CS 505 Spring 2025 — Homework 1 (Sample Solutions)

Alexander R. Block

Notes

These are sample solutions and do not constitute all possible answers for the homework. These solutions are meant to guide the students and give you an idea of the level of detail that is expected of you for future homework, the midterm exam, and the final project.

1 Problem 1 (Turing Machine Equivalences) (20 Points)

1.1 Part 1 (10 Points)

A *single-tape* Turing machine is a TM with a single read/write tape that is infinite in one direction. This tape is used as the input, output, and work tape. Formally prove that for every function $f\colon \{0,1\}^* \to \{0,1\}^*$ and time constructible $T\colon \mathbb{N} \to \mathbb{N}$, if f is computable in time T by a k-tape Turing machine (for $k \geq 3$), then it is computable in time $O(k \cdot T^2)$ on a single-tape Turing machine.

Note. You are expected to give the full formal description of the single-tape Turing machine. This includes specifying its set of states, and its transition function $\delta\colon Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$. You are also expected to give the run-time analysis of your single-tape Turing machine.

Proof of Problem 1 Part 1. Let f be a function that is computable in time T by a k-tape Turing machine, for $k \geq 3$ and time constructible T. Let M_f be the corresponding k-tape Turing machine with input alphabet Σ, tape alphabet Γ, set of states Q, and transition function δ. We will construct a single-tape Turing machine that will compute f in time $O(k \cdot T^2)$. We denote this machine by $\widehat{M}_f$.

For machine $\widehat{M}_f$, we let $\widehat{Q}$, $\widehat{\Sigma}$, $\widehat{\Gamma}$, and $\widehat{\delta}$ denote the set of states, input alphabet, tape alphabet, and transition function of $\widehat{M}_f$. First, we define $\widehat{\Sigma} := \Sigma$. Second, we define $\widehat{\Gamma}$ as follows. For every $\gamma \in \Gamma$, we add γ and a special $\widehat{\gamma}$ to $\widehat{\Gamma}$. We'll use these special symbols with the $\widehat{\cdot}$ to track the head positions of the k-tape Turing machine. We will also add $2k$ new symbols $\triangleright_i, \widehat{\triangleright}_i$ for each $i \in [k]$, which we will use to track the start of tape i.

Before we define $\widehat{Q}$ and $\widehat{\delta}$, we outline the high-level idea behind the single-tape Turing machine's simulation. By definition, $\widehat{M}_f(x)$ will begin its computation with its single tape initialized as $(\triangleright, x, \square, \dots)$, with the tape-head at the start position. Suppose $|x| = n$. For ease of notation, let $A = (\triangleright, x, \square, \dots)$ denote $\widehat{M}_f$'s single tape. At a high-level, $\widehat{M}_f(x)$ will do the following.

1. We'll let $A[1]$ be the first $\square$ right after the input x (at position $n+2$ of the tape if we originally started indexing by 0), and $A[-n] = \triangleright$ be the start of the tape.

2. Starting with position $A[1]$, we'll stagger the $(k-2)$ work tapes and the output tape we need to simulate. That is, we'll let $A[1]$, $A[1+k]$, $A[1+2k]$, $\dots$, be the indices of the second (i.e., first work tape) tape of M_f. Generalizing, we'll let $A[i]$, $A[i+k]$, $A[i+2k]$, $\dots$, be the indices of tape i of M_f, for $2 \leq i \leq k$.

3. First, $\widehat{M}_f(x)$ will put the start symbol $\widehat{\triangleright}_i$ at position $A[i+1]$ for $i \in [k]$. This symbolizes initializing the start symbol and the initial tape head positions of M_f.

4. Now, we can simulate $M_f(x)$. During any point of the simulation, we must read the k symbols under each of the k tape heads of M_f. Our single tape machine will linearly scan the tape and collect the symbols under the head, storing them in the current state. Once collected, the single-tape machine can execute the transition function of M_f, then scan the entire tape again.

Since k is fixed, we will augment our set of states as follows. We set $\widehat{Q} = Q \times Q^* \times \Gamma^k \times \{L, R, S, \square\}^k \times [k] \cup \{0\}$, where $Q^* = \{\widehat{q}_{start}, q_{reset-head}, q_{end-input}, q_{init-tapes}, q_{start-sim}, \{q_{exec,j}, q_{exec,j,s}, q_{exec,j,m}\}_{j\in[k]}\}$.

We now define the transition function $\widehat{\delta}$ as follows.

