Limitations of Finite Automata

So far in this course, we've studied a number of properties of regular languages. First, they are quite robust, with different characterizations based on DFAs, NFAs, and regular expressions. Second, they enjoy strong closure properties: unions, concatenations, stars, complements, intersections, and reverses (among other operations) of regular languages are all regular. And finally, the regular languages capture diverse computational problems appearing in arithmetic, networks, compilers, genomics, etc.

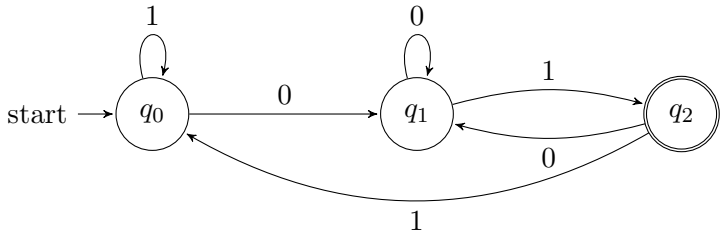
All of these results illustrate the *power* of finite automata to recognize interesting and useful languages. But an equally important objective in the theory of computation is to understand the *limitations* of any computational model you're studying. Some examples of these kinds of questions are:

1. Given a language L, what is the minimum number of states needed in a DFA that recognizes L?
2. Are there languages L that are not regular? That is, are there languages that cannot be recognized by DFAs using any number of states?
3. Given a language L, can we determine whether L is regular or non-regular?

In this note, we'll explore a simple but powerful technique for answering these questions. It is not the only one – you can see Chapter 1.4 of Sipser for a different technique, and we'll explain some of the relative advantages and disadvantages of the two methods in Section 6.

1 A Simple Example

Consider the regular language $A = \{w \in \{0,1\}^* \mid w \text{ ends with } 01\}$. This language is recognized by the following DFA using three states.



It's not too hard to see that any DFA recognizing A needs at least two states, as it needs at least one accept state and one reject state – and these have to be different states. But can we argue that it needs at least three?

Intuitively, the answer seems like, "yes." Any DFA recognizing M seems to need to "remember" whether the string it's seen so far satisfies one of the following three possibilities: It's already ended in 01, it's ended with a 0 (and so seeing another 1 should lead it to accept), or neither, in which case, the DFA needs to see another 01 in order to accept. Each of these possibilities should have to correspond to a different state. So how can we make this intuition formal?

Let M be a DFA recognizing A. We'll consider what happens when we run M on the following three "test" strings: $x = \varepsilon, y = 0, w = 01$. (If it seems like we just pulled these strings out of thin air, don't worry about it...we'll come back to how to choose them later.)

We'll argue that M needs to end up in a different state upon reading each of the strings x, y, w, and hence that it requires at least three states. Because $x, y \in A$ while $w \notin A$, it follows that M needs to end up in a different state when reading w than it does when reading either x or y. So the interesting part is to show that M needs to end up in a different state when reading x than it does when reading y.

To show this, we're going to argue by contradiction. Suppose, for the sake of contradiction, that M ended up in the same state upon reading both x and y. The key observation is that no matter how we extend x and y – as long as we extend them in the same way – the DFA M will always do the same thing on the extended strings. For example, if M ends up in the same state on ε and 0, then it must end up in the same state on 0 and 00. And on 01 and 001. And on 011 and 0011, etc.

Do you see how this might cause a problem? If M ends up in the same state on ε and 0, then it ends up in the same state on 1 and 01. (What we get if we extend both x and y by the single symbol 1.) But $1 \notin A$ while $01 \in A$, so if M recognizes A, then the state that it enters upon reading these strings has to be both a reject state and an accept state. That can't happen, so by contradiction, we conclude that M must have been in different states upon reading $x = \varepsilon$ and $y = 0$. Hence, any DFA recognizing A needs at least three states – one for each of x, y, w.

2 Distinguishability

Let's try to formalize the idea that went into the argument above. We want to abstract and generalize the property exhibited by the test strings x, y, w that forced a DFA recognizing A to go into different states when reading these strings. This property is that A can "distinguish" each pair of these strings when they are extended by some suffix.

Definition 1. Strings x and y are *distinguishable* by a language L if there exists a (possibly empty) string z such that exactly one of xz and yz is in L. The string z is called a *distinguishing extension* of x and y.

Returning to our example, $x = \varepsilon$ and $y = 0$ are distinguishable by the language A via the distinguishing extension $z = 1$. This is because $xz = 1 \notin A$ while $yz = 01 \in A$.

Definition 2. A set S of strings is *pairwise distinguishable* by a language L if every pair of distinct strings $x, y \in S$ is distinguishable by L.

Again returning to our example, the set $S = \{x, y, w\} = \{\varepsilon, 0, 01\}$ is pairwise distinguishable by A. (What are distinguishing extensions for the pairs x, w and y, w?)

3 Lower Bound on Number of DFA States

The fact that $\{\varepsilon, 0, 01\}$ was pairwise distinguishable by A means that any DFA recognizing A requires at least three states. We can generalize this to any language with a pairwise distinguishable set.

