Homework 5

Problem 1. Prove the following corollary of the time hierarchy theorem: for all constants $c_1 > c_0 \geq 1$, $\mathrm{TIME}(n^{c_0}) \subsetneq \mathrm{TIME}(n^{c_1})$.

Solution. Assume n^{c_0} and n^{c_1} are all time-constructible. Then we only need to prove n^{c_0} is $o(n^{c_1}/\log(n^{c_1}))$ in order to prove the corollary by the time hierarchy theorem.

$\forall \varepsilon > 0$, we will prove that $\exists N, \forall n > N, n^{c_0} < \varepsilon \frac{n^{c_1}}{\log(n^{c_1})}$. The inequality is equivalent to $c_1 \log n \times n^{c_0} < \varepsilon n^{c_1} \iff c_1 \log n < \varepsilon n^{c_1 - c_0}$. Since $c_1 > c_0$, we have $c_1 - c_0 > 0$, and there is always a large enough n for a positive-exponent power function to surpass a logarithmic function. Thus there exists some N such that the inequality holds for all $n > N$. Therefore, n^{c_0} is $o(n^{c_1}/\log(n^{c_1}))$, and the corollary is proved.

Problem 2.

(a) Show that P is closed under union, concatenation, and complement. That is, $A \cup B, A \circ B, A^c \in \mathrm{P}$ if $A, B \in \mathrm{P}$. Note that the concatenation $A \circ B$ of two languages A and B is defined as

$$A \circ B = \{xy \mid x \in A, y \in B\}.$$

(b) Show that P is closed under the star operation. That is, $A^* \in \mathrm{P}$ if $A \in \mathrm{P}$ where $A^* = \{x_1 x_2 \cdots x_k \mid k \geq 0, x_j \in A \text{ for } j = 1, 2, \ldots, k\}$. (Hint: You may need to use dynamic programming to maintain a table whose i, j-th entry indicates whether $x_i \cdots x_j \in A^*$)

Solution.

(a) $P = \bigcup_{c>0} \mathrm{TIME}(n^c)$, so if $A, B \in P$, there exists TMs T_A, T_B that decide A and B respectively in time $O(n^{c_A}), O(n^{c_B})$ for some $c_A, c_B > 0$.

1. Consider TM $T_{A \cup B}$: on any input w with length $|w| = n$, run $T_A(w)$ and $T_B(w)$ separately, and accept if either of them accepts. Thus $T_{A \cup B}$ can decide $A \cup B$, and since $T_A(w)$ and $T_B(w)$ will end in time $O(n^{c_A}), O(n^{c_B})$ respectively, $T_{A \cup B}$ will end in time $O(n^{c_A} + n^{c_B}) = O(n^{\max\{c_A, c_B\}})$.

2. For any input w with length $|w| = n$, say $w = w_1 w_2 ... w_n, w_i \in \Sigma$, there are $n+1$ kinds of separations of w: $w = \epsilon w = w_1(w_2...w_n) = (w_1 w_2)(w_3...w_n) = ... = w\epsilon$. Denote $w_i...w_j$ as w_{i-1}^j and $w_i^i = \epsilon$. Consider TM $T_{A \circ B}$: on any input w: for all separations $w_0^k w_k^n$, run $T_A(w_0^k)$

and $T_B(w_k^n)$ in sequence. If for some k both $T_A(w_0^k)$ and $T_B(w_k^n)$ accept, then accept. Then $T_{A\circ B}$ can decide $A \circ B$. Since $T_A(w_0^k)$ and $T_B(w_k^n)$ will end in time $O(k^{c_A}), O((n-k)^{c_B})$ respectively, $T_{A\circ B}$ will end in time $\sum_{k=1}^{n+1}(O(k^{c_A}) + O((n-k)^{c_B})) \leq O((n+1) \times (n^{c_A} + n^{c_B})) = O(n^{\max\{c_A, c_B\}+1})$.

3. Consider TM T_{A^c} that decides A^c in time $O(n^{c_A})$ by running T_A and accepting if it rejects.

Therfore, $A \cup B, A \circ B, A^c$ can all be decided in polynomial time, indicating $A \cup B, A \circ B, A^c \in P$.

