Computability and Complexity

Assignment 4

Ari Feiglin

Exercise 4.1:

Prove, disprove, or show an equivalence between an open question: $\mathbf{P}/_{\mathsf{poly}}$ is closed under Karp reductions.

We will first prove the following lemma

Lemma:

Let $\mathcal{C}$ be a class of decision problems, then $\mathbf{P}^{\mathcal{C}}$ is closed under Karp reductions.

Proof:

Let S be a decision problem in $\mathbf{P}^{\mathcal{C}}$ and let f be a Karp reduction from another decision problem S' to S. Since S is in $\mathbf{P}^{\mathcal{C}}$, there exists a decision problem $S_0 \in \mathcal{C}$ and a polynomial-time oracle machine A^{S_0} which solves S. Now, we claim that $S' \in \mathbf{P}^{S_0}$ as we can define the polynomial-time oracle machine $B^{S_0}(x)$ which simply runs $A^{S_0}(f(x))$ (using its access to an oracle for S_0). Computing $f(x)$ takes polynomial time, and running A^{S_0} on $f(x)$ takes polynomial time: it takes polynomial time in $|f(x)|$, but since $f(x)$ takes polynomial time, $|f(x)|$ is polynomial in $|x|$, so $A^{S_0}(f(x))$ takes polynomial time in $|x|$.

B^{S_0} obviously decides S', as $x \in S'$ if and only if $f(x) \in S'$, which is if and only if $B^{S_0}(x) = A^{S_0}(f(x)) = 1$.

So B^{S_0} is a polynomial-time oracle machine which decides S', and therefore $S' \in \mathbf{P}^{S_0} \subseteq \mathbf{P}^{\mathcal{C}}$ as required. ■

Now, since

$$\mathbf{P}/_{\mathsf{poly}} = \mathbf{P}^{\mathsf{Sparse}}$$

by the above lemma, $\mathbf{P}/_{\mathsf{poly}}$ is closed under Karp reductions, as required.

Exercise 4.2:

Prove that $\mathsf{co}(\mathbf{NP} \cap \mathsf{Sparse}) \subseteq \mathbf{NP}/_{\mathsf{log}}$, where $\mathbf{NP}/_{\mathsf{log}}$ is defined analogously to $\mathbf{P}/_{\mathsf{poly}}$ but the algorithms used are non-deterministic.

Let $S \in \mathsf{co}(\mathbf{NP} \cap \mathsf{Sparse})$, which means that $S^c \in \mathbf{NP} \cap \mathsf{Sparse}$. Since $S^c \in \mathbf{NP}$ there exists a non-deterministic algorithm A which decides S^c in polynomial time, suppose its runtime is bound by the polynomial $p(n)$. And since $S^c \in \mathsf{Sparse}$, there exists a polynomial $q(n)$ such that for every n, $|S^c \cap \{0,1\}^n| \leq q(n)$.

Our goal is to define a non-deterministic polynomial-time algorithm $B(a,x)$ and a sequence of advice $\{a_n\}_{n=0}^{\infty}$ such that the lengths of a_n are logarithmic in n and for every x,

$$x \in S \iff B(a_{|x|}, x) = 1$$

Let us define a_n to be the number of elements in S^c of length n, ie

$$a_n = |S^c \cap \{0,1\}|$$

This means that $a_n \leq q(n)$, and so we require at most $\log(q(n))$ bits for a_n. Since $q(n)$ is a polynomial, this means that $\log(q(n))$ is logarithmic in n, (ie. $|a_n| \leq \log(q(n)) \in O(\log(n))$). So $\{a_n\}_{n=0}^{\infty}$ is logarithmic advice. Now let us define B:

1. **function** $B(a,x)$
2. $\quad n \leftarrow |x|$
3. $\quad$ **if** $(|a| > \log(q(n)))$ **return** 0 $\quad \triangleright$ *a is not a valid piece of advice.*
4. $\quad$ **for** $(i = 1$ **to** $a)$
5. $\quad\quad$ **choose** $x_i \in \{0,1\}^n$
6. $\quad\quad$ **if** $(x_i = x_j$ for some $j < i)$ **return** 0
7. $\quad\quad$ **if** $(x_i = x)$ **return** 0
8. $\quad\quad$ **if** $(A(x_i) = 0)$ **return** 0

9. **end for**
10. **return** 1
11. **end function**

B is polynomial-time, as each step takes polynomial time: choosing x_i takes n time, checking if $x_i = x_j$ over all $j < i$ takes n^2 time, checking if $x_i = x$ takes n time, and running $A(a, x_i)$ takes polynomial time. And these steps are repeated a times, and a refers to a number $\leq q(n)$. So all in all this takes $O\Big(q(n)\big(p(n)+n^2\big)\Big)$ time, which is polynomial.

