You have 3 hours to complete this exam. You may state without proof any fact taught in class or assigned as homework. Your algorithms in Problems 3–7 must terminate on all inputs.

1. Draw the transition diagram of a one-tape deterministic Turing machine that:

(2 pts) a. decides the language of binary strings containing a prefix with more 1s than 0s;

(3 pts) b. computes the function \( w \mapsto ww \) on strings over the alphabet \( \Sigma = \{0\} \).

Solution.

a. 

b. The Turing machine below first transforms the input \( w \) into \( w \# w \), copying it symbol by symbol, and then transforms \( w \# w \) into \( ww \). In the first stage, \( \bullet \) indicates the symbol being copied.

For a language $L \subseteq \Sigma^*$, recall that $SCRAMBLE(L)$ is the set of all $w \in \Sigma^*$ such that $w$ is either a string in $L$ or can be made into one by changing the order of the symbols. Prove that if $L$ is Turing-recognizable, so is $SCRAMBLE(L)$.

**Solution.** Let $M$ be a Turing machine that recognizes $L$. Since a language is Turing-recognizable if and only if it is recognized by an NTM, it suffices to give an NTM for $SCRAMBLE(L)$. Our NTM works by permuting the input nondeterministically and running $M$ on the permuted input. The NTM constructs the permuted input symbol by symbol, storing it in the blank region of the tape separated from $w$ by a blank. The construction amounts to making $|w|$ passes over $w$, where each pass nondeterministically parks the head in some cell of $w$ and appends a copy of that symbol to the string being constructed. Each time, we write “x” over the selected symbol of $w$ to prevent it from being selected again. When all of $w$ has been overwritten with x’s, we move the permuted string symbol by symbol to be beginning of the tape, position the read/write head in the first cell, and pass control to $M$. 
Give an algorithm for deciding, on input a DFA \( D \) with alphabet \( \{0, 1\} \), whether \( D \) accepts every binary string in which the number of 0s and the number of 1s differ by at most 2018.

**Solution.** First, construct the following PDA \( P \), for the language of binary strings in which the numbers of 0s and 1s differ by at most 2018.

\[
\begin{align*}
0, 1 & \rightarrow \varepsilon \\
1, 0 & \rightarrow \varepsilon \\
0, \varepsilon & \rightarrow 0 \\
1, \varepsilon & \rightarrow 1 \\
\varepsilon, \varepsilon & \rightarrow \bot \\
\varepsilon, 0 & \rightarrow \varepsilon \\
\varepsilon, 1 & \rightarrow \varepsilon \\
\varepsilon, \varepsilon & \rightarrow \varepsilon \\
\varepsilon & \rightarrow \varepsilon
\end{align*}
\]

2018 edges

Then, swap the accept and reject states in \( D \) to obtain a DFA \( \overline{D} \) for the complementary language. Next, combine \( P \) and \( \overline{D} \) using the Cartesian product construction to obtain a PDA \( P' \) that recognizes the intersection of the languages of \( P \) and \( \overline{D} \). Finally, convert \( P' \) into an equivalent context-free grammar \( G' \). Our construction ensures that \( G' \) generates some string if and only if \( D \) rejects some string in which the numbers of 0s and 1s differ by at most 2018. With this in mind, use the algorithm from class to determine whether the language of \( G' \) is empty. If so, accept \( D \); otherwise, reject.

An NFA \( N \) is said to accept a given string \( w \) ambiguously if \( N \) can end up in two or more of the accept states upon processing \( w \). Give an algorithm for deciding, on input an NFA \( N \), whether \( N \) accepts some string ambiguously.

**Solution.** Let \( N = (Q, \Sigma, \delta, q_0, F) \) be given. To start with, convert \( N \) into an equivalent DFA \( D \) using the algorithm from class. Recall that the states of \( D \) are subsets of states of \( N \). Then, use a graph traversal algorithm to check whether there is a path in \( D \) from the start state to a state that contains two or more elements of \( F \). If so, accept \( N \); otherwise, reject.
(3 pts)  5  Give an algorithm for computing, on input a pair of regular expressions over the binary alphabet, the exact number of strings that satisfy them both. Your output should be a specific integer or “infinitely many.”

