

Solutions for cs154 Homework #5

Easy Problems (2 Points each)

Part (a) (Sipser 5.3) Find a match in the following instance of the PCP: $\left\{ \left[\frac{ab}{abab} \right], \left[\frac{b}{a} \right], \left[\frac{aba}{b} \right], \left[\frac{aa}{a} \right] \right\}$

There are several solutions. Here is one: $\left[\frac{ab}{abab} \right] \left[\frac{ab}{abab} \right] \left[\frac{aba}{b} \right] \left[\frac{b}{a} \right] \left[\frac{b}{a} \right] \left[\frac{aa}{a} \right] \left[\frac{aa}{a} \right]$

Part (b) (Sipser 7.1) Answer each part TRUE or FALSE:

True (a) $2n = O(n)$	False (b) $n^2 = O(n)$	False (c) $n^2 = O(n \log^2 n)$
True (d) $n \log n = O(n^2)$	True (e) $3^n = 2^{O(n)}$	True (f) $2^{2^n} = O(2^{2^n})$

Part (c) (Sipser 7.5) Is the following formula satisfiable? $(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{x} \vee y) \wedge (\bar{x} \vee \bar{y})$

Not satisfiable. There is no assignment of x and y that makes the formula true.

Part (d) (Sipser 7.6) Show that P is closed under union, concatenation and complement.

Union: Given two languages $P_1 \in P$ and $P_2 \in P$, the TM that accepts $P_1 \cup P_2$ is the following: Upon receiving the input string w , check if $w \in P_1$. If not, then check if $w \in P_2$. Accept the input iff P_1 or P_2 accepts. Since each membership check requires polynomial time, the overall time is polynomial.

Concatenation: Given two languages $P_1 \in P$ and $P_2 \in P$, the TM that accepts $P_1 P_2$ is the following: An input string of length n can be split into two strings in n different ways. For each split, we check if the first substring belongs to P_1 and the second belongs to P_2 . If any split succeeds, the TM accepts. Note that overall time is polynomial.

Complement: Simply switch the accept and reject states of the TM.

Part (e) (Sipser 7.10) A **triangle** in an undirected graph is a 3-clique. Show that $TRIANGLE \in P$, where $TRIANGLE = \{ \langle G \rangle \mid G \text{ contains a triangle} \}$.

Let V and E denote the set of vertices and edges of the graph. We enumerate all triples $\langle u, v, w \rangle$ where $u, v, w \in V$ and $u < v < w$ and check whether all three edges (u, v) , (v, w) and (w, u) exist in E or not. Enumeration of all triples requires $O(|V|^3)$ time. Checking whether all three edges belong to E takes $O(|E|)$ time. Overall time is $O(|V|^3|E|)$ which is polynomial in the length of the input. Therefore, $TRIANGLE \in P$.

Problem 1 (10 Points)

(Sipser 5.17) Show that the PCP is decidable over a unary alphabet, that is, over the alphabet $\Sigma = \{1\}$.

If there is a domino with the same string on top and bottom, we accept. Otherwise, if there are two dominoes, one with more 1s on the top than the bottom, and another with more 1s on the bottom than the top, we accept. In all other cases, we reject.

Proof of correctness: If there is some domino with same string on top and bottom, we have found a solution consisting of just this domino. If all dominoes are top-heavy or bottom-heavy, there is no possible match. Otherwise there exists a domino that is top heavy and another that is bottom-heavy. Let the difference in top and bottom strings of these two dominoes be a and b respectively. Then making b copies of the first and a copies of the second domino makes the top string match the bottom string.

Problem 2 (10 Points)

(Sipser 5.21) **A two dimensional finite automaton** (2DIM-DFA) is defined as follows. The input is an $m \times n$ rectangle, for any $m, n \geq 2$. The squares along the boundary of the rectangle contain the symbol $\#$ and the internal squares contain symbols over the input alphabet Σ . The transition function is a mapping $Q \times \Sigma \rightarrow Q \times \{L, R, U, D\}$ to indicate the next state and the new head position (Left, Right, Up, Down). The machine accepts when it enters one of the designated accept states. It rejects if it tries to move off the input rectangle or if it never halts. Two such machines are equivalent if they accept the same rectangles. Consider the problem of testing whether two of these machines are equivalent. Formulate this problem as a language, and show that it is undecidable.

Solution: The language is $EQ_{2DFA} = \{\langle 2DFA_1, 2DFA_2 \rangle \mid L(2DFA_1) = L(2DFA_2)\}$

We also define $E_{2DFA} = \{\langle 2DFA \rangle \mid L(2DFA) \text{ is empty}\}$.