Now, if $x \in S$ then there are choices B can make such that $x_1, \ldots, x_{a_n}$ are all the values in $S^c \cap \{0,1\}^n$, and since there are exactly a_n such values they will all be distinct. And furthermore if B makes these choices, it will also verify that $x_i \neq x_j$ and $x_i \neq x$ (since $x \notin S^c$). And since $x_i \in S^c$, there exists a run of $A(x_i)$ which returns one. So there exist choices such that $B(a_n, x) = 1$, as required.

Now, if $B(a_{|x|}, x) = 1$ then the $x_1, \ldots, x_{a_n}$ must all be distinct from one another and from x, and $A(x_i) = 1$ so $x_i \in S^c \cap \{0,1\}^n$. Since there are only a_n values in $S^c \cap \{0,1\}^n$ meaning $S^c \cap \{0,1\}^n = \{x_1, \ldots, x_{a_{|x|}}\}$ and since $x \neq x_i$ this means that $x \notin S^c \cap \{0,1\}^n$ and so $x \notin S^c$ (since $|x| = n$), meaning $x \in S$.

So we have shown that

$$x \in S \iff B(a_{|x|}, x) = 1$$

So B decides S. And since a_n is logarithmic in n, and B is a polynomial-time advised non-deterministic algorithm which decides S, this means that

$$S \in {}^{\mathbf{NP}}\!/_{\mathsf{log}}$$

so we have shown that for every $S \in \mathsf{co}(\mathbf{NP} \cap \mathsf{Sparse})$, $S \in {}^{\mathbf{NP}}\!/_{\mathsf{log}}$, meaning $\mathsf{co}(\mathbf{NP} \cap \mathsf{Sparse}) \subseteq {}^{\mathbf{NP}}\!/_{\mathsf{log}}$ as required.

Exercise 4.3:

Given a directed graph $G = (V, E)$, and suppose $V = (v_1, \ldots, v_n)$. Then we say that G is *sorted* if for every edge $(v_i, v_j) \in E$, $i < j$. Let us define the decision problem

$$\mathsf{Sorted\text{-}st\text{-}conn} = \{G \mid G = (V, E) \text{ where } V = (v_1, \ldots, v_n) \text{ is sorted, and there exists a path from } v_1 \text{ to } v_n.\}$$

Prove that Sorted-st-conn is **NL**-complete

Note:

The original question defined Sorted-st-conn weirdly, and I didn't immediately understand what the definition meant. But in any case by solving the question as phrased above, I will also solve the original question.

Let us call the decision problem as defined in the original question Sorted-st-conn$'$. The definition I gave of Sorted-st-conn can obviously be reduced to Sorted-st-conn$'$, as $G \in$ Sorted-st-conn if and only if $(G, 1, \ldots, |V|) \in$ Sorted-st-conn$'$. And Sorted-st-conn$'$ also in **NL**, since we can verify that G is sorted (details are given below), and we can check that v_i and v_j are connected in logarithmic So if Sorted-st-conn is **NL**-complete, so is Sorted-st-conn$'$ as required.

Let us first show that Sorted-st-conn is in **NL**. So given a graph $G = (V, E)$ what we do is we simply iterate over every edge in E and verify that the first node comes before the second node in V. We can do this by storing a pointer to the current edge, and then iterating over the vertices in V. If while iterating over V we reach the first vertex in the edge, we go to the next edge. And if we reach the second vertex, we return zero. If we make it through every edge, then we return one. So we will return one if for every edge, the first vertex comes before the second in V, meaning we return one if and only if G is sorted. This means that verifying that G is sorted can be done deterministically in logarithmic space.

Next we must verify that there exists a path from v_1 to v_n, and since we know that st-conn $\in$ **NL** this can be done non-deterministically in logarithmic space. So we must verify that G is both sorted and that there exists a path from v_1 to v_n, and since these both can be done in logarithmic space using a non-deterministic algorithm, Sorted-st-conn $\in$ **NL** as required.

Now we will show that Sorted-st-conn is **NL**-complete. We will do this by defining a log-space reduction from st-conn to Sorted-st-conn. Suppose we have an input (G, s, t) for st-conn, suppose $G = (V, E)$ and let $n = |V|$. Then let us define $G' = (V', E')$ as follows:

(1) For every vertex $v \in V$, add $|V|$ copies of v, denoted $v^1, \ldots, v^n$.

