You can answer in any order that you like, but indicate the question numbers in your answers. The questions are roughly sorted by difficulty except for ★ questions, which are somewhat harder.

1 Universal Languages and Time Bounds

Place in as low a complexity class as you can. For this question, M denotes an arbitrary Turing machine. Which languages are complete for their classes? Briefly justify your answers.

(a) $L := \{(M, x) : M(x) \text{ halts}\}$

RE (recursively enumerable): run M on x using a universal Turing machine; accept iff it accepts. We did not define **RE**-completeness. But under a natural definition this language is **RE**-complete under many-one (and poly-time) reductions. Suppose $L' = L(M') \in \mathbf{RE}$ for some Turing machine M'. We can decide if an instance x is in L' or not by mapping it to an instance M^*, where M^* operates as follows. If M' loops, M^* loops. If M' accepts, M^* accepts. If M' rejects, M^* *loops.*

(b) $L := \{M : \exists x\ M(x) \text{ halts}\}$

RE: run M in parallel on all inputs; accept iff M accepts at some point. (Recall that to run a machine "in parallel" on all inputs at the i-th step, we run i steps of M for the first i inputs.) This language is also **RE**-complete, with a reduction that is similar to the one given above. Suppose $L' = L(M') \in \mathbf{RE}$ for some Turing machine M'. We can decide if x is in L' or not by mapping it to an instance $M^*[x]$, where $M^*[x]$ has x hardwired in and operates as follows. If $M'(x)$ loops, $M^*[x]$ loops. If $M'(x)$ accepts, $M^*[x]$ accepts. If $M'(x)$ rejects, $M^*[x]$ loops.

(c) $L := \{(M, x, t) : M(x) \text{ halts within time } t\}$

EXP-complete: run M on x for t steps; accept iff M accepts. The number of steps is at most exponential in the length of the input since $t \leq 2^{|(M,x,t)|}$. Every language $L' = L(M') \in \mathbf{EXP}$ reduces to L. Map an instance x for L' to instance (M', x, t) for L, where t is an upper bound on the number of steps that $M'(x)$ makes. Note this reduction runs in poly-time since the encoding of (M', x) as well as (an upper bound) for t can be computed in poly time. We have that $x \in L' \iff (M', x, t) \in L$.

(d) $L := \{(M, t) : \exists x\ M(x) \text{ halts within time } t\}$

NEXP-complete: $L \in \mathbf{NEXP}$: consider the verifier $V((M, t), x)$ that given an instance (M, t) and x runs $M(x)$ for at most t steps and accepts iff M accepts. Algorithm V runs in time exponential in the length of input (M, t) since $t \leq 2^{|(M,x,t)|}$. Every language $L' = L(V') \in \mathbf{NEXP}$ reduces to L, where $V'(z, x)$ is a verifier for L' which runs in exponential time in $|z|$. Map an instance z for L' to instance $(V'(z, \cdot), t)$ for L, where t is an upper bound for the time complexity of V' on z. Note the reduction is poly time since the encoding of $V'(z, \cdot)$ as well as an upper bound for t can be computed in poly time. We have that $z \in L' \iff \exists x\ V'(z, x) = 1 \iff \exists x\ (V'(z, x), t) \in L$.

(e) $L := \{(M, t) : \forall x\ M(x) \text{ halts within time } t\}$

coNEXP-complete: The solution is similar to previous item. The only difference is that in a verifier-based definition for **coNEXP**, we accept iff the verifier accepts for all x.

(f) $L := \{(M, 1^t) : \forall x\ M(x) \text{ halts within time } t\}$

coNP-complete: The solution is similar to previous item. The only difference is that the verifier now runs in polynomial time in $|z|$.

2 Relations between Classes

Assume **P** = **PSPACE**. Now consider each of the following statements and say whether it is true, false, or an open question. In case you say true or false, also give a short justification of your answer.

We use

$$\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} = \mathbf{NPSPACE} \subseteq \mathbf{EXP} \subseteq \mathbf{NEXP}.$$

and the facts that $\mathbf{NL} \subsetneq \mathbf{PSPACE}$ and $\mathbf{P} \subsetneq \mathbf{EXP}$.

(a) **P** = **NP**
Yes. We have that $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE}$. It follows that if **P** = **PSPACE** then **P** = **NP**.

(b) **NP** = **coNP**
Yes. By (a), we have **P** = **NP**. Since **P** = **coP**, it follows that **NP** = **coNP** as well.