Theorem 1. Suppose S is a set of strings that is pairwise distinguishable by L. Then any DFA recognizing L requires at least $|S|$ states.

Proof. Let S be a set of k strings that is pairwise distinguishable by L. Let M be a DFA with $k-1$ or fewer states. We will show that M cannot possibly recognize L, which will prove the theorem.

Consider what happens if we run M on each of the strings comprising S. Since M has at most $k-1$ states, the pigeonhole principle implies that there are two different strings $x, y \in S$ such that M ends up in the same state when reading each of x and y. Let us call this state q.

Now since S is pairwise distinguishable by L, that in particular means x and y are distinguishable by L. So there is a string z such that exactly one of xz or yz is in L. But since M ended up in the same state q when reading either x or y, that means it ends up in the same state upon reading either xz or yz. Since this state is either an accept state or a reject state (but not both), M must exhibit the wrong behavior on either xz or yz. So M does not recognize the language L. □

Example 1. Let $B = \{w \in \{0,1\}^* \mid |w| \leq 4\}$. We will show that any DFA recognizing B requires at least 5 states by constructing a pairwise distinguishable set of size 5.

One example of such a pairwise distinguishable set is $S = \{\varepsilon, 0, 00, 000, 0000\}$. You should convince yourself that any pair of strings in S has a distinguishing extension. For example, $x = 0$ and $y = 00$ are distinguishable via the extension $z = 00$.

Example 2. Generalizing the previous example, let $B_k = \{w \in \{0,1\}^* \mid |w| \leq k\}$. The language B_k has a pairwise distinguishable set of size $k+1$, so any DFA recognizing B_k requires at least $k+1$ states.

We can take our pairwise distinguishable set to be $S_k = \{0^i \mid 0 \leq i \leq k\}$. Let $x = 0^i, y = 0^j \in S_k$ with $0 \leq i < j \leq k$. Then the string $z = 0^{k-j}$ is a distinguishing extension for x, y because $xz = 0^{k-j+i} \notin B_k$, while $yz = 0^k \in B_k$.

Example 3. Let $C = \{w \in \{0,1\}^* \mid w$ represents a number divisible by $3\}$. We will show that any DFA recognizing C requires at least 3 states by constructing a pairwise distinguishable set of size 3.

We can take our set $S = \{0, 1, 10\}$ (the binary representations of 0, 1, 2). Then 0, 1 are distinguishable via the extension 1; 0, 10 are distinguishable via 01; and 1, 10 are distinguishable via 1.

Some intuition for choosing a distinguishing set. Like many constructions involving regular languages, there's more of an art than a science to this. I like to start by asking myself, "what information about the input string does a DFA seem to need to remember" in order to recognize the given language. In Example 1, it seems like we need to remember either 1) whether the string already has length at least 4, otherwise 2) a count of the length of the string seen so far. There are 5 possibilities here, corresponding to the string seen so far having length $0, 1, 2, 3$, or at least 4. To construct a pairwise distinguishable set, then, we just construct 5 strings that result in each of thee 5 possibilities.

In general, if it seems like a DFA needs to keep track of k different possibilities, think of how to construct k different strings that result in each of those possibilities. Then try to argue that those strings are actually pairwise distinguishable by the language.

4 Proving Non-Regularity

The technique we've described so far shows that when a language has a large pairwise distinguishable set, then it requires a large DFA. We can take this idea to the extreme as follows: If a language has an *infinite* pairwise distinguishable set, then it can't be recognized by any DFA whatsoever.

Corollary 1. Suppose there is an infinite set S of strings that is pairwise distinguishable by L. Then L is not regular.

Proof. Suppose, for the sake of contradiction, that L were regular. Then it would have a DFA using some number k of states. However, since L has an infinite pairwise distinguishable set, it, in particular, has a pairwise distinguishable set of size $k+1$. This contradicts Theorem 1. □

Example 4. We will show that the language $L_1 = \{a^n b^n \mid n \geq 0\}$ is non-regular by exhibiting an infinite pairwise distinguishable set. We can take as our set $\{a^i \mid i \geq 0\}$. To see that this is pairwise distinguishable, consider a^i, a^j for $0 \leq i < j$. Then $z = b^i$ is a distinguishing extension because $a^i b^i \in L$ while $a^j b^i \notin L$.

Example 5. The language $L_2 = \{w \in \{a, b\}^* \mid w \text{ has an equal number of } a\text{'s and } b\text{'s}\}$ is non-regular using the same infinite pairwise distinguishable set $\{a^i \mid i \geq 0\}$. Again, consider a^i, a^j for $0 \leq i < j$. Then $z = b^i$ is a distinguishing extension, because $a^i b^i \in L$ while $a^j b^i \notin L$.

Another way to see this is using Example 4 together with closure properties of the regular languages. Suppose for the sake of contradiction that L_2 were regular. Then $L_1 = L_2 \cap L(a^* b^*)$ is the intersection of L_2 with a regular language (the language of a regular expression), hence regular. So L_1 is regular, contradicting Example 4.

