CMIMC 2016
Power Round

INSTRUCTIONS

1. Do not look at the test before the proctor starts the round.
2. This test consists of several problems, some of which are short-answer and some of which require proofs.
3. Answers should be written and clearly labeled on sheets of blank paper. Each numbered problem should be *on its own sheet*. If you have multiple pages, number them as well (e.g. 1/3, 2/3).
4. Write your team ID on the upper-right corner and the problem and page number of the problem whose solution you are writing on the upper-left corner on each page you submit. Papers missing these will not be graded. Problems with more than one submission will not be graded.
5. Write legibly. Illegible handwriting will not be graded.
6. In your solution for any given problem, you may assume the results of previous problems, even if you have not solved them. You may not do the same for later problems.
7. Problems are not ordered by difficulty. They are ordered by progression of content.
8. No computational aids other than pencil/pen are permitted.
9. If you believe that the test contains an error, submit your protest in writing to Porter 100.

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). DFAs 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, $\{0, 1\}^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 100, \dots\}$.*

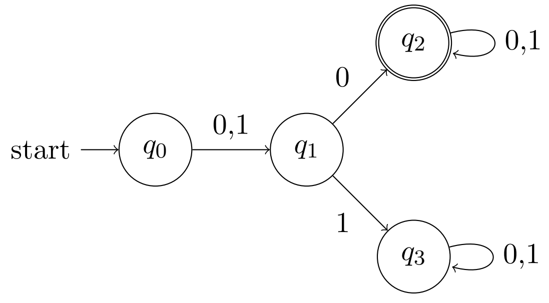
Definition 2.4. *A* language *over an alphabet Σ is any subset of Σ^*.*

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

3 DFAs

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:

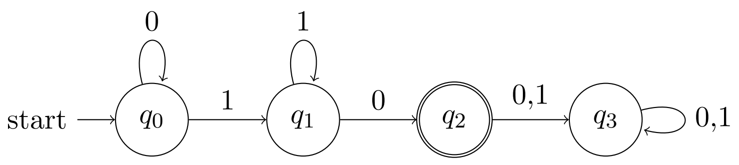


Each circle represents a *state*. A state with two circles is an *accepting state*, which means the DFA will accept if the string ends up there. Formally, a DFA over an alphabet Σ is defined as a finite set of states (with one start state and some of which can be accepting states), each with an arrow 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. [**2**] Consider the following DFA:

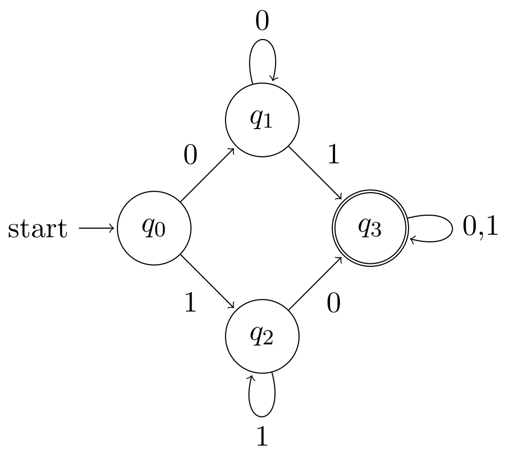


For each of the following strings, indicate whether the DFA accepts or rejects:

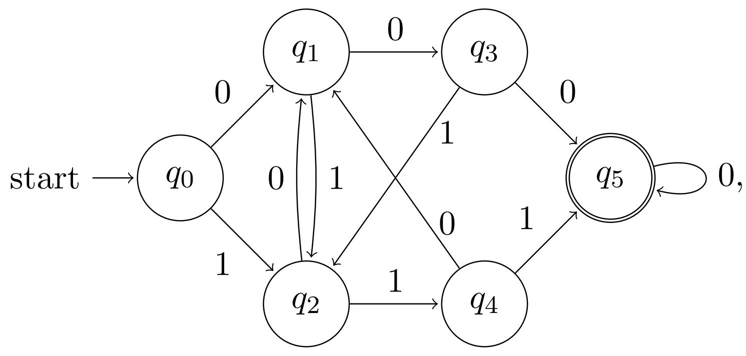
string	accepts	rejects
0000		
1111		
0010		
0110		
0100		

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

(a) [**2**] See below.



(b) [**4**] See below.



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.

   (a) [**2**] $L_1 = \{w \in \{0,1\}^* \mid w = 001\}$
   (b) [**2**] $L_2 = \{w \in \{a,b\}^* \mid w \text{ has even length}\}$
   (c) [**6**] $L_3 = \{w \in \{0,1\}^* \setminus \varepsilon \mid w \text{ when interpreted as an integer in binary is divisible by } 5\}$

4. [**5**] 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.) Prove that your k is minimal.

