CS244 Theory of Computation
Homework 3 Solution

Problem 1

The ***Kolmogorov complexity*** of a bit string b, $K_L(b)$, is the length of the shortest program in language L that outputs b and only b. Is $K_L(b)$ computable? Prove your answer.

Solution

$K_L(b)$ is uncomputable.

Lemma. *For each $n \in \mathbb{N}$, there exists a bit string b such that $K_L(b) > n$.*

Proof. Assume for contradiction that for some $n \in \mathbb{N}$, all the bit strings b have $K_L(b) \leq n$. Then $1 + 2 + 4 + \cdots + 2^n = 2^{n+1} - 1$ programs can generate infinitely many bit strings. However, each program can generate at most one bit string. □

Proof. We prove it for the Turing Machine language. It follows the same procedure for other languages. Assume for contradiction that $K_{\mathsf{TM}}(b)$ is computable. Then there is a TM M on input b, outputs $K_{\mathsf{TM}}(b)$. The following TM S outputs the bit string b with $K_{\mathsf{TM}}(b)$:

```
$S$:
  1. Obtain via the recursion theorem, $\langle S \rangle$.
  2. Go through each possible $b \in \{0,1\}^*$ in string order ($\varepsilon$, 0, 1, 00, 01, 10, 11, 000, ...) and run $M$
     on $b$ to get $K_{\mathsf{TM}}(b)$. If $K_{\mathsf{TM}}(b) > |\langle S \rangle|$ (the length of $S$), output $b$ and halt.
```

The TM S outputs b and only b, so $K_{\mathsf{TM}}(b) \leq |\langle S \rangle|$, which is a contradiction. Thus $K_{\mathsf{TM}}(b)$ is uncomputable. □

Problem 2

Let $SHUFFLE = \{\langle w, x, y \rangle \mid w = a_1 b_1 \ldots a_k b_k$ for $k \geq 0$ where $x = a_1 a_2 \ldots a_k$ and $y = b_1 b_2 \ldots b_k$, each $a_i, b_i \in \Sigma^*\}$.

(a) Show that $SHUFFLE \in$ NP.

Proof. Construct a polynomial-time verifier V:

```
$V$ on input $\langle\langle w, x, y \rangle, \langle a_1, a_2, \ldots, a_k, b_1, b_2, \ldots, b_k \rangle\rangle$:
  1. Test whether each $a_i, b_i \in \Sigma^*$.
  2. Test whether $w = a_1 b_1 \ldots a_k b_k$.
  3. Test whether $x = a_1 a_2 \ldots a_k$.
  4. Test whether $y = b_1 b_2 \ldots b_k$.
  5. If all tests pass, *accept*. Otherwise, *reject*.
```

□

(b) Show that *SHUFFLE* $\in$ P.

Proof. The polynomial time algorithm M deciding *SHUFFLE* using dynamic programming is as follows:

```
$M$ on input $\langle w, x, y \rangle$, where $w = w_1w_2 \dots w_r$, $x = x_1x_2 \dots x_s$, $y = y_1y_2 \dots y_t$:
 1. If $r \neq s + t$, reject.
 2. Assign $table(0, 0) \leftarrow$ True.
 3. For $i = 1$ to $s$:
 4.     Assign $table(i, 0) \leftarrow (table(i-1, 0) \wedge (w_{i-1} = x_{i-1}))$.
 5. For $j = 1$ to $t$:
 6.     Assign $table(0, j) \leftarrow (table(0, j-1) \wedge (w_{j-1} = y_{j-1}))$.
 7. For $i = 1$ to $s$:
 8.     For $j = 1$ to $t$:
 9.         Assign $table(i, j) \leftarrow ((w_{i+j-1} = x_{i-1}) \wedge table(i-1, j)) \vee ((w_{i+j-1} = y_{i-1}) \wedge table(i, j-1))$.
10. If $table(s-1, t-1) = True$, accept; otherwise, reject.
```

The total running time of the algorithm is $O(st)$, which is polynomial in the size of the input. Thus *SHUFFLE* $\in$ P. ☐

Problem 3

Let *SET-SPLITTING* $= \{\langle S, C \rangle \mid S$ is a finite set and $C = \{C_1, \dots, C_k\}$ is a collection of subsets of S, where the elements of S can be colored red or blue so every C_i has at least one red element and at least one blue element$\}$. Show that *SET-SPLITTING* is NP-complete.

Proof. First, we show that *SET-SPLITTING* $\in$ NP. Construct a polynomial-time verifier V:

V on input $\langle \langle S, C \rangle, P \rangle$, where P is the coloring scheme:

1. Test whether each element of S is colored either red or blue by P.
2. Test whether each C_i in C has at least one red element and at least one blue element.
3. If both pass, *accept*; otherwise, *reject*."

Next, we show that *3SAT* $\leq_P$ *SET-SPLITTING*. Given a 3cnf-formula ϕ, where there are m variables $x_1, x_2, \dots, x_m$, the reduction is as follows:

1. $S = \{x_1, x_2, \dots, x_m, \overline{x_1}, \overline{x_2}, \dots, \overline{x_m}, a\}$ (a is a new variable).
2. $C = \{\{x_1, \overline{x_1}\}, \{x_2, \overline{x_2}\}, \dots, \{x_m, \overline{x_m}\}\} \cup \{\{y_1, y_2, y_3, a\} \mid (y_1 \vee y_2 \vee y_3)$ is a clause in $\phi\}$.

