Lecture 2

Countability for languages and deterministic finite automata

The main goal of this lecture is to introduce the finite automata model, but first we will finish our introductory discussion of alphabets, strings, and languages by connecting them with the notion of countability.

2.1 Countability and languages

We discussed a few examples of languages last time, and considered whether or not those languages were finite or infinite. Now let us think about countability for languages.

Languages are countable

We will begin with the following proposition.[1]

Proposition 2.1. *For every alphabet Σ, the language Σ^* is countable.*

Let us focus on how this proposition may be proved just for the binary alphabet $\Sigma = \{0, 1\}$ for simplicity—the argument is easily generalized to any other alphabet. To prove that Σ^* is countable, it suffices to define an onto function

$$f : \mathbb{N} \rightarrow \Sigma^*. \tag{2.1}$$

[1] In mathematics, names including *proposition*, *theorem*, *corollary*, and *lemma* refer to facts, and which name you use depends on the nature of the fact. Informally speaking, *theorems* are important facts that we're proud of and *propositions* are also important facts, but we're embarrassed to call them theorems because they were too easy to prove. *Corollaries* are facts that follow easily from theorems, and *lemmas* (or *lemmata* for Latin purists) are boring technical facts that nobody would care about except for the fact that they are useful for proving certain theorems.

In fact, we can easily obtain a one-to-one and onto function f of this form by considering the *lexicographic ordering* of strings. This is what you get by ordering strings by their length, and using the "dictionary" ordering among strings of equal length. The lexicographic ordering of Σ^* begins like this:

$$\varepsilon,\ 0,\ 1,\ 00,\ 01,\ 10,\ 11,\ 000,\ 001,\ \ldots \tag{2.2}$$

From the lexicographic order we can define a function f of the form (2.1) by setting $f(n)$ to be the n-th string in the lexicographic ordering of Σ^* starting from 0. Thus, we have

$$f(0) = \varepsilon,\ f(1) = 0,\ f(2) = 1,\ f(3) = 00,\ f(4) = 01, \tag{2.3}$$

and so on. An explicit method for finding $f(n)$ is to write $n + 1$ in binary and throw away the leading 1.

It is not hard to see that the function f we've just defined is an onto function—every binary string appears as an output value of the function f. It therefore follows that Σ^* is countable. It is also the case that f is a one-to-one function.

It is easy to generalize this argument to any other alphabet. This first thing we need to do is to decide on an ordering of the alphabet symbols themselves. For the binary alphabet we order the symbols in the way we were trained: first 0, then 1. If we started with a different alphabet, such as $\Gamma = \{\heartsuit, \diamondsuit, \clubsuit, \spadesuit\}$, it might not be clear how to order the symbols, but it doesn't matter as long as we pick a single ordering and stay consistent with it. Once we've ordered the symbols in a given alphabet Γ, the lexicographic ordering of the language Γ^* is defined in a similar way to what we did above, using the ordering of the alphabet symbols to determine what is meant by "dictionary" ordering. From the resulting lexicographic ordering we obtain a one-to-one and onto function $f : \mathbb{N} \to \Gamma^*$.

Remark 2.2. A brief remark is in order concerning the term *lexicographic order*. Some people use this term to mean something different (namely, dictionary ordering *without* first ordering strings according to length), and they use the term *quasi-lexicographic order* to refer to what we are calling lexicographic order. This isn't a problem—there are plenty of cases in which people don't all agree on what terminology to use. What is important is that everyone is clear about what they mean. In this course, *lexicographic order* means strings are ordered first by length, and by "dictionary" ordering among strings of the same length.

It follows from the fact that the language Σ^* is countable, for any choice of an alphabet Σ, that every language $A \subseteq \Sigma^*$ is countable. This is because every subset of a countable set is also countable, which is a simple fact that you will be asked to prove as a homework problem.

