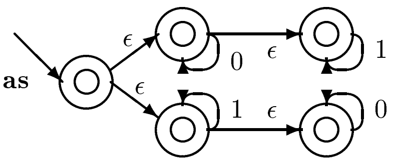
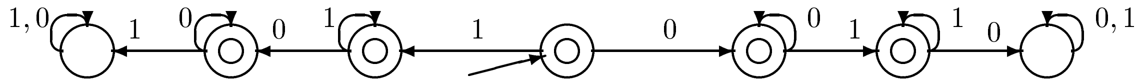
Autumn 1997–98, Prof. Boneh
Solution Set
CS 154, Problem Set 2

2.1 (1 point)

PROBLEM: **Construct a DFA accepting the same language as**

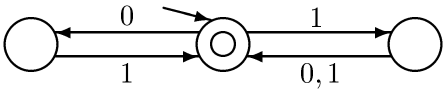




Notes: The subset construction gives the same answer, but it includes some unreachable states, which I've deleted. The remaining states in the subset construction are $\{1,2,3,4,5\}$, $\{2,3,5\}$, $\{3,4,5\}$, $\{3\}$, $\{5\}$, and $\{\}$.

2.2 (1 point)

PROBLEM: **Construct an NFA accepting the same language as the RE** $(01 \cup 10 \cup 11)^*$**.**



Notes: The formal construction is much larger. Note that we can't apply the formal construction directly because we never defined how to construct an NFA for triple union. Instead we parenthesize to get two regular, double unions.

2.3 (2 points)

PROBLEM: **Prove that the following languages aren't regular: (a)** $L_1 = \{0^i1^j : \mathbf{gcd}(i,j) = 1\}$**; (b)** $L_2 = \{1^r0^i1^j0^k : r \geq 5$ **and** $i + j < k\}$**.**

(a) Given m, we pick $w = 0^p1^{(p-1)!}$, where p is some prime greater than $m+1$. Since p is prime, nothing below p shares a factor with it, so $w \in L$. But however the adversary divides $w = xyz$, $xy^0z = 0^{p'}1^{(p-1)!}$, $1 < p' < p$ (since $1 \leq |y| \leq m < p-1$). Thus, $\gcd(p', (p-1)!) = p' > 0$. So $xy \notin L$, so L is not regular.

(b) My favorite proof, due to Satyaki Das: if L_2 is regular, so is L_2^r, so if L_2^r is not regular, neither is L_2. Given m, we pick $w = 0^{m+2}10^m1^5 \in L_2^r$. Since $|xy| < m$, y is all 0s, so xz has $k \leq i + j$ (since $|y| \geq 1$). Thus, L_2^r can't be pumped, and L_2 is not regular.

Notes: The gcd question was the hardest on the problem set. Here are some w that can be pumped: $0^p1^{(p-1)!}$, $0^p1^{(p+1)\cdots(p+m)}$. These require p to be prime so we don't accidentally have a large gcd in w. The first looks at xy^0z, the second at xy^2z. Another popular w, $0^{m!+1}1^{m!}$, requires no primes, since x and $x+1$ are always relatively prime. Once we pump to xy^2z, we'll have $i = m! + x$, and x divides both i and j.

The other problem is easier to pump, but you *must* select a w when $r = 5$. This way, if $|xy| < 5$, you can pump xy^0z to get an illegal string (with too few leading 1s). Otherwise all pumpings are legal in this case.

2.4 (3 points)

PROBLEM: **Describe an algorithm that given a DFA M outputs some shortest word in $L(M)$.**

First, ignore the hint. Then, for $i = 0$ to ∞ do: Run M on each word of length i. If M accepts a word, print it and halt. Obviously we print a word of minimal length in M. (We never halt if $L(M) = \emptyset$, but then the problem's not well defined anyway.)

Notes: This is obviously inefficient. Better is to store, with each non-accepted string, the final state it ended up in. That way, when analyzing $w\sigma$, we start not at the beginning but where w left off. Then it's pretty clear we're just doing a shortest-path algorithm from q_0, where each edge has length 1. Figure out which final state's path back to q_0 is shortest, and read the letters off that path to get the shortest string. (remember, in a FA arcs are labeled with letters.) You'll get w^r.

Of course, when all edges have length 1, a shortest path algorithm degenerates into BFS. Hence the hint. When doing BFS on a non-tree, you mark nodes so you don't process them twice. Then the running time is $O(|\Sigma||Q|)$. Note this implies the shortest word has length at most $|Q|$.

2.5 (3 points)

PROBLEM: **Suppose there are m words $w_1, \ldots, w_m$ that are not L-equivalent. Prove any M accepting L must contain at least m states.**

The proof is clearest if we augment δ to take strings: define $\delta^*(q, \sigma) = \delta(q, \sigma)$ and $\delta^*(q, \sigma w) = \delta^*(\delta(q, \sigma), w)$. Since δ is deterministic, δ^* is too.

Let $s_i = \delta^*(q_0, w_i)$. There are m s_is but fewer than m states, so by the pigeonhole principle there exist w_i, w_j for which $s_i = s_j$, that is, $\delta^*(q_0, w_i) = \delta^*(q_0, w_j)$. Now $\delta^*(q_0, w_i z) = \delta^*(\delta^*(q_0, w_i), z) = \delta^*(\delta^*(q_0, w_j), z) = \delta^*(q_0, w_j z)$. Thus $w_i z$ and $w_j z$ end in the same state for all z (accepting or not), so w_i and w_j are L-equivalent, contrary to our assumption. Thus M has at least m states.

Notes: The crux of the proof is noting δ is deterministic, so if z starts in the same state in two different situations, it still does the same thing each time.

2.6 (Extra Credit)

PROBLEM: **Given L, define $L_{1/2} = \{x : \exists y \quad |y| = |x|$ and $xy \in L\}$. Prove that if L is regular, so is $L_{1/2}$.**

We consider the cross product machine of $M(L)$ and $\mathbf{M(L^r)}$. That is, given an input x, we run it both forwards and backwards on L. However, when running backwards we ignore the actual contents of x and transition (non deterministically) where *any* letter lets us transition – we just have x to make sure we do the right number of transitions. This corresponds to the arbitrary y (of the same length as x). The forward-running part looks at x normally. If the two machines end up in the same state q, we accept. This means the first machine took us from q_0 to q, and the reverse machine took us from q_F to q (backwards), in the same number of steps. Thus $M(L)$ would take us from q_0 to q_F on xy^r.

See me if you're interested in the formal definition and proof.

Notes: Most people did it the following wrong way: $M(L_{1/2}) = M(L)$, except that the final states were all states reached in the middle of (even length) words in L. The problem is that every word landing in these "middle" states would be accepted, not just words who caused us to add the state to $F_{1/2}$. Consider the obvious machine for 0^*11. Then the state looping on 0 will be in $F_{1/2}$ (since $0011 \in L$, and halfway through 0011 we're in the looping state). But then the machine accepts 0 too, which is clearly wrong.