Homework 4: Time Complexity

CS 212 Nature of Computation
Habib University
Ali Muhammad Asad - aa07190

Fall 2023

1. $\boxed{\text{25 points}}$ (a) $\boxed{\text{15 points}}$ Show that the language, $ALL_{DFA} = \{A \mid A \text{ is a DFA and } L(A) = \Sigma^*\}$, is in P.

   (b) $\boxed{\text{15 points}}$ Argue why the following is a valid or invalid approach to show that $ALL_{NFA} \in$ P.

   On input N (where N is an NFA),
   1. Convert N to a DFA, D, using the conversion algorithm studied in the course.
   2. If the polynomial time decider for ALL_{DFA} accepts D, *accept*; else *reject*.

(a) $ALL_{DFA} = \{A \mid A \text{ is a DFA and } L(A) = \Sigma^*\}$ describes the language of all DFAs that accept every possible string made from their alphabet Σ.

We construct a new language, $E_{DFA} = \{A \mid A \text{ is a DFA and } L(A) = \phi\}$. Let E be the Turing Machine that determines E_{DFA}. By `Theorem 4.4` of the book, we know that this language is decidable. Similarly, from `Theorem 4.4`, we can also conclude that $E_{DFA} \in P$ since its corresponding Turing Machine traverses over each state once, and a DFA has n states, therefore, E_{DFA} is in P.
Now we can construct a Turing Machine T that decides ALL_{DFA} as follows:

$T =$ "On input $\langle A \rangle$, where A is a DFA:
1. Construct a DFA M' that recognizes $\overline{L(A)}$, by flipping the accept and reject states of A.
2. Run the Turing Machine E on input $\langle M' \rangle$, where E determines E_{DFA}.
3. If E accepts, **accept**
4. Else **reject**."

Through the above construction, we have effectively created a Turing Machine T that decides ALL_{DFA} by using the Turing Machine E that decides E_{DFA}. The first step can be done in linear time relative to the number of states in A, running in $O(n)$ time. The second step runs in polynomial time since E decides E_{DFA} in polynomial time. The third step is a constant time operation. Thus, T runs in polynomial time, and $ALL_{DFA} \in P$. The E_{DFA} holds significance as leverages the fact that the emptiness problem for DFAs is in P, and thus its complement. ∎

(b) The given approach is not a valid approach to show that $ALL_{NFA} \in P$. This is because DFAs can be exponentially larger than their NFA counterparts, so converting an NFA to a DFA will not necessarily give a polynomial time algorithm. If the NFA has n states, and the corresponding DFA has m states, then $m \leq 2^n$ depending on the number of unreachable states in the DFA, which is exponential. Therefore, converting an NFA to a DFA is not a polynomial time operation, and the given approach is not valid.

2. $\boxed{\text{20 points}}$ Show that the class NP is closed under concatenation.

Consider two languages $L_1, L_2 \in$ NP and let N_1 and N_2 be their respective non-deterministic polynomial-time deciders.
Let $L = L_1 \circ L_2$ be the language generated by the concatenation of L_1 and L_2. Then we can construct a non-deterministic polynomial-time decider N for L as follows:

$N =$ "On input $w = w_1 w_2 w_3 ... w_n$:

1. for i in 0 to n:
   (a) Simulate N_1 on $w_1 w_2 w_3 ... w_i$.
   (b) If N_1 accepts,
      i. simulate N_2 on $w_{i+1} w_{i+2} ... w_n$
      ii. If N_2 accepts, **accept**.
2. **reject**."

By the above construction, N utlizes N_1 and N_2, which were non-deterministic polynomial time deciders for L_1 and L_2 respectively, thus N is non-deterministic polynomial-time decider itself. The steps in the loop take, altogether, polynomial time; $O(n^k)$, in the worst case. In the worst case, the loop itself runs $n + 1$ times, which implies the worst running time of the algorithm is $O(n^{k+1})$. Therefore, N halts in all cases, and since it utilizes non-deterministic polynomial-time deciders, it is a non-deterministic polynomial-time decider for L.

Hence proved that NP is closed under concatenation.

■

3. $\boxed{\text{25 points}}$ Define a *coding* κ to be a mapping, $\kappa : \Sigma^* \to \Sigma^*$ (not necessarily one-to-one). For some string $x = x_1 x_2 \cdots x_n \in \Sigma^*$, we define $\kappa(x) = \kappa(x_1)\kappa(x_2)\cdots\kappa(x_n)$ and for a language $L \subseteq \Sigma^*$, we define $\kappa(L) = \{\kappa(x) : x \in L\}$. Show that the class NP is closed under *codings*.

We need to show that for an arbitrary language L, if $L \in NP$, and if κ is a coding defined on the alphabet of L, then $\kappa(L) \in NP$. Since $L \in NP$, there exists a non-deterministic Turing Machine M that verifies L in polynomial time. Then we can construct a deterministic polynomial time verifier, V for $\kappa(L)$ as follows:

$V =$ "On Input $\langle w, \langle x, c \rangle \rangle$

1. Compute $\kappa(x)$ from x. If $\kappa(x) \neq w$, reject. Else go to step 2.

2. Simulate M on x with certificate c, $\langle x, c\rangle$. If M accepts, **accept**; else **reject**. "

The above shows that for any string $w \in \kappa(L)$, we have $\langle x, c\rangle$ as certificate of w, where c is the certificate for x in L. Thus, if $w = \kappa(x)$, then we make the string $\langle x, c\rangle$ as certificate of w if c is the certificate for x. Further, the verifier V for $\kappa(L)$ can verify a string w in polynomial time by leveraging the verifier M for L. It uses the fact that if $w \in \kappa(L)$, then there must be some string $x \in L$ such that $\kappa(x) = w$, and x can be verified by M in polynomial time with the appropriate certificate c.

