Problem 1: The proof that the $\sqrt{2} \notin \mathbb{Q}$ is the seed for proving irrationality of other integer roots, whose proofs techniques seem to take advantage of two main techniques; we'll look at both.

Both techniques tread the path taken by the $\sqrt{2}$ proof before diverging. For the first, assume two irreducible co-prime integers a and b:

$$\sqrt{3} = \frac{a}{b} \tag{1}$$
$$3 = \frac{a^2}{b^2} \tag{2}$$
$$3b^2 = a^2 \tag{3}$$
$$\therefore 3 \mid a^2 \implies 3 \mid a \tag{4}$$
$$\therefore a = 3c \tag{5}$$
$$3b^2 = (3c)^2 \tag{6}$$
$$3b^2 = 9c^2 \tag{7}$$
$$b^2 = 3c^2 \tag{8}$$
$$\therefore 3 \mid b^2 \implies 3 \mid b \tag{9}$$
$$\perp(1) \tag{10}$$

(1) sets up the problem (for contradiction), (2) squares both sides, (3) multiplies both by b^2. (4) says that 3 divides a^2 which comes from (3). This also means that 3 divides a by recognizing that since a is irreducible, a^2 is necessarily a multiplication of prime factors that come from a. This means we can replace a with a 3 times some number c (5) and substitute this value in (3) to get (6) and (7). Algebra gives us a contradiction with (1) at (10) because we found that both a and b are both reducible. To prove the bonus question for all primes p, recycle the same proof but replace 3 and 3^2 with p and p^2 and the contradiction again arises.

The other technique goes to step (3) and then examines two cases. In the first, we assume b is even, so $3b^2$ and a^2 are also even, which leads to the same contradiction in the $\sqrt{2}$ proof. So case two chooses odd b, which means $3b^2$ and a^2 are odd. By some algebraic trickery, we can exchange each odd number with another number such that:

$$b = 2p + 1 \tag{4}$$
$$a = 2q + 1 \tag{5}$$
$$3(2p+1)(2p+1) = (2q+1)(2q+1) \qquad \text{from } (3,4,5) \tag{6}$$
$$3(4p^2 + 4p + 1) = 4q^2 + 4q + 1 \tag{7}$$
$$12p^2 + 12p + 3 = 4q^2 + 4q + 1 \tag{8}$$
$$12p^2 + 12p + 2 = 4q^2 + 4q \tag{9}$$
$$2(6p^2 + 6p + 1) = 2(2q^2 + 2q) \tag{10}$$
$$6p^2 + 6p + 1 = 2q^2 + 2q \tag{11}$$
$$\perp \tag{12}$$

Here the contradiction comes from (3, 4, 5, 11) where we can see that both sides have contradictory parity, with the left side odd and the right side even.

Problem 2: For the inductive proof, we start with a base case $|A| = 1$ then $|P(A)| = 2^1 = 2$. The elements of the power set of the singleton set are the empty set and the single element. We choose to start with $n = 1$ as per the prompt (assume $n \geq 1$ throughout) so we have the proof sketch:

(1) The set $A_1 = \{x\}$ and $\mathcal{P}(A_1) = \{x, \{\}\}$
(2) Show $|A_1| = 1$ and $|\mathcal{P}(A_1)| = 2^1 = 2$
(3) Assume $|A_n| = n \implies |\mathcal{P}(A_n)| = 2^n$
(4) Show $|A_{n+1}| = n + 1 \implies |\mathcal{P}(A_{n+1})| = 2^{n+1}$
(5) $|\mathcal{P}(A_{n+1})| = 2^n \cdot 2$
(6) $|\mathcal{P}(A_{n+1})| = |\mathcal{P}(A_n)| \cdot 2$
(7) $A_{n+1} = A_n \cup \{y \mid y \notin A_n\}$
(8) $|\mathcal{P}(A_{n+1})| = |\mathcal{P}(A_n \cup \{y\})|$

We need to show (5), that the cardinality of the power set is multiplied by two each time a new element is added to the set. Since the power set of A_n is the set of all sets in A_n, when we add a new element y to A_n we need to also add the union of that element with every subset already in the power set of A_n to get the power set A_{n+1}. Since the union of an element with the empty set (from A_n) is just the element itself, we can count one union for every subset already in A_n, doubling the amount of subsets. A way to visualize this is to think of all the 2^n subsets in A_n as nodes in a graph with zero edges. Suppose you added a new node that was connected by a single edge to every other node in the graph. You would have 2^n edges, representing the newly added subsets.

Problem 3: The average human head has approximately 100,000 hair follicles. Assume a higher maximum follicle number such that the amount of hairs on a single human head is bounded by the range [0, 200,000]. Meanwhile, the population of the United States is approximately 327 million people, each of whom possesses (for the sake of argument) one head. Suppose we found 200,000 people who miraculously each had one more hair than the previous, such that if we lined them up we would have one person at the beginning with 0 hairs, the second person with 1 hair, the third with 2 (and so on) and the person at the end with exactly 200,000 hairs, our upper bound (no person can have more hairs on their head). Next, suppose we added one more person to our sample size, such that we had 200,001 specimen. By the pigeon-hole principle (ie, there are more people in our total population size than hair follicles), the $200{,}001^{st}$ person would have to have a hair-count in the allowed range, meaning s/he would have the exact same amount of hairs as another person in the sample. Therefore, it is true that there exist two people in the United States with the same amount of hairs on their respective heads.

The proof can be extended to 100 people by the following reasoning. We can assign each person to one of 200,000 holes, corresponding to the amount of hairs on their head. Notice that once we have exceeded 200,000 people, we have necessarily found at least one match. In fact, each time we have a multiple of 200,000 people added to the holes, we would have at least one more match. If each person were assigned to the next hole (modulo 200,000), there would be around 1600 people in each hole (divide 327 million by 200 thousand), meaning that not only are there definitely 100 people with the same amount of hairs on their heads, but there are at least 1600 people with the exact same amount of hairs. The actual number of people with the exact same amount of hairs is likely much higher, since such an even distribution at the extremes is unlikely, and we would probably see a more normal distribution about the mean.

Problem 4: The way to prove that the cardinality of the set of binary strings of countably infinite length is uncountably infinite is to use Cantor's diagonalization method. First, we need to arrange a table of all the infinitely long binary strings in a table, such that each next binary string is arranged directly below the previous string. These strings can be arranged in any order, as long as each next string is different than all the strings that came before it. As an example table, we could have the following:

1	**1**	1	1	1	1	1	1	1	1	1	1	1	...
2	0	**0**	0	0	0	0	0	0	0	0	0	0	...
3	1	0	**1**	0	1	0	1	0	1	0	1	0	...
4	0	1	0	**1**	0	1	0	1	0	1	0	1	...
4	1	1	0	0	**1**	1	0	0	1	1	0	0	...
6	0	0	1	1	0	**0**	1	1	0	0	1	1	...
7	1	1	1	0	0	0	**1**	1	1	0	0	0	...
8	0	0	0	1	1	1	0	**0**	0	1	1	1	...
9	1	0	0	1	0	0	1	0	**0**	1	0	0	...
10	0	1	1	0	1	1	0	0	1	**1**	0	0	...
11	1	0	1	1	1	0	1	0	1	1	**1**	0	...
12	0	1	0	0	0	1	0	1	0	0	0	**1**	...