(c) **P** = **L**
No. Since $\mathbf{L} \subseteq \mathbf{NL} \subsetneq \mathbf{PSPACE}$, it follows that if **P** = **PSPACE** then $\mathbf{L} \subsetneq \mathbf{P}$.

(d) **NP** = **EXP**
No. We know $\mathbf{P} \subsetneq \mathbf{EXP}$. By (a) **P** = **NP**. Hence $\mathbf{NP} \subsetneq \mathbf{EXP}$.

(e) **NP** = **L**
No. We have that **P** = **NP** = **PSPACE**. Since we know $\mathbf{L} \subsetneq \mathbf{PSPACE}$, it follows that $\mathbf{L} \subsetneq \mathbf{NP}$.

(f) **NPSPACE** = **P**
Yes. Since **PSPACE** = **NPSPACE**, it follows that **NPSPACE** = **P**.

3 Kleene Star

The Kleene star of a language L is the language

$$L^\star := \{x_1 \cdots x_k : k \geq 0 \text{ and } x_1, \ldots, x_k \in L\} .$$

That is, $L^\star$ consists of strings formed by concatenating a finite number of elements of L.

(a) Show that **NP** is closed under Kleene star.

Proof. In order to prove that $L^\star \in \mathbf{NP}$, we have to build a verifier $V^\star(\cdot,\cdot)$ for $L^\star$ that runs in polynomial time $f_{L^\star}(n)$. To do that, we can ask the prover to provide us a list of k tuples of form (s_i, y_i) such that:

- s_i is in the range $1..|x|$ and $s_i < s_{i+1}$;
- y_i is the "proof" that $x[(s_{i-1}+1)..s_i] \in L$, where we assume $s_0 = 0$.
- $s_k = |x|$.

In this case, k can be any integer in the range $1..|x|$, where $|x|$ is equal to the size of the input string. In the case we have an empty string, we don't need any information from the prover and our verifier can just say YES. Thus, the following program can verify the output of the prover is correct and that the input $x \in L^\star$.

Program $V^\star(x, y')$
If $x = \epsilon$ **then** say YES
Check that y' has the form $[(s_1, y_1), \ldots, (s_k, y_k)]$ and
$k \in [0..|x|]$, $s_i \in [0..|x|]$, $s_i < s_{i+1}$, and $s_k = |x|$.
Check if $V(x[1..s_1], y_1) =$ YES and $V(x[s_1..s_2], y_2) =$ YES
and $\cdots$ and $V(x[s_{k-1}..s_k], y_k) =$ YES.
If checks are true **then** say YES **else** say NO.

Now that we have constructed the verifier, it remains to prove that it runs in polynomial time. If the input is the empty string, the verifier runs in a constant time. For any other inputs, we need at most to verify $|x|$ substrings, each of which takes polynomial time. Thus, the total running time $f_{L^\star}(n)$ is also polynomial. □

(b) Show that **P** is closed under Kleene star.

Proof. Let $L \in \mathbf{P}$. Then, there is a Turing Machine M that decides L in polynomial time $f_L(n)$. In order to prove that $L^\star \in \mathbf{P}$, we need to construct a Turing Machine $M^\star$ that decides $L^\star$ in polynomial time, say $f_{L^\star}(n)$. To do that, we will construct a matrix A of size $(|x|+1) \times (|x|+1)$, where $|x|$ is the size of the input string. For each element $A_{i,j}$, we have:

$$A_{i,j} = \begin{cases} 1 & \text{if } (i < j) \text{ and the substring } x[i..(j-1)] \in L; \\ 0 & \text{otherwise.} \end{cases}$$

We can see that the generated matrix A may be interpreted as the adjacency matrix of a graph with $|x|+1$ nodes (where $A_{i,j} = 0$ means that there are no edges between nodes i and j and $A_{i,j} = 1$ means that there is an edge between these nodes). Doing that, to find out if $x \in L^\star$ is equivalent to find a path from node 1 to node $|x|+1$ in the graph. For example, the fact that substrings $x[2..5]$ and $x[6..8] \in L$ is equivalent to have edges (2,6) and (6,9), respectively, in the generated graph. Nevertheless, the problem of finding a path in a graph can be solved using the reachability algorithm, which runs in time (n^2) for a graph with n nodes. Therefore, the following Turing Machine can decide $L^\star$:

