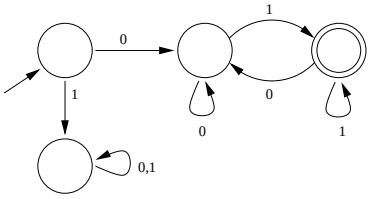
CS–172 Computability & Complexity, Spring 2009
February 10, 2009

Homework 1 Solutions

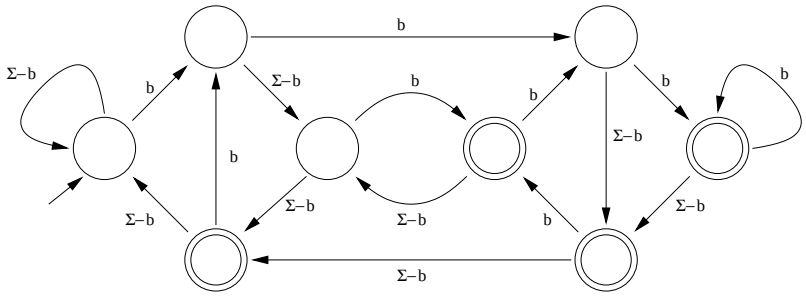
Note: These solutions are not necessarily model answers. Rather, they are designed to be tutorial in nature, and sometimes contain a little more explanation than an ideal solution. Also, bear in mind that there may be more than one correct solution. The maximum total number of points available is 32.

1. DFAs for these three languages are as follows:

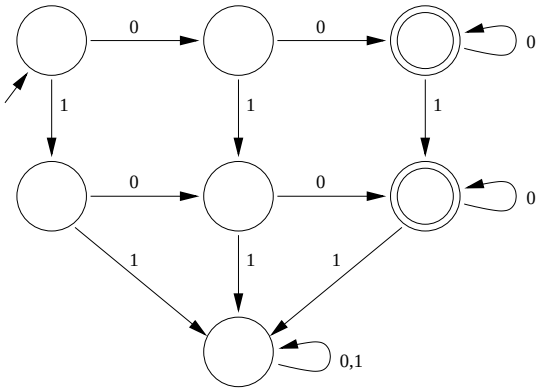
(a) The set of all 0-1 strings that begin with 0 and end with 1. *3pts*



(b) The set of all words over the English alphabet whose third-last letter is 'b'. In the machine below, each state "remembers" the sequence of the last three symbols seen, distinguishing only between 'b' and non-'b' letters; thus it has eight states (corresponding to all 3-letter strings over an alphabet of two letters). Here Σ denotes the English alphabet. *3pts*



(c) The set of all 0-1 strings that contain at least two 0's and at most one 1. In the machine below, the state shifts one place to the right for each 0, and one place down for each 1, stopping when the counts reach 2. It accepts if the first count reaches 2 and the second does not. *3pts*



2. As suggested in the Hint, we describe the sets of input strings w that cause the DFA to end in each of the states (starting from the initial state q_0): *5pts*

- state q_0: all strings w in which every 0 is followed by a 1;
- state q_1: all strings w that end in 0 in which every other 0 is followed by a 1;
- state q_2: all strings w with a pair of consecutive 0's.

Note that these definitions cover all possible input strings, so all we have to do to show the correctness of the DFA is prove that if we run the DFA on a string w then it will terminate in state q_i **if** w satisfies the property ascribed to q_i above. (If some input strings were not covered, then we would have to prove the **only if** direction as well, as it might be possible for the DFA to accept more strings than we claim it does.)

The proof is by induction on the length of the input string w.

Base case: $|w|$ = 0, that is, $w = \epsilon$ (the empty string). The DFA terminates in state q_0, whose property is satisfied by the empty string: there are no 0's that are not followed by a 1. The definitions for the other two states are satisfied vacuously.

Induction step: We make the induction hypothesis that our definitions hold for $w' = \{0,1\}^n$ with $n \geq 0$; we now prove that they also hold for $w = w'a$ with $a \in \{0,1\}$. We consider the states in which the DFA might have determined after processing w':

- state q_0: If $a = 0$ then the DFA transitions to state q_1. By the induction hypothesis every 0 in w' is followed by a 1; $w'a$ now also ends in a 0 so the condition for q_1 is met. If $a = 1$ then the DFA stays in state q_0. Again by the induction hypothesis every 0 in w' is followed by a 1; this remains true with the addition of an extra 1.
- state q_1: If $a = 0$ then the DFA progresses to state q_2. By the induction hypothesis w' ends in 0; the addition of a gives the string a pair of consecutive 0's, and so the condition for q_2 is met. If $a = 1$ then the DFA transitions back to state q_0. As above, w' ends in 0 but otherwise has the properties of the strings that end up in q_0. Adding a 1 meets the requirement that every 0 in the whole string is followed by a 1, including the last 0.
- state q_2: By the induction hypothesis, w' already has a pair of consecutive 0's, so any string containing w' will have this property; therefore the DFA stays in state q_2 as it should.

We have now proved that the DFA will end up in state q_2 given any string w with a pair of consecutive 0's. Since the other two states are accepting, the DFA accepts iff w does not contain a pair of consecutive 0's.

