Analysis of Algorithms

Homework 6 – P vs NP

Abraham Murciano & Elad Harizy

January 17, 2021

Question 1

Part A

We are to prove that if the languages $\mathcal{L}_1$ and $\mathcal{L}_2$ are in P, meaning that there automata that can tell us whether a word is in the language or not in polynomial time ($O(n * k)$ for some constant k), then $\mathcal{L}_1 \cup \mathcal{L}_2 \in P$.

Since $\mathcal{L}_1$ and $\mathcal{L}_2$ can be decided in polynomial time, their union can also, as explained in Part B.

Part B

We are told that the languages $\mathcal{L}_1, \mathcal{L}_2$ can be decided in polynomial time using algorithms A_1, A_2 respectively, with running times $O(n^{k_1}), O(n^{k_2})$. To decide their union, one would have to decide each one individually, and decide their union based on their logical disjunction. Thus the complexity of deciding their union is $O(n^{k_1} + n^{k_2})$, or $O(n^{\max(k_1,k_2)})$, which is polynomial.

Question 2

Part A

We must prove that if $\mathcal{L} \in P$ then $\forall k \in \mathbb{N}, \mathcal{L}^k \in P$. Meaning that for any constant k, we can decide the concatenation of the language to itself k times, in polynomial time.

We will use a lemma which states that if $\mathcal{L}_1, \mathcal{L}_2 \in P$, then $\mathcal{L}_1\mathcal{L}_2 \in P$. (Proof omitted.)

We will prove this by induction.

For $k = 0, \mathcal{L}^k = \mathcal{L}^0 = \{\varepsilon\} \in P$.

Assume that for $k = n, \mathcal{L}^k = \mathcal{L}^n \in P$.

Then for $k = n+1, \mathcal{L}^k = \mathcal{L}^{n+1} = \mathcal{L}^n\mathcal{L}$. However we know that both $\wedge$ and $\mathcal{L}^n$ are in P, so using our lemma, their concatenation, $\mathcal{L}^{n+1}$ must be in P.

Part B

Given that algorithm A_1 decides $\mathcal{L}$ in $O(n^c)$ time, we are to find the complexity of an algorithm A_2 which decides $\mathcal{L}^k$ for some constant k.

To decide $\mathcal{L}^2$, the complexity would be $O((n^c)^2)$, or $O(n^{2c})$. This is because after each character, we must check if the remainder of the input is also in $\mathcal{L}$. So if we repeat this process k times, the algorithm results in a complexity of $O(n^{kc})$.

Part C

Assuming $\mathcal{L} \in P$, and is decidable in $O(n^c)$, we are to suggest an algorithm that decides $\mathcal{L}^*$ in polynomial time. We will use a dynamic programming approach to solve this. If $w = w_1w_2 \dots w_n$ is a word, we shall denote by $w_{i,j}$ (when $i \leq j$) the substring of w which is $w_iw_{i+1} \dots w_j$.

We can decide that $w \in \mathcal{L}^*$ if and only if at least one of the following hold true.

- $w = \varepsilon$
- $w \in \mathcal{L}$
- $\exists uv = w, \text{such that} u \in \mathcal{L}^* \wedge v \in \mathcal{L}^*$

Using this we can compose the following algorithm.

```
function KleeneInP(ℒ, w)
    if w = ε then return True
    if w ∈ ℒ then return True
    for i from 1 to |w| do
        if KleeneInP(ℒ, w_{1,i}) ∧ KleeneInP(ℒ, w_{i+1,|w|}) then
            return True
    return False
```

Assuming all results are stored in a table and are only computed once, there are $\frac{n^2}{2}$ different substrings $w_{i,j}$ for which the function is called. And each of those calls it checks if the substring it received is in $\mathcal{L}$, which takes $O(n^c)$ time. Thus the time complexity of this algorithm is at most $O(n^2 \cdot n^c) = O(n^{2c})$