Program $M^\star(x)$
  If $x = \epsilon$ **then** accept and halt
  Initialize matrix A with zeros
  For $i = 1$ to $|x|$
    For $j = i$ to $|x|$
      Run $M(x[i..j])$
      If it accepts **then**
        $A[i, j + 1] = 1$
  Run the Reachability algorithm to check if there is a path
    from node 1 to node $|x| + 1$.
  If there is a path **then**
    accept and halt
  else
    reject and halt

To analyze the runtime of this machine, we can see that the initialization takes time $\mathcal{O}(|x|^2)$. Besides that, we have $\mathcal{O}(|x|^2)$ operations due to both **For** commands, each of which takes at most $f_L(|x|)$ time. In addition, the runtime due to the reachability algorithm is $\mathcal{O}(|x|^2)$. Therefore, the total runtime for the machine $M^\star$ is a polynomial function of the size of the input. □

4 PCP Variants

Determine which of the following variants of PCP are decidable:

(a) PCP over a unary alphabet.

Proof. This problem is easily decidable. If some tile has the same number of 1's on the top and bottom, there is a trivial match, so accept. If all tiles have more 1's on the top than on the bottom, or vice-versa, there cannot be a match, so reject. Otherwise, let t_1 be a tile with $a_1 > 0$ more 1's on the top than on the bottom and let t_2 be a tile with $a_2 > 0$ more 1's on the bottom than on the top. Choosing a_2 of t_1 tiles and a_1 of t_2 tiles yields an equal number of 1's on both top and bottom and hence a match; so accept in this case. □

(b) PCP over a binary alphabet.

Proof. This problem is undecidable as we can use the binary alphabet to injectively encode PCP instances over arbitrary alphabets. Given an instance of PCP over alphabet $\Sigma = \{a_1, \ldots, a_n\}$, one can reduce it to an instance of PCP over the binary alphabet by applying the homomorphism h that maps the symbol a_i to string 10^i. The crucial property about this mapping is that, for $u, v \in \Sigma^\star$ we have that $u = v$ iff $h(u) = h(v)$ (i.e., h is *injective*). Let P be an instance of PCP over Σ. Take $h(P)$ to be the PCP instance over the binary alphabet, where any string w appearing in any tile of P is replaced by $h(w)$. Given that $u = v$ iff $h(u) = h(v)$, we can conclude that P is in PCP iff $h(P)$ is in binary PCP. Since PCP over Σ is undecidable, so is PCP over $\{0, 1\}$. □

★(c) PCP (over an arbitrary alphabet, as usual) but now the goal is determine whether or not there is an *infinite* sequence $i_1, i_2, \ldots$ such that $t_{i_1} t_{i_2} \cdots = b_{i_1} b_{i_2} \cdots$, where $\{(t_i, b_i)\}_{i=1}^n$ is a finite set of tiles.

Proof. The proof of undecidability of PCP relies on a reduction between an encoding of a TM as a PCP instance where the TM halts if and only if there exists a PCP solution. (The details are recalled below.)
In this reduction an infinite titling can exist because: (1) one can repeat the (finite) tiling infinitely; or (2) the machine does not halt and there is an infinite tiling. The only case in the reduction where an infinite tiling does not exist is for TM that block (i.e. the TM to a state that is not a final state and has no outgoing transition for the current symbol; which generally corresponds to rejecting states).
We can make a small modification to the reduction so that there is an infinite tiling iff the machine does not terminate: we consider only TM that never block but can only terminate by accepting or run infinitely and we insert a new symbol \$ appearing in the top tiles corresponding to final states (therefore blocking the tiling when the TM terminates).
The reduction to halting now is: run the infinite PCP decider; if it accepts (an infinite tiling exists), then reject (the machine loops); if it rejects (no infinite tiling exits), then accept (machine terminates).

The (standard) PCP reduction The idea was to "force" the tiles to follow a TM computation. The word shared between the top and bottom tiles will correspond to a computation history, i.e., it will be a sequence of words $u_j = \#w_1 \cdots w_k q_i w_{k+1} \cdots w_n$ with u_j representing the state of the machine after the j-th step. u_j represents a tape with contents $w_1 \cdots w_n$ and the current state q_i of the TM being on w_k.
In order to make sure that u_{j+1} is a valid transition from u_j, the bottom tiles will be one transition behind, i.e., when $b_0 \cdots b_i$ represents $u_0 \cdots u_j$ then $t_0 \cdots t_i$ represents $u_0 \cdots u_{j+1}$.
Our tiles are:

- copy the band state (i.e., $t = a, b = a$ for tape symbol a);
- copy the separation between two steps of computation ($t = b = \#$);
- for each transition $(q, x) \to (R, q'z)$ (read x in state q, write z move right and go to state q') we have $b = xqy$ that "reads" q and x and $t = zy'q$ for each tape symbol y.

