. If this boolean vector is not chosen, then the only other way for $M(x) = 1$ is for $M(x)$ to return 1 on line 8. These are disjoint events so

$$\mathbb{P}(M(\varphi) = 1) = \mathbb{P}(\tau \text{ satisfies } \varphi, \text{ or } M(x) \text{ returns 1 on line 8}) = \mathbb{P}(\varphi(\tau) = 1) + \mathbb{P}(M(x) \text{ returns 1 on line 8})$$

Now, the probability that $M(x)$ returns 1 on line 8 is dependent only on $\varphi(\tau)$ being false and x being 1 on line 5. Using dependent probability

$$\mathbb{P}(\text{returns 1 on line 8}) = \mathbb{P}(\text{returns 1 on line 8} \mid x = 1 \text{ on line } 5 \wedge \varphi(\tau) = 0) \cdot \mathbb{P}(x = 1 \text{ on line } 5 \wedge \varphi(\tau) = 0)$$

Now we know that if we reach line 6, ie. if $x = 1$ on line 5 and $\varphi(\tau) = 0$, then the probability we don't return 1 is going to be the probability that at each iteration, $x = 0$. This has a probability of $\frac{1}{2^n}$, and so

$$\mathbb{P}(\text{returns 1 on line 8} \mid x = 1 \text{ on line 5} \wedge \varphi(\tau) = 0) = 1 - \frac{1}{2^n}$$

Now, using conditional probability again

$$\mathbb{P}(x = 1 \text{ on line 5} \wedge \varphi(\tau) = 0) = \mathbb{P}(x = 1 \text{ on line 5} \mid \varphi(\tau) = 0) \cdot \mathbb{P}(\varphi(\tau) = 0) = \frac{1}{2}\,\mathbb{P}(\varphi(\tau) = 0)$$

since x is chosen uniformly from $\{0, 1\}$. If we set $\alpha = \mathbb{P}(\varphi(\tau) = 1)$ then we get that $\alpha \geq \frac{1}{2^n}$ as explained earlier, and

$$\mathbb{P}(M(\varphi) = 1) = \alpha + \left(1 - \frac{1}{2^n}\right) \cdot \frac{1}{2}(1 - \alpha) = \alpha\left(\frac{1}{2} + \frac{1}{2^{n+1}}\right) + \frac{1}{2} - \frac{1}{2^{n+1}}$$

Since $\alpha \geq \frac{1}{2^n}$, this is greater than

$$\geq \frac{1}{2^{n+1}} + \frac{1}{2^{2n+1}} + \frac{1}{2} - \frac{1}{2^{n+1}} = \frac{1}{2} + \frac{1}{2^{2n+1}}$$

Now if $\varphi \notin \mathsf{SAT}$, then τ will not satisfy φ so the probability $M(\varphi) = 0$ is the probability $x = 0$ on line 5 or the probability that at every iteration on line 8, $x = 0$. These events are disjoint and so

$$\mathbb{P}(M(\varphi) = 0) = \mathbb{P}(x = 0 \text{ on line 5}) + \mathbb{P}(\text{on every iteration, } x = 0 \text{ on line 8})$$

Now we know that on line 5, the probability $x = 0$ is $\frac{1}{2}$. Using conditional probability,

$$\mathbb{P}(\text{on every iteration, } x = 0 \text{ on line 8}) = \mathbb{P}(\text{on every iteration, } x = 0 \text{ on line 8} \mid x = 1 \text{ on line 5}) \cdot \mathbb{P}(x = 1 \text{ on line 5})$$

So if we get to line 6 (ie. $x = 1$ on line 5), then the probability that on every iteration $x = 0$ on line 8 is $\frac{1}{2^n}$. Thus we get that this is equal to

$$= \frac{1}{2^n} \cdot \frac{1}{2}$$

And so

$$\mathbb{P}(M(\varphi) = 0) = \frac{1}{2} + \frac{1}{2^{n+1}} \geq \frac{1}{2} + \frac{1}{2^{2n+1}}$$

So we have shown that for every fomula φ,

$$\mathbb{P}(M(\varphi) \text{ is correct}) \geq \frac{1}{2} + \frac{1}{2^{2n+1}}$$

And so this means that M shows that $\mathsf{SAT} \in \mathbf{BPP}_{1/2+2^{-(2n+1)}} = \mathbf{BPP}$. And as we said above, since $\mathbf{BPP}$ is closed under Karp reductions and SAT is $\mathbf{NP}$-complete, this implies $\mathbf{NP} \subseteq \mathbf{BPP}$.