- $\widehat{\delta}((q_{start}, \widehat{q}_{start}, \square^k, \square^k, 0), \triangleright)$:
  - Output $((q_{start}, q_{end-input}, \square^k, \square^k, 0), \widehat{\triangleright}_1, R)$.
- $\widehat{\delta}((q_{start}, q_{end-input}, (\square^k), \square^k, 0), \gamma)$:
  - If $\gamma \neq \square$, then output $((q_{start}, q_{end-input}, \square^k), \square^k, 0), \gamma, R)$.
  - If $\gamma = \square$, then output $((q_{start}, q_{init-tapes}, \square^k, \square^k, 1), \widehat{\triangleright}_2, R)$
- $\widehat{\delta}((q_{start}, q_{init-tapes}, \square^k, \square^k, j), \gamma)$:
  - If $j < k$, output $((q_{start}, q_{init-tapes}, \square^k, \square^k, j+1), \widehat{\triangleright}_{j+2}, R)$.
  - If $j = k$, output $((q_{start}, q_{reset-head}, \triangleright^k, \square^k, 0), \widehat{\triangleright}_k, L)$.
- $\widehat{\delta}((q, q_{reset-head}, (\gamma_j)_{j\in[k]}, \square^k, i), \gamma)$:
  - If $\gamma \neq \widehat{\triangleright}_{i+1}$, then output $((q, q_{reset-head}, (\gamma_j)_{j\in[k]}, \square^k, i), \gamma, L)$.
  - Else if $i = 0$, then output $((q, q_{start-sim}, (\gamma_j)_{j\in[k]}, \square^k, 0), \gamma, S)$.
  - Else output $((q_{start}, q_{exec,i,s}, (\gamma_j)_{j\in[k]}, \square^k, 0), \widehat{\triangleright}_{i+1}, S)$.
- $\widehat{\delta}((q, q_{start-sim}, (\gamma_j)_{j\in[k]}, \square^k, 0), \widehat{\triangleright}_1)$:
  - If $q = q_{halt}$, then halt.
  - Obtain $(q', (\gamma'_2, \ldots, \gamma'_k), D_1, \ldots, D_k) \leftarrow \delta(q, \gamma_1, \ldots, \gamma_k)$.
  - Output $((q', q_{exec,1,s}, (\gamma'_1, \ldots, \gamma'_k)), (D_1, \ldots, D_k), 0, S)$.
- $\widehat{\delta}((q, q_{exec,j,s}, (\beta_1, \ldots, \beta_{j-1}, \gamma_j, \ldots, \gamma_k), (\square^{j-1}, D_j, \ldots, D_k), 0), \gamma)$:
  - If $\gamma \neq \triangleright_j$ or $\gamma \neq \widehat{\triangleright}_j$, then output $((q, q_{exec,j,s}, (\beta_1, \ldots, \beta_{j-1}, \gamma_j, \ldots, \gamma_k), (\square^{j-1}, D_j, \ldots, D_k), 0), \gamma, R)$.
  - Else output $((q, q_{exec,j}, (\beta_1, \ldots, \beta_{j-1}, \gamma_j, \ldots, \gamma_k), (\square^{j-1}, D_j, \ldots, D_k), 0), \gamma, S)$.
- $\widehat{\delta}((q, q_{exec,j}, (\beta_1, \ldots, \beta_{j-1}, \gamma_j, \ldots, \gamma_k), (\square^{j-1}, D_j, \ldots, D_k), i), \gamma)$:
  - If $q_{exec,1}$ and $\gamma \neq \widehat{\alpha}$, then output $(q, q_{exec,1}, (\gamma_1, \ldots, \gamma_k), (D_1, \ldots, D_k), 0), \gamma, R)$.
  - If $i = 0$ and $\gamma = \widehat{\alpha}$, then output $((q, q_{exec,j,m}, (\beta_1, \ldots, \beta_{j-1}, \gamma_j, \ldots, \gamma_k), (\square^j, D_{j+1}, \ldots, D_k), 0), \gamma_j, D_j)$.
  - If $1 \leq i < k$, output $((q, q_{exec,j}, (\beta_1, \ldots, \beta_{j-1}, \gamma_j, \ldots, \gamma_k), (\square^{j-1}, D_j, \ldots, D_k), i+1), \gamma, R)$.
  - If $i = k$, then output $((q, q_{exec,j}, (\beta_1, \ldots, \beta_{j-1}, \gamma_j, \ldots, \gamma_k), (\square^{j-1}, D_j, \ldots, D_k), 0), \gamma, S)$.
- $\widehat{\delta}((q, q_{exec,j,m}, (\beta_1, \ldots, \beta_{j-1}, \gamma_j, \ldots, \gamma_k), (\square^j, D_{j+1}, \ldots, D_k), 0), \beta)$:
  - If $q_{exec,k,m}$, then output $((q, q_{reset-head}, (\beta_1, \ldots, \beta_{k-1}, \gamma_k), (\square^k), 0, \widehat{\beta}, L)$.
  - Else output $((q, q_{reset-head}, (\beta_1, \ldots, \beta_j, \gamma_{j+1}, \ldots, \gamma_k), (\square^j, D_{j+1}, \ldots, D_k), j+1, \widehat{\beta}, L)$.

At a high-level, we are encoding the necessary information in the set of states of $\widehat{M_f}$, as well as utilizing a counter in the states of $\widehat{M_f}$. Essentially, we are tracking M_f's current state, $\widehat{M_f}$'s current state, the current contents under the heads of M_f's tapes, the current directions we need to execute for each of the k tapes, and a counter i. $\widehat{M_f}$ on input x proceeds as follows.

- Moves to the end of its input.
- Starts marking the start of the $k-1$ staggered tape heads of M_f's non-work tape.
- Resets the tape head to the start of its single-tape.
- Begins simulation of M_f.
- Simulates one tape at a time. First, it seeks out the start of the current tape. Once found, it scans the tape in steps of size k until it finds the current tape head. It then executes the write and move of the current tape head according to what is stored from M_f's transition function. It then reads the symbol under the current tape head (after moving) and stores it in its memory. It then resets the tape head, and finds the next tape.
- Once all k transitions have been executed, the tape head is reset, and we begin the simulation again.
- If the simulation ever starts with q_{halt}, then the machine halts.

Since the original machine computes $M_f(x)$ in time $O(T(|x|))$, it is clear that our machine takes $O(k \cdot T(|x|)^2)$, since we reset the position of the single tape head after correctly performing any of the transitions of tape-head i. In the worst-case, this is $O(k \cdot T(|x|))$ steps, and it is done at most $O(T(|x|))$ times, yielding the upper bound. □

1.2 Part 2 (10 Points)

Define a *RAM Turing machine* (i.e., random access memory Turing machine) to be a Turing machine that has *random access memory*. We formalize this as follows. A RAM Turing machine has 4 tapes: an input tape, an output tape, and 2 work tapes. One work tape is the *address tape*, and the other is the *memory tape*. Let $A \colon \mathbb{N} \to \mathbb{N}$ be the array denoting the memory tape. The RAM Turing machine has random access to the memory tape; that is, in a single step of the computation, the RAM Turing machine can move the memory tape head to any location i for $i \in \mathbb{N}$ (i.e., move the tape head to $A[i]$).

The RAM Turing machine has two additional tape alphabet symbols: `R` and `W`, along with an additional state q_{access}. Whenever the machine enters the state q_{access}, it reads from the address tape. If the read contents have the form $\langle i \rangle_2 \texttt{R}$, then the machine reads symbol $A[i]$ and writes it to the cell directly to the right of the `R` symbol. If the read contents have the form $\langle i \rangle_2 \texttt{W} \gamma$, then the Turing machine writes γ to $A[i]$. Here, $\langle i \rangle_2$ denotes the binary representation of i.