**Solution.** Convert the regular expressions into NFAs, and the NFAs further into DFAs. Combine the two resulting DFAs into a DFA $D$ that recognizes the intersection of their languages, using the Cartesian product construction. Let $k$ be the number of states in $D$. Next, check to see if $D$ accepts a string of length at least $k$ (see below). If so, the pumping lemma ensures that $D$ accepts infinitely many strings, so output “infinitely many.” If not, run $D$ consecutively on every string of length less than $k$, keeping a running tally of the number of accepted strings, and output the final result.

To check whether $D$ accepts a string of length at least $k$, proceed as follows. Build a DFA for $\Sigma^k\Sigma^*$, such as $D' = (\{0, 1, 2, \ldots, k\}, \Sigma, \delta, 0, \{k\})$ with $\delta(q, \sigma) = \min\{q + 1, k\}$. Then, combine $D$ and $D'$ via the Cartesian product construction to obtain a DFA $D''$ that accepts precisely those strings that are accepted by $D$ and have length at least $k$. Finally, run a graph traversal algorithm to see if an accept state is reachable in $D''$ from the start state.

(3 pts)  6  Give an algorithm for deciding, on input a PDA $P$ and a DFA $D$, whether $P$ accepts only those strings that contain a substring accepted by $D$.

**Solution.** Modify $D$ by adding self-loops, on every $\sigma \in \Sigma$, to the start state and all accept states. The resulting NFA $N$ accepts a string if and only if it has a substring accepted by $D$. Convert $N$ into an equivalent DFA and swap the accept and reject states in it, thus obtaining a DFA $D'$ for the language of strings that do not have a substring accepted by $D$. Now use the Cartesian product construction to obtain a PDA $P'$ for the intersection of the languages of $P$ and $D'$. Our construction ensures that $L(P') = \emptyset$ if and only if $P$ accepts only those strings that contain a substring accepted by $D$. So, convert $P'$ into an equivalent CFG and check to see whether its language is empty. If so, accept $(P, D)$; otherwise, reject.
Give an algorithm that decides, on input a Turing machine $M$ and a string $w$, whether $M$’s head ever moves off the portion of the tape originally containing $w$, when $M$ runs with $w$ as input.

**Solution.** A complete snapshot of $M$’s computation at any specific point in time is given by the current state, the location of the read/write head, and the contents of the tape. Thus, the number of possible snapshots of $M$ operating within the first $|w|$ cells is at most $|Q| \cdot |w| \cdot |\Gamma|^{|w|}$. With this in mind, the algorithm is to just run $M$ on $w$ for $|Q| \cdot |w| \cdot |\Gamma|^{|w|} + 1$ steps. If $M$ has moved beyond the $|w|$th cell during that time, output “yes.” If it hasn’t, it must be that $M$ has repeated some snapshot, which in turn means that $M$ will cycle through those earlier snapshots forever and will never move beyond the $|w|$th cell. So here, we output “no.”

Is there an undecidable language $L \subseteq 0^*$? Prove your answer.

**Solution.** Let $f$ be the standard one-to-one correspondence between $\{0,1\}^*$ and $0^*$. In other words, $f$ maps $\varepsilon, 0, 1, 00, 01, 10, 11, \ldots$ to $\varepsilon, 0, 00, 000, 0000, 00000, \ldots$, respectively. Now, fix any undecidable language $A \subseteq \{0,1\}^*$, which we know exists. Define

$$L = \{f(x) : x \in A\}.$$ 

If $L$ were decidable, we could decide $A$ by converting the input $x$ to $f(x)$ and checking whether $f(x) \in L$. Thus, $A$ is undecidable.