E_{2DFA} is Undecidable: Reduction from E_{LBA} (See Sipser Theorem 5.9). Recall that $E_{LBA} = \{\langle M \rangle \mid M \text{ is an LBA and } L(M) \text{ is empty}\}$. A decider for E_{LBA} could be built as follows: Construct a $2DFA$ that *verifies the computation history of LBA* such that there is a 1-1 correspondence between $L(LBA)$ and the set of rectangles accepted by the $2DFA$. The idea is the following: $2DFA$ will accept those rectangles that correspond to a valid computation history of LBA . The string in the top row will be the input, alongwith the head position and the start state of the LBA. The $2DFA$ will just verify that subsequent rows correspond to successive valid computations by the LBA. Essentially, this means verifying that the string is identical except for a change in head position, one tape character and the current state of the LBA. The $2DFA$ accepts iff the last row has an accept state of the LBA.

EQ_{2DFA} is Undecidable: Reduction from E_{2DFA} . To decide whether a given $2DFA$ has an empty language, we simply construct the pair $\langle 2DFA, 2DFA_{empty} \rangle$ and feed it to EQ_{2DFA} (and accept iff EQ_{2DFA} does), where $2DFA_{empty}$ is a hard-coded $2DFA$ that does not accept any rectangles.

It is also possible to directly reduce E_{LBA} to EQ_{2DFA} .

Problem 3 (10 Points)

(Sipser 7.12) Let $MODEXP = \{\langle a, b, c, p \rangle \mid a, b, c \text{ and } p \text{ are binary integers such that } a^b \equiv c \pmod{p}\}$. Show that $MODEXP \in P$. Note that the most obvious algorithm does not run in polynomial time. *Hint: Try it first where b is a power of 2.*

Let $|a|, |b|, |c|$ and $|p|$ denote the sizes of a, b, c and p respectively. The size of a^b is quite large. It is $O(|a|^b)$ bits, which is exponential in the size of the input. However, $(a^b \bmod p)$ can be computed in $O(\text{poly}(|a|, |b|, |p|))$ time as we will shortly show. Computing $(c \bmod p)$ requires $O(\text{poly}(|c|, |p|))$ time. Comparing the two moduli requires only $O(\text{poly}(|p|))$ time. The overall time is thus polynomial.

How do we compute $(a^b \bmod p)$ in $O(\text{poly}(|a|, |b|, |p|))$ time? The idea is to first compute a table containing $(a \bmod p)$, $(a^2 \bmod p)$, $(a^4 \bmod p)$, $(a^8 \bmod p)$, \dots . Note that $a^{2^i} \bmod p = (a^{2^{i-1}} \bmod p)^2 \bmod p$, which requires $O(\text{poly}(|p|))$ time to compute. The first modulus $(a \bmod p)$ requires $O(\text{poly}(|a|, |p|))$ time. What is the maximum i for which we precompute $a^i \bmod p$? It is $O(\log b)$. The reason is that from the table of moduli, we can quickly compute $a^b \bmod p$ by multiplying together only those entries in the table that correspond to 1-bits in the binary expansion of b . This allows us to compute $a^b \bmod p$ in $O(\text{poly}(|a|, |b|, |p|))$ time.

Problem 4 (10 Points)

(Sipser 7.13) Show that P is closed under the star operation. *Hint: On input $y = y_1 \dots y_n$ for $y_i \in \Sigma$, build a table indicating for each $i \leq j$ whether the substring $y_i \dots y_j \in A^*$ for any $A \in P$.*

Given a language $L \in P$, we can recognize if $y \in L^*$ in $O(\text{poly}(|y|))$ time as follows. Let $y = y_1 y_2 \dots y_n$. Scan the string y from left to right. Mark y_i if $y_1 y_2 \dots y_i \in L$ (this can be done in polynomial time since $L \in P$). If $y_1 y_2 \dots y_i \notin L$, then mark y_i if there exists some $j \leq i$ such that y_j is marked and $y_{j+1} y_{j+2} \dots y_i \in L$. This again requires polynomial time because there are only i choices for j and each membership test requires polynomial time. At the end, check if y_n is marked.

Problem 5 (10 Points)

(Sipser 7.15) Let $UNARY - SSUM$ be the subset sum problem in which all numbers are represented in unary. Why does the NP-completeness proof for $SUBSET - SUM$ fail to show that $UNARY - SSUM$ is NP-complete? Show that $UNARY - SSUM \in P$.

Solution: Given a set X of integers, we have to identify if the sum of numbers in some $Y \subseteq X$ is t or not.

Let p be the sum of negative integers in X . If there is no negative integer, we set $p \leftarrow 0$. Let q be the sum of all positive integers in X .

Construct a bit-array $EXISTS[]$ whose index ranges from p to q . Initially, all bits are set to 0. We iterate over elements of X one by one. For each element x , we do two things:

- a) For each bit $EXISTS[i]$ that is already 1, set $EXISTS[i + x] \leftarrow 1$.
- b) Set $EXISTS[x] \leftarrow 1$.

Finally, we accept iff the entry $EXISTS[t]$ is true.

Analysis: Computing p and q is $O(n)$ where n is the length of the input string. The size of the array is $q - p = O(n)$. There are $O(n)$ iterations. Each iteration checks $O(n)$ bits. Running time is clearly polynomial.