Show that any function $f \colon \{0,1\}^* \to \{0,1\}$ that is computable on a time T RAM Turing machine (for time constructible T), then $f \in \mathbf{DTIME}(T^2)$.

Note. You do not need to give the super formal Turing machine description for this problem. Also remember that the number of tapes in the Turing machine cannot grow with the input length (it must be a constant).

Proof of Problem 1 Part 2. Let M_f be the RAM Turing machine which computes f in time T. We construct a k-tape Turing machine N_f which computes f in $O(T^2)$ time. Our Turing machine N_f will have 4 tapes, just as the machine M_f has. It will have an input tape, an output tape, and 2 work tapes. N_f will have one work tape that acts as the address tape, and one that acts as the memory tape.

Now, $N_f(x)$ will operate nearly identically to $M_f(x)$. In particular, it will copy exactly what $M_f(x)$ does on its input, output, and address tapes. Now, the only issue is the memory tape. $M_f(x)$ can access location $A[i]$ in constant time on its memory tape A. However, $N_f(x)$ must instead linearly scan its memory tape to reach position i.

Since $M_f(x)$ runs in time $O(T(|x|))$, and since $M_f(x)$ must linearly scan its address tape to read the address i, it must hold that the maximum number of positions of A that $M_f(x)$ accesses is $O(T/\log(T))$ since reading the address takes time at most $O(\log(T))$. Now, on machine $N_f(x)$, in the worst case, $M_f(x)$ can access position $A[T]$, followed by position $A[1]$, followed by $A[T]$, and so on. So, in the worst case, moving from position $A[i]$ to position $A[j]$ on the memory tape takes $O(T)$ time. Thus, in the worst case, $N_f(x)$ takes time $O(T(|x|)^2)$ to compute $f(x)$. □

2 Problem 2 (Undecidability) (15 Points)

2.1 Part 1 (5 Points)

In class, we showed that the halting problem, specified by the language

$$L_H = \{(\alpha, x) \colon M_\alpha(x) \text{ halts in a finite number of steps}\}$$

was undecidable via a reduction to another language. Prove that L_H is undecidable directly. I.e., do not reduce undecidability of L_H to another language; show the proof (via contradiction) directly.

Proof of Problem 2 Part 1. Suppose towards contradiction that L_H is decidable. That means there exists a Turing machine M_H which, on input (α, x), outputs 1 if and only if $(\alpha, x) \in L_H$, which happens if and only if $M_\alpha(x)$ halts in a finite number of steps.

Define a new Turing machine D which takes as input (α) and does the following.

1. Let $b = M_H(\alpha, \alpha)$.
2. If $b = 1$, then loop forever.
3. If $b = 0$, then output 1.

Now consider running $D(\langle D \rangle)$ for any string x. Since M_H is a decider, it always halts. If $M_H(\langle D \rangle, \langle D \rangle) = 1$, it implies that $D(\langle D \rangle)$ halts. However, in this case, D loops forever. If $M_H(\langle D \rangle, \langle D \rangle) = 0$, then it implies $D(\langle D \rangle)$ loops forever. But in this case, D outputs 1. Thus, we have derived a contradiction, and L_H is undecidable. □

2.2 Part 2 (5 Points)

A function $f \colon \{0,1\}^* \to \{0,1\}^*$ is a *partial function* if it is not necessarily defined on all inputs. We say that a Turing machine M computes partial function f if for every x that $f(x)$ is defined on, we have $M(x) = f(x)$, and if x is not defined on f, then $M(x)$ loops infinitely.

Let S be a set of partial functions. Define f_S to be the function defined as follows.

- $f_S(\alpha) = 1$ if and only if M_α computes some function $g \in S$.
- $f_S(\alpha) = 0$ if and only if M_α does not compute any function $g \in S$.

(Rice's Theorem): Prove that if $S \neq \emptyset$ and if S is not equal to the set of all partial functions computable by some Turing machine, then f_S is uncomputable. Equivalently, the language $L_S = \{\alpha \colon f_S(\alpha) = 1\}$ is undecidable.

Hint: can you reduce this problem to the Halting problem?

Proof of Problem Part 2. Suppose that L_S is decidable and let D_S be its decider. This implies that on input α, $D_S(\alpha) = 1$ if and only if there exists some function $g \in S$ such that $M_\alpha(x) = g(x)$ for all x where $g(x)$ is defined. First, fix some β such that $D_S(\beta) = 1$. Next, notice that for any machine M_I such that M_I loops forever, we have that $D_S(\langle M_I \rangle) = 0$.

We now construct a decider for the Halting problem L_H. Let M_H denote this machine. We define M_H as follows.

- $M_H(\alpha, x)$:
  1. Construct machine $\widehat{M}$ such that for any input y:
     - Run $M_\alpha(x)$ (note that M_α and x are hard-coded into $\widehat{M}$).
     - Return $M_\beta(y)$.
  2. Return $D_S(\widehat{M})$.

We now argue that M_H is a decider. First, constructing the machine $\widehat{M}$ takes a finite amount of time given the input (α, x). Next, because D_S is a decider, it halts on all inputs. Therefore, M_H halts on all inputs. Now, we have the following equivalences.

$$M_H(\alpha, x) = 1 \iff D_S(\widehat{M}) = 1 \iff \widehat{M} \in L_S \iff M_\alpha(x) \text{ halts.}$$

Here, the last equivalence follows since whenever $M_\alpha(x)$ halts, it outputs $M_\beta(y)$. By our choice of β, we know that $\beta \in L_S$, so $\widehat{M} = M_\beta$ whenever $M_\alpha(x)$ halts.

Next, we have

$$M_H(\alpha, x) = 0 \iff D_S(\widehat{M}) = 0 \iff M_\alpha(x) \text{ never halts.}$$

The last equivalence follows since if $M_\alpha(x)$ halts, then $\widehat{M} = M_\beta$ and we have assumed $\beta \in L_S$, which would imply $D_S(\widehat{M}) = 1$. But since $D_S(\widehat{M}) = 0$, this is not the case, and this can only happen if $M_\alpha(x)$ does not halt. In particular, $\widehat{M}$ does not halt.

