Lecture 3

Finite Automata

3.1 Deterministic finite automata

The first computational model we will study in the class is *deterministic finite automata*. This is a very simple computational model, in which the computing device ("automaton") simply has a finite set Q of internal states it can be in. This automaton will read an input string one symbol at a time, and for each symbol it reads, it may move from its current internal state to one of the other internal states according to a fixed set of rules. This automaton will have some of its internal states labeled "accept state", and the rest labeled "reject state". Once the automaton finishes reading the string, it will accept the string (effectively outputting "1" or "True") if the final state it has reached is an accept state, and will reject the string (effectively outputting "0" or "False") if the final state is a reject state.

Formally, a deterministic finite automaton (usually abbreviated DFA) is a tuple

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

where:

1. Q is a finite nonempty set (representing the set of internal states of the automaton),
2. Σ is a finite nonempty set (representing the alphabet of the input strings the DFA can receive),
3. δ is a function $\delta\colon Q \times \Sigma \to Q$ called the transition function of the DFA (the interpretation of this function is that if the automaton has internal state q and reads the symbol a, it will move to internal state $\delta(q, a)$),
4. q_0 is an element of Q called the initial state or start state (representing the internal state of the DFA before it reads any input symbols), and
5. F is a subset of Q (representing the set of accept states of the automaton).

As an example, consider the task of taking as input a Boolean string, and accepting it if its parity is 1 (that is, if the number of 1 symbols in the string is odd). We can solve this task using a DFA, as follows. We'll have two states, q_0 and q_1, representing "parity has been 0 so far" and "parity has been 1 so far", respectively. Hence we'll have $Q = \{q_0, q_1\}$. The start state will be q_0, because at the beginning we have seen zero 1 symbols (so the parity of what we've seen so far is 0 at the beginning). The only accept state will be q_1, since we want to accept strings with odd parity; we will set $F = \{q_1\}$. The alphabet will be $\Sigma = \{0, 1\}$, since we are going to receive Boolean strings as

input. Finally, the transition function δ will keep us in the same state if we see a 0, and move us to the other state if we see a 1: that is, $\delta(q_0, 0) = q_0$, $\delta(q_1, 0) = q_1$, $\delta(q_0, 1) = q_1$, and $\delta(q_1, 1) = q_0$. This completes the definition of a DFA $M = (Q, \Sigma, \delta, q_0, F)$.

Upon receiving an input string x, the DFA M will start at the state q_0, and read each symbol $x_1, x_2, \ldots, x_n$ in x one by one (here $n = |x|$ is the length of the string x). When M reads x_1, it will move from q_0 to the state $p_1 = \delta(q_0, x_1)$. Then when M reads x_2, it will move to the state $p_2 = \delta(p_1, x_2)$. It will keep proceeding in this way, each time moving from p_i (the state after reading i bits of x, which is either equal to q_0 or to q_1) to the state $p_{i+1} = \delta(p_i, x_{i+1})$. In the end, it will arrive at the state p_n. The DFA M will then accept if and only if $p_n \in F$, that is, if $p_n = q_1$. It is not hard to see that this will happen if and only if the input string x has odd parity.

3.2 State diagrams

DFAs are often conveniently represented in a state diagram, as in Figure 3.1. In this diagram, states are represented by circles, with accept states circled twice. The transition function δ is represented using arrows that point from one state to another, and which are each labeled by an alphabet symbol. An arrow from q_0 to q_1 labeled by 1 should be interpreted to mean that $\delta(q_0, 1) = q_1$, or equivalently, that when the automaton is in internal state q_0 and reads a 1 symbol, it will move to internal state q_1. Note that staying in the same internal state is represented as a loop: that is, if $\delta(q_0, 0) = q_0$, this means the automaton would stay at q_0 if it was already there and read a 0 symbol, and this is represented as a loop on the state q_0 that is labeled by 0. Finally, the unique start state is marked by an arrow pointing to it from nowhere; in Figure 3.1, the start state is q_0.

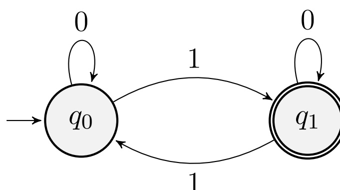

Figure 3.1: The DFA M which computes the parity of a string.

