Tutorial 9
Space Complexity

1. Show that
   (a) **PSPACE** is closed under union, intersection, complement and Kleene Closure.

   Solution: Let $L_1, L_2 \in$ **PSPACE** and let $\mathcal{M}_1, \mathcal{M}_2$ be deterministic TMs deciding L_1, L_2 respectively in polynomial space. We describe constructions of DTMs running in polynomial space for the following languages.

   $L_1 \cup L_2$: On input x, run $\mathcal{M}_1$ on x. If it accepts, then accept. Otherwise, run $\mathcal{M}_2$ on x. If it accepts, then accept; else reject.

   $L_1 \cap L_2$: On input x, run $\mathcal{M}_1$ on x and then run $\mathcal{M}_2$ on x. If both accept, then accept; otherwise reject.

   L_1^c: On input x, run $\mathcal{M}_1$ on x and negate its output. (This can be done since $\mathcal{M}$ is deterministic).

   L_1^*: Observe that $x \in L_1^*$ iff one of the following holds: $x = \varepsilon$, $x \in L_1$, $x = wz$ such that $w \in L_1^*$ and $z \in L_1^*$. Let $x = x_1 x_2 \cdots x_n$ and let $x_{i,j}$ (for $1 \leq i \leq j \leq n$) denote the substring $x_i x_{i+1} \cdots x_j$ of x. We will use dynamic programming to build a matrix $T = (t_{i,j})$ with $t_{i,j} = 1$ if $x_{i,j} \in L_1^*$ and 0 otherwise. Essentially we consider all substrings of x, starting with substrings of length 1 and ending with all substrings of length n. Below is the pseudocode:

   ```
   Input: x = x_1 ··· x_n

     Initialise t_{i,j} to 0.
     If x = ε, then accept;
     For k = 1, ..., n:
       For i = 1, ..., n − k + 1:
         j = i + k − 1;
         Run M_1 on x_{i,j};
         If M_1 accepts, then set t_{i,j} = 1;
         Else
           For ℓ = i, ..., j − 1:
             If t_{i,ℓ} = 1, and t_{ℓ+1,j} = 1, then set t_{i,j} = 1;
     If t_{1,n} = 1, then accept; Else reject;
   ```

   We now analyze the space complexity of the above algorithm. In each inner loop, we run $\mathcal{M}_1$ once which takes polynomial space (since $L_1 \in$ **PSPACE**). Storing T requires $O(n^2)$-space and each execution of the loop requires constant number of cells to store the indices. Hence the algorithm runs in polynomial space.

   Alternatively, using the fact that **PSPACE** = **NPSPACE**, we can define a poly-space NDTM that decides L_1^* that guesses an index in x; checks whether $x_{1,i} \in L_1$ using $\mathcal{M}_1$ and then recursively checks whether $x_{i+1,n} \in L_1^*$.

   It is easy to verify that the first three run in polynomial space.

(b) **NL** is closed under union, intersection and Kleene Closure.

Solution: Union and intersection can be dealt with similar to the previous question.

For any language $L \in \mathbf{NL}$, we show that $L^* \in \mathbf{NL}$. Let $\mathcal{M}$ be an NDTM deciding L in logarithmic space. We show how to construct an NDTM deciding L^*. Let $x = x_1 \cdots x_n \in \Sigma^*$. If $x = \varepsilon$, then accept. Otherwise, non-deterministically choose $i \in \{1, \ldots, n\}$ and let $w = x_1 \cdots x_i$, $z = x_{i+1} \cdots x_n$. Run $\mathcal{M}$ with w as the input. If $\mathcal{M}$ accepts, then recursively check if $z \in L^*$. Else, reject.

The above algorithm always correctly determines if $x \in L^*$. If $x \in L^*$, then x can be written as $w_1 w_2 \cdots w_k$ for some k such that $|w_j| \geq 1$ for each j. Therefore, there exists a sequence of guesses leading to $w_1, \ldots, w_k$ in that order. Suppose that $x \notin L^*$. Then for any w with $|w| \geq 1$ such that $x = wz$, either $w \notin L$ or $z \notin L^*$ and so all branches of computation of the NDTM are rejecting. Furthermore, guessing a prefix only takes logarithmic space, that is to index into a position in x. Checking whether $w \in L$ takes logarithmic space since $\mathcal{M}$ is a logspace machine. Once w is checked, it does not have to be remembered for the recursive calls. Hence, $x \in L^*$ can be checked in log space and as a consequence $L^* \in \mathbf{NL}$.