Example 6. Consider the unary alphabet $\Sigma = \{1\}$. The language $L_3 = \{1^{n^2} \mid n \geq 0\}$ is non-regular. We will show that this language is its own pairwise distinguishing set. Let $x = 1^{i^2}, y = 1^{j^2} \in L_3$ with $0 \leq i < j$. We claim that $z = 1^{2i+1}$ is a distinguishing extension for x, y. To see this, $xz = 1^{i^2+2i+1} = 1^{(i+1)^2} \in L_3$. On the other hand, $yz = 1^{j^2+2i+1}$ consists of a number of 1's that lies strictly between j^2 and the next perfect square, $(j+1)^2 = j^2 + 2j + 1$ (since $i < j$). Therefore, $yz \notin L_3$, so z indeed distinguishes x and y.

5 Characterizing Minimal DFA Size (Optional)

Let's turn our attention back to regular languages. Now that we have a lower bound method – a technique for showing that a regular language *requires* a large number of DFA states – we can ask ourselves, "how good is this method?" It turns out that this method is extremely good; in fact, it's optimal via the following converse to Theorem 1.

Theorem 2. Suppose L is a language with no pairwise distinguishable set of size $> k$. Then L is recognizable by a DFA with k states.

In other words, if L is a language that requires more than k states, then it has a pairwise distinguishable set of size greater than k. It may, of course, be hard to find this set.

Proof. The cleanest way to present the proof makes use of a bit of jargon, namely, a certain equivalence relation on strings associated to any language L. For a language L, let $\equiv_L$ be the relation defined by $x \equiv_L y$ iff x and y are *not* distinguishable by L. In other words, $x \equiv_L y$ if for every string z, either xz, yz are both in L or xz, yz are both not in L. **Exercise:** Convince yourself that this is indeed an equivalence relation. That is, check that it is symmetric ($x \equiv_L y \implies y \equiv_L x$), reflexive ($x \equiv_L x$), and transitive ($x \equiv_L y$ and $y \equiv_L w \implies x \equiv_L w$).

If L has no pairwise distinguishable set of size $> k$, then the relation $\equiv_L$ breaks Σ^* into at most k equivalence classes. For simplicity, say there are exactly k classes $C_1, \ldots, C_k$. We define a k-state DFA recognizing L as follows. The states of the DFA are $q_1, \ldots, q_k$, one corresponding to each class C_i. We define transitions between the states as follows. Let q_i be a state and let x be any string in C_i. Then for any symbol $\sigma \in \Sigma$, let $\delta(q_i, \sigma) = q_j$ where C_j is the class containing $x\sigma$.

To see that this transition function is well-defined, we need to check that $x\sigma \equiv_L y\sigma$ whenever $x, y \in C_i$. This follows because if z were a distinguishing extension for $x\sigma$ and $y\sigma$, then σz would be a distinguishing extension for x, y.

Finally, we set the start state of the DFA to the state q_i such that $C_i \ni \varepsilon$, and take the set of accept states to be those q_i for which $C_i \subseteq L$.

Now we argue that this DFA recognizes L. For any string $x \in \Sigma^*$, the definition of the transition function ensures that the DFA reaches a state q_j such that C_j is the class containing x. By the definition of the set of accept states of the DFA, q_j is an accept state iff $x \in L$. Hence the DFA recognizes L. □

6 Comparison with the Pumping Lemma (Optional)

The classic method for showing that languages are non-regular in courses like this one is via the "Pumping Lemma," stated as follows.

Theorem 3. Let L be a regular language. There exists a "pumping length" $p \geq 1$ such that every string $w \in L$ with length $|w| \geq p$ can be written as $w = xyz$, where $|y| > 0$ and $|xz| \leq p$, such that for every $i \geq 0$, the string $xy^iz \in L$.

The proof of the Pumping Lemma relies on the following simple and beautiful idea: If L is recognized by a DFA with p states, then the pigeonhole principle implies that every input w of length at least p causes M to visit some state twice during its execution.

To use the Pumping Lemma to prove a language L is not regular, one proceeds by contradiction. Assume the language L is regular. Then it has some pumping length p. Now judiciously choose some string w of length at least p, and show that no matter how one divides w into a concatenation xyz, there is some pumping xy^iz that is not in L.

Now for some pros and cons.

Pros:

- The proof of the Pumping Lemma illustrates a beautiful and important idea that will come up later in the course as we look at algorithms (Turing machines) solving computational problems about DFAs.
- It sets up an analogous result, the Pumping Lemma for context-free languages.

Cons:

- The Pumping Lemma is not a universal method for showing that languages are non-regular. (Unlike the method described in this note.) That is, there are some non-regular languages that *can* be pumped.
- In my opinion, the complexity of the statement of the Pumping Lemma makes it hard to appreciate how elegant the underlying idea is. There is also a disconnect between this idea and the actual process of constructing a (family) of unpumpable strings. Meanwhile, I think the process of constructing a pairwise distinguishable set forces you to think about how a DFA would try to recognize the language you're given.

7 Bibliographic Note

The characterization of regular languages obtained by combining Theorems 1 and 2 is known as the Myhill-Nerode Theorem, after John Myhill and Anil Nerode who proved it around 1958.