The state diagram of a deterministic finite automaton is a good way to visualize the machine. To figure out what the machine M will do on a given input, say 0110, imagine placing a checker piece on the first state (the one the arrow from nowhere is pointing to). In our case, this is the state q_0. Next, take the first bit of the input string, which is 0, and move the checker piece along the arrow labeled 0 which starts from the current state. In our case, this causes the checker to remain on q_0. We then proceed to the next bit of the input, which is a 1, and again follow the arrow, sliding the checker to q_1. The next 1 causes the checker to slide back to q_0, and the final input symbol 0 causes the checker to remain at q_0. Now that we've completed reading the input, we look at the state the checker is on, and check whether it is an accept state. In this case, the state is q_0, which does not have the double circle, and is not an accept state; therefore, we conclude that the DFA M rejects the input 0110.

Question 3.1. *What does the DFA in Figure 3.2 do?*

Answer: since the first three states of this DFA are all accept states (they are all circled twice), it might make sense to see how to get to the only reject state q_3. First, note that once we reach the reject state q_3, we can never leave: from q_3, if we see either a 0 or a 1, we remain in the same

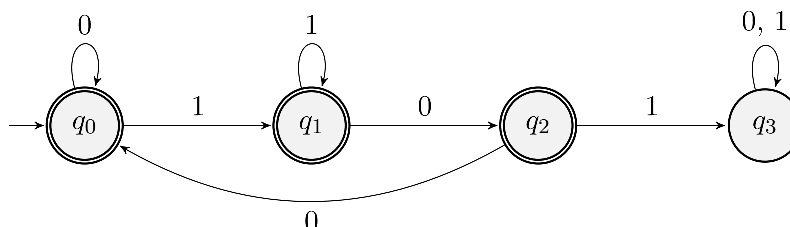

Figure 3.2: Another DFA.

state q_3. So once q_3 has been reached, the DFA will reject no matter what further symbols it reads. How can we get to q_3? The DFA starts at q_0, so one way to get to q_3 is for the DFA to see a string starting with 101. Also notice that q_1 can only be reached if the previous symbol was a 1, and indeed, whenever the DFA sees a 1, it must end up in either q_1 or q_3 (since all arcs labeled 1 go to q_1 or q_3). Next, if the DFA sees the symbols 10 in a row, then it must end up in either q_2 or q_3 (because after seeing the 1, it is in either q_1 or q_3, and after seeing the 0, these become either q_2 or q_3). Going one step further, we see that after seeing 101, this DFA will always end up at q_3, even if the 101 was in the middle of a string. Conversely, the only way to reach q_1 is to see a 1, which means the only way to reach q_2 is to see the symbols 10 in a row, and so the only way to reach q_3 for the first time is to see the sequence 101. In conclusion, this DFA accepts all strings which do not contain 101 as a substring.

3.3 Formally defining what a DFA accepts

Now that we have seen state diagrams and some examples, you hopefully have an understanding of what it means for a DFA to accept a string x. To make this understanding a little more formal, we'll introduce the following definition. Let $(Q, \Sigma, \delta, q_0, F)$ be a DFA. Then we define the extended transition function δ^* to be a function $\delta^* \colon Q \times \Sigma^* \to Q$, as follows. For all states $q \in Q$, we define $\delta^*(q, \epsilon) = q$. Further, for all states $q \in Q$, for all strings $x \in \Sigma^*$, and for all symbols $c \in \Sigma$, we recursively define $\delta^*(q, xc) = \delta(\delta^*(q, x), c)$.

This definition is saying that $\delta^*(q, x)$ is equal to the state the DFA reaches if it were to start at q and read the string x. To see this, note that if $x = \epsilon$, the DFA does not read anything and says at q; this is why we defined $\delta^*(q, \epsilon) = q$. On the other hand, if the DFA starts at q and receives a nonempty string y, then we can write $y = xc$ where c is the last symbol of y and x is the string constituting the rest of y. In this case, the final state the DFA will go to after reading y is determined by going to the state it will reach after reading x (the state $\delta^*(q_0, x)$), and then following the transition for reading symbol c (the state $\delta(\delta^*(q_0, x), c)$).

In conclusion, the definition of δ^* allows us to talk about where the DFA will end up after reading an entire string. This is as opposed to δ, which can only be used to tell us where the DFA will go after reading a single symbol.

With δ^* in hand, it is easy to define what it means for a DFA to accept a string. We say that a DFA $M = (Q, \Sigma, \delta, q_0, F)$ accepts a string $x \in \Sigma^*$ if and only if $\delta^*(q_0, x) \in F$, where δ^* is the extended transition function of M. We give one further definition, of the set of all strings accepted by a DFA M. This is called the language recognized by M.