2. A ladder is a sequence of strings $s_1, s_2, \ldots, s_k$, wherein every string differs from the preceding one in exactly one character. For example, the following is a ladder of English words, starting with "head" and ending with "free": head, hear, near, fear, bear, beer, deer, deed, feed, feet, fret, free. Let

$$\mathsf{LADDER}_{DFA} = \{\langle \mathcal{M}, s, t \rangle : \mathcal{M} \text{ is a DFA and } L(\mathcal{M}) \text{ consists of a ladder of strings starting with } s \text{ and ending with } t\},$$

where $s, t \in \Sigma^*$ and $\mathcal{M}$ is defined over the input alphabet Σ. Show that LADDER_{DFA} is in **PSPACE**.

Hint: Use the fact that **PSPACE** = **NPSPACE**.

Solution: Since **NPSPACE** = **PSPACE**, it suffices to show that $\mathsf{LADDER}_{DFA} \in \mathbf{NPSPACE}$. We describe a non-deterministic algorithm that decides LADDER_{DFA}. Given an instance $\langle \mathcal{M}, s, t \rangle$, reject if $|s| \neq |t|$. Otherwise, consider a graph G whose vertices are labeled with strings in $\Sigma^{|s|}$. There is a directed edge from vertex u to vertex v if u and v differ in exactly one character and $u, v \in L(\mathcal{M})$. Then, $\langle \mathcal{M}, s, t \rangle \in \mathsf{LADDER}_{DFA}$ iff there is a path from s to t in G. This can be checked non-deterministically in polynomial space (although G has exponential number of vertices) – guess the path, storing at each step only the label of the current vertex (which is of size $|s|$). At step, say starting at vertext u, non-deterministically choose a new vertext v connected to u via an edge and check if $\mathcal{M}$ accepts v.

To ensure that the machine always halts, maintain a counter which is incremented after each guess. Reject and halt when the counter crosses $|\Sigma|^{|s|}$. Since any path from s to t (without loops) can be of length at most $|\Sigma|^{|s|}$.

3. In the generalised version of the game Tic-Tac-Toe, 2 players places marks X (crosses) and O (noughts) on an $m \times n$ grid. A player wins if she is the first to place k marks in a row, column or diagonal. The game ends in a draw if no such sequence is present when all the mn cells of the grid are filled. Assuming that X always starts, show that the language

$$\mathsf{GTICTACTOE} = \{\langle m, n, k, c \rangle : c \text{ is an intermediate configuration on the } m \times n \text{ board with next move by } X \text{ and } \exists \text{ a winning strategy for } X\}$$

is in **PSPACE**.

Solution: Given an instance $\langle m, n, k, c\rangle$, we will describe a recursive procedure to check if there exists a winning strategy for X. The validity of c may be checked in polynomial space by checking that there are equal number of crosses and noughts. Further, it is possible to check that the game has not already ended by determining whether there exists a series of k crosses or noughts in a row, column or diagonal (this would take $O(nm)$-space since there are nm possible cells and $k < \min\{m, n\}$). This as well can be done in polynomial space. If there are k consecutive X marks, then accept. If there are k consecutive marks of both X and O or only O, reject. If there is no unoccupied cell, then reject. Otherwise, repeat the following for each unoccupied cell u in configuration c: *Put the marker X on u. If the resulting configuration d is a winning configuration for X then accept. If the grid is completely filled, then reject. Otherwise, generate configurations $e_1, e_2, \ldots, e_\ell$ obtained by marking exactly one unoccupied cell in d with O. Make a recursive call on each e_i, reusing the space for each call. If all the calls return 'accept', then accept. Otherwise, continue with checking a different unoccupid cell in c.* If all the configurations d lead to rejection, then reject. Each recursion requires remembering c, d, e_i at a time; checking whether there are k consecutive X marks ina row/column/diagonal requires $O(nm)$-space; the depth of the recursion is $O(mn)$. Based on these observations it is straightforward to see that the procedure requires space $O(m^2n^2)$-space thus implying that GTICTACTOE $\in$ **PSPACE**.

4. Let $\mathbf{polyL} = \cup_{c>0}\mathbf{DSPACE}(\log^c n)$. Let **SC** (named after Stephen Cook) be the class of languages that can be decided by deterministic machines that run in polynomial time and $\log^c n$ space for some $c > 0$.