The set of all languages over any alphabet is uncountable

Next we will consider the set of all languages over a given alphabet. If Σ is an alphabet, then saying that A is a language over Σ is equivalent to saying that A is a subset of Σ^*, and being a subset of Σ^* is the same thing as being an element of the power set of Σ^*. These three statements are therefore equivalent, for any choice of an alphabet Σ:

1. A is a language over the alphabet Σ.
2. $A \subseteq \Sigma^*$.
3. $A \in \mathcal{P}(\Sigma^*)$.

We have observed, for any choice of an alphabet Σ, that every language $A \subseteq \Sigma^*$ is countable, and it is natural to consider next if the set of all languages over Σ is countable. It is not.

Proposition 2.3. *Let Σ be an alphabet. The set $\mathcal{P}(\Sigma^*)$ is uncountable.*

To prove this proposition, we don't need to repeat the same sort of diagonalization argument used to prove that $\mathcal{P}(\mathbb{N})$ is uncountable—we can simply combine that theorem with the fact that there exists a one-to-one and onto function from $\mathbb{N}$ to Σ^*.

In greater detail, let

$$f : \mathbb{N} \rightarrow \Sigma^* \tag{2.4}$$

be a one-to-one and onto function, such as the function we obtained from the lexicographic ordering of Σ^* like before. We can extend this function, so to speak, to the power sets of $\mathbb{N}$ and Σ^* as follows. Let

$$g : \mathcal{P}(\mathbb{N}) \rightarrow \mathcal{P}(\Sigma^*) \tag{2.5}$$

be the function defined as

$$g(A) = \{f(n) \,:\, n \in A\} \tag{2.6}$$

for all $A \subseteq \mathbb{N}$. In words, the function g simply applies f to each of the elements in a given subset of $\mathbb{N}$. It is not hard to see that g is one-to-one and onto—we can express the inverse of g directly, in terms of the inverse of f, as follows.

$$g^{-1}(B) = \{f^{-1}(w) \,:\, w \in B\} \tag{2.7}$$

for every $B \subseteq \Sigma^*$.

Now, because there exists a one-to-one and onto function of the form (2.5), we conclude that $\mathcal{P}(\mathbb{N})$ and $\mathcal{P}(\Sigma^*)$ have the "same size." That is, because $\mathcal{P}(\mathbb{N})$ is uncountable, the same must be true of $\mathcal{P}(\Sigma^*)$. To be more formal about this statement, one may assume toward contradiction that $\mathcal{P}(\Sigma^*)$ is countable, which implies that there exists an onto function of the form

$$h : \mathbb{N} \rightarrow \mathcal{P}(\Sigma^*). \tag{2.8}$$

By composing this function with the inverse of the function g specified above, we obtain an onto function

$$g^{-1} \circ h : \mathbb{N} \rightarrow \mathcal{P}(\mathbb{N}), \tag{2.9}$$

which contradicts what we already know, which is that $\mathcal{P}(\mathbb{N})$ is uncountable.

2.2 Deterministic finite automata

The first model of computation we will discuss in this course is a very simple one, called the *deterministic finite automata* model. You should have already learned something about finite automata (also called *finite state machines*) in CS 241, so we aren't necessarily starting from the very beginning—but we do of course need a formal definition to proceed mathematically.

Two points to keep in mind as you consider the definition are the following.

1. The definition is based on sets (and functions, which can be formally described in terms of sets, as you may have learned in a discrete mathematics course). This is not surprising: set theory provides the foundation for much of mathematics, and it is only natural that we look to sets as we formulate definitions.
2. Although deterministic finite automata are not very powerful in computational terms, the model is important nevertheless, and it is just the start. Do not be bothered if it seems like a weak and useless model—we're not trying to model general purpose computers at this stage, and finite automata are far from useless.

