15-453: Formal Languages, Automata and Computability
Homework # 8 Solutions

1

For this problem, we give one proof using polynomial-time nondeterministic Turing machines, and another using polynomial-time verifiers. However, we point out that it is not significantly more difficult to use the other strategy for either proof.

Union

Strategy. Add a branching start state.

Outline. Let $A, B \in \mathsf{NP}$, i.e. there are nondeterministic Turing machines M and N which decide A and B, respectively. Then consider an NTM which has an extra state which moves nondeterministically to the start states of M and of N. This adds a constant to the runtime of M and N, so it is still polynomial time.

Proof. Let $M = (Q_M, \Sigma, q_{0M}, Acc_M, \delta_M)$ and $N = (Q_N, \Sigma, q_{0N}, Acc_N, \delta_N)$ be NFAs which decide A and B, respectively, in polynomial time. We construct a new Turing machine O as follows:

- $Q = Q_M \sqcup Q_N \sqcup \{q_s\}$
- $q_0 = \{q_s\}$
- $Acc = Acc_M \sqcup Acc_N$
- $\delta(q, \sigma) = \begin{cases} \delta_M(q, \sigma), & q \in Q_M \\ \delta_N(q, \sigma), & q \in Q_N \\ \{(q', \sigma, L) \mid q' \in q_{0M} \sqcup q_{1M}\}, & q = q_s \end{cases}$
- $O = (Q, \Sigma, q_0, Acc, \delta)$

Then O's tree branches to the start states of M and N with the input in the tape, and from there branches exactly as M and N do. Then it reaches a state in Acc iff one of its branches reaches a state in Acc_M or Acc_N, i.e. M accepts or N does. Thus $L(O) = L(M) \cup L(N) = A \cup B$. Moreover, any path in the tree is a path in M's tree or N's tree after the first step, so any path in O's tree is one step longer than some path in M's tree or N's tree; thus O runs within a constant of the maximum of M and N's runtime, so it runs in polynomial time. □

Concatenation

Strategy. Construct a verifier.

Outline. The cerficates for strings in the concatenation language of A and B is the length of the prefix in A, along with the certificate of the prefix in A and the certificate of the suffix in B. A verifier simply uses the length to separate the prefix and suffix, and then runs the two verifiers for A and B with the certificates.

Proof. Let A and B be languages in NP, i.e. there is some poly-time Turing machine M s.t. $x \in A$ iff there is a $c_A(x)$, where $|c(x)|$ is bounded by some polynomial in $|x|$, for which M accepts $\langle x, c_A(x)\rangle$. Similarly, let N be the poly-time verifier for B, and c_B its (partial) certificate function. Then let Z be the concatenation language of A and B, and for $xy \in Z$ (where $x \in A, y \in B$) let $c_Z(xy) = \langle |x|, c_A(x), c_B(y)\rangle$. If our triple function writes the length of its first two elements and then its three elements, $|c_Z(xy)| = \log|x| + \log|c_A(x)| + |x| + |c_A(x)| + |c_B(y)|$. Thus $|c_Z(xy)|$ is bounded by the sum of a polynomial in $|x|$ and a polynomial in $|y|$, which is certainly bounded by the sum of these polynomials in $|x| + |y|$, which is a polynomial in $|xy| = |x| + |y|$. Thus these are poly-length certificates. Then we give the following description of a verifier O for Z:

- If the input is not a pair, or its second argument is not a triple, reject
- Otherwise, the input is $\langle z, \langle l, u, v\rangle\rangle$ Define x to be the first l symbols of z and y to be the rest.
- Run M on $\langle x, u\rangle$. If it rejects, reject.
- Run N on $\langle y, v\rangle$. If it rejects, reject.
- Otherwise, $z = xy$, and $x \in A$ and $y \in B$. Accept.

The runtime of O is a polynomial in the input plus the sum of the runtime of M on shorter input and the runtime of N on shorter input; thus it is polynomial. We have already established that strings in z have accepting poly-length certificates. But any strings with accepting certificates must be in Z by the justification in the last step. Thus O is a verifier for Z, so $Z \in \mathsf{NP}$. □

2

Suppose for contradiction that there is an infinite subset S of incompressible strings and a Turing-Machine M which recognizes S. We define a Turing Machine C, which ignores its input and:

1) Obtains its own description $\langle C\rangle$.

2) Iterates through S until it finds a string with $|x| > |\langle C, \varepsilon\rangle|$ (using dovetailing, described below).

3) Prints x.

Notice that S must contain strings larger than $|\langle C \rangle|$ because it is infinite and hence C is guaranteed to eventually find such an x. Hence we have that $\langle C, \epsilon \rangle$ is a description of x and therefore $K(x) \leq |\langle C, \varepsilon \rangle| < |x|$. However we have that x is an incompressible string. This is a contradiction. Hence, there exists no infinite Turing-Recognizable subset of incompressible strings.

