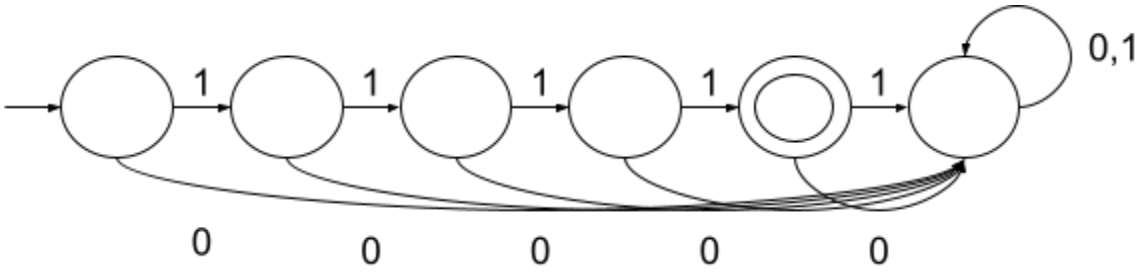
Aughdon Breslin and Isabella Cruz
9/18/20
“I pledge my honor that I have abided by the Stevens Honor System.”

Problem 1

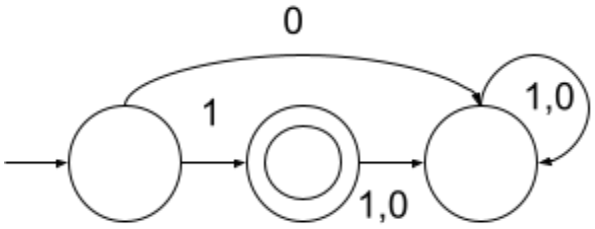
a)



We cannot reduce the number of states to less than 6 because we need to have a start state and a dead state, which take up 2 states. Then we need to have one state for each of the inputs that progress towards (or is) the accept state. If we tried to cut out any of the ‘1’ inputs, we would have an FSA that does not accept only exactly ‘1111’. Either the FSA would not accept specifically ‘1111’ or it would accept more input strings than only ‘1111’. This can be expressed via the Pigeonhole Principle.

b.) Any language that requires an exact input string must have an FSA of size k = length of input + 2. That 2 corresponds to 1 state for the start and 1 for the dead state for when that input sequence is reached but additional inputs exist.

For example, a language A = {1} requires a 3-state FSA to ensure that only exactly an input of a single 1 is accepted.



For k-state FSAs where k > 3, the structure of the start state, accept state, and the dead state would remain the same. The sequence from the start to the accept state would increase according to the length of the accepted input string. Only the input that follows the accepted string would progress towards the accept state, while any deviating inputs would immediately go to the dead state forever. The extension would follow the pattern as seen in 1a.

If the number of states are attempted to be made fewer, the FSA would either not validate the language's inputs, or it would validate more than exactly the language's inputs. This can be displayed via the Pigeonhole Principle as discussed in class. Therefore, for all $k \geq 3$ there is a language that can be accepted by a k-state FSA that cannot be recognized by any FSA with fewer states.

Problem 2
A language is a set of strings. A language is regular if it is recognized by a finite automaton. Given that language A is regular, prove that A^R (A reversed) is regular.

A: Since we know language A is regular, we know that it is recognized by a finite automaton. In a simple form, a finite automaton is a graph with vertices and directed edges. We can simply reverse the direction of those edges, as well as switch the start and accept states to show that A^R is regular as well.

Any states that are no longer reachable starting from the start state can be dropped, which may turn the FSA from a DFA to an NFA. When converting back to a DFA (if we want to convert back), whenever all the clones in our graph die, the machine can instead go to a dead state, Φ.

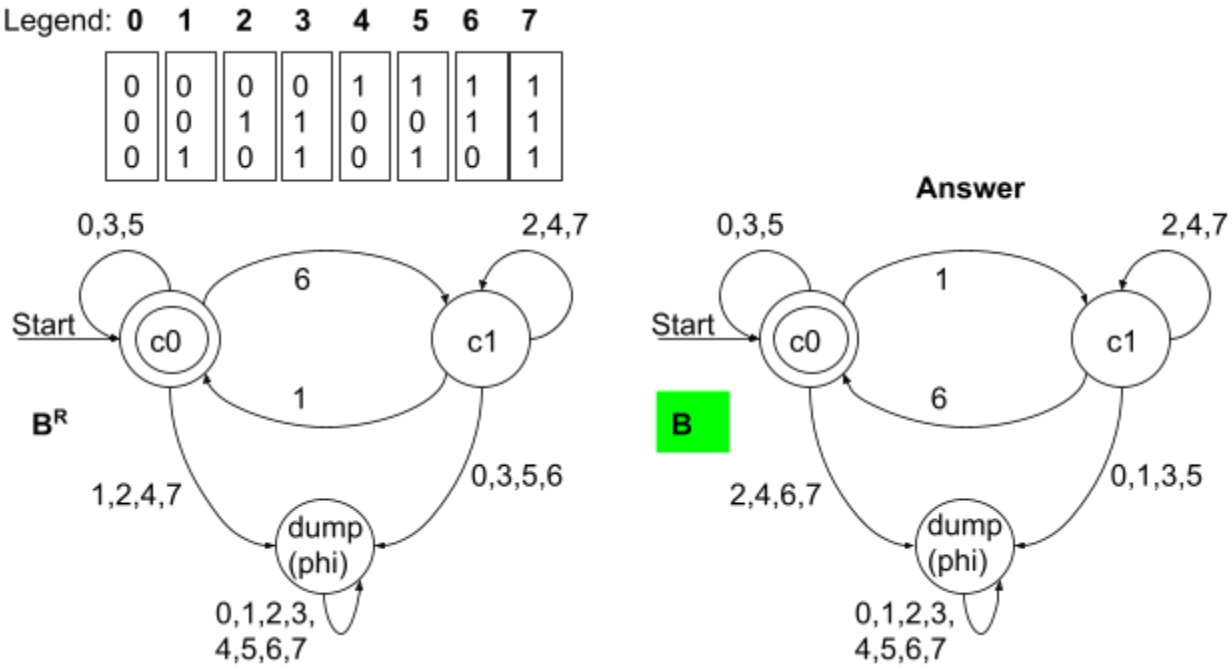
If there are multiple accept states in A, this switch of accept and start states will cause multiple start states in A^R. However, this can be resolved by creating a new start state, removing the status of the previous multiple start states, and instead having the new start state point to the previous start states with an input value of the empty string, ϵ. This would convert the DFA to an NFA, but it can be reverted to a DFA if need be using methods displayed in class.