Definition 3.2. *Let M be a DFA. The language recognized by M, denoted $L(M)$, is the set of all strings in Σ^* accepted by M, where Σ is the alphabet of M.*

The language recognized by a DFA corresponds to the task the DFA is solving. For the task of computing the parity of a Boolean string, the language will be the set of all strings of odd parity (that is, the set of all strings with an odd number of 1s).

Note that $L(M)$ is the language of *all* strings recognized by a DFA. If you're given a DFA and want to describe its language, remember to find all the strings the DFA accepts, not only some of the strings.

3.4 Nondeterministic finite automata

A generalization of deterministic finite automata is the model of *nondeterministic finite automata*, abbreviated NFA. An NFA is like a DFA, except that a state q may have multiple outgoing arcs with the same label. This means that when an NFA processes a string x, there are many paths it is allowed to follow. We say that an NFA accepts a string x if *some* path it can follow for x leads it to an accept state.

There are a couple more conventions we will use for NFAs. First, NFAs can have ϵ-*transitions*, which are arcs labeled by ϵ rather than by an alphabet symbol such as 0 or 1. If an NFA is at a state q and there is an ϵ-transition from q to q', then the NFA is *allowed* to go from q to q', but doesn't have to; it may also remain at q. The second convention is that NFAs need not always have an allowed transition. For example, from the state q, perhaps there is no arc going out of q with the label 0, even though $0 \in \Sigma$. This is allowed, unlike for DFAs. The interpretation is if the NFA is at state q and sees a 0, it must reject the entire string (effectively "falling into an abyss", going to an always-reject state and staying there regardless of what symbols it reads afterwards).

Figure 3.3 gives example of a state diagram for an NFA.

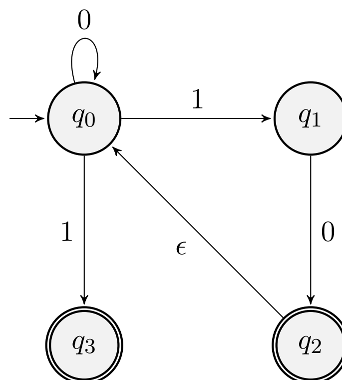

Figure 3.3: An example of an NFA.

Note that in this NFA, the start state is q_0 at the top left. From this start state, if the NFA sees a 0, it remains at q_0, but if it sees a 1, it can go *either* to q_1 or to q_3. What if the NFA sees the string 10? In this case, upon seeing the first symbol 1, the NFA can go to either q_1 or q_3; then, when it sees the symbol 0, then from q_1 it transitions to q_2, and from q_3 there is no transition—this means the NFA falls off a cliff and always rejects, so that there are no remaining viable states along that path. This means the NFA can only be in state q_2 after seeing 10. But wait! We are not yet done, because there is an ϵ-transition coming out of q_2. This transition can either be followed, or not. Therefore, after reading 10, the set of possible states the NFA could be in is $\{q_0, q_2\}$. Note that one of these states, the state q_2, is an accept state; hence this NFA accepts the string 10.

It might be a useful exercise to determine the language $L(M)$ of the NFA M in Figure 3.3. We

can see that this NFA does not accept the string ϵ, since its only possible state is q_0 if it sees no symbols. This NFA accepts the string 1 (due to q_3) but not the string 0 (since that forces it to remain at q_0). It also accepts the string 10 (due to q_2), and 101 (by going from q_2 to q_0 using the ϵ transition after seeing 10, and then going to q_3 after seeing the last 1).

On the other hand, this NFA will never accept any string with 11 as a substring, becuse from every possible state, there are no two consecutive 1 transitions available. That is, after seeing a single 1 in the middle of a string, it is not possible for the NFA to be in any state other than q_1 or q_3; then, after seeing another 1, it is not possible for this NFA to be in any state at all, so it falls into the abyss and forever rejects the string. So we know something about the language of this NFA: it does not contain strings with two consecutive 1s. It also doesn't contain any all-0 string: such strings force the NFA to stay at q_0, and q_0 is not an accept state.

Does this NFA accept all strings with no two consecutive 1s that have at least one 1? No: the string 100 is not accepted by this NFA, because it forces the NFA to be at the state q_0 after reading this string. Instead, this NFA accepts exactly the strings that do not have two consecutive 1s, and end with either 1 or 10. To see why, first note that if a string ends with neither of these, then it either ends with 00 or it is the string ϵ or 0. However, in all of these cases, the only possible state for the NFA is q_0, which is not an accept state; hence the NFA does not accept strings that do not end in 1 or 10. Conversely, if the input string does not have two consecutive 1s and does end in 1 or 10, then it must either be exactly the string 1 or exactly the string 10, or else end in 01 or 010. In the former cases, the NFA clearly accepts 1 and 10. For the latter two cases, the input string has the form $x01$ or $x010$ for some string x. Observe that regardless of the value of x, so long as it does not contain two consecutive 1s, q_0 will be a possible state after seeing $x0$, simply by always going through the loop $q_0 \to q_0$ when seeing a 0, and $q_0 \to q_1 \to q_2 \to q_0$ when seeing 10. Therefore, after seeing $x01$ or $x010$, the NFA can reach either q_3 or q_2, meaning it accepts.