Each string in the table is infinitely long (denoted by the ellipses) and the vertical height is also infinitely tall. Notice the numbers on the left, providing a one-to-one correspondence between the natural positive integers and the set of binary strings. This correspondence shows that the set of binary strings is countably infinite. Of course, the dagger is on the diagonal. Cantor's method tells us that we can create another binary string by taking the bit in each of the diagonal positions from top left to bottom right (in bold and underlined) and flip the bit, thus guaranteeing that this string differs from every other string in the infinite set of strings. So in the example, our bit string would be 010001011000... The objection one often makes when initially looking at this proof is to claim that the diagonal string would eventually show up in an infinite set of binary strings, and indeed, we could append our set with this newly discovered string. But then we could just flip the bit on the diagonal again and we'd have a new string. This shows that there exist elements outside of the infinite set itself, and that no matter how we arrange our set to include every string, strings of infinite length will still not be described by, or be members of, the set of infinitely long binary strings. Therefore, the cardinality of this set is uncountable.

Problem 5: We will show that the triple prime conjecture is false by observing several repeating patterns and properties that are always true of candidate members of this set which exclude them from the set. Indeed, not only is this set not infinite, but there is only 1 element of this set of triple-primes. It helps to arrange all the candidate triple primes in a vertical table like the one on the following page. From the table, we can observe a few patterns. First, there are 5 different 'types' of candidate elements. After we add 2 to any of these numbers five times, we've added 10 and we repeat the same pattern, incrementing the tens digit. So each set of 5 three-tuples is ten more than the previous.

1	3	5				
	3	5	7			
		5	7	9		
			7	9	11	
				9	11	13
11	13	**15**				
	13	**15**	17			
		15	17	19		
			17	19	21	
				19	21	23
21	23	**25**				
	23	**25**	27			
		25	27	29		
			27	29	31	
				29	31	33
31	33	**35**				
	33	**35**	37			
		35	37	39		
			37	39	41	
				39	41	43
41	43	**45**				
	43	**45**	47			
		45	47	49		
			47	49	51	
				49	51	53
51	53	**55**				
	53	**55**	57			
		55	57	59		
			57	59	61	
				59	61	63
61	63	**65**				
	63	**65**	67			
		65	67	69		
			67	69	71	
				69	71	63
71	73	**75**				
	73	**75**	77			
		75	77	79		
			77	79	81	
				79	81	83
81	83	**85**				
	83	**85**	87			
		85	87	89		
			87	89	91	
				89	91	93
91	93	**95**				
	93	**95**	97			
		95	97	99		

For the following elimination discussion, we are not including the first two triple-primes $\{1, 3, 5\}, \{3, 5, 7\}$ (the first is not a triple-prime because it contains 1 and the second is the only triple-prime) or the third $\{5, 7, 9\}$ (which fails because it has a 9 in it).

Begin by eliminating multiples of 5. After the first 5 three-tuples, each successive set of 5 three-tuples begins with three three-tuples which include one element whose last digit is 5. We can eliminate all 3 of these tuples for the entire (infinite) table because each contains an element divisible by 5 and hence is of course not prime. Since we are adding 10 each time we wrap to a new set, we will never fail to find a number which has a 5 as a last digit in the first three tuples. Eliminating, we get:

7	**9**	11	
	9	11	13
17	19	21	
	19	21	23
27	29	31	
	29	31	33
37	39	41	
	39	41	43
47	49	51	
	49	51	53
57	59	61	
	59	61	63
67	69	71	
	69	71	63
77	79	81	
	79	81	83
87	89	91	
	89	91	93

The next observation we can make is that, of the remaining two types, the first contains numbers whose *last digits* are [7, 9, 1] respectively and the second type contains numbers with [9, 1, 3] as the last digit of each. Since we add 10 each time we wrap the set, this will be the case to infinity.

For the next step, we can show that every three-tuple contains exactly one composite number in the following way. Notice that 9 is composite in the first line. Eliminate the first line and the one below it, because the 9s stack. For the proof, we create the resulting table with line numbers below.

1	17	19	21	
2		19	21	23
3	27	29	31	
4		29	31	33
5	37	3**9**	41	
6		3**9**	41	43
7	47	49	51	
8		49	51	53
9	57	59	61	
10		59	61	63
11	67	6**9**	71	
12		6**9**	71	63
13	77	79	81	
14		79	81	83
15	87	89	91	
16		89	91	93

Next, we can see that every line numbered $6n$ or $6n-1\ \forall n \geq 1$ can be eliminated because it contains number whose last digit is divisible by 3. This can be shown mathematically. First, observe that line 6 and 5 (6-1) each contain 39. Since 39 can be broken up as $39 = 30 + 9$, we know it is divisible by 3, since both 30 and 9 are divisible by 3. Every 6^{th} set of 2 three-tuples contains a number $30n + 9$ because we add 30 each time, maintaining divisibility by 3.

1	17	19	2**1**	
2		19	2**1**	23
3	27	29	31	
4		29	31	33
7	47	49	5**1**	
8		49	5**1**	53
9	57	59	61	
10		59	61	63
13	77	79	8**1**	
14		79	8**1**	83
15	87	89	91	
16		89	91	93

We can continue in this fashion, by noting that every line numbered $(6n-5)$ and $(6n-4)$ contains a number which ends in a 1 and is divisible by 3, since $21+30n$ is likewise always divisible by 3. This leaves us with the last two types below.

3	2**7**	29	31	
4		29	31	3**3**
9	5**7**	59	61	
10		59	61	6**3**
15	8**7**	89	91	
16		89	91	9**3**

Continue another time with lines numbered $(6n-3)$ because each contains a number that is $27+30n$. Lines numbered $(6n-2)$ all contain a number $33+30n$. Algebraically, we have now eliminated every single candidate element because we have eliminated every candidate element whose line number corresponds to a multiple 6 or every number between each multiple of 6.

Problem 6: We can use Cantor's diagonal method to show that the interval (0, 1) is uncountably infinite. In addition, there exist certain functions that map every member of $\mathbb{R}$ to the interval (0,1). To map the real numbers to the interval, we can choose the continuous, infinite on-the-interval and asymptotic-at-the-bounds piecewise function $f(x) =$

$$\begin{cases} -tan(\frac{\pi}{2} - \frac{\pi x}{2}) + 2 & x < 1/2 \\ tan(\frac{\pi x}{2}) & x \geq 1/2 \end{cases}$$

Next we need to find a mapping from the real numbers to the interval; we can choose the function $g(x)$ which is the inverse of the first function divided by two:

$$\frac{(tan^{-1}(x-2) + \frac{\pi}{2})}{\pi}$$

With these two function we can show that a bijection exists between the reals and the interval.

Problem 7: From the lone axiom MI we can only get two strings (theorems) using rules 1 and 2:

$$MI[1] - - > MIU \tag{1}$$
$$MI[2] - - > MII \tag{2}$$

Using rule 1 and then repeating rule 2 on MII we can derive $M(IIU)^n$. Using rule 2 on MIU, we can derive $M(IU)^n$:

$$MII[1]n - - > M + (IIU) * 2^n \tag{3}$$
$$MIU[2]n - - > M + (IU) * 2^n \tag{4}$$

We can ignore all strings of the form on the right side (3)(4) because they are not useful in reducing the string to MU. That is, it isn't possible to get anything else out of these strings using the rules. This is because for both, the string ends in IIU or IU and once we have a single U ending a string, the rules don't allow us to derive anything but duplicating the part after M because there are no III or UU substrings.