(2) For every vertex $v \in V$, add edges (v^i, v^{i+1}) for every $1 \leq i < n$.

(3) For every edge $(u, v) \in V$, add edges (u^i, v^{i+1}) for every $1 \leq i < n$.

Suppose $V = \{v_1, \ldots, v_n\}$ where $v_1 = s$ and $v_n = t$. Then we order V' as

$$V' = (v_1^1, \ldots, v_n^1, v_1^2, \ldots, v_n^2, \ldots, v_1^n, \ldots, v_n^n)$$

Then $G' = (V', E')$ is sorted as for every edge in E', it is either of the form (v^i, u^{i+1}) and v^i comes before u^{i+1} in V', since the ith layer of V comes before the $i+1$th layer in V'.

G' can be computed in logarithmic space, as in order to construct it we iterate over V n times, and on the ith iteration we create the ith layer (ie. $v_1^i, \ldots, v_n^i$). We must start the iteration at s and end it at t, but this can be done with a constant number of pointers as well. This creates V', and it only requires a constant number of pointers, so this step requires $O(\log(|G|))$ space. To create E, we iterate over V and for every $v \in V$ we add the edges (v^i, v^{i+1}) for $1 \leq i < n$, this requires a pointer of $\log(n)$ bits to point to the current vertex, as well as a counter of $\log(n)$ bits for i. Then we iterate over the edges in E and for every edge (u, v), we add the edges (u^i, v^{i+1}) for $1 \leq i < n$, this requires a pointer of $\log(|G|)$ bits to point to the current edge, as well as a counter of $\log(n)$ bits for i. So all in all we need a logarithmic number of bits with respect to the input, $|G|$.

Now we claim that $(G, s, t) \in$ st-conn if and only if $G' \in$ Sorted-st-conn. If $(G, s, t) \in$ st-conn, then there exists a path from s to t. We can assume that this path is simple (by removing cycles), so it has a length bound by $|V| = n$. So there exists a path of the form

$$s = u_1 \to v_2 \to \cdots \to u_m = t, \quad m \leq n$$

Then for every i, since (u_i, u_{i+1}) is an edge in G, (u_i^i, u_{i+1}^{i+1}) is an edge in G'. So

$$u_1^1 \to u_2^2 \to \cdots \to u_m^m$$

is a path from u_1^1 to u_m^m in G'. Recall that $u_1 = s$ so $u_1^1 = s^1 = v_1^1$ and $u_m = t$ so $u_m^m = t^m = v_n^m$. Now, we can continue this path

$$u_1^1 \to u_2^2 \to \cdots \to u_m^m \to u_m^{m+1} \to \cdots \to u_m^n$$

since (u_m^i, u_m^{i+1}) is an edge in G'. Since $u_m = t = v_n$ we get that $u_m^n = v_n^n$ and so this forms a path from $v_1^1 = u_1^1$ to $v_n^n = u_m^n$, which are the first and last vertices in V' respectively. Thus $G' \in$ Sorted-st-conn as required.

Now suppose $G' \in$ Sorted-st-conn, so there exists a path from v_1^1 to v_n^n. The path will be of the form

$$v_{i_1}^{\ell_1} \to \cdots \to v_{i_1}^{\ell_1} \to v_{i_2}^{\ell_1+1} \to \cdots \to v_{i_2}^{\ell_2} \to \cdots \to v_{i_m}^{\ell_{m-1}+1} \to \cdots \to v_{i_m}^{\ell_m}$$

Where $i_1 = 1$, $\ell_1 = 1$, $i_m = n$, and $\ell_m = n$. This means that for every $1 \leq j < m$, $(v_{i_j}^{\ell_j}, v_{i_{j+1}}^{\ell_j+1})$ is an edge in G', so $(v_{i_j}, v_{i_{j+1}})$ is an edge in G. This means that the following is a path in G:

$$v_{i_1} \to v_{i_2} \to \cdots \to v_{i_m}$$

And since $v_{i_1} = v_1 = s$ and $v_{i_m} = v_n = t$, this means that s and t are connected in G. So $(G, s, t) \in$ st-conn.

Therefore $(G, s, t) \in$ st-conn if and only if $G' \in$ Sorted-st-conn So $(G, s, t) \mapsto G'$ is a log-space reduction from st-conn to Sorted-st-conn, and since st-conn is **NL**-complete and Sorted-st-conn $\in$ **NL**, this means that Sorted-st-conn is **NL**-complete, as required.