Thus, we have constructed a decider for L_H, which is a contradiction since L_H is undecidable. □

2.3 Part 3 (5 Points)

Given a Turing machine M, let $\mathcal{L}(M) \subset \{0,1\}^*$ denote the language that M recognizes (i.e., for all $x \in \mathcal{L}(M)$, $M(x) = 1$). Define a new language

$$L_{\text{ETM}} = \{\alpha \colon \mathcal{L}(M_\alpha) = \emptyset\}.$$

Prove that L_{ETM} is undecidable.

Proof of Problem Part 3. Suppose by way of contradiction that L_{ETM} is decidable. Let D_{ETM} be the decider for L_{ETM}. We'll now build a new Turing machine M_H which decides the Halting problem L_H.

M_H takes as input (α, x) and must decide if $M_\alpha(x)$ halts in a finite number of steps. Our machine M_H on input (α, x) does the following.

1. Define Turing machine $T(y)$ which does the following:
   - Run $M_\alpha(x)$ (note: these are hard-coded into T).
   - Return 1.
2. Return $1 - D_{\text{ETM}}(\langle T \rangle)$.

We argue that M_H decides the language L_H. On input (α, x):

$$M_H(\alpha, x) = 1 \iff D_{\text{ETM}}(\langle T \rangle) = 0 \iff \mathcal{L}(T) \neq \emptyset \iff \exists y \text{ s.t. } T(y) = 1 \iff M_\alpha(x) \text{ halts.}$$
$$M_H(\alpha, x) = 0 \iff D_{\text{ETM}}(\langle T \rangle) = 1 \iff \mathcal{L}(T) = \emptyset.$$

Notice that for any $y \in \{0,1\}^*$, $T(y)$ accepts if and only if $M_\alpha(x)$ halts. So if $\mathcal{L}(T) = \emptyset$, it must be the case that $M_\alpha(x)$ never halts. Now, since we have assumed that D_{ETM} is a decider, it halts on all possible inputs. This implies that M_H also halts on all possible inputs (note that constructing the machine T takes a finite amount of work). Thus, M_H is a decider for L_H. This is a contradiction since L_H is undecidable. Thus, L_{ETM} must be undecidable. □

3 Problem 3 (P and NP) (25 Points)

3.1 Part 1 (10 Points)

Let $G = (V, E)$ be an undirected graph with vertex set V and edge set $E \subseteq V \times V$. A *path of length* n in G is an ordered tuple of $n+1$ vertices $(v_1, \ldots, v_{n+1})$ such that $(v_i, v_{i+1}) \in E$ for all $i \in [n]$. We say that a path is *simple* if every vertex in the path is unique (i.e., $v_i \neq v_j$ for all $i \neq j$). Define two languages

$$\text{SPATH} = \{(G, u, v, k) \colon G \text{ contains a simple path from } a \text{ to } b \text{ of length at most } k\};$$

and

$$\text{LPATH} = \{(G, u, v, k) \colon G \text{ contains a simple path from } a \text{ to } b \text{ of length at least } k\};$$

1. Show that SPATH $\in$ **P**.
2. Show that LPATH $\in$ **NP**.

Proof of Problem 3 Part 1. First, we show that SPATH $\in$ **P**. Two possible approaches to deciding SPATH are the following.

1. Run a depth-bounded breadth-first-search on the graph G, starting at vertex u, and never exploring a vertex that has already been visited. Only run the breadth-first-search up to depth k. If v is encountered before all vertices $\leq k$ steps from u have been visited, output accept. Otherwise, output reject. In the worst case, this algorithm takes $O(|V| + |E|)$, which is polynomial in the size of the input graph G.
2. Run Dijkstra's algorithm starting at vertex u. Then check if the distance between u and v is at most k. Accept if this is true, reject if this is not true. In the worst case, this algorithm takes $O(|V|^2)$ time.

The above algorithms are given in terms of a standard program, which is roughly equivalent to a RAM Turing machine. Since RAM Turing machines are poly-time equivalent to standard Turing machines, all the above algorithms still run in time that is polynomial in the length of the input.

Now, for showing LPATH $\in$ **NP**, we give a non-deterministic Turing machine which decides LPATH. Let D denote this NTM. On input (G, u, v, k), the NTM D does the following.

- Guess a path $P = (u, s_1, s_2, \ldots, s_\ell, v)$ such that $\ell \geq k$ and $s_i \neq s_j$ for all distinct i, j.
- If P is a valid path, output accept. Else output reject.

Guessing the path P above takes at most $O(|V|)$ time. Then, checking that P is a valid path takes at most $O(|V|^2)$ time (since you need to scan the set of edges to check if it is a valid edge). This is clearly polynomial time in the length of the input, and there exists a non-deterministic choice of the path P if and only if there is a simple path from u to v of length at least k in the graph G. Thus, D decides LPATH, and so LPATH $\in$ **NP**. □

3.2 Part 2 (5 Points)

Define a function pad$\colon \{0,1\}^* \times \mathbb{N} \to \{0,1\}^* \times \{\#\}^*$ as follows. For $x \in \{0,1\}^*$ and $k \in \mathbb{N}$, $\text{pad}(x, k) = (x, \#^j)$, where $j = \max\{0, (k - |x|)\}$ and $\#^j$ denotes writing the symbol $\#$ j times (e.g., $\#^3 = \#\#\#$).

Let $A \subseteq \{0,1\}^*$ be any language and let $T\colon \mathbb{N} \to \mathbb{N}$ be a function. Let $\mathbf{pad}(A, T)$ denote the following language:

$$\mathbf{pad}(A, T) = \{\text{pad}(x, T(|x|)) \mid x \in A\}.$$

Prove that if $A \in \mathbf{DTIME}(n^6)$ then $\mathbf{pad}(A, n^2) \in \mathbf{DTIME}(n^3)$.

Proof of Problem 3 Part 3. Let $A \in \mathbf{DTIME}(n^6)$. Let M_A be the decider for the language A which runs in time $O(n^6)$ on any input $x \in \{0,1\}^n$. We build a decider D for the language $\mathbf{pad}(A, n^2)$.