Next, we can focus on strings derived from (2) MII. Repeatedly applying rule 2:

$$MII[2]n - - > M + (II) * 2^n \tag{5}$$

From this we can continually apply rule 3 and rule 4 (if we have UU) to the end of the string to get back to (1), which doesn't help us. This is because, for every III that we remove from the end of (5), we get

$$M + 2^n(II) - 3(III) \tag{6}$$

Mathematically, we will always end up with the axiom MIU or the string MII doing this because $2^n - 3k \ \forall k \geq 1$ will always either be odd or even matching the parity of k. If n is odd, we can reduce to MI and if n is even, we can reduce to MII. We can prove this inductively:

$$\begin{aligned}
& & 2^1 &= 2 \\
& \therefore & (3k &\nmid 2^1) \\
\textit{Assume} & & (3k &\nmid 2^n) \\
& \therefore \textit{case}\,1: & 2^n &= 3k+1 \\
& ||\textit{case}\,2: & 2^n &= 3k+3 \\
& \therefore & 2^{n+1} &= 2(3k+1) = 6k+2 \\
& || & 2^{n+1} &= 2(3k+2) = 6k+4 = 6k+3+1
\end{aligned}$$

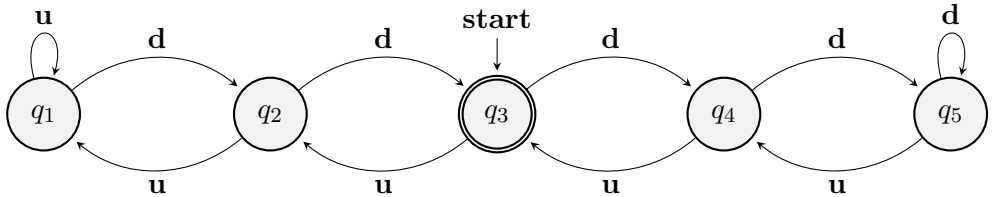
This lemma simply shows that $3k$ can never divide 2^n and therefore the equation $2^n - 3k = 0$ is unsolvable. Therefore we will never end up with the string MU, we will always end up with either MIU or MII. But we're not done.

We can also have the string MUI by applying [3] to the front of $MIIII$ from (5). Here again we are stuck either appending IU or doubling UI.

Finally, we can also have an infinite amount of strings of the form $M+I^+ + \{U^*, I^*\}$ by applying [3] to (6) in various ways to reach strings which have any number of U between some number of I. But notice that we always have an I at the beginning of these strings, and that we have already shown we can't mathematically to get from MI to (5) to MU so the same lemma applies in this situation as well. Namely, we can always reduce down to MIU or MII, but because we can't get the right multiples of 3 and 2, we'll never get MU.

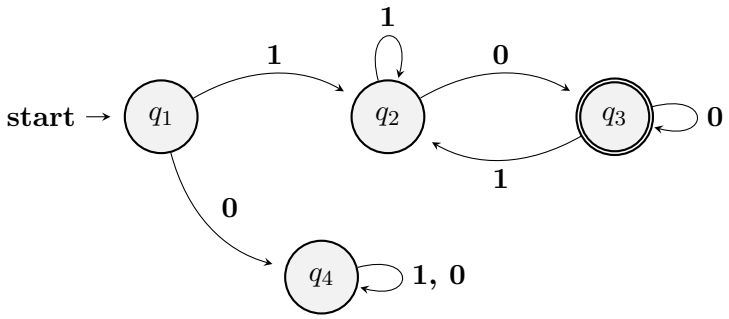
Questions from the Sipser Text:

1.3

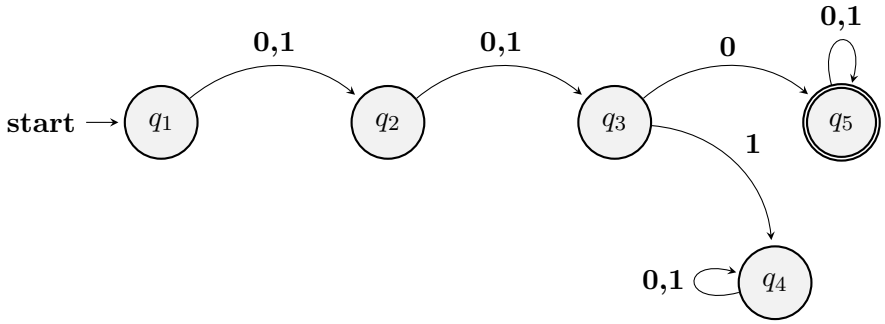


1.6

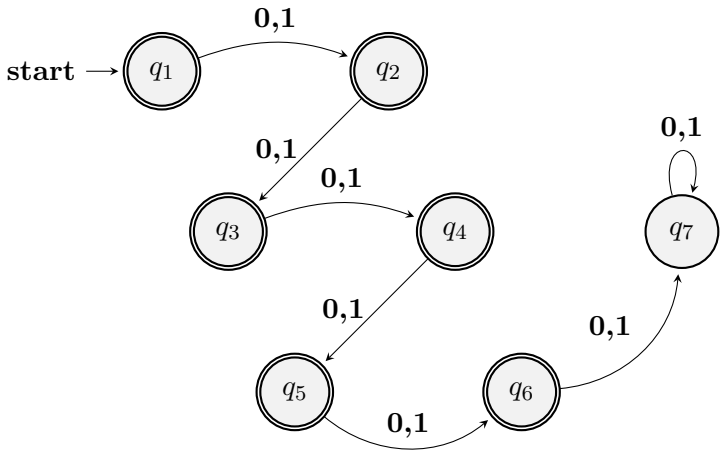
a. Begin with 1, end with a 0



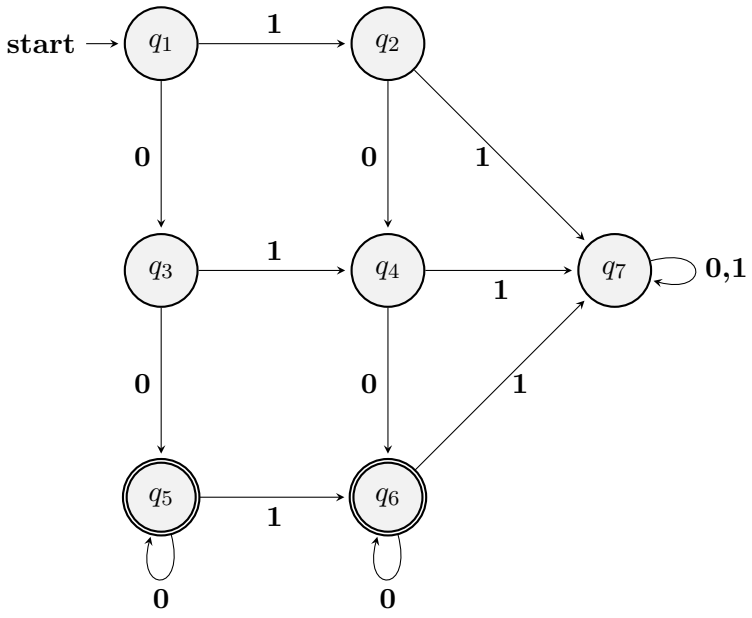
d. Length at least 3, third symbol a 0



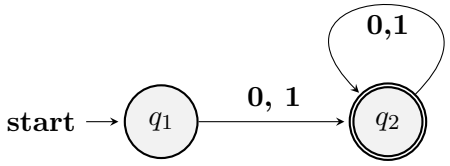
g. Length at most 5



j. At least two 0s, at most one 1



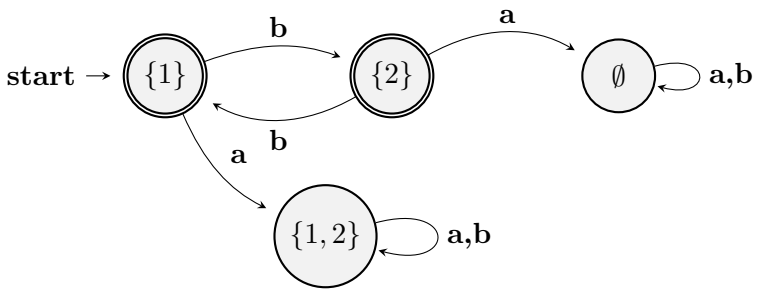
n. All strings except empty



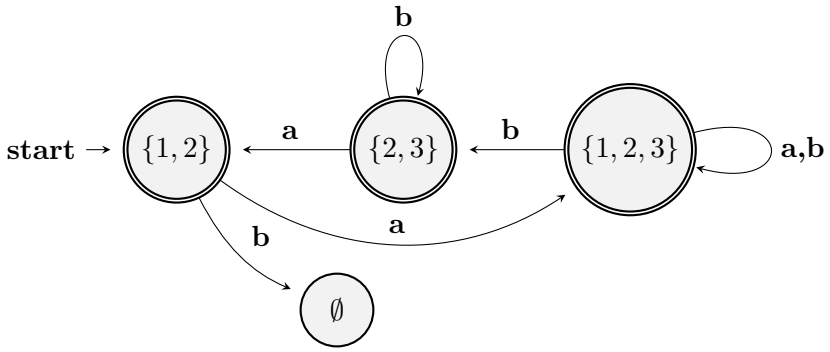
1.11 In order to turn a given NFA into an NFA with only one accept state, simply create a new accept state q_{accept} and add ϵ edges to q_{accept} from all existing accept states and revoke their accepting nature. Now, q_{accept} is the only accepting state in the NFA, but the NFA is the same as the one it was derived from, since adding epsilon edges in this way will not change the strings that either NFA accepts.