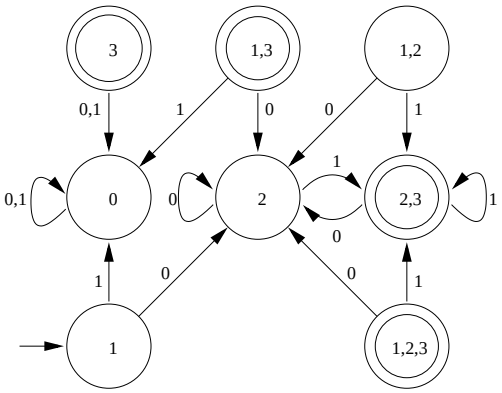
3. (a) Simply interchange the accepting and non-accepting states of M, but leave everything else unaltered. I.e., if $M = (Q, \Sigma, \delta, q_0, F)$, then $M' = (Q, \Sigma, \delta, q_0, Q - F)$. Since δ is unchanged, we have *2pts*

$$\begin{aligned} M \text{ accepts } w &\Leftrightarrow \delta(q_0, w) \in F \\ &\Leftrightarrow \delta(q_0, w) \notin Q - F \\ &\Leftrightarrow M' \text{ does not accept } w. \end{aligned}$$

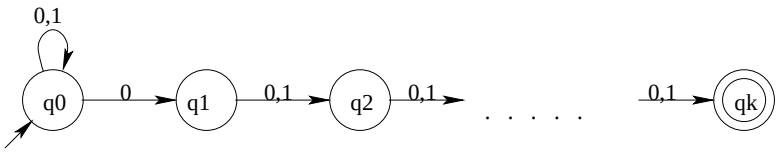
(b) This construction does *not* work when M is nondeterministic. The reason is that, by our definition of acceptance for NFAs, when M accepts only one, not all, of its computations has to be accepting. Thus in the above construction the modified machine M' could accept some of the same strings as M. Here is a simple counterexample: both the automata below accept (for example) the string '0'. *2pts*

M M'

4. The following DFA shows the full construction. Note that if we pruned away states unreachable from the start state, the resulting DFA would be the same as in problem 1(a): *4pts*



5. (a) An NFA for L_k is as shown. At each step, if it sees a 0 it "guesses" if this is in fact the kth last symbol by going into state q_1; from there, it verifies its guess by moving in exactly $k-1$ more steps to the final state. This NFA has $k+1$ states. *3pts*



(b) We construct a DFA such that each of the states represents one of the 2^k possible length-k suffixes. Exactly half of these states, those representing suffixes that start with 0, are accepting states. (If a string ends up in one of the other states, the string must have had a 1 in the kth position from the end, and therefore must be rejected.) We transition from state to state by simply lopping off the first digit of the suffix and appending the next character to be read. (For instance, if $k=4$ then one transition is $(s_{1001}, 0) \to s_{0010}$.) *3pts*

$$\begin{aligned} Q &= \{s_z : z \in \{0,1\}^k\} \\ \Sigma &= \{0,1\} \\ \delta &= \{(s_{aw}, b) \to s_{wb} : a, b \in \{0,1\}, w \in \{0,1\}^{k-1}\} \\ q_0 &= s_{1^k} \\ F &= \{s_z : z \text{ starts with } 0\} \end{aligned}$$

Note the choice of initial state $q_0 = s_{1^k}$. This is in line with the fact that no zero has been seen when the computation begins.

(c) Let M_k be any DFA that accepts L_k. Let us call two strings x and y *distinguishable by* L_k if there is a string w such that *exactly one* of xw and yw is in L_k (i.e., either $xw \in L_k$ and $yw \notin L_k$ or vice versa). *4pts*

The point about distinguishable strings is the following. We claim that for any pair of distinguishable strings x and y, M_k must end up in *different* states after processing x and y. We can prove this by contradiction. Supose on the contrary that x and y are distinguishable but $\delta^*(q_0, x) = \delta^*(q_0, y)$.[1] Then clearly for any symbol a we must have $\delta^*(q_0, xa) = \delta^*(q_0, ya)$, and repeating, for any string w we have $\delta^*(q_0, xw) = \delta^*(q_0, yw)$. (We could prove this formally by a simple induction on the length

[1] Here δ^* is the extended transition function defined in the obvious way: for a state q and string w, $\delta^*(q, w)$ is the state of the DFA after reading string w starting in state q. Formally, we define $\delta^*(q, a) = \delta(q, a)$ for each symbol a, and $\delta^*(q, wa) = \delta(\delta^*(q, w), a)$.

of w, with $|w| = 1$ as the base case.) Therefore, it must be the case that M_k either accepts both xw and yw, or rejects them both. So x and y are not distinguishable, a contradiction.

Now we show that the language L_k has many distinguishable pairs of strings. Consider any two length-k strings x, y with $x \neq y$. Since $x \neq y$, they differ in at least one position: so suppose without loss of generality that $x_i = 0$, $y_i = 1$. Let $w = 1^{i-1}$. Then $xw \in L_k$ and $yw \notin L_k$. (This is because in xw the kth letter from the end is x_i, and similarly for yw.) Thus x and y are distinguishable.

Putting the previous two paragraphs together, we conclude that for any two length-k strings, M_k must end up in different states. Since there are 2^k such strings, M_k must have at least 2^k states.