Definition 2.4. A *deterministic finite automaton* (or *DFA*, for short) is a 5-tuple

$$M = (Q, \Sigma, \delta, q_0, F), \tag{2.10}$$

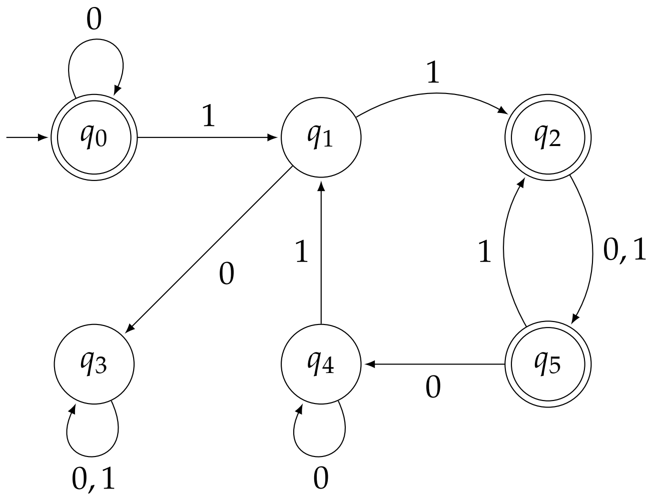
where Q is a finite and nonempty set (whose elements we will call *states*), Σ is an alphabet, δ is a function (called the *transition function*) having the form

$$\delta : Q \times \Sigma \rightarrow Q, \tag{2.11}$$

$q_0 \in Q$ is a state (called the *start state*), and $F \subseteq Q$ is a subset of states (whose elements we will call *accept states*).

State diagrams

It is common that DFAs are expressed using *state diagrams*, such as this one:



State diagrams express all 5 parts of the formal definition of DFAs:

1. States are denoted by circles.
2. Alphabet symbols label the arrows.
3. The transition function is determined by the arrows and the circles they connect.
4. The start state is determined by the arrow coming in from nowhere.
5. The accept states are those with double circles.

For the example above we have the following. The state set is

$$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}, \tag{2.12}$$

the alphabet is

$$\Sigma = \{0, 1\}, \tag{2.13}$$

the start state is q_0, the set of accepts states is

$$F = \{q_0, q_2, q_5\}, \tag{2.14}$$

and the transition function $\delta : Q \times \Sigma \rightarrow Q$ is as follows:

$$\begin{aligned} &\delta(q_0,0) = q_0, \qquad &&\delta(q_1,0) = q_3, \qquad &&\delta(q_2,0) = q_5,\\ &\delta(q_0,1) = q_1, &&\delta(q_1,1) = q_2, &&\delta(q_2,1) = q_5,\\ &\delta(q_3,0) = q_3, &&\delta(q_4,0) = q_4, &&\delta(q_5,0) = q_4,\\ &\delta(q_3,1) = q_3, &&\delta(q_4,1) = q_1, &&\delta(q_5,1) = q_2. \end{aligned} \tag{2.15}$$

In order for a state diagram to correspond to a DFA, it must be that for every state and every symbol, there is exactly one arrow exiting from that state labeled by that symbol.

Of course you can also go the other way and write down a state diagram from a formal description of a 5-tuple $(Q, \Sigma, \delta, q_0, F)$. It is a routine exercise to do this.

DFA computations

I imagine you already know what it means for a DFA $M = (Q, \Sigma, \delta, q_0, F)$ to *accept* or *reject* a given input string $w \in \Sigma^*$, or that at least you have guessed it based on a moment's thought about the definition. It is easy enough to say it in words, particularly when we think in terms of state diagrams: we start on the start state, follow transitions from one state to another according to the symbols of w (reading one at a time, left to right), and we accept if and only if we end up on an accept state (and otherwise we reject).

This all makes sense, but it is useful nevertheless to think about how it is expressed formally. That is, how do we define in precise, mathematical terms what it means for a DFA to accept or reject a given string? In particular, phrases like "follow transitions" and "end up on an accept state" can be replaced by more precise mathematical notions.

Here is one way to define acceptance and rejection more formally. Notice again that the definition focuses on sets and functions.