Hence, $\kappa(L) \in NP$, and NP is closed under codings.

■

4. $\boxed{\text{25 points}}$ Show that 2SAT $\in$ P, where 2SAT $= \{\phi \mid \phi$ is a satisfiable 2cnf-formula$\}$. You must give a high level description of the algorithm, and show that it runs in polynomial time. *Hint*: A disjunctive clause $(x_1 \lor x_2)$ is logically equivalent to $\neg x_1 \implies x_2$ or to $\neg x_2 \implies x_1$.

A `cnf-formula` comprises of several clauses, each of which is connected with $\land$s. A `2cnf-formula` has several clauses each of which has at most two literals. `2cnf-formula` is satisfiable if there exists an assignment of truth values to the variables such that the formula evaluates to true.

Consider any arbitrary `2cnf-formula` ϕ with n variables and m clauses. Then 2SAT can be shown to be decidable in polynomial-time by the construction of a graph G, and using path searching within the graph.

Let $G = (V, E)$ be such a graph, such that:

$$V = \{x \mid x \text{ is a literal in } \phi\}$$
$$E = \{(x_1, x_2) \mid x_1 \implies x_2 \text{ is a clause in } \phi\}$$

Our graph will have $2n$ vertices, where each vertex represents a true or not true literal for each variable in ϕ. Hence, for n literals, we have $2n$ vertices, intuitively. Then for each clause $(x_1, x_2) \in \phi$, create a directed edge from $\neg x_1$ to x_2 and from $\neg x_2$ to x_1. This is because the clause $(x_1 \lor x_2)$ is logically equivalent to $\neg x_1 \implies x_2$ and $\neg x_2 \implies x_1$, and signifies that if x_1 is not true, x_2 must be true for the clause to be satisfied, and vice versa. Then by this construction of edges, we ensure that there exists a directed edge $(x_1, x_2) \in G$ iff there exists a clause $(\neg x_1 \lor x_2) \in \phi$.

Then by our construction, we can also ensure and **claim** that a `2cnf-formula` is satisfiable iff there exists a variable x such that:

- there is a path from x to $\neg x$ in G, and
- there is a path from $\neg x$ to x in G.

We can quickly go about proving this claim through a simple contradiction. Suppose there are path(s) from x to $\neg x$ and $\neg x$ to x for some variable x in G, but there also exists a satisfying assignment for ϕ. Let $p(x_1, x_2, ..., x_n)$ be this assignment. Now there can be two cases for this satisfying assignment as follows:

Case 1: Let $p(x_1, x_2, ..., x_n)$ be such that $x = \text{TRUE}$

Then let the path x to $\neg x$ be such; $x \to \alpha_1 \to \alpha_2 \to ... \to \alpha_n \to \neg x$. Now if x is TRUE, then α_1 must also be TRUE because there is a directed edge from x to α_1, which represents the clause $\neg x \vee \alpha_1$. If $\neg x$ were FALSE (which it is, since x is TRUE), α_1 must be true to satisfy the clause. Applying this same reasoning recursively along the path, we get that α_2 must also be true because of the clause $\neg\alpha_1 \vee \alpha_2$, and each subsequent α_i must be TRUE because of the clause $\neg\alpha_{i-1} \vee \alpha_i$. This implies that $\neg x$ must be TRUE to satisfy the clause $\neg\alpha_n \vee \neg x$, which is a contradiction since x was assigned TRUE. Thus, if there is a path from x to $\neg x$, the assumption that ϕ is satisfiable with x being TRUE is false.

Case 2: Let $p(x_1, x_2, ..., x_n)$ be such that $x = \text{FALSE}$

Then let the path $\neg x$ to x be such; $\neg x \to \alpha_1 \to \alpha_2 \to ... \to \alpha_n \to x$. We follow the same reasoning as in Case 1, but with the negation of x being TRUE, and ultimately arrive at x being TRUE, which is a contradiction. Thus, if there is a path from $\neg x$ to x, the assumption that ϕ is satisfiable with x being FALSE is false.

Through this, we can conclude that by checking for the existence of a path from x to $\neg x$ or $\neg x$ to x in G, we can determine whether a `2cnf-formula` ϕ is satisfiable or not. The existance of such a path can be determined by trivial graph-traversal algorithms such as DFS or BFS, both of which take polynomial time of $O(V + E)$ time where V is the number of vertices and E is the number of edges in G. Since G has $2n$ vertices and m edges, the algorithm runs in $O(n + m)$ time, which is polynomial time. Thus, 2SAT is decidable in polynomial time, and 2SAT $\in$ P.

A high level description following from the above construction and proof can be given as follows:

1. Construct the graph G as described above, that is, for each clause $(x_1, x_2) \in \phi$, create a directed edge from $\neg x_1$ to x_2 and from $\neg x_2$ to x_1.
2. For each variable x_i, check if there is a path from x_i to $\neg x_i$. If such a path exists, reject.
3. For each variable x_i, check if there is a path from $\neg x_i$ to x_i. If such a path exists, reject.
4. If all variables have been visited, accept.

The above algorithm runs in polynomial time. The creation of the graph can be done in poly-time since we create two vertices for each variable in the `2cnf-formula`, and create edges for each clause in the formula. The graph traversal in steps 2 and 3 can be done in poly-time as well; perforing a DFA or BFS for each vertex to find paths takes $O(V + E)$ time per search. Since we are doing this for $2n$ vertices, the total time taken is $O(2n(2n + E))$, which is polynomial time. Thus, the algorithm runs in polynomial time, and 2SAT $\in$ P.