Dovetailing: Consider the set of all strings ordered in shortlex order. Dovetailing is a process in which the i^{th} step of the process runs i steps of a machine M's computation on the first i strings. This process ensures that for any string s and any number t, the machine M will eventually run on string s for at least t steps. Thus, we can dovetail our recognizer M and every string $x \in S$ will eventually be accepted. When this happens, we check if $|x| > |\langle C, \varepsilon \rangle|$

3

A CNF formula is a conjunct of disjuncts. Hence, an assignment will not satisfy F if it fails to satisfy any of its clauses. A clause is not satisfied if the assignment sets every literal in the clause to 0 (or false).

Consider the NFA given by $(Q, \Sigma, \delta, Q_0, F)$ with the following definitions:

- $Q = \{q_{i,j} | 1 \leq i \leq c, 1 \leq j \leq m\}$
- $\Sigma = \{0, 1\}$
- δ is defined for all $1 \leq i \leq c$ and $1 \leq j < m$ as follows:

$$\delta(q_{i,j}, 0)) = \begin{cases} \{q_{i,j+1}\} & \text{if variable } j \text{ appears unnegated in clause } i \text{ or doesn't appear in it at all} \\ \emptyset & \text{otherwise} \end{cases}$$

$$\delta(q_{i,j}, 1)) = \begin{cases} \{q_{i,j+1}\} & \text{if variable } j \text{ appears negated in clause } i \text{ or doesn't appear in it at all} \\ \emptyset & \text{otherwise} \end{cases}$$

- $Q_0 = \{q_{i,1} | 1 \leq i \leq c\}$
- $F = \{q_i, m | 1 \leq i \leq c\}$

This NFA initially branches into a separate set of states for each clause, and then runs deterministically. Each set has m states, one for each variable, with the first state being a start state and the last being an accept state.

Each transition from a state moves to the next state only if the assignment to the variable in the state sets the literal in that clause to false (or if the variable doesn't appear in the clause). Hence we reach the final accept state only if all the literals in the clause are set to false, i.e. the assignment

fails to satisfy the clause. If even a single clause isn't satisfied, the whole CNF isn't satisfied, so we are done.

We have that this NFA has cm states. There are also at most $2cm$ transitions in δ, so this NFA can be constructed in polynomial time.

Now, assume we could minimize NFAs in polynomial time. Consider the SAT problem. If we have a CNF that is not satisfiable, then no assignment can satisfy it and the above constructed NFA for this problem would accept all inputs. This would minimize to an NFA with one state that accepts all inputs. So, we can create a TM that takes in a CNF and creates the above NFA, minimizes it and compares it to the one-state NFA that accepts all inputs. If they're the same, we reject, else we accept. This TM decides SAT in polynomial time. Hence SAT $\in \mathsf{P}$ and so $\mathsf{P} = \mathsf{NP}$.

4

We prove this by induction on n.

Base Case We take L to be a semi-decidable language (i.e. in Σ_1^0) with semi-decider M; assume WLOG that M either accepts or loops, since if not we can replace transitions to its reject state to a state with a self-loop instead. Then $x \mapsto \langle M, x\rangle$ is a mapping reduction from L to $\textsc{Halts} = \textsc{SuperHalts}_0$, because M halts on x iff $x \in L$. Thus $L \leq_m \textsc{SuperHalts}_0$.
Moreover, $\textsc{SuperHalts}_0 \in \Sigma_1^0$ because it can be semi-decided by simulating its input.

Inductive Step Suppose the theorem is true for n; we prove it for $n+1$.
Take $L \in \Sigma_{n+2}^0$, i.e. L is semi-decidable in some $B \in \Sigma_{n+1}^0$.
By inductive hypothesis $B \leq_m \textsc{SuperHalts}_n$, so L is semi-decidable in $\textsc{SuperHalts}_n$. We let M be an oracle Turing machine with oracle for $\textsc{SuperHalts}_n$ which semi-decides it; because $\textsc{SuperHalts}_n \in \Sigma_{n+1}^0$ by inductive hypothesis, we can run $\textsc{SuperHalts}_{n+1}$ on this. As before, take M to either accept or loop. Then $x \mapsto \langle M, x\rangle$ is a mapping reduction from L to $\textsc{SuperHalts}_{n+1}$, so $L \leq_m \textsc{SuperHalts}_{n+1}$.
Moreover, $\textsc{SuperHalts}_{n+1} \in \Sigma_{n+2}^0$ because it can be semi-decided in $\textsc{SuperHalts}_n$ by simulating its input, and using the decider for $\textsc{SuperHalts}_n$ to decide any oracle calls the input makes.