Midterm 1 Solutions

Question 1

The automaton M_k for L_k is given by $M_k = (Q, \{0,1\}, \delta, q_0, A)$, where

$$\begin{aligned} Q &= \{q_0, \dots, q_k\} \\ A &= \{q_0, \dots, q_{k-1}\}\,, \end{aligned}$$

and δ is defined as

$$\begin{aligned} \delta(q_i, 0) &= q_0 && \text{for } 0 \le i < k \\ \delta(q_i, 1) &= q_{i+1} && \text{for } 0 \le i \le k \\ \delta(q_k, x) &= q_k && \text{for } x \in \{0,1\}\,. \end{aligned}$$

For $0 \le i < k$, the automaton is in state q_i when it has read exactly i 1s since the last 0 (or the start of the input if no 0s have been read so far); the automaton is in state q_k if it has, at some point, read k consecutive 1s. In this last case, it rejects; otherwise, it accepts.

Grading notes. 10 marks for a correct automaton; 6/10 if the automaton accepts the complement of the required language. Drop 2 marks if no start state is indicated. If the automaton is close to correct, drop 1 mark for each edge that needs to be changed. A state diagram or description of the automaton in words is fine, as long as it's precise enough to indicate exactly what the automaton is.

We give three alternative proofs that the automaton is minimal. (Only one proof is needed.)

1. We demonstrate a set of $k+1$ pairwise distinguishable strings, namely the strings strings 1^i for $0 \le i \le k$ (where $1^0 = \epsilon$). To see that $1^i \not\approx_{L_k} 1^j$ when $0 \le i < j \le k$, consider the string 1^{k-i-1}. We have $1^i 1^{k-i-1} = 1^{k-1} \in L_k$ but $1^j 1^{k-i-1} \notin L_k$ since $j + k - i - 1 \ge k$. Hence, the set $\{1^i \mid 0 \le i \le k\}$ is a set of $k+1$ pairwise distinguishable strings, so any automaton that accepts L_k must have at least $k+1$ states by Myhill–Nerode.

2. We show that none of the states of M_k are equivalent to each other. The simplest way to do this is to observe, as above, that, for $0 \le i < j \le k$, $q_i \not\equiv q_j$ because reading the string 1^{k-i-1} from state q_i leads to q_{k-1}, which is accepting, but reading that string from state q_j leads to q_k, which is rejecting. All states are inequivalent (i.e., the $\equiv$-equivalence classes are $\{q_0\}, \dots, \{q_k\}$), and the state diagram is connected, so the state minimization algorithm does not reduce the number of states. Therefore, M_k is optimal.

3. We show that no states are equivalent to each other by constructing $\equiv$ as the fixed point of the relations $\equiv_0, \equiv_1, \dots$. We claim that, for $0 \le i < k$, the equivalence classes of $\equiv_i$ are $\{q_0, \dots, q_{k-i-1}\}$, $\{q_{k-i}\}$, …, $\{q_k\}$. If the claim is true then the equivalence classes of $\equiv$ are $\{q_0\}, \dots, \{q_k\}$. Since the state diagram of M_k is connected, the state minimization algorithm does not reduce the number of states, so M_k is optimal.

   We prove the claim by induction on i. For $i = 0$, the equivalence classes are the set of accepting states and the set of rejecting states, which are $\{q_0, \dots, q_{k-0-1}\}$ and $\{q_k\}$, as claimed. Suppose that, for some $i \in \{0, \dots, k-2\}$, the equivalence classes of q_i are $\{q_0, \dots, q_{k-i-1}\}$, $\{q_{k-i}\}$, …, $\{q_k\}$. Then $\{q_{k-i}\}, \dots, \{q_k\}$ are also equivalence classes of $\equiv_{i+1}$. For any $p, q \in \{q_0, \dots, q_{k-(i+1)-1}\}$, we have $p \equiv_i q$ and, for $a \in \{0,1\}$, we have $\delta(p,a), \delta(q,a) \in \{q_0, \dots, q_{k-i-1}\}$, so $\delta(p,a) \equiv_i \delta(q,a)$. Therefore, $p \equiv_{i+1} q$. However, we have $\delta(q_{k-(i+1)}, 1) = q_{k-1} \not\equiv_i p$. So the remaining equivalence classes of $\equiv_{i+1}$ are $\{q_0, \dots, q_{k-(i+1)-1}\}$ and $\{q_{k-(i+1)}\}$. This completes the proof of the claim.