- D takes as input a string $w \in \{0,1\}^*$ and does the following.
  1. Parse w as $(x, \#^j)$ for some j. That is, scan the input w right until encountering the first $\#$ symbol, then count the number of $\#$ symbols, then stop when D hits a $\square$.
  2. Compute $i = |x|$.
  3. Check if $j = i^2 - i$; reject if equality does not hold.
  4. Output $M_A(x)$.

We now argue that D is a decider for $\mathbf{pad}(A, n^2)$. First, by definition of $\text{pad}(x, n^2)$, the input string w must have the form $(x, \#^j)$, where $j = \max 0, |x|^2 - |x|$. This is exactly what step (3) of $D(w)$ checks and computes. If w is not in the correct form, then $D(w)$ rejects. Now, if w is in the correct form, we have that $(x, \#^{|x|^2-|x|}) \in \mathbf{pad}(A, n^2)$ if and only if $x \in A$. Running the machine $M_A(x)$ exactly checks this. Note that since M_A is a decider for A, this Turing machine halts on all inputs and accepts if and only if $x \in A$. So D is a decider for $\mathbf{pad}(A, n^2)$.

Now, we argue the time complexity of D. Let $n = |w|$.

- Step (1) takes $O(n)$ time.
- Step (2) takes $O(i) = O(n)$ time.
- Step (3) takes $O(i^2) = O(n^2)$ time since n^2 is a time constructible function.
- Step (4) takes $O(i^6)$ time since $|x| = i$.

Notice that rejecting w due to being malformed takes $O(n^2)$ time. If D runs $M_A(x)$, then we know this takes $O(i^6)$ time. Since $|w| = n = i + i^2 - i = i^2$, we have that $i = \sqrt{n}$. So $M_A(x)$ runs in time $O(n^3)$ time. Thus, $D(w)$ runs in $O(n^3)$ time in the worst case. $\square$

3.3 Part 3 (10 Points)

1. Prove that $\mathbf{P} \subseteq \mathbf{NP} \cap \mathbf{coNP}$.
2. Prove that if $\mathbf{P} = \mathbf{NP}$ then $\mathbf{NP} = \mathbf{coNP}$.

Proof of Problem 3 Part 3. First, we prove that $\mathbf{P} \subseteq \mathbf{NP} \cap \mathbf{coNP}$. Let $L \in \mathbf{P}$. Notice that $\mathbf{P} = \mathbf{coP}$ since for any $L \in \mathbf{P}$, we can build a decider for the complement language $\overline{L} \in \mathbf{coP}$ by simply running the decider for L and flipping the output. Now, we know that by definition $\mathbf{P} \subseteq \mathbf{NP}$ since any deterministic decider is also a non-deterministic decider (with a single computation branch). For any $L \in \mathbf{P} \subseteq \mathbf{NP}$, note that $\overline{L} \in \mathbf{coNP}$. By our above argument, we also know that $\overline{L} \in \mathbf{P}$. Therefore, $\mathbf{P} \subseteq \mathbf{coNP}$.

Next, suppose that $\mathbf{P} = \mathbf{NP}$. Let $L \in \mathbf{NP}$. By assumption, we know that $L \in \mathbf{P}$. By our previous discussion, we know that $\overline{L} \in \mathbf{P}$. By our assumption, we have $\overline{L} \in \mathbf{NP}$. By definition of $\mathbf{coNP}$, we also have that $\overline{L} \in \mathbf{coNP}$. This implies that $\mathbf{NP} = \mathbf{coNP}$. $\square$

4 Problem 4 (Reductions) (20 Points)

4.1 Part 1 (10 Points)

The *subset sum* problem asks the following: given n non-negative integers $A_1, \ldots, A_n$ and a target $t \in \mathbb{N}$, decide whether there exists a subset $S \subset [n]$ such that $\sum_{i \in S} A_i = t$. Formally, it is given by the language

$$\text{SUBSET-SUM} = \{(A_1, \ldots, A_n; t) \ : \ \exists S \subset [n] \text{ s.t. } \sum_{i \in S} A_i = t\}.$$

Prove that SUBSET-SUM is **NP**-complete via a reduction from 3SAT (i.e., show 3SAT $\leq_p$ SUBSET-SUM).

Proof of Problem 4 Part 1. First, we show that SUBSET-SUM $\in$ **NP**. Given the string $w = (A_1, \ldots, A_n; t)$, we construct a non-deterministic decider D which decides if $w \in$ SUBSET-SUM. Note that D can check if w is well-formed in polynomial time. Now, assuming w is well-formed, the decider D simply guesses some subset $S \subset [n]$ such that $\sum_{i \in S} A_i = t$. D checks if this equality holds; if it does, output accept, else output reject. Now if $w \in$ SUBSET-SUM, there exists a subset S satisfying $\sum_{i \in S} A_i = t$. This implies that D can choose the set S in its non-deterministic guess. Moreover, if there does not exist such a subset S, then D can never guess a correct subset. So D decides SUBSET-SUM.

We now show that SUBSET-SUM is **NP**-complete. We do a reduction from 3SAT. Let ϕ be a 3CNF formula with m variable $x_1, \ldots, x_m$ and n clauses $\phi = \phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_n$, where each $\phi_i = (a_i \vee b_i \vee c_i)$. We now define $2m + 2n$ integers as follows. First, for every $i \in [m]$, we define the integer $T_i = 10^{m+n-i} + \sum_{j \in [n]} 10^{n-1-j} \cdot \mathbf{1}(x_i \in \phi_{n-j+1})$ and $F_i = 10^{m+n-i} + \sum_{j \in [n]} 10^{n-j} \cdot \mathbf{1}(\overline{x}_i \in \phi_{n-j+1})$. Here, $\mathbf{1}$ is the indicator function which outputs 1 if and only if its input expression is true, and outputs 0 otherwise. That is, $\mathbf{1}(x_i \in \phi_{n-j+1})$ evaluates to 1 if and only if variable x_i is in clause ϕ_{n-j+1}; similar for $\overline{x}_i \in \phi_{n-j}$. This gives us $2m$ integes.

We can equally view the integers T_i, F_i as the integers that correspond to the rows of the following table.