Problem 3

Dump Logic: If you end up in the 'dump' state, there is no member of Σ that can make the addition possible.

	C0 (S & A STATE)	C1	DUMP
0 0 0 (0 in DFA)	C0	C1	DUMP
0 0 1 (1 in DFA)	DUMP (cannot get odd adding 2 evens)	C0	DUMP
0 1 0 (2 in DFA)	DUMP (cannot get even adding odd + even)	C1 (b/c odd+odd = even)	DUMP
0 1 1 (3 in DFA)	C0	C1 (b/c odd+odd = even)	DUMP
1 0 0 (4 in DFA)	DUMP (cannot get even adding odd + even)	C1 (b/c odd+odd = even)	DUMP
1 0 1 (5 in DFA)	C0	C1 (b/c odd+odd = even)	DUMP
1 1 0 (6 in DFA)	C1 (cause odd + odd = even)	C1 (b/c odd+odd = even)	DUMP
1 1 1 (7 inDFA)	DUMP (cannot add 2 odds and get odd)	DUMP	DUMP

A:

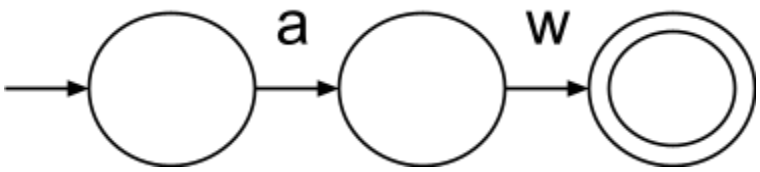

Legend: 0 1 2 3 4 5 6 7
0 0 0 0 1 1 1 1
0 0 1 0 1 0 1 1
0 1 0 1 0 1 0 1
Start
c0
c1
6
1
0,3,5
2,4,7
B^R
1,2,4,7
0,3,5,6
dump
(phi)
0,1,2,3,
4,5,6,7
Answer
Start
c0
c1
1
6
0,3,5
2,4,7
B
2,4,6,7
0,1,3,5
dump
(phi)
0,1,2,3,
4,5,6,7

Optional Problem

For any string σ, over alphabet Σ, we define the string SHIFT(σ) as follows: if $\sigma = aw$, $a \in \Sigma$, $w \in \Sigma^*$ then SHIFT(σ) = wa. For example, SHIFT(0111) = 1110, *and* SHIFT(10110) = 01101. Prove that if L is regular, then so is SHIFT(L) = {SHIFT(σ): $\sigma \in L$}.

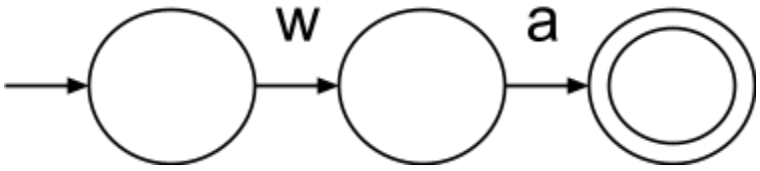
A: Since $a \in \Sigma$ and $w \in \Sigma^*$, we can abstract the input of a to simply be a, and the inputs (nonexistent or otherwise) of w to simply be w.

Thus we can represent σ = aw as a simple NFA:



If $\sigma = aw$ and SHIFT(σ) = wa, this is identical to reversing the string.

Thus we can represent SHIFT(σ) = wa as the reverse of the above NFA:



Since we are given that L is regular, and $\sigma \in L$, we can use the solution from **Problem 2** to show that {SHIFT(σ): $\sigma \in L$} is also regular. Therefore, if L is regular, then so is SHIFT(L) = {SHIFT(σ): $\sigma \in L$}.