Definition 2.5. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA and let $w \in \Sigma^*$ be a string. The DFA M *accepts* the string w if one of the following statements holds:

1. $w = \varepsilon$ and $q_0 \in F$.
2. $w = \sigma_1 \cdots \sigma_n$ for a positive integer n and symbols $\sigma_1, \ldots, \sigma_n \in \Sigma$, and there exist states $r_0, \ldots, r_n \in Q$ such that $r_0 = q_0$, $r_n \in F$, and

$$r_{k+1} = \delta(r_k, \sigma_{k+1}) \tag{2.16}$$

   for all $k \in \{0, \ldots, n-1\}$.

If m does not accept w, then M *rejects* w.

In words, the formal definition of acceptance is that there exists a sequence of states $r_0, \ldots, r_n$ such that the first state is the start state, the last state is an accept state, and each state in the sequence is determined from the previous state and the corresponding symbol read from the input as the transition function describes: if we are in the state q and read the symbol σ, the new state becomes $p = \delta(q, \sigma)$. The first statement in the definition is simply a special case that handles the empty string.

It is natural to consider why we would prefer a formal definition like this to what is perhaps a more human-readable definition. Of course, the human-readable version beginning with "Start on the start state, follow transitions …" is effective for explaining the concept of a DFA, but the formal definition has the benefit that it reduces the notion of acceptance to elementary mathematical statements about sets and functions. It is also quite succinct and precise, and leaves no ambiguities about what it means for a DFA to accept or reject.

It is sometimes useful to define a new function

$$\delta^* : Q \times \Sigma^* \to Q, \tag{2.17}$$

based on a given transition function

$$\delta : Q \times \Sigma \to Q, \tag{2.18}$$

in the following recursive way:

1. $\delta^*(q, \varepsilon) = q$ for every $q \in Q$, and
2. $\delta^*(q, w\sigma) = \delta(\delta^*(q, w), \sigma)$ for all $q \in Q$, $\sigma \in \Sigma$, and $w \in \Sigma^*$.

Intuitively speaking, $\delta^*(q, w)$ is the state you end up on if you start at state q and follow the transitions specified by the string w.

It is the case that a DFA $M = (Q, \Sigma, \delta, q_0, F)$ accepts a string $w \in \Sigma^*$ if and only if $\delta^*(q_0, w) \in F$. A natural way to argue this formally, which we will not do in detail, is to prove by induction on the length of w that $\delta^*(q, w) = p$ if and only if one of these two statements is true:

1. $w = \varepsilon$ and $p = q$.
2. $w = \sigma_1 \cdots \sigma_n$ for a positive integer n and symbols $\sigma_1, \ldots, \sigma_n \in \Sigma$, and there exist states $r_0, \ldots, r_n \in Q$ such that $r_0 = q$, $r_n = p$, and

$$r_{k+1} = \delta(r_k, \sigma_{k+1}) \tag{2.19}$$

   for all $k \in \{0, \ldots, n-1\}$.

Once that equivalence is proved, the statement $\delta^*(q_0, w) \in F$ can be equated to M accepting w.

Remark 2.6. By now it is evident that we will not formally prove every statement we make in this course. If we did, we wouldn't get very far, and even then we might look back and feel as if we could probably have been even more formal. If we insisting on proving everything with more and more formality, we could in principle reduce every mathematical claim we make to axiomatic set theory—but then we would have covered very little material about computation in a one-term course, and our proofs would most likely be incomprehensible (and quite possibly would contain as many errors as you would expect to find in a complicated and untested program written in assembly language). Naturally we won't take this path, but from time to time we will discuss the nature of proofs, how we would prove something if we took the time to do it, and how certain high-level statements and arguments could be reduced to more basic and concrete steps pointing in the general direction of completely formal proofs that could be verified by a computer.