Exercise 4.4:

For the following statements either prove, disprove, or show that they are equivalent to an open question.

(1) **NL** is closed under Kleene closures.

(2) **L** is closed under Kleene closures.

(1) This is true. Suppose $A \in$ **NL**, then we will prove that A^* is in **NL** as well. Suppose $N(x)$ is a non-deterministic log-space algorithm which solves A. Then let us define

```
1. function N*(x)
2.     ℓ ← 1
3.     while (ℓ ≤ |x|)
4.         choose h ∈ [ℓ, |x|]
5.         if (N(x[ℓ : h]) = 0)  return 0    ▷ x[ℓ : h] includes x[ℓ] and x[h].
6.         ℓ ← h + 1
7.     end while
8.     return 1
```

9. **end function**

This algorithm is log-space as we can simply store pointers for ℓ and h and pass them to N, and we can alter N's code so that indexing x beyond $[\ell : h]$ gives an empty character. The pointers for ℓ and h are bound by $|x|$ and thus require $\log|x|$ space. So all in N^* requires only $O(\log|x|)$ space, as required.

If $x \in A^*$ then suppose $x = x_1 \cdots x_t$ then if at the ith iteration of the while loop we choose $h = |x_1 \cdots x_i|$, then at first $x[\ell : h] = x_1$ and then $x[\ell : h] = x_2$ and so on. And so for every iteration of the while loop there is a run of $N\big(x[\ell : h]\big)$ which accepts. And so there are choices N^* can make (choosing $h = |x_1 \cdots x_i|$ on every iteration, and the choices necessary for N to accept) which will have N^* accept x.

Now suppose that there exists a run such that $N^*(x) = 1$ then let h_i be the ith choice for h, and suppose the while loop iterates t times. Let $h_0 = 0$. Then this means that on the first iteration the input to N is $x[h_0 + 1 : h_1]$ and then on the second iteration the input is $x[h_1 + 1 : h_2]$ and so on. So we have that for every $0 \leq i < t$, $N\big(x[h_i + 1 : h_{i+1}]\big) = 1$. Thus for every $0 \leq i < t$, $x[h_i + 1 : h_{i+1}] \in A$. Notice that the final ℓ is equal to $h_t + 1$ and since $h_t \leq |x|$ and $\ell > |x|$, we have that $h_t = |x|$. This means that

$$x = x[h_0 : h_1] \# x[h_1 + 1 : h_2] \# \cdots \# x[h_{t-1} + 1 : h_t]$$

And since $x[h_i + 1, h_{i+1}] \in A$, we have that $x \in A^*$.

So N^* accepts x if and only if $x \in A^*$. Since N^* is a log-space non-deterministic algorithm, we have that A^* is in **NL**, as required.

(2) We claim that **L** is closed under Kleene closures if and only if **L** = **NL**. If **L** = **NL**, by above **NL** is closed under Kleene closures and therefore so is **L**. Now, suppose **L** is closed under Kleene closures, then we will show that there exists a decision problem $A \in \mathbf{L}$ such that there exists a log-space reduction from st-conn to S^*. Since **L** is closed under Kleene closures, this means that $S^* \in \mathbf{L}$ and since st-conn is **NL**-complete, this would mean that S^* is also **NL**-complete and in **L**, which would mean **L** = **NL**.

So suppose that **L** is closed under Kleene closures. Let us define

$$A = \{v\omega(v, u) \mid \omega \text{ is a binary string}\}$$

(We assume that you can differentiate between strings of the form v, (w, u), and ω. This can be accomplished in many different ways, for example by utilizing another character which acts as a delimiter.) We will call strings of the form v "vertices", and (v, u) "edges". A is obviously in **L**, since all we need to do is compare the first vertex in the string with the first vertex in the edge at the end of x (we must also verify that the string begins with a vertex and ends with an edge). So we only need two pointers, and each pointer requires $\log(n)$ bits so all in all this algorithm requires $O(\log(n))$ space. And so this algorithm decides **L** in logarithmic time, meaning $A \in \mathbf{L}$.

Now we will show a log-space reduction from st-conn to A^*. Let $(G, s, t) \in$ st-conn. We can assume that s has an out-degree of one and an in-degree of zero, and similarly t has an in-degree of one and an out-degree of zero. We can do this by adding two new vertices, s' and t' as well as the edges (s', s) and (t, t') to get a new graph G'. Then $(G, s, t) \in$ st-conn if and only if $(G', s', t') \in$ st-conn, this is a log-space reduction (to the decision problem of st-conn where $\textsf{in-deg}(s) = \textsf{out-deg}(t) = 0$ and $\textsf{out-deg}(s) = \textsf{in-deg}(t) = 1$).