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. [**3**] Is any language consisting of a finite number of words (sometimes called a "finite language") regular?

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) [**2**] 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) [**4**] Using this, show that L is irregular, i.e. show that D does not exist.

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. [**4**] Show that the language $\{1^{2^n} \mid n \in \mathbb{N}\}$ is irregular.

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.*

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

(a) [**2**] Suppose M is a DFA which decides L. Let p be the number of states of M. Consider an input
$$s = s_1s_2\dots s_n \qquad \text{with} \quad n \geq p.$$
Let q_k be the state the automata 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) [**6**] Let i and j be as above. Define
$$x = s_1s_2\dots s_i, \quad y = s_{i+1}\dots s_j, \quad \text{and} \quad z = s_{j+1}\dots s_n.$$
Show that these x, y, and z fit the description given in the Pumping Lemma, where $P = p$.

(c) [**4**] Using the Pumping Lemma, show that the language
$$L = \{s \mid |s| \text{ is a prime number}\}$$
is not regular.

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.

(a) **COMPLEMENT** [**2**]: 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** [**6**]: 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.

(c) **UNION** [**6**]: 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** [**2**]: 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.

There are other set operations (e.g. concatenation) that can also be shown to be closed under regularity, but doing so requires the notion of a non-deterministic finite automaton, which is beyond the scope of this power round.

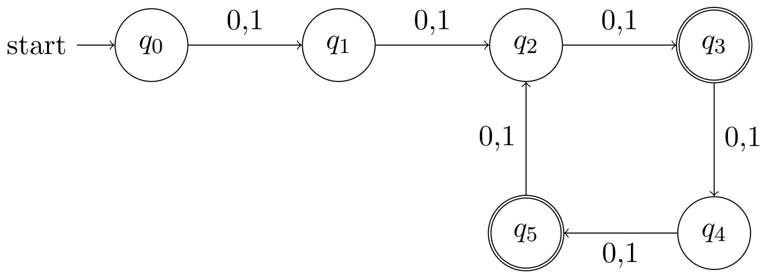
6. [**8**] For any language $L \in \{0,1\}^*$, define a language L_0 which consists of the set of words of the form
$$a_1s_{1,2}a_2s_{2,3}a_3\dots a_{k-1}s_{k-1,k}a_k$$
where $a_1a_2a_3\dots a_k$ is a word in L and $s_{i,j}$ is 1 when $a_i \neq a_j$ and zero otherwise. For example, the word 1101 in L corresponds to the word 1011011 in L_0.

Suppose L is regular. Must L_0 also be regular?

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.

7. [**2**] 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.



8. [**8**] 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.)

9. (a) [**4**] 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) [**5**] 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.

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 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[1] *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.*

10. [**2**] Let $L = \{0^n 1^n \mid n \in \mathbb{N}\}$. What is $S_{111100}(L)$?
11. [**3**] 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?
12. [**5**] Show that the notions of equivalency and indistinguishability are identical. In other words, let M be a DFA and let $A = L(M)$. Then for any $x, y \in \Sigma^*$, if $x \equiv_M y$, then $x \equiv_A y$.

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 $\equiv_A$.*

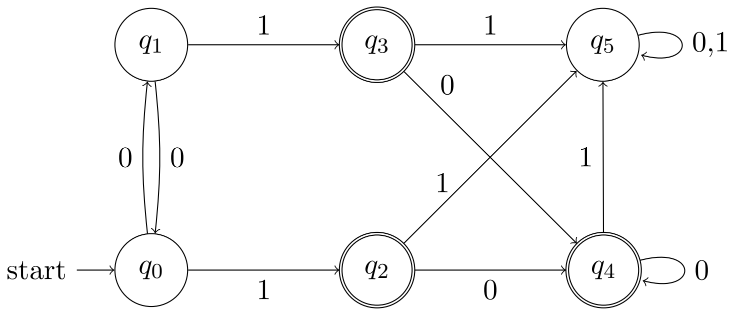
13. [**8**] Prove this theorem.

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.)

- 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 in the reduced DFA. Return this new DFA. (For example, if $\{a, b\}$, $\{b, c\}$, and $\{c, a\}$ are all unmarked, combine the states a, b, c into one big state.)

[1] 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 subsets; we then define the number of equivalence classes of $\sim$ as the number of subsets in this partition.

14. [**4**] Perform this algorithm on the following DFA. What is the result?



15. [**6**] Show using the Myhill-Nerode Theorem that this procedure produces a DFA with the minimal number of states.