3.5 A formal definition of NFAs

To formally define NFAs, we need to modify the definition of DFAs to alter the transition function. An NFA will be a tuple

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

where (just like in a DFA) Q and Σ are nonempty finite sets representing the set of states and the alphabet respectively, $F \subseteq Q$ is the set of accept states, and $q_0 \in Q$ is the start state. We will assume that the symbol ϵ is not in Σ. The function δ will be different than in a DFA: it will be a function $\delta\colon Q \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}(Q)$ which takes in a state $q \in Q$ and either a symbol in Σ or ϵ, and outputs a *set* of states to which the NFA may transition.

For example, in the NFA from Figure 3.3, we will have $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{0, 1\}$, $F = \{q_2, q_3\}$, and the accept state will be q_0. The function δ will be defined by $\delta(q_0, 0) = \{q_0\}$, $\delta(q_0, 1) = \{q_1, q_3\}$, $\delta(q_0, \epsilon) = \varnothing$, $\delta(q_1, 0) = \{q_2\}$, $\delta(q_1, 1) = \varnothing$, $\delta(q_1, \epsilon) = \varnothing$, $\delta(q_2, 0) = \varnothing$, $\delta(q_2, 1) = \varnothing$, $\delta(q_2, \epsilon) = \{q_0\}$, $\delta(q_3, 0) = \varnothing$, $\delta(q_3, 1) = \varnothing$, $\delta(q_3, \epsilon) = \varnothing$. Note that since q_0 has two outgoing arcs with label 1, we have $\delta(q_0, 1)$ be a set of size 2. Similarly, q_1 has no outgoing arcs labeled ϵ, so we have $\delta(q_1, \epsilon) = \varnothing$. It should be clear that this transition function δ precisely determines the set of arcs in Figure 3.3 and vice versa.

This gives a formal definition of an NFA. Next, we should formally define what it means for an NFA to accept a string. To do so, we will first define the extended transition function δ^*, just as we did for DFAs. However, for NFAs, ϵ-transitions make things a little more subtle.

We will first define the ϵ-*closure* of a set of states. This is a map from sets of states to sets of states, usually denoted by ϵ. That is, we define ϵ as a function $\epsilon\colon \mathcal{P}(Q) \to \mathcal{P}(Q)$. The ϵ-closure of a

set of states $S \subseteq Q$ will be the set of all states one can reach by starting from some state in S and following only ϵ-transitions in the NFA. Formally, we want

1. $S \subseteq \epsilon(S)$,
2. for all $q \in \epsilon(S)$, $\delta(q, \epsilon) \subseteq \epsilon(S)$, and
3. $\epsilon(S)$ does not contain any state except for the ones forced by conditions (1) and (2).

The last condition is actually not a mathematically rigorous criterion; it's not considered rigorous to talk about the states "forced" to be in $\epsilon(S)$ by conditions (1) and (2). Below, I'll explain the way to make such definitions fully mathematically formal, but if you don't follow this discussion, just remember the informal definition above for $\epsilon(S)$, which amounts to the same thing.

The way mathematicians usually handle such recursive definitions, in a way that makes them fully formal, is as follows. First, consider all the sets A that satisfy the conditions we want, in this case (1) that $S \subseteq A$ and (2) $q \in A \Rightarrow \delta(q, \epsilon) \subseteq A$. Let $\mathcal{C} \subseteq \mathcal{P}(Q)$ be the set of all such sets A which satisfy both of these properties. Note that some such sets will be too large—for example, Q itself always satisfies both of these properties. So what we want to do is find the minimal set in $\mathcal{C}$, and set $\epsilon(S)$ to be that set. The way to do this formally is to define $\epsilon(S)$ to be $\bigcap \mathcal{C}$, the intersection of all the sets in $\mathcal{C}$.