	1	2	3	4	$\cdots$	m	ϕ_1	ϕ_2	$\cdots$	ϕ_n
T_1	1	0	0	0	$\cdots$	0	1	0	$\cdots$	1
F_1	1	0	0	0	$\cdots$	0	0	1	$\cdots$	0
T_2	0	1	0	0	$\cdots$	0	1	0	$\cdots$	0
F_2	0	1	0	0	$\cdots$	0	0	0	$\cdots$	1
T_3	0	0	1	0	$\cdots$	0	1	0	$\cdots$	0
F_3	0	0	1	0	$\cdots$	0	0	0	$\cdots$	1
$\vdots$					$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$
T_m	0	0	0	0	$\cdots$	1	0	0	$\cdots$	0
F_m	0	0	0	0	$\cdots$	1	0	0	$\cdots$	0

Now for each clause, we define 2 more integers as follows. For every $i \in [n]$, define $C_i, C_i' = 10^{n-i}$. Adjusting the above table, this gives us $2n$ new rows.

	1	2	3	4	$\cdots$	m	ϕ_1	ϕ_2	$\cdots$	ϕ_n
T_1	1	0	0	0	$\cdots$	0	1	0	$\cdots$	1
F_1	1	0	0	0	$\cdots$	0	0	1	$\cdots$	0
T_2	0	1	0	0	$\cdots$	0	1	0	$\cdots$	0
F_2	0	1	0	0	$\cdots$	0	0	0	$\cdots$	1
T_3	0	0	1	0	$\cdots$	0	1	0	$\cdots$	0
F_3	0	0	1	0	$\cdots$	0	0	0	$\cdots$	1
$\vdots$					$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$
T_m	0	0	0	0	$\cdots$	1	0	0	$\cdots$	0
F_m	0	0	0	0	$\cdots$	1	0	0	$\cdots$	0
C_1							1	0	$\cdots$	0
C_1'							1	0	$\cdots$	0
C_2							0	1	$\cdots$	0
C_2'							0	1	$\cdots$	0
$\vdots$							$\vdots$	$\vdots$	$\ddots$	$\vdots$
C_n							0	0	$\cdots$	1
C_n'							0	0	$\cdots$	1

Here, blanks in the table correspond to a 0. Finally, we set our integers to be A_k as follows. For $k \in [2(m+n)]$:

$$A_k = \begin{cases} T_i & k \leq 2m \wedge k = 2i - 1 \text{ for } i \in [n] \\ F_i & k \leq 2m \wedge k = 2i \text{ for } i \in [n] \\ C_i & k > 2m \wedge k = 2m + 2i - 1 \text{ for } i \in [n] \\ C'_i & k > 2m \wedge k = 2m + 2i \text{ for } i \in [n] \end{cases}.$$

We now set our target integer to be $t = \underbrace{11\cdots1}_{m \text{ times}}\underbrace{33\cdots3}_{n \text{ times}}$. Corresponding to the table, we add a final row for the target integer, where the goal is to collect rows such that they sum to the final target row.

	1	2	3	4	$\cdots$	m	ϕ_1	ϕ_2	$\cdots$	ϕ_n
T_1	1	0	0	0	$\cdots$	0	1	0	$\cdots$	1
F_1	1	0	0	0	$\cdots$	0	0	1	$\cdots$	0
T_2	0	1	0	0	$\cdots$	0	1	0	$\cdots$	0
F_2	0	1	0	0	$\cdots$	0	0	0	$\cdots$	1
T_3	0	0	1	0	$\cdots$	0	1	0	$\cdots$	0
F_3	0	0	1	0	$\cdots$	0	0	0	$\cdots$	1
$\vdots$					$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$
T_m	0	0	0	0	$\cdots$	1	0	0	$\cdots$	0
F_m	0	0	0	0	$\cdots$	1	0	0	$\cdots$	0
C_1							1	0	$\cdots$	0
C'_1							1	0	$\cdots$	0
C_2							0	1	$\cdots$	0
C'_2							0	1	$\cdots$	0
$\vdots$							$\vdots$	$\vdots$	$\ddots$	$\vdots$
C_n							0	0	$\cdots$	1
C'_n							0	0	$\cdots$	1
t	1	1	1	1	$\cdots$	1	3	3	$\cdots$	3

First, notice that we can construct these integers in time that is polynomial in the length of ϕ. In particular, construction of C_i, C'_i takes $O(n)$ time, and construction of T_i, F_i takes $O(n)$ time (just scan through the clauses). This gives an upper bound of $O(n^2)$ to construct all the integers.

Now, we argue that ϕ is satisfiable if and only if there exists a subset $S \subseteq [2(m+n)]$ such that $\sum_{k \in S} A_k = t$. First, suppose ϕ is satisfiable. Then, we construct a subset S such that $\sum_{k \in S} A_k = t$. First, let $w \in \{0,1\}^m$ denote the satisfying assignment of ϕ. If $w_j = 1$, then we interpret the literal x_j as being true/1 (and $\overline{x}_j$ being false/0). If $w_j = 0$, then we interpret the literal $\overline{x}_j$ as true/1 (and x_j as false/0).

For every $j \in [m]$, if $w_j = 1$, we add the index k corresponding to the integer T_j; otherwise, if $w_j = 0$, we add the index k corresponding to the integer F_j. Intuitively, T_j represents x_j as being true, and F_j represents $\overline{x}_j$ as being true. Since ϕ is a satisfying assignment, we cannot pick both T_j and F_j, we can only pick one or the other. After picking each of these integers, notice we have $\sum_{k \in S} A_k = \underbrace{11\cdots1}_{m \text{ times}} d_1 d_2 \cdots d_n$, where each $d_i \in \{1,2,3\}$ for $i \in [n]$. Now each digit d_i is at least 1 because w is a satisfying assignment, so every clause must have at least one satisfied literal in it. Since ϕ is a 3CNF, each clause contains exactly 3 literals. Under assignment w, this implies that each clause can be satisfied by at most 3 literals. This gives the upper bound on each digit. Now, let $T \subset [n]$ denote the indices such that $d_i < 3$. If $d_i = 1$, we then add the indices k and $k+1$ to S such that $A_k = C_i$ and $A_{k+1} = C'_i$. If $d_i = 2$, we add the index k to S such that $A_k = C_i$. All together, this gives us $\sum_{k \in S} A_k = t$, as desired.