1.16

a. After removing unreachable states, we have the following DFA:

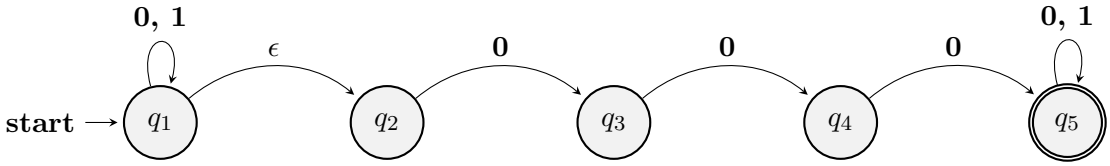


b. After removing unreachable states, we have the following DFA:

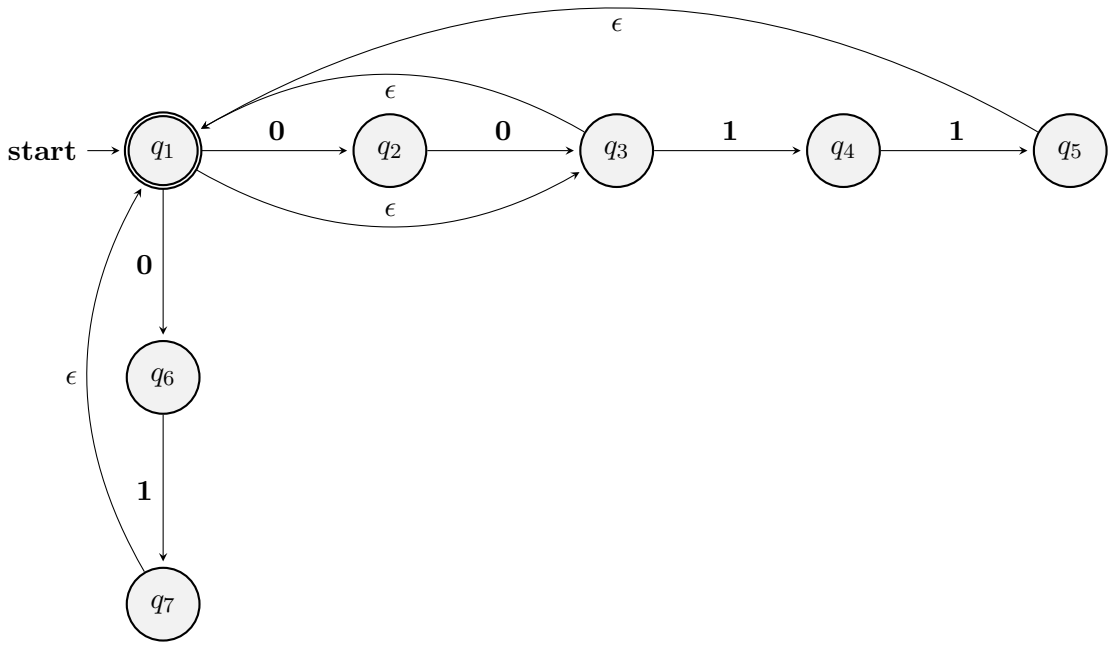


1.19

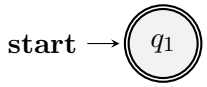
a. $(0 \cup 1)^*000(0 \cup 1)^*$



b. $(((00)^*(11)) \cup 01)^*$



c. $\emptyset^*$



1.20 Strings following 'are' are accepted and strings following 'not' are not accepted.
a. are: ab, aab not: ba, baa
b. are: ab, $abab$ not: b, ba
c. are: a, b not: ab, aba
d. are: aaa, $aaaaaa$ not: b, bb
e. are: aba, $aabaa$ not: bbb, bab
f. are: aba, bab not: a, b
g. are: b, ab not: ba, bb
h. are: a, bb not: b, ϵ

1.29
a. $A_1 = \{0^n 1^n 2^n \mid n \geq 0\}$

Choose $s = 0^p 1^p 2^p$. Using the PL, we can see a few possible cases for the string xyz. Since $|xy| \leq p$, we can see that xy must be a string of zeros. Then, $xyyz$ would have more 0s than 1s or 2s. If xy consisted of all 1s, we could have more 1s than 0s or 2s. The same argument applies if y consists of all 2s.

The only other option has y getting the order of the 0s, 1s, and 2s, jumbled up. This would happen if y consisted of more than one alphabet symbol, because in this case y^2 would have a 0 following a 1 or a 2 or both, a 1 following a 2, etc.

b. $A_2 = \{www \mid w \in \{a, b\}^*\}$

Choose $s = ab^p ab^p ab^p$. The PL tells us that s must be able to be broken up into xyz. Since $|xy| \leq p$, we know that xy consists of $ab^n | n \leq p-1$ and z consists of $b^{p-n}ab^p ab^p$. The PL also states that each $xy^i z \in A_2$. Next we can break xy up into $x = ab^m$, $y = b^{n-m}$. Then, for the string $xyyz$, we would have $ab^m b^{2(n-m)} b^{p-n} ab^p ab^p = ab^{(2n-m-n+p)} ab^p ab^p = ab^{(n-m)+p} ab^p ab^p$. We know that $|y| = b^{n-m} > 0$. This means that for sufficiently large i, y^i has more than p bs which means the string is not in the language. Another possibility, though, is if $|x| = 0$. In this case, we would have $y = ab^n$ and we would still wind up with string $ab^{n+p} ab^p ab^p$ with more than p bs in the first w.

c. $A_3 = \{a^{2^n} | n \geq 0\}$.

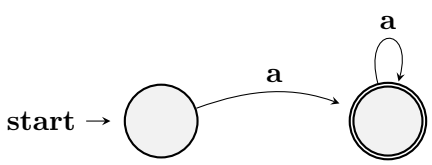
Choose $s = a^{2^p}$. Then, xyz would consist of a string a^n in xy where $|xy|, n \leq p$ as well as a string of $a^{2^p - n}$ in z. Since $|xy| \leq p < 2^p$, $|xy| < 2^p$ and $|xyy| \leq y + y < 2^p + 2^p$. But $|xyyz| > 2^p$ since the PL states that $|y| > 0$. This allows strings whose length lies in between powers of 2, since $2^p > |xyyz| < 2^p + 2^p$. Answer admittedly ripped right out the book I can't figure out another way.

