Note 1: Non-Regular Languages and Minimizing Finite Automata

Note: This note is based in part on an earlier version by Luca Trevisan.

Consider the following two questions: (1) Given a regular language L, how can we construct a DFA for L that has the minimum possible number of states? (2) Given a non-regular language L, how can we prove that L is in fact not regular? Both of these questions are very natural: the first has important applications to the efficiency of procedures that use finite automata. The second explores the computational limitations of finite automata. As we shall see, these questions are closely related (essentially because the minimum size of an automaton for a non-regular language is not finite!) In this lecture note, we will show how to design an efficient algorithm for state minimization, and along the way we'll see how to show that certain languages are not regular.

1 Distinguishable strings

Everything we do will hinge on the following definition. Let L be a language over the alphabet Σ.

Definition 1 (Distinguishable strings)**.** *We say that two strings $x, y \in \Sigma^*$ are* distinguishable w.r.t. L *if there exists $z \in \Sigma^*$ such that $xz \in L$ and $yz \notin L$, or vice versa.*

Example: Consider the language $L_1 = \{0^n 1^n : n \geq 1\}$. Then the strings $x = 0$ and $y = 00$ are distinguishable w.r.t. L_1, since if we take $z = 1$ then $xz \in L_1$ but $yz \notin L_1$. On the other hand, the strings $x = 1$ and $y = 10$ are indistinguishable since neither xz nor yz belongs to L_1 for any choice of z.

Exercise: Find two strings that are distinguishable, and two that are indistinguishable, for the (regular) language $L_2 = (0 \cup 1)^*(01)(0 \cup 1)^*$ of strings that contain 01 as a substring.

The importance of distinguishability lies in the following simple observation:

Lemma 2. *Let L be a language that is recognized by a DFA M, and let x, y be any two distinguishable strings w.r.t. L. Then the states reached by M on inputs x, y respectively must be different.*

Proof. Suppose for the sake of contradiction that M reaches the same state, q, on both inputs x, y. Let z be a string such that $xz \in L$ and $yz \notin L$ (or vice versa). Then on inputs xz and yz, M must also reach the same state, since in both cases its transitions are determined by the input z starting from state q. But this is a contradiction, since it implies that M must either accept both of xz, yz or reject both of them. □

We can now generalize the notion of distinguishability to *sets* of strings.

Definition 3 (Distinguishable set of strings)**.** *Let L be a language over Σ. We say that a set of strings $S \subseteq \Sigma^*$ is distinguishable w.r.t. L if every pair of distinct strings $x, y \in S$ are distinguishable w.r.t. L.*

Example: Let L_2 be the language of strings containing 01 as a substring, as in the Exercise above. Then the set of strings $\{1, 10, 01\}$ is distinguishable w.r.t. L_2. To see this, note that the pair $\{1, 10\}$ are distinguished by the string $z = 1$, and both the pairs $\{1, 01\}$ and $\{10, 01\}$ are distinguished by $z = \varepsilon$.

Exercise: Let $L_1 = \{0^n 1^n : n \geq 1\}$, as in the Example above. Show that the set of strings $\{0, 00, 000\}$ is distinguishable w.r.t. L_1.

Using Lemma 2, we can now prove a simple but powerful fact about the size of finite automata.

Theorem 4. *Let L be any language, and suppose there is a set of k distinguishable strings w.r.t. L. Then any DFA recognizing L must have at least k states.*

Proof. If L is not regular then there is no DFA recognizing L, so the statement is trivially true. So suppose L is regular and is recognized by a DFA M. Let $x_1, \ldots, x_k$ denote the k distinguishable strings w.r.t. L. By Lemma 2, we know that M must reach *distinct* states $q_1, \ldots, q_k$ on these k inputs. Hence M has at least k states. □

Example: Any DFA recognizing the language L_2 above must have at least three states, since we identified a set of three distinguishable strings in the previous Example.

2 Non-regular languages

Theorem 4 already gives us an effective tool to prove that certain languages are *not* regular. To illustrate this, consider our other running example language $L_1 = \{0^n 1^n : n \geq 1\}$. We claim that the *infinite* set of strings $S = \{0^k : k \geq 1\}$ is distinguishable w.r.t. L_1. To see this, consider any two distinct strings $x = 0^k$ and $y = 0^\ell$ in S, where $k > \ell \geq 1$. Then we can see these are distinguishable by taking $z = 1^k$.