For the other direction, suppose that $S \subset [2(m+n)]$ such that $\sum_{k \in S} A_k = t$. From S, we construct a satisfying assignment for the formula ϕ. Notice that in order for the subset to sum to the target t, we must pick m of the $2m$ integers T_i, F_i, otherwise we cannot get the first m 1's in the integer t. Moreover, for any

index $k, k+1$ such that $A_k = T_i$ and $A_{k+1} = F_i$, we cannot have both k and $k+1$ in S, otherwise this digit in the target would equal 2. From this, we construct the satisfying assignment for ϕ as follows. For every $i \in [m]$, if $k \in S$ such that $A_k = T_i$, then set x_i to true; otherwise, set $\overline{x}_i$ to true. We now argue this is a satisfying assignment. Notice that any indices k corresponding to $A_k = C_j$ or $A_k = C'_j$ for some $j \in [n]$ can only contribute at most 2 to the n least significant digits of the target t. That is, in our subset sum, we know that the n least significant digits of t are all 3. In the worst case, 2 integers C_j and C'_j contribute at most 2 to the $(n-j)$-th digit. So the remaining contributions to reach 3 in the n least significant digits must come from the integers corresponding to T_i, F_i. In particular, for every clause ϕ_j, since our subset S sums to the target t, there exists an index i such that either T_i is true and $x_i \in \phi_j$, or F_i is true and $\overline{x}_i \in \phi_j$. Since we pick exactly one of F_i or T_i the clause ϕ_j must be satisfied. Thus, we have constructed a satisfying assignment for ϕ. □

4.2 Part 2 (10 Points)

The *k-independent set* problem for an undirected graph $G = (V, E)$ asks one to find a subset $S \subset V$ such that (1) $|S| \geq k$ and (2) for all $u, v \in S$, we have $(u, v) \notin E$. Let INDSET be the language

$$\text{INDSET}_k = \{(G = (V, E), k) \colon \exists S \subset V \text{ s.t. } |S| \geq k \text{ and } (u, v) \notin E \ \forall u, v \in S\}.$$

We showed that INDSET_k is **NP**-complete in class. Prove that the following language is **NP**-complete by reducing it from INDSET_k (i.e., $\text{INDSET}_k \leq_p \text{CLIQUE}_k$):

$$\text{CLIQUE}_k = \{(G, k) \colon G \text{ has a } k\text{-clique}\},$$

where a k-clique is a set of vertices $S \subset V$ such that $|S| = k$ and $\forall u, v \in S$, it holds that $(u, v) \in E$.

Hint: if G has an independent set of size k, think about what how these vertices are connected in the complement graph, where $\overline{G} = (V, \overline{E})$. That is, $\overline{G}$ contains all edges which are not present in G.

Proof of Problem 4 Part 2. First, notice that $\text{CLIQUE}_k \in$ **NP** since we can construct a deterministic Turing machine that on input (G, k), when given a certificate $w \subset V$, we can verify in polynomial time if w corresponds to a k-clique. The verifier machine does the following.

1. Check that $|w| = k$; i.e., contains k unique vertices. Reject if not.
2. For every $u, v \in w$, check if $(u, v) \in E$ (the edge set of G). If not, reject.
3. Output accept if the for-loop above completes without rejecting.

Clearly, this algorithm runs in time $O(n^2)$, where $n = |V|$, which is polynomial in the input length.

Now, we prove that CLIQUE_k is **NP**-hard by a reduction from INDSET_k. Given any graph G, we let $\overline{G}$ denote the complement graph, where G and $\overline{G}$ have the same vertex set, but opposite edge sets. That is, for every $(u, v) \in E(G)$, we have that $(u, v) \notin E(\overline{G})$, and for every $(u, v) \notin E(G)$, we have $(u, v) \in E(\overline{G})$. Given a graph G, we can construct the complement graph $\overline{G}$ in $O(n^2)$ time, where $n = |V|$. Now we argue that $\overline{G}$ has a k-clique if and only if G as a k-independent set.

First, suppose that G has a k-independent set S. That is, $|S| \geq k$ and for every $u, v \in S$, we have $(u, v) \notin E(G)$. By definition, we know that $(u, v) \in E(\overline{G})$. Since $|S| \geq k$, the set S is a clique of size at least k in $\overline{G}$.

Now, suppose that $\overline{G}$ as a k-clique. Let $T \subset V$ be the vertices corresponding to this clique, where $|T| \geq k$. By definition of a clique, for all $u, v \in T$, we have $(u, v) \in E(\overline{G})$. By definition of the complement graph, we know that $(u, v) \notin E(G)$ for all $u, v \in T$. Then clearly we have that T is an independent set in graph G of size at least k. □

5 Problem 5 (One More Reduction) (20 Points)

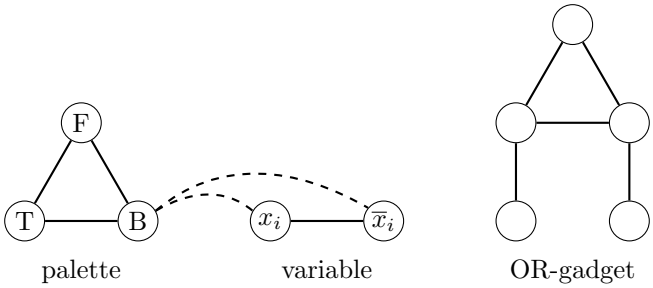
Let $G = (V, E)$ be an undirected graph. A *k-coloring* of G is an assignment of k colors to nodes/vertices such that no adjacent nodes share a color. In other words, given k distinct colors (i.e., labels), label each node/vertex of G such that any two adjacent nodes (i.e., nodes which share an edge) do not share a color.

Let

$$3COL = \{G = (V, E) \colon G \text{ is 3-colorable}\}$$

be the set of all graphs that have a valid 3-coloring. Prove that $3COL$ is **NP**-complete. For proving **NP**-hardness, reduce from 3SAT (i.e., 3SAT $\leq_p$ $3COL$).