- for each transition $(q, x) \to (R, q'z)$, we also have $b = xq\#$, $t = z\_q\#$ that extends the tape with the base tape symbol $\_$.
- for each transition $(q, x) \to (L, q'z)$ and each tape symbol a, we have $b = xqa$ that "reads" x in state q and $t = qza$ that prints z and moves left (the a ensures we are not at the end of the tape).
- for each transition $(q, x) \to (L, q'z)$, we have $b = xq\#$ that "reads" x in state q and $t = qz\#$ that prints z and moves left except when $z = \_$ (the base tape symbol) in which case the top tile is $t = q\#$. This transition makes sure that we only keep the "active" part of the tape (i.e. the part after the head of the TM composed of an infinite repetition of $\_$).
- finally, for each final accepting state q we have $b = q\#$, $t = \epsilon$ to terminate the tilling. This termination only works with a tape that is completely erased but we can enforce that the TM erases its tape before accepting.

This reduction works only if we add a first domino that is $b = u_0$ and $t = u_0\#u_1$. However nothing ensures that this is the first domino that will be used (and if not $t = a, b = a$ will be an infinite tiling). To ensure that our first domino is really the first domino used, we modify all tiles to insert $\star$ after all characters in the top tiles and $\star$ before all chars in the bottom tiles. Finally we add a $\star$ at the beginning of our first top tile and a $\star$ to our final dominos (the $(b = q\#, t = \epsilon)$ for q an accepting state). This way the first tile that can be used is our first domino. □

5 Randomization and Nondeterminism

A language $L \in \mathbf{BP} \cdot \mathbf{NP}$ if there exists a polynomial-time deterministic Turing machine M such that

$$x \in L \implies \Pr_{r \in \{0,1\}^{m(n)}} \left[\exists y \in \{0,1\}^{k(n)} \; M(x,y;r) = 1 \right] \geq 2/3$$
$$x \notin L \implies \Pr_{r \in \{0,1\}^{m(n)}} \left[\exists y \in \{0,1\}^{k(n)} \; M(x,y;r) = 1 \right] \leq 1/3$$

where $m(n), k(n) \leq \text{poly}(n)$. A language $L \in \mathbf{NP} \cdot \mathbf{BP}$ if there exists a polynomial-time deterministic Turing machine M such that

$$x \in L \implies \exists y \in \{0,1\}^{k(n)} \Pr_{r \in \{0,1\}^{m(n)}} [M(x,y;r) = 1] \geq 2/3$$
$$x \notin L \implies \forall y \in \{0,1\}^{k(n)} \Pr_{r \in \{0,1\}^{m(n)}} [M(x,y;r) = 1] \leq 1/3$$

where $m(n), k(n) \leq \text{poly}(n)$. Show that

$$\mathbf{NP} \cdot \mathbf{BP} \subseteq \mathbf{BP} \cdot \mathbf{NP} \; .$$

Idea of the proof. Let $L = L(M) \in \mathbf{NP} \cdot \mathbf{BP}$ with M satisfying the conditions of $\mathbf{BP} \cdot \mathbf{NP}$. We consider a machine $\overline{M}_p$ that takes (x, y) and $p(|x|)$ sets of random coins $(r_1, \ldots, r_{p(|x|)})$ for some polynomial p. Machine $\overline{M}_p$ accepts (x, y) when the *majority* of the runs over coins $(r_i)_{1..p(|x|)}$ are accepting. We will show that for sufficiently large p, machine $\overline{M}_p$ satisfies the conditions of $\mathbf{NP} \cdot \mathbf{BP}$.