(b) Say TM T can decide A in $O(n^c)$ time. We will construct a TM T_* to decide A^* by iteratively deciding the substrings of input w. For any input w, $w = w_1...w_n$, $w_i \in \Sigma$, we use w_{i-1}^j to denote substring $w_i w_{i+1}...w_j$, and $w_i^i = \epsilon$. We will maintain a table M (2-dim array) where $M_{ij} = 1$ if w_i^j is in A^* and $M_{ij} = 0$ if not. We will update and use this table by using a multi-track TM, with the second track containing M at a place close to the head. Since the size of this table is $(n+1)^2$, fetching a number at M_{ij} will cost at most $O(n^2)$ complexity. Meanwhile, we will keep the table near the head (to prevent searching for a distance too long) by moving the table each time the head moves, which also cost $O(n^2)$ time complexity. This way, we can assume TM T_* can always fetch or update data M_{ij} in $O(n^2)$ each step, but each step of T_* will cost $O(n^2)$ time complexity for moving the table along with the head. Now let T_* be like this:

Algorithm 1: T_* to decide A^*

Data: $w = w_1 w_2 ... w_n$

Initialize M to be all zeroes.
for $k = 1$ *to* n **do**
    for $i = 0$ *to* $n - k + 1$ **do**
        if $T(w_i^k)$ *accepts* **then**
            $M_{i,i+k-1} = 1$; break;
        end
        else
            for $j = i + 1$ *to* $k - 1$ **do**
                if $M_{i,j} = 1$ *and* $M_{j,k} = 1$ **then**
                    $M_{i,i+k-1} = 1$; break;
                end
            end
        end
    end
end
if $M_{0,n-1} = 1$ **then**
    accept;
end
else
    reject;
end

This T_* will iteratively decides whether each w's substrings, from shortest to longest, belongs to A^*. If a substring $w_i^k \in A^*$, then either $w_i^k \in A$ or $\exists j, i < j < k, w_i^j \in A^* \wedge w_j^k \in A^*$. The algorithm above first check whether $w_i^k \in A$ and then traverse all j to check whether $w_i^j \in A^* \wedge w_j^k \in A^*$. Because w_i^j and w_j^k are both shorter than w_i^k, so whether they are in A^* were already checked in previous steps, and were recorded in the table M, so we only need to read M_{ij} and M_{jk}. After deciding whether $w_i^k \in A^*$ we store it in M for later use. Finally, by reading $M_{0,n-1}$, we decide whether $w \in A^*$.

In the above algorithm, the two outer iterations are for k and i, resulting in $O(n^2)$ iterations. Within each iteration, we first run $T(w_i^k)$ and then check the table for $k - i - 1$ times, costing at most $O(n^c) + n \times O(n^2)$. So the total process costs $O(n^2 \times (n^c + n^3)) = O(n^{\max\{c+2,5\}})$ steps. Meanwhile since each step takes at most $O(n^2)$ time, the total time complexity is at most $O(n^{\max\{c+4,7\}})$, which is still polynomial.

Problem 3. Karatsuba algorithm is an efficient algorithm for multiplying two natural numbers of $n = 2^k$ bits, outperforming the straightforward $O(n^2)$ primary-school method. The key idea is as follows. First, we write the numbers as $a2^\ell + b$ and $c2^\ell + d$ where $\ell = n/2$ and $a, b, c, d \in \{0, 1, \dots, 2^\ell - 1\}$. So the product is

$$ac2^{2\ell} + (ad + bc)2^\ell + bd,$$

and this reduces the computation of the product to four multiplications (ac, ad, bc, bd) of shorter numbers. Second, Karatsuba's key idea is that three multiplications suffice for the computation as one can first compute ac, bd. Then the coefficient in front of 2^ℓ can be computed by one extra multiplication as $ad + bc = (a + b)(c + d) - ac - bd$. Show that Karatsuba's algorithm has time complexity $O(n^{\log_2 3}) \approx O(n^{1.585})$.

Solution. Assume the Karatsuba algorithm has time complexity $t(n)$. for a input with two natural numbers of length n, this algorithm divides the multiplication problem into 3 multiplication problems each with input numbers of length $l = n/2$, and use $O(n)$ time to add them together. Thus, we have $t(n) = 3t(n/2) + O(n)$. For input of length n, the recursion will go on for $\log_2 n$ times. In the first recursion layer, the time consumption is $O(n)$. In the k-th recursion layer, there will be 3^k subproblems, each with input length $n/2^k$, and the time consumption for each subproblem is $O(n/2^k)$. Thus, the total time consumption for the k-th recursion layer is $3^k \times O(n/2^k) = O((\frac{3}{2})^k n)$. Therefore, the total consumption for the whole recursion is $\sum_{k=1}^{\log_2 n} O((\frac{3}{2})^k n) = O(n \times \frac{(\frac{3}{2})^{\log_2 n} - 1}{\frac{3}{2} - 1}) = O(n \times (\frac{3}{2})^{\log_2 n}) = O(n \times n^{\log_2 \frac{3}{2}}) = O(n^{(\log_2 \frac{3}{2} + 1)}) = O(n^{\log_2 3})$.