We claim that this implies that L_1 is not regular. To see this, suppose for contradiction that L_1 is regular, and let M be a DFA that recognizes it. Let k be the number of states in M. But we saw in the previous paragraph that there are more than k distinguishable strings w.r.t. L_1, so by Theorem 4 M must have more than k states, a contradiction.

We can formalize the above argument in the following corollary.

Corollary 5. *If, for some language L, there is an infinite set S of distinguishable strings w.r.t. L, then L is not regular.*

Here's another example.

Example: The language $L_3 = \{ww : w \in \{0,1\}^*\}$ is not regular. To prove this, by Corollary 5 it suffices to find an infinite set of distinguishable strings w.r.t. L_3. We claim that the set $S = \{0^k 1 : k \geq 1\}$ does the job. Let $x = 0^k 1$, $y = 0^\ell 1$ be any two distinct strings in S, with $k \neq \ell$. Then we can see that x, y are distinguishable by considering xz, yz for the string $z = x$: clearly $xz = xx \in L_3$, but $yz = 0^\ell 10^k 1 \notin L_3$.

Exercise: In the previous Example, why didn't we use the set $S' = \{0^k : k \geq 1\}$ in place of S? [You should still be able to make the proof work with this choice, but it's a bit less natural. Note that our choice of S was based on the fact that strings of the form $0^k 10^k 1$ capture the "essence" of the language L_3, while strings like 0^{2k} do not. Picking the right set S to use in these proofs requires developing this kind of intuition, which comes with practice.]

Note: The notion of distinguishability provides a useful tool for proving that certain languages are not regular. You should try it out on some more examples (including those in the homework). A more traditional approach is based on the so-called "Pumping Lemma", which you can read about in Section 1.4 of the Sipser book.

Exercise: For each of the examples of non-regular languages in Section 1.4 of Sipser, use the above technique to prove that it is non-regular.

3 The Myhill-Nerode theorem

We now build on the notion of distinguishability in order to solve the first question asked at the beginning of this note, namely, how to find a smallest automaton that recognizes a given language. Let L be a language over Σ. For any two strings $x, y \in \Sigma^*$, we write $x \sim_L y$ to denote that x, y are *indistinguishable* w.r.t. L. I.e., $x \sim_L y$ means that, for every string z, either xz, yz are both in L or both are not in L.

Exercise: Show that $\sim_L$ is an *equivalence relation*, i.e., show that (i) $\forall\, x : x \sim_L x$; (ii) $\forall\, x, y : x \sim_L y \Leftrightarrow y \sim_L x$; (iii) $\forall\, x, y, z : (x \sim_L y) \wedge (y \sim_L z) \Rightarrow x \sim_L z$. (These all follow trivially from the definition of $\sim_L$.)

Since $\sim_L$ is an equivalence relation, it partitions Σ^* into *equivalence classes*, so that all strings in a given class are indistinguishable from one another, but strings in different classes are distinguishable. For any string $x \in \Sigma^*$, we will use the notation $[x]$ to denote the equivalence class containing x.

Exercise: For any equivalence class of $\sim_L$, show that either all strings in the class are in L or all are not in L. [Hint: take $z = \varepsilon$ as a distinguishing string.]

Now we have already seen in Theorem 4 that any DFA that recognizes L must have at least as many states as the number of equivalence classes of $\sim_L$ (Why?). Surprisingly, it turns out that the converse is also true, i.e., there always exists a DFA with this minimal number of states that recognizes L! This is stated in the following famous result from 1958, known as the Myhill-Nerode Theorem.

Theorem 6 (Myhill-Nerode)**.** *Let L be a language over Σ. If $\sim_L$ has infinitely many equivalence classes then L is not regular. Otherwise, L is regular and is recognized by a minimal DFA whose number of states is equal to the number of equivalence classes of $\sim_L$.*

Proof. The first part of the theorem, when there are infinitely many equivalence classes, is just Corollary 5 above. So now assume that $\sim_L$ has a finite number, k, of equivalence classes. We define a DFA that has one state for each equivalence class. The start state is $[\varepsilon]$, the equivalence class of the empty string ε. The accepting states are all states $[x]$ for $x \in L$.

What about the transition function δ? The obvious way to define this is $\delta([x], a) = [xa]$. However, we need to check that this definition makes sense. If $x \sim_L x'$ then the states $[x]$ and $[x']$ are the same, so we need to check that the states $[xa]$ and $[x'a]$ are also the same. I.e., we need to check that, for every z, $xaz \in L$ if and only if $x'az \in L$. But this is clearly true because x, x' are indistinguishable, so appending az to both of them gives us strings that are either both in L or both not in L.