Hint: let $\{T, F, B\}$ be your three colors ("True", "False", and "Base"). Use the following three subgraphs to help in your reduction. The OR-gadget can help you capture satisfiability of clauses.



Proof of Problem 5. First, we show that 3COL $\in$ **NP**. We can construct a non-deterministic decider D for deciding $3COL$. On input G, the decider D will use 3 colors and guess a coloring of the graph G. In time $O(n^2)$, the decider D can check if the 3 coloring is valid. If it is valid, it outputs accept, and rejects otherwise (that is, if two adjacent vertices have the same colors). Here, $n = |V|$. Clearly, if $G \in$ 3COL, then the decider D can non-deterministically guess this 3 coloring and accept if it guesses the 3 coloring. Now, if $G \notin$ 3COL, then any coloring guessed by D will not be a valid 3 coloring, so D will always reject. Thus, D decides $3COL$ and $3COL \in$ **NP**.

Now, we show that 3SAT $\leq_p$ 3COL. Let ϕ be a 3CNF formula with n clauses $\phi_1, \ldots, \phi_n$ and m literals $x_1, \ldots x_m$. First, for each clause $\phi_i = (a_i \vee b_i \vee c_i)$, we construct a chained OR-gadget, where a_i, b_i, c_i can be any variable or their negation (e.g., $a_i = x_1$, $b_i = \overline{x}_7$, and $c_i = x_{15}$). The OR-gadget will have the form depicted in the following figure.

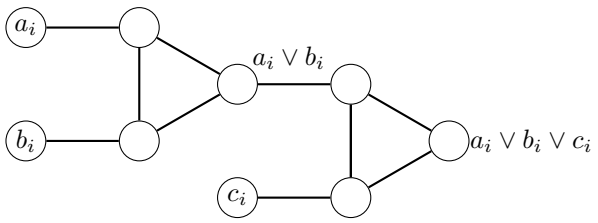


Figure 1: OR-gadget for clause ϕ_i.

For each clause ϕ_i, we construct an OR-gadget for that clause according to Figure 1.

Next, for each literal x_i and its negation $\overline{x}_i$, we add x_i and $\overline{x}_i$ as nodes to the graph and connect them. For each of these pairs, we connect them to the "palette" triangle depicted. In particular, we always connect them to the node in the palette triangle corresponding to the color B. Finally, we connect the output of each extended OR-gadget to both B and F in our palette triangle.

First, we prove a useful property about the OR-gadget.

Lemma 5.1. *If an input node of the OR-gadget is colored T, then there is a valid 3-coloring of the OR-gadget such that the output node is colored T. Moreover, if both the input nodes are colored F, then there is no valid 3 coloring such that the output node is colored T. Here, the input nodes are the nodes with degree 1, and the output node is the node with degree 2.*

For the first part of the lemma, notice that if one input node is colored T, then the node adjacent to it must be B or F. If the other input node is colored T, color one of the adjacent nodes F and the other node B. This corresponds to the two nodes with degree 3 in the OR-gadget. Now, we can color the output node with T since the other two nodes of the triangle are colored F and B. If the other input node is colored F, color the node adjacent to F with B, and the node adjacent to T with F. Then again, we can color the output node with T. Finally, if the other input node is colored B, color the node adjacent to it with F, and the node adjacent to T with B. Then we can again color the output node with T. For the second part of the lemma, notice that if both input nodes are colored F, then in order to have a valid 3 coloring, one of the adjacent nodes must be colored B and the other must be T. Thus, we cannot color the output node T.

With Lemma 5.1 established, we can now proceed with the proof. First, notice that construction of the graph G is polynomial in the size of ϕ. In particular, the graph has at most $3+2m+6n$ nodes. In the worst case, the graph takes $O(n^2)$ time to construct.

Now suppose that ϕ has a satisfying assignment. We construct a valid 3 coloring of our constructed graph. First, if variable $x_i = 1$ under this assignment, we color its corresponding node with T and color $\overline{x}_i$ with F; reverse if $\overline{x}_i = 1$. Note this is a valid coloring since x_i and $\overline{x}_i$ are connected to each other and the node we always color B. Without loss of generality, suppose $x_i = 1$. Then for every clause ϕ_j which contains x_i, one of the input nodes to the OR-gadget corresponding to ϕ_j is colored with T. By Lemma 5.1, we know that we can produce a valid 3 coloring of the OR-gadget where the output node is colored with T. Since the extended OR-gadget is the chaining of 2 OR-gadgets, this is also true. Thus, we can color the output node of the extended OR-gadget for clause ϕ_j with T. Note that we must do this because this output node is connected to both B and F in the palette triangle. Since ϕ is satisfiable, every extended OR-gadget must contain at least one input node colored T, so we can produce a valid 3 coloring of all the extended OR-gadgets such that the output nodes are colored T. Thus, we have produced a valid 3 coloring of our constructed graph.

Next, suppose that our constructed graph is 3 colorable. We construct a satisfying assignment for ϕ by simply taking all the literal nodes x_i or $\overline{x}_i$ which have been assigned T. Note that at least one of these two nodes must be assigned T since they are connected to each other, and both are connected to the node colored B. Moreover, we now have that at least one input node of every OR-gadget corresponding to clause ϕ_j is colored T. By construction of the OR-gadget, under a valid 3 coloring, the output node must be colored T (since it is connected to both B and F in the palette triangle). By Lemma 5.1, we know this 3 coloring is possible. Thus, we know that under the assignment constructed via selecting the literal nodes colored with T, the output node of every OR-gadget is colored T, and at least one of every input node is colored T. Thus, all clauses are satisfied.

Alternatively, suppose this assignment is not a satisfying assignment. Then there is a clause ϕ_j such that it is not satisfied. This implies all input nodes are set to F. But by Lemma 5.1, this would imply that the output node of the OR-gadget for ϕ_j is not colored T. But then we would not have a valid 3 coloring, as by construction of the OR-gadget, the only valid coloring for the output node is T. This contradicts that the graph has a valid 3 coloring. □