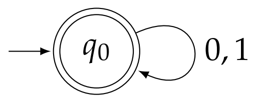
Languages recognized by DFAs and regular languages

Suppose $M = (Q, \Sigma, \delta, q_0, F)$ is a DFA. We may then consider the set of all strings that are accepted by M. This language is denoted $\mathrm{L}(M)$, so that

$$\mathrm{L}(M) = \bigl\{ w \in \Sigma^* \,:\, M \text{ accepts } w \bigr\}. \tag{2.20}$$

We refer to this as the *language recognized by M*.[2] It is important to understand that this is a single, well-defined language consisting precisely of those strings accepted by M and not containing any strings rejected by M.

For example, here is a very simple DFA over the binary alphabet $\Sigma = \{0, 1\}$:



If we call this DFA M, then it is easy to describe the language recognized by M; it is

$$\mathrm{L}(M) = \Sigma^*. \tag{2.21}$$

That's because M accepts exactly those strings in Σ^*. Now, if you were to consider a different language over Σ, such as

$$A = \{ w \in \Sigma^* \,:\, |w| \text{ is a prime number} \}, \tag{2.22}$$

then of course it is true that M accepts every string in A. However, M also accepts some strings that are not in A, so A is not the language recognized by M.

We have one more definition for this lecture, which introduces some very important terminology.

Definition 2.7. Let Σ be an alphabet and let $A \subseteq \Sigma^*$ be a language over Σ. The language A is *regular* if there exists a DFA M such that $A = \mathrm{L}(M)$.

[2] Alternatively, we might also refer to $\mathrm{L}(M)$ as the *language accepted by M*. Unfortunately this terminology sometimes leads to confusion because it overloads the term *accept*.

We have not seen too many DFAs thus far, so we don't have too many examples of regular languages to mention at this point, but we will see plenty of them throughout the course.

Let us finish off the lecture with a question.

Question 1. For a given alphabet Σ, is the number of regular languages over the alphabet Σ countable or uncountable?

The answer is "countable." The reason is that there are countably many DFAs over any alphabet Σ, and we can combine this fact with the observation that the function that maps each DFA to the regular language it recognizes is, by definition, an onto function to obtain the answer to the question.

When we say that there are countably many DFAs, we really should be a bit more precise. In particular, we are not considering two DFAs to be different if they are exactly the same except for the names we have chosen to give the states. This is reasonable because the names we give to different states of a DFA has no influence on the language recognized by that DFA—we may as well assume that the state set of a DFA is $Q = \{q_0, \ldots, q_{m-1}\}$ for some choice of a positive integer m. (In fact, people often don't even bother assigning names to states when drawing state diagrams of DFAs, because the state names are irrelevant to the way DFAs works.)

To see that there are countably many DFAs over a given alphabet Σ, we can use a similar strategy to what we did when proving that the set rational numbers $\mathbb{Q}$ is countable. First imagine that there is just one state: $Q = \{q_0\}$. There are only finitely many DFAs with just one state over a given alphabet Σ. (In fact there are just two, one where q_0 is an accept state and one where q_0 is a reject state.) Now consider the set of all DFAs with two states: $Q = \{q_0, q_1\}$. Again, there are only finitely many (although now it is more than two). Continuing on like this, for any choice of a positive integer m, there will be only finitely many DFAs with m states for a given alphabet Σ. Of course the number of DFAs with m states grows exponentially with m, but this is not important—it is enough to know that the number is finite. If you chose some arbitrary way of sorting each of these finite lists of DFAs, and then you concatenated the lists together starting with the 1 state DFAs, then the 2 state DFAs, and so on, you would end up with a single list containing every DFA. From such a list you can obtain an onto function from $\mathbb{N}$ to the set of all DFAs over Σ in a similar way to what we did for the rational numbers.

Because there are uncountably many languages $A \subseteq \Sigma^*$, and only countably many regular languages $A \subseteq \Sigma^*$, we can immediately conclude that some languages are not regular. This is just an existence proof—it doesn't give us a specific language that is not regular, it just tells us that there is one. We'll see methods later that allow us to conclude that certain specific languages are not regular.