A new definition for $\mathbf{NP} \cdot \mathbf{BP}$. When $L \in \mathbf{NP} \cdot \mathbf{BP}$ there exists a polynomial-time deterministic Turing machine $\overline{M}_p$ such that:

$$x \in L \implies \exists y \in \{0,1\}^{k(n)} \Pr_{r \in \{0,1\}^{m(n) \times p(n)}} \left[\overline{M}_p(x,y;r) = 1 \right] \geq 1 - (2/3)^{1+2k(n)}$$
$$x \notin L \implies \forall y \in \{0,1\}^{k(n)} \Pr_{r \in \{0,1\}^{m(n) \times p(n)}} \left[\overline{M}_p(x,y;r) = 1 \right] \leq (1/3)^{1+2k(n)}$$

This definition is indeed equivalent. The outputs $(b_i)_{i=1..p(|x|)}$ of the $p(|x|)$ runs of the machine are independent and identically distributed. When $x \in L$ we expect the mean of b_i to be around $2/3$. (Formally, by the Chernoff bound, the sample mean will deviate from $2/3$ with overwhelming probability; that is

$$\Pr_{r \in \{0,1\}^{m(n) \times p(n)}} \left[\left| \frac{1}{p(n)} \sum b_i - 2/3 \right| \geq \varepsilon \cdot (2/3) \right] \leq e^{-\frac{\varepsilon^2}{4} \cdot (2/3) \cdot p(n)} \; .$$

If we set $\varepsilon := 1/8$ so that $2/3 - \varepsilon > 1/2$, the majority will be the wrong answer with probability at most $e^{-p(n)/384}$. Hence we can set $p(n) := 384 \ln(3/2)(1 + 2k(n))$. The case $x \notin L$ is similar: we look at deviation from $1/3$, again setting $\varepsilon := 1/8$ so that $1/3 + \varepsilon < 1/2$.)

Proof that $\overline{M}_p$ validates the $\mathbf{BP} \cdot \mathbf{NP}$ condition when $x \in L$. By definition of $\mathbf{NP} \cdot \mathbf{BP}$ there exists y_x such that

$$\Pr_{r \in \{0,1\}^{m(n) \times p(n)}} [\overline{M}(x, y_x; r) = 1] \geq 1 - (2/3)^{1+2k(n)} \geq 2/3 \; .$$

That is, at least $2/3$ of the r are good for y_x. Then

$$\Pr_{r \in \{0,1\}^{m(n) \times p(n)}} [\exists y \in \{0,1\}^{k(n)} \; \overline{M}(x,y;r) = 1] \geq 2/3 \; ,$$

because $y_x \in \{0,1\}^{k(n)}$ and $2/3$ of r will be good for this y_x. (Note any $p(n) \geq 1$ works here.)

Proof that $\overline{M}_p$ validates the $\mathbf{BP} \cdot \mathbf{NP}$ condition when $x \notin L$. We have that for all $y \in \{0,1\}^{k(n)}$

$$\Pr_{r \in \{0,1\}^{m(n) \times p(n)}} \left[\overline{M}_p(x, y; r) = 1 \right] \leq (1/3)^{1+2k(n)} .$$

Therefore the number of $(y, r) \in \{0,1\}^{k(n)} \times \{0,1\}^{m(n) \times p(n)}$ with $\overline{M}_p(x, y; r) = 1$ is bounded by

$$\frac{2^{k(n)+m(n)\times p(n)}}{3^{2k(n)+1}} = \frac{2^{k(n)}}{3^{k(n)}} \times \frac{2^{m(n)\times p(n)}}{3^{k(n)+1}} \leq \frac{2^{m(n)\times(n)}}{3^{k(n)+1}} \leq \frac{2^{m(n)\times p(n)}}{2^{k(n)}} \times \frac{1}{3}.$$

Since there are at most $2^{p(n)\times m(n)}$ values for r and at most $2^{k(n)}$ values for y, by pigeonhole there can exist at most $2^{m(n)\times p(n)}/3$ distinct r for which there is a y such that $\overline{M}_p(x, y; r) = 1$; that is,

$$\Pr_{r \in \{0,1\}^{m(n) \times p(n)}} \left[\exists y \in \{0,1\}^{k(n)} \ \overline{M}_p(x, y; r) = 1 \right] \leq 1/3 .$$

In any case $L \in \mathbf{BP} \cdot \mathbf{NP}$ is recognized by $\overline{M}_p$ and thus $\mathbf{NP} \cdot \mathbf{BP} \subseteq \mathbf{BP} \cdot \mathbf{NP}$. □

6 Separation via Closure

For a language $L \subseteq \Sigma^\star$ and a function $f : \Sigma^\star \longrightarrow \Sigma^\star$, let

$$L_f := \{x \in \Sigma^\star : f(x) \in L\} \ .$$

We say that a complexity class is *closed under polynomial composition* if for any L in the class and any polynomial-time computable function f it is the case that language L_f is also in that class.