So we have a well defined automaton. It remains to check that it recognizes L. For any input $x = x_1 \ldots x_n$, the DFA starts in state $[\varepsilon]$, then moves to state $[x_1]$, then to $[x_1x_2]$ and so on, ending up in state $[x]$. By definition, this is an accepting state if and only if $x \in L$, so the DFA does indeed recognize precisely L.

Finally, the fact that the number of states is minimal follows from Theorem 4. □

4 State minimization

Theorem 6 ensures that, for any given automaton M, there exists a minimal equivalent automaton with a well defined number of states. However, it doesn't give us an algorithm for constructing this automaton. We will now see a polynomial time algorithm for this task. The algorithm will run in time $O(sn^3)$, where $n = |Q|$ is the number of states in M and $s = |\Sigma|$ is the alphabet size. (Note that the description of M has size $O(sn)$, which is dominated by the space needed to write down the transition function.) There is also a more complicated $O(sn \log(sn))$ algorithm for this problem but we won't discuss it here.

Let M be a DFA recognizing some language $L \subseteq \Sigma^*$. Consider another equivalence relation, $\sim_M$, defined by

$$x \sim_M y \Leftrightarrow M \text{ ends up in the same state on inputs } x, y.$$

The claim in the following exercise is almost immediate:

Exercise: Prove that $\sim_M$ is indeed an equivalence relation, and that $\sim_M$ is a *refinement* of $\sim_L$, i.e., if $x \sim_M y$ then $x \sim_L y$.

This exercise means that each equivalence class of $\sim_L$ is the union of a set of equivalence classes of $\sim_M$. But the equivalence classes of $\sim_M$ correspond to states of M, so we can think of each equivalence class of $\sim_L$ as corresponding to a *set* of states of M (or, more correctly, to the strings that take M to this set of states). This suggests that the key to finding a minimum DFA for L is to *merge* states of M until we get down to the equivalence relation $\sim_L$, which we know is minimal.

To describe this process, it will be convenient to switch to an equivalence relation on *states* instead of strings.

Definition 7 (Equivalent states)**.** *Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. We say that two states $p, q \in Q$ are* equivalent*, denoted $p \equiv_M q$, if for every string $x \in \Sigma^*$, the states that M reaches on input x starting in p, q, respectively, are either both accepting or both non-accepting.*

Another way to think of this definition is that $p \equiv_M q$ if and only if the language recognized by M with start state p is the same as the language recognized by M with start state q.

Exercise: Verify that $\equiv_M$ is an equivalence relation on the state set Q.

The idea behind this definition is that, if states p, q are equivalent, then they can be merged into a single state, since such a merge will not change the language accepted by M. Our goal is to construct an automaton with just one state for each equivalence class of $\equiv_M$. We'll prove at the end of the note that this DFA is in fact minimal.

But how do we do detect equivalent states? It seems at first sight that testing whether $p \equiv_M q$ requires us to test infinitely many strings $x \in \Sigma^*$! Fortunately, as we'll see shortly, we can actually restrict attention to strings of a bounded length. First we need a modified version of the last definition that considers only strings up to a given length.

Definition 8. *Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. We say that two states $p, q \in Q$ are* equivalent up to length k*, denoted $p \equiv_M^k q$, if for every string $x \in \Sigma^*$ of length at most k, the states that M reaches on input x starting in p, q, respectively, are either both accepting or both non-accepting.*

We now show that the equivalence relations $\equiv_M^k$ satisfy a simple recurrence.

Lemma 9. *The equivalence relations $\equiv_M^k$ satisy the following recurrence:*

- $p \equiv_M^0 q$ *if and only if* p, q *are both accepting or both non-accepting. (I.e., the equivalence classes of* $\equiv_M^0$ *are just* F *and* $Q \setminus F$*.)*
- *For* $k \geq 1$*, we have* $p \equiv_M^k q$ *iff* $p \equiv_M^{k-1} q$ *and, for every* $a \in \Sigma$*,* $\delta(p, a) \equiv_M^{k-1} \delta(q, a)$*.*