Grading notes. 15 marks for a correct proof, either using $\approx$, or $\equiv$, or the relations $\equiv_i$. If the automaton in the first part of the question was incorrect, give full marks for a correct proof that it is minimal for whatever language it accepts, or for a correct proof that any DFA accepting L_k needs at least $k+1$ states.

For the proof with $\approx$, drop 5 marks if the answer does not include a set of $k+1$ strings that are pairwise distinguishable (either because of the wrong number of strings or because some of them are indistinguishable). Drop 5 marks if the set of indistinguishable strings are not shown to be indistinguishable (i.e., it should be demonstrated that, for all u, v in the set, there is some w such that $uw \in L_k$ and $vw \notin L_k$, or vice-versa). Drop 2 marks if the significance of having $k+1$ pairwise distinguishable strings is not mentioned (it's not necessary to mention Myhill–Nerode by name but it should at least be stated that a minimum automaton must have at least as many states as there are distinguishable strings).

For the proof with $\equiv$, drop 5 marks if the answer does not claim that all the states are inequivalent (e.g., by saying that or by listing the equivalence classes). Drop 5 marks if it doesn't exhibit a string showing this for each pair. Drop 2 marks if the significance of all the states being inequivalent is not explained (i.e., the state reduction algorithm doesn't reduce the number of states).

For the proof with the $\equiv_i$, drop 5 marks if the answer doesn't state what the equivalence classes are for each i (either explicitly or with a clear enough dot-dot-dot). Drop 5 marks if the construction of the successive relations $\equiv_i$ is not justified. (It is not required to set out the induction formally as I have done.) Drop 2 marks if the significance of all the states being inequivalent is not explained.

Question 2

Suppose L is decided by some machine M. We can enumerate L in lexicographic order as follows. First, write # to the output tape. Then, generate all binary strings $s_1, s_2, \dots$ in lexicographic order. For each i in turn, simulate M on s_i and, if M accepts, add $s_i\#$ to the enumeration. Every string $w \in L$ will eventually appear in the enumeration because there are only finitely many simulations to perform before the simulation on w, and M is a decider so each of these simulations terminates.

Conversely, suppose that L is enumerated in lexicographic order by some machine M. To decide whether a string w is in L, work as follows. Simulate M. If $w \in L$, then M will eventually write $\#w\#$ to its output tape, at which point we accept w. If $w \notin L$ then, since L is infinite, it contains infinitely many strings $w' > w$. As soon as M writes $\#w'\#$ to its output tape, where $w' > w$, we know it will never write $\#w\#$ (because it enumerates in order), so we can reject w. This decides L.

Grading notes. 10 marks for constructing the enumeration; 15 for the decider. The answer must justify that the claimed enumerator really is an enumerator (every string in L is eventually included in the enumeration) and how the decider halts for every input (when accepting, because every string in L appears in the enumeration eventually; for rejecting, because there's always a bigger element that will appear).

Comment. If L wasn't necessarily infinite, we'd need to be a little more careful. Suppose that $L = \{0, 1\}$, we have an in-order enumerator and we want to decide if $00 \in L$. Now we're stuck because the enumerator is entitled to write $\#0\#1\#$ and then loop forever, as long as it never writes more output. There is no $w \in L$ that is lexicographically greater than 00, so our decision procedure, which involves waiting for such a string, will fail. The solution in this case is to observe that, if L is finite, it's definitely decidable so we may, in fact, assume L to be infinite. This gives a non-constructive proof: it's undecidable whether an enumerator outputs an infinite set (exercise: prove this) so there's no algorithm that will convert an in-order enumerator for a set into a decider for that set.