($\rightarrow$) If ϕ is satisfiable, then there is an assignment making ϕ True and each clause contains at least one literal that is True. Color the elements of S such that all True literals are colored red and all False literals are colored blue; specially, color a blue. A variable can either be assigned True or False, so in each $\{x_i, \overline{x_i}\}$ there is exactly one red element and one false element. Also, in each $\{y_1, y_2, y_3, a\}$, at least one of y_1, y_2, y_3 is colored red, and at least a is colored blue. Thus this is a valid coloring scheme.

($\leftarrow$) If there is a valid coloring scheme for $\langle S, C \rangle$, assign False to all literals in the same color as a and assign True to all literals in the other color. At least one literal in each clause is True, and $\{x_i, \overline{x_i}\}$s guarantee that the assignment is not contradictory. Thus ϕ is satisfiable.

The reduction can be computed in polynomial time. $3SAT$ is NP-complete, thus *SET-SPLITTING* is NP-complete. ☐

CS244 Theory of Computation Homework 3 Solution

(Optional Problems)

Problem 4

(Optional) Let $MODEXP = \{\langle a,b,c,p\rangle \mid a,b,c,p$ are positive binary integers such that $a^b \equiv c \pmod p\}$. Show that $MODEXP \in \mathrm{P}$.

Proof. The square-and-multiply algorithm M for $MODEXP$ is as follows.

$M =$ "On input $\langle a,b,c,p\rangle$, where a,b,c,p are positive binary integers and $b = b_{l-1}b_{l-2}\dots b_1b_0$:

```
1. Assign $d \leftarrow 1$, $x \leftarrow a \bmod p$.
2. For $i$ from 0 to $l-1$:
3.     If $b_i = 1$, then assign $d \leftarrow d \cdot x \bmod p$.
4.     Assign $x \leftarrow x^2 \bmod p$.
5. Assign $y \leftarrow c \bmod p$.
6. If $x = y$, *accept*; otherwise, *reject*."
```

Before the i-th iteration, x has the value $a^{2^i} \bmod p$, thus $d = \left(\prod_{i=0}^{l-1}(a^{2^i})^{b_i}\right) \bmod p = a^{\sum_{i=0}^{l-1} 2^i b_i} \bmod p = a \bmod p$. So the algorithm is correct.
Time analysis: Stage 1, 5, and 6 are executed only once. Stage 3 and 4 run l times, each time taking $O(k^2)$ time, where k is the length of p. The total running time is polynomial in the size of the input, so $MODEXP \in \mathrm{P}$. □

Problem 5

(Optional) Show that if P = NP, then every language $A \in \mathrm{P}$, except $A = \emptyset$ and $A = \Sigma^*$, is NP-complete.

Proof. First, $A \in \mathrm{P}$ and P = NP, so $A \in \mathrm{NP}$.
Next, we take any language B in NP (thus in P) and show that B is polynomial time reducible to A. $A = \emptyset$ and $A = \Sigma^*$, so there is some string $x \in A$ and some string $y \notin A$. Map all strings in B to x and all strings not in B to y. Since $B \in \mathrm{P}$, we can decide whether a string w is in B in polynomial time. Thus the reduction can be computed in polynomial time.
Thus A is NP-complete. □

Problem 6

(Optional) Show that if P = NP, a polynomial time algorithm exists that produces a satisfying assignment when given a satisfiable Boolean formula. (Hint: Use the satisfiability tester repeatedly to find the assignment bit-by-bit.)

Proof. Assume there are k variables $x_1, x_2, \ldots, x_k$. First, substitute x_1 by True and see if the resulting formula is satisfiable. If not, substitute x_1 by False. Repeat the process for other variables. Given P = NP, there is a polynomial time algorithm deciding if a Boolean formula is satisfiable. We decide satisfiability k times, where k is definitely less than the size of the input. Also, we substitute each occurrence of each variable at most twice, the running time of which is also polynomial. Thus the total running time is polynomial. □

Problem 7

(Optional)

(a) Explain why the following argument fails to show that *MIN-FORMULA* $\in$ coNP:

i. If $\phi \notin$ *MIN-FORMULA*, then ϕ has a smaller equivalent formula.

ii. An NTM can verify that $\phi \in \overline{\textit{MIN-FORMULA}}$ by guessing that formula.

(b) Show (despite part a) that if P = NP, then *MIN-FORMULA* $\in$ P.

Solution

(a) After guessing the smaller equivalent formula, we have to test whether it is equivalent to ϕ, which not necessarily can be carried out in polynomial time.

(b) *Proof.* If P = NP, then coNP = coP = P.

To test if two formulas are not equivalent, we can guess the assignment where the two formulas evaluate to different values and check that in polynomial time. So testing if two formulas are not equivalent is in NP and testing if two formulas are equivalent is in coNP, thus in P.

Since we can test equivalence of two formulas in polynomial time, the argument in part a shows that *MIN-FORMULA* $\in$ coNP. Thus *MIN-FORMULA* $\in$ P. □