For a language A, let $A^\sharp := \{x0^{|x|^2-|x|} : x \in A\}$ be the language consisting of elements x of A appended with $|x|^2 - |x|$ zeros.

(a) Show that **NP** is closed under composition.

Proof. If $L \in \mathbf{NP}$ then there is a verifier $V(\cdot,\cdot)$ for L that runs in polynomial time f_L. To prove that $L_f \in \mathbf{NP}$, we have to build a verifier $V_f(\cdot,\cdot)$ that runs in polynomial time $f_{L_f}(n)$. To do that, we ask the prover to provide us a "proof", say y', that $f(x) \in L$. Then, the following program can verify if the output of the prover is correct.

Program $V_f(x, y')$
  Compute $x' = f(x)$
  Check if $V(x', y') =$ YES
  If check is true **then** say YES **else** say NO.

As f is polynomial time computable function, then there is a Turing Machine M and polynomial function $g(\cdot)$ such that: (1) $M(x) = f(x)$, and (2) $M(x)$ halts in $g(|x|)$ steps. Using the fact that $\mathbf{TIME}(f(n)) \subset \mathbf{SPACE}(f(n))$, then we can say that $|f(x)| \leq g(|x|)$. Therefore, the input of the verifier $V(\cdot,\cdot)$ in the program $V_f(\cdot,\cdot)$ has a size that is a polynomial function of the original input x. As the verifier $V(\cdot,\cdot)$ runs in a polynomial time, then $V_f(\cdot,\cdot)$ also runs in a polynomial time. □

(b) Show that if $A \in \mathbf{SPACE}(n^2)$ then $A^\sharp \in \mathbf{SPACE}(n)$.

Proof. By assumption, there is an algorithm M_A and a constant c such that, on any input x, algorithm M_A outputs if $x \in A$ and 0 otherwise. Moreover, M_A uses at most $c|x|^2$ space on its work tapes. We now design an algorithm $M_{A^\sharp}$ to decide $A^\sharp$ in linear space.

Algorithm $M_{A^\sharp}$ takes an input y. It first computes $m = \sqrt{|y|}$, and rejects if this is not an integer This takes poly(log($|y|$)) space. Next it checks that the last $m^2 - m$ bits of y are zeros and rejects if not. Now it lets x be the first m bits of y and outputs $M_A(x)$. The running time of $M_{A^\sharp}$ is $c|x|^2 + \text{poly}(\log(|y|)) + \mathcal{O}(|y|)$. But this is $\mathcal{O}(|y|)$ because $|x|^2 = \mathcal{O}(|y|)$. So $A^\sharp \in \mathbf{SPACE}(n)$. □

(c) Show that $\mathbf{SPACE}(n)$ is *not* closed under composition.

Proof. To prove that $\mathbf{SPACE}(n)$ is *not* closed under composition, it's sufficient to show that there is an example in which this is not true. To do that, we can make use of the example of the previous item of this problem, which we rewrite as $A = \{x \in \Sigma^\star : f(x) \in A^\sharp\}$. In this case, $f(x)$ represents the function that appends zeros to the original input string. As we have seen, $A^\sharp \in \mathbf{SPACE}(n)$ and $A \in \mathbf{SPACE}(n^2)$.

Using the Space Hierarchy Theorem, we can see that $\mathbf{SPACE}(n) \neq \mathbf{SPACE}(n \log n) \subseteq \mathbf{SPACE}(n^2)$. Therefore, $\mathbf{SPACE}(n)$ is *not* closed under composition. □

(d) Conclude that $\mathbf{NP} \neq \mathbf{SPACE}(n)$.

Proof. In order for the two classes be equal, they should have the same properties. As we proved in the first item of this problem that **NP** is closed under composition. However, as we also have proved in the previous item of this problem, $\mathbf{SPACE}(n)$ is *not* closed under composition. Therefore, we conclude that both classes are different. □

7 NP-Completeness

CLIQUE-COVER is the following problem:

INPUT: A graph $G = (V, E)$ and a positive integer $K \leq |V|$;
QUESTION: Can the vertices of G be partitioned into $k \leq K$ disjoint sets $V_1, \ldots, V_k$ such that for every $i = 1, \ldots, k$ the subgraph induced by V_i is a complete graph?