Question 3

a) Let $N = (Q, \Sigma, \Delta, q_0, A)$ be a $\forall$FA. For any $w \in \Sigma^*$, $w \in \overline{L(N)}$ if N has at least one rejecting computation. This computation could reject for one of two reasons: it could end in a rejecting state of N, or it could end early because, at some point, the automaton gets "stuck" because it was in a state q and read some symbol a for which $\Delta(q, a) = \emptyset$. There are now two main ways to proceed.

1. Let $q_{\text{fail}} \notin Q$ and consider the NFA $N' = (Q \cup \{q_{\text{fail}}\}, \Sigma, \Delta', q_0, (Q \cup \{q_{\text{fail}}\}) \setminus A)$ where

   $$\Delta'(q,a) = \begin{cases} q_{\text{fail}} & \text{if } q = q_{\text{fail}} \text{ or } \Delta(q,a) = \emptyset \\ \Delta(q,a) & \text{otherwise.} \end{cases}$$

   By the definition of NFAs, N' accepts its input if, and only if, there is a sequence of transitions to one of its accepting states. This happens if, and only if, N has a sequence of transitions to one of its rejecting states or a sequence of transitions that gets "stuck". But this exactly describes the conditions under which N *rejects* its inputs.

   Therefore, N accepts the complement of $L(N)$, which means that $\overline{L(N)}$ is regular (proved in class). Since the complement of a regular language is regular (proved in class), this means that $L(N)$ is regular.

2. We can use the powerset construction directly but, again, we must be careful about F getting stuck. We directly construct a DFA $M = (\mathcal{P}(Q), \Sigma, \delta, \{q_0\}, A')$. We set $A' = \mathcal{P}(A) \setminus \{\emptyset\}$ and

   $$\delta(S,a) = \begin{cases} \emptyset & \text{if } \Delta(q,a) = \emptyset \text{ for some } q \in S \\ \bigcup_{q \in S} \Delta(q,a) & \text{otherwise.} \end{cases}$$

Note that $\delta(\emptyset, a) = \bigcup_{q \in \emptyset} \Delta(q, a) = \emptyset$, as you would expect.

Thus, the state of M represents the set of states that N "could" be in (as in the proof that every NFA is equivalent to some DFA) except that if any possible computation of N "gets stuck" after reading some part of the input, the state of the DFA will be $\emptyset$ from that point on, and the DFA will reject, as required, since N has a non-accepting computation.

Note that we can't just set $\delta(S, a) = \bigcup_{q \in S} \Delta(q, a)$, as in the NFA-to-DFA proof. That's because, as long as $\Delta(q, a) \neq \emptyset$ for at least one $q \in S$, any $q' \in S$ for which $\Delta(q', a) = \emptyset$ will be "forgotten about" when we take the union.

Grading notes. 40/40 for any correct proof. Subtract 10 marks if the answer doesn't cope with "short paths" in F.[1] If the answer constructs or describes a DFA, subtract 5 marks for each of the state space, transition function, start state and accepting states that are not clearly defined; subtract 3 marks if a powerset construction is used but $\emptyset$ is accepting. Score around 10/40 if the answer uses an invalid technique but still demonstrates knowledge of material taught in the course (e.g., closure properties of regular languages).

b) Every regular language is accepted by some $\forall$FA because every DFA is a $\forall$FA: a DFA has exactly one possible computation for each possible input and it accepts if all possible computations (i.e., that one computation) accepts, and rejects otherwise. Every regular language is accepted by some DFA, so every regular language is accepted by some $\forall$FA.

Grading notes. 10/10 for any correct proof.

[1] This turned out to be everybody's answer.