We now claim that $\bigcap \mathcal{C}$ satisfies (1) and (2), meaning that $\bigcap \mathcal{C} \in \mathcal{C}$. To see this, first note that since S is a subset of each set in $\mathcal{C}$, we have $S \subseteq \bigcap \mathcal{C}$, so (1) holds. Next, for each $q \in \bigcap \mathcal{C}$, we know that $q \in A$ for each $A \in \mathcal{C}$ (by definition of intersection), and therefore $\delta(q, \epsilon) \subseteq A$ for each $A \in \mathcal{C}$ (since each $A \in \mathcal{C}$ satisfies (2)). This means that $\delta(q, \epsilon) \subseteq \bigcap \mathcal{C}$, and so $\bigcap \mathcal{C}$ satisfies (2). We conclude that $\bigcap \mathcal{C}$ is in $\mathcal{C}$ (since it satisfies (1) and (2)), and hence it must be the smallest set in $\mathcal{C}$, since it is the intersection of all of them. Thus setting $\epsilon(S)$ to be $\bigcap \mathcal{C}$ gives us the right definition, satisfying the informal condition (3) as well. (This trick, of taking the intersection of all sets satisfying some desired properties in order to identify the *minimum* set satisfying those properties, shows up again and again in mathematics.)

Now that we've established the definition of ϵ-closure, we are ready to define the extended transition function δ^* of an NFA. We define this as follows:

1. for all $q \in Q$, set $\delta^*(q, \epsilon) = \epsilon(\{q\})$
2. for all $q \in Q$, all $x \in \Sigma^*$, and all $c \in \Sigma$, set $\delta^*(q, xc) = \epsilon\left(\bigcup_{p \in \delta^*(q,x)} \delta(p, c)\right)$.

What is this definition saying? It is setting $\delta^*(q, \epsilon)$ to be the set of all states reachable from q via ϵ-transitions; so far, this is sensible. Next, the action of $\delta^*(q, xc)$ is defined recursively: first, we apply $\delta^*(q, x)$, to get a set of states reachable from q after reading the string x; second, for each state p in this set, we apply $\delta(p, c)$ to get a set of states reachable from p using a single arc labeled c; third, we union all those sets together to get the set of all states reachable from p by reading x and taking an additional c-transition; and finally, we take the ϵ-closure of this set, to get the set of all states reachable from q after reading x and then c.

Now that we have δ^*, we can talk formally talk about the set of all states reachable after reading a string. We can finally define what it means for an NFA to accept a string x: it means that an accept state is reachable upon reading x, or formally, it means that $\delta^*(q_0, x) \cap F \neq \varnothing$ (that is, there is some state in F that is also in $\delta^*(q_0, x)$, or in other words, some accept state that is reachable from the start state after reading the string x).

Finally, we define the language recognized by an NFA M, denoted $L(M)$, to be the set of all strings accepted by M; that is, $L(M) = \{x \in \Sigma^* : \delta^*(q_0, x) \cap F \neq \varnothing\}$.

3.6 Conclusion and next steps

So far, we have introduced three ways of generating or recognizing a language. First, there were regular expressions, introduced last class; Second, DFAs; and finally, NFAs. Next lecture, we will show that all three models actually have the same expressive power—that is, the following three statements about a language A are equivalent:

1. there is some regular expression R such that $L(R) = A$
2. there is some DFA M such that $L(M) = A$
3. there is some NFA N such that $L(N) = A$.

Of course, it is easy to see why (2) implies (3): every DFA can immediately be converted to an NFA doing the same thing.[1] Much more surprising is that an NFA can also be converted to a DFA, that both can be convered into a regular expression, and that a regular expression can be converted into a DFA or NFA. Next lecture, we will show how to implement these conversions.

Stepping back a bit, recall that we defined a language to be *regular* if it has a regular expression. The equivalences above imply that we could also define a language to be regular if it is recognized by a DFA, or by an NFA. Hence there are actually three equivalent definitions of an NFA, each of which on the surface looks quite different.

What is the "right" way to think about regular languages? Which definition best explains them? Personally, I like to think of regular languages in terms of the DFA definition: they are the languages recognized by a finite state machine. The "finite state" part is important: it means the machine has a fixed, constant amount of memory, one that does not grow with the input. In other words, regular languages are the languages that can be recognized by a machine with $O(1)$ memory; they are the tasks you can solve with machine that does not even have enough memory to store the input (indeed, it must receive the input in a stream, one symbol at a time).

[1] Formally, a DFA doesn't quite qualify as an NFA, because its transition function $\delta(q, c)$ outputs a single state p instead of a set of states; but we can easily fix this by setting $\delta(q, c) = \{p\}$ instead of $\delta(q, c) = p$. We would similarly need to set $\delta(q, \epsilon) = \varnothing$ to declare that there are no ϵ-transitions. After this modification, the DFA becomes an NFA.