So assuming these conditions on the degrees of s and t, let us define the log-space reduction from st-conn to A^*. Suppose $E = \{e_1, \ldots, e_m\}$ where $e_i = (v'_i, v_i)$, and suppose that $e_1 = (s, v_1)$ and $e_m = (v'_m, t)$, then we map (G, s, t) to

$$(se_1v_1\, e_2v_2\, \ldots\, e_mt)^{|V|-1}(se_1v_1\, \ldots\, e_m)$$

Let us denote this string by ω^G. Constructing ω^G can be done by simply iterating over E, and so can be done in logarithmic time. If $(G, s, t) \in$ st-conn then suppose a simple path from s to t is

$$s \to v_{i_1} \to v_{i_2} \to \cdots \to v_{i_{k-1}} \to v_{i_k} = t$$

Since all the edges of G are of the form (v'_i, v_i), $v_{i_{j-1}} = v'_{i_j}$. Since the length of the path is bound by $|V|$, we have that

$$\begin{aligned}\omega^G &= s\omega_1(v'_{i_1}, v_{i_1})v_{i_1}\omega_2(v_{i_1}, v_{i_2})v_{i_2} \cdots v_{i_{k-1}}\omega_k(v_{i_{k-1}}, v_{i_k}) \\ &= s\,\omega_1\, e_{i_1} v_{i_1}\, \omega_2\, \cdots v_{i_{k-1}}\, \omega_k\, e_{i_k}\end{aligned}$$

This is because we can take the first instance of e_{i_1} in ω^G, and so we get $s\omega_1 e_{i_1}$. Then since e_{i_1} is followed v_{i_1}, we can take the next instance of e_{i_2} and we get $v_{i_1}\omega_2 e_{i_2}$, and so on. For the transition from $v_{i_{k-1}}$ to t, instead of taking the next instance e_{i_k} (which must exist since $k \leq |V|$ and between so far between v_{i_j} and $e_{i_{j+1}}$ we have at most skipped

to the next layer, of which there are $|V|$), we take the final instance of e_{i_k} which is the last object in ω^G (since e_{i_k} is the only edge connected to t, so it must be equal to e_m).

Since $e_{i_j} = (v_{i_{j-1}}, v_{i_j})$, $v_{i_{j-1}}\omega_j e_{i_j}$ is in A, and so $\omega^G \in A^*$.

Now, suppose that $\omega^G \in A^*$, so there exist $\omega_1, \ldots, \omega_k \in A$ such that $\omega^G = \sigma_1 \cdots \sigma_k$. Since $\sigma_k \in A$, it must be of the form $v\omega(v,u)$ and since it is followed by σ_{k+1}, we have that $\sigma_k\sigma_{k+1}$ is of the form $v\omega(v,u)v'\omega'(v',u')$. But in ω^G, edges are followed by their second vertex, so $u = v'$. So in general we get that

$$\sigma_k = v_{i_{j-1}}\omega_j e_{i_j}$$

Where $e_{i_j} = (v_{i_{j-1}}, v_{i_j})$. This means that

$$\omega^G = v_{i_0}\,\omega_1\,e_{i_1}v_{i_1}\,\omega_2\,e_{i_2}\cdots v_{i_{k-1}}\,\omega_k\,e_{i_k}$$

Since ω^G begins with s, this means $v_{i_0} = s$, and since ω^G ends with $e_m = (v'_m, t)$ but $e_{i_k} = (v_{i_{k-1}}, v_{i_k})$, so $v_{i_k} = t$. So we have that

$$(v_{i_{j-1}}, v_{i_j}) \in E$$

And therefore

$$v_{i_0} \to v_{i_1} \to \cdots \to v_{i_{k-1}} \to v_{i_k}$$

is a path in G. But since $v_{i_0} = s$ and $v_{i_k} = t$, this means there exists a path in G from s to t. Thus $(G, s, t) \in \textsf{st-conn}$.

So we have shown that $(G, s, t) \in \textsf{st-conn}$ if and only if $\omega^G \in A^*$, so $(G, s, t) \mapsto \omega^G$ is a log-space reduction from st-conn to A^*. Therefore A^* is **NL**-complete, and since it is in **L** since **L** is closed under Kleeene closures, $\mathbf{L} = \mathbf{NL}$ as required.