   (a) It is an open problem whether PATH $\in$ **SC**. Why does Savitch's theorem not resolve this question?

   Solution: We know that PATH $\in$ **P** – simply use breadth-first search. We also know that it is in **NL** – nondeterministically guess a path from the source to the target vertex, keeping track of the current vertex at each step, thus taking logarithmic space. By Savitch's theorem, we have $\mathbf{NSPACE}(S(n)) \subseteq \mathbf{DSPACE}(S(n)^2)$ and hence PATH $\in$ **NL** $= \mathbf{NSPACE}(\log n) \subseteq \mathbf{DSPACE}(\log^2 n) \subseteq$ **polyL**. The property that $\mathbf{NSPACE}(\log n) \subset \mathbf{DTIME}(2^{\log n}) = \mathbf{DTIME}(n)$ is not preserved by Savitch's theorem. That is, the transformation used in Savitch's theorem to transform a non-deterministic $\log n$-space algorithm to a deterministic $\log^2 n$-space algorithm does not ensure that the resulting algorithm runs in polynomial time. Since $\mathbf{DSPACE}(\log^2 n) \subset \mathbf{DTIME}(2^{\log^2 n}) = \mathbf{DSPACE}(n^{\log n})$, all we can say about a language in $\mathbf{DSPACE}(\log^2 n)$ is that there exists a TM deciding it running in time at most $n^{\log n}$ which is not polynomial in n.

   (b) Is **SC** $=$ **polyL** $\cap$ **P**?

   Solution: Clearly, it holds that **SC** $\subseteq$ **polyL** $\cap$ **P**. The other direction is not necessarilily true. For any language $L \in$ **polyL** $\cap$ **P**, we can say that there exist Turing machines $\mathcal{M}_1, \mathcal{M}_2$ running in polylogarithmic space and polynomial time respectively that decide membership in L. This does not imply that there exists a single Turing machine that runs (simultaneously) in polynomial time and polylogarithmic space.

5. Show that 2SAT is in **NL**.

Solution: Consider a 2SAT instance $\phi = \wedge_{i=1}^{m}(x_i \vee y_i)$ over variables $u_1, \ldots, u_n$. Let G be a graph with $2n$ vertices corresponding to the literals $u_i, \neg u_i$ for each i. For each clause $x \vee y$, add directed edge $(\neg x, y)$ and $(\neg y, x)$ to G, so that (v, w) is an edge in G iff $\neg v \vee w$ is a clause in ϕ.

The edge $(\neg x, y)$ indicates that if $x = 1$ then y must be 1 in order to satisfy the clause $x \vee y$. Note that G can be constructed in logarithmic space.

Let v, w be arbitrary vertices in G. If G contains a path from v to w, the there must exist one from $\neg w$ to $\neg v$. Let the path be $w \to z_1 \to \cdots \to z_k \to v$. By our construction, edges $\neg v \to \neg z_k$, $\neg z_j \to \neg z_{j-1}$ (for $j \in [2, k]$) and $\neg z_1 \to w$ also belong to G. Hence there is a path from $\neg w$ to $\neg v$.

We now claim that ϕ is unsatisfiable iff there exists a variable u such that there are paths from u to $\neg u$ and from $\neg u$ to u. Suppose, for the sake of contradiction that, there exist paths $u \to \neg u$ and $\neg u \to u$ and ϕ has a satisfying assignment with $u = 1$ (the case $u = 0$ can be analysed similarly). Let $u \to \cdots v \to w \cdots \to \neg u$ be the path from u to $\neg u$. Since $u = 1$, all literals in the path u to w must be true. Similarly, since $\neg u = 0$, all literals in the path from w to $\neg u$ must be false. Since there is an edge (v, w) and $v = 1$ and $w = 0$, the clause $\neg v \vee w$ is not satisfied, thus contradicting our assumption that ϕ is satisfiable.

Hence checking for the existence of paths $u \to \neg u$ and $\neg u \to u$ will determine whether or not ϕ is satisfiable. We know that PATH is in $\mathbf{NL}$ and we need polynomially many queries to PATH. Therefore, $\mathsf{2SAT} \in \mathbf{NL}$. Note that this simultaneously shows that $\mathsf{2SAT}$ and $\overline{\mathsf{2SAT}}$ are in $\mathbf{NL}$ since $\mathbf{NL} = \mathbf{coNL}$.