Show that CLIQUE-COVER is **NP**-complete.

Proof. To show that CLIQUE COVER is **NP**-complete, we need to prove two things:

- CLIQUE COVER is in **NP**;
- $A \leq_p$ CLIQUE COVER for some A which is **NP**-complete.

In order to prove that CLIQUE COVER is in **NP**, we have to build a verifier $V^\star(\cdot, \cdot)$ for CLIQUE COVER that runs in polynomial time. To do so, we can ask the prover to provide us a partition of G into $k \leq K$ disjoint sets $V_1, \ldots, V_k$ such that for every $i = 1, \ldots, k$, the subgraph induced by V_i is a complete graph. Since the number of operations is polynomial and each operation runs in polynomial time, the total running time is also polynomial.

For the second part, we will use COLORING for the reduction. Here we consider COLORING as equivalent to k-COLORING, differing only in the aspect that k is an explicit input to the problem. More precisely, we have to provide a function f that reduces COLORING to CLIQUE COVER in polynomial time. In particular, this function has to map the original graph G and the input k to a new graph G' and an integer k' such that $(G; k) \in$ COLORING if and only if $(G'; k') \in$ CLIQUE COVER. This can be written as follows:

$$f : (G(V, E); k) \to (G'(V', E'); k')$$

The idea for constructing the new graph G' is to create a "reversed" graph of G, i.e., $V' = V$ and $\{u, v\} \in E'$ if and only if $\{u, v\} \notin E$. We map k into k'.

$(G; k) \in$ **COLORING** $\implies$ $(G'; k') \in$ **CLIQUE COVER**. If $(G; k) \in$ COLORING, then there exists a map $c : V \to \{1, \ldots, k\}$ such that $c(u) \neq c(v)$ for all edges $\{u, v\} \in E$. Let V_i be the subset of nodes u such that $c(u) = i$. As we know that two nodes in G with the same color are not connected to each other, it implies that all nodes in V_i are not connected to each other in G. So, when we consider the "reverse" graph G', all nodes in V_i must be fully connected. As we have at most k different colors, then so is the number of subsets V_i. As one node cannot have more than one color, then all subsets V_i are disjoint. In addition, as all nodes are mapped to at least one color, then the union of all subsets V_i is equal to V. Therefore, as k is directly mapped to k', we can conclude that $(G'; k') \in$ CLIQUE COVER.

$(G'; k') \in$ **CLIQUE COVER** $\implies$ $(G; k) \in$ **COLORING**. If $(G'; k') \in$ CLIQUE COVER, then there exists a partition of the vertices of G' into $k' \leq |V|$ disjoint sets $V_1, \ldots, V_k$ such that for every $i = 1, \ldots, k'$ the subgraph induced by V_i is a complete graph. Let c be a mapping such that all nodes in V_i are mapped to a color i in the original graph G. As we know that all the nodes in V_i are fully connected to each other in G', it implies that they are not connected to each other in G. As we know that all the subsets V_i are disjoint, then each node v is mapped to only one color in G. Also, as we know that we have at most k' subsets and that their union is equal to V, then all nodes are mapped to one color and we have at most k' different colors. Therefore, as k is mapped directly to k', we can conclude that $G \in$ COLORING. □

8 Bonus Question: Sometimes Space = Time

Show that there exists a function $T(n) \geq n$ such that $\mathbf{DTIME}(T(n)) = \mathbf{DSPACE}(T(n))$.

Proof. We assume that the underlying languages are binary. Let

$$T(n) := \text{maximum number of steps that an } n^2 \text{ state TM can make on the empty tape .}$$

This is, $T(n)$ is (closely related to) the busy-beaver function. Clearly, $\mathbf{DTIME}(T(n)) \subseteq \mathbf{DSPACE}(T(n))$. For the converse inclusion, let $L = L(M) \in \mathbf{DSPACE}(T(n))$. For $x \in L$, define $M[x]$ to be the TM that has x hardwired in and then runs $M(x)$. Machine $M[x]$ has at most $c \cdot |x| + |k|$ states (for some constant c). For sufficiently large $|x|$, machine $M[x]$ has at most $|x|^2$ states and thus runs in time at most $T(|x|)$. Hence $L \in \mathbf{DTIME}(T(n))$.

□