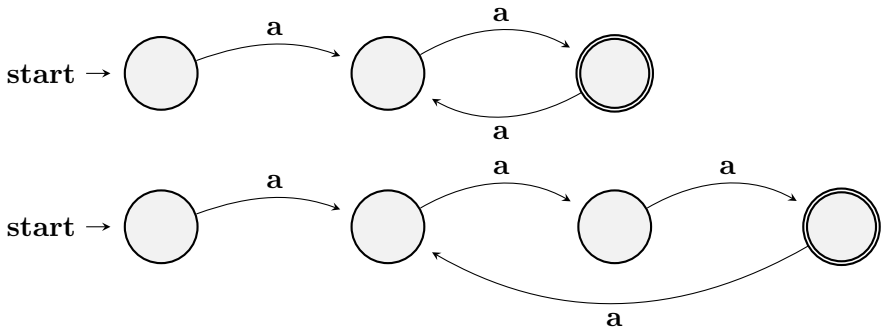
1.30

The simplest DFA for the language consists of a start state which has a self loop on 0 and a transition to a single accept state on 1 with a self loop on 1. The reason the proof doesn't work is because the language doesn't require we have the same amount of 0s and 1s. So if we choose $s = 0^p 1^p$, we have xyz such that $|xy| \leq p$, $|y| > 0$. Going through the same cases for the proof in 1.73, we can have y consists of **1**. only 0s, **2**. only 1s, or **3**. a combination of 1s and 0s. For **1**, $xyyz$ will have more 0s than 1s. Not a problem! For **2**, we will have more 1s than 0s. Again, not a problem. Finally for **3**, we just apply rules 2 and 3 of the PL, which say that $|xy| \leq p$, $|y| > 0$, so we actually can't have y be a mix of 0s and 1s, since $|0^p| = p$ so we would violate the rules to include a 1. Therefore, the PL applies to s and the language is regular. Hence, the proof is incorrect.

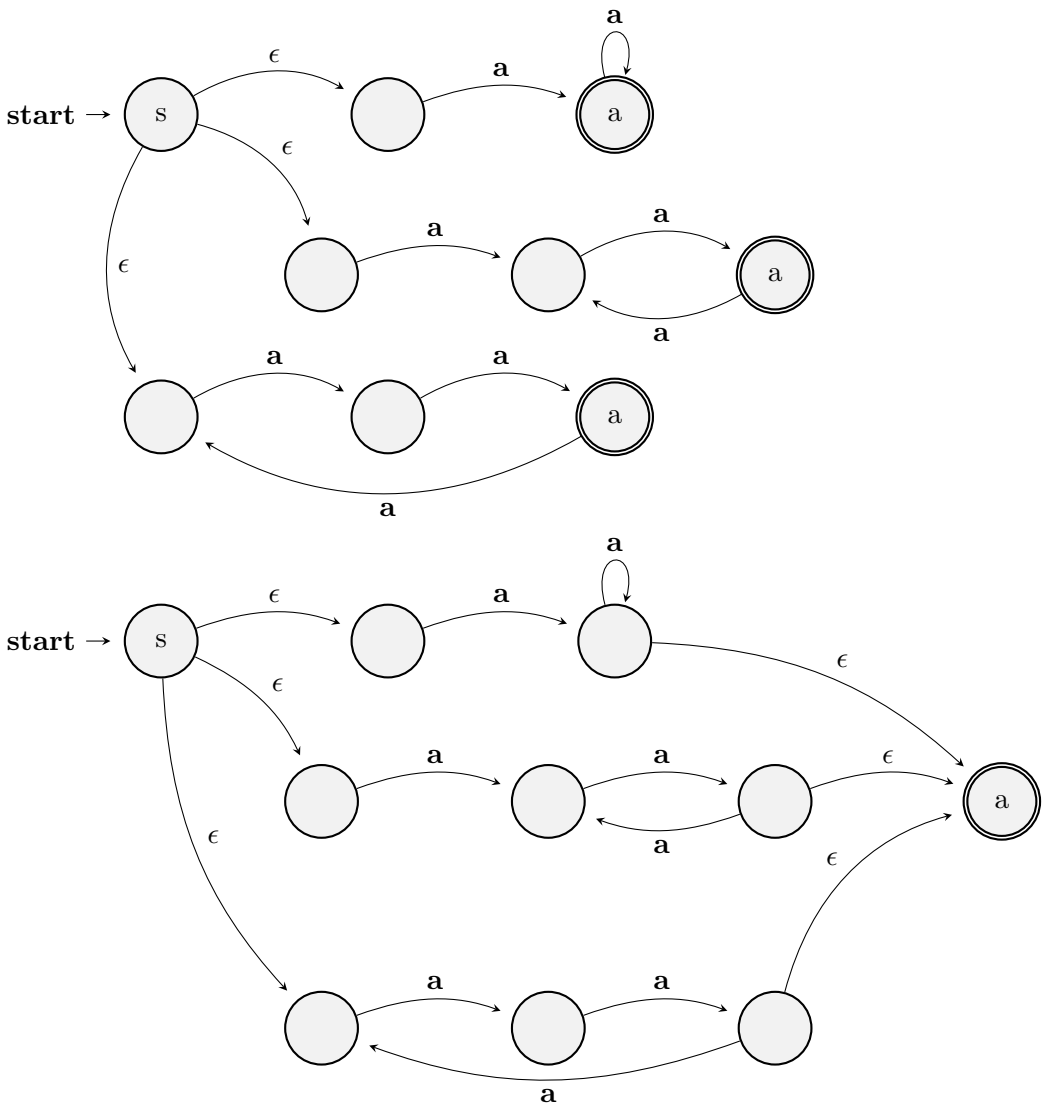
1.36

The problem suggests a language B_n which contains all strings whose length is a multiple of n concatenations of the string a. We know that if a language is indeed regular, it can also be described by an equivalent DFA. For the following, we have $\{n * k \mid n, k \in \mathbb{Z}^+\}$ (This means we can't have a string of length zero). To show that B_n is regular, we first construct the following DFAs for $n = 1, 2, 3$ respectively:

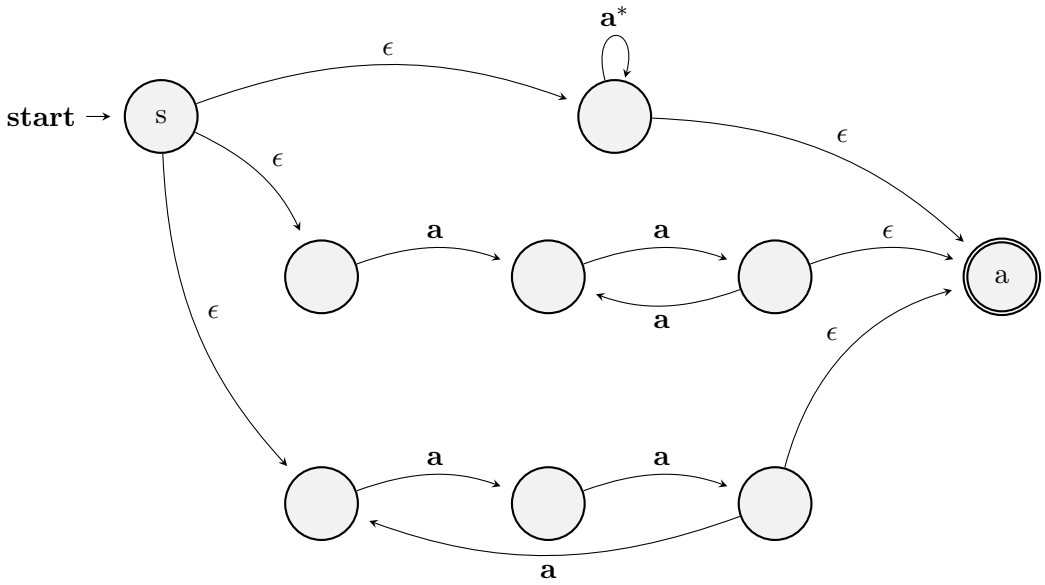


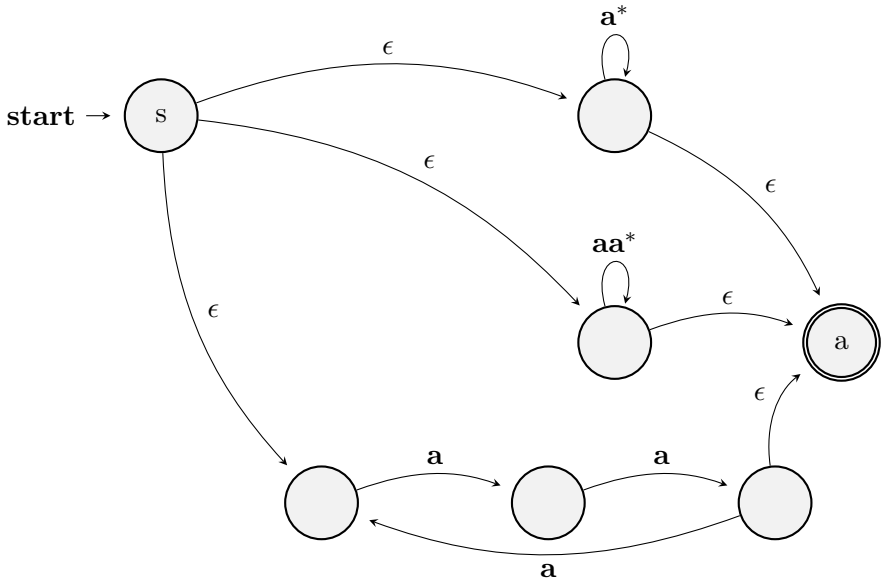


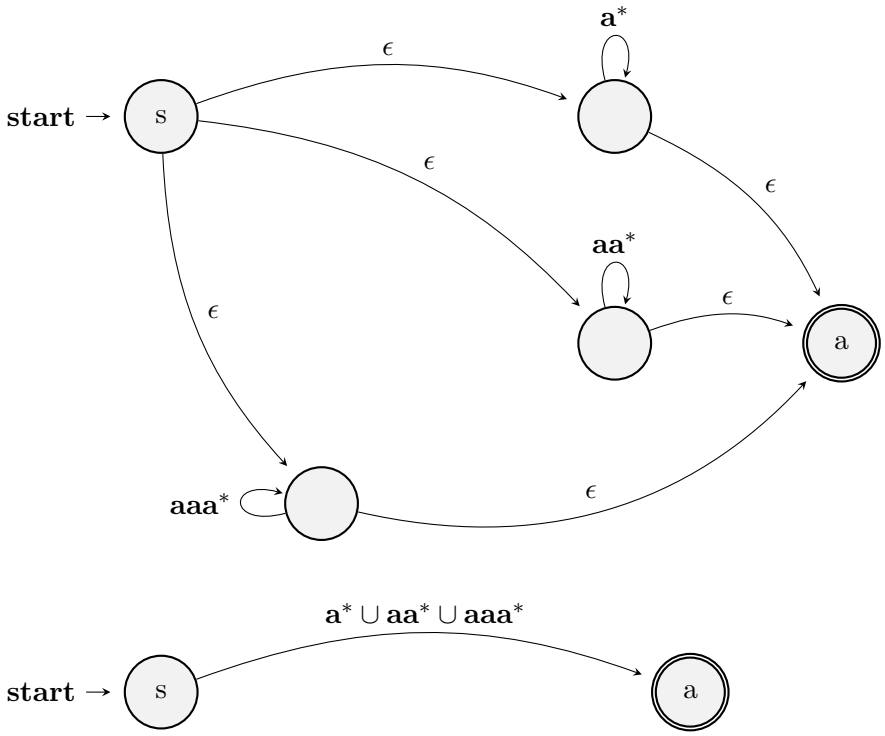
We can go on in this way forever, constructing ever-longer DFAs that accept ever-larger k-multiples for each next n. From this observation, we can construct a GNFA from the union of all the above DFAs which will allow us to include all the DFAs which are equivalent to B_n for some n. For example, from the first three DFAs above, we can union them, first adding a new start state, then a new accept state from all the original accept states:



Now we begin the process of ripping out states and recombining:







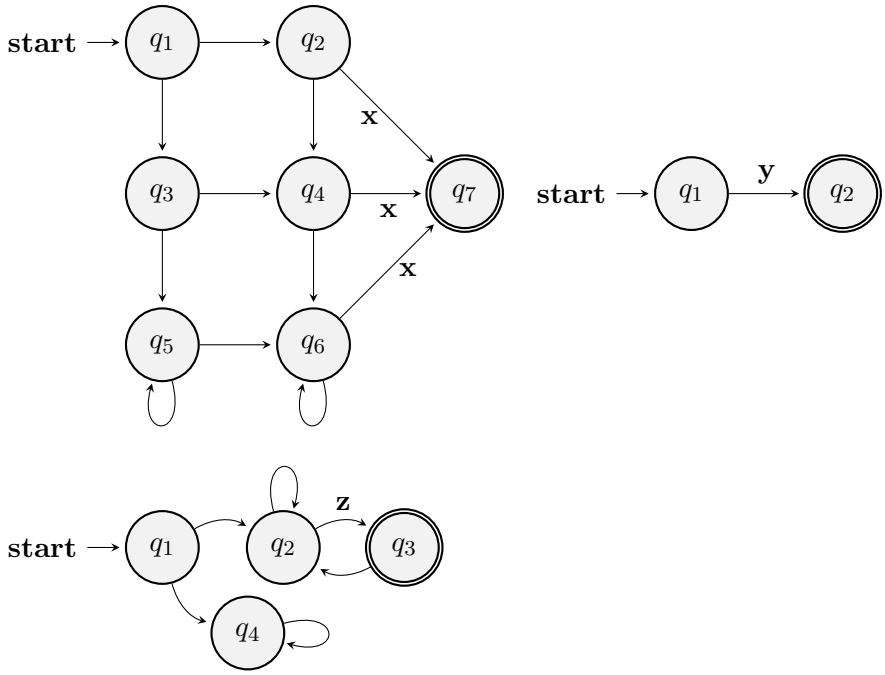
From this we can generalize (further) the GNFA which accepts B_n by doing this forever, so the arrow from start to accept would be $\mathbf{a}^* \cup \mathbf{aa}^* \cup \mathbf{aaa}^* \cup ... \cup \mathbf{a}^k$. Therefore, B_n is regular.

1.38

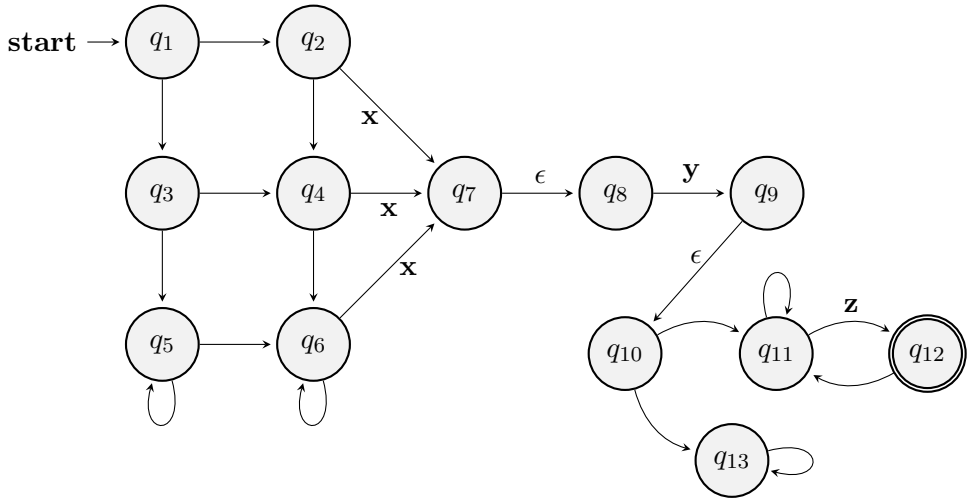
One way to do this is to convert the **all**-NFA into a DFA, and then, instead of just making all states containing a state which is an accept state an accept state, also do the following: for every accept state, make each member of that accept state's set of states an accept state. So, if we refer to figure 1.44 in the Sipser text, you'd still have the same diagram, but the set of accept states would also include (for this example) the states with $\{2\}$, $\{3\}$ and the state with $\{2, 3\}$. Since we know that a language is regular if we can construct some FSM which will accept it, our work is done.

1.43

The *DROPOUT* operation is similar to (but not the same as) the reverse of a concatenation operation. Theorem 1.47 states that the class of regular languages is closed under the concatenation operation. To see why it is also closed under *DROPOUT*, first consider the following picture. For $\{x, z \in \Sigma^*\}$, suppose we have 3 NFAs. The first, N_1, ends in an accept state after reading any string x and the third, N_3, after reading any string z. The second NFA N_2 has a single start state with an arrow to another state on reading input y (note that below x and z are strings that lead to accept states, whereas y is just a character).



We can concatenate these three NFAs using the construction in figure 1.48:



From here it is easy to see that we can use the dropout operation on N_2 by ripping its states out and drawing an epsilon arrow from q_7 to q_{10}. Formally,

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize x
$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize y
$N_3 = (Q_3, \Sigma, \delta_3, q_3, F_3)$ recognize z

Construct $N = (Q, \Sigma, \delta, q, F)$ to recognize $N_1 \circ N_2 \circ N_3$

1. $Q = Q_1 \cup Q_2 \cup Q_3$
2. The start state is the start state of N_1

3. The accept state is the accept state of N_3
4. Define δ such that all transitions are the same except those from accept states of N_1 and N_2, which now have epsilon transitions to start states of N_2 and N_3, respectively.
5. F is the set of accept states for N_3
6. Remove the states of N_2.
7. Add transitions from the accept states of N_1 to the start state of N_3

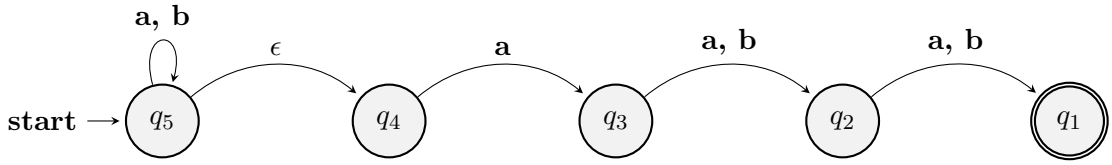
The NFA containing y could contain many states, and the proof would still work. Therefore, regular languages are closed under $DROPOUT$.

1.45

If A is regular, then there exists some DFA which describes the strings that A accepts. If $wx \in A$ and $x \in B$, then there exists some string(s) $x \in A \cup B$ which, once concatenated onto $w \in A$ lead to some accept state in A. We can construct a new language $A \setminus B$ by finding the states in A which we end up in after reading input w. For these states, some of them will lead to an accept state on x. Make these the new accept states and make the states that were accept states from these states on x into non-accepting states.

1.60

The following NFA demonstrates one example of how these C_k NFAs work in general. Since we know that there need to be $k+1$ states in total for any k, we can count the states as follows. The first state is the $(k+1)^{th}$ state, the second state following the epsilon arrow, is the k^{th} state. This will be the same for any k. Then, there will be $k-1$ states following the second state, leading to a total of $k+1$ states. For example, below is the NFA recognizing C_4. Notice that there is no transition from the last state, meaning any string leading to state q_0 must end in that state. The first state accepts Σ^*. Then, at any point, nondeterministically peel off to state q_3 upon reading an a. after reading the a, read Σ^{k-1} symbols. Of course, if there are more or less, the computation dies. Extend this to any k by adding more states after state q_k until you have $k-1$ states.



Formally, we can define N as follows:

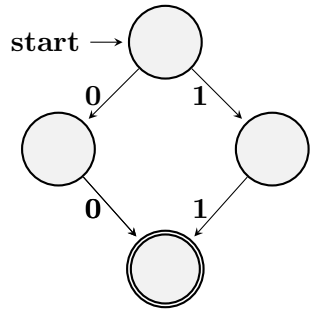
Let $N = (Q, \Sigma, \delta, q, F)$ recognize C_k

1. The start state is the start state of q_{k+1}
2. Following the start state, there are an additional k states
3. The accept state is q_1
4. Define δ such that
$\delta(q_{k+1}, a, b) = q_k$,
$\delta(q_{k+1}, \epsilon) = q_k$,
$\delta(q_k, a) = q_{k-1}$

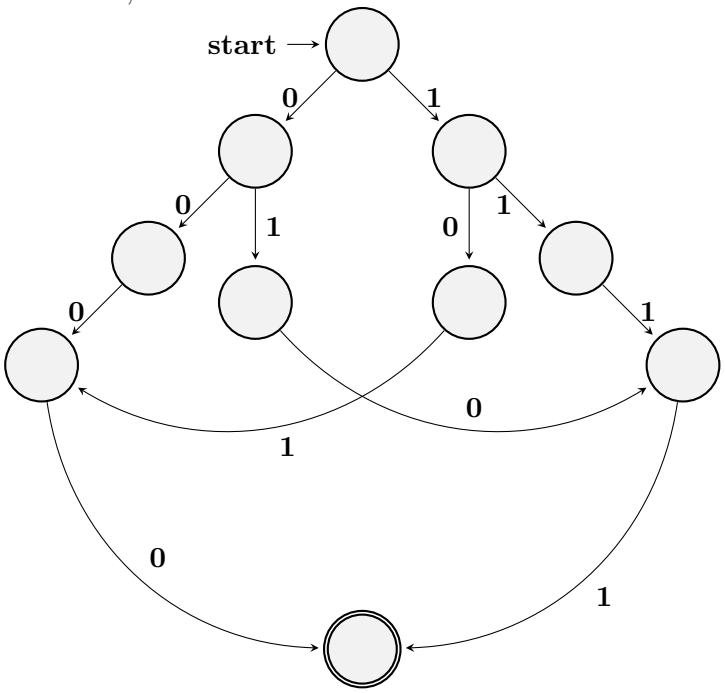
$\delta(q_{k-n}, a, b) = q_{k-(n+1)}$, where $n \leq (k-1)$

1.69

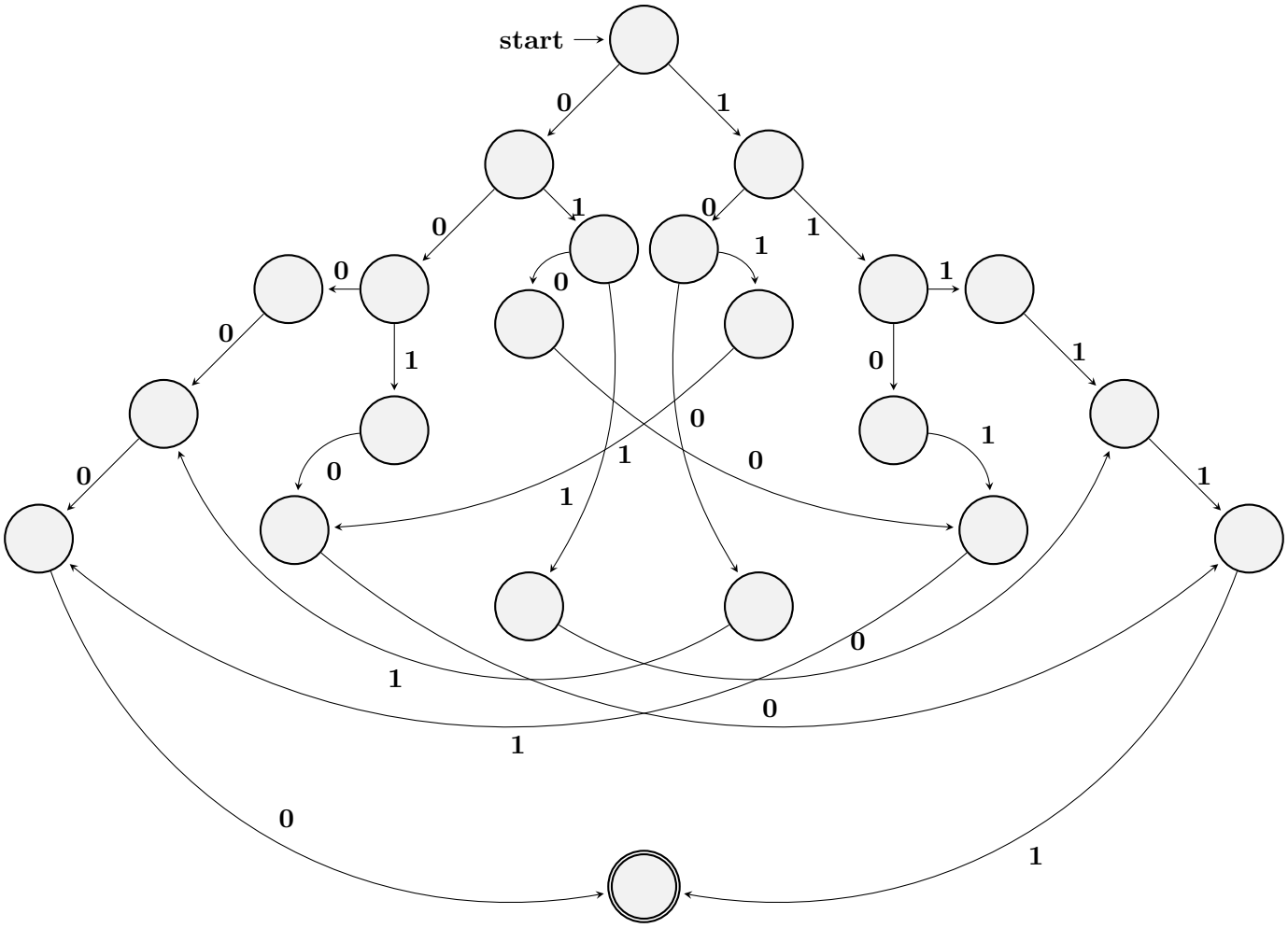
a. For the following, the dead state is omitted for clarity, but each state that doesn't have an arrow for 1 or 0 would necessarily go there. The question asks us to show that no DFA will recognize WW_k with fewer than 2^k states, we'll start with a diagram. Suppose that $k = 1$. We could make the DFA that accepts WW_k as follows, by recognizing that, with a binary alphabet, we can only have the following strings: $\{0, 1\}$. The DFA has a minimum of 5 states (with the dead state).



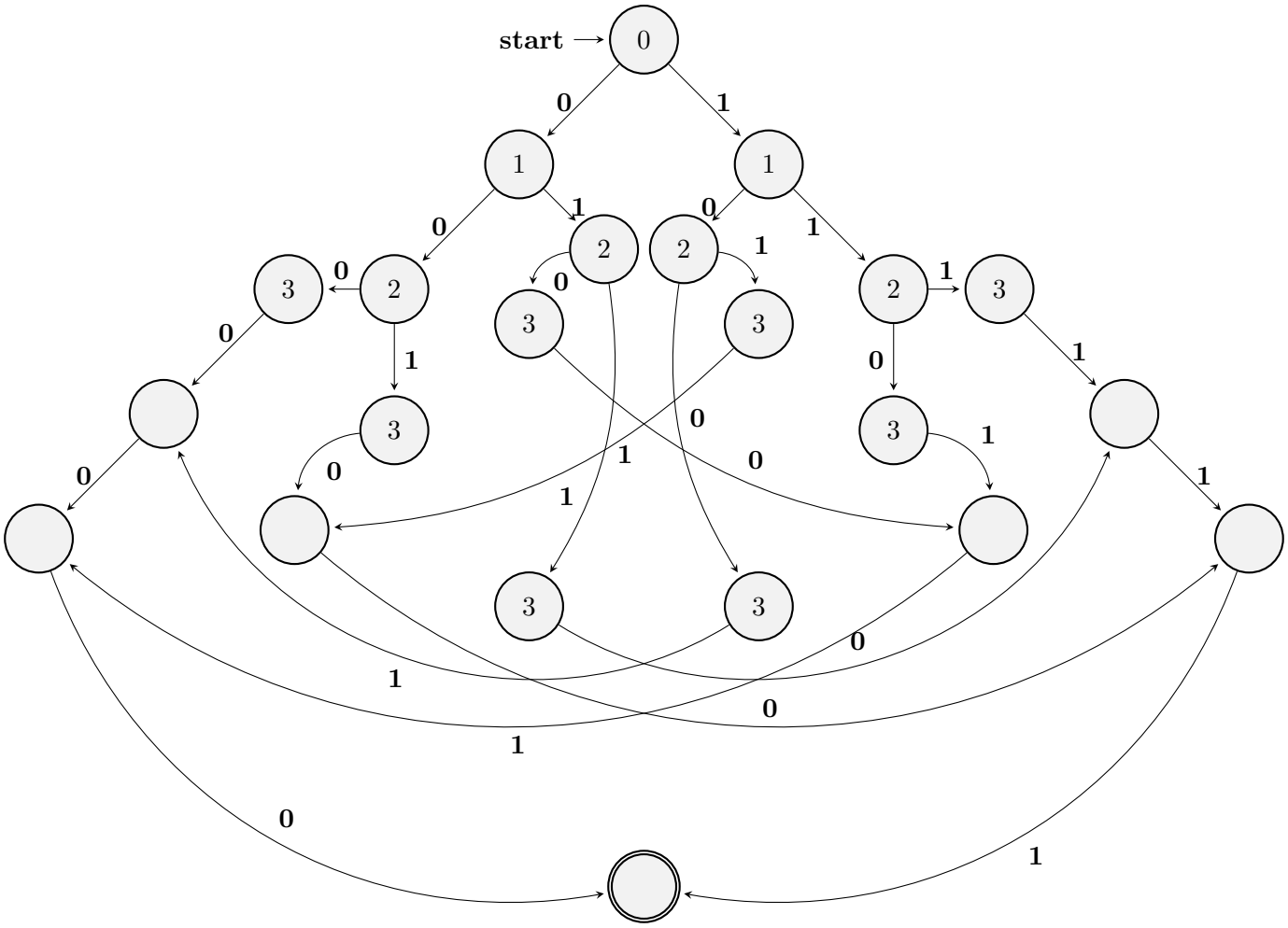
For $k = 2$, we can construct:



Continuing in this fashion, we can continue to increment k by adding more states. These get very complicated very quickly. Here's $k = 3$, (if you can follow the transitions!):



To see that the DFA will always have more than 2^k states, notice that for every k, we have to keep track of at least the first w symbols. After the first w symbols, we can reuse some states because they share a substring at the end. But for the first w in ww there are 2^k states needed for *each* k, because we add 2^k new states for each of the possible binary strings of length k. In addition, we must always have an accept state, a start state, and a dead state. Here is the same diagram with the k^{th} level labeling each state that must be added each time k increases:



b. For the NFA which recognizes the complement of WW_k, we have can accept any string that doesn't have length $2k$ (in the upper half of the diagram) and then we can accept any even-length string which doesn't contain a duplicate. Below is the NFA for k=3: