CMIMC Power Round 2016

CMIMC Staff

April 2, 2016

Abstract

The following is the Power Round for this year. Thanks to everyone, especially Course 15-251, for helping out with advice and problems and what not yayayay! (Ok idk what else to put here, darn)

1 Introduction

When working with computation on an abstract level, it is helpful to develop mathematical models for machines. There are several mathematical models which work, but they all have similar characteristics:

- They must be simple to understand.
- They must encompass many different computational tasks (though not necessarily all).
- They must be hardware independent. (Theoretical computer scientists don't really care about technical limitations; they instead care about which types of problems *can* be solved by a computer.)

One such model of computation is known as a *Deterministic Finite Automaton* (DFA). DFA's are simplistic models of computation that still have deep theories and implications for computational theories in the real world. In this power round, we will explore the underlying theory behind these mysterious models, solving interesting problems and proving fundamental results in automata theory along the way.

2 Introduction to Languages

Before we begin, we introduce the notation we shall be using to designate inputs to DFAs. This section is mainly just definitions - manipulating languages will not come until later sections.

Definition 2.1. *An* alphabet *is any finite, non-empty set of symbols. We often denote an alphabet by Σ. Examples of alphabets are $\Sigma_1 = \{0, 1\}$ and $\Sigma_2 = \{a, b, c\}$.*

Definition 2.2. *A* string *is any finite sequence of elements from an alphabet. For any string x, $|x|$ denotes the number of characters in the string. The empty string ε represents the empty sequence.*

Definition 2.3. *For an alphabet Σ, the notation Σ^* denotes the set of all strings over Σ. For example, $\{0, 1\}^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 100, \dots\}$.*

Definition 2.4. *A* language *over an alphabet Σ is any subset of Σ^*.*

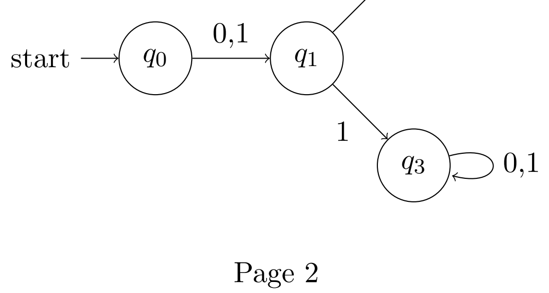
Now that we have some basic notation, it's time to move on to the meat of this power round.

3 DFAs

Unlike most other sections of this round, we skip the flavortext and go straight to a definition.

Definition 3.1. *A* deterministic finite automaton *(DFA) is a mechanism that takes in a string as an input and either accepts or rejects it. This is analogous to, say, a boolean function which takes in a string and outputs either* `true` *(i.e. accepts the string) or* `false` *(i.e. rejects it).*

When drawn, DFAs look like a bunch of arrows and circles. Here is an example of a DFA:

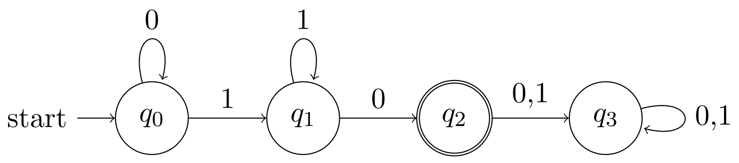


Each circle represents a *state*. A state with two circles is an *accepting state*, which means the DFA will accept if the string ends up there. Formally, a DFA over an alphabet Σ is defined as a finite set of states (with one start state and some of which can be accepting states), each with an arrow pointing to another state for each element of Σ. For a DFA, the language that it *admits* is the language of all strings over Σ that accept when inputted into the DFA.

To read a DFA, start with the string in the state indicated by the "start" arrow. Look at the first letter in the string, then follow the arrow corresponding to that letter to the next state. From that state, follow the arrow corresponding to the second letter of the string, and so on until the string has no more letters. At that point, the DFA accepts if you are in an accepting state and rejects otherwise.

For example, let's input the string 1011 into the above DFA. It starts at q_0. The first letter is 1, so move to q_1. The second letter is 0 so move to q_2. The third letter is 1 so move to q_2. The fourth letter is 1, so move to q_2. There are no more letters left, and we are in an accepting state, so this DFA will accept the string 1011.

1. Consider the following DFA:



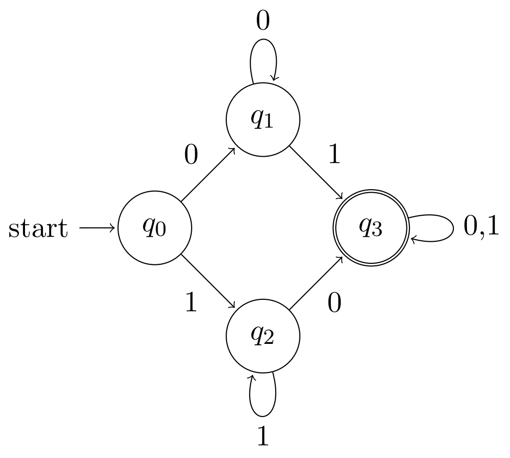
For each of the following strings, indicate whether the DFA accepts or rejects:

string	accepts	rejects
0000		
1111		
0010		
0110		
0100		

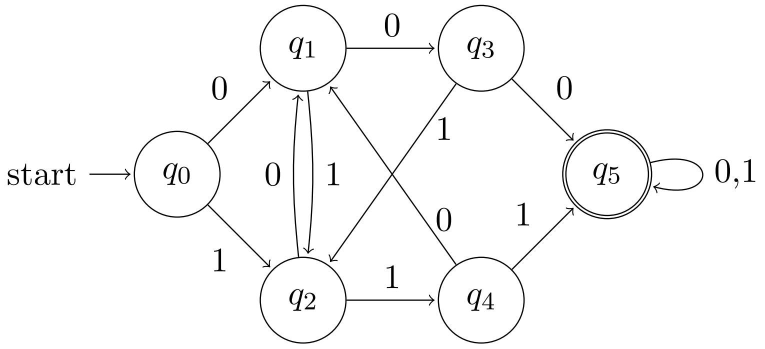
Solution. The first, second, and fifth strings would be reject; the third and fourth would be accepted. (In general, this DFA accepts all strings which are of the form $w = 0^m 1^n 0$, where m and n are integers with $m \geq 0$ and $n \geq 1$.)

2. Consider the following DFAs. In simplest terms, what languages would they accept?

(a) See below.



(b) See below.



Solution.

(a) The set of strings $w \in \{0,1\}^*$ which have at least one 0 and at least one 1. Alternately, the set of strings in $\{0,1\}^*$ which are not of the form 0^m or 1^j for nonnegative integers m and n.

(b) The set of strings $w \in \{0,1\}\infty$ such that w contains either one of the substrings 000 or 111.

Note. If teams do not specify the alphabet $\{0,1\}^*$ for either problem, take off one point from the total combined score of parts (a) and (b).

3. For each of the problems below, construct a DFA which accepts the given language. You do not need to prove that your submission works, but it is recommended that you at least provide a summary of how your DFA works. (This is so the graders do not take off points for DFA's which look different from the ones in the official solutions but still work.)

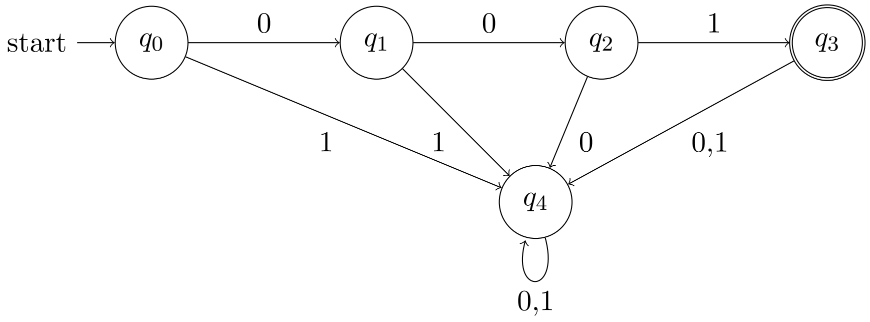
(a) $L_1 = \{w \in \{0,1\}^* \mid w = 001\}$

(b) $L_2 = \{w \in \{a,b\}^* \mid w \text{ has even length}\}$

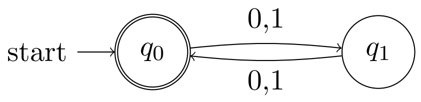
(c) $L_3 = \{w \in \{0,1\}^* \setminus \varepsilon \mid w \text{ when expressed as an integer in binary is divisible by } 5\}$

Solution. The following DFA's are samples.

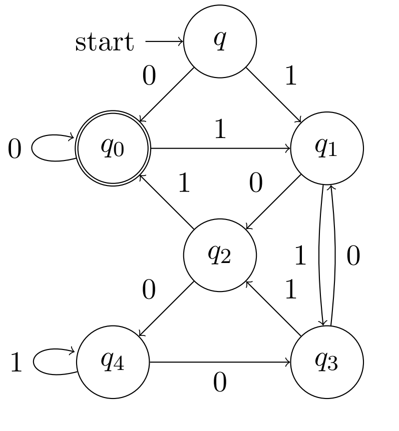
(a) In general, any such DFA should keep track of how close the string is to completing the 001 sequence.



(b) All the DFA needs to do is keep track of the parity of the number of steps taken. This should take at most two states. (Note that ε has length 0 and thus should also be accepted.)



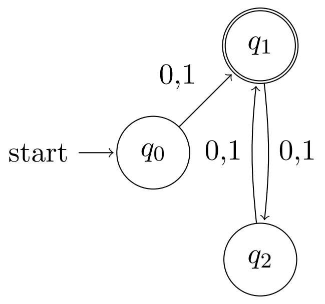
(c) Keeping track of the value of the integer modulo 5 is important. The fact that we read the number from left to right suggests that we can actually do this in only five states; if we were reading from right to left, we would need 20.



4. Let D be a DFA with exactly three states. Suppose this DFA accepts the string 000 but *not* the empty string ε. What is the smallest positive integer $k > 3$ such that D *must* also accept 0^k? (Here, exponentiation is shorthand for repetition; for example, 0^4 represents the string 0000.)

Solution. The answer is $k = 5$.

First it suffices to show that there exists a DFA for which $k = 5$. The following is one such DFA.



Now it suffices to show that 5 is maximal. Let D be a DFA with three states which accepts 000 but not ε. Note that by definition, the start state of D is not an accept state. Let s_1, s_2, and A be the states which the DFA halts after inputting strings 0, 00, and 000 respectively. We now split the analysis of the possible values for the s_i into several very short cases. (These are very very rough sketches.)

- **CASE 1:** $s_1 = q_0$. Then the DFA must be located at the state q_0 for all time steps. This implies that q_0 must be an accept state, which contradicts the fact that ε is rejected.
- **CASE 2:** $s_2 = q_0$. Then $s_1 = A$. Since s_3 is an accept state, we know that 0 is accepted, and therefore that 0^{2k-1} is accepted by D for all positive integers $k \geq 1$. In particular, we know that D accepts 0^5. Contradiction!
- **CASE 3:** $s_3 = q_0$. Trivial contradiction.
- **CASE 4:** $s_1 = A \neq q_0$ and $s_2 \neq q_0$. In this case, 0000 lands on state s_2, and then 00000 lands on state A, which is a accept state.
- **CASE 5:** $s_1 \neq q_0$ and $s_2 = A \neq q_0$. Then the DFA stays on the state A for all time steps after 00. In particular, D must accept 0^5.

These are all possible cases, and so indeed $k = 5$ works.

4 Languages and Regularity

It is useful to understand both the capabilities and limitations of any model of computation such as a DFA. This roughly translates to determining which problems either do not have solutions or have solutions which extend beyond the scope of the aforementioned model of computation. This next section, as well as the ones which succeed it, help develop the theory behind what these DFA machines can actually do.

Definition 4.1. *We say that a language Σ^* is regular if there exists a DFA D which accepts all words $w \in \Sigma^*$ and rejects all words $w_0 \notin \Sigma^*$. If this is the case, we say that D decides the language or is a decider for the language. (Either phrase is fine.)*

As an example, the language

$$L = \{x \in \{0,1\}^* \mid \text{the second digit from the left is a } 0\}$$

is regular, because the DFA pictured in the previous section is a decider for L.

1. Is any language consisting of a finite number of words (sometimes called a "finite language") regular?

   Solution. Yes. Let L be a finite language, and define

   $$\mathcal{S} = \{n \in \mathbb{N} \mid \exists w \in L \text{ with } |w| = n\}.$$

   Since L is finite, $\mathcal{S}$ is finite. This implies by the Well-Ordering Principle that $\mathcal{S}$ has this maximum. Let k denote this maximum. Construct a DFA that contains $|\Sigma|^{k+1} - 1$ states, one for each possible word of length at most k in $|\Sigma|^*$. Such a DFA can then mark a state as accepting iff the word associated with that state is in L. This DFA decides L.

Determining whether a language is regular is somewhat straightforward: construct a DFA which decides it! This technique is the basis for almost all problems which ask you to show that some language is regular. Proving a language is irregular, however, is a bit more difficult; one has to come up with a precise argument that works against *all* possible DFAs. The key to doing this relies in the 'F' in DFA. The following problem highlights the general technique.

2. Consider the language $L = \{0^n 1^n : n \in \mathbb{N}\}$.

   (a) Suppose D is a DFA with n states which decides L. Furthermore, let S be a set of consisting of k words $w \in \{0,1\}^*$. What is the smallest value of k such that for all such sets S, there exist two words in S which land in the same state at the end of the simulation? Provide brief justification.

   (b) Using this, show that L is irregular, i.e. show that D does not exist. (Hint: how can you use the main idea of the previous problem to arrive at a contradiction?)

   Solution.

   (a) The answer is $k = n + 1$ by the Pigeonhole Principle.

   (b) As above, suppose D is a DFA with n states which decides L. Consider the set of $n+1$ strings

   $$\mathcal{S} = \{0^k \mid k \in \mathbb{N}, 0 \leq k \leq n\} = \{\epsilon, 0, 0^2, \ldots, 0^n\}.$$

   There are $n+1$ strings in this set, so by the previous problem two of them, say 0^i and 0^j, must land in th same state. Now recall that since D is deterministic, for any string $s \in \{0,1\}^*$, $0^i s$ and $0^j s$ must also land on the same state. With this in mind, set $s = 1^i$. Then $0^i 1^i$ is accepted, but $0^j 1^i$ is rejected - contradiction! Thus, D cannot exist, meaning that L is irregular.

Before we move on to the main meat of the section regarding regularity (the problems), an interlude. Up until now, we have been informal about what exactly a DFA is in a mathematical sense. We have refrained from doing this in order to prevent definitions from interfering with intution. However, some of the problems from this point forward benefit greatly from formal notation, so we will introduce it here.

Definition 4.2. *Formally, a deterministic finite automaton M is a 5-tuple*

$$M = (Q, \Sigma, \delta, q_0, F),$$

where

- *Q is the (finite) set of states of M;*
- *Σ is the alphabet of the strings inputted to and processed by M;*
- *$\delta : Q \times \Sigma \mapsto Q$ is a transition function which details exactly how the states and alphabet function with each other (in other words which arrows point to which states);*
- *$q_0 \in Q$ is the start state of M;*
- *$F \subseteq Q$ is the set of accepting states of M.*

While formal notation is not necessary for completing the rest of this Power Round, it may help make some of the solutions easier to write.

3. Show that the language $\{1^{2^n} \mid n \in \mathbb{N}\}$ is irregular.

   Solution. The idea is similar. Suppose there exists a DFA D which decides this language, and suppose this DFA has k states. Consider the $k+1$ strings

   $$1, \quad 1^2, \quad 1^4, \quad \dots \quad 1^{2^k}.$$

   By the Pigenhole Principle, there exist two strings, say 1^{2^i} and 1^{2^j}, which end on the same string.

4. The above two problems demonstrate the usefulness of elementary combinatorics to prove results regarding regularity. Using these basic ideas, we can construct very powerful results in automata theory. One of the most fundamental is stated below.

   Theorem 4.1 (Pumping Lemma). *Let L be a regular language. Then there exists an integer $P \geq 1$ such that any $w \in L$ with $|w| \geq P$ can be written as $w = xyz$ with y not equal to the empty string such that*

   - *$|xy| \leq P$, and*
   - *for all integers $i \geq 0$, the string xy^iz is also in L.*

   We call P the pumping length of L.

   In this problem we give a proof of the lemma and then see how it can be used to solve some regularity problems.

   (a) Suppose M is a DFA which decides L. Let p be the number of states of M. Consider an input

   $$s = s_1 s_2 \dots s_n \qquad \text{with} \quad n \geq p.$$

   Let q_k be the state the automaton is in after reading the character s_k. Show that there exist integers $0 \leq i < j \leq p$ such that $q_i = q_j$.

   (b) Let i and j be as above. Define

   $$x = s_1 s_2 \dots s_i, \quad y = s_{i+1} \dots s_j, \quad \text{and} \quad z = s_{j+1} \dots s_n.$$

   Show that these x, y, and z fit the description given in the Pumping Lemma, where $P = p$.

(c) Using the Pumping Lemma, show that the language

$$L = \{s : |s| \text{ is a prime number}\}$$

is not regular.

Solution.

(a) Trivial.

(b) Note that the first condition is trivial, since $|xy| = j \leq p$. We now demonstrate that the second condition holds for this choice of x, y, and z. Denote the function $\delta_0 : \Sigma \mapsto Q$ which takes any word $w \in \Sigma$ and outputs the state $q \in Q$ which w lands on after being fed through D. Note that by the manner in which we chose our i and j, $\delta_0(x) = \delta_0(xy)$. Now we prove that $\delta_0(x) = \delta_0(xy^k)$ for all positive integers k. The base case $k = 1$ is already taken care of. Now for the induction step, suppose the proposition holds true for some k, and write

$$\delta_0(xy^{k+1}) = \delta_0(xy^k y) = \delta_0(xy) = \delta_0(x).$$

This completes the proof. (Here, we use the fact that if $\delta_0(a) = \delta_0(b)$ for some strings a and b, then $\delta_0(ac) = \delta_0(bc)$ for all c.) From here, the result is easy, as we now know that $\delta_0(xyz) = \delta_0(xy^iz)$ for all i, and since xyz is in L, xy^iz must also be in L.

(c) Suppose for the sake of contradiction that L is regular, so it has a pumping length P. Denote by n a prime number greater than P, and consider a string s of length $|s| = n$. Now take $s = 0 \ldots 0$, and write $s = xyz$, where x, y, and z are analogous to the x, y, z in the statement of the Pumping Lemma. If the length of y is l, then xy^iz is of length $n + (i-1)l$. By the Pumping Lemma, we know that all such words are in our language, so in particular there must exist n and l such that $n + (i-1)l$. However, setting $i = n + 1$ gives a contradiction. Hence L cannot be regular.[1]

5. One natural question to ask is how regularity behaves under certain string and set operations. These might include the union and intersection of two languages as well as their concatenation. It turns out that regular languages are closed under many different kinds of operations. The following problems will ask to prove closure of regularity under different types of operations. They are roughly ordered by difficulty, with a few notable exceptions.

   (a) **COMPLEMENT:** Let L be a regular language over the alphabet Σ, and set

   $$L^c = \{x \in \Sigma^* \mid x \notin L\}.$$

   Prove that L^c is also regular.

   (b) **INTERSECTION:** Suppose A and B are two regular languages over a common alphabet Σ. Prove that

   $$A \cap B = \{x \in \Sigma^* \mid x \in A \text{ and } x \in B\}$$

   is also regular.

[1]Proof taken from https://math.berkeley.edu/ moorxu/oldsite/notes/154/154main.pdf

(c) **UNION:** Suppose A and B are two regular languages over a common alphabet Σ. Prove that

$$A \cup B = \{x \in \Sigma^* \mid x \in A \text{ or } x \in B\}$$

is also regular.

(d) **SET DIFFERENCE:** Suppose A and B are two regular languages over a common alphabet Σ. Prove that

$$A \setminus B = \{x \in \Sigma^* \mid x \in A \text{ and } x \notin B\}$$

is also regular. (Hint: use some of the results from above!)

Solution.

(a) Let $D = (Q, \Sigma, \delta, q_0, F)$ be the DFA which decides L. Consider the alternate DFA $D' = (Q, \Sigma, \delta, q_0, Q \setminus F)$. In other words, D' is the DFA which results when all the accept states of D are turned into reject states and vice versa.

Now we show this decides L^c. First consider a word $w \in L^c$. Note that when w is run on D, it does not accept. In other words, its ending state is some state in $Q \setminus F$. But this is now an accept state in D', so in fact w is accepted by D'. Similarly, if a word $v \in L$, then v is not accepted by D'. The conclusion follows.

(b) Since A and B are regular languages, there are DFAs $M = (Q, \Sigma, \delta, q_0, F)$ and $M' = (Q', \Sigma, \delta', q_0', F')$ that decide A and B respectively. To show that $A \cap B$ is regular, we construct a DFA $M'' = (Q'', \Sigma, \delta'', q_0'', F'')$ that decides $A \cap B$. Set

- $Q'' = Q \times Q' = \{(q, q') : q \in Q, q' \in Q'\}$,
- δ'' is such that for $(q, q') \in Q''$ and $a \in \Sigma$,
$$\delta''((q, q'), a) = (\delta(q, a), \delta'(q', a)),$$
- $q_0'' = (q_0, q_0')$,
- $F'' = \{(q, q') : q \in F \text{ and } q' \in F'\}$.

This completes the definition of M''. It remains to show that M'' indeed decides the language $L_1 \cup L_2$.

First, we show that $L_1 \cap L_2$ is in the language decided by M''. Call this language $L(M'')$. Suppose $w \in L_1 \cap L_2$, which means w either belongs to L_1 or it belongs to L_2. WLOG assume $w \in L_1$. Let n be the length of w. Then we know that w induces a sequence of states r_0, r_1, …, $r_n \in Q$ such that $r_0 = q_0$, $\delta(r_{i-1}, w_i) = r_i$ for each $i \in \{1, 2, \dots, n\}$, and $r_n \in F$. This w will also induce a sequence of states of M'', r_0', r_1', …, $r_n' \in Q'$ such that $r_0' = q_0'$, $\delta'(r_{i-1}', w_i) = r_i'$ for each $i \in \{1, 2, \dots, n\}$, and $r_n' \in F'$. Due to how we have defined M'', when we run w on M'', it will induce the sequence of states (r_0, r_0'), $(r_1, r_1'$, …, $(r_n, r_n') \in Q''$ such that $(r_0, r_0') = (q_0, q_0')$, $\delta''((r_{i-1}, r_{i-1}'), w_i) = (\delta(r_i, a), \delta'(r_i', a))$ for each $i \in \{1, 2, \dots, n\}$. The final state (r_n, r_n') is such that $r_n \in F$ and $r_n' \in F'$, which by the definition of F'' ensures that $(r_n, r_n') \in F''$. Therefore M'' accepts w, i.e. $w \in L(M'')$.

For the other direction, run the reverse argument. Consider an element in $L(M'')$. Since its sequence of states ends in an accepting state, its components must also both be accepted. This means that the sequence of states in the first component end in something in F, while the sequence of states in the second component end in something in F'. Hence $w \in A \cap B$ as well, completing the proof.

(c) Run a similar argument as before, except set $F'' = \{(q, q') : q \in F \text{ or } q' \in F'\}$ instead.

(d) Remark that

$$A \setminus B = \{x \in \Sigma^* \mid x \in A \text{ and } x \notin B\} = A \cap B^c.$$

From the previous parts of this problem, we know that B^c is regular, so $A \cap B^c$ must be regular as well. This completes the proof.

6. For any language $L \in \{0,1\}^*$, define a language L_0 which consists of the set of words of the form

$$a_1 s_{1,2} a_2 s_{2,3} a_3 \ldots a_{k-1} s_{k-1,k} a_k$$

where $a_1 a_2 a_3 \ldots a_k$ is a word in L. Here, $s_{i,j}$ is 1 when $a_i \neq a_j$ and zero otherwise. For example, the word 1101 in L corresponds to the word 1011011 in L_0.

Suppose L is regular. Must L_0 also be regular? (Note: if necessary, you may quote any results from the previous two problems without proof.)

Solution. I claim the answer is yes.

For ease of notation, let $s(x, y)$ denote the parity of x and y, i.e. $s(x, y) = 1$ iff $x \neq y$. Now denote by f the function taking strings to strings such that

$$f(x_1 x_2 \ldots x_n) = x_1 s(x_1, x_2) x_2 s(x_2, x_3) \ldots s(x_{n-1}, x_n) x_n.$$

Consider the two languages

$$L_1 = \{a_1 x_1 a_2 x_2 \ldots a_{n-1} x_{n-1} a_n \mid a_1 a_2 \ldots a_n \in L,\ x_1 x_2 \ldots x_{n-1} \in \{0,1\}^*\}$$

and

$$L_2 = \{f(w) \mid w \in \{0,1\}^*\}.$$

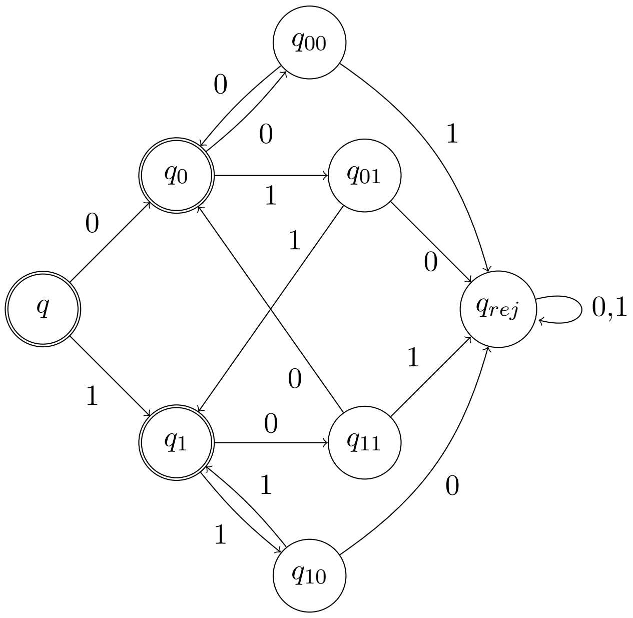
We shall now show that L_1 and L_2 are both regular languages. This implies that their intersection is also regular. But their intersection is exactly L_0. Hence if we can show L_1 and L_2 are regular, then we are done.

First, we show that L_1 is regular. Since L is regular, there exists a DFA $D = (Q, \Sigma, \delta, q_0, F)$ which decides it. Now construct another DFA $D' = (Q', \Sigma, \delta', q_0, F)$ with the following properties:

- $Q \subseteq Q'$ with $|Q'| = 3|Q|$. This means that there exists a bijection $g : \{0,1\} \times Q \mapsto Q' \setminus Q$.
- For all $w = (e, q) \in \{0,1\} \times Q$, we have $\delta'(w) = g(w)$ and $\delta'(e, \delta'(w)) = \delta(w)$.

This decides L_1. To prove this, first note that any string $w \in \{0,1\}^*$ with $|w|$ even cannot be accepted by D'. This is because any such string must map to a state in $Q' \setminus Q$, which we know contains zero accept states. Now remark that $\delta'(e, \delta'(w)) = \delta(w)$, we can use an induction-esque argument to prove that the state which $a_1 a_2 \ldots a_n$ lands on and the state which $a_1 x_1 a_2 x_2 \ldots a_{n-1} x_n$ lands on are identical. Hence a string of odd length in $\{0,1\}^*$ is accepted by D' iff the string formed by the odd-indexed characters of the string is accepted by D. (This is obviously not a rigorous argument, but it should suffice for grading purposes.)

Now we show that L_2 is regular. Consider the following DFA:



I claim this decides L_2. To prove this, we induct on the number of time steps. Consider any accepting string w. Then w must end with either a 0 or a 1.

- If w ends in a 0, then $w00$ and $w11$ should accept, while $w01$ and $w10$ should reject.
- If w ends in a 1, then $w10$ and $w01$ should accept, while $w00$ and $w11$ should reject.
- In either case, $w0$ and $w1$ should reject.

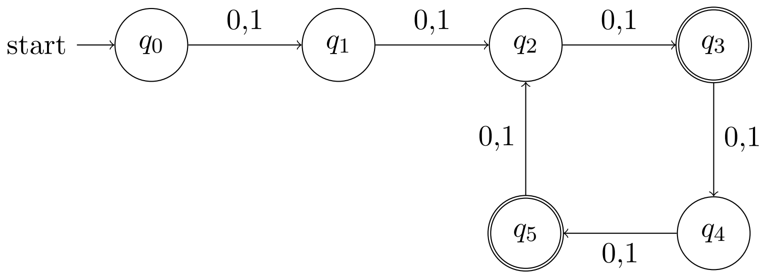
It is not hard to see that all three of these conditions are satisfied. Hence we can build on the length of the input and prove that L_2 is indeed the language decided by D.

We have shown L_1 and L_2 to both be regular; thus L_0 must indeed be regular.

5 DFA Minimization

One of the biggest overall themes in the field of computer science is optimization. If wew have a program that can solve a problem but does so in exponential time (e.g. the number of steps needed to solve the problem on input of length n is roughly 2^n), then the algorithm isn't really that practical! Computer scientists aim to design and implement processes which are efficient in some sense. As a result, it might be beneficial to see how we can explore this type of minimization in DFAs. The most natural way to do so involves looking at the number of states for a DFA which decides some language L.

1. Below is a picture of a DFA which accepts some language L. Although this DFA is a decider for L, it is somewhat inefficient. Propose two modifications to this DFA so that it still decides L but requires fewer states to do so.



Solution. There are many different possible answers for this. I personally thought of the following:

(a) Condense the cycle of length 4 into a cycle of length 2 (which can be done since the accept and reject states alternate).

(b) Remove one of the nodes from the beginning, and switch the accept and reject states of the cycle of length 2.

2. Consider the language
$$L = \{0^n 1^m \mid n - m \equiv 0 \pmod 3\}.$$
Note that this language is a more general version of one that we've explored before. However, in fact this language is regular! What is the smallest number of states needed in a DFA that decides L? (Your answer must both establish a minimum number of states and show that this minimum is achievable.)

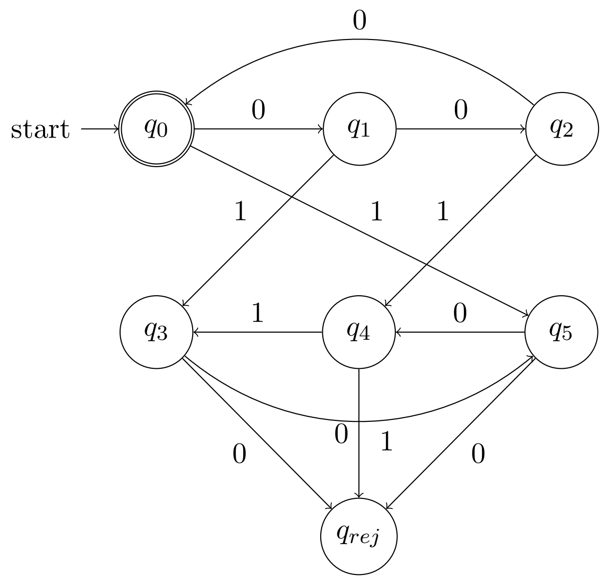
Solution. I claim the answer is 7 states.

Suppose there exists a DFA D which decides L in fewer than 7 states. Consider the set of words
$$\mathcal{S} = \{0, 00, 000, 1, 11, 111, 10\}.$$
By the Pigeonhole Principle, there must exist two strings which end p on the same after after ron on D. We now divide into cases.

- **CASE 1: One of the strings is 10.** Then note that the second string must be one of the other six. It is not hard to see that for each of the other strings there exists a suffix which when added to the end of the string produces an accepting string. However, $10x$ must reject for any string x. So we get a contradiction.
- **CASE 2: Both strings contain all zeroes.** Call these strings 0^i and 0^j respectively. Then note that appending 0^{6-i} to both strings yields that 0^6 and 0^{6+j-i} must either both accept or both reject. This is false, because the former accepts while the latter rejects.
- **CASE 3: Both strings contain all ones.** Use an argument as above.
- **CASE 4: One string contains all ones while the other contains all zeroes.** Suppose the strings are 0^i and 1^j for some i and j between 1 and 3. Now append the string 0^{6-i} to both strings. This gives that 0^6 and $1^j 0^{6-i}$ must either both accept or both reject. But this is false, since 0^6 accepts but any string of the form $1^m 0^n$ must automatically be rejected.

We have exhausted all cases, and so we are done.

It suffices to show that 7 is attainable. Here is such a DFA.



3. (a) Define

$$\mathcal{L}_n = \{x \mid x \in \{0,1\}^* \text{ and the } n^{\text{th}}\text{-symbol from the left is a 1 }\}.$$

Show that there is some constant number c (independent of n) such that there is a DFA that accepts $\mathcal{L}_n$ with at most $n + c$ states.

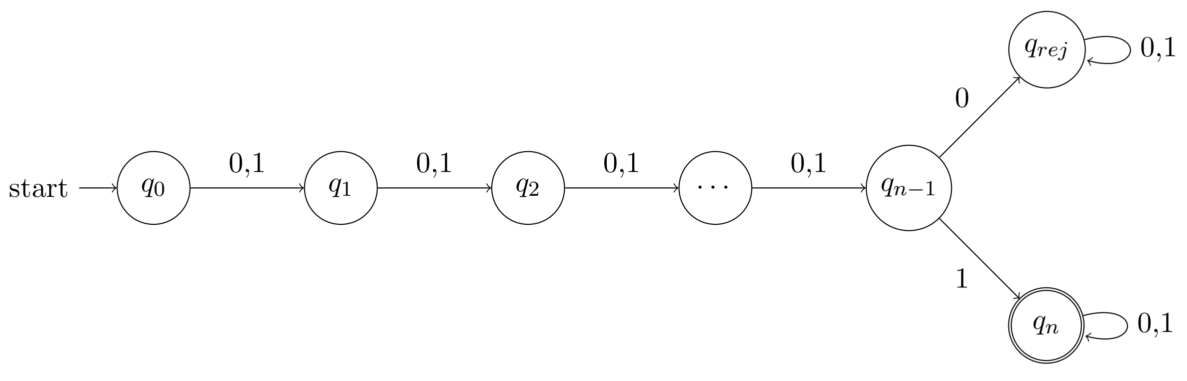
(b) Define

$$\mathcal{R}_n = \{x \mid x \in \{0,1\}^* \text{ and the } n^{\text{th}}\text{-symbol from the right is a 1 }\}.$$

Show that any DFA that accepts $\mathcal{R}_n$ has at least 2^n states.

Solution.

(a) The following is one such DFA.



This DFA works because it takes no care to the first $n-1$ characters of the input string, but then checks the n^{th} character of the string to see if it is a 1. In terms of n, this DFA has $n+2$ states, so $c = 2$ works.

(b) Let D be a DFA which decides $\mathcal{R}_n$. Suppose for the sake of contradiction that D has fewer than 2^n states. Consider the set of strings $w \in \{0,1\}^*$ with $|w| \leq n$. There are 2^n such possible strings. Thus, by the Pigenhole Principle two of these strings must end up in the same state after run by the DFA.

Since these two strings are distinct, they must differ in some bit. Let this bit be the i^{th} bit to the right. Denote by A the string which has a 1 in this bit and B the string which has a 0 at this bit. Now append the string 0^{n-i} to both strings. Then $A0^{n-i}$ has a 1 in the n^{th} bit, while $B0^{n-i}$ has a 0 in this bit. But it must be true that either both of hese strings are accepted by D or both are rejected by D - contradiction! Hence our original assumption was false and D must have at least 2^n states as desired.

It turns out that DFAs are simple enough in nature that we can algorithmically construct a minimal DFA for any given language L! (The only caveat is that we need to find some DFA which decides L first before applying this algorithm, but, eh, that's a worthy trade-off. To explore this idea, we need to develop the notion of equivalence classes.

Definition 5.1. *Let Σ be an alphabet, and let A be any language over Σ^*. For any word $v \in \Sigma^*$, let*

$$S_v(A) = \{w \in \Sigma^* \mid vw \in L\}.$$

We then say that two strings $x, y \in \Sigma^*$ are equivalent iff $S_x(A) = S_y(A)$. We denote this mathematically through the relation $x \equiv_A y$. Note that $\equiv_A$ is an equivalence relation, and so we may refer to the equivalence classes[2] *of A.*

There exists a similar notion of equivalence with regards to DFAs.

Definition 5.2. *Given a DFA $M = (Q, \Sigma, \delta, q_0, F)$, we say that two strings $x, y \in \Sigma*$ are indistinguishable iff $\delta*(q_0, x) = \delta^*(q_0, y)$. In other words, the state reached by M on input x is the same as the state reached by M on input y. We denote the notion of equivalence as $x \equiv_M y$. Just as above, note that $\equiv_M$ is an equivalence relation and so we may refer to the equivalence classes of M.*

1. Let $L = \{0^n 1^n \mid n \in \mathbb{N}\}$. What is $S_{111100}(L)$?

   Solution. $S_{111100}(L)$ is the lonely singleton $\{00\}$. This is not hard to show true.

2. Suppose M is a DFA with N states. At most how many equivalence classes can M have? What is an example of a DFA M for which equality does not hold?

   Solution. At most N states. Proof uses a Pigeonhole argument. An example of when equality does not hold is when we have redundant states, e.g. the DFA in the example above.

3. Show that the notions of equivalency and indistinguishability are identical. In other words, let M be a DFA and let $A = L(M)$ (i.e. A is the language which is decided by M). Then for any $x, y \in \Sigma^*$, if $x \equiv_M y$, then $x \equiv_A y$.

   Solution. Denote by $\delta^*(q_0, x)$ the state reached by M on input x. This function will be used to make the solution easier to write.

[2]For any set X equipped with an equivalence relation $\sim$ and for any a in X, we say that the equivalence class of a is the subset of all elements $x \in X$ for which $x \sim a$. It follows that these equivalence classes partition X into groups; we then define the number of equivalence classes of $\sim$ as the number of groups in this partition.

Let $z \in \sigma^*$. Clearly, $\delta^*(q_0, xz) = \delta^*(q_0, yz)$. Then

$$xz \in A \iff \delta^*(q_0, xz) \in F \iff \delta^*(q_0, yz) \in F \iff yz \in A.$$

Since this holds true for arbitrary z, we must have $x \equiv_A y$.

We now have enough background knowledge to state an important result.

Theorem 5.1 (Myhill-Nerode)**.** *A language A is regular iff the equivalence relation $\equiv_A$ has a finite number of equivalence classes. Furthermore, there exists some DFA M with $L(M) = A$ such that every state of M uniquely corresponds to an equivalence class of $\equiv_A$.*

1. Prove this theorem.

   Solution. The only if direction is easy. Note that since A is regular, there exists a DFA D which decides it. By the previous problem, we know that $\equiv_A$ has at most as many equivalence classes as $\equiv_D$. But $\equiv_D$ has a finite number of equivalence classes - in fact, as most the number of states of D.

   The if direction is a bit tougher. The main idea here is to use the equivalence classes to construct the DFA explicitly.

   Suppose a language L has a finite number of equivalence classes, labeled T_1, T_2, $\ldots$, T_n. (The use of T is to distinguish these suffix sets from the suffix set of a word in L.) We state a lemma.

   LEMMA: If two words v and w have the same suffix sets S_v and S_w, then for any $a \in \Sigma$ we will have $S_{va} = S_{wa}$.

   Proof. Note that by the definition of suffix sets, if a word $x \in S_{va}$, then $ax \in S_v$ and vice versa. Since $S_v = S_w$, we have $ax \in S_w \iff x \in S_{wa}$. Since this holds for arbitrary a, we have $S_{va} = S_{wa}$ as desired. □

   We now construct the DFA D. Let $q_1, \ldots, q_n$ be states which correspond to the equivalence classes T_1 through T_n. The start state q of D is the state which corresponds to the equivalence class containing ε. Since each state is the suffix set for some string, we will denote each state by some canonical string. For the transition function, let $\delta(S_v, a) = S_{va}$, where v is the string we choose to represent the state S_v. By the above claim, the transition function is the same regardless of the choice of v.

   Now it suffices to show that this DFA works. Let $w = w_1 \ldots w_n$ be a word accepted by the DFA. By the definition of the transition function, it must be the case that $\delta(S_\varepsilon, w_1) = S_{w_1}$, $\delta(S_{w_1}, w_2) = S_{w_1 w_2}$, and so on; an inductive argument shows that

$$\delta(S_{w_1 \ldots w_{n-1}}, w_n) = S_w.$$

   Furthermore, since this is accepted, we now that S_w must contain ε, which implies $w \in L$. Reversing the argument above gives that every word in L is accepted by this DFA.

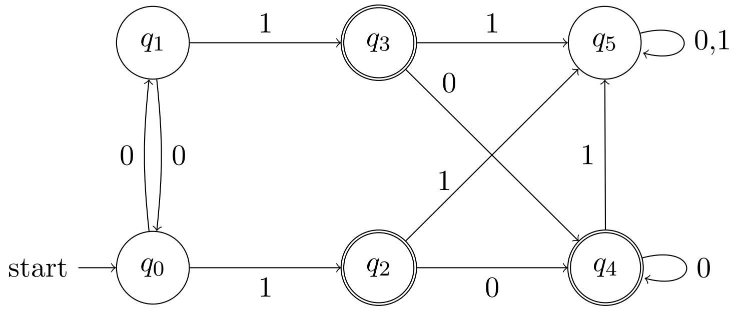
   Hence we have shown that D is the DFA which decides the language A, and thus A must be regular.[3]

Using this theorem, we now can construct an algorithm for minimizing any DFA! The algorithm is as follows. (Note: in this algorithm, we use the official definition of a DFA given in the previous section.)

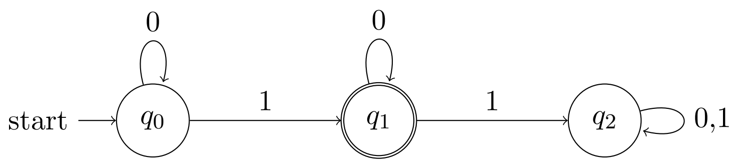
[3]Thanks to the CMU course 15-251 F15 for providing the basis for the solution to this problem.

- Draw a table for all pairs of states (Q_i, Q_j) (not just those which are connected directly). Initially, these pairs will be unmarked.
- Mark all state pairs (Q_i, Q_j) with the property that $Q_i \in F$ and $Q_j \notin F$ and mark them.
- For some input A, if there exist states Q_i and Q_j with the property that $(\delta(Q_i, A), \delta(Q_j, A))$ is marked, then mark the pair (Q_i, Q_j). Repeat this until there are no more states that can be marked.
- Combine all the unmarked pairs of states into a single state in the reduced DFA. Return this new DFA.

1. Perform this algorithm on the following DFA. What is the result?



Solution. This should be the resulting output.



2. Show using the Myhill-Nerode Theorem that this procedure produces a DFA with the minimal number of states.

   Solution. Note that each marked pair represents a pair of states which cannot possibly be part of the same equivalence class - just note that ε cannot appear in both. By inductively crossing out more and more pairs of states, we can thus reduce the number of states we have to check for equivalence. (This is not a formal argument; rather, this is a general idea of what is going on.)