Proof. The base case $\equiv_M^0$ is immediate from the definition. To see the inductive step, suppose first that $p \not\equiv_M^k q$. Then there exists a string x of length $\leq k$ that M accepts when starting from p but rejects when starting from q (or vice versa). If the length of x is $\leq k-1$ then $p \not\equiv_M^{k-1} q$. Otherwise, write $x = ax'$, so that x' has length $k-1$. But then x' proves that $\delta(p, a) \not\equiv_M^{k-1} \delta(q, a)$. Conversely, suppose that either $p \not\equiv_M^{k-1} q$ or $\delta(p, a) \not\equiv_M^{k-1} \delta(q, a)$ for some a. In the first case, clearly we have $p \not\equiv_M^k q$. In the second case, let x' be the string of length $\leq k-1$ that proves that $\delta(p, a) \not\equiv_M^{k-1} \delta(q, a)$; then ax' of length $\leq k$ proves that $p \not\equiv_M^k q$. □

Lemma 9 implies a simple dynamic programming procedure for computing $\equiv_M^k$ for any k, starting from $\equiv_M^0$. The running time will be $O(k \cdot |Q|^2 \cdot |\Sigma|) = O(ksn^2)$, since there are k levels of recursion and the time to compute $\equiv_M^k$from $\equiv_M^{k-1}$. is $O(|Q|^2 \cdot |\Sigma|)$ (as we need to consider each pair of states, and each symbol for the kth transition) .

But how large should k be? Here we make the important observation that if the recursive procedure ever fails to make progress, in the sense that $\equiv_M^k = \equiv_M^{k-1}$ for some k, then it has reached a fixed partition that will never change; i.e., $\equiv_M^{k'}$ will be the same as $\equiv_M^k$ for all $k' \geq k$. (You should check that you understand why!) But this implies that the procedure always terminates with a fixed partition after at most $|Q| - 1$ steps, since it starts with two classes and can't create more than $|Q|$ classes in total. Thus we will set $k = |Q| - 1$ from now on. Note that this means we can compute $\equiv_M$ in $O(|Q|^3|\Sigma|) = O(sn^3)$ steps.

But now we claim that then $\equiv_M^k$ is exactly the relation $\equiv_M$ that we want! To see this, note that if $p \equiv_M q$ then certainly $p \equiv_M^k q$. And if $p \not\equiv_M q$ then there is a string x of some finite length k' that proves this, meaning that $p \not\equiv_M^{k'} q$; if $k' \leq k$ then this implies $p \not\equiv_M^k q$, while if $k' > k$, the fact that $\equiv_M^{k'} = \equiv_M^k$ again implies $p \not\equiv_M^k q$. Hence $p \equiv_M q$ if and only if $p \equiv_M^k q$, so the relations are the same.

Now we're ready to spell out our state minimization algorithm. The input is a DFA $M = (Q, \Sigma, \delta, q_0, F)$.

- Let $k = |Q| - 1$, and compute the equivalence classes of $\equiv_M^k$ using the recursive procedure implied by Lemma 9. Then define a new DFA $M' = (Q', \Sigma, \delta', q_0', F')$ as follows. The states Q' correspond to the equivalence classes; q_0' is the equivalence class $[q_0]$; and F' is the set of equivalence classes that contain an accepting state in F. (Recall from an earlier Exercise that each equivalence class consists of only accepting or only non-accepting states.) The transistion function δ' is defined by $\delta'([q], a) = [\delta(q, a)]$.
- Remove from Q' all the states that are unreachable from q_0'. This can be done easily using a depth-first search of the graph of the automaton. Removing these states does not change the language recognized by the automaton as they never occur in any computation. Output the resulting DFA $M'' = (Q'', \Sigma, \delta'', q_0', F'')$.

We now claim the following.

Theorem 10. *The DFA* M'' *constructed by the above algorithm is a minimal DFA that recognizes the same language as* M*. The algorithm runs in time* $O(sn^3)$*, where* $s = |\Sigma|$ *and* $n = |Q|$*.*

Proof. That M'' recognizes the same language as M follows from the way we constructed it; a formal proof is left to the reader. Similarly, the runtime analysis follows from our earlier discussion.

To see that M'' is minimal, let $t = |Q''|$ be the number of states in M''. Since we have removed unreachable states, every state $[q] \in Q''$ is reachable by at least one input string $x_{[q]}$. Now consider two different states $[p], [q]$. Since the states of M'' are equivalence classes of $\equiv_M$, we know that $p \not\equiv_M q$, so there must be some string y such that M accepts y starting from p and rejects y starting from q (or vice versa). But this implies that M'' accepts y starting from $[p]$ and rejects y starting from $[q]$. Hence y proves that the strings $x_{[p]}$ and $x_{[q]}$ are distinguishable. This means we have a set of t distinguishable strings (one for each equivalence class), which by Theorem 